JAVA FOR THE STUDY OF EVOLUTION

VOLUME XIII

THE HAND

José del Carmen Rodríguez Santamaría
The EvolJava Community

Contents

1 Getting started with JavaFX 3D **1**
- 1.1 Predefined shapes . . . 1
- 1.2 Animation . . . 6
- 1.3 Conclusion . . . 9

2 A hand made of boxes **11**
- 2.1 A very primitive hand . . . 11
- 2.2 A hand for a pianist . . . 34
- 2.3 A hand that can feel . . . 34
- 2.4 From a hand to a pair of pliers . . . 76
- 2.5 A hand with opposite thumb . . . 76
- 2.6 A hand with opposable thumb . . . 76
- 2.7 An animation . . . 98
- 2.8 Just for fun . . . 122
- 2.9 In hindsight . . . 138
- 2.10 Conclusion . . . 138

3 Evolution explains a virtual hand **139**
- 3.1 Design of the fossil record . . . 140
- 3.2 Directed Evolution under trial . . . 170
- 3.3 Natural Evolution under trial . . . 241
- 3.4 Conclusion . . . 264

4 3D from 2D panes **267**
- 4.1 Interactive graphics . . . 283
- 4.2 Conclusion . . . 304

5 Meshes **305**
- 5.1 A box as a mesh . . . 305
- 5.2 Wallpapers . . . 313

5.3 Orientation 325
5.4 Exercises 332
5.5 Conclusion 348

6 Molograms **349**
6.1 Downloading molograms 349
6.2 Displaying molograms 350
6.3 The skeleton of the hand 385
6.4 Skinned hand 405
6.5 Conclusion 405

7 Evolution vs. our hand **407**
7.1 The hand as a marvel 407
7.2 JavaFX: layers 407
7.3 Evolution as a null hypothesis 410
7.4 Computing the null hypothesis. 427
7.5 From silhouettes to 3D-meshes 428
7.6 Our prediction as a 3D-mesh 464
7.7 Conclusion 482

Answers to exercises **485**

Preface

The series *Java for the Study of Evolution* is directed to scientists that want to manage a serious but not excessively expensive tool to study evolution by direct experimentation under perfectly controlled conditions. These requirements cannot be met in nature but only in simulations and mathematical models. In consequence, the series has three main purposes:

1. To endow the community of **researchers in biology and evolution** with *high level programming*, enabling an accurate study of models and simulations of the most diverse nature.

2. To clearly show how this tool is used to study the fundamental questions of evolution.

3. To suggest that the study of Java could be *very fruitful* for **undergraduates** in biological sciences even more than calculus alone.

This is the 13th volume: We face the challenge of learning how to test evolutionary theories that pretend to explain the hand. To that aim, we learn here how to use JavaFX 3D graphics and animations. This is a must for everyone in our community because there is plenty of room for everybody at high level of expertise. To fill in the prerequisites, try with Vol I, else V, and Vol XII.

Introduction

The human hand is a mysterious marvel at which the Author wonders even more than at an eye. So, our purpose along this volume is to illustrate how to study evolutionary theories that purport to be explanations of its existence. Nevertheless, to attack a mystery is best done if one is armed with extremely powerful weapons. In this regard, we are committed to show that JavaFX 3D, the very modern Java facility for 3D graphics and animations, is appropriate and so, it is introduced in chapter I.

In chapter II we learn how to make an ensemble of boxes into a 3D shape that resembles a human hand. Simple as this problem could be, it was rather complex for the Author so, it was solved by a human guided evolutionary process. This material allowed us to face in chapter III to the problem of testing in a virtual world an Evolutionary Theory of the origin of the hand.

To fill in the needs of experts we consecrate chapters from IV to the end in which we learn how to work with meshes that are approximations to 3D shapes by means of triangles. Developed material allows us to depict in a 3D mesh a mandatory prediction of the Evolutionary Theory saying that the hand of the chimp, the bonobo and the man are related by a direct common ancestor.

Prerequisites: try with Vol I else V, plus Vol XII.

This is an invitation to rediscover science as the pleasure of making truths with own hands, truths that are the more credible the more they are supported by hard and wise work.

Bogotá, Colombia, *José Rodríguez*
March 2015

Chapter 1

Getting started with JavaFX 3D

Space and time together

1 ***Purpose.*** *In Vol IV of this series we got acquainted with 3D graphics and animations. We made everything with our own hands. There is, nevertheless, a great difference between a handmade craft and a professional one. We learned at that volume how to devise the basic machinery for 3D rendering but now we pursue to manage JavaFX 3D Graphics that is proposed as the ordinary tool for professionals (Cindy Castillo and John Yoon, [12], 2013). Here we begin with this task. As it is used in Java, there is place for beginners as well as another for experts. This introductory chapter is for beginners.*

1.1 Predefined shapes

2 ***The conception.*** *The three dimensional service of JavaFX is conceived to be a simulation of real TV: it has camera, lights, scene with moving objects. What one sees on the screen is what the camera sees. There is, nevertheless, a place for beginners and that is where we start with.*

3 ***Predefined shapes.***

Spheres, cubes and cylinders are included in JavaFX 3D. The following code shows us how to draw them.

```
/*
 *  Program M3 PredefinedShapes

    Spheres, cylinders and boxes
```

```
    are predefined in JavaFX and
    are ready to be enjoyed.
    Here we draw them.
 */
package predefinedshapes;


import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.Box;
import javafx.scene.shape.Cylinder;
import javafx.scene.shape.Sphere;



public class PredefinedShapes extends Application
{

@Override
public void start(Stage stage)
{

    stage.setTitle("Predefined shapes");
    Group root = new Group();
    Scene scene = new Scene(root, 1000, 800);
    //Color of Background
    scene.setFill(Color.color(0.2, 0.2, 0.2, 1.0));

    //Sphere
    double radius = 50;
    Sphere sphere = new Sphere(radius);
    //Position
    sphere.setLayoutX(100);
    sphere.setLayoutY(200);
    //Cylinder(radius, height)
    Cylinder cylinder = new Cylinder(50, 100);
```

```
    //Material for skinning shapes
    PhongMaterial material = new PhongMaterial(Color.color(1.0, 0.7,
    0.8));
    //Material is used as skin of cylinder
    cylinder.setMaterial(material);
    //Position
    cylinder.setLayoutX(300);
    cylinder.setLayoutY(200);
    //Box(xLength,yLength,zLength)
    Box box = new Box(50,70,100);
    PhongMaterial material2 = new PhongMaterial(Color.BLUE);
    box.setMaterial(material2);
    box.setLayoutX(500);
    box.setLayoutY(200);
    //Booking
    root.getChildren().addAll(cylinder, box, sphere);
    stage.setScene(scene);
    stage.show();
}


/**
 * @param args the command line arguments
 */
public static void main(String[] args)
{
    launch(args);
}

}//End of main class M3 PredefinedShapes
```

4 ***Exercise .*** *Run the previous program and play with the code. Set new positions for the different objects.*

5 ***Three dimensional objects can be given an initial rotation:***

```
 /*
 *  Program M5 PredefinedShapes2
```

```
    Cylinders and boxes
    get a rotation to exhibit their three dimensionality.
 */

package predefinedshapes2;


import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.geometry.Point3D;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.Box;
import javafx.scene.shape.Cylinder;
import javafx.scene.shape.Sphere;



public class PredefinedShapes2 extends Application
{

@Override
public void start(Stage stage)
{

    stage.setTitle("Predefined shapes");
    Group root = new Group();
    Scene scene = new Scene(root, 1000, 800);
    //Color of Background
    scene.setFill(Color.color(0.2, 0.2, 0.2, 1.0));

    //Sphere
    double radius = 50;
    Sphere sphere = new Sphere(radius);
    //Position
    sphere.setLayoutX(100);
    sphere.setLayoutY(200);
```

```
    //Cylinder(radius, height)
    Cylinder cylinder = new Cylinder(50, 100);
    //Material for skinning shapes
    PhongMaterial material = new PhongMaterial(Color.color(1.0, 0.7,
    0.8));
    //Material is used as skin of cylinder
    cylinder.setMaterial(material);
    //Position
    cylinder.setLayoutX(300);
    cylinder.setLayoutY(200);
    //Axis of rotation
    cylinder.setRotationAxis(new Point3D(1,1,1));
    //Angle of rotation in degrees
    cylinder.setRotate(60);

    //Box(xLength,yLength,zLength)
    Box box = new Box(50,70,100);
    PhongMaterial material2 = new PhongMaterial(Color.BLUE);
    box.setMaterial(material2);
    box.setLayoutX(500);
    box.setLayoutY(200);
    //Axis of rotation
    box.setRotationAxis(new Point3D(1,0,1));
    //Angle of rotation in degrees
    box.setRotate(30);
    //Booking
    root.getChildren().addAll(cylinder, box, sphere);
    stage.setScene(scene);
    stage.show();
}


/**
 * @param args the command line arguments
 */
public static void main(String[] args)
{
    launch(args);
}
```

```
}//End of main class M5 PredefinedShapes2
```

6 ***Exercise .*** *Run the previous program and play with the code. Set new rotations for the different objects.*

1.2 Animation

Let us see how to animate our 3D predefined shapes.

7 ***Shapes can be animated:***

```
/*
 *  Program M7 PredefinedShapes3

    Cylinder, sphere and box
    get movement.
    Direction of axes:
    X: horizontal,
    Y: vertical, positive downwards
    Z: perpendicular to the screen

 */

package predefinedshapes3;


import javafx.animation.KeyFrame;
import javafx.animation.KeyValue;
import javafx.animation.Timeline;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.geometry.Point3D;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.Box;
```

```
import javafx.scene.shape.Cylinder;
import javafx.scene.shape.Sphere;
import javafx.util.Duration;



public class PredefinedShapes3 extends Application
{
    private Timeline animation  = new Timeline();


@Override
public void start(Stage stage)
{

    stage.setTitle("Predefined shapes in movement");
    Group root = new Group();
    Scene scene = new Scene(root, 1000, 800);
    //Color of Background
    scene.setFill(Color.color(0.2, 0.2, 0.2, 1.0));

    //========Initialization============

    //Sphere
    double radius = 50;
    Sphere sphere = new Sphere(radius);
    //Position
    sphere.setLayoutX(100);
    sphere.setLayoutY(200);

    //Cylinder(radius, height)
    Cylinder cylinder = new Cylinder(50, 100);
    //Material for skinning shapes
    PhongMaterial material = new PhongMaterial(Color.color(1.0, 0.7,
    0.8));
    //Material is used as skin of cylinder
    cylinder.setMaterial(material);
    //Position
    cylinder.setLayoutX(300);
    cylinder.setLayoutY(200);
```

```
//Box(xLength,yLength,zLength)
Box box = new Box(50,70,100);
PhongMaterial material2 = new PhongMaterial(Color.BLUE);
box.setMaterial(material2);
box.setLayoutX(500);
box.setLayoutY(200);


//Booking
root.getChildren().addAll(cylinder, box, sphere);

//==========Animation============

//Axis of rotation of box
box.setRotationAxis(new Point3D(1,0,1));

//Duration of animation
double endTime = 4000.0;

animation.getKeyFrames().addAll
(
    ////Sphere
    //Translation: initial position at initial time
    new KeyFrame(Duration.ZERO,
        new KeyValue(sphere.translateXProperty(), 0.0)),
    //Translation: final position at final time
    new KeyFrame(new Duration(endTime),
        new KeyValue(sphere.translateXProperty(), 500)),

    ////Cylinder
    //Rotation: initial rotate at initial time
    new KeyFrame(Duration.ZERO,
        new KeyValue(cylinder.rotateProperty(), 0.0)),
    //Rotation: final rotate at final time
    new KeyFrame(new Duration(endTime),
        new KeyValue(cylinder.rotateProperty(), 360)),

    ////Box
    //Translation: initial position at initial time
```

```
        new KeyFrame(Duration.ZERO,
            new KeyValue(box.translateYProperty(), 0.0)),
        //Translation: final position at final time
        new KeyFrame(new Duration(endTime),
            new KeyValue(box.translateYProperty(), 500)),
        //Rotation: initial rotate at initial time
        new KeyFrame(Duration.ZERO,
            new KeyValue(box.rotateProperty(), 0.0)),
        //Rotation: final rotate at final time
        new KeyFrame(new Duration(endTime),
            new KeyValue(box.rotateProperty(), 360))

    );
    animation.play();
    stage.setScene(scene);
    stage.show();
}



public static void main(String[] args)
{
    launch(args);
}

}//End of main class M7 PredefinedShapes3
```

8 ***Exercise .*** *Run the previous program and play with the code.*

9 ***Exercise .*** *Randomize the previous program: colors, axes of rotation and direction of movement must be decided by randomness.* *Answer*

1.3 Conclusion

We got acquainted with the 3D facility of JavaFX. As usually, we find it both charming and challenging.

Chapter 2

A hand made of boxes

`Biological characters`

10 ***Objective.*** *We tinker here with boxes to design a 3D shape that resembles a human hand.*

2.1 A very primitive hand

We saw in Vol IV how to build own 3D graphics facility. We illustrated the task by designing in program D146 OppThumbImproved an interactive hand that was built with just boxes. Our former intention here was to rewrite that program into JavaFX 3D. Nevertheless, early experience showed that this directive of work was plagued with extreme difficulty. So, we decided to build a JavaFX hand from scratch. This failure explains why JavaFX exists instead of an improved version of Swing. We see here an example of a disconnection along a human driven evolutionary path of software development.

11 ***Boxes are made into a hand.*** *The following code shows an ensemble of boxes that resembles a primitive hand that has all its fingers on the same line but without opposable thumb.*

```
/*
 * Program M11 HandOnlyBoxes.
 * An ensemble of boxes is arranged into
 * something similar to a 3D hand
 * with five fingers on the same line
 * but without opposable thumb.
 *
```

```
 */

package handonlyboxes;

import java.lang.reflect.Field;
import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;
import javafx.geometry.Insets;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.PerspectiveCamera;
import javafx.scene.PointLight;
import javafx.scene.Scene;
import javafx.scene.SubScene;
import javafx.scene.control.Label;
import javafx.scene.control.Slider;
import javafx.scene.layout.HBox;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.Box;
import javafx.scene.transform.Rotate;
import static javafx.scene.transform.Rotate.X_AXIS;
import javafx.stage.Stage;



public class HandOnlyBoxes extends Application
{

    //Slider for zooming
    private Slider zSlider;
    //Position of graphic along the z-axis,
    //normal to the screen.
    //It is a double property because
    //it will get involved into a binding
```

```
    private final DoubleProperty zPos
            = new SimpleDoubleProperty(2000);



    //For listener of mouse
    private double anchorAngleX = 0;
    private double anchorAngleY = 0;
    private double anchorX, anchorY;
    //Angles of rotation
    private final DoubleProperty angleX
            = new SimpleDoubleProperty(0);
    private final DoubleProperty angleY
            = new SimpleDoubleProperty(0);

    PerspectiveCamera camera
            = new PerspectiveCamera(false);


    //An array with all colors that are named in JavaFX
    private static final Color JavaFXColors[]
                                = new Color[148];

    //Generator of pseudo random numbers
    private static final Random rd = new Random();

    //Initalization of number of phalanges in each finger
    private final int nPhalanges = 3;


    public void init(Stage primaryStage)
{
    primaryStage.setTitle( "An ensemble of boxes"
            + " that resembles a 3D hand."
            + " To rotate it, click and drag the mouse.");
    //The whole drawing  is created
    //as a JavaFX space-like object
    final Group parent = new Group(create3dContent());

    //Initial position
```

```
parent.setTranslateX(450);
parent.setTranslateY(400);
parent.setTranslateZ(0);
Rotate xRotate;
Rotate yRotate;
//Initialization of rotation
parent.getTransforms().setAll
(
  xRotate = new Rotate(0, Rotate.X_AXIS),
  yRotate = new Rotate(0, Rotate.Y_AXIS)
);

//Graphic is attached to root
final Group root = new Group(parent);
//Window of the scene
final Scene scene = new Scene(root, 900, 900, true);
//Background
scene.setFill(Color.SKYBLUE);

//Listener of mousePressed
scene.setOnMousePressed
(
    event ->
    {
        anchorX = event.getSceneX();
        anchorY = event.getSceneY();
        anchorAngleX = angleX.get();
        anchorAngleY = angleY.get();
    }
);


//Listener of mouseDragged
scene.setOnMouseDragged
(
    event ->
    {
        angleX.set(anchorAngleX
                - (anchorY - event.getSceneY()));
        angleY.set(anchorAngleY
```

```
                + anchorX - event.getSceneX());
        }
    );

    //Rotation gets bound to
    //the angle determined by  mouseDragged
    xRotate.angleProperty().bind(angleX);
    yRotate.angleProperty().bind(angleY);

    //Illumination
    PointLight pointLight = new PointLight(Color.WHITE);
    pointLight.setTranslateX(100);
    pointLight.setTranslateY(200);
    pointLight.setTranslateZ(-2000);
    root.getChildren().add(pointLight);

    //Camera
    scene.setCamera(camera);
    primaryStage.setScene(scene);
    PerspectiveCamera subSceneCamera
            = new PerspectiveCamera(false);
    scene.setCamera(subSceneCamera);

    //To zoom down the graphic,
    //it is translated away from  the beholder,
    //along the Z-direction (normal to the screen).
    //Change zPos to change initial setting.
    zSlider = new Slider();
    zSlider.setMax(5000);
    zSlider.setMin(1);


    //Bind zPos to the reading of the slider
    zSlider.valueProperty().bindBidirectional(zPos);
    //Execute negative zoom
    parent.translateZProperty().bind(zPos);
    System.out.println("zPos = " + zPos);
```

```
    //Other JavaFX controls, labels and containers
    Label zLabel = new Label("Zoom");
    HBox subSceneGroup = new HBox(70,
        new HBox(10, zLabel, zSlider));

    subSceneGroup.setPadding(new Insets(10, 100, 0, 100));
    SubScene subScene
            = new SubScene(subSceneGroup, 880, 200);
    subScene.setLayoutX(10);
    subScene.setLayoutY(800);
    root.getChildren().add(subScene);

    primaryStage.show();
}


public void play()
{

}

@Override public void stop()
{

}

@Override
public void start(Stage primaryStage)  throws Exception
{
    init(primaryStage);
    primaryStage.show();
    play();
}


public static void main(String[] args)
{
    launch(args);
}
```

```
private Node create3dContent()
{
    //Colors that have a name in JavaFX
    getColors();
    Hand  h =  new Hand();
    return new Group(h);
}

//Colors with a name in JavaFX are put into an array
static private Color[] getColors()
{
    //Colors with a name in JavaFX
    final Field[] fields = Color.class.getFields();


    int N = 0;
    //Colors are put into an array
    for (final Field field : fields)
    {
        if (field.getType() == Color.class)
        {
            try
            {
                final Color color = (Color) field.get(null);
                //Array to keep colors
                JavaFXColors[N] = color;
                /*
                //The name of the color
                final String colorName = field.getName();
                System.out.println("N = " + N + "  "
                        + color.toString() + " " + colorName);
                */
                N++;
            } catch (IllegalArgumentException
                    | IllegalAccessException ex)
            {
                Logger.getLogger(HandOnlyBoxes.class
```

```
                    .getName()).log(Level.SEVERE, null, ex);
            }
        }
    }

    return JavaFXColors;
}


//A random color is synthesized
 private static Color randomColor()
 {
    //A JavaFX named color is picked
    //but the first, Transparent, is ignored
    int n = rd.nextInt(JavaFXColors.length-1)+1;
    Color c = JavaFXColors[n];
    //Some light colors are not visualized so,
    //they might be darkened:
    //c = c.darker();
    return c;
 }



  //=======================================================
  //============Beginning of Inner class Hand============
  //=======================================================




private class Hand extends Group
{

    //Constructor
    Hand()
    {
        //Arm is constructed
        Box arm = new Box(100,20,400);
```

```
Color c = randomColor();
PhongMaterial materialArm = new PhongMaterial(c);
arm.setMaterial(materialArm);
arm.setTranslateY(20);
//Palm is constructed
Box palm = new Box(125,20,200);
Color c2 = randomColor();
PhongMaterial materialPalm = new PhongMaterial(c2);
palm.setMaterial(materialPalm);
palm.setTranslateZ(300);
palm.setTranslateY(20);
final Group armPalm = new Group(arm, palm);

//5 fingers on the same line.
//No opposable thumb
Finger Fingers[] = new Finger[5];
Group fingersGroup = new Group();
for(int i = 0; i < 5; i++)
{
    //Each finger has 3 phalanges
    Fingers[i] = new Finger(3);
    for(int j = 0; j < nPhalanges; j++)
    {
        Box f = Fingers[i].Phalanges[j];
        f.setTranslateX(12.5 * (5-2*(i))-12.5);
        f.setTranslateZ(420 + 40*j);
        f.setTranslateY(20);
        armPalm.getChildren().add(f);
        fingersGroup.getChildren().add(f);
        //f.getTransforms().add(new Rotate(40,X_AXIS));
    }
}

//We use the following constructor for rotation:
//Rotate(double angle,
//double pivotX, double pivotY, double pivotZ,
//Point3D axis)
//Creates a three-dimensional Rotate transform
//with pivot and
//around the specified axis
```

```
        fingersGroup.getTransforms()
                .add(new Rotate( 15,0,0, 390, X_AXIS));



        getChildren().addAll(armPalm, fingersGroup );
    }
}//End of inner class Hand


private class Finger extends Group
{

    //Our finger is a composition of 3 boxes or phalanges
    private   Box[] Phalanges    = new Box [nPhalanges];



    Finger(int nPhalanges)
    {

        for(int i =0; i < nPhalanges; i++)
        {
             final Box p = new Box(25,20, 40);
             Color c = randomColor();
             PhongMaterial materialFinger
                     = new PhongMaterial(c);
             p.setMaterial(materialFinger);
             Phalanges[i] = p;

        }
    }
}//End of inner class Finger


}//End of main class M11 HandOnlyBoxes
```

12 ***Exercise.*** *Run the previous program and play with the code.*

We plan to pass to animating the hand, which means rotating fingers and phalanges. So, let us review JavaFX 3D rotations.

13 ***Some theory about rotations.***

Our hand extends itself in the z-direction and to move fingers or phalanges, we must rotate them around the `X_AXIS`. By commanding a rotation we will see a hand that rotates around its center. But beware, fingers do not rotate around the center of the hand but around the end of the palm. Moreover, the axis of rotation of a phalanx is its connection with its mother phalanx. Thus, we ask: how can we implement the corresponding changes?

A **rotation** in JavaFX 3D is specified by an angle of rotation, a pivot and an axis of rotation. Now, the axis is described by a 3D point that denotes the line that passes through the origin of coordinates and the point. To make a parallel translation of this axis until reaching a giving point, we declare that target point as pivot.

We have many alternate forms to call the constructor of a rotation (Oracle, [13], 2014):

```
Rotate()
Creates a default Rotate transform (identity).
Rotate(double angle)
Creates a two-dimensional Rotate transform.
Rotate(double angle, double pivotX, double pivotY)
Creates a two-dimensional Rotate transform with pivot.
Rotate(double angle, double pivotX, double pivotY, double pivotZ)
Creates a simple Rotate transform with three-dimensional pivot.
Rotate(double angle, double pivotX, double pivotY,
    double pivotZ, Point3D axis)
Creates a three-dimensional Rotate transform with pivot.
Rotate(double angle, Point3D axis)
Creates a three-dimensional Rotate transform.
```

As we see, the pivot can be specified by the constructor

```
Rotate(double angle,
       double pivotX, double pivotY, double pivotZ, Point3D axis)
```

One can activate these transformation by packing them inside an **Xform** or an **XformWithPivot** . These are classes that declare and initialize the transformation that will be used in the program. Example:

```
//Kit of available families of transformations
private class XformWithPivot extends Group
{
    public Translate t = new Translate();

    public Rotate rx = new Rotate();
    { rx.setAxis(Rotate.X_AXIS); }
    public Rotate ry = new Rotate();
    { ry.setAxis(Rotate.Y_AXIS); }
    public Rotate rz = new Rotate();
    { rz.setAxis(Rotate.Z_AXIS); }
    //Rotation = angle + pivot + axis
    public Rotate rwpx = new Rotate(0,0,0,400, X_AXIS);
    {
        rwpx.setAxis(Rotate.X_AXIS);
        rwpx.setPivotZ(200);
    }
    public Rotate rwpy = new Rotate();
    { ry.setAxis(Rotate.Y_AXIS); }



    public Scale s = new Scale();

    public XformWithPivot()
    {
        super();
        getTransforms().addAll(t, rx, ry, rz, rwpx, rwpy, s);
    }
}
```

An Xform functions as a heading node to which other Xforms or specific graphic classes (that are themselves space-like JavaFX objects) can be added. Actually, instances of graphic classes can be grouped and submitted to the transformation available at a declared XForm. Let us apply all this to control with a slider the movement of the hand.

14 ***The following code shows an ensemble of boxes that are made into a hand that can be closed by a slider.***

```
/*
```

```
 * Program M14 HandOnlyBoxes2.
 * It is shown how to make an ensemble of boxes
 * into a 3D closable hand
 * that is controlled by a slider.
 */

package handonlyboxes2;

import java.lang.reflect.Field;
import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;
import javafx.geometry.Insets;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.PerspectiveCamera;
import javafx.scene.PointLight;
import javafx.scene.Scene;
import javafx.scene.SubScene;
import javafx.scene.control.Label;
import javafx.scene.control.Slider;
import javafx.scene.layout.HBox;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.Box;
import javafx.scene.transform.Rotate;
import static javafx.scene.transform.Rotate.X_AXIS;
import javafx.stage.Stage;



public class HandOnlyBoxes2 extends Application
{

    //Slider for zooming
    private Slider zSlider;
```

```
    //Position of graphic along the z-axis,
    //normal to the screen.
    //It is a double property because
    //it will get involved into a binding
    private final DoubleProperty zPos
            = new SimpleDoubleProperty(2000);

    //Slider for angle of rotation of fingers
    private Slider angleSlider;
    //Position of brick along the z-axis,
    //normal to the screen.
    //It is a double property because
    //it will get involved into a binding
    private final DoubleProperty angleValue
            = new SimpleDoubleProperty(0);

    //private final Group fingersGroup = new Group();

    //For listener of mouse
    private double anchorAngleX = 0;
    private double anchorAngleY = 0;
    private double anchorX, anchorY;
    //Angles of rotation
    private final DoubleProperty angleX
            = new SimpleDoubleProperty(0);
    private final DoubleProperty angleY
            = new SimpleDoubleProperty(0);

    PerspectiveCamera camera
            = new PerspectiveCamera(false);

    //The application is instantiated
    static HandOnlyBoxes2 p = new HandOnlyBoxes2();



    //An array with all colors that are named in JavaFX
    private static final Color JavaFXColors[]
                                = new Color[148];
```

```
    //Generator of pseudo random numbers
    private static final Random rd = new Random();


    //Initalization of number of cubes in each finger
    private final int nPhalanges = 3;

public void init(Stage primaryStage)
{
    primaryStage.setTitle("An ensemble of boxes"
            + " act coherently as a thumbless hand");
    //The whole drawing  is created
    //as a JavaFX space-like object
    final Group parent = new Group(create3dContent());

    //Initial position
    parent.setTranslateX(450);
    parent.setTranslateY(400);
    parent.setTranslateZ(0);
    Rotate xRotate;
    Rotate yRotate;
    //Initialization of rotation
    parent.getTransforms().setAll
    (
      xRotate = new Rotate(0, Rotate.X_AXIS),
      yRotate = new Rotate(0, Rotate.Y_AXIS)
    );

    //Graphic is attached to root
    final Group root = new Group(parent);
    //Window of the scene
    final Scene scene = new Scene(root, 900, 900, true);
    //Background
    scene.setFill(Color.SKYBLUE);

    //Listener of mousePressed
    scene.setOnMousePressed
    (
        event ->
        {
```

```
            anchorX = event.getSceneX();
            anchorY = event.getSceneY();
            anchorAngleX = angleX.get();
            anchorAngleY = angleY.get();
        }
    );


    //Listener of mouseDragged
    scene.setOnMouseDragged
    (
        event ->
        {
            angleX.set(anchorAngleX
                    - (anchorY - event.getSceneY()));
            angleY.set(anchorAngleY
                    + anchorX - event.getSceneX());
        }
    );

    //Rotation gets bound to
    //the angle determined by  mouseDragged
    xRotate.angleProperty().bind(angleX);
    yRotate.angleProperty().bind(angleY);

    //Illumination
    PointLight pointLight = new PointLight(Color.WHITE);
    pointLight.setTranslateX(100);
    pointLight.setTranslateY(200);
    pointLight.setTranslateZ(-2000);
    root.getChildren().add(pointLight);

    //Camera
    scene.setCamera(camera);
    primaryStage.setScene(scene);
    PerspectiveCamera subSceneCamera
            = new PerspectiveCamera(false);
    scene.setCamera(subSceneCamera);

    //To zoom down the graphic,
```

```
//it is translated away from  the beholder,
//along the Z-direction (normal to the screen).
//Change zPos to change initial setting.
zSlider = new Slider();
zSlider.setMax(5000);
zSlider.setMin(1);


//Bind zPos to the reading of the slider
zSlider.valueProperty().bindBidirectional(zPos);
//Execute negative zoom
parent.translateZProperty().bind(zPos);
System.out.println("zPos = " + zPos);


//This slider   controls the closing of the hand.
angleSlider = new Slider();
angleSlider.setMax(90);
angleSlider.setMin(0);
angleSlider.setValue(10);

//To close the hand, an angle of rotation is enabled.
//Bind angleValue to the reading of the slider
angleSlider.valueProperty().bindBidirectional(angleValue);



//Other JavaFX controls, labels and containers
Label zLabel = new Label("Zoom");
HBox subSceneGroup = new HBox(70,
    new HBox(10, zLabel, zSlider, angleSlider));

subSceneGroup.setPadding(new Insets(10, 100, 0, 100));
SubScene subScene
        = new SubScene(subSceneGroup, 880, 200);
subScene.setLayoutX(10);
subScene.setLayoutY(800);
root.getChildren().add(subScene);

primaryStage.show();
```

```
}


public void play()
{

}

@Override public void stop()
{

}

@Override
public void start(Stage primaryStage)  throws Exception
{
    init(primaryStage);
    primaryStage.show();
    play();
}


public static void main(String[] args)
{
    launch(args);
}



private Node create3dContent()
{
    XformWithPivot sceneRoot = new XformWithPivot();
    sceneRoot.rx.setAngle(45);
    sceneRoot.ry.setAngle(110);
    getColors();
    //The constructor of hand is invoked here.
    //h and hCopy are instance of the main class Hand
    Hand  h =  new Hand();
    sceneRoot.getChildren().add(h);
    return new Group(sceneRoot);
```

```
}

//Colors with a name in JavaFX are put into an array
static private Color[] getColors()
{
    //Colors with a name in JavaFX
    final Field[] fields = Color.class.getFields();


    int N = 0;
    //Colors are put into an array
    for (final Field field : fields)
    {
        if (field.getType() == Color.class)
        {
            try
            {
                final Color color = (Color) field.get(null);
                //Array to keep colors
                JavaFXColors[N] = color;
                /*
                //The name of the color
                final String colorName = field.getName();
                System.out.println("N = " + N + "  "
                        + color.toString() + " " + colorName);
                */
                N++;
            } catch (IllegalArgumentException
                    | IllegalAccessException ex)
            {
                Logger.getLogger(HandOnlyBoxes2.class
                    .getName()).log(Level.SEVERE, null, ex);
            }
       }
    }

    return JavaFXColors;
}
```

```
//A random color is synthesized
 private static Color randomColor()
 {
    //A JavaFX named color is picked
    //but the first, Transparent, is ignored
    int n = rd.nextInt(JavaFXColors.length-1)+1;
    Color c = JavaFXColors[n];
    //Some light colors are not visualized so,
    //they might be darkened:
    //c = c.darker();
    return c;
 }


  //=====================================================
  //=============Beginning of Inner class Hand===========
  //=====================================================




private class Hand extends Group
{

    //Rotations about the center
    final Rotate rx = new Rotate(0,Rotate.X_AXIS);
    final Rotate ry = new Rotate(0,Rotate.Y_AXIS);
    final Rotate rz = new Rotate(0,Rotate.Z_AXIS);
    //A node with all elements
    XformWithPivot handGroup = new XformWithPivot();

    //Nested definition of finger nodes is necesary
    //so that a phalange drags the rest of the finger:

    //Group with phalanges 0 of all fingers + fingers1Group
    XformWithPivot fingers0Group = new XformWithPivot();
    //Group with phalanges 1 of all fingers + fingers2Group
    XformWithPivot fingers1Group = new XformWithPivot();
    ///Group with phalanges 2 of all fingers
```

```
XformWithPivot fingers2Group = new XformWithPivot();
//Group with all phalanges of all fingers
XformWithPivot fingersGroup = new XformWithPivot();
Hand()
{

    //Arm is constructed
    Box arm = new Box(100,20,400);
    Color c = randomColor();
    PhongMaterial materialArm = new PhongMaterial(c);
    arm.setMaterial(materialArm);
    arm.setTranslateY(20);
    //Palm is constructed
    Box palm = new Box(125,20,200);
    Color c2 = randomColor();
    PhongMaterial materialPalm = new PhongMaterial(c2);
    palm.setMaterial(materialPalm);
    palm.setTranslateZ(300);
    palm.setTranslateY(20);
    final Group armPalm = new Group(arm, palm);

    //5 fingers on the same line.
    //No opposable thumb
    Finger Fingers[] = new Finger[5];

    for(int i = 0; i < 5; i++)
    {
        //Each finger has 3 phalanges
        Fingers[i] = new Finger(3);
        for(int j =0; j < nPhalanges; j++)
        {
            Box ph = Fingers[i].Phalanges[j];
            ph.setTranslateX(12.5 * (5-2*(i))-12.5);
            ph.setTranslateZ(420 + 40*j);
            ph.setTranslateY(20);
            armPalm.getChildren().add(ph);
            if (j == 0) fingers0Group.getChildren().add(ph);
            if (j == 1) fingers1Group.getChildren().add(ph);
            if (j == 2) fingers2Group.getChildren().add(ph);
```

```
    }
}
//Groups are nested
fingers1Group.getChildren().addAll( fingers2Group);
fingers0Group.getChildren().addAll( fingers1Group);
fingersGroup.getChildren().addAll(fingers0Group);
handGroup.getChildren().addAll(armPalm, fingersGroup );
//The group of 5 fingers must rotate around
//the end of the palm that is parallel to the X_AXIS
//This is the pivot of rotation
fingersGroup.rwpx.pivotZProperty().setValue(400.0);
fingersGroup.rwpx.pivotYProperty().setValue(10.0);
//The angle of rotation is controlled by a slider
fingersGroup.rwpx.angleProperty()
        .bindBidirectional(angleValue);



//The group of  fingers past phalanx 0
//must rotate around
//the end of the 0-th phalanx.
//This is the pivot of rotation
fingers1Group.rwpx.pivotZProperty().setValue(440.0);
fingers1Group.rwpx.pivotYProperty().setValue(10.0);
//The angle of rotation is controlled by a slider
fingers1Group.rwpx.angleProperty()
        .bindBidirectional(angleValue);


//The group of  fingers past phalanx 1
//must rotate around
//the end of the 1st phalanx.
//This is the pivot of rotation
fingers2Group.rwpx.pivotZProperty().setValue(480.0);
fingers2Group.rwpx.pivotYProperty().setValue(10.0);
//The angle of rotation is controlled by a slider
fingers2Group.rwpx.angleProperty()
        .bindBidirectional(angleValue);

getChildren().addAll(handGroup );
```

```
    }//End of constructor of class Hand


}//End of inner class Hand


//A finger is an ensemble of phalanges
private class Finger extends Group
{
    final Rotate rx = new Rotate(0,Rotate.X_AXIS);
    final Rotate ry = new Rotate(0,Rotate.Y_AXIS);
    final Rotate rz = new Rotate(0,Rotate.Z_AXIS);
    //Our finger is a composition of 3 boxes or phalanges
    private   Box[] Phalanges    = new Box [nPhalanges];


    //Constructor
    Finger(int nPhalanges)
    {
        for(int i =0; i < nPhalanges; i++)
        {
             final Box p = new Box(25,20, 40);
             Color c = randomColor();
             PhongMaterial materialFinger
                     = new PhongMaterial(c);
             p.setMaterial(materialFinger);
             Phalanges[i] = p;
        }
    }
}//End of inner class Finger



//Kit of available families of transformations
private class XformWithPivot extends Group
{
    public Rotate rx = new Rotate();
    { rx.setAxis(Rotate.X_AXIS); }
    public Rotate ry = new Rotate();
    { ry.setAxis(Rotate.Y_AXIS); }
```

```
    public Rotate rz = new Rotate();
    { rz.setAxis(Rotate.Z_AXIS); }
    //Rotation = angle + pivot + axis.
    //Every rotation is around X_AXIS
    public Rotate rwpx = new Rotate(0,0,0,0, X_AXIS);

    public XformWithPivot()
    {
        super();
        getTransforms().addAll( rx, ry, rz, rwpx );
    }
}//End of class XformWithPivot

}//En of main class M14 HandOnlyBoxes2
```

15 ***Exercise.*** *Run the previous program and play with the code.*

16 ***Exercise.*** *Extend the control of the rotation slider to the wrist.* Answer

2.2 A hand for a pianist

We have seen how to make an ensemble of boxes into a 3D shape that somehow resembles a hand with all fingers on one line. A hand that can bend its fingers to form a grip, as our hand does it, also can be used to play the piano. An improvement that would ease its work would be to enable transversal movement of fingers.

17 ***Exercise.*** *Enable transversal opening of the fingers together with another slider to control it. This type of movement is very useful to deal with every sort of keyboards.* Answer

2.3 A hand that can feel

Let us now enable the sense of touch:

18 ***The following code shows a thumbless hand that is closable, that can move its fingers transversally and that reacts specifically to touching with the mouse.***

```
/*
 * Program M18 HandOnlyBoxes5.
 * It is shown how to make an ensemble of boxes
 * into a 3D closable hand
 * whose fingers can be moved transversally.
 * The hand lacks an opposable thumb.
 * The hand can react to touching with the mouse:
 * touched phalanx changes its color to red.

 * We copy the anatomical structure:
 * phalanges form fingers that together
 * with the palm conform the hand.
 * It is found that JavaFX is highly compatible
 * with the anatomical organization.
 */

package handonlyboxes5;

import java.lang.reflect.Field;
import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.beans.binding.NumberBinding;
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;
import javafx.geometry.Insets;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.PerspectiveCamera;
import javafx.scene.PointLight;
import javafx.scene.Scene;
import javafx.scene.SubScene;
import javafx.scene.control.Label;
import javafx.scene.control.Slider;
import javafx.scene.input.PickResult;
import javafx.scene.layout.HBox;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
```

```
import javafx.scene.shape.Box;
import javafx.scene.transform.Rotate;
import static javafx.scene.transform.Rotate.X_AXIS;
import static javafx.scene.transform.Rotate.Y_AXIS;
import static javafx.scene.transform.Rotate.Z_AXIS;
import javafx.stage.Stage;



public class HandOnlyBoxes5 extends Application
{
    //Node with all elements
    XformWithPivot handGroup = new XformWithPivot();
    XformWithPivot finger4Ph2Group = new XformWithPivot();
    //Slider for zooming
    private Slider zSlider;
    //Position of graphic along the z-axis,
    //normal to the screen.
    //It is a double property because
    //it will get involved into a binding
    private final DoubleProperty zPos
            = new SimpleDoubleProperty(2000);

    //Slider for angle of rotation of fingers
    private Slider closeHandSlider;
    //To close the hand,
    //each phalange suffers a rotation.
    //It is a double property because
    //it will get involved into a binding
    private final DoubleProperty closeProperty
            = new SimpleDoubleProperty(0);

    //Slider for opening fingers thanks to
    //their transversal rotation
    private Slider openingFingersSlider;
    //To open the fingers,
    //each phalange suffers a transversal rotation.
    //It is a double property because
    //it will get involved into a binding
    private final DoubleProperty openingFingersProperty
```

```
            = new SimpleDoubleProperty(0);
    private final DoubleProperty openingFingersProperty3
            = new SimpleDoubleProperty(0);


    //For listener of mouse
    private double anchorAngleX = 0;
    private double anchorAngleY = 0;
    private double anchorX, anchorY;
    //Angles of rotation of the scene
    private final DoubleProperty angleX
            = new SimpleDoubleProperty(0);
    private final DoubleProperty angleY
            = new SimpleDoubleProperty(0);

    PerspectiveCamera camera
            = new PerspectiveCamera(false);

    //The application is instantiated
    static HandOnlyBoxes5 p = new HandOnlyBoxes5();



    //An array with all colors that are named in JavaFX
    private static final Color JavaFXColors[]
                                = new Color[148];

    //Generator of pseudo random numbers
    private static final Random rd = new Random();

    //Number of fingers
    private final int nFingers = 5;

    //Number of phalanges of each finger
    private final int nPhalanges = 3;

public void init(Stage primaryStage)
{
    primaryStage.setTitle("An ensemble of boxes"
            + " act coherently as a thumbless hand");
```

```
//The whole drawing  is created
//as a JavaFX space-like object
final Group parent = new Group(create3dContent());

//Initial position
parent.setTranslateX(450);
parent.setTranslateY(400);
parent.setTranslateZ(0);
Rotate xRotate;
Rotate yRotate;
//Initialization of rotation
parent.getTransforms().setAll
(
  xRotate = new Rotate(0, Rotate.X_AXIS),
  yRotate = new Rotate(0, Rotate.Y_AXIS)
);

//Graphic is attached to root
final Group root = new Group(parent);
//Window of the scene
final Scene scene = new Scene(root, 900, 900, true);
//Background
scene.setFill(Color.SKYBLUE);

//Listener of mousePressed
scene.setOnMousePressed
(
    event ->
    {
        anchorX = event.getSceneX();
        anchorY = event.getSceneY();
        anchorAngleX = angleX.get();
        anchorAngleY = angleY.get();
    }
);


//Listener of mouseDragged
scene.setOnMouseDragged
(
```

```
        event ->
        {
            angleX.set(anchorAngleX
                    - (anchorY - event.getSceneY()));
            angleY.set(anchorAngleY
                    + anchorX - event.getSceneX());
        }
    );

    //Rotation of the scene gets bound to
    //the angle determined by  mouseDragged
    xRotate.angleProperty().bind(angleX);
    yRotate.angleProperty().bind(angleY);

    //Illumination
    PointLight pointLight = new PointLight(Color.WHITE);
    pointLight.setTranslateX(100);
    pointLight.setTranslateY(200);
    pointLight.setTranslateZ(-2000);
    root.getChildren().add(pointLight);

    //Camera
    scene.setCamera(camera);
    primaryStage.setScene(scene);
    PerspectiveCamera subSceneCamera
            = new PerspectiveCamera(false);
    scene.setCamera(subSceneCamera);

    //To zoom down the graphic,
    //it is translated away from  the beholder,
    //along the Z-direction (normal to the screen).
    //Change zPos to change initial setting.
    zSlider = new Slider();
    zSlider.setMax(5000);
    zSlider.setMin(1);


    //Bind zPos to the reading of the slider
    zSlider.valueProperty().bindBidirectional(zPos);
    //Execute negative zoom
```

```
parent.translateZProperty().bind(zPos);



//This slider   controls the closing of the hand.
closeHandSlider = new Slider();
closeHandSlider.setMax(90);
closeHandSlider.setMin(0);
closeHandSlider.setValue(10);

//To close the hand, an angle of rotation is enabled.
//Bind angleProperty to the reading of the slider
closeHandSlider.valueProperty()
        .bindBidirectional(closeProperty);


//This slider controls the openning of fingers.
openingFingersSlider = new Slider();
openingFingersSlider.setMax(15);
openingFingersSlider.setMin(0);
openingFingersSlider.setValue(10);

//To open fingers, an angle of rotation is enabled.
//Bind openingFingersProperty to the reading of the slider
openingFingersSlider.valueProperty()
        .bindBidirectional(openingFingersProperty);


//Other JavaFX controls, labels and containers
Label zLabel = new Label("Zoom");
Label angleLabel = new Label("Close hand");
HBox subSceneGroup = new HBox(70,
    new HBox(10, zLabel, zSlider,
            angleLabel, closeHandSlider, openingFingersSlider));

subSceneGroup.setPadding(new Insets(10, 100, 0, 100));
SubScene subScene
        = new SubScene(subSceneGroup, 880, 200);
subScene.setLayoutX(10);
subScene.setLayoutY(800);
```

```
root.getChildren().add(subScene);

primaryStage.show();

//The sense of touch is here
parent.setOnMouseClicked((event)->
{
    //Information that can be retrieved
    PickResult res = event.getPickResult();
    //Which node
    Node node = res.getIntersectedNode();
    //Id of node (null if not declared)
    String nodeId  = event.getPickResult()
            .getIntersectedNode().getId();
    //which face of the node (negative if is not enabled
    int face = event.getPickResult().getIntersectedFace();
    System.out.println("res = " + res);
    System.out.println("node = " + node);
    System.out.println("nodeID = " + nodeId);
    System.out.println("face" + face);
    //Change color to red, rotate around Z_AXIS
    if (node instanceof Box)
        {
            ((Box) node).setMaterial(
                    new PhongMaterial( Color.RED));
            handGroup.rotationAxisProperty().setValue(Z_AXIS);
            int val = rd.nextInt(91);
            handGroup.rotateProperty().setValue(val);
            finger4Ph2Group.rotateProperty().setValue(val);
        }



    /*
    //This does not function
    if (node instanceof XformWithPivot)
        {
            openingFingersSlider.valueProperty()
            .unbindBidirectional(openingFingersProperty);
            ((XformWithPivot) node).rwpx.setAngle(50);
```

```
                openingFingersSlider.valueProperty()
            .bindBidirectional(openingFingersProperty);


            }
        */

    });

}


public void play()
{

}

@Override public void stop()
{

}

@Override
public void start(Stage primaryStage)  throws Exception
{
    init(primaryStage);
    primaryStage.show();
    play();
}


public static void main(String[] args)
{
    launch(args);
}


//Make the drawing
private Node create3dContent()
{
```

```
    //The scene can be rotated
    XformWithPivot sceneRoot = new XformWithPivot();
    sceneRoot.rx.setAngle(45);
    sceneRoot.ry.setAngle(110);
    getColors();
    //The constructor of hand is invoked.
    Hand  h =  new Hand();
    sceneRoot.getChildren().add(h);
    return new Group(sceneRoot);
}

//Colors with a name in JavaFX are put into an array
static private Color[] getColors()
{
    //Colors with a name in JavaFX
    final Field[] fields = Color.class.getFields();


    int N = 0;
    //Colors are put into an array
    for (final Field field : fields)
    {
        if (field.getType() == Color.class)
        {
            try
            {
                final Color color = (Color) field.get(null);
                //Array to keep colors
                JavaFXColors[N] = color;
                /*
                //The name of the color
                final String colorName = field.getName();
                System.out.println("N = " + N + "  "
                        + color.toString() + " " + colorName);
                */
                N++;
            } catch (IllegalArgumentException
                    | IllegalAccessException ex)
            {
                Logger.getLogger(HandOnlyBoxes5.class
```

```
                    .getName()).log(Level.SEVERE, null, ex);
            }
        }
    }

    return JavaFXColors;
}


//A random color is synthesized
 private static Color randomColor()
 {
    //A JavaFX named color is picked
    //but the first, Transparent, is ignored
    int n = rd.nextInt(JavaFXColors.length-1)+1;
    Color c = JavaFXColors[n];
    //Some light colors are not visualized so,
    //they might be darkened:
    //c = c.darker();
    return c;
 }


  //=================================================
  //=============Inner class XformWithPivot===========
  //=================================================



//Kit of available families of transformations
private class XformWithPivot extends Group
{
    //Declaration and initialization
    public Rotate rx = new Rotate();
    { rx.setAxis(Rotate.X_AXIS); }
    public Rotate ry = new Rotate();
    { ry.setAxis(Rotate.Y_AXIS); }
    public Rotate rz = new Rotate();
    { rz.setAxis(Rotate.Z_AXIS); }
    //Rotation = angle (one coordinate)
```

```
    //+ pivot (3 coordinates) + axis.
    //This rotation is around X_AXIS
    public Rotate rwpx = new Rotate(0,0,0,0, X_AXIS);
    //This rotation is around Y_AXIS
    public Rotate rwpy = new Rotate(0,0,0,0, Y_AXIS);

    public XformWithPivot()
    {
        super();
        getTransforms().addAll( rx, ry, rz, rwpx, rwpy );
    }
}//End of class XformWithPivot



  //=========================================
  //============Inner class Hand============
  //=========================================



private class Hand extends Group
{



    //Nested definition of   nodes is implemented
    //to have, say,  the palm dragging the fingers,
    //and a phalanx dragging the rest of its the finger.


    //Each finger generates various natural groups:
    //Group with the extreme phalanx of finger0
    //Symbolic initialization
    XformWithPivot finger0Ph2Group = new XformWithPivot();
    //Group with the two extreme phalanges
    XformWithPivot finger0Ph21Group = new XformWithPivot();
    //Group with all three phalanges. It is the finger
    XformWithPivot finger0Ph210Group = new XformWithPivot();
    //Finger1
```

```
    XformWithPivot finger1Ph2Group = new XformWithPivot();
    XformWithPivot finger1Ph21Group = new XformWithPivot();
    XformWithPivot finger1Ph210Group = new XformWithPivot();
    //Finger2
    XformWithPivot finger2Ph2Group = new XformWithPivot();
    XformWithPivot finger2Ph21Group = new XformWithPivot();
    XformWithPivot finger2Ph210Group = new XformWithPivot();
    //Finger3
    XformWithPivot finger3Ph2Group = new XformWithPivot();
    XformWithPivot finger3Ph21Group = new XformWithPivot();
    XformWithPivot finger3Ph210Group = new XformWithPivot();
    //Finger4

    XformWithPivot finger4Ph21Group = new XformWithPivot();
    XformWithPivot finger4Ph210Group = new XformWithPivot();



    //Group with all  fingers
    //XformWithPivot allFingersgroup = new XformWithPivot();
    //Group with palm and fingers
    XformWithPivot palmGroup = new XformWithPivot();


Hand()
{

    //Arm is constructed
    Box arm = new Box(100,20,400);
    Color c = randomColor();
    PhongMaterial materialArm = new PhongMaterial(c);
    arm.setMaterial(materialArm);
    arm.setTranslateY(20);
    //Palm is constructed
    Box palm = new Box(125,20,140);
    Color c2 = randomColor();
    PhongMaterial materialPalm = new PhongMaterial(c2);
    palm.setMaterial(materialPalm);
    palm.setTranslateZ(270);
    palm.setTranslateY(20);
```

```
//5 fingers on the same line.
//No opposable thumb

Finger Fingers[] = new Finger[nFingers];

//Groups are constructed
for(int i = 0; i < nFingers; i++)
{
    //Each finger has nPhalanges phalanges
    Fingers[i] = new Finger(nPhalanges);
    for(int j =0; j < nPhalanges; j++)
    {
        Box ph = Fingers[i].Phalanges[j].box;

        ph.setTranslateX(12.5 * (5-2*(i))-12.5);
        ph.setTranslateZ(360 + 40*j);
        ph.setTranslateY(20);

        //Natural groups are instantiated
        //Finger 0 and its subgroups
        if (i == 0)
        {
            if (j == 2) finger0Ph2Group.getChildren().add(ph);

            if (j == 1) finger0Ph21Group
                .getChildren().addAll(finger0Ph2Group, ph);
            if (j == 0) finger0Ph210Group
                .getChildren().addAll(finger0Ph21Group, ph);
        }

        //Finger 1  and its subgroups
        if (i == 1)
        {
            if (j == 2) finger1Ph2Group.getChildren().add(ph);
            if (j == 1) finger1Ph21Group
                .getChildren().addAll(finger1Ph2Group, ph);
            if (j == 0) finger1Ph210Group
```

```
                .getChildren().addAll(finger1Ph21Group, ph);
        }


        //Finger 2  and its subgroups
        if (i == 2)
        {
            if (j == 2) finger2Ph2Group.getChildren().add(ph);
            if (j == 1) finger2Ph21Group
                .getChildren().addAll(finger2Ph2Group, ph);
            if (j == 0) finger2Ph210Group
                .getChildren().addAll(finger2Ph21Group, ph);
        }


        //Finger 3 and its subgroups
        if (i == 3)
        {
            if (j == 2) finger3Ph2Group.getChildren().add(ph);
            if (j == 1) finger3Ph21Group
                .getChildren().addAll(finger3Ph2Group, ph);
            if (j == 0) finger3Ph210Group
                .getChildren().addAll(finger3Ph21Group, ph);
        }

        //Finger 4 and its subgroups
        if (i == 4)
        {
            if (j == 2) finger4Ph2Group.getChildren().add(ph);
            if (j == 1) finger4Ph21Group
                .getChildren().addAll(finger4Ph2Group, ph);
            if (j == 0) finger4Ph210Group
                .getChildren().addAll(finger4Ph21Group, ph);
        }

    }
}

//Fingers are attached to the palm
palmGroup.getChildren().addAll(palm, finger0Ph210Group,
```

```
        finger1Ph210Group,
        finger2Ph210Group,
        finger3Ph210Group,
        finger4Ph210Group );
//Palm and arm are attached to hand
handGroup.getChildren().addAll(arm, palmGroup );



//========OPENING/CL0SING OF THE HAND============

NumberBinding closePropertyHalf
        = closeProperty.multiply(0.5);

//The slider that closes the hand extends its
//control to the palm:
//it must rotate around
//the end of the arm that is parallel to the X_AXIS
//This is the pivot of rotation
palmGroup.rwpx.pivotZProperty().setValue(200.0);
palmGroup.rwpx.pivotYProperty().setValue(10.0);
//The angle of rotation is controlled by a slider
palmGroup.rwpx.angleProperty()
        .bind(closePropertyHalf);



//Specifications of movement for subgroups of finger 0
finger0Ph2Group.rwpx.pivotZProperty().setValue(420.0);
finger0Ph2Group.rwpx.pivotYProperty().setValue(10.0);
finger0Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger0Ph21Group.rwpx.pivotZProperty().setValue(380.0);
finger0Ph21Group.rwpx.pivotYProperty().setValue(10.0);
finger0Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger0Ph210Group.rwpx.pivotZProperty().setValue(340.0);
finger0Ph210Group.rwpx.pivotYProperty().setValue(10.0);
```

```
finger0Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

//Specifications of movement for subgroups of finger 1
finger1Ph2Group.rwpx.pivotZProperty().setValue(420.0);
finger1Ph2Group.rwpx.pivotYProperty().setValue(10.0);
finger1Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger1Ph21Group.rwpx.pivotZProperty().setValue(380.0);
finger1Ph21Group.rwpx.pivotYProperty().setValue(10.0);
finger1Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger1Ph210Group.rwpx.pivotZProperty().setValue(340.0);
finger1Ph210Group.rwpx.pivotYProperty().setValue(10.0);
finger1Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

//Specifications of movement for subgroups of finger 2
finger2Ph2Group.rwpx.pivotZProperty().setValue(420.0);
finger2Ph2Group.rwpx.pivotYProperty().setValue(10.0);
finger2Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger2Ph21Group.rwpx.pivotZProperty().setValue(380.0);
finger2Ph21Group.rwpx.pivotYProperty().setValue(10.0);
finger2Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger2Ph210Group.rwpx.pivotZProperty().setValue(340.0);
finger2Ph210Group.rwpx.pivotYProperty().setValue(10.0);
finger2Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

//Specifications of movement for subgroups of finger 3
finger3Ph2Group.rwpx.pivotZProperty().setValue(420.0);
finger3Ph2Group.rwpx.pivotYProperty().setValue(10.0);
finger3Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);
```

```
finger3Ph21Group.rwpx.pivotZProperty().setValue(380.0);
finger3Ph21Group.rwpx.pivotYProperty().setValue(10.0);
finger3Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger3Ph210Group.rwpx.pivotZProperty().setValue(340.0);
finger3Ph210Group.rwpx.pivotYProperty().setValue(10.0);
finger3Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

//Specifications of movement for subgroups of finger 4
finger4Ph2Group.rwpx.pivotZProperty().setValue(420.0);
finger4Ph2Group.rwpx.pivotYProperty().setValue(10.0);
finger4Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger4Ph21Group.rwpx.pivotZProperty().setValue(380.0);
finger4Ph21Group.rwpx.pivotYProperty().setValue(10.0);
finger4Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger4Ph210Group.rwpx.pivotZProperty().setValue(340.0);
finger4Ph210Group.rwpx.pivotYProperty().setValue(10.0);
finger4Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

//======OPENING/CLOSING OF FINGERS========

//Fingers that are at the border must
//open twice as those
//near the center.
NumberBinding openingFingersPropertyTwo
        = openingFingersProperty.multiply(2);
//Specifications of movement for subgroups of finger 0
finger0Ph210Group.rwpy.pivotXProperty().setValue(0.0);
finger0Ph210Group.rwpy.pivotYProperty().setValue(10.0);
finger0Ph210Group.rwpy.pivotZProperty().setValue(340.0);
finger0Ph210Group.rwpy.angleProperty()
        .bind(openingFingersPropertyTwo);
```

```
    //Specifications of movement for subgroups of finger 1
    finger1Ph210Group.rwpy.pivotXProperty().setValue(0.0);
    finger1Ph210Group.rwpy.pivotYProperty().setValue(30.0);
    finger1Ph210Group.rwpy.pivotZProperty().setValue(340.0);

    finger1Ph210Group.rwpy.angleProperty()
            .bind(openingFingersProperty);

    //Finger two remains invariant under opening of fingers

    //Opening in the contrary direction of fingers 3 and 4
    NumberBinding openingFingersPropertyMinus
            = openingFingersProperty.multiply(-1);

    //Specifications of movement for finger 3
    finger3Ph210Group.rwpy.pivotXProperty().setValue(0.0);
    finger3Ph210Group.rwpy.pivotYProperty().setValue(30.0);
    finger3Ph210Group.rwpy.pivotZProperty().setValue(340.0);
    finger3Ph210Group.rwpy.angleProperty()
            .bind(openingFingersPropertyMinus);

    NumberBinding openingFingersPropertyMinusTwo
            = openingFingersProperty.multiply(-2);

    //Specifications of movement for finger 4
    finger4Ph210Group.rwpy.pivotXProperty().setValue(0.0);
    finger4Ph210Group.rwpy.pivotYProperty().setValue(10.0);
    finger4Ph210Group.rwpy.pivotZProperty().setValue(340.0);
    finger4Ph210Group.rwpy.angleProperty()
            .bind(openingFingersPropertyMinusTwo);

    getChildren().addAll(handGroup );

}//End of constructor of class Hand


}//End of inner class Hand
```

```
//A finger is an ensemble of phalanges
private class Finger extends Group
{

    Phalanx[] Phalanges    = new Phalanx [nPhalanges];

    //Constructor
    Finger(int nPhalanges)
    {
        for(int i =0; i < nPhalanges; i++)
        {
            Phalanx ph = new Phalanx(25,20,40);
            Phalanges[i] = ph;
        }
    }
}//End of inner class Finger



//A phalanx is a JavaFX 3D box
private class Phalanx extends Group
{
    Box box;

    //Constructor
    Phalanx(int width, int length, int height)
    {
        box = new Box(width, length, height);
        Color c = randomColor();
        PhongMaterial materialFinger
                = new PhongMaterial(c);
        box.setMaterial(materialFinger);
        //ph.getChildren().add(box);
    }
}//End of inner class phalanx

}//En of main class M18 HandOnlyBoxes5
```

19 ***Exercise.*** *Run the previous program and play with the code.*

We have seen how to change the color of a box by clicking over it and how to produce a rotation around the `Z_AXIS` as a reflex movement. To pursue an extension of the possibilities of rotation to include `X_AXIS` and `Y_AXIS`, we must keep in mind that rotations around those axes have been bound to a slider and that bound properties cannot be modified at will. The next code shows one way as these troubles can be circumvented to create a sense of touch with specific answer thanks to implementing a reflex for every phalanx group.

20 ***A hand with a sense of touch is implemented by the next code.***

```
/*
 * Program M20 HandOnlyBoxes6.
 * It is shown how to make an ensemble of boxes
 * into a 3D hand that can be rotated with the mouse
 * and that have sense of touch with
 * particularized reaction.
 * The hand lacks an opposable thumb.

 * We copy the anatomical structure:
 * phalanges form fingers that together
 * with the palm conform the hand.
 * It is found that JavaFX is highly compatible
 * with the anatomical organization.

 * Inspired by Swing based program
 * D146 OppThumbImproved in Vol IV.
 * The Author imagined that that program
 * would easily be transcribed to JavaFX
 * but early experience showed that
 * that was not the case:
 * present program was written de novo
 * without much appeal to the swing program.
 * This failure explains why JavaFX exists
 * instead of an improved version of Swing.
 * One has every right to claim that
 * we have here a case of speciation
```

```
 * (in the sense of complexity theory
 * of software development).
 */

package handonlyboxes6;

import java.lang.reflect.Field;
import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;
import javafx.event.EventHandler;
import javafx.geometry.Insets;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.PerspectiveCamera;
import javafx.scene.PointLight;
import javafx.scene.Scene;
import javafx.scene.SubScene;
import javafx.scene.control.Label;
import javafx.scene.control.Slider;
import javafx.scene.input.MouseEvent;
import javafx.scene.input.PickResult;
import javafx.scene.layout.HBox;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.Box;
import javafx.scene.transform.Rotate;
import static javafx.scene.transform.Rotate.X_AXIS;
import static javafx.scene.transform.Rotate.Y_AXIS;
import static javafx.scene.transform.Rotate.Z_AXIS;
import javafx.stage.Stage;



public class HandOnlyBoxes6 extends Application
{
```

```
 //Nested definition of   nodes is implemented
//to have, say,  the palm dragging the fingers,
//and a phalanx dragging the rest of its the finger.


//Each finger generates various natural groups of control:
//Group with the extreme phalanx of finger0
//Symbolic initialization
XformWithPivot finger0Ph2Group = new XformWithPivot();
//Group with the two extreme phalanges
XformWithPivot finger0Ph21Group = new XformWithPivot();
//Group with all three phalanges. It is the finger
XformWithPivot finger0Ph210Group = new XformWithPivot();
//Finger1
XformWithPivot finger1Ph2Group = new XformWithPivot();
XformWithPivot finger1Ph21Group = new XformWithPivot();
XformWithPivot finger1Ph210Group = new XformWithPivot();
//Finger2
XformWithPivot finger2Ph2Group = new XformWithPivot();
XformWithPivot finger2Ph21Group = new XformWithPivot();
XformWithPivot finger2Ph210Group = new XformWithPivot();
//Finger3
XformWithPivot finger3Ph2Group = new XformWithPivot();
XformWithPivot finger3Ph21Group = new XformWithPivot();
XformWithPivot finger3Ph210Group = new XformWithPivot();
//Finger4
XformWithPivot finger4Ph2Group = new XformWithPivot();
XformWithPivot finger4Ph21Group = new XformWithPivot();
XformWithPivot finger4Ph210Group = new XformWithPivot();

//Group with all  fingers
//XformWithPivot allFingersgroup = new XformWithPivot();
//Group with palm and fingers
XformWithPivot handGroup = new XformWithPivot();
//Node with all elements
XformWithPivot armGroup = new XformWithPivot();


//Slider for zooming
```

```
private Slider zSlider;
//Position of graphic along the z-axis,
//normal to the screen.
//It is a double property because
//it will get involved into a binding
private final DoubleProperty zPos
        = new SimpleDoubleProperty(2000);


//For listener of mouse
private double anchorAngleX = 0;
private double anchorAngleY = 0;
private double anchorX, anchorY;
//Angles of rotation of the scene
private final DoubleProperty angleX
        = new SimpleDoubleProperty(0);
private final DoubleProperty angleY
        = new SimpleDoubleProperty(0);

PerspectiveCamera camera
        = new PerspectiveCamera(false);

//The application is instantiated
static HandOnlyBoxes6 p = new HandOnlyBoxes6();



//An array with all colors that are named in JavaFX
private static final Color JavaFXColors[]
                            = new Color[148];

//Generator of pseudo random numbers
private static final Random rd = new Random();

//Number of fingers
private final int nFingers = 5;

//Number of phalanges of each finger
private final int nPhalanges = 3;
```

```
public void init(Stage primaryStage)
{


    primaryStage.setTitle("An ensemble of boxes"
            + " act coherently as a thumbless hand");
    //The whole drawing  is created
    //as a JavaFX space-like object
    final Group parent = new Group(create3dContent());

    //Initial position
    parent.setTranslateX(450);
    parent.setTranslateY(400);
    parent.setTranslateZ(0);
    Rotate xRotate;
    Rotate yRotate;
    //Initialization of rotation
    parent.getTransforms().setAll
    (
      xRotate = new Rotate(0, Rotate.X_AXIS),
      yRotate = new Rotate(0, Rotate.Y_AXIS)
    );

    //Graphic is attached to root
    final Group root = new Group(parent);
    //Window of the scene
    final Scene scene = new Scene(root, 900, 900, true);
    //Background
    scene.setFill(Color.SKYBLUE);

    //Listener of mousePressed
    scene.setOnMousePressed
    (
        event ->
        {
            anchorX = event.getSceneX();
            anchorY = event.getSceneY();
            anchorAngleX = angleX.get();
            anchorAngleY = angleY.get();
        }
```

```
);


//Listener of mouseDragged
scene.setOnMouseDragged
(
    event ->
    {
        angleX.set(anchorAngleX
                - (anchorY - event.getSceneY()));
        angleY.set(anchorAngleY
                + anchorX - event.getSceneX());
    }
);

//Rotation of the scene gets bound to
//the angle determined by  mouseDragged
xRotate.angleProperty().bind(angleX);
yRotate.angleProperty().bind(angleY);

//Illumination
PointLight pointLight = new PointLight(Color.WHITE);
pointLight.setTranslateX(100);
pointLight.setTranslateY(200);
pointLight.setTranslateZ(-2000);
root.getChildren().add(pointLight);

//Camera
scene.setCamera(camera);
primaryStage.setScene(scene);
PerspectiveCamera subSceneCamera
        = new PerspectiveCamera(false);
scene.setCamera(subSceneCamera);

//To zoom down the graphic,
//it is translated away from  the beholder,
//along the Z-direction (normal to the screen).
//Change zPos to change initial setting.
zSlider = new Slider();
zSlider.setMax(5000);
```

```
zSlider.setMin(1);


//Bind zPos to the reading of the slider
zSlider.valueProperty().bindBidirectional(zPos);
//Execute negative zoom
parent.translateZProperty().bind(zPos);







//Other JavaFX controls, labels and containers
Label zLabel = new Label("Zoom");

HBox subSceneGroup = new HBox(70,
    new HBox(10, zLabel, zSlider));

subSceneGroup.setPadding(new Insets(10, 100, 0, 100));
SubScene subScene
        = new SubScene(subSceneGroup, 880, 200);
subScene.setLayoutX(10);
subScene.setLayoutY(800);
root.getChildren().add(subScene);

primaryStage.show();

//The sense of touch is here
parent.setOnMouseClicked((MouseEvent event) ->
{
    //Information that can be retrieved
    PickResult res = event.getPickResult();
    //Which node
    Node node = res.getIntersectedNode();
    //Id of node (null if not declared)
    String nodeId  = event.getPickResult()
            .getIntersectedNode().getId();
```

```
//which face of the node (negative if is not enabled
int face = event.getPickResult()
        .getIntersectedFace();
System.out.println("res = " + res);
System.out.println("node = " + node);
System.out.println("nodeID = " + nodeId);
System.out.println("face" + face);



//This is how reflexes are implemented
if (node instanceof Box)
{
    //Name tag is reported
    System.out.println("ID of node " + nodeId);
    //Color is changed to random color
    ((Box) node).setMaterial(
            new PhongMaterial( randomColor()));
    //Angle of rotation
    double angle;
    //Reflex is programmed specifically
    switch (nodeId)
    {
        case "Arm" :
        {
            armGroup.rotationAxisProperty()
                    .setValue(Z_AXIS);
            int val = rd.nextInt(91);
            armGroup.rotateProperty()
                    .setValue(val);
            break;
        }

        case "Palm" :
        {
            System.out.println("Palm detected "  );
            handGroup.rwpx.pivotZProperty()
                    .setValue(200.0);
            handGroup.rwpx.pivotYProperty()
                    .setValue(10.0);
```

```
    angle = rd.nextInt(91);
    handGroup.rwpx.angleProperty()
            .setValue(angle);
    break;
}

//Specifications of movement
//for subgroups of finger 0
case "Phalanx00" :
{

    finger0Ph210Group.rwpx
            .pivotZProperty()
            .setValue(340.0);
    finger0Ph210Group.rwpx
            .pivotYProperty()
            .setValue(10.0);
    angle = rd.nextInt(91);
    finger0Ph210Group.rwpx
            .angleProperty()
            .setValue(angle);
    break;
}

case "Phalanx01" :
{

    finger0Ph21Group.rwpx.
            pivotZProperty()
            .setValue(380.0);
    finger0Ph21Group.rwpx
            .pivotYProperty()
            .setValue(10.0);
    angle = rd.nextInt(91);
    finger0Ph21Group.rwpx
            .angleProperty()
            .setValue(angle);
    break;
}
```

```
case "Phalanx02" :
{

    finger0Ph2Group.rwpx
            .pivotZProperty()
            .setValue(420.0);
    finger0Ph2Group.rwpx
            .pivotYProperty()
            .setValue(10.0);
    angle = rd.nextInt(91);
    finger0Ph2Group.rwpx
            .angleProperty()
            .setValue(angle);
    break;
}

//Specifications of movement
//for subgroups of finger 1
case "Phalanx10" :
{

    finger1Ph210Group.rwpx
            .pivotZProperty()
            .setValue(340.0);
    finger1Ph210Group.rwpx
            .pivotYProperty()
            .setValue(10.0);
    angle = rd.nextInt(91);
    finger1Ph210Group.rwpx
            .angleProperty()
            .setValue(angle);
    break;
}

case "Phalanx11" :
{

    finger1Ph21Group.rwpx
            .pivotZProperty()
            .setValue(380.0);
```

```
        finger1Ph21Group.rwpx
                .pivotYProperty()
                .setValue(10.0);
        angle = rd.nextInt(91);
        finger1Ph21Group.rwpx
                .angleProperty()
                .setValue(angle);
        break;
    }

    case "Phalanx12" :
    {

        finger1Ph2Group.rwpx
                .pivotZProperty()
                .setValue(420.0);
        finger1Ph2Group.rwpx
                .pivotYProperty()
                .setValue(10.0);
        angle = rd.nextInt(91);
        finger1Ph2Group.rwpx
                .angleProperty()
                .setValue(angle);
        break;
    }



    //Specifications of movement for subgroups of finger 2
    case "Phalanx20" :
    {

        finger2Ph210Group.rwpx
                .pivotZProperty()
                .setValue(340.0);
        finger2Ph210Group.rwpx
                .pivotYProperty()
                .setValue(10.0);
        angle = rd.nextInt(91);
        finger2Ph210Group.rwpx
```

```
                .angleProperty()
                .setValue(angle);
        break;
    }

    case "Phalanx21" :
    {

        finger2Ph21Group.rwpx
                .pivotZProperty()
                .setValue(380.0);
        finger2Ph21Group.rwpx
                .pivotYProperty()
                .setValue(10.0);
        angle = rd.nextInt(91);
        finger2Ph21Group.rwpx
                .angleProperty()
                .setValue(angle);
        break;
    }

    case "Phalanx22" :
    {

        finger2Ph2Group.rwpx
                .pivotZProperty()
                .setValue(420.0);
        finger2Ph2Group.rwpx
                .pivotYProperty()
                .setValue(10.0);
        angle = rd.nextInt(91);
        finger2Ph2Group.rwpx
                .angleProperty()
                .setValue(angle);
        break;
    }

    //Specifications of movement for subgroups of finger 3
    case "Phalanx30" :
    {
```

```
        finger3Ph210Group.rwpx
                .pivotZProperty()
                .setValue(340.0);
        finger3Ph210Group.rwpx
                .pivotYProperty()
                .setValue(10.0);
        angle = rd.nextInt(91);
        finger3Ph210Group.rwpx
                .angleProperty()
                .setValue(angle);
        break;
    }

    case "Phalanx31" :
    {

        finger3Ph21Group.rwpx
                .pivotZProperty()
                .setValue(380.0);
        finger3Ph21Group.rwpx
                .pivotYProperty()
                .setValue(10.0);
        angle = rd.nextInt(91);
        finger3Ph21Group.rwpx
                .angleProperty()
                .setValue(angle);
        break;
    }

    case "Phalanx32" :
    {

        finger3Ph2Group.rwpx
                .pivotZProperty()
                .setValue(420.0);
        finger3Ph2Group.rwpx
                .pivotYProperty()
                .setValue(10.0);
        angle = rd.nextInt(91);
```

```
        finger3Ph2Group.rwpx
                .angleProperty()
                .setValue(angle);
        break;
    }

    //Specifications of movement for subgroups of finger 4
    case "Phalanx40" :
    {

        finger4Ph210Group.rwpx
                .pivotZProperty()
                .setValue(340.0);
        finger4Ph210Group.rwpx
                .pivotYProperty()
                .setValue(10.0);
        angle = rd.nextInt(91);
        finger4Ph210Group.rwpx
                .angleProperty()
                .setValue(angle);
        break;
    }

    case "Phalanx41" :
    {

        finger4Ph21Group.rwpx
                .pivotZProperty()
                .setValue(380.0);
        finger4Ph21Group.rwpx
                .pivotYProperty()
                .setValue(10.0);
        angle = rd.nextInt(91);
        finger4Ph21Group.rwpx
                .angleProperty()
                .setValue(angle);
        break;
    }

    case "Phalanx42" :
```

```
                {

                    finger4Ph2Group.rwpx
                            .pivotZProperty()
                            .setValue(420.0);
                    finger4Ph2Group.rwpx
                            .pivotYProperty()
                            .setValue(10.0);
                    angle = rd.nextInt(91);
                    finger4Ph2Group.rwpx
                            .angleProperty()
                            .setValue(angle);
                    break;
                }
            }
        } //End if(node)
    });
}


public void play()
{

}

@Override public void stop()
{

}

@Override
public void start(Stage primaryStage)  throws Exception
{
    init(primaryStage);
    primaryStage.show();
    play();
}


public static void main(String[] args)
```

```
{
    launch(args);
}


//Make the drawing
private Node create3dContent()
{
    //The scene can be rotated
    XformWithPivot sceneRoot = new XformWithPivot();
    sceneRoot.rx.setAngle(45);
    sceneRoot.ry.setAngle(110);
    getColors();
    //The constructor of hand is invoked.
    Hand  h =  new Hand();
    sceneRoot.getChildren().add(h);
    return new Group(sceneRoot);
}


//Colors with a name in JavaFX are put into an array
static private Color[] getColors()
{
    //Colors with a name in JavaFX
    final Field[] fields = Color.class.getFields();


    int N = 0;
    //Colors are put into an array
    for (final Field field : fields)
    {
        if (field.getType() == Color.class)
        {
            try
            {
                final Color color = (Color) field.get(null);
                //Array to keep colors
                JavaFXColors[N] = color;
                /*
                //The name of the color
```

```
                final String colorName = field.getName();
                System.out.println("N = " + N + "   "
                        + color.toString() + " " + colorName);
                */
                N++;
            } catch (IllegalArgumentException
                    | IllegalAccessException ex)
            {
                Logger.getLogger(HandOnlyBoxes6.class
                    .getName()).log(Level.SEVERE, null, ex);
            }
        }
    }

    return JavaFXColors;
}


//A random color is synthesized
 private static Color randomColor()
 {
    //A JavaFX named color is picked
    //but the first, Transparent, is ignored
    int n = rd.nextInt(JavaFXColors.length-1)+1;
    Color c = JavaFXColors[n];
    //Some light colors are not visualized so,
    //they might be darkened:
    //c = c.darker();
    return c;
 }


  //=================================================
  //============Inner class XformWithPivot===========
  //=================================================



//Kit of available families of transformations
private class XformWithPivot extends Group
```

```
{
    //Declaration and initialization
    public Rotate rx = new Rotate();
    { rx.setAxis(Rotate.X_AXIS); }
    public Rotate ry = new Rotate();
    { ry.setAxis(Rotate.Y_AXIS); }
    public Rotate rz = new Rotate();
    { rz.setAxis(Rotate.Z_AXIS); }
    //Rotation = angle (one coordinate)
    //+ pivot (3 coordinates) + axis.
    //This rotation is around X_AXIS
    public Rotate rwpx = new Rotate(0,0,0,0, X_AXIS);
    //This rotation is around Y_AXIS
    public Rotate rwpy = new Rotate(0,0,0,0, Y_AXIS);

    public XformWithPivot()
    {
        super();
        getTransforms().addAll( rx, ry, rz, rwpx, rwpy );
    }
}//End of class XformWithPivot



  //=========================================
  //============Inner class Hand============
  //=========================================



private class Hand extends Group
{
```

```
Hand()
{

    //Arm is constructed
    Box arm = new Box(100,20,400);
    //Name tag is given
    arm.setId("Arm");
    Color c = randomColor();
    PhongMaterial materialArm = new PhongMaterial(c);
    arm.setMaterial(materialArm);
    arm.setTranslateY(20);
    //Palm is constructed
    Box palm = new Box(125,20,140);
    //Name tag is given
    palm.setId("Palm");
    Color c2 = randomColor();
    PhongMaterial materialPalm = new PhongMaterial(c2);
    palm.setMaterial(materialPalm);
    palm.setTranslateZ(270);
    palm.setTranslateY(20);



    //5 fingers on the same line.
    //No opposable thumb

    Finger Fingers[] = new Finger[nFingers];

    //Groups (for control) are constructed
    for(int i = 0; i < nFingers; i++)
    {
        //Each finger has nPhalanges phalanges
        Fingers[i] = new Finger(nPhalanges);
        for(int j =0; j < nPhalanges; j++)
        {
            Box ph = Fingers[i].Phalanges[j].box;
            //Identifer is given
            ph.setId("Phalanx" + i + j);
```

```
ph.setTranslateX(12.5 * (5-2*(i))-12.5);
ph.setTranslateZ(360 + 40*j);
ph.setTranslateY(20);

//Natural groups are instantiated
//Finger 0 and its subgroups
if (i == 0)
{
    if (j == 2) finger0Ph2Group.getChildren().add(ph);

    if (j == 1) finger0Ph21Group
        .getChildren().addAll(finger0Ph2Group, ph);
    if (j == 0) finger0Ph210Group
        .getChildren().addAll(finger0Ph21Group, ph);
}

//Finger 1  and its subgroups
if (i == 1)
{
    if (j == 2) finger1Ph2Group.getChildren().add(ph);
    if (j == 1) finger1Ph21Group
        .getChildren().addAll(finger1Ph2Group, ph);
    if (j == 0) finger1Ph210Group
        .getChildren().addAll(finger1Ph21Group, ph);
}


//Finger 2  and its subgroups
if (i == 2)
{
    if (j == 2) finger2Ph2Group.getChildren().add(ph);
    if (j == 1) finger2Ph21Group
        .getChildren().addAll(finger2Ph2Group, ph);
    if (j == 0) finger2Ph210Group
        .getChildren().addAll(finger2Ph21Group, ph);
}


//Finger 3 and its subgroups
if (i == 3)
```

```
            {
                if (j == 2) finger3Ph2Group.getChildren().add(ph);
                if (j == 1) finger3Ph21Group
                    .getChildren().addAll(finger3Ph2Group, ph);
                if (j == 0) finger3Ph210Group
                    .getChildren().addAll(finger3Ph21Group, ph);
            }

            //Finger 4 and its subgroups
            if (i == 4)
            {
                if (j == 2) finger4Ph2Group.getChildren().add(ph);
                if (j == 1) finger4Ph21Group
                    .getChildren().addAll(finger4Ph2Group, ph);
                if (j == 0) finger4Ph210Group
                    .getChildren().addAll(finger4Ph21Group, ph);
            }

        }
    }

    //Fingers and palm conform handGroup
    handGroup.getChildren().addAll(
            palm,
            finger0Ph210Group,
            finger1Ph210Group,
            finger2Ph210Group,
            finger3Ph210Group,
            finger4Ph210Group );
    //Palm and arm are attached to armGroup
    armGroup.getChildren().addAll(arm, handGroup );
    getChildren().addAll(armGroup );

}//End of constructor of class Hand


}//End of inner class Hand
```

```
//A finger is an ensemble of phalanges
private class Finger extends Group
{

    Phalanx[] Phalanges    = new Phalanx [nPhalanges];

    //Constructor
    Finger(int nPhalanges)
    {
        for(int i =0; i < nPhalanges; i++)
        {
            Phalanx ph = new Phalanx(25,20,40);
            Phalanges[i] = ph;
        }
    }
}//End of inner class Finger



//A phalanx is a JavaFX 3D box
private class Phalanx extends Group
{
    Box box;

    //Constructor
    Phalanx(int width, int length, int height)
    {
        box = new Box(width, length, height);
        Color c = randomColor();
        PhongMaterial materialFinger
                = new PhongMaterial(c);
        box.setMaterial(materialFinger);
        //ph.getChildren().add(box);
    }
}//End of inner class phalanx

}//En of main class M20 HandOnlyBoxes6
```

21 ***Exercise.*** *Run the previous program and play with the code.*

2.4 From a hand to a pair of pliers

To be able to close the hand to form a grip is quite different than being able to grasp things to manipulate them.

22 ***Exercise.*** *With the minimum number of changes remade someone of the hands previously shown into a hand with the function of a pair of pliers. This is achieved in our species by an opposable thumb but this is not the simplest evolutionary solution.* Answer

2.5 A hand with opposite thumb

Let us see how to devise a hand with an opposite thumb.

23 ***Exercise.*** *Devise a hand with opposite thumb, i.e., its thumb faces other fingers and allows a strong grip upon closing.* Answer

2.6 A hand with opposable thumb

The hand of apes does not have an opposite thumb, as some robots do. Instead, it has an opposable thumb which is a thumb that is aligned with the palm but after a complex movement can opposite other fingers to form a tight grip.

24 ***Let us devise a hand with opposable thumb, i.e., its thumb is aligned with other fingers but after a complex movement it can oppose them to make a strong grip upon closing.***

```
package handonlyboxes9;

/*
 * Program M24 HandOnlyBoxes9.
 * It is shown how to make an ensemble of boxes
 * into a 3D closable hand
 * whose fingers can be moved transversally.
 * The hand has an opposable thumb, i.e.,
 * its thumb is aligned with the other fingers
```

```
 * but after a complex movement opposites them
 * to form a strong grip upon closing.
 * The hand can react to touching with the mouse:
 * touched phalanx changes its color to red.

 * We try to copy the anatomical structure:
 * phalanges form fingers that together
 * with the palm conform the hand.
 * The thumb is initially rotated
 * around the Z_AXIS and around the X_AXIS,
 * just as our hands.

 * It is found that JavaFX is highly compatible
 * with the anatomical organization
 * and that all this is very complex.
 */



import java.lang.reflect.Field;
import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.beans.binding.NumberBinding;
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;
import javafx.geometry.Insets;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.PerspectiveCamera;
import javafx.scene.PointLight;
import javafx.scene.Scene;
import javafx.scene.SubScene;
import javafx.scene.control.Label;
import javafx.scene.control.Slider;
import javafx.scene.input.PickResult;
import javafx.scene.layout.GridPane;
import javafx.scene.paint.Color;
```

```
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.Box;
import javafx.scene.transform.Rotate;
import static javafx.scene.transform.Rotate.X_AXIS;
import static javafx.scene.transform.Rotate.Y_AXIS;
import static javafx.scene.transform.Rotate.Z_AXIS;
import javafx.stage.Stage;



public class HandOnlyBoxes9 extends Application
{
    //Node with all elements
    XformWithPivot armGroup = new XformWithPivot();
    XformWithPivot finger4Ph2Group = new XformWithPivot();
    //Slider for zooming
    private Slider zSlider;
    //Position of graphic along the z-axis,
    //normal to the screen.
    //It is a double property because
    //it will get involved into a binding
    private final DoubleProperty zPos
            = new SimpleDoubleProperty(2000);

    //Slider for angle of rotation of fingers
    private Slider closeHandSlider;
    //To close the hand,
    //each phalange suffers a rotation.
    //It is a double property because
    //it will get involved into a binding
    private final DoubleProperty closeProperty
            = new SimpleDoubleProperty(0);

    //Slider for opening fingers thanks to
    //their transversal rotation
    private Slider openFingersSlider;
    //To open the fingers,
    //each phalange suffers a transversal rotation.
    //It is a double property because
    //it will get involved into a binding
```

```
    private final DoubleProperty openFingersProperty
            = new SimpleDoubleProperty(0);



    //Slider for opening closing the thumb
    private Slider rotateThumbSlider;
    //To close the thumb we need a rotation.
    //It is a double property because
    //it will get involved into a binding
    private final DoubleProperty rotateThumbProperty
            = new SimpleDoubleProperty(0);

    //For listener of mouse
    private double anchorAngleX = 0;
    private double anchorAngleY = 0;
    private double anchorX, anchorY;
    //Angles of rotation of the scene
    private final DoubleProperty angleX
            = new SimpleDoubleProperty(0);
    private final DoubleProperty angleY
            = new SimpleDoubleProperty(0);

    PerspectiveCamera camera
            = new PerspectiveCamera(false);

    //The application is instantiated
    static HandOnlyBoxes9 p = new HandOnlyBoxes9();



    //An array with all colors that are named in JavaFX
    private static final Color JavaFXColors[]
                                = new Color[148];

    //Generator of pseudo random numbers
    private static final Random rd = new Random();

    //Number of fingers
    private final int nFingers = 4;
```

```
    //Number of phalanges of each finger
    private final int nPhalanges = 3;

public void init(Stage primaryStage)
{
    primaryStage.setTitle("An ensemble of boxes"
            + " act coherently as a   hand");
    //The whole drawing  is created
    //as a JavaFX space-like object
    final Group parent = new Group(create3dContent());

    //Initial position
    parent.setTranslateX(450);
    parent.setTranslateY(400);
    parent.setTranslateZ(0);
    Rotate xRotate;
    Rotate yRotate;
    //Initialization of rotation
    parent.getTransforms().setAll
    (
      xRotate = new Rotate(0, Rotate.X_AXIS),
      yRotate = new Rotate(0, Rotate.Y_AXIS)
    );

    //Graphic is attached to root
    final Group root = new Group(parent);
    //Window of the scene
    final Scene scene = new Scene(root, 900, 900, true);
    //Background
    scene.setFill(Color.SKYBLUE);

    //Listener of mousePressed
    scene.setOnMousePressed
    (
        event ->
        {
            anchorX = event.getSceneX();
            anchorY = event.getSceneY();
            anchorAngleX = angleX.get();
```

```
            anchorAngleY = angleY.get();
        }
    );


    //Listener of mouseDragged
    scene.setOnMouseDragged
    (
        event ->
        {
            angleX.set(anchorAngleX
                    - (anchorY - event.getSceneY()));
            angleY.set(anchorAngleY
                    + anchorX - event.getSceneX());
        }
    );

    //Rotation of the scene gets bound to
    //the angle determined by  mouseDragged
    xRotate.angleProperty().bind(angleX);
    yRotate.angleProperty().bind(angleY);

    //Illumination
    PointLight pointLight = new PointLight(Color.WHITE);
    pointLight.setTranslateX(100);
    pointLight.setTranslateY(200);
    pointLight.setTranslateZ(-2000);
    root.getChildren().add(pointLight);

    //Camera
    scene.setCamera(camera);
    primaryStage.setScene(scene);
    PerspectiveCamera subSceneCamera
            = new PerspectiveCamera(false);
    scene.setCamera(subSceneCamera);

    //To zoom down the graphic,
    //it is translated away from  the beholder,
    //along the Z-direction (normal to the screen).
    //Change zPos to change initial setting.
```

```
zSlider = new Slider();
zSlider.setMax(5000);
zSlider.setMin(1);


//Bind zPos to the reading of the slider
zSlider.valueProperty().bindBidirectional(zPos);
//Execute negative zoom
parent.translateZProperty().bind(zPos);



//This slider   controls the closing of the hand.
closeHandSlider = new Slider();
closeHandSlider.setMax(90);
closeHandSlider.setMin(0);
closeHandSlider.setValue(10);

//To close the hand, an angle of rotation is enabled.
//Bind angleProperty to the reading of the slider
closeHandSlider.valueProperty()
        .bindBidirectional(closeProperty);


//This slider controls the openning of fingers.
openFingersSlider = new Slider();
openFingersSlider.setMax(15);
openFingersSlider.setMin(0);
openFingersSlider.setValue(10);

//To open fingers, an angle of rotation is enabled.
//Bind openFingersProperty to the reading of the slider
openFingersSlider.valueProperty()
        .bindBidirectional(openFingersProperty);



//Slider to rotate  the thumb
rotateThumbSlider = new Slider();
```

```
rotateThumbSlider.setMax(60);
rotateThumbSlider.setMin(0);
rotateThumbSlider.setValue(0);
rotateThumbSlider.adjustValue(rotateThumbSlider.getValue());

//To rotate the thumb, an angle of rotation is enabled.
//Bind closeThumbProperty to the reading of the slider
rotateThumbSlider.valueProperty()
        .bindBidirectional(rotateThumbProperty);



//Other JavaFX controls, labels and containers
GridPane gridPane = new GridPane();
gridPane.setVgap(4);
gridPane.setHgap(10);
gridPane.setPadding(new Insets(5, 5, 5, 5));
//Column 0, Row   0
gridPane.add(new Label("Zoom: "), 0, 0);
gridPane.add(zSlider, 1, 0);
gridPane.add(new Label("Close hand: "), 2, 0);
gridPane.add(closeHandSlider, 3, 0);
gridPane.add(new Label("Open fingers: "), 0, 1);
gridPane.add(openFingersSlider, 1, 1);
gridPane.add(new Label("Rotate thumb: "), 0, 2);
gridPane.add(rotateThumbSlider, 1,2);



 SubScene subScene
        = new SubScene(gridPane, 880, 200);
subScene.setLayoutX(10);
subScene.setLayoutY(800);
root.getChildren().add(subScene);

primaryStage.show();

//The sense of touch is here
parent.setOnMouseClicked((event)->
{
```

```
        //Information that can be retrieved
        PickResult res = event.getPickResult();
        //Which node
        Node node = res.getIntersectedNode();
        //Id of node (null if not declared)
        String nodeId  = event.getPickResult()
                .getIntersectedNode().getId();
        //which face of the node (negative if is not enabled
        int face = event.getPickResult().getIntersectedFace();
        System.out.println("res = " + res);
        System.out.println("node = " + node);
        System.out.println("nodeID = " + nodeId);
        System.out.println("face" + face);
        //Change color to red, rotate around Z_AXIS
        if (node instanceof Box)
            {
                ((Box) node).setMaterial(
                        new PhongMaterial( Color.RED));
                armGroup.rotationAxisProperty()
                        .setValue(Z_AXIS);
                int val = rd.nextInt(91);
                armGroup.rotateProperty().setValue(val);
                finger4Ph2Group.rotateProperty().setValue(val);
            }




    });

}


public void play()
{

}

@Override public void stop()
```

```
{

}

@Override
public void start(Stage primaryStage)  throws Exception
{
    init(primaryStage);
    primaryStage.show();
    play();
}


public static void main(String[] args)
{
    launch(args);
}


//Make the drawing
private Node create3dContent()
{
    //The scene can be rotated
    XformWithPivot sceneRoot = new XformWithPivot();
    sceneRoot.rx.setAngle(45);
    sceneRoot.ry.setAngle(110);
    getColors();
    //The constructor of hand is invoked.
    Hand  h =  new Hand();
    sceneRoot.getChildren().add(h);
    return new Group(sceneRoot);
}

//Colors with a name in JavaFX are put into an array
static private Color[] getColors()
{
    //Colors with a name in JavaFX
    final Field[] fields = Color.class.getFields();
```

```
    int N = 0;
    //Colors are put into an array
    for (final Field field : fields)
    {
        if (field.getType() == Color.class)
        {
            try
            {
                final Color color = (Color) field.get(null);
                //Array to keep colors
                JavaFXColors[N] = color;
                /*
                //The name of the color
                final String colorName = field.getName();
                System.out.println("N = " + N + "   "
                        + color.toString() + " " + colorName);
                */
                N++;
            } catch (IllegalArgumentException
                    | IllegalAccessException ex)
            {
                Logger.getLogger(HandOnlyBoxes9.class
                    .getName()).log(Level.SEVERE, null, ex);
            }
       }
    }
    return JavaFXColors;
}


//A random color is synthesized
 private static Color randomColor()
 {
    //A JavaFX named color is picked
    //but the first, Transparent, is ignored
    int n = rd.nextInt(JavaFXColors.length-1)+1;
    Color c = JavaFXColors[n];
    //Some light colors are not visualized so,
    //they might be darkened:
    //c = c.darker();
```

```
    return c;
 }


  //=================================================
  //=============Inner class XformWithPivot============
  //=================================================



//Kit of available families of transformations
private class XformWithPivot extends Group
{
    //Declaration and initialization
    public Rotate rx = new Rotate();
    { rx.setAxis(Rotate.X_AXIS); }
    public Rotate ry = new Rotate();
    { ry.setAxis(Rotate.Y_AXIS); }
    public Rotate rz = new Rotate();
    { rz.setAxis(Rotate.Z_AXIS); }
    //Rotation = angle (one coordinate)
    //+ pivot (3 coordinates) + axis.
    //This rotation is around X_AXIS
    public Rotate rwpx = new Rotate(0,0,0,0, X_AXIS);
    //This rotation is around Y_AXIS
    public Rotate rwpy = new Rotate(0,0,0,0, Y_AXIS);
    //This rotation is around Z_AXIS
    public Rotate rwpz = new Rotate(0,0,0,0, Z_AXIS);

    public XformWithPivot()
    {
        super();
        getTransforms().addAll( rx, ry, rz, rwpx, rwpy, rwpz );
    }
}//End of class XformWithPivot



  //=========================================
  //=============Inner class Hand============
```

```
  //=======================================


private class Hand extends Group
{


    //Nested definition of   nodes is implemented
    //to have, say,  the palm dragging the fingers,
    //and a phalanx dragging the rest of its the finger.


    //Each finger generates various natural groups:
    //Group with the extreme phalanx of finger0
    //Symbolic initialization
    XformWithPivot finger0Ph2Group = new XformWithPivot();
    //Group with the two extreme phalanges
    XformWithPivot finger0Ph21Group = new XformWithPivot();
    //Group with all three phalanges. It is the finger
    XformWithPivot finger0Ph210Group = new XformWithPivot();
    //Finger1
    XformWithPivot finger1Ph2Group = new XformWithPivot();
    XformWithPivot finger1Ph21Group = new XformWithPivot();
    XformWithPivot finger1Ph210Group = new XformWithPivot();
    //Finger2
    XformWithPivot finger2Ph2Group = new XformWithPivot();
    XformWithPivot finger2Ph21Group = new XformWithPivot();
    XformWithPivot finger2Ph210Group = new XformWithPivot();
    //Finger3
    XformWithPivot finger3Ph2Group = new XformWithPivot();
    XformWithPivot finger3Ph21Group = new XformWithPivot();
    XformWithPivot finger3Ph210Group = new XformWithPivot();
    //Finger4
    XformWithPivot finger4Ph2Group = new XformWithPivot();
    XformWithPivot finger4Ph21Group = new XformWithPivot();
    XformWithPivot finger4Ph210Group = new XformWithPivot();
    //Thumb
    XformWithPivot thumbPh2Group = new XformWithPivot();
    XformWithPivot thumbPh21Group = new XformWithPivot();
```

```
XformWithPivot thumbPh210Group = new XformWithPivot();


//Group with palm and fingers
XformWithPivot palmGroup = new XformWithPivot();


Hand()
{

    //Arm is constructed
    Box arm = new Box(100,20,400);
    Color c = randomColor();
    PhongMaterial materialArm = new PhongMaterial(c);
    arm.setMaterial(materialArm);
    arm.setTranslateY(20);
    //Palm is constructed
    Box palm = new Box(125,20,140);
    Color c2 = randomColor();
    PhongMaterial materialPalm = new PhongMaterial(c2);
    palm.setMaterial(materialPalm);
    palm.setTranslateZ(270);
    palm.setTranslateY(20);



    //Ordinary  fingers

    Finger Fingers[] = new Finger[nFingers];

    //Groups are constructed
    for(int i = 0; i < nFingers; i++)
    {
        //Each finger has nPhalanges phalanges
        Fingers[i] = new Finger(nPhalanges);
        for(int j =0; j < nPhalanges; j++)
        {
            Box ph = Fingers[i].Phalanges[j].box;

            ph.setTranslateX(15.625 * (4-2*(i))-15.625);
```

```
ph.setTranslateZ(360 + 40*j);
ph.setTranslateY(20);

//Natural groups are instantiated
//Finger 0 and its subgroups
if (i == 0)
{
    if (j == 2) finger0Ph2Group.getChildren()
            .add(ph);

    if (j == 1) finger0Ph21Group
            .getChildren()
            .addAll(finger0Ph2Group, ph);
    if (j == 0) finger0Ph210Group
            .getChildren()
            .addAll(finger0Ph21Group, ph);
}

//Finger 1  and its subgroups
if (i == 1)
{
    if (j == 2) finger1Ph2Group
            .getChildren().add(ph);
    if (j == 1) finger1Ph21Group
            .getChildren()
            .addAll(finger1Ph2Group, ph);
    if (j == 0) finger1Ph210Group
            .getChildren()
            .addAll(finger1Ph21Group, ph);
}


//Finger 2  and its subgroups
if (i == 2)
{
    if (j == 2) finger2Ph2Group
            .getChildren().add(ph);
    if (j == 1) finger2Ph21Group
            .getChildren()
            .addAll(finger2Ph2Group, ph);
```

```
                if (j == 0) finger2Ph210Group
                        .getChildren()
                        .addAll(finger2Ph21Group, ph);
            }


            //Finger 3 and its subgroups
            if (i == 3)
            {
                if (j == 2) finger3Ph2Group.getChildren().add(ph);
                if (j == 1) finger3Ph21Group
                        .getChildren()
                        .addAll(finger3Ph2Group, ph);
                if (j == 0) finger3Ph210Group
                        .getChildren()
                        .addAll(finger3Ph21Group, ph);
            }

        }
    }

    //Our thumb  has 3 phalanges
    Finger thumb = new Finger(3);
    for(int j = 0; j < nPhalanges; j++)
        {
            Box ph = thumb.Phalanges[j].box;
            //When the thumb is aligned with the hand
            //it has a rotation around its vertical axis
            ph.setRotationAxis(Z_AXIS);
            ph.setRotate(45);
            //Initial position
            ph.setTranslateX(-78.125);
            ph.setTranslateY(15);
            ph.setTranslateZ(250 + 40*j);
        }
    //Natural groups of control
    thumbPh2Group.getChildren().add(thumb
            .Phalanges[2].box);
    thumbPh21Group.getChildren()
            .addAll(thumbPh2Group
```

```
                , thumb.Phalanges[1].box);
thumbPh210Group.getChildren()
        .addAll(thumbPh21Group
                , thumb.Phalanges[0].box);
//Original inclination of the thumb
//in within the   plane of the palm
thumbPh210Group.getTransforms()
        .add(new Rotate(-55, //Angle
                -78.125, 15, 250, //Pivot
                Y_AXIS)); //Axis

//Fingers are attached to the palm
palmGroup.getChildren().addAll(palm
        , finger0Ph210Group,
        finger1Ph210Group,
        finger2Ph210Group,
        finger3Ph210Group,
        thumbPh210Group);
//Palm and arm are attached to armGroup
armGroup.getChildren().addAll(arm, palmGroup );



//========OPENING/CL0SING OF THE HAND============

NumberBinding closePropertyHalf
        = closeProperty.multiply(0.5);

//The slider that closes the hand extends its
//control to the palm:
//it must rotate around
//the end of the arm that is parallel to the X_AXIS
//This is the pivot of rotation
palmGroup.rwpx.pivotZProperty().setValue(200.0);
palmGroup.rwpx.pivotYProperty().setValue(10.0);
//The angle of rotation is controlled by a slider
palmGroup.rwpx.angleProperty()
        .bind(closePropertyHalf);
```

```
//Specifications of movement for subgroups of finger 0
finger0Ph2Group.rwpx.pivotZProperty().setValue(420.0);
finger0Ph2Group.rwpx.pivotYProperty().setValue(10.0);
finger0Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger0Ph21Group.rwpx.pivotZProperty().setValue(380.0);
finger0Ph21Group.rwpx.pivotYProperty().setValue(10.0);
finger0Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger0Ph210Group.rwpx.pivotZProperty().setValue(340.0);
finger0Ph210Group.rwpx.pivotYProperty().setValue(10.0);
finger0Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

//Specifications of movement for subgroups of finger 1
finger1Ph2Group.rwpx.pivotZProperty().setValue(420.0);
finger1Ph2Group.rwpx.pivotYProperty().setValue(10.0);
finger1Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger1Ph21Group.rwpx.pivotZProperty().setValue(380.0);
finger1Ph21Group.rwpx.pivotYProperty().setValue(10.0);
finger1Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger1Ph210Group.rwpx.pivotZProperty().setValue(340.0);
finger1Ph210Group.rwpx.pivotYProperty().setValue(10.0);
finger1Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

//Specifications of movement for subgroups of finger 2
finger2Ph2Group.rwpx.pivotZProperty().setValue(420.0);
finger2Ph2Group.rwpx.pivotYProperty().setValue(10.0);
finger2Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger2Ph21Group.rwpx.pivotZProperty().setValue(380.0);
```

```
finger2Ph21Group.rwpx.pivotYProperty().setValue(10.0);
finger2Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger2Ph210Group.rwpx.pivotZProperty().setValue(340.0);
finger2Ph210Group.rwpx.pivotYProperty().setValue(10.0);
finger2Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

//Specifications of movement for subgroups of finger 3
finger3Ph2Group.rwpx.pivotZProperty().setValue(420.0);
finger3Ph2Group.rwpx.pivotYProperty().setValue(10.0);
finger3Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger3Ph21Group.rwpx.pivotZProperty().setValue(380.0);
finger3Ph21Group.rwpx.pivotYProperty().setValue(10.0);
finger3Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger3Ph210Group.rwpx.pivotZProperty().setValue(340.0);
finger3Ph210Group.rwpx.pivotYProperty().setValue(10.0);
finger3Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);



//======OPENING/CLOSING OF FINGERS========

//Fingers that are at the border must
//open twice faster as those
//near the center.
NumberBinding openFingersPropertyTwo
        = openFingersProperty.multiply(2);
//Specifications of movement for subgroups of finger 0
finger0Ph210Group.rwpy.pivotXProperty().setValue(0.0);
finger0Ph210Group.rwpy.pivotYProperty().setValue(10.0);
finger0Ph210Group.rwpy.pivotZProperty().setValue(340.0);
finger0Ph210Group.rwpy.angleProperty()
        .bind(openFingersPropertyTwo);
```

```
//Specifications of movement for subgroups of finger 1
finger1Ph210Group.rwpy.pivotXProperty().setValue(0.0);
finger1Ph210Group.rwpy.pivotYProperty().setValue(30.0);
finger1Ph210Group.rwpy.pivotZProperty().setValue(340.0);

finger1Ph210Group.rwpy.angleProperty()
        .bind(openFingersProperty);

//Finger two remains invariant under opening of fingers

//Opening in the contrary direction of fingers 3 and 4
NumberBinding openFingersPropertyMinus
        = openFingersProperty.multiply(-1);

//Specifications of movement for finger 3
finger3Ph210Group.rwpy.pivotXProperty().setValue(0.0);
finger3Ph210Group.rwpy.pivotYProperty().setValue(30.0);
finger3Ph210Group.rwpy.pivotZProperty().setValue(340.0);
finger3Ph210Group.rwpy.angleProperty()
        .bind(openFingersPropertyMinus);

NumberBinding openFingersPropertyMinusTwo
        = openFingersProperty.multiply(-2);

//Specifications of movement for finger 4
finger4Ph210Group.rwpy.pivotXProperty().setValue(0.0);
finger4Ph210Group.rwpy.pivotYProperty().setValue(10.0);
finger4Ph210Group.rwpy.pivotZProperty().setValue(340.0);
finger4Ph210Group.rwpy.angleProperty()
        .bind(openFingersPropertyMinusTwo);


//====== The thumb ========

//Specifications of movement for the thumb
NumberBinding rotateThumbPropertyDown
        = rotateThumbProperty.multiply(0.2);
```

```
NumberBinding rotateThumbPropertyUp
        = rotateThumbProperty.multiply(2);

//Rotations of phalanges 2 and 1 to make the grip.
//Rotations around X_AXIS and Y_AXIS at the same time
//Phalanx 2
thumbPh2Group.rwpx.pivotXProperty().setValue(-70.125);
thumbPh2Group.rwpx.pivotYProperty().setValue(15);
thumbPh2Group.rwpx.pivotZProperty().setValue(340.0);
thumbPh2Group.rwpx.angleProperty()
        .bind(rotateThumbPropertyDown);
//Phalanges 2 and 1
thumbPh21Group.rwpx.pivotXProperty().setValue(-70.125);
thumbPh21Group.rwpx.pivotYProperty().setValue(15);
thumbPh21Group.rwpx.pivotZProperty().setValue(290.0);
thumbPh21Group.rwpx.angleProperty()
        .bind(rotateThumbPropertyDown);



//Phalanx 2
thumbPh2Group.rwpy.pivotXProperty().setValue(-70.125);
thumbPh2Group.rwpy.pivotYProperty().setValue(15);
thumbPh2Group.rwpy.pivotZProperty().setValue(340.0);
thumbPh2Group.rwpy.angleProperty()
        .bind(rotateThumbPropertyDown);
//Phalanges 2 and 1
thumbPh21Group.rwpy.pivotXProperty().setValue(-70.125);
thumbPh21Group.rwpy.pivotYProperty().setValue(15);
thumbPh21Group.rwpy.pivotZProperty().setValue(290.0);
thumbPh21Group.rwpy.angleProperty()
        .bind(rotateThumbPropertyDown);


//Rotation of the whole thumb
//to get opposite to other fingers
thumbPh210Group.rwpz.pivotXProperty().setValue(-70.125);
thumbPh210Group.rwpz.pivotYProperty().setValue(15);
thumbPh210Group.rwpz.pivotZProperty().setValue(250.0);
thumbPh210Group.rwpz.angleProperty()
```

```
                .bind(rotateThumbPropertyUp);



        getChildren().addAll(armGroup );

    }//End of constructor of class Hand


}//End of inner class Hand



//A finger is an ensemble of phalanges
private class Finger extends Group
{

    Phalanx[] Phalanges    = new Phalanx [nPhalanges];

    //Constructor
    Finger(int nPhalanges)
    {
        for(int i =0; i < nPhalanges; i++)
        {
            Phalanx ph = new Phalanx(31.25,20,40);
            Phalanges[i] = ph;
        }
    }
}//End of inner class Finger



//A phalanx is a JavaFX 3D box
private class Phalanx extends Group
{
    Box box;

    //Constructor
```

```
    Phalanx(double width, int length, int height)
    {
        box = new Box(width, length, height);
        Color c = randomColor();
        PhongMaterial materialFinger
                = new PhongMaterial(c);
        box.setMaterial(materialFinger);
    }
}//End of inner class phalanx

}//En of main class M24 HandOnlyBoxes9
```

25 ***Exercise.*** *Run the previous program and play with the code.*

26 ***Exercise.*** *The hand of the previous program has an opposable thumb that thanks to a rotation can get opposite to other finger to form a tight grip. Add the possibility to execute traversal movements of the thumb (traversal to the direction of rotation). This kind of movement enables fine control of the thumb, say, to write and draw.* Answer

2.7 An animation

We have simulated a hand that is controlled at will by the brain. The sliders represent the decisions of the brain to control the movements of fingers. Let us show now how can we produce an animation, say, that shows a hand that opens and closes its fingers to form a grip.

27 ***An animated hand is implemented by the next code.***

```
/*
 * Program M27 HandOnlyBoxess11.
 * It is shown how to make an ensemble of boxes
 * into a 3D closable hand
 * whose fingers can be moved transversally.
 * The hand has an opposable thumb, i.e.,
 * its thumb is aligned with the other fingers
 * but after a complex movement opposites them
```

```
 * to form a strong grip upon closing.
 * The thumb can also get transversal movements.
 * The hand can react to touching with the mouse:
 * touched phalanx changes its color to red.

 * An animated show is included.

 * We try to copy the anatomical structure:
 * phalanges form fingers that together
 * with the palm conform the hand.
 * The thumb is initially rotated
 * around the Z_AXIS and around the X_AXIS,
 * just as our hands.

 * It is found that JavaFX is highly compatible
 * with the anatomical organization.
 */

package handonlyboxess11;



import java.lang.reflect.Field;
import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.animation.KeyFrame;
import javafx.animation.KeyValue;
import javafx.animation.SequentialTransition;
import javafx.animation.Timeline;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.beans.binding.NumberBinding;
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;
import javafx.geometry.Insets;

import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.PerspectiveCamera;
```

```
import javafx.scene.PointLight;
import javafx.scene.Scene;
import javafx.scene.SubScene;
import javafx.scene.control.Label;
import javafx.scene.control.Slider;
import javafx.scene.input.PickResult;
import javafx.scene.layout.GridPane;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.Box;
import javafx.scene.transform.Rotate;
import static javafx.scene.transform.Rotate.X_AXIS;
import static javafx.scene.transform.Rotate.Y_AXIS;
import static javafx.scene.transform.Rotate.Z_AXIS;
import javafx.stage.Stage;
import javafx.util.Duration;



public class HandOnlyBoxess11 extends Application
{
    //Node with all elements
    XformWithPivot armGroup = new XformWithPivot();
    XformWithPivot finger4Ph2Group = new XformWithPivot();
    //Slider for zooming
    private Slider zSlider;
    //Position of graphic along the z-axis,
    //normal to the screen.
    //It is a double property because
    //it will get involved into a binding
    private final DoubleProperty zPos
            = new SimpleDoubleProperty(2000);

    //Slider for angle of rotation of fingers
    private Slider closeHandSlider;
    //To close the hand,
    //each phalange suffers a rotation.
    //It is a double property because
    //it will get involved into a binding
    private final DoubleProperty closeProperty
```

```
        = new SimpleDoubleProperty(0);

    //Slider for opening fingers thanks to
    //their transversal rotation
    private Slider openFingersSlider;
    //To open the fingers,
    //each phalange suffers a transversal rotation.
    //It is a double property because
    //it will get involved into a binding
    private final DoubleProperty openFingersProperty
        = new SimpleDoubleProperty(0);

    //Slider for opening/closing the thumb
    private Slider closeThumbSlider;
    //To close the thumb we need a rotation.
    //It is a double property because
    //it will get involved into a binding
    private final DoubleProperty closeThumbProperty
        = new SimpleDoubleProperty(0);

    //Slider for rotating the thumb to form the grip
    private Slider rotateThumbSlider;
    //To form the grip we need a rotation of the thumb.
    //It is a double property because
    //it will get involved into a binding
    private final DoubleProperty rotateThumbProperty
        = new SimpleDoubleProperty(0);

    //For listener of mouse
    private double anchorAngleX = 0;
    private double anchorAngleY = 0;
    private double anchorX, anchorY;
    //Angles of rotation of the scene
    private final DoubleProperty angleX
        = new SimpleDoubleProperty(0);
    private final DoubleProperty angleY
        = new SimpleDoubleProperty(0);

    PerspectiveCamera camera
        = new PerspectiveCamera(false);
```

```
    //The application is instantiated
    static HandOnlyBoxess11 p = new HandOnlyBoxess11();



    //An array with all colors that are named in JavaFX
    private static final Color JavaFXColors[]
                                = new Color[148];

    //Generator of pseudo random numbers
    private static final Random rd = new Random();

    //Number of fingers
    private final int nFingers = 4;

    //Number of phalanges of each finger
    private final int nPhalanges = 3;

    //Overall JavaFX animation
    private static  SequentialTransition allST
                    = new SequentialTransition();
    private final Timeline animation  = new Timeline();

public void init(Stage primaryStage)
{
    primaryStage.setTitle("An ensemble of boxes"
            + " act coherently as a   hand");
    //The whole drawing  is created
    //as a JavaFX space-like object
    final Group parent = new Group(create3dContent());

    //Initial position
    parent.setTranslateX(450);
    parent.setTranslateY(400);
    parent.setTranslateZ(0);
    Rotate xRotate;
    Rotate yRotate;
    //Initialization of rotation
    parent.getTransforms().setAll
```

```
(
  xRotate = new Rotate(0, Rotate.X_AXIS),
  yRotate = new Rotate(0, Rotate.Y_AXIS)
);

//Graphic is attached to root
final Group root = new Group(parent);
//Window of the scene
final Scene scene = new Scene(root, 900, 900, true);
//Background
scene.setFill(Color.SKYBLUE);

//Listener of mousePressed
scene.setOnMousePressed
(
    event ->
    {
        anchorX = event.getSceneX();
        anchorY = event.getSceneY();
        anchorAngleX = angleX.get();
        anchorAngleY = angleY.get();
    }
);


//Listener of mouseDragged
scene.setOnMouseDragged
(
    event ->
    {
        angleX.set(anchorAngleX
                - (anchorY - event.getSceneY()));
        angleY.set(anchorAngleY
                + anchorX - event.getSceneX());
    }
);

//Rotation of the scene gets bound to
//the angle determined by  mouseDragged
xRotate.angleProperty().bind(angleX);
```

```
yRotate.angleProperty().bind(angleY);

//Illumination
PointLight pointLight = new PointLight(Color.WHITE);
pointLight.setTranslateX(100);
pointLight.setTranslateY(200);
pointLight.setTranslateZ(-2000);
root.getChildren().add(pointLight);

//Camera
scene.setCamera(camera);
primaryStage.setScene(scene);
PerspectiveCamera subSceneCamera
        = new PerspectiveCamera(false);
scene.setCamera(subSceneCamera);

//To zoom down the graphic,
//it is translated away from  the beholder,
//along the Z-direction (normal to the screen).
//Change zPos to change initial setting.
zSlider = new Slider();
zSlider.setMax(5000);
zSlider.setMin(1);


//Bind zPos to the reading of the slider
zSlider.valueProperty().bindBidirectional(zPos);
//Execute negative zoom
parent.translateZProperty().bind(zPos);



//This slider   controls the closing of the hand.
closeHandSlider = new Slider();
closeHandSlider.setMax(90);
closeHandSlider.setMin(0);
closeHandSlider.setValue(10);

//To close the hand, an angle of rotation is enabled.
//Bind angleProperty to the reading of the slider
```

```
closeHandSlider.valueProperty()
        .bindBidirectional(closeProperty);


//This slider controls the openning of fingers.
openFingersSlider = new Slider();
openFingersSlider.setMax(15);
openFingersSlider.setMin(0);
openFingersSlider.setValue(10);

//To open fingers, an angle of rotation is enabled.
//Bind openFingersProperty to the reading of the slider
openFingersSlider.valueProperty()
        .bindBidirectional(openFingersProperty);


//Slider to open  the thumb
closeThumbSlider = new Slider();
closeThumbSlider.setMax(55);
closeThumbSlider.setMin(0);
closeThumbSlider.setValue(0);
closeThumbSlider.adjustValue(closeThumbSlider.getValue());

//To open the thumb, an angle of rotation is enabled.
//Bind closeThumbProperty to the reading of the slider
closeThumbSlider.valueProperty().bindBidirectional(closeThumbProperty);


//Slider to rotate  the thumb
rotateThumbSlider = new Slider();
rotateThumbSlider.setMax(50);
rotateThumbSlider.setMin(0);
rotateThumbSlider.setValue(0);
rotateThumbSlider.adjustValue(rotateThumbSlider.getValue());

//To rotate the thumb, an angle of rotation is enabled.
//Bind closeThumbProperty to the reading of the slider
rotateThumbSlider.valueProperty()
        .bindBidirectional(rotateThumbProperty);
```

```
//Other JavaFX controls, labels and containers
GridPane gridPane = new GridPane();
gridPane.setVgap(4);
gridPane.setHgap(10);
gridPane.setPadding(new Insets(5, 5, 5, 5));
//Column 0, Row   0
gridPane.add(new Label("Zoom: "), 0, 0);
gridPane.add(zSlider, 1, 0);
gridPane.add(new Label("Close hand: "), 2, 0);
gridPane.add(closeHandSlider, 3, 0);
gridPane.add(new Label("Open fingers: "), 0, 1);
gridPane.add(openFingersSlider, 1, 1);
gridPane.add(new Label("Close thumb: "), 0, 2);
gridPane.add(closeThumbSlider, 1, 2);
gridPane.add(new Label("Rotate thumb: "), 2, 2);
gridPane.add(rotateThumbSlider, 3,2);



SubScene subScene
        = new SubScene(gridPane, 880, 200);
subScene.setLayoutX(10);
subScene.setLayoutY(800);
root.getChildren().add(subScene);

primaryStage.show();

//The sense of touch is here
parent.setOnMouseClicked((event)->
{
    //Information that can be retrieved
    PickResult res = event.getPickResult();
    //Which node
    Node node = res.getIntersectedNode();
    //Id of node (null if not declared)
    String nodeId  = event.getPickResult()
            .getIntersectedNode().getId();
    //which face of the node (negative if is not enabled
```

```
        int face = event.getPickResult().getIntersectedFace();
        System.out.println("res = " + res);
        System.out.println("node = " + node);
        System.out.println("nodeID = " + nodeId);
        System.out.println("face" + face);
        //Change color to red, rotate around Z_AXIS
        if (node instanceof Box)
            {
                ((Box) node).setMaterial(
                        new PhongMaterial( Color.RED));
                armGroup.rotationAxisProperty()
                        .setValue(Z_AXIS);
                int val = rd.nextInt(91);
                armGroup.rotateProperty().setValue(val);
                finger4Ph2Group.rotateProperty().setValue(val);
            }




    });

    SequentialTransition a = new SequentialTransition();

}


public void play()
{

}

@Override public void stop()
{

}

@Override
public void start(Stage primaryStage)  throws Exception
```

```
{
    init(primaryStage);
    primaryStage.show();

    play();
}


public static void main(String[] args)
{
    launch(args);
}


//Make the drawing
private Node create3dContent()
{
    //The scene can be rotated
    XformWithPivot sceneRoot = new XformWithPivot();
    sceneRoot.rx.setAngle(45);
    sceneRoot.ry.setAngle(110);
    getColors();
    //The constructor of hand is invoked.
    Hand  h =  new Hand();
    sceneRoot.getChildren().add(h);
    return new Group(sceneRoot);
}

//Colors with a name in JavaFX are put into an array
static private Color[] getColors()
{
    //Colors with a name in JavaFX
    final Field[] fields = Color.class.getFields();


    int N = 0;
    //Colors are put into an array
    for (final Field field : fields)
    {
        if (field.getType() == Color.class)
```

```
        {
            try
            {
                final Color color = (Color) field.get(null);
                //Array to keep colors
                JavaFXColors[N] = color;
                /*
                //The name of the color
                final String colorName = field.getName();
                System.out.println("N = " + N + "  "
                        + color.toString() + " " + colorName);
                */
                N++;
            } catch (IllegalArgumentException
                    | IllegalAccessException ex)
            {
                Logger.getLogger(HandOnlyBoxess11.class
                    .getName()).log(Level.SEVERE, null, ex);
            }
       }
    }
    return JavaFXColors;
}


//A random color is synthesized
 private static Color randomColor()
 {
    //A JavaFX named color is picked
    //but the first, Transparent, is ignored
    int n = rd.nextInt(JavaFXColors.length-1)+1;
    Color c = JavaFXColors[n];
    //Some light colors are not visualized so,
    //they might be darkened:
    //c = c.darker();
    return c;
 }


  //================================================
```

```
  //=============Inner class XformWithPivot============
  //===================================================


//Kit of available families of transformations
private class XformWithPivot extends Group
{
    //Declaration and initialization
    public Rotate rx = new Rotate();
    { rx.setAxis(Rotate.X_AXIS); }
    public Rotate ry = new Rotate();
    { ry.setAxis(Rotate.Y_AXIS); }
    public Rotate rz = new Rotate();
    { rz.setAxis(Rotate.Z_AXIS); }
    //Rotation = angle (one coordinate)
    //+ pivot (3 coordinates) + axis.
    //This rotation is around X_AXIS
    public Rotate rwpx = new Rotate(0,0,0,0, X_AXIS);
    //This rotation is around Y_AXIS
    public Rotate rwpy = new Rotate(0,0,0,0, Y_AXIS);
    //This rotation is around Z_AXIS
    public Rotate rwpz = new Rotate(0,0,0,0, Z_AXIS);

    public XformWithPivot()
    {
        super();
        getTransforms().addAll( rx, ry, rz, rwpx, rwpy, rwpz );
    }
}//End of class XformWithPivot



  //=======================================
  //=============Inner class Hand==========
  //=======================================


private class Hand extends Group
```

```
{


    //Nested definition of   nodes is implemented
    //to have, say,  the palm dragging the fingers,
    //and a phalanx dragging the rest of its the finger.


    //Each finger generates various natural groups:
    //Group with the extreme phalanx of finger0
    //Symbolic initialization
    XformWithPivot finger0Ph2Group = new XformWithPivot();
    //Group with the two extreme phalanges
    XformWithPivot finger0Ph21Group = new XformWithPivot();
    //Group with all three phalanges. It is the finger
    XformWithPivot finger0Ph210Group = new XformWithPivot();
    //Finger1
    XformWithPivot finger1Ph2Group = new XformWithPivot();
    XformWithPivot finger1Ph21Group = new XformWithPivot();
    XformWithPivot finger1Ph210Group = new XformWithPivot();
    //Finger2
    XformWithPivot finger2Ph2Group = new XformWithPivot();
    XformWithPivot finger2Ph21Group = new XformWithPivot();
    XformWithPivot finger2Ph210Group = new XformWithPivot();
    //Finger3
    XformWithPivot finger3Ph2Group = new XformWithPivot();
    XformWithPivot finger3Ph21Group = new XformWithPivot();
    XformWithPivot finger3Ph210Group = new XformWithPivot();
    //Finger4
    XformWithPivot finger4Ph2Group = new XformWithPivot();
    XformWithPivot finger4Ph21Group = new XformWithPivot();
    XformWithPivot finger4Ph210Group = new XformWithPivot();
    //Thumb
    XformWithPivot thumbPh2Group = new XformWithPivot();
    XformWithPivot thumbPh21Group = new XformWithPivot();
    XformWithPivot thumbPh210Group = new XformWithPivot();


    //Group with palm and fingers
```

```
    XformWithPivot palmGroup = new XformWithPivot();


Hand()
{

    //Arm is constructed
    Box arm = new Box(100,20,400);
    Color c = randomColor();
    PhongMaterial materialArm = new PhongMaterial(c);
    arm.setMaterial(materialArm);
    arm.setTranslateY(20);
    //Palm is constructed
    Box palm = new Box(125,20,140);
    Color c2 = randomColor();
    PhongMaterial materialPalm = new PhongMaterial(c2);
    palm.setMaterial(materialPalm);
    palm.setTranslateZ(270);
    palm.setTranslateY(20);



    //4 fingers on the same line.

    Finger Fingers[] = new Finger[nFingers];

    //Groups are constructed
    for(int i = 0; i < nFingers; i++)
    {
        //Each finger has nPhalanges phalanges
        Fingers[i] = new Finger(nPhalanges);
        for(int j =0; j < nPhalanges; j++)
        {
            Box ph = Fingers[i].Phalanges[j].box;

            ph.setTranslateX(15.625 * (4-2*(i))-15.625);
            ph.setTranslateZ(360 + 40*j);
            ph.setTranslateY(20);

            //Natural groups are instantiated
```

```
//Finger 0 and its subgroups
if (i == 0)
{
    if (j == 2) finger0Ph2Group.getChildren()
            .add(ph);

    if (j == 1) finger0Ph21Group
            .getChildren()
            .addAll(finger0Ph2Group, ph);
    if (j == 0) finger0Ph210Group
            .getChildren()
            .addAll(finger0Ph21Group, ph);
}

//Finger 1  and its subgroups
if (i == 1)
{
    if (j == 2) finger1Ph2Group
            .getChildren().add(ph);
    if (j == 1) finger1Ph21Group
            .getChildren()
            .addAll(finger1Ph2Group, ph);
    if (j == 0) finger1Ph210Group
            .getChildren()
            .addAll(finger1Ph21Group, ph);
}


//Finger 2  and its subgroups
if (i == 2)
{
    if (j == 2) finger2Ph2Group
            .getChildren().add(ph);
    if (j == 1) finger2Ph21Group
            .getChildren()
            .addAll(finger2Ph2Group, ph);
    if (j == 0) finger2Ph210Group
            .getChildren()
            .addAll(finger2Ph21Group, ph);
}
```

```
            //Finger 3 and its subgroups
            if (i == 3)
            {
                if (j == 2) finger3Ph2Group.getChildren().add(ph);
                if (j == 1) finger3Ph21Group
                        .getChildren()
                        .addAll(finger3Ph2Group, ph);
                if (j == 0) finger3Ph210Group
                        .getChildren()
                        .addAll(finger3Ph21Group, ph);
            }

        }
    }

    //Our thumb  has 3 phalanges
    Finger thumb = new Finger(3);
    for(int j = 0; j < nPhalanges; j++)
        {
            Box ph = thumb.Phalanges[j].box;
            //When the thumb is aligned with the hand
            //it has a rotation around its vertical axis
            ph.setRotationAxis(Z_AXIS);
            ph.setRotate(45);
            //Initial position
            ph.setTranslateX(-78.125);
            ph.setTranslateY(15);
            ph.setTranslateZ(250 + 40*j);



        }
    //Natural groups of control
    thumbPh2Group.getChildren().add(thumb
            .Phalanges[2].box);
    thumbPh21Group.getChildren()
            .addAll(thumbPh2Group
                    , thumb.Phalanges[1].box);
```

```
thumbPh210Group.getChildren()
        .addAll(thumbPh21Group
                , thumb.Phalanges[0].box);
//Original inclination of the thumb
//in within the   plane of the palm
thumbPh210Group.getTransforms()
        .add(new Rotate(-55, //Angle
                -78.125, 15, 250, //Pivot
                Y_AXIS)); //Axis

//Fingers are attached to the palm
palmGroup.getChildren().addAll(palm
        , finger0Ph210Group,
        finger1Ph210Group,
        finger2Ph210Group,
        finger3Ph210Group,
        thumbPh210Group);
//Palm and arm are attached to armGroup
armGroup.getChildren().addAll(arm, palmGroup );



//========OPENING/CL0SING OF THE HAND============

NumberBinding closePropertyHalf
        = closeProperty.multiply(0.5);

//The slider that closes the hand extends its
//control to the palm:
//it must rotate around
//the end of the arm that is parallel to the X_AXIS
//This is the pivot of rotation
palmGroup.rwpx.pivotZProperty().setValue(200.0);
palmGroup.rwpx.pivotYProperty().setValue(10.0);
//The angle of rotation is controlled by a slider
palmGroup.rwpx.angleProperty()
        .bind(closePropertyHalf);
```

```
//Specifications of movement for subgroups of finger 0
finger0Ph2Group.rwpx.pivotZProperty().setValue(420.0);
finger0Ph2Group.rwpx.pivotYProperty().setValue(10.0);
finger0Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger0Ph21Group.rwpx.pivotZProperty().setValue(380.0);
finger0Ph21Group.rwpx.pivotYProperty().setValue(10.0);
finger0Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger0Ph210Group.rwpx.pivotZProperty().setValue(340.0);
finger0Ph210Group.rwpx.pivotYProperty().setValue(10.0);
finger0Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

//Specifications of movement for subgroups of finger 1
finger1Ph2Group.rwpx.pivotZProperty().setValue(420.0);
finger1Ph2Group.rwpx.pivotYProperty().setValue(10.0);
finger1Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger1Ph21Group.rwpx.pivotZProperty().setValue(380.0);
finger1Ph21Group.rwpx.pivotYProperty().setValue(10.0);
finger1Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger1Ph210Group.rwpx.pivotZProperty().setValue(340.0);
finger1Ph210Group.rwpx.pivotYProperty().setValue(10.0);
finger1Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

//Specifications of movement for subgroups of finger 2
finger2Ph2Group.rwpx.pivotZProperty().setValue(420.0);
finger2Ph2Group.rwpx.pivotYProperty().setValue(10.0);
finger2Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger2Ph21Group.rwpx.pivotZProperty().setValue(380.0);
finger2Ph21Group.rwpx.pivotYProperty().setValue(10.0);
```

```
finger2Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger2Ph210Group.rwpx.pivotZProperty().setValue(340.0);
finger2Ph210Group.rwpx.pivotYProperty().setValue(10.0);
finger2Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

//Specifications of movement for subgroups of finger 3
finger3Ph2Group.rwpx.pivotZProperty().setValue(420.0);
finger3Ph2Group.rwpx.pivotYProperty().setValue(10.0);
finger3Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger3Ph21Group.rwpx.pivotZProperty().setValue(380.0);
finger3Ph21Group.rwpx.pivotYProperty().setValue(10.0);
finger3Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger3Ph210Group.rwpx.pivotZProperty().setValue(340.0);
finger3Ph210Group.rwpx.pivotYProperty().setValue(10.0);
finger3Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);



//======OPENING/CLOSING OF FINGERS========

//Fingers that are at the border must
//open twice faster as those
//near the center.
NumberBinding openFingersPropertyTwo
        = openFingersProperty.multiply(2);
//Specifications of movement for subgroups of finger 0
finger0Ph210Group.rwpy.pivotXProperty().setValue(0.0);
finger0Ph210Group.rwpy.pivotYProperty().setValue(10.0);
finger0Ph210Group.rwpy.pivotZProperty().setValue(340.0);
finger0Ph210Group.rwpy.angleProperty()
        .bind(openFingersPropertyTwo);
```

```
//Specifications of movement for subgroups of finger 1
finger1Ph210Group.rwpy.pivotXProperty().setValue(0.0);
finger1Ph210Group.rwpy.pivotYProperty().setValue(30.0);
finger1Ph210Group.rwpy.pivotZProperty().setValue(340.0);

finger1Ph210Group.rwpy.angleProperty()
        .bind(openFingersProperty);

//Finger two remains invariant under opening of fingers

//Opening in the contrary direction of fingers 3 and 4
NumberBinding openFingersPropertyMinus
        = openFingersProperty.multiply(-1);

//Specifications of movement for finger 3
finger3Ph210Group.rwpy.pivotXProperty().setValue(0.0);
finger3Ph210Group.rwpy.pivotYProperty().setValue(30.0);
finger3Ph210Group.rwpy.pivotZProperty().setValue(340.0);
finger3Ph210Group.rwpy.angleProperty()
        .bind(openFingersPropertyMinus);

NumberBinding openFingersPropertyMinusTwo
        = openFingersProperty.multiply(-2);

//Specifications of movement for finger 4
finger4Ph210Group.rwpy.pivotXProperty().setValue(0.0);
finger4Ph210Group.rwpy.pivotYProperty().setValue(10.0);
finger4Ph210Group.rwpy.pivotZProperty().setValue(340.0);
finger4Ph210Group.rwpy.angleProperty()
        .bind(openFingersPropertyMinusTwo);

//====== The thumb ========
//Specifications of movement for the thumb


NumberBinding rotateThumbPropertyDown
        = rotateThumbProperty.multiply(0.2);

NumberBinding rotateThumbPropertyUp
```

```
        = rotateThumbProperty.multiply(2);


//Tranversal movement
thumbPh210Group.rwpy.pivotXProperty().setValue(-78.125);
thumbPh210Group.rwpy.pivotZProperty().setValue(240.0);
thumbPh210Group.rwpy.angleProperty().bind(closeThumbProperty);


//Rotations of phalanges 2 and 1 to make the grip.
//Rotations around X_AXIS and Y_AXIS at the same time
//Phalanx 2
thumbPh2Group.rwpx.pivotXProperty().setValue(-70.125);
thumbPh2Group.rwpx.pivotYProperty().setValue(15);
thumbPh2Group.rwpx.pivotZProperty().setValue(310.0);
thumbPh2Group.rwpx.angleProperty()
        .bind(rotateThumbPropertyDown);
//Phalanges 2 and 1
thumbPh21Group.rwpx.pivotXProperty().setValue(-70.125);
thumbPh21Group.rwpx.pivotYProperty().setValue(15);
thumbPh21Group.rwpx.pivotZProperty().setValue(290.0);
thumbPh21Group.rwpx.angleProperty()
        .bind(rotateThumbPropertyDown);

//Phalanx 2
thumbPh2Group.rwpy.pivotXProperty().setValue(-70.125);
thumbPh2Group.rwpy.pivotYProperty().setValue(15);
thumbPh2Group.rwpy.pivotZProperty().setValue(310.0);
thumbPh2Group.rwpy.angleProperty()
        .bind(rotateThumbPropertyDown);
//Phalanges 2 and 1
thumbPh21Group.rwpy.pivotXProperty().setValue(-70.125);
thumbPh21Group.rwpy.pivotYProperty().setValue(15);
thumbPh21Group.rwpy.pivotZProperty().setValue(290.0);
thumbPh21Group.rwpy.angleProperty()
        .bind(rotateThumbPropertyDown);

//Rotation of the whole thumb
//to get opposite to other fingers
thumbPh210Group.rwpz.pivotXProperty().setValue(-70.125);
```

```
thumbPh210Group.rwpz.pivotYProperty().setValue(15);
thumbPh210Group.rwpz.pivotZProperty().setValue(250.0);
thumbPh210Group.rwpz.angleProperty()
        .bind(rotateThumbPropertyUp);

getChildren().addAll(armGroup );

//===========Animation=========

double endTime1 = 4000.0; //1000 = 1 second
double endTime2 = 8000.0;

animation.getKeyFrames().addAll
(
    //We move   finger 0 and all fingers obey because
    //the angleProperty bounds them in an ensemble.

    //Initial setting
    new KeyFrame(Duration.ZERO,
        //Permanent condition
        new KeyValue(finger0Ph2Group.rwpx
                .pivotZProperty(), 42.0),
        //Permanent condition
        new KeyValue(finger0Ph2Group.rwpx
            .pivotYProperty(), 10.0),
        //X Rotate: initial position at initial time
        new KeyValue(finger0Ph2Group.rwpx
            .angleProperty(), 0)),

    // X Rotate: final position at final time
    new KeyFrame(new Duration(endTime1),
        new KeyValue(finger0Ph2Group.rwpx
                .angleProperty(), 90)),

    //The thumb closes the grip
    //rotateThumbProperty: initial position at initial time
    new KeyFrame(new Duration(endTime1),
        //Permanent condition
        new KeyValue(rotateThumbProperty, 0)),
    //rotateThumbProperty: final position at final time
```

```
        new KeyFrame(new Duration(endTime2),
            new KeyValue(rotateThumbProperty, 50))

    );

animation.play();

}//End of constructor of class Hand


}//End of inner class Hand



//A finger is an ensemble of phalanges
private class Finger extends Group
{

    Phalanx[] Phalanges    = new Phalanx [nPhalanges];

    //Constructor
    Finger(int nPhalanges)
    {
        for(int i =0; i < nPhalanges; i++)
        {
            Phalanx ph = new Phalanx(31.25,20,40);
            Phalanges[i] = ph;
        }
    }
}//End of inner class Finger



//A phalanx is a JavaFX 3D box
private class Phalanx extends Group
{
    Box box;

    //Constructor
```

```
    Phalanx(double width, int length, int height)
    {
        box = new Box(width, length, height);
        Color c = randomColor();
        PhongMaterial materialFinger
                = new PhongMaterial(c);
        box.setMaterial(materialFinger);
    }
}//End of inner class phalanx

}//En of main class M27 HandOnlyBoxess11
```

28 ***Exercise.*** *Run the previous program and play with the code.*

29 ***Exercise.*** *Complete at pleasure the previous animation.* Answer

2.8 Just for fun

To become an expert you must enjoy keeping playful. This is the spirit of the next program

30 ***The next code shows a humanoid that can walk.***

```
/*
 * Program M30 Humanoid.
 * A JavaFX humanoid is presented.
 * It is controllable from a GUI.
 */

package humanoid;



import java.lang.reflect.Field;
import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.animation.SequentialTransition;
```

```
import javafx.animation.Timeline;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.beans.binding.NumberBinding;
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;
import javafx.geometry.Insets;

import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.PerspectiveCamera;
import javafx.scene.PointLight;
import javafx.scene.Scene;
import javafx.scene.SubScene;
import javafx.scene.control.Label;
import javafx.scene.control.Slider;
import javafx.scene.input.PickResult;
import javafx.scene.layout.GridPane;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.Box;
import javafx.scene.shape.Cylinder;
import javafx.scene.shape.Sphere;
import javafx.scene.transform.Rotate;
import static javafx.scene.transform.Rotate.X_AXIS;
import static javafx.scene.transform.Rotate.Y_AXIS;
import static javafx.scene.transform.Rotate.Z_AXIS;
import javafx.stage.Stage;



public class Humanoid extends Application
{
    //Node with all elements
    XformWithPivot humanoidGroup = new XformWithPivot();

    //Slider for zooming
    private Slider zSlider;
    //Position of graphic along the z-axis,
```

```
//normal to the screen.
//It is a double property because
//it will get involved into a binding
private final DoubleProperty zProperty
        = new SimpleDoubleProperty(2000);

//Slider for angle of rotation of fingers
private Slider walkSlider;
//Walking is controlled by just one parameter
//(that finally is a multiple of transcurred time)
private final DoubleProperty walkProperty
        = new SimpleDoubleProperty(0);


//For listener of mouse
private double anchorAngleX = 0;
private double anchorAngleY = 0;
private double anchorX, anchorY;
//Angles of rotation of the scene
private final DoubleProperty angleX
        = new SimpleDoubleProperty(0);
private final DoubleProperty angleY
        = new SimpleDoubleProperty(0);

PerspectiveCamera camera
        = new PerspectiveCamera(false);

//The application is instantiated
static Humanoid p = new Humanoid();



//An array with all colors that are named in JavaFX
private static final Color JavaFXColors[]
                            = new Color[148];

//Generator of pseudo random numbers
private static final Random rd = new Random();
```

```
    //Overall JavaFX animation
    private static  SequentialTransition allST
                    = new SequentialTransition();
    private final Timeline animation  = new Timeline();

public void init(Stage primaryStage)
{
    primaryStage.setTitle("An ensemble of boxes"
            + " act coherently as a   hand");
    //The whole drawing  is created
    //as a JavaFX space-like object
    final Group parent = new Group(create3dContent());

    //Initial position
    parent.setTranslateX(450);
    parent.setTranslateY(400);
    parent.setTranslateZ(0);
    Rotate xRotate;
    Rotate yRotate;
    //Initialization of rotation
    parent.getTransforms().setAll
    (
      xRotate = new Rotate(0, Rotate.X_AXIS),
      yRotate = new Rotate(0, Rotate.Y_AXIS)
    );

    //Graphic is attached to root
    final Group root = new Group(parent);
    //Window of the scene
    final Scene scene = new Scene(root, 900, 900, true);
    //Background
    scene.setFill(Color.SKYBLUE);

    //Listener of mousePressed
    scene.setOnMousePressed
    (
        event ->
        {
            anchorX = event.getSceneX();
```

```
            anchorY = event.getSceneY();
            anchorAngleX = angleX.get();
            anchorAngleY = angleY.get();
        }
    );


    //Listener of mouseDragged
    scene.setOnMouseDragged
    (
        event ->
        {
            angleX.set(anchorAngleX
                    - (anchorY - event.getSceneY()));
            angleY.set(anchorAngleY
                    + anchorX - event.getSceneX());
        }
    );

    //Rotation of the scene gets bound to
    //the angle determined by  mouseDragged
    xRotate.angleProperty().bind(angleX);
    yRotate.angleProperty().bind(angleY);

    //Illumination
    PointLight pointLight = new PointLight(Color.WHITE);
    pointLight.setTranslateX(100);
    pointLight.setTranslateY(200);
    pointLight.setTranslateZ(-2000);
    root.getChildren().add(pointLight);

    //Camera
    scene.setCamera(camera);
    primaryStage.setScene(scene);
    PerspectiveCamera subSceneCamera
            = new PerspectiveCamera(false);
    scene.setCamera(subSceneCamera);

    //To zoom down the graphic,
    //it is translated away from  the beholder,
```

```
//along the Z-direction (normal to the screen).
//Change zProperty to change initial setting.
zSlider = new Slider();
zSlider.setMax(5000);
zSlider.setMin(1);
zSlider.setValue(0);


//Bind zProperty to the reading of the slider
zSlider.valueProperty().bindBidirectional(zProperty);
//Execute negative zoom
parent.translateZProperty().bind(zProperty);

//Slider for walking
walkSlider = new Slider();
walkSlider.setMax(45);
walkSlider.setMin(-45);
walkSlider.setValue(0);


//Bind walkProperty to the reading of the slider
walkSlider.valueProperty().bindBidirectional(walkProperty);



//Other JavaFX controls, labels and containers
GridPane gridPane = new GridPane();
gridPane.setVgap(4);
gridPane.setHgap(10);
gridPane.setPadding(new Insets(5, 5, 5, 5));
//Column 0, Row   0
gridPane.add(new Label("Zoom: "), 0, 0);
gridPane.add(zSlider, 1, 0);

gridPane.add(new Label("Walk: "), 0, 1);
gridPane.add(walkSlider, 1, 1);
```

```
    SubScene subScene
            = new SubScene(gridPane, 880, 200);
    subScene.setLayoutX(10);
    subScene.setLayoutY(800);
    root.getChildren().add(subScene);

    primaryStage.show();

    //The sense of touch is here
    parent.setOnMouseClicked((event)->
    {
        //Information that can be retrieved
        PickResult res = event.getPickResult();
        //Which node
        Node node = res.getIntersectedNode();
        //Id of node (null if not declared)
        String nodeId  = event.getPickResult()
                .getIntersectedNode().getId();
        //which face of the node (negative if is not enabled
        int face = event.getPickResult().getIntersectedFace();
        System.out.println("res = " + res);
        System.out.println("node = " + node);
        System.out.println("nodeID = " + nodeId);
        System.out.println("face" + face);
        //Change color to red, rotate around Z_AXIS
        if (node instanceof Box)
        {
            ((Box) node).setMaterial(
                    new PhongMaterial( Color.RED));
            humanoidGroup.rotationAxisProperty()
                    .setValue(Z_AXIS);
            int val = rd.nextInt(91);
            humanoidGroup.rotateProperty().setValue(val);
        }
    });

    SequentialTransition a = new SequentialTransition();

}
```

```
public void play()
{

}

@Override public void stop()
{

}

@Override
public void start(Stage primaryStage)  throws Exception
{
    init(primaryStage);
    primaryStage.show();

    play();
}


public static void main(String[] args)
{
    launch(args);
}


//Make the drawing
private Node create3dContent()
{
    //The scene can be rotated
    XformWithPivot sceneRoot = new XformWithPivot();

    getColors();
    //The constructor of body is invoked.
    Body  b =  new Body();
    sceneRoot.getChildren().add(b);
    //Initial view
    sceneRoot.rx.setAngle(125);
    sceneRoot.ry.setAngle(150);
```

```
    sceneRoot.rz.setAngle(90);

    return new Group(sceneRoot);
}

//Colors with a name in JavaFX are put into an array
static private Color[] getColors()
{
    //Colors with a name in JavaFX
    final Field[] fields = Color.class.getFields();


    int N = 0;
    //Colors are put into an array
    for (final Field field : fields)
    {
        if (field.getType() == Color.class)
        {
            try
            {
                final Color color = (Color) field.get(null);
                //Array to keep colors
                JavaFXColors[N] = color;
                /*
                //The name of the color
                final String colorName = field.getName();
                System.out.println("N = " + N + "  "
                        + color.toString() + " " + colorName);
                */
                N++;
            } catch (IllegalArgumentException
                    | IllegalAccessException ex)
            {
                Logger.getLogger(Humanoid.class
                    .getName()).log(Level.SEVERE, null, ex);
            }
       }
    }
    return JavaFXColors;
}
```

```
//A random color is synthesized
 private static Color randomColor()
 {
    //A JavaFX named color is picked
    //but the first, Transparent, is ignored
    int n = rd.nextInt(JavaFXColors.length-1)+1;
    Color c = JavaFXColors[n];
    //Some light colors are not visualized so,
    //they might be darkened:
    //c = c.darker();
    return c;
 }


  //=================================================
  //=============Inner class XformWithPivot===========
  //=================================================



//Kit of available families of transformations
private class XformWithPivot extends Group
{
    //Declaration and initialization
    public Rotate rx = new Rotate();
    { rx.setAxis(Rotate.X_AXIS); }
    public Rotate ry = new Rotate();
    { ry.setAxis(Rotate.Y_AXIS); }
    public Rotate rz = new Rotate();
    { rz.setAxis(Rotate.Z_AXIS); }
    //Rotation = angle (one coordinate)
    //+ pivot (3 coordinates) + axis.
    //This rotation is around X_AXIS
    public Rotate rwpx = new Rotate(0,0,0,0, X_AXIS);
    //This rotation is around Y_AXIS
    public Rotate rwpy = new Rotate(0,0,0,0, Y_AXIS);
    //This rotation is around Z_AXIS
    public Rotate rwpz = new Rotate(0,0,0,0, Z_AXIS);
```

```
    public XformWithPivot()
    {
        super();
        getTransforms().addAll( rx, ry, rz, rwpx, rwpy, rwpz );
    }
}//End of class XformWithPivot



  //=========================================
  //=============Inner class Hand============
  //=========================================



private class Body extends Group
{



//Constructor of the humanoid
//It is assembled completely in situ
//without Xforms
Body()
{
    //sphere of the head
    Sphere headSphere = new Sphere();
    headSphere.setRadius(40);
    Color c = randomColor();
    //Phong is a  word that in Vietnamese means wind.
    //It is used there as a name for babies.
    PhongMaterial materialHeadSphere = new PhongMaterial(c);
    headSphere.setMaterial(materialHeadSphere);
    //Box of the head
    Box headBox = new Box(100,60,70);
    Color c4 = randomColor();
    PhongMaterial materialHeadBox = new PhongMaterial(c4);
    headBox.setMaterial(materialHeadBox);
```

```
//Neck is constructed
Box neck = new Box(20,20,50);
Color c2 = randomColor();
PhongMaterial materialNeck = new PhongMaterial(c2);
neck.setMaterial(materialNeck);
neck.setTranslateZ(50);

//Shoulders are constructed
Box shoulders = new Box(240,60,10);
Color c3 = randomColor();
PhongMaterial materialShoulders = new PhongMaterial(c3);
shoulders.setMaterial(materialShoulders);
shoulders.setTranslateZ(80);

//Trunk
Box trunk = new Box(140,60,170);
Color c5 = randomColor();
PhongMaterial materialTrunk = new PhongMaterial(c5);
trunk.setMaterial(materialTrunk);
trunk.setTranslateZ(210);

//Arm1
Cylinder arm1 = new Cylinder();
arm1.setHeight(300);
arm1.setRadius(20);
arm1.setRotationAxis(X_AXIS);
arm1.setRotate(90);
arm1.setTranslateZ(220);
arm1.setTranslateX(100);
Color c6 = randomColor();
PhongMaterial materialArms = new PhongMaterial(c6);
arm1.setMaterial(materialArms);

//Arm2
Cylinder arm2 = new Cylinder();
arm2.setHeight(300);
arm2.setRadius(20);
arm2.setRotationAxis(X_AXIS);
arm2.setRotate(90);
arm2.setTranslateZ(220);
```

```
arm2.setTranslateX(-100);
arm2.setMaterial(materialArms);

//=====Leg1=====
//Leg11
Cylinder leg11 = new Cylinder();
leg11.setHeight(140);
leg11.setRadius(20);
Color c7 = randomColor();
PhongMaterial materialLeg11 = new PhongMaterial(c7);
leg11.setMaterial(materialLeg11);
leg11.setRotationAxis(X_AXIS);
leg11.setRotate(90);
leg11.setTranslateZ(365);
leg11.setTranslateX(50);

//Leg12
Cylinder leg12 = new Cylinder();
leg12.setHeight(140);
leg12.setRadius(20);
Color c8 = randomColor();
PhongMaterial materialLeg12 = new PhongMaterial(c8);
leg12.setMaterial(materialLeg12);
leg12.setRotationAxis(X_AXIS);
leg12.setRotate(90);
leg12.setTranslateZ(505);
leg12.setTranslateX(50);


//Foot1
Box foot1 = new Box(40,120, 30);
Color c9 = randomColor();
PhongMaterial materialFoot1 = new PhongMaterial(c9);
foot1.setMaterial(materialFoot1);
foot1.setTranslateZ(590);
foot1.setTranslateX(50);



//====Leg2====
```

```
//Leg21
Cylinder leg21 = new Cylinder();
leg21.setHeight(140);
leg21.setRadius(20);
Color c10 = randomColor();
PhongMaterial materialLeg21 = new PhongMaterial(c10);
leg21.setMaterial(materialLeg21);
leg21.setRotationAxis(X_AXIS);
leg21.setRotate(90);
leg21.setTranslateZ(365);
leg21.setTranslateX(-50);

//Leg22
Cylinder leg22 = new Cylinder();
leg22.setHeight(140);
leg22.setRadius(20);
Color c11 = randomColor();
PhongMaterial materialLeg22 = new PhongMaterial(c11);
leg22.setMaterial(materialLeg22);
leg22.setRotationAxis(X_AXIS);
leg22.setRotate(90);
leg22.setTranslateZ(505);
leg22.setTranslateX(-50);


//Foot2
Box foot2 = new Box(40, 120, 30);
Color c12 = randomColor();
PhongMaterial materialFoot2 = new PhongMaterial(c12);
foot2.setMaterial(materialFoot2);
foot2.setTranslateZ(590);
foot2.setTranslateX(-50);




//========Definition of control groups============
//Every item belongs in a group

//leg1 = foot1 + leg12 + leg11
```

```
XformWithPivot foot1Group = new XformWithPivot();
foot1Group.getChildren().add(foot1);
XformWithPivot leg12Group = new XformWithPivot();
leg12Group.getChildren().addAll(foot1Group, leg12);
XformWithPivot leg1Group = new XformWithPivot();
leg1Group.getChildren().addAll(leg12Group, leg11);


//leg2  = foot2 + leg22 + leg21
XformWithPivot foot2Group = new XformWithPivot();
foot2Group.getChildren().add(foot2);
XformWithPivot leg22Group = new XformWithPivot();
leg22Group.getChildren().addAll(foot2Group, leg22);
XformWithPivot leg2Group = new XformWithPivot();
leg2Group.getChildren().addAll(leg22Group, leg21);

//trunk
XformWithPivot trunkGroup = new XformWithPivot();
trunkGroup.getChildren()
        .addAll(leg1Group, leg2Group,trunk);

//arm1
XformWithPivot arm1Group = new XformWithPivot();
arm1Group.getChildren().addAll(arm1);
//arm2
XformWithPivot arm2Group = new XformWithPivot();
arm2Group.getChildren().addAll(arm2);

XformWithPivot shouldersGroup = new XformWithPivot();
shouldersGroup.getChildren().addAll(arm1Group,
        arm2Group, shoulders, trunkGroup);

//Palm and arm are attached to humanoidGroup
humanoidGroup.getChildren()
    .addAll
    (
        headSphere
        , headBox
        , neck
        , shouldersGroup
```

```
        );


    getChildren().addAll(humanoidGroup );

    //Control section

    //Legs go in contrary direction
    NumberBinding walkPropertyMinus
            = walkProperty.multiply(-1);

    //Leg1 is rotated
    leg1Group.rwpx.pivotZProperty().setValue(303);
    leg1Group.rwpx.angleProperty()
            .bindBidirectional(walkProperty);
    //Leg2 is rotated
    leg2Group.rwpx.pivotZProperty().setValue(303);
    leg2Group.rwpx.angleProperty()
            .bind(walkPropertyMinus);
    //Other movements to look more natural
    //???

    humanoidGroup.translateYProperty().bindBidirectional(walkProperty);


    //===========Animation=========

   //???




animation.play();

}//End of constructor of class Hand


}//End of inner class Hand
```

```
}//En of main class M30 Humanoid
```

31 ***Exercise.*** *Run the previous program and play with the code.*

32 ***Challenge.*** *Personalize the humanoid. Some ideas: divide each arm and each foot into two sections, make each part of the body to move at walking to look more naturally, include an animation with different rhythm of walking, insert an alive hand, have it jumping and moving arms to greet. Bellow you will learn more of JavaFX and you will get more ideas to continue the process of personalization.*

2.9 In hindsight

We have enjoyed the difficult task of converting an ensemble of boxes into a 3D shape that resembles a human hand. Now, we have a natural question: What did we suffer for? Well, we explored a path from simplicity to complexity with just one purpose: to get authority to judge the Evolutionary Theory.

33 ***Exercise.*** *From the stand of complexity theory propose a path that should had been stepped through by evolution in order to have a pair of hands such as ours.* *Answer*

2.10 Conclusion

We tinkered with 3D shapes to starting up with JavaFX 3D. We began with a hand that has all its 5 finger on the same line ad that can bend fingers to form a grip. Next we got a hand that can move its fingers transversally. Next we got a hand that can feel and another that can grasp things using two fingers as with a pair of pliers. Then we found a hand with an opposite thumb. At last we encountered a hand with opposable thumb.

Chapter 3

Evolution explains a virtual hand

Evolution under trial

34 ***Introduction and objective.*** *Evolution is a universal tool for problem solving. So, it is used by human beings to design objects, software included. De facto, we saw in the previous chapter how we, as software developers, devised a 3D shape that resembles a human hand. We have battled complexity with evolution: we devised and evolutionary process in the space of problems to make a path from simplicity to complexity.*

Therefore, we can say that our final product is explained by evolution. Being so, evolution is a very general term that encompasses everything. One cannot make science with such a generality. That is why we use to distinguish two types of evolution: Natural Evolution and Directed Evolution. Natural Evolution corresponds to a perfectly automatic mechanism composed of rounds of purposeless mutation + recombination ensued by natural differential reproduction. This description of Natural Evolution is due to Darwin and followers and so, we can equate Natural and Darwinian Evolutions. On the other hand, Directed Evolution results when evolution is armed with an artificial selection that is plugged by the observer together with hints that accelerate the rate of evolution towards complexity. Lamarckian and Directed Evolutions are synonyms for us because we consider that Lamarck made it clear that Evolution can be guided and accelerated (see Minds and hearts in Vol VII).

Our challenge in this chapter in this: Can we decide whether or not Evolution, Darwinian else Lamarckian, is a plausible explanation of a given virtual fossil record?

To tackle this question we follow the scientific method. Firstly, we devise a fossil record. Secondly, we propose that it is explained by Directed Evolution without hints, i.e., by Darwinian Evolution in which we plug the criterion for selection or differential reproduction. We pass to compute next the ensuing predictions of what the fossil record shall be. Thirdly, we compare predictions with the actual fossil record. If divergence is important, we are invited to reject the proposal that the fossil record is explained by considered type Evolution. If on the other hand, Directed Evolution without hints seems to be a good explanation of the fossil record, then we ask whether or not Darwinian Evolution is also a good explanation. To answer this question we repeat the same methodology as before.

It may happen that one is interested in falsifying a theory, i.e., in showing that it is false. Now, the general criterion to rule out a theory is to compute a mandatory prediction that is not supported by nature. In regard with Evolution we have a falsifying procedure by default: both, Natural and Directed Darwinian Evolutions shall pay a price (in general very high) in terms of wasted resourced to arrive to perfection (something complex and functional). Therefore, both types of evolution predict in general a huge amount of fossils that after reconstruction shall appear as imperfect in regard with function.

Now, if the frequency of imperfection found in the fossil record is depleted beyond random effects, we have two options. Fist, to propose that Natural Evolution explains the fossil record and to argue that the posed problem was easy for evolution (solved in some few generations and some few individuals) and that therefore left fossils were not too many and so they did not appear in the fossil record because of destruction + sampling effects. But if the posed problem is proved to be difficult for Evolution that would imply too many generations of evolution with too many imperfect fossils that are not found in nature. This will amount to a rejection of the evolutionary theory.

Let us implement this program.

3.1 Design of the fossil record

We take a proto-hand with one function and consecutively add more functions to devise a fossil record with a trend towards complexity.

35 ***An hypothetic fossil record of hands is shown by the next code.*** *It is assumed that simpler fossils are found in deeper strata and that each found fossil fits in one and only one of presented categories.*

```
/*
 * Program M35 FossilRecord.
 * We devise an hypothetic reconstructed fossil record.
 * First we have a hand that resembles a paddle
 * and that can be used to swing.
 * Next a hand with fingers that  plays the role of a hook
 * and that is useful for hanging from branches of trees.
 * Next a hand that has an extra finger. That is a preadaptation
 * that next will become an opposable thumb.
 * Next,  we get an opposable thumb
 * that is useful for grabbing clubs.
 * Next, the thumb is armed with fine control instrumentation
 * that is necessary  for the construction of tools.
 * Next,  the possibility to separate fingers is enabled,
 * a property that is needed to play the piano.
 * It is assumed that simpler fossils are found
 * in deeper  strata and that each found fossil
 * fits in one and only one category.
 * Sliders simulate muscles and neural connections.
 * The User simulates the brain.
 */

package fossilrecord;



import java.lang.reflect.Field;
import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.animation.Interpolator;
import javafx.animation.KeyFrame;
import javafx.animation.KeyValue;
import javafx.animation.Timeline;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.beans.binding.NumberBinding;
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;
import javafx.event.ActionEvent;
```

```
import javafx.geometry.Insets;

import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.PerspectiveCamera;
import javafx.scene.PointLight;
import javafx.scene.Scene;
import javafx.scene.SubScene;
import javafx.scene.control.Label;
import javafx.scene.control.Menu;
import javafx.scene.control.MenuBar;
import javafx.scene.control.MenuItem;
import javafx.scene.control.SeparatorMenuItem;
import javafx.scene.control.Slider;
import javafx.scene.input.PickResult;
import javafx.scene.layout.GridPane;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.Box;
import javafx.scene.transform.Rotate;
import static javafx.scene.transform.Rotate.X_AXIS;
import static javafx.scene.transform.Rotate.Y_AXIS;
import static javafx.scene.transform.Rotate.Z_AXIS;
import javafx.stage.Stage;
import javafx.util.Duration;



public class FossilRecord extends Application
{
    //Node with all elements
    XformWithPivot armGroup = new XformWithPivot();
    XformWithPivot finger4Ph2Group = new XformWithPivot();
    //Slider for zooming
    private Slider zSlider;
    //Position of graphic along the z-axis,
    //normal to the screen.
    //It is a double property because
    //it will get involved into a binding
```

```
    private final DoubleProperty zPos
            = new SimpleDoubleProperty(2000);

    //Hand that serves as a hook
    private final Label hookLabel
            = new Label("Hook: ");
    //Slider for control of the hook
    private final Slider hookSlider  = new Slider();
    //To open/close the hand,
    //each phalange suffers a rotation.
    //It is a double property because
    //it will get involved into a binding
    private final DoubleProperty hookProperty
            = new SimpleDoubleProperty(0);


    //To oppose the thumb we need a rotation.
    //It is a double property because
    //it will get involved into a binding
    private final DoubleProperty opposableThumbProperty
            = new SimpleDoubleProperty(0);


    //Slider for rotating the thumb to form the grip
    private final Slider opposableThumbSlider = new Slider();
    private final Label opposableThumbLabel
            = new Label("Opposable thumb: ");

    //The thumb can get separated from the palm
    private final DoubleProperty separableThumbProperty
            = new SimpleDoubleProperty(0);
    private final Slider separableThumbSlider = new Slider();
    private final Label separableThumbLabel
            = new Label("Separable thumb: ");

    //Slider for opening fingers thanks to
    //their transversal rotation
    private final Slider separableFingersSlider = new Slider();
    //To open the fingers,
    //each phalange suffers a transversal rotation.
```

```
    //It is a double property because
    //it will get involved into a binding
    private final DoubleProperty separableFingersProperty
            = new SimpleDoubleProperty(0);
    private final Label separableFingersLabel
            = new Label("Separable fingers: ");

    //For listener of mouse
    private double anchorAngleX = 0;
    private double anchorAngleY = 0;
    private double anchorX, anchorY;
    //Angles of rotation of the scene
    private final DoubleProperty angleX
            = new SimpleDoubleProperty(0);
    private final DoubleProperty angleY
            = new SimpleDoubleProperty(0);

    PerspectiveCamera camera
            = new PerspectiveCamera(false);

    //The application is instantiated
    static FossilRecord p = new FossilRecord();



    //An array with all colors that are named in JavaFX
    private static final Color JavaFXColors[]
                                = new Color[148];

    //Generator of pseudo random numbers
    private static final Random rd = new Random();

    //Number of fingers (without including the thumb).
    //The fifth finger s used as extra finger
    private static int nFingers = 5;

    //Number of phalanges of each finger
    private final int nPhalanges = 3;

    //Fingers with all 3 phalanges
```

```
    XformWithPivot finger0Ph210Group = new XformWithPivot();
    XformWithPivot finger1Ph210Group = new XformWithPivot();
    XformWithPivot finger2Ph210Group = new XformWithPivot();
    XformWithPivot finger3Ph210Group = new XformWithPivot();

    //Extra finger
    private final  XformWithPivot finger4Ph210Group
            = new XformWithPivot();
    //Thumb
    private final  XformWithPivot thumbPh210Group
            = new XformWithPivot();

    private static String title
            =  "An hypothetic fossil record. ";
    private static String subTitle  =  "";



    //Initial setting = hand as a hook


public void init(Stage primaryStage)
{
    //A hand that resembles a paddle is shown at the outset
    subTitle = "First stratum = paddle";
    primaryStage.setTitle(title + subTitle);
    hookLabel.setVisible(false);
    hookSlider.setVisible(false);
    separableThumbLabel.setVisible(false);
    separableThumbSlider.setVisible(false);
    opposableThumbLabel.setVisible(false);
    opposableThumbSlider.setVisible(false);
    separableFingersLabel.setVisible(false);
    separableFingersSlider.setVisible(false);




    //Menubar declaration and initialization
    MenuBar menuBar = new MenuBar();
```

```
// --- Menu General
Menu fossilsMenu = new Menu("Fossils");

// --Menu item declaration and initialization
MenuItem paddleMenu = new MenuItem("Paddle");


//Listener
paddleMenu.setOnAction((ActionEvent t) ->
{
    subTitle = "First stratum = paddle";
    primaryStage.setTitle(title + subTitle);
    hide(finger0Ph210Group);
    hide(finger1Ph210Group);
    hide(finger2Ph210Group);
    hide(finger3Ph210Group);
    hide(finger4Ph210Group);
    hide(thumbPh210Group);
    hookLabel.setVisible(false);
    hookSlider.setVisible(false);
    opposableThumbLabel.setVisible(false);
    opposableThumbSlider.setVisible(false);
    separableThumbLabel.setVisible(false);
    separableThumbSlider.setVisible(false);
    separableFingersLabel.setVisible(false);
    separableFingersSlider.setVisible(false);
});



//MenuItem added to menu
fossilsMenu.getItems().addAll(paddleMenu);

// --Menu item declaration and initialization
MenuItem hookMenu = new MenuItem("Hook");

//Listener
hookMenu.setOnAction((ActionEvent t) ->
{
    subTitle = "Second stratum = Hook";
```

```
    primaryStage.setTitle(title + subTitle);
    show(finger0Ph210Group);
    show(finger1Ph210Group);
    show(finger2Ph210Group);
    show(finger3Ph210Group);
    hide(finger4Ph210Group);
    hide(thumbPh210Group);
    hookLabel.setVisible(true);
    hookSlider.setVisible(true);
    opposableThumbLabel.setVisible(false);
    opposableThumbSlider.setVisible(false);
    separableThumbLabel.setVisible(false);
    separableThumbSlider.setVisible(false);
    separableFingersLabel.setVisible(false);
    separableFingersSlider.setVisible(false);
});



//MenuItems added to menu
fossilsMenu.getItems().addAll(hookMenu);



// --Menu item declaration and initialization
MenuItem extraFingerMenu
        = new MenuItem("Extra finger");

//Listener
extraFingerMenu.setOnAction((ActionEvent t) ->
{
    subTitle = "Third stratum = extra finger";
    primaryStage.setTitle(title + subTitle);
    show(finger0Ph210Group);
    show(finger1Ph210Group);
    show(finger2Ph210Group);
    show(finger3Ph210Group);
    show(finger4Ph210Group);
    hide(thumbPh210Group);
    hookLabel.setVisible(true);
```

```
        hookSlider.setVisible(true);
        opposableThumbLabel.setVisible(false);
        opposableThumbSlider.setVisible(false);
        separableThumbLabel.setVisible(false);
        separableThumbSlider.setVisible(false);
        separableFingersLabel.setVisible(false);
        separableFingersSlider.setVisible(false);
    });



    //MenuItems added to menu
    fossilsMenu.getItems().addAll(extraFingerMenu);

    // --Menu item declaration and initialization
    MenuItem opposableThumbMenu
            = new MenuItem("OpposableThumb");

    opposableThumbMenu.setOnAction((ActionEvent t) ->
    {
        subTitle = "Fourth stratum = opposable thumb";
        primaryStage.setTitle(title + subTitle);
        show(finger0Ph210Group);
        show(finger1Ph210Group);
        show(finger2Ph210Group);
        show(finger3Ph210Group);
        hide(finger4Ph210Group);
        show(thumbPh210Group);
        hookLabel.setVisible(true);
        hookSlider.setVisible(true);
        opposableThumbLabel.setVisible(true);
        opposableThumbSlider.setVisible(true);
        separableThumbLabel.setVisible(false);
        separableThumbSlider.setVisible(false);
        separableFingersLabel.setVisible(false);
        separableFingersSlider.setVisible(false);
    });

    // --Menu item declaration and initialization
    MenuItem separableThumbMenu = new MenuItem("Separable thumb");
```

```
separableThumbMenu.setOnAction((ActionEvent t) ->
{
    subTitle = "Fifth stratum = separable thumb";
    primaryStage.setTitle(title + subTitle);
    show(finger0Ph210Group);
    show(finger1Ph210Group);
    show(finger2Ph210Group);
    show(finger3Ph210Group);
    hide(finger4Ph210Group);
    show(thumbPh210Group);
    hookLabel.setVisible(true);
    hookSlider.setVisible(true);
    opposableThumbLabel.setVisible(true);
    opposableThumbSlider.setVisible(true);
    separableThumbLabel.setVisible(true);
    separableThumbSlider.setVisible(true);
    separableFingersLabel.setVisible(false);
    separableFingersSlider.setVisible(false);


});

// --Menu item declaration and initialization
MenuItem separableFingersMenu = new MenuItem("Separable Fingers");

separableFingersMenu.setOnAction((ActionEvent t) ->
{
    subTitle = "Sixth stratum = separable fingers";
    primaryStage.setTitle(title + subTitle);
    show(finger0Ph210Group);
    show(finger1Ph210Group);
    show(finger2Ph210Group);
    show(finger3Ph210Group);
    hide(finger4Ph210Group);
    show(thumbPh210Group);
    hookLabel.setVisible(true);
    hookSlider.setVisible(true);
    opposableThumbLabel.setVisible(true);
    opposableThumbSlider.setVisible(true);
```

```
    separableThumbLabel.setVisible(true);
    separableThumbSlider.setVisible(true);
    separableFingersLabel.setVisible(true);
    separableFingersSlider.setVisible(true);

});

// --Menu item declaration and initialization
MenuItem exit = new MenuItem("Exit");
exit.setOnAction((ActionEvent t) ->
{
    //Finish everything and exit
    System.exit(0);
});

//Booking
fossilsMenu.getItems()
        .addAll(opposableThumbMenu
                , separableThumbMenu
                ,  separableFingersMenu
                , new SeparatorMenuItem()
                , exit);
menuBar.getMenus()
        .addAll(fossilsMenu);


//The whole drawing  is created
//as a JavaFX space-like object
final Group parent = new Group(create3dContent());


//Initial position
parent.setTranslateX(450);
parent.setTranslateY(400);
parent.setTranslateZ(0);
Rotate xRotate;
Rotate yRotate;
//Initialization of rotation
parent.getTransforms().setAll
(
```

```
  xRotate = new Rotate(0, Rotate.X_AXIS),
  yRotate = new Rotate(0, Rotate.Y_AXIS)
);

//Graphic is attached to root
final Group root = new Group(parent);
//Window of the scene
final Scene scene = new Scene(root, 900, 900, true);
//Background
scene.setFill(Color.SKYBLUE);

//Listener of mousePressed
scene.setOnMousePressed
(
    event ->
    {
        anchorX = event.getSceneX();
        anchorY = event.getSceneY();
        anchorAngleX = angleX.get();
        anchorAngleY = angleY.get();
    }
);

root.getChildren().add(menuBar);
//Listener of mouseDragged
scene.setOnMouseDragged
(
    event ->
    {
        angleX.set(anchorAngleX
                - (anchorY - event.getSceneY()));
        angleY.set(anchorAngleY
                + anchorX - event.getSceneX());
    }
);

//Rotation of the scene gets bound to
//the angle determined by  mouseDragged
xRotate.angleProperty().bind(angleX);
yRotate.angleProperty().bind(angleY);
```

```
//Illumination
PointLight pointLight = new PointLight(Color.WHITE);
pointLight.setTranslateX(100);
pointLight.setTranslateY(200);
pointLight.setTranslateZ(-2000);
root.getChildren().add(pointLight);

//Camera
scene.setCamera(camera);
primaryStage.setScene(scene);
PerspectiveCamera subSceneCamera
        = new PerspectiveCamera(false);
scene.setCamera(subSceneCamera);

//To zoom down the graphic,
//it is translated away from  the beholder,
//along the Z-direction (normal to the screen).
//Change zPos to change initial setting.
zSlider = new Slider();
zSlider.setMax(5000);
zSlider.setMin(1);


//Bind zPos to the reading of the slider
zSlider.valueProperty().bindBidirectional(zPos);
//Execute negative zoom
parent.translateZProperty().bind(zPos);



//This slider controls the opposing o f the thumb;
hookSlider.setMax(90);
hookSlider.setMin(0);
hookSlider.setValue(10);

//To form a hook, an angle of rotation is enabled.
//Bind hookProperty to the reading of the slider
hookSlider.valueProperty()
        .bindBidirectional(hookProperty);
```

```
//Slider to oppose  the thumb
opposableThumbSlider.setMax(50);
opposableThumbSlider.setMin(0);
opposableThumbSlider.setValue(0);
opposableThumbSlider
        .adjustValue(opposableThumbSlider.getValue());

//To oppose the thumb, an angle of rotation is enabled.
//Bind opposableThumbProperty to the reading of the slider
opposableThumbSlider.valueProperty()
        .bindBidirectional(opposableThumbProperty);


//Slider to separate the thumb
separableThumbSlider.setMax(55);
separableThumbSlider.setMin(0);
separableThumbSlider.setValue(0);
separableThumbSlider
        .adjustValue(separableThumbSlider.getValue());

//To separate the thumb, an angle of rotation is enabled.
//Bind separableThumbProperty to the reading of the slider
separableThumbSlider.valueProperty()
        .bindBidirectional(separableThumbProperty);


 //This slider controls the separation of fingers.
separableFingersSlider.setMax(15);
separableFingersSlider.setMin(0);
separableFingersSlider.setValue(10);

//To separate fingers, an angle of rotation is enabled.
//Bind separableFingersProperty to the reading of the slider
separableFingersSlider.valueProperty()
        .bindBidirectional(separableFingersProperty);


//Other JavaFX controls, labels and containers
GridPane gridPane = new GridPane();
```

```
gridPane.setVgap(4);
gridPane.setHgap(10);
gridPane.setPadding(new Insets(5, 5, 5, 5));
//Column 0, Row   0
gridPane.add(new Label("Zoom: "), 0, 0);
gridPane.add(zSlider, 1, 0);
gridPane.add(hookLabel, 2, 0);
gridPane.add(hookSlider, 3, 0);
gridPane.add(opposableThumbLabel, 0, 1);
gridPane.add(opposableThumbSlider, 1,1);
gridPane.add(separableThumbLabel, 2, 1);
gridPane.add(separableThumbSlider, 3, 1);

gridPane.add(separableFingersLabel, 0, 2);
gridPane.add(separableFingersSlider, 1, 2);



SubScene subScene
        = new SubScene(gridPane, 880, 200);
subScene.setLayoutX(10);
subScene.setLayoutY(800);
root.getChildren().add(subScene);

primaryStage.show();

//The sense of touch is here
parent.setOnMouseClicked((event)->
{
    //Information that can be retrieved
    PickResult res = event.getPickResult();
    //Which node
    Node node = res.getIntersectedNode();
    //Identity label of node (null if not declared)
    String nodeId  = event.getPickResult()
            .getIntersectedNode().getId();
    //which face of the node (negative if is not enabled
    int face = event.getPickResult().getIntersectedFace();
    System.out.println("res = " + res);
    System.out.println("node = " + node);
```

```
        System.out.println("nodeID = " + nodeId);
        System.out.println("face" + face);
        //Change color to red, rotate around Z_AXIS
        if (node instanceof Box)
            {
                ((Box) node).setMaterial(
                        new PhongMaterial( Color.RED));
                armGroup.rotationAxisProperty()
                        .setValue(Z_AXIS);
                int val = rd.nextInt(91);
                armGroup.rotateProperty().setValue(val);
                finger4Ph2Group.rotateProperty().setValue(val);
            }




    });


}



@Override
public void start(Stage primaryStage)  throws Exception
{
    init(primaryStage);
    primaryStage.show();
}


public static void main(String[] args)
{
    launch(args);
}


//Make the drawing
```

```
private Node create3dContent()
{
    //The scene can be rotated
    XformWithPivot sceneRoot = new XformWithPivot();
    sceneRoot.rx.setAngle(45);
    sceneRoot.ry.setAngle(110);
    getColors();
    //The constructor of hand is invoked.
    Hand  h =  new Hand();
    sceneRoot.getChildren().add(h);
    return new Group(sceneRoot);
}

//Colors with a name in JavaFX are put into an array
static private Color[] getColors()
{
    //Colors with a name in JavaFX
    final Field[] fields = Color.class.getFields();


    int N = 0;
    //Colors are put into an array
    for (final Field field : fields)
    {
        if (field.getType() == Color.class)
        {
            try
            {
                final Color color = (Color) field.get(null);
                //Array to keep colors
                JavaFXColors[N] = color;
                /*
                //The name of the color
                final String colorName = field.getName();
                System.out.println("N = " + N + "  "
                        + color.toString() + " " + colorName);
                */
                N++;
            } catch (IllegalArgumentException
                    | IllegalAccessException ex)
```

```
            {
                Logger.getLogger(FossilRecord.class
                    .getName()).log(Level.SEVERE, null, ex);
            }
        }
    }
    return JavaFXColors;
}


//A random color is synthesized
 private static Color randomColor()
 {
    //A JavaFX named color is picked
    //but the first, Transparent, is ignored
    int n = rd.nextInt(JavaFXColors.length-1)+1;
    Color c = JavaFXColors[n];
    //Some light colors are not visualized so,
    //they might be darkened:
    //c = c.darker();
    return c;
 }


//Group is hidden
private static void  hide(Group group)
{

    Timeline hideGroup  = new Timeline();
    hideGroup.setCycleCount(1);
    Duration drtn =   Duration.millis(1);
    hideGroup.getKeyFrames().addAll(
        //Visible property is flipped from true to false
        new KeyFrame(Duration.ZERO,
            new KeyValue(group
                .visibleProperty()
                , true
                ,Interpolator.LINEAR)),
        new KeyFrame(drtn,
            new KeyValue(group
```

```
                .visibleProperty()
                , false
                ,Interpolator.LINEAR)));
     hideGroup.play();
}

//Group is shown
private static void  show(Group group)
{

    Timeline showGroup  = new Timeline();
    showGroup.setCycleCount(1);
    Duration drtn =   Duration.millis(1);
    double endTime1 = 1;
    showGroup.getKeyFrames().addAll(
    //Visible property is flipped from false to true
    new KeyFrame(Duration.ZERO,
        new KeyValue(group
            .visibleProperty()
            , false
            ,Interpolator.LINEAR)),
    new KeyFrame(drtn,
        new KeyValue(group
            .visibleProperty()
            , true
            ,Interpolator.LINEAR)));
    showGroup.play();
}


//================================================
//============Inner class XformWithPivot==========
//================================================



//Kit of available families of transformations
private class XformWithPivot extends Group
{
    //Declaration and initialization
```

```
    public Rotate rx = new Rotate();
    { rx.setAxis(Rotate.X_AXIS); }
    public Rotate ry = new Rotate();
    { ry.setAxis(Rotate.Y_AXIS); }
    public Rotate rz = new Rotate();
    { rz.setAxis(Rotate.Z_AXIS); }
    //Rotation = angle (one coordinate)
    //+ pivot (3 coordinates) + axis.
    //This rotation is around X_AXIS
    public Rotate rwpx = new Rotate(0,0,0,0, X_AXIS);
    //This rotation is around Y_AXIS
    public Rotate rwpy = new Rotate(0,0,0,0, Y_AXIS);
    //This rotation is around Z_AXIS
    public Rotate rwpz = new Rotate(0,0,0,0, Z_AXIS);

    public XformWithPivot()
    {
        super();
        getTransforms().addAll( rx, ry, rz, rwpx, rwpy, rwpz );
    }
}//End of class XformWithPivot



  //=========================================
  //============Inner class Hand============
  //=========================================



private class Hand extends Group
{



    //Nested definition of   nodes is implemented
    //to have, say,  the palm dragging the fingers,
    //and a phalanx dragging the rest of its the finger.
```

```
    //Each finger generates various natural groups:
    //Group with the extreme phalanx of finger0
    //Symbolic initialization
    XformWithPivot finger0Ph2Group = new XformWithPivot();
    //Group with the two extreme phalanges
    XformWithPivot finger0Ph21Group = new XformWithPivot();

    //Finger1
    XformWithPivot finger1Ph2Group = new XformWithPivot();
    XformWithPivot finger1Ph21Group = new XformWithPivot();

    //Finger2
    XformWithPivot finger2Ph2Group = new XformWithPivot();
    XformWithPivot finger2Ph21Group = new XformWithPivot();

    //Finger3
    XformWithPivot finger3Ph2Group = new XformWithPivot();
    XformWithPivot finger3Ph21Group = new XformWithPivot();

    //Finger4
    XformWithPivot finger4Ph2Group = new XformWithPivot();
    XformWithPivot finger4Ph21Group = new XformWithPivot();

    //Thumb
    XformWithPivot thumbPh2Group = new XformWithPivot();
    XformWithPivot thumbPh21Group = new XformWithPivot();


    //Group with palm and fingers
    XformWithPivot palmGroup = new XformWithPivot();


Hand()
{

    //Arm is constructed
    Box arm = new Box(100,20,400);
    Color c = randomColor();
    PhongMaterial materialArm = new PhongMaterial(c);
    arm.setMaterial(materialArm);
```

```
arm.setTranslateY(20);
//Palm is constructed
Box palm = new Box(125,20,140);
Color c2 = randomColor();
PhongMaterial materialPalm = new PhongMaterial(c2);
palm.setMaterial(materialPalm);
palm.setTranslateZ(270);
palm.setTranslateY(20);



//All fingers on the same line.

Finger Fingers[] = new Finger[nFingers];

//Groups are constructed
for(int i = 0; i < nFingers; i++)
{
    //Each finger has nPhalanges phalanges
    Fingers[i] = new Finger(nPhalanges);
    for(int j =0; j < nPhalanges; j++)
    {
        Box ph = Fingers[i].Phalanges[j].box;

        ph.setTranslateX(15.625 * (4-2*(i))-15.625);
        ph.setTranslateZ(360 + 40*j);
        ph.setTranslateY(20);

        //Natural groups are instantiated
        //Finger 0 and its subgroups
        if (i == 0)
        {
            if (j == 2) finger0Ph2Group.getChildren()
                    .add(ph);

            if (j == 1) finger0Ph21Group
                    .getChildren()
                    .addAll(finger0Ph2Group, ph);
            if (j == 0) finger0Ph210Group
                    .getChildren()
```

```
            .addAll(finger0Ph21Group, ph);
}

//Finger 1  and its subgroups
if (i == 1)
{
    if (j == 2) finger1Ph2Group
            .getChildren().add(ph);
    if (j == 1) finger1Ph21Group
            .getChildren()
            .addAll(finger1Ph2Group, ph);
    if (j == 0) finger1Ph210Group
            .getChildren()
            .addAll(finger1Ph21Group, ph);
}


//Finger 2  and its subgroups
if (i == 2)
{
    if (j == 2) finger2Ph2Group
            .getChildren().add(ph);
    if (j == 1) finger2Ph21Group
            .getChildren()
            .addAll(finger2Ph2Group, ph);
    if (j == 0) finger2Ph210Group
            .getChildren()
            .addAll(finger2Ph21Group, ph);
}


//Finger 3 and its subgroups
if (i == 3)
{
    if (j == 2) finger3Ph2Group.getChildren().add(ph);
    if (j == 1) finger3Ph21Group
            .getChildren()
            .addAll(finger3Ph2Group, ph);
    if (j == 0) finger3Ph210Group
            .getChildren()
```

```
                    .addAll(finger3Ph21Group, ph);
        }


        //Finger 4 and its subgroups
        if (i == 4)
        {
            if (j == 2) finger4Ph2Group.getChildren().add(ph);
            if (j == 1) finger4Ph21Group
                    .getChildren()
                    .addAll(finger4Ph2Group, ph);
            if (j == 0) finger4Ph210Group
                    .getChildren()
                    .addAll(finger4Ph21Group, ph);
        }

    }
}

//Our thumb  has 3 phalanges
Finger thumb = new Finger(3);
for(int j = 0; j < nPhalanges; j++)
    {
        Box ph = thumb.Phalanges[j].box;
        //When the thumb is aligned with the hand
        //it has a rotation around its vertical axis
        ph.setRotationAxis(Z_AXIS);
        ph.setRotate(45);
        //Initial position
        ph.setTranslateX(-78.125);
        ph.setTranslateY(15);
        ph.setTranslateZ(250 + 40*j);



    }
//Natural groups of control
thumbPh2Group.getChildren().add(thumb
        .Phalanges[2].box);
thumbPh21Group.getChildren()
```

```
        .addAll(thumbPh2Group
                , thumb.Phalanges[1].box);
thumbPh210Group.getChildren()
        .addAll(thumbPh21Group
                , thumb.Phalanges[0].box);
//Original inclination of the thumb
//in within the   plane of the palm

thumbPh210Group.getTransforms()
        .add(new Rotate(-55, //Angle
                -78.125, 15, 250, //Pivot
                Y_AXIS)); //Axis

//Fingers are attached to the palm
palmGroup.getChildren().addAll(palm
        , finger0Ph210Group
        , finger1Ph210Group
        , finger2Ph210Group
        , finger3Ph210Group
        , finger4Ph210Group
        , thumbPh210Group);
//Palm and arm are attached to armGroup
armGroup.getChildren().addAll(arm, palmGroup );

//Only palm is shown at start
hide(finger0Ph210Group);
hide(finger1Ph210Group);
hide(finger2Ph210Group);
hide(finger3Ph210Group);
hide(finger4Ph210Group);
hide(thumbPh210Group);



//========HAND AS A HOOK============


//Specifications of movement for subgroups of finger 0
finger0Ph2Group.rwpx.pivotZProperty().setValue(420.0);
finger0Ph2Group.rwpx.pivotYProperty().setValue(10.0);
```

```
finger0Ph2Group.rwpx.angleProperty()
        .bindBidirectional(hookProperty);

finger0Ph21Group.rwpx.pivotZProperty().setValue(380.0);
finger0Ph21Group.rwpx.pivotYProperty().setValue(10.0);
finger0Ph21Group.rwpx.angleProperty()
        .bindBidirectional(hookProperty);

finger0Ph210Group.rwpx.pivotZProperty().setValue(340.0);
finger0Ph210Group.rwpx.pivotYProperty().setValue(10.0);
finger0Ph210Group.rwpx.angleProperty()
        .bindBidirectional(hookProperty);

//Specifications of movement for subgroups of finger 1
finger1Ph2Group.rwpx.pivotZProperty().setValue(420.0);
finger1Ph2Group.rwpx.pivotYProperty().setValue(10.0);
finger1Ph2Group.rwpx.angleProperty()
        .bindBidirectional(hookProperty);

finger1Ph21Group.rwpx.pivotZProperty().setValue(380.0);
finger1Ph21Group.rwpx.pivotYProperty().setValue(10.0);
finger1Ph21Group.rwpx.angleProperty()
        .bindBidirectional(hookProperty);

finger1Ph210Group.rwpx.pivotZProperty().setValue(340.0);
finger1Ph210Group.rwpx.pivotYProperty().setValue(10.0);
finger1Ph210Group.rwpx.angleProperty()
        .bindBidirectional(hookProperty);

//Specifications of movement for subgroups of finger 2
finger2Ph2Group.rwpx.pivotZProperty().setValue(420.0);
finger2Ph2Group.rwpx.pivotYProperty().setValue(10.0);
finger2Ph2Group.rwpx.angleProperty()
        .bindBidirectional(hookProperty);

finger2Ph21Group.rwpx.pivotZProperty().setValue(380.0);
finger2Ph21Group.rwpx.pivotYProperty().setValue(10.0);
finger2Ph21Group.rwpx.angleProperty()
        .bindBidirectional(hookProperty);
```

```
finger2Ph210Group.rwpx.pivotZProperty().setValue(340.0);
finger2Ph210Group.rwpx.pivotYProperty().setValue(10.0);
finger2Ph210Group.rwpx.angleProperty()
        .bindBidirectional(hookProperty);

//Specifications of movement for subgroups of finger 3
finger3Ph2Group.rwpx.pivotZProperty().setValue(420.0);
finger3Ph2Group.rwpx.pivotYProperty().setValue(10.0);
finger3Ph2Group.rwpx.angleProperty()
        .bindBidirectional(hookProperty);

finger3Ph21Group.rwpx.pivotZProperty().setValue(380.0);
finger3Ph21Group.rwpx.pivotYProperty().setValue(10.0);
finger3Ph21Group.rwpx.angleProperty()
        .bindBidirectional(hookProperty);

finger3Ph210Group.rwpx.pivotZProperty().setValue(340.0);
finger3Ph210Group.rwpx.pivotYProperty().setValue(10.0);
finger3Ph210Group.rwpx.angleProperty()
        .bindBidirectional(hookProperty);

//Specifications of movement for subgroups of finger 4
finger4Ph2Group.rwpx.pivotZProperty().setValue(420.0);
finger4Ph2Group.rwpx.pivotYProperty().setValue(10.0);
finger4Ph2Group.rwpx.angleProperty()
        .bindBidirectional(hookProperty);

finger4Ph21Group.rwpx.pivotZProperty().setValue(380.0);
finger4Ph21Group.rwpx.pivotYProperty().setValue(10.0);
finger4Ph21Group.rwpx.angleProperty()
        .bindBidirectional(hookProperty);

finger4Ph210Group.rwpx.pivotZProperty().setValue(340.0);
finger4Ph210Group.rwpx.pivotYProperty().setValue(10.0);
finger4Ph210Group.rwpx.angleProperty()
        .bindBidirectional(hookProperty);



//====== The thumb ========
```

```
//Specifications of movement for the thumb


NumberBinding opposableThumbPropertyDown
        = opposableThumbProperty.multiply(0.2);

//Tranversal movement that separate thumb
thumbPh210Group.rwpy.pivotXProperty().setValue(-78.125);
thumbPh210Group.rwpy.pivotZProperty().setValue(240.0);
thumbPh210Group.rwpy.angleProperty().bind(separableThumbProperty);


//Rotations of phalanges 2 and 1 to make the grip.
//Rotations around X_AXIS and Y_AXIS at the same time
//Phalanx 2
thumbPh2Group.rwpx.pivotXProperty().setValue(-70.125);
thumbPh2Group.rwpx.pivotYProperty().setValue(15);
thumbPh2Group.rwpx.pivotZProperty().setValue(310.0);
thumbPh2Group.rwpx.angleProperty()
        .bind(opposableThumbPropertyDown);
//Phalanges 2 and 1
thumbPh21Group.rwpx.pivotXProperty().setValue(-70.125);
thumbPh21Group.rwpx.pivotYProperty().setValue(15);
thumbPh21Group.rwpx.pivotZProperty().setValue(290.0);
thumbPh21Group.rwpx.angleProperty()
        .bind(opposableThumbPropertyDown);

//Phalanx 2
thumbPh2Group.rwpy.pivotXProperty().setValue(-70.125);
thumbPh2Group.rwpy.pivotYProperty().setValue(15);
thumbPh2Group.rwpy.pivotZProperty().setValue(310.0);
thumbPh2Group.rwpy.angleProperty()
        .bind(opposableThumbPropertyDown);
//Phalanges 2 and 1
thumbPh21Group.rwpy.pivotXProperty().setValue(-70.125);
thumbPh21Group.rwpy.pivotYProperty().setValue(15);
thumbPh21Group.rwpy.pivotZProperty().setValue(290.0);
thumbPh21Group.rwpy.angleProperty()
        .bind(opposableThumbPropertyDown);
```

```
//Rotation of the whole thumb
//to get opposite to other fingers
thumbPh210Group.rwpz.pivotXProperty().setValue(-70.125);
thumbPh210Group.rwpz.pivotYProperty().setValue(15);
thumbPh210Group.rwpz.pivotZProperty().setValue(250.0);
thumbPh210Group.rwpz.angleProperty()
        .bind(opposableThumbProperty.multiply(2));


//======SEPARATION OF FINGERS========

//Fingers that are at the border must
//open twice faster as those
//near the center.
NumberBinding separableFingersPropertyTwo
        = separableFingersProperty.multiply(2);
//Specifications of movement for subgroups of finger 0
finger0Ph210Group.rwpy.pivotXProperty().setValue(0.0);
finger0Ph210Group.rwpy.pivotYProperty().setValue(10.0);
finger0Ph210Group.rwpy.pivotZProperty().setValue(340.0);
finger0Ph210Group.rwpy.angleProperty()
        .bind(separableFingersPropertyTwo);


//Specifications of movement for subgroups of finger 1
finger1Ph210Group.rwpy.pivotXProperty().setValue(0.0);
finger1Ph210Group.rwpy.pivotYProperty().setValue(30.0);
finger1Ph210Group.rwpy.pivotZProperty().setValue(340.0);

finger1Ph210Group.rwpy.angleProperty()
        .bind(separableFingersProperty);

//Finger two remains invariant under opening of fingers

//Opening in the contrary direction of fingers 3 and 4
NumberBinding separableFingersPropertyMinus
        = separableFingersProperty.multiply(-1);

//Specifications of movement for finger 3
finger3Ph210Group.rwpy.pivotXProperty().setValue(0.0);
```

```
    finger3Ph210Group.rwpy.pivotYProperty().setValue(30.0);
    finger3Ph210Group.rwpy.pivotZProperty().setValue(340.0);
    finger3Ph210Group.rwpy.angleProperty()
            .bind(separableFingersPropertyMinus);

    NumberBinding separableFingersPropertyMinusTwo
            = separableFingersProperty.multiply(-2);

    //Specifications of movement for finger 4
    finger4Ph210Group.rwpy.pivotXProperty().setValue(0.0);
    finger4Ph210Group.rwpy.pivotYProperty().setValue(10.0);
    finger4Ph210Group.rwpy.pivotZProperty().setValue(340.0);
    finger4Ph210Group.rwpy.angleProperty()
            .bind(separableFingersPropertyMinusTwo);


    getChildren().addAll(armGroup );




}//End of constructor of class Hand


}//End of inner class Hand




//A finger is an ensemble of phalanges
private class Finger extends Group
{

    Phalanx[] Phalanges    = new Phalanx [nPhalanges];

    //Constructor
    Finger(int nPhalanges)
    {
        for(int i =0; i < nPhalanges; i++)
        {
```

```
            Phalanx ph = new Phalanx(31.25,20,40);
            Phalanges[i] = ph;
        }
    }
}//End of inner class Finger



//A phalanx is a JavaFX 3D box
private class Phalanx extends Group
{
    Box box;

    //Constructor
    Phalanx(double width, int length, int height)
    {
        box = new Box(width, length, height);
        Color c = randomColor();
        PhongMaterial materialFinger
                = new PhongMaterial(c);
        box.setMaterial(materialFinger);
    }
}//End of inner class phalanx

}//End of main class M35 FossilRecord
```

36 ***Exercise.*** *Run the previous program and play with the code.*

37 ***Exercise.*** *Play the role of an archaeologist that founds the fossil record depicted in the previous program and make the corresponding report. Evolution is invoked as the obvious explanation of the record but no distinction of Darwinian from Lamarckian evolution was made.* *Answer*

3.2 Directed Evolution under trial

As it was shown by the previous exercise, our fictitious fossil record can be naturally accommodated along an evolutionary process that will be ipso facto accepted by current science. Things are not like that in our community.

38 ***Our insight.***

The evolution we have in mind is a mechanistic phenomenon that is mediated by genetic material. Now, the genome is software, i.e., it contains verbal instructions to do things. Conveyed instructions can be rewritten in any human language rather easily, say, most of them encode for the construction of poly-peptides that can be edited to become proteins. The last purpose of the genetic software is to continue extant life on Earth by making a specific living being.

After thousands of hours of experience in software development, the Author considers that the genetic software is a marvel. Actually, life continues to inspire inventors, say, robots that can see in 3D are just trying to imitate our natural capability for vision. So, the genome is genetic software that is immensely creative and difficult to concoct. But, what does it mean that a piece of software is difficult to develop? Operationally, it means that the developer makes many bugs in the process of design with the aggravating factor that the fixation of committed bugs generate more bugs.

So, the natural stand of a software developer to judge a proposal of evolution is that of software design. When evolution is in this regard invoked to explain the existence of software we speak of directed evolution: evolution in which selection is guided by the designer to attain a predefined final output.

There is nevertheless two great differences between the insight of a software engineer and that of a biologist. First: engineers work in the realm of purposeful design while natural evolution has no purpose at all apart from surviving and reproducing. Second: engineers use a lot of tricks to easy their work. For instance, the very fact of using Java as programming language is by far more easy than using, say, Assembler or Lisp. Besides, engineers use a great library of hints (all what they were taught at the University) altogether with a tremendous power of making hints (what is usually called intelligence or creativeness). The importance of hints is that they diminish the proportion of bugs and the ensuing generation of more bugs at the time of correction. That is why we distinguish two types of evolution: Darwinian and Lamarckian. The first relies on nude evolution: random mutation and recombination + natural differential reproduction while the second incorporates hints to accelerate the pace of evolution.

For the sake of clarity, let us begin our analysis of our fictitious fossil record from the stand of engineers which is characterized by teleonomy (design with a purpose) + hints.

39 ***Analysis by an engineer.*** *The fossil seems to be designed by an engineer:*

complexity of designing a hand was attacked by dividing the task in various subtasks to next generate a process that begins with simplicity and adds more and more functions to end up with a highly developed hand.

Let us imagine now how various evolutionists would face our fictitious fossil record.

40 ***Analysis from the stand of Directed Darwinian Evolution.***

To try to explain our fictitious fossil record by Directed Darwinian Evolution we need to define an evolutionary environment which includes the objects that can suffer mutation and recombination and to implement the purpose of designing a hand as the selection rule. It happens that the purpose of designing a hand is too complex to be simulated. So, to illustrate the modus operandis of a Directed Darwinian Evolutionist let us pick up for analysis a sub-purpose: let us pay attention to the change from the paddle to the hand with fingers that functions as a hook.

A Directed Evolutionist might explain this change as follows: a paddle is composed of two boxes, one serves as support, as arm, and the other as the very paddle. So, the genome must include a set of instructions to built boxes of varying dimensions together with the instructions to concatenate those units. It is clear that the mutations of this machinery are capable of making the changes that are necessary to pass from a paddle to a hand with fingers and that functions as a hook. Nevertheless, let us recall that the fossil record shows no intermediate stage in within. To explain this vacuum by Directed Darwinian Evolution we must postulate that the corresponding problem was easy for the considered type of evolution. This implies that there should had been a battery of mutations that made the required changes highly probable. Therefore, the evolutionary process was very fast leaving very few fossils that next were destroyed by wearing.

We think that this argument is reasonable. Concretely, gene amplification of the instructions encoding for boxes could had created in first place a finger beginning with just one phalanx. Then, another operation of gene amplification could had multiplied the number of fingers. And that would be all to it.

We find the aforementioned discussion simple, nice and worth of passing from being a fuzzy proposal to a concrete simulation in JavaFX. Its final purpose would be to use evolution to chase the known parameters of an ensemble of boxes to make a hand that functions as a hook. The code follows.

41 ***The next program shows how to make a hand in three steps. First: the first phalanx is chased. Second: the other two phalanges are synthesized. Third: the other fingers are made.***

```
/*
 * Program M41 FossilRecord2.
 * An explicit list of boxes is made into a hand.
 * The code is naturally designed in three steps.
 * First: a phalanx is selected for.
 * Second: a finger is made thanks to a triplication
 * of the code for one phalanx.
 * Third: the information that encodes for a finger
 * is amplified (copied) as many times as fingers
 * are suppossed to exist.
 */

package fossilrecord2;



import java.lang.reflect.Field;
import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.animation.Interpolator;
import javafx.animation.KeyFrame;
import javafx.animation.KeyValue;
import javafx.animation.Timeline;
import javafx.application.Application;
import static javafx.application.Application.launch;

import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;
import javafx.event.ActionEvent;
import javafx.geometry.Insets;

import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.PerspectiveCamera;
```

```
import javafx.scene.PointLight;
import javafx.scene.Scene;
import javafx.scene.SubScene;
import javafx.scene.control.Label;
import javafx.scene.control.Menu;
import javafx.scene.control.MenuBar;
import javafx.scene.control.MenuItem;
import javafx.scene.control.SeparatorMenuItem;
import javafx.scene.control.Slider;
import javafx.scene.input.PickResult;
import javafx.scene.layout.GridPane;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.Box;
import javafx.scene.transform.Rotate;
import static javafx.scene.transform.Rotate.X_AXIS;
import static javafx.scene.transform.Rotate.Y_AXIS;
import static javafx.scene.transform.Rotate.Z_AXIS;
import javafx.stage.Stage;
import javafx.util.Duration;



public class FossilRecord2 extends Application
{
    //Node with all elements
    XformWithPivot armGroup = new XformWithPivot();
    XformWithPivot finger4Ph2Group = new XformWithPivot();
    //Slider for zooming
    private Slider zSlider;
    //Position of graphic along the z-axis,
    //normal to the screen.
    //It is a double property because
    //it will get involved into a binding
    private final DoubleProperty zPos
            = new SimpleDoubleProperty(2000);
```

```
//For listener of mouse
private double anchorAngleX = 0;
private double anchorAngleY = 0;
private double anchorX, anchorY;
//Angles of rotation of the scene
private final DoubleProperty angleX
        = new SimpleDoubleProperty(0);
private final DoubleProperty angleY
        = new SimpleDoubleProperty(0);

PerspectiveCamera camera
        = new PerspectiveCamera(false);

//The application is instantiated
static FossilRecord2 p = new FossilRecord2();



//An array with all colors that are named in JavaFX
private static final Color JavaFXColors[]
                            = new Color[148];

//Generator of pseudo random numbers
private static final Random rd = new Random();




private static String title
        =  "A hand is made in three stages.  ";
private static String subTitle  =  "";

//Finger 0 and its phalanges:
Box ph00 = new Box(31.25,20,40);
Box ph01 = new Box(31.25,20,40);
Box ph02 = new Box(31.25,20,40);
//Finger 1 and its phalanges:
Box ph10 = new Box(31.25,20,40);
Box ph11 = new Box(31.25,20,40);
```

```
    Box ph12 = new Box(31.25,20,40);
    //Finger 2 and its phalanges:
    Box ph20 = new Box(31.25,20,40);
    Box ph21 = new Box(31.25,20,40);
    Box ph22 = new Box(31.25,20,40);
    //Finger 3 and its phalanges:
    Box ph30 = new Box(31.25,20,40);
    Box ph31 = new Box(31.25,20,40);
    Box ph32 = new Box(31.25,20,40);

    //The first phalanx is made into a group
    Group firstPhalanx = new Group();

    //Fingers are declared as groups
    Group finger0 = new Group();
    Group finger1 = new Group();
    Group finger2 = new Group();
    Group finger3 = new Group();


public void init(Stage primaryStage)
{
    //A hand with just one finger is shown at the outset
    subTitle = "First: one phalanx is built";
    primaryStage.setTitle(title + subTitle);
    hide(finger0);
        show(firstPhalanx);
    hide(finger1);
    hide(finger2);
    hide(finger3);

    //Menubar declaration and initialization
    MenuBar menuBar = new MenuBar();
    // --- Menu General
    Menu fossilsMenu = new Menu("A hand in three steps. ");

    // --Menu item declaration and initialization
    MenuItem onePhalanxMenu = new MenuItem("First phalanx is made");
     //Listener
    onePhalanxMenu.setOnAction((ActionEvent t) ->
```

```
{
    subTitle = "First: one phalanx is made";
    primaryStage.setTitle(title + subTitle);
    hide(finger0);
    show(firstPhalanx);
    hide(finger1);
    hide(finger2);
    hide(finger3);
});

//MenuItem added to menu
fossilsMenu.getItems().addAll(onePhalanxMenu);

// --Menu item declaration and initialization
MenuItem oneFingerMenu = new MenuItem("One finger is made");
 //Listener
oneFingerMenu.setOnAction((ActionEvent t) ->
{
    subTitle = "Second: one finger is made";
    primaryStage.setTitle(title + subTitle);
    show(finger0);
    hide(finger1);
    hide(finger2);
    hide(finger3);
});

//MenuItem added to menu
fossilsMenu.getItems().addAll(oneFingerMenu);

// --Menu item declaration and initialization
MenuItem allFingersMenu
        = new MenuItem("Three more fingers are made");

allFingersMenu.setOnAction((ActionEvent t) ->
{
    subTitle = "Third: three more fingers are made";
    primaryStage.setTitle(title + subTitle);
    show(finger0);
    show(finger1);
    show(finger2);
```

```
    show(finger3);
});



//MenuItems added to menu
fossilsMenu.getItems().addAll(allFingersMenu);




// --Menu item declaration and initialization
MenuItem exit = new MenuItem("Exit");
exit.setOnAction((ActionEvent t) ->
{
    //Finish everything and exit
    System.exit(0);
});

//Booking
fossilsMenu.getItems()
        .addAll(
                 new SeparatorMenuItem()
                , exit);
menuBar.getMenus()
        .addAll(fossilsMenu);


//The whole drawing  is created
//as a JavaFX space-like object
final Group parent = new Group(create3dContent());


//Initial position
parent.setTranslateX(450);
parent.setTranslateY(400);
parent.setTranslateZ(0);
Rotate xRotate;
Rotate yRotate;
//Initialization of rotation
```

```
parent.getTransforms().setAll
(
  xRotate = new Rotate(0, Rotate.X_AXIS),
  yRotate = new Rotate(0, Rotate.Y_AXIS)
);

//Graphic is attached to root
final Group root = new Group(parent);
//Window of the scene
final Scene scene = new Scene(root, 900, 900, true);
//Background
scene.setFill(Color.SKYBLUE);

//Listener of mousePressed
scene.setOnMousePressed
(
    event ->
    {
        anchorX = event.getSceneX();
        anchorY = event.getSceneY();
        anchorAngleX = angleX.get();
        anchorAngleY = angleY.get();
    }
);

root.getChildren().add(menuBar);
//Listener of mouseDragged
scene.setOnMouseDragged
(
    event ->
    {
        angleX.set(anchorAngleX
                - (anchorY - event.getSceneY()));
        angleY.set(anchorAngleY
                + anchorX - event.getSceneX());
    }
);

//Rotation of the scene gets bound to
//the angle determined by  mouseDragged
```

```
xRotate.angleProperty().bind(angleX);
yRotate.angleProperty().bind(angleY);

//Illumination
PointLight pointLight = new PointLight(Color.WHITE);
pointLight.setTranslateX(100);
pointLight.setTranslateY(200);
pointLight.setTranslateZ(-2000);
root.getChildren().add(pointLight);

//Camera
scene.setCamera(camera);
primaryStage.setScene(scene);
PerspectiveCamera subSceneCamera
        = new PerspectiveCamera(false);
scene.setCamera(subSceneCamera);

//To zoom down the graphic,
//it is translated away from  the beholder,
//along the Z-direction (normal to the screen).
//Change zPos to change initial setting.
zSlider = new Slider();
zSlider.setMax(5000);
zSlider.setMin(1);


//Bind zPos to the reading of the slider
zSlider.valueProperty().bindBidirectional(zPos);
//Execute negative zoom
parent.translateZProperty().bind(zPos);



//Other JavaFX controls, labels and containers
GridPane gridPane = new GridPane();
gridPane.setVgap(4);
gridPane.setHgap(10);
gridPane.setPadding(new Insets(5, 5, 5, 5));
//Column 0, Row   0
gridPane.add(new Label("Zoom: "), 0, 0);
```

```
gridPane.add(zSlider, 1, 0);



SubScene subScene
        = new SubScene(gridPane, 880, 200);
subScene.setLayoutX(10);
subScene.setLayoutY(800);
root.getChildren().add(subScene);

primaryStage.show();

//The sense of touch is here
parent.setOnMouseClicked((event)->
{
    //Information that can be retrieved
    PickResult res = event.getPickResult();
    //Which node
    Node node = res.getIntersectedNode();
    //Identity label of node (null if not declared)
    String nodeId  = event.getPickResult()
            .getIntersectedNode().getId();
    //which face of the node (negative if is not enabled
    int face = event.getPickResult().getIntersectedFace();
    System.out.println("res = " + res);
    System.out.println("node = " + node);
    System.out.println("nodeID = " + nodeId);
    System.out.println("face" + face);
    //Change color to red, rotate around Z_AXIS
    if (node instanceof Box)
        {
            ((Box) node).setMaterial(
                    new PhongMaterial( Color.RED));
            armGroup.rotationAxisProperty()
                    .setValue(Z_AXIS);
            int val = rd.nextInt(91);
            armGroup.rotateProperty().setValue(val);
            finger4Ph2Group.rotateProperty().setValue(val);
        }
```

```
    });


}



@Override
public void start(Stage primaryStage)  throws Exception
{
    init(primaryStage);
    primaryStage.show();
}


public static void main(String[] args)
{
    launch(args);
}


//Make the drawing
private Node create3dContent()
{
    //The scene can be rotated
    XformWithPivot sceneRoot = new XformWithPivot();
    sceneRoot.rx.setAngle(45);
    sceneRoot.ry.setAngle(110);
    getColors();
    //The constructor of hand is invoked.
    Hand  h =  new Hand();
    sceneRoot.getChildren().add(h);
    return new Group(sceneRoot);
}
```

```
//Colors with a name in JavaFX are put into an array
static private Color[] getColors()
{
    //Colors with a name in JavaFX
    final Field[] fields = Color.class.getFields();


    int N = 0;
    //Colors are put into an array
    for (final Field field : fields)
    {
        if (field.getType() == Color.class)
        {
            try
            {
                final Color color = (Color) field.get(null);
                //Array to keep colors
                JavaFXColors[N] = color;
                /*
                //The name of the color
                final String colorName = field.getName();
                System.out.println("N = " + N + "  "
                        + color.toString() + " " + colorName);
                */
                N++;
            } catch (IllegalArgumentException
                    | IllegalAccessException ex)
            {
                Logger.getLogger(FossilRecord2.class
                    .getName()).log(Level.SEVERE, null, ex);
            }
       }
    }
    return JavaFXColors;
}


//A random color is synthesized
 private static Color randomColor()
 {
```

```
    //A JavaFX named color is picked
    //but the first, Transparent, is ignored
    int n = rd.nextInt(JavaFXColors.length-1)+1;
    Color c = JavaFXColors[n];
    //Some light colors are not visualized so,
    //they might be darkened:
    //c = c.darker();
    return c;
 }


//Group is hidden
private static void  hide(Group group)
{

    Timeline hideGroup  = new Timeline();
    hideGroup.setCycleCount(1);
    Duration drtn =   Duration.millis(1);
    hideGroup.getKeyFrames().addAll(
        //Visible property is flipped from true to false
        new KeyFrame(Duration.ZERO,
            new KeyValue(group
                .visibleProperty()
                , true
                ,Interpolator.LINEAR)),
        new KeyFrame(drtn,
            new KeyValue(group
                .visibleProperty()
                , false
                ,Interpolator.LINEAR)));
     hideGroup.play();
}

//Group is shown
private static void  show(Group group)
{

    Timeline showGroup  = new Timeline();
    showGroup.setCycleCount(1);
    Duration drtn =   Duration.millis(1);
```

```
    double endTime1 = 1;
    showGroup.getKeyFrames().addAll(
    //Visible property is flipped from false to true
    new KeyFrame(Duration.ZERO,
        new KeyValue(group
            .visibleProperty()
            , false
            ,Interpolator.LINEAR)),
    new KeyFrame(drtn,
        new KeyValue(group
            .visibleProperty()
            , true
            ,Interpolator.LINEAR)));
    showGroup.play();
}


//================================================
//=============Inner class XformWithPivot===========
//================================================



//Kit of available families of transformations
private class XformWithPivot extends Group
{
    //Declaration and initialization
    public Rotate rx = new Rotate();
    { rx.setAxis(Rotate.X_AXIS); }
    public Rotate ry = new Rotate();
    { ry.setAxis(Rotate.Y_AXIS); }
    public Rotate rz = new Rotate();
    { rz.setAxis(Rotate.Z_AXIS); }
    //Rotation = angle (one coordinate)
    //+ pivot (3 coordinates) + axis.
    //This rotation is around X_AXIS
    public Rotate rwpx = new Rotate(0,0,0,0, X_AXIS);
    //This rotation is around Y_AXIS
    public Rotate rwpy = new Rotate(0,0,0,0, Y_AXIS);
    //This rotation is around Z_AXIS
```

```
    public Rotate rwpz = new Rotate(0,0,0,0, Z_AXIS);

    public XformWithPivot()
    {
        super();
        getTransforms().addAll( rx, ry, rz, rwpx, rwpy, rwpz );
    }
}//End of class XformWithPivot



  //=======================================
  //============Inner class Hand===========
  //=======================================



private class Hand extends Group
{

   //A set of boxes are organized as a hand


Hand()
{
    Color c ;
    PhongMaterial m;
    //Arm is constructed
    Box arm = new Box(100,20,400);
    c = randomColor();
    m = new PhongMaterial(c);
    arm.setMaterial(m);
    arm.setTranslateX(0);
    arm.setTranslateZ(0);
    arm.setTranslateY(20);
    //Palm is constructed
    Box palm = new Box(125,20,140);
    c = randomColor();
    m = new PhongMaterial(c);
    palm.setMaterial(m);
```

```
palm.setTranslateX(0);
palm.setTranslateZ(270);
palm.setTranslateY(20);

//Finger 0 and its phalanges:
//Phalanx 0
c = randomColor();
m = new PhongMaterial(c);
ph00.setMaterial(m);
ph00.setTranslateX(15.625*4 -15.625);
ph00.setTranslateZ(360 );
ph00.setTranslateY(20);
//Phalanx 1
c = randomColor();
m = new PhongMaterial(c);
ph01.setMaterial(m);
ph01.setTranslateX(15.625*4 -15.625);
ph01.setTranslateZ(360 + 40);
ph01.setTranslateY(20);
//Phalanx 2
c = randomColor();
m = new PhongMaterial(c);
ph02.setMaterial(m);
ph02.setTranslateX(15.625*4 -15.625);
ph02.setTranslateZ(360 + 80);
ph02.setTranslateY(20);

//Finger 1 and its phalanges:
//Phalanx 0
c = randomColor();
m = new PhongMaterial(c);
ph10.setMaterial(m);
ph10.setTranslateX(15.625*2 -15.625);
ph10.setTranslateZ(360 );
ph10.setTranslateY(20);
//Phalanx 1
c = randomColor();
m = new PhongMaterial(c);
ph11.setMaterial(m);
ph11.setTranslateX(15.625*2 -15.625);
```

```
ph11.setTranslateZ(360 + 40);
ph11.setTranslateY(20);
//Phalanx 2
c = randomColor();
m = new PhongMaterial(c);
ph12.setMaterial(m);
ph12.setTranslateX(15.625*2 -15.625);
ph12.setTranslateZ(360 + 80);
ph12.setTranslateY(20);

//Finger 2 and its phalanges:
//Phalanx 0
c = randomColor();
m = new PhongMaterial(c);
ph20.setMaterial(m);
ph20.setTranslateX(15.625*0 -15.625);
ph20.setTranslateZ(360 );
ph20.setTranslateY(20);
//Phalanx 1
c = randomColor();
m = new PhongMaterial(c);
ph21.setMaterial(m);
ph21.setTranslateX(15.625*0 -15.625);
ph21.setTranslateZ(360 + 40);
ph21.setTranslateY(20);
//Phalanx 2
c = randomColor();
m = new PhongMaterial(c);
ph22.setMaterial(m);
ph22.setTranslateX(15.625*0 -15.625);
ph22.setTranslateZ(360 + 80);
ph22.setTranslateY(20);

//Finger 3 and its phalanges:
//Phalanx 0
c = randomColor();
m = new PhongMaterial(c);
ph30.setMaterial(m);
ph30.setTranslateX(15.625*(-2) -15.625);
ph30.setTranslateZ(360 );
```

```
    ph30.setTranslateY(20);
    //Phalanx 1
    c = randomColor();
    m = new PhongMaterial(c);
    ph31.setMaterial(m);
    ph31.setTranslateX(15.625*(-2) -15.625);
    ph31.setTranslateZ(360 + 40);
    ph31.setTranslateY(20);
    //Phalanx 2
    c = randomColor();
    m = new PhongMaterial(c);
    ph32.setMaterial(m);
    ph32.setTranslateX(15.625*(-2) -15.625);
    ph32.setTranslateZ(360 + 80);
    ph32.setTranslateY(20);

    //The first phalanx is made into a group
    firstPhalanx.getChildren().add(ph00);
    //Phalanges are grouped into fingers
    finger0.getChildren().addAll(     ph01, ph02);
    finger1.getChildren().addAll(ph10, ph11, ph12);
    finger2.getChildren().addAll(ph20, ph21, ph22);
    finger3.getChildren().addAll(ph30, ph31, ph32);
    //Booking
    getChildren().addAll(arm, palm
            , firstPhalanx
            , finger0
            , finger1
            , finger2
            , finger3);
}


}
}//End of main class M41 FossilRecord2
```

42 ***Exercise.*** *Run the previous program and play with the code.*

Let us face now the problem of designing the evolutionary environment that will be responsible for the development of the code for our hooked hand. We

need to design the chromosomes, strings that will encode our proposals for a hand, and the function that will measure the fitness of each individual. Mutation and recombination will be as usual.

43 ***Design of chromosomes.***

We need to add four fingers to the hand. All fingers have the same number of phalanges and all phalanges have the same measures. So, our chromosomes will have:

1. A substring that encodes for the measures of a box that will play the role of model for every phalanx.
2. A substring for the position of the first phalanx.
3. Two substrings for the position of the second and third phalanges of the first finger.
4. The substring that encodes for the position of all other phalanges of all other fingers.

44 ***Design of the fitness function.***

In principle, the fitness function must measure the quality of the hand to be useful as a hook. This is a very abstract property whose correct definition would demand a very serious work. So, we have opted for the silly solution: since we already have the JavaFX code for the final product, we will define the fitness function as a measure of the difference between the proposed values given by a chromosome and the actual values derived from our code for the final product. For instance, we must arrive to the measures of the model box for all phalanges that shall be (31.25,20,40).

45 ***The code for stage 1 of directed evolution of the hand as a hook follows. The measures of the model box are selected for in this first stage.***

```
//  Program M45 FossilRecord3

/*  A genetic algorithm synthesizes the measures
    of a box that shall be (31.25,20,40) = target.
    The first measure is encoded
    by a subtring with 5 chars with a period included,
    the second and third measures
```

```
    by substrings 2 chars long.
    This algorithm is not good to find the target exactly
    but is very good to find a nice approximation.
*/


package fossilrecord3;
import java.util.Random;

public class FossilRecord3
{
    //Number of chromosomes = evolving strings
    private static final int nIndiv = 200 ;
    //Tolerated error: if distance to target gets less
    //than this, the algorithm stops
    private static final  double  toleratedError = 1.2;
    //Mutation rate per site  per generation
    private static final double mutationRate = 0.05;
    //Mutation rate per chromosome  per generation
    private static final double recRate = 0.1;
    //Chromosomes
    private static String Individual[ ] = new String[nIndiv];
    private static final String Individual2[ ] = new String[nIndiv];
    //Distances to target
    private static final Double Distance[] = new Double[nIndiv];
    //Fitness of each individual
    private static final Double Fitness[] = new Double[nIndiv];
    //The least distance to target
    private static double leastDistance = 1000;
    //Actual generation
    private static int generation = 0;
    //Each string encodes for three numbers
    //= three lengths of target box
    private static double d1 = 0;
    private static int int1 = 0;
    private static int int2 = 0;
    // Turn on of the random generator
    private static final Random r  = new Random();
    //Boolean flags for printing
    private static final boolean print = false;
```

```
    private static final boolean printDecode = false;
    private static final boolean printSacelFitnesses = false;
    private static final boolean printReproduction = false;
    private static final boolean printMutation = false;
    private static final boolean printRecombination = false;
    //Random integer as char
    private  static  Character  randomChar()

    // This method generates random chars
    // that represent digits
    {
     return  ( char ) ( r.nextInt( 10 ) + 48  );
    }


//Popultion is initialized
private static void Initialization( )
{
    /* We generate nIndiv individuals (strings)
    9  = 5 + 2 +2 characters long.
    Sequences are completely random */

    for(int i = 0; i <  nIndiv; i++)
    {
       //Initialization in blank.
       Individual[i]="";

       //An individual is assembled char by char
       for(int j = 0; j< 8; j++)
       {
           // We generate a random numeric char
           Character c =  randomChar();
           //System.out.println( "char = " + c);
           // Char c is converted into a string s
           String s = "" + c;
           // s is concatenated to Individual[i]
           Individual[i] = Individual[i]+s;

       }
       //A period is inserted at place 2
```

```
        Individual[i] = Individual[i].substring(0, 2)
                + "." + Individual[i].substring(2);
    }
    //Test:
    //Individual[0] = "31.252040";
}

// This method transforms each string into
//three  numbers, one double and two ints.
//The incoming string is the genotype.
//The set with three numbers is the phenotype.
 private static void decode(String s)
{
    //Double is retrieved
    String s1 = s.substring(0, 5);
    d1 = Double.parseDouble(s1);
    //First int is retrieved
    String s2 = s.substring(5,7);
    Double d2 = Double.parseDouble(s2);
    int1 = (int) Math.floor(d2);
    //Second int is retrieved
    String s3 = s.substring(7,9);
    Double d3 = Double.parseDouble(s3);
    int2 = (int) Math.floor(d3);
    if (printDecode)
    {
        System.out.println("string = " + s
        + "\nSubstring s1 = " + s1 + " double1 = " + d1
        + "\nSubstring s2 = " + s2 + " int1 = " + int1
        + "\nSubstring s3 = " + s3 + " int2 = " + int2);
    }
}


//The distance function of chromosome s
//measures the squared euclidean distance
 //of its phenotype = three numbers to
//the target(31.25,20,40).
private static Double distance(String s)
{
```

```
    //d1, int1 and int2 are calculated
    decode(s);
    //The squared distance to target is measured
    Double distance = (31.25 - d1)*(31.25 - d1)
            + (20 - int1)*(20 - int1)
            + (40 - int2)*(40 - int2);
    return distance;
}

//Fitness = rescaled distance that takes values
//in within 0(the worst) and 1(the best)
private static void calculateFitness()
{
    if (printSacelFitnesses)
        System.out.println(  "\n====Fitnesses are rescaled: " +
                   "0(the worst) and 1(the best)");
    //Min and max distances are found
    double maxD = Distance[0];
    double minD = Distance[0];
    double b;
    for(int i = 1; i < nIndiv; i++)
    {
        if ( Distance[i] > maxD ) maxD =  Distance[i];
        if ( Distance[i] < minD )
        {
            minD =  Distance[i];
            leastDistance = Distance[i];
        }
    }
    //Fitness is calculated
    if (minD == maxD) //All   equally fit
    {
      for(int i = 0; i < nIndiv; i++)
            Fitness[i] = 1.0;
    }
    else
    {
        //Distance to target determine fitness:
        //fitness[maxD] = 0;    fitness[minD] = 1;
        for(int i = 0; i < nIndiv; i++)
```

```
        {
            b = 1;
            b = b/(minD - maxD);
            double a = -b*maxD;
            Fitness[i] = a + b * Distance[i];
            //Selection is sharpened
            Fitness[i] = Math.pow(Fitness[i],7) ;
        }
    }
    //Report
    if (printSacelFitnesses)
    {
        System.out.println("Individual, distance and fitness");
        for(int i = 0; i < nIndiv; i++)
        {
            System.out.println(
                 i + "\t" + Individual[i] + "\t" +
             Distance[i] + "\t" + Fitness[i]);
        }
    }
}


//The fitness of each individual is evaluated
private static void assessFitness()
{
    //Distance to target is calculated
    for(int i = 0; i < nIndiv; i++)
    {
        Distance[i] = distance(Individual[i]);
    }
    //Fitness is an inversion of the distance,
    //The less far, the more fit
    calculateFitness();
}

//Probability of cloning is proportional to fitness.
private static void Reproduction()
{
    //Numbers of clones of each individual
```

```
int count[] = new int[nIndiv];
//The probability of reproduction
//is proportional to fitness.
//Cumulative distribution
double cumulate[] = new double[nIndiv];
cumulate[0] =   Fitness[0];
for (int i = 1; i < nIndiv; i++)
    cumulate[i]= cumulate[i-1] + Fitness[i];


if (printReproduction)
{
    System.out.println( "i \tfitness cumulate ");
    for (int i = 0; i < nIndiv; i++)
    {
        System.out.println( i + "\t" + Fitness[i] + " \t "
                + cumulate[i]);
    }
}

//Maximal value
double max = cumulate[nIndiv-1];
//Counter of chosen individuals
int W = 0;

//System.out.println("Generated numbers: ");
for (int i = 0; i < nIndiv;i++)
{
    //A random number less than max is generated
    //with a uniform distribution
    double number = max * r.nextFloat();
    if (printReproduction)
        System.out.println("Random number =  " + number);
    //Probability of cloning is made
    //in proportion to fitness.
    //Is indiv 0 to be chosen?
    if (number < cumulate[0])
    {
        if (printReproduction)
            System.out.println(
```

```
                    "chosen individual  = 0 ");
            count[0]++;
            //New population
            Individual2[W] = Individual[0];
            W++;
        }
        //Is worm l to be chosen?
        for (int j=0; j < nIndiv-1;j++)
        {

            int l = j+1;
            if ( ( cumulate[j] <= number) & (number < cumulate[l]) )
            {
                if (printReproduction)
                    System.out.println(
                        "Chosen individual = " + l);
                count[l]++;
                //New population
                Individual2[W] = Individual[l];
                W++;
            }
        }

    }

   //Output
    if (printReproduction)
    {
    System.out.println("\nEvents and their observed "
            + "and expected frequencies: ");
    int sum1 = 0;
    Double sum2 = 0.0;
    for (int i=0; i < nIndiv;i++)
    {
        System.out.println(i + "\t" + count[i] + "\t"
                +  Fitness[i]*nIndiv/cumulate[nIndiv-1]);
        sum1 = sum1 + count[i];
        sum2 = sum2 + Fitness[i]*nIndiv/cumulate[nIndiv-1];

    }
```

```
    System.out.println("Totals  " + sum1 + "\t"
                +  sum2);
    }
    //New population becomes current one
    Individual = Individual2;
}



//Individuals pass under mutation
private static void Mutation()
{
    // We consider only one type of mutation:
    // substitution of one char by another one.


    Character c;
     String s;

    for(int i = 0; i < nIndiv; i++)
    {
        if (printMutation)
            System.out.println("Incoming individ " + Individual[i]);
        String v = Individual[i];
        // w = mutated individual.
        String w = "";
        int l = v.length();
        //Mutation char by char according to mutationRate
        for(int k = 0; k < l; k++)
        {
            //Touch not the period
            if ( k == 2) c = Individual[i].charAt(2);
            else
            {
                // Change digit k?
                double  p = r.nextDouble();
                if (p < mutationRate)    c =  randomChar();
                else
                    { //no mutation
                        c = Individual[i].charAt(k);
```

```
                    }


            }
            // Char c is converted into a string s
            s = c.toString();
            // s is concatenated to w
            w = w+s;
        }

        Individual[i] = w;
        if (printMutation)
        System.out.println("Mutated          " + Individual[i]);

    }
}

//Individual l is recombined to another
//that is chosen at random
private static String recInd(int l)
{
    int length = Individual[l].length();
    //Random site for recombination
    int site = r.nextInt(length);
    //Take a part from l
    String s = Individual[l].substring(0, site);
    //Partner at random
    int k = r.nextInt(nIndiv);
    //Another part from k
    String t = Individual[k].substring(site);
    //Return concatenation
    return s+t;
}

//ALl individuals pass under recombination
//according to recRate
private static void Recombination()
{
    for(int i = 0; i < nIndiv; i++)
    {
```

```
        if (printRecombination)
            System.out.println("Incoming individ " + Individual[i]);
        // Recombine?
        double  p = r.nextDouble();
        if (p < recRate)
            Individual[i] = recInd(i);
        if (printRecombination)
            System.out.println("Recombinant      " + Individual[i]);
    }
}

//Report of best exponents
private static void report( )
{


    for(int i = 0; i < nIndiv; i++)
    {
       if (Fitness[i] > 0.999999)
            System.out.println("Gen = " + generation +
                "  Chromosome =  " +   Individual[i]
                + " Distance = " + Distance[i]);
    }
}

//One generation of evolution
private static void oneGeneration()
{
    //Fitness of each indivdual is assessed
    assessFitness();
    report();
    //The probability of cloning is proportional to fitness
    Reproduction();
    //The new population is subjected to mutation
    Mutation();
    //The new population is subjected to recombination
    Recombination();
}

//Genetic algorithm to fit a target string
```

```
//that represents the measures of a box
public static void main(String[] args)
{
    System.out.println("ddd");
    Initialization( ) ;
    generation = 0;
    while ( leastDistance  > toleratedError)
    {

       oneGeneration();
       generation++;
    }
}
} //End of Program M45 FossilRecord3
```

46 ***Exercise.*** *Run the previous program and play with the code.*

47 ***Exercise.*** *Complete the previous program to include the evolutionary chasing of the position of the first phalanx of the first finger and the other two stages until the total definition of a hand: the other phalanges of the first finger, and the other fingers.* *Answer*

48 ***Let us plug the genetic algorithm that synthesizes the parameters of the hand into a visual show .***

```
/*
 * Program M48 FossilRecord5.
 *
 * Evolution is given the task of fitting
 * the pattern of a hand that is composed of boxes.
 * The task is accomplished in 4 stages, each one
 * producing an approximation to key parameters:
   -The measures of the model box
   -Coordinates of the first phalanx
   -Z-coordinates of phalanges 2 and 3 of first finger
   -X-coordinates of the other 3 fingers.
 */

package fossilrecord5;
```

```
import java.lang.reflect.Field;
import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.animation.ScaleTransition;
import javafx.animation.SequentialTransition;
import javafx.animation.TranslateTransition;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;
import javafx.geometry.Insets;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.PerspectiveCamera;
import javafx.scene.PointLight;
import javafx.scene.Scene;
import javafx.scene.SubScene;
import javafx.scene.control.Label;
import javafx.scene.control.Slider;
import javafx.scene.input.PickResult;
import javafx.scene.layout.GridPane;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.Box;
import javafx.scene.transform.Rotate;
import static javafx.scene.transform.Rotate.X_AXIS;
import static javafx.scene.transform.Rotate.Y_AXIS;
import static javafx.scene.transform.Rotate.Z_AXIS;
import javafx.stage.Stage;
import javafx.util.Duration;



public class FossilRecord5 extends Application
{


    //===== Evolutionary module==============
```

```
//Number of chromosomes = evolving strings
private static final int nIndiv = 200 ;
//Tolerated error: if distance to target gets less
//than this, the algorithm stops
private static final  double  toleratedError = 5;
//Mutation rate per site  per generation
private static final double mutationRate = 0.05;
//Recombination rate per chromosome  per generation
private static final double recRate = 0.1;
//Chromosomes
private static String Individual[ ]
        = new String[nIndiv];
private static final String Individual2[ ]
        = new String[nIndiv];
//Distances to target
private static final Double Distance[]
        = new Double[nIndiv];
//Fitness of each individual
private static final Double Fitness[]
        = new Double[nIndiv];
//The least distance to target
private static double leastDistance;
//Index of best individual, the closest to target.
private static int bestIndividual;
//Last generation
private static int lastGeneration = 0;
//Each string encodes for various numbers
private static double d1 = 0;
private static int int1 = 0;
private static int int2 = 0;
private static double d2 = 0;
private static double d3 = 0;
//Number of chars per string
private static int numberOfChars;

// Turn on of the random generator
private static final Random r  = new Random();
//Place of insertion of the decimal period
//if negative, there is no period
```

```
private static  int InsertionPeriod  = 2;

//Boolean flags for printing
private static final boolean printInitialization3 = false;
private static final boolean printDecode = false;
private static final boolean printSacelFitnesses = false;
private static final boolean printReproduction = false;
private static final boolean printMutation = false;
private static final boolean printMutation3 = false;
private static final boolean printRecombination = false;
private static final boolean printMoveBoxTo = false;
private static final boolean printMoreReport = false;

//Tag
private static String landMark = "";
//Costo of algorithm
//= number of generations * population size
private static int costOfAlgorithm = 0;

//========DRAWING MODULE==========

//Animation
private static final SequentialTransition animation
        = new SequentialTransition();
//Node with all elements
XformWithPivot armGroup = new XformWithPivot();
//Finger 4
XformWithPivot finger4Ph2Group = new XformWithPivot();
//Slider for zooming
private Slider zSlider;
//Position of graphic along the z-axis,
//normal to the screen.
//It is a double property because
//it will get involved into a binding
private final DoubleProperty zPos
        = new SimpleDoubleProperty(2000);



//For listener of mouse
```

```
private double anchorAngleX = 0;
private double anchorAngleY = 0;
private double anchorX, anchorY;
//Angles of rotation of the scene
private final DoubleProperty angleX
        = new SimpleDoubleProperty(0);
private final DoubleProperty angleY
        = new SimpleDoubleProperty(0);

PerspectiveCamera camera
        = new PerspectiveCamera(false);



//An array with all colors that are named in JavaFX
private static final Color JavaFXColors[]
                            = new Color[148];



private static final String title
        =  "Evolution fits the pattern of a hand";


//Finger 0 and its phalanges:
private static final Box ph00 = new Box(1,1,1);
private static final Box ph01 = new Box(1,1,1);
private static final Box ph02 = new Box(1,1,1);
//Finger 1 and its phalanges:
private static final Box ph10 = new Box(1,1,1);
private static final Box ph11 = new Box(1,1,1);
private static final Box ph12 = new Box(1,1,1);
//Finger 2 and its phalanges:
private static final Box ph20 = new Box(1,1,1);
private static final Box ph21 = new Box(1,1,1);
private static final Box ph22 = new Box(1,1,1);
//Finger 3 and its phalanges:
private static final Box ph30 = new Box(1,1,1);
private static final Box ph31 = new Box(1,1,1);
```

```
    private static final Box ph32 = new Box(1,1,1);

    //Position of phalanx 00
    private static double ph00X = 0;
    private static double ph00Z = 0;
    private static double ph00Y = 0;
    //Phalanges 1 and 2
    private static double ph01Z = 200;
    private static double ph02Z = 500;


    //The first phalanx is made into a group
    Group firstPhalanx = new Group();

    //Fingers are declared as groups
    Group finger0 = new Group();
    Group finger1 = new Group();
    Group finger2 = new Group();
    Group finger3 = new Group();

public void init(Stage primaryStage)
{
    primaryStage.setTitle(title);

    //The whole drawing  is created
    //as a JavaFX space-like object
    final Group parent = new Group(create3dContent());


    //Initial position
    parent.setTranslateX(450);
    parent.setTranslateY(400);
    parent.setTranslateZ(0);
    Rotate xRotate;
    Rotate yRotate;
    //Initialization of rotation
    parent.getTransforms().setAll
    (
      xRotate = new Rotate(0, Rotate.X_AXIS),
      yRotate = new Rotate(0, Rotate.Y_AXIS)
```

```
);

//Graphic is attached to root
final Group root = new Group(parent);
//Window of the scene
final Scene scene = new Scene(root, 900, 900, true);
//Background
scene.setFill(Color.SKYBLUE);

//Listener of mousePressed
scene.setOnMousePressed
(
    event ->
    {
        anchorX = event.getSceneX();
        anchorY = event.getSceneY();
        anchorAngleX = angleX.get();
        anchorAngleY = angleY.get();
    }
);


//Listener of mouseDragged
scene.setOnMouseDragged
(
    event ->
    {
        angleX.set(anchorAngleX
                - (anchorY - event.getSceneY()));
        angleY.set(anchorAngleY
                + anchorX - event.getSceneX());
    }
);

//Rotation of the scene gets bound to
//the angle determined by  mouseDragged
xRotate.angleProperty().bind(angleX);
yRotate.angleProperty().bind(angleY);

//Illumination
```

```
PointLight pointLight = new PointLight(Color.WHITE);
pointLight.setTranslateX(100);
pointLight.setTranslateY(200);
pointLight.setTranslateZ(-2000);
root.getChildren().add(pointLight);

//Camera
scene.setCamera(camera);
primaryStage.setScene(scene);
PerspectiveCamera subSceneCamera
        = new PerspectiveCamera(false);
scene.setCamera(subSceneCamera);

//To zoom down the graphic,
//it is translated away from  the beholder,
//along the Z-direction (normal to the screen).
//Change zPos to change initial setting.
zSlider = new Slider();
zSlider.setMax(5000);
zSlider.setMin(1);


//Bind zPos to the reading of the slider
zSlider.valueProperty().bindBidirectional(zPos);
//Execute negative zoom
parent.translateZProperty().bind(zPos);



//Other JavaFX controls, labels and containers
GridPane gridPane = new GridPane();
gridPane.setVgap(4);
gridPane.setHgap(10);
gridPane.setPadding(new Insets(5, 5, 5, 5));
//Column 0, Row   0
gridPane.add(new Label("Zoom: "), 0, 0);
gridPane.add(zSlider, 1, 0);
```

```
SubScene subScene
        = new SubScene(gridPane, 880, 200);
subScene.setLayoutX(10);
subScene.setLayoutY(800);
root.getChildren().add(subScene);

primaryStage.show();

//The sense of touch is here
parent.setOnMouseClicked((event)->
{
    //Information that can be retrieved
    PickResult res = event.getPickResult();
    //Which node
    Node node = res.getIntersectedNode();
    //Identity label of node (null if not declared)
    String nodeId  = event.getPickResult()
            .getIntersectedNode().getId();
    //which face of the node (negative if is not enabled
    int face = event.getPickResult().getIntersectedFace();
    System.out.println("res = " + res);
    System.out.println("node = " + node);
    System.out.println("nodeID = " + nodeId);
    System.out.println("face" + face);
    //Change color to red, rotate around Z_AXIS
    if (node instanceof Box)
        {
            ((Box) node).setMaterial(
                    new PhongMaterial( Color.RED));
            armGroup.rotationAxisProperty()
                    .setValue(Z_AXIS);
            int val = r.nextInt(91);
            armGroup.rotateProperty().setValue(val);
            finger4Ph2Group.rotateProperty().setValue(val);
        }
```

```
    });


}



@Override
public void start(Stage primaryStage)  throws Exception
{
    init(primaryStage);
    primaryStage.show();
}


public static void main(String[] args)
{
    launch(args);
}

//=====================================
//========= Evolutionary module==========
//=====================================

//Random integer as char
private  static  Character  randomChar()

// This method generates random chars
// that represent digits
{
 return  ( char ) ( r.nextInt( 10 ) + 48  );
}


//Population is initialized.
//Each chromosome must fit the pattern given by
//"31.252040" which encodes for
//three numbers, say (31.25, 20, 40)
private static void Initialization( )
```

```
{
    /* We generate nIndiv individuals (strings)
    9  = 5 + 2 +2 characters long.
    Sequences are completely random */

    for(int i = 0; i <  nIndiv; i++)
    {
        //Initialization in blank.
        Individual[i]="";

        //An individual is assembled char by char
        for(int j = 0; j< numberOfChars; j++)
        {
            // We generate a random numeric char
            Character c =  randomChar();
            //System.out.println( "char = " + c);
            // Char c is converted into a string s
            String s = "" + c;
            // s is concatenated to Individual[i]
            Individual[i] = Individual[i]+s;

        }
        if (InsertionPeriod > 0)
        //A period is inserted at place InsertionPeriod
        Individual[i]
            = Individual[i].substring(0, InsertionPeriod)
            + "." + Individual[i].substring(InsertionPeriod);
    }
    //Test:
    //Individual[0] = "31.252040";
}

//Population is initialized.
//Each chromosome must fit the pattern given by three
//numbers of type double with their signs
//(+15.625, -15.625, -46.875)
private static void Initialization3( )
{
```

```
    for(int i = 0; i <  nIndiv; i++)
    {
        //Initialization in blank.
        Individual[i]="";
        String s;
        //Three numbers of type double are composed
        for(int j = 0; j < 3; j++)
        {
            //Sign is invented
            int l = r.nextInt(2);
            if ( l == 0) s = "+";
            else  s = "-";


            //A numeric substring is assembled char by char
            for(int k = 0; k <  5; k++)
            {
                // We generate a random numeric char
                Character c =  randomChar();
                //System.out.println( "char = " + c);
                // Char c is concatenated to string s
                s = s + c;

            }

            //A period is inserted at place 3,
            s  = s.substring(0, 3)
                     + "." + s.substring(3);
            Individual[i] = Individual[i] + s;
        }
        if (printInitialization3)
            System.out.println(Individual[i]);
    }
}


// This method transforms each string into
//three  numbers, one double and two ints.
//The incoming string is the genotype.
//The set with three numbers is the phenotype.
```

```
//Case: measures of the model box
 private static void decode1(String s)
{
    //Double is retrieved
    String s1 = s.substring(0, 5);
    d1 = Double.parseDouble(s1);
    //First int is retrieved
    String s2 = s.substring(5,7);
    d2 = Double.parseDouble(s2);
    int1 = (int) Math.floor(d2);
    //Second int is retrieved
    String s3 = s.substring(7,9);
    d3 = Double.parseDouble(s3);
    int2 = (int) Math.floor(d3);
    if (printDecode)
    {
        System.out.println("string = " + s
        + "\nSubstring s1 = " + s1 + " double1 = " + d1
        + "\nSubstring s2 = " + s2 + " int1 = " + int1
        + "\nSubstring s3 = " + s3 + " int2 = " + int2);
    }
}

//This method transforms each string into
//three  numbers.
//Case: Position of the model box
 private static void decode1c(String s)
{
    //Double is retrieved
    String s1 = s.substring(0, 6);
    d1 = Double.parseDouble(s1);
    //First int is retrieved
    String s2 = s.substring(6,9);
    d2 = Double.parseDouble(s2);
    int1 = (int) Math.floor(d2);
    //Second int is retrieved
    String s3 = s.substring(9,11);
    d3 = Double.parseDouble(s3);
    int2 = (int) Math.floor(d3);
    if (printDecode)
```

```
    {
        System.out.println("string = " + s
        + "\nSubstring s1 = " + s1 + " double1 = " + d1
        + "\nSubstring s2 = " + s2 + " int1 = " + int1
        + "\nSubstring s3 = " + s3 + " int2 = " + int2);
    }
}

//This method transforms each string into
//two  numbers.
//Case: Z-position of phalanges 2 and 3 of first finger
 private static void decode2(String s)
{

    //First int is retrieved
    String s2 = s.substring(0,3);
    d2 = Double.parseDouble(s2);
    int1 = (int) Math.floor(d2);
    //Second int is retrieved
    String s3 = s.substring(3,6);
    d3 = Double.parseDouble(s3);
    int2 = (int) Math.floor(d3);
    if (printDecode)
    {
        System.out.println("string = " + s
        + "\nSubstring s2 = " + s2 + " int1 = " + int1
        + "\nSubstring s3 = " + s3 + " int2 = " + int2);
    }
}

//This method transforms each string into
//three  numbers of type double.
//Case: Z-position of fingers 2, 3 and 4
 private static void decode3(String s)
{
    //First double is retrieved
    String s1 = s.substring(0,7);
    d1 = Double.parseDouble(s1);
    //Second double is retrieved
    String s2 = s.substring(7,14);
```

```
    d2 = Double.parseDouble(s2);
    //Third double is retrieved
    String s3 = s.substring(14,21);
    d3 = Double.parseDouble(s3);

    if (printDecode)
    {
        System.out.println("string = " + s
                + "\nSubstring s1 = " + s1 + " d1 = " + d1
        + "\nSubstring s2 = " + s2 + " d2 = " + d2
        + "\nSubstring s3 = " + s3 + " d3 = " + d3);
    }
}


//The distance function upon chromosome s
//measures the squared euclidean distance
 //of its phenotype (three numbers) to
//the target(31.25,20,40).
private static Double distance1(String s)
{
    //d1, int1 and int2 are calculated
    decode1(s);
    //The squared distance to target is measured
    Double distance = (31.25 - d1)*(31.25 - d1)
            + (20 - int1)*(20 - int1)
            + (40 - int2)*(40 - int2);
    return distance;
}

//The distance function upon chromosome s
//measures the squared euclidean distance
//of its phenotype  (three numbers) to
//the target(46.875, 360, 20).
private static Double distance1c(String s)
{
    //d1, int1 and int2 are calculated
    decode1c(s);
    //The squared distance to target is measured
    Double distance = (46.875 - d1)*(46.875 - d1)
```

```
            + (360 - int1)*(360 - int1)
            + (20 - int2)*(20 - int2);
    return distance;
}

//The distance function upon chromosome s
//measures the squared euclidean distance
//of its phenotype (three numbers) to
//the target(400, 440).
private static Double distance2(String s)
{
    //int1 and int2 are calculated
    decode2(s);
    //The squared distance to target is measured
    Double distance = 0.0 +
            (400 - int1)*(400 - int1)
            + (440 - int2)*(440 - int2);
    return distance;
}


//The distance function upon chromosome s
//measures the squared euclidean distance
 //of its phenotype (three numbers) to
//the target(+15.625, -15.625, -46.875).
private static Double distance3(String s)
{
    //d1, int1 and int2 are calculated
    decode3(s);
    //The squared distance to target is measured
    Double distance = (15.625 - d1)*(15.625 - d1) +
            (-15.625 - d2)*(-15.625 - d2)
            + (-46.875 - d3)*(-46.875 - d3);
    return distance;
}


//Fitness = rescaled distance that takes values
//in within 0(the worst) and 1(the best)
private static void calculateFitness()
```

```
{
    if (printSacelFitnesses)
        System.out.println(
            "\n====Fitnesses are rescaled: " +
            "0(the worst) and 1(the best)");
    //Min and max distances are found
    double maxD = Distance[0];
    double minD = Distance[0];
    double b;
    for(int i = 1; i < nIndiv; i++)
    {
        if ( Distance[i] > maxD ) maxD =  Distance[i];
        if ( Distance[i] < minD )
        {
            minD =  Distance[i];
            leastDistance = Distance[i];
            bestIndividual = i;
        }
    }
    //Fitness is calculated
    if (minD == maxD) //All   equally fit
    {
      for(int i = 0; i < nIndiv; i++)
            Fitness[i] = 1.0;
    }
    else
    {
        //Distance to target determine fitness:
        //fitness[maxD] = 0;    fitness[minD] = 1;
        for(int i = 0; i < nIndiv; i++)
        {
            b = 1;
            b = b/(minD - maxD);
            double a = -b*maxD;
            Fitness[i] = a + b * Distance[i];
            //Selection is sharpened
            Fitness[i] = Math.pow(Fitness[i],7) ;
        }
    }
    //Report
```

```
    if (printSacelFitnesses)
    {
        System.out.println("Individual, "
                + "distance and fitness");
        for(int i = 0; i < nIndiv; i++)
        {
            System.out.println(
                 i + "\t" + Individual[i] + "\t" +
             Distance[i] + "\t" + Fitness[i]);
        }
    }
}


//The fitness of each individual is evaluated.
//Case: Measures of model box
private static void assessFitness1()
{
    //Distance to target is calculated
    for(int i = 0; i < nIndiv; i++)
    {
        Distance[i] = distance1(Individual[i]);
    }
    //Fitness is an inversion of the distance,
    //The less far, the more fit
    calculateFitness();
}

//The fitness of each individual is evaluated.
//Case: Position of model box
private static void assessFitness1c()
{
    //Distance to target is calculated
    for(int i = 0; i < nIndiv; i++)
    {
        Distance[i] = distance1c(Individual[i]);
    }
    //Fitness is an inversion of the distance,
    //The less far, the more fit
    calculateFitness();
```

```
}

//The fitness of each individual is evaluated.
//Case: Z-Position of phalanges 2 and 3
private static void assessFitness2()
{
    //Distance to target is calculated
    for(int i = 0; i < nIndiv; i++)
    {
        Distance[i] = distance2(Individual[i]);
    }
    //Fitness is an inversion of the distance,
    //The less far, the more fit
    calculateFitness();
}

//The fitness of each individual is evaluated.
//Case: X-Position of fingers 2, 3 and 4
private static void assessFitness3()
{
    //Distance to target is calculated
    for(int i = 0; i < nIndiv; i++)
    {
        Distance[i] = distance3(Individual[i]);
    }
    //Fitness is an inversion of the distance,
    //The less far, the more fit
    calculateFitness();
}

//Probability of cloning is proportional to fitness.
private static void Reproduction()
{
    //Numbers of clones of each individual
    int count[] = new int[nIndiv];
    //The probability of reproduction
    //is proportional to fitness.
    //Cumulative distribution
    double cumulate[] = new double[nIndiv];
    cumulate[0] =   Fitness[0];
```

```
for (int i = 1; i < nIndiv; i++)
    cumulate[i]= cumulate[i-1] + Fitness[i];


if (printReproduction)
{
    System.out.println( "i \tfitness cumulate ");
    for (int i = 0; i < nIndiv; i++)
    {
        System.out.println( i + "\t" + Fitness[i]
                + " \t "  + cumulate[i]);
    }
}

//Maximal value
double max = cumulate[nIndiv-1];
//Counter of chosen individuals
int W = 0;

//System.out.println("Generated numbers: ");
for (int i = 0; i < nIndiv;i++)
{
    //A random number less than max is generated
    //with a uniform distribution
    double number = max * r.nextFloat();
    if (printReproduction)
        System.out.println("Random number =  " + number);
    //Probability of cloning is made
    //in proportion to fitness.
    //Is indiv 0 to be chosen?
    if (number < cumulate[0])
    {
        if (printReproduction)
            System.out.println(
                    "chosen individual  = 0 ");
        count[0]++;
        //New population
        Individual2[W] = Individual[0];
        W++;
    }
```

```
        //Is worm l to be chosen?
        for (int j=0; j < nIndiv-1;j++)
        {

            int l = j+1;
            if ( ( cumulate[j] <= number)
                    & (number < cumulate[l]) )
            {
                if (printReproduction)
                    System.out.println(
                        "Chosen individual = " + l);
                count[l]++;
                //New population
                Individual2[W] = Individual[l];
                W++;
            }
        }

    }

   //Output
    if (printReproduction)
    {
    System.out.println("\nEvents and their observed "
            + "and expected frequencies: ");
    int sum1 = 0;
    Double sum2 = 0.0;
    for (int i=0; i < nIndiv;i++)
    {
        System.out.println(i + "\t" + count[i] + "\t"
                +  Fitness[i]*nIndiv/cumulate[nIndiv-1]);
        sum1 = sum1 + count[i];
        sum2 = sum2 + Fitness[i]*nIndiv/cumulate[nIndiv-1];

    }
    System.out.println("Totals  " + sum1 + "\t"
                +  sum2);
    }
    //New population becomes the current one
    Individual = Individual2;
```

```
}


//Individuals pass under mutation
//Cases: one, one-continued and two
private static void Mutation()
{
    // We consider only one type of mutation:
    // substitution of one char by another one.


    Character c;
     String s;

    for(int i = 0; i < nIndiv; i++)
    {
        if (printMutation)
            System.out.println("Incoming individ "
                    + Individual[i]);
        String v = Individual[i];
        // w = mutated individual.
        String w = "";
        int l = v.length();
        //Mutation char by char according to mutationRate
        for(int k = 0; k < l; k++)
        {
            //Touch not the period
            if ( k == InsertionPeriod) c
                    = Individual[i].charAt(2);
            else
            {
                // Change digit k?
                double  p = r.nextDouble();
                if (p < mutationRate)    c =  randomChar();
                else
                    { //no mutation
                        c = Individual[i].charAt(k);
                    }
```

```
            }
            // Char c is converted into a string s
            s = c.toString();
            // s is concatenated to w
            w = w+s;
        }

        Individual[i] = w;
        if (printMutation)
        System.out.println("Mutated          "
                + Individual[i]);

    }
}


//Mutation of a string that represents a double
//as -15.625
private static String mutateDouble(String d)
{

    String s = "";

    char c = d.charAt(0);
    String cs = ""+ c;
    double  p = r.nextDouble();
    // Change sign?
    if (p < mutationRate)
    {
        if (cs.contentEquals("-")) cs = "" + '+';
        else cs = "" + '-';

    }
    s = s + cs;
    //Mutation char by char according to mutationRate
    for(int k = 1; k < 7; k++)
    {
        //Touch not the period
        if ( k == 3) c = '.';
```

```
        else
        {
            // Change digit k?
            p = r.nextDouble();
            if (p < mutationRate)    c =  randomChar();
            else
                { //no mutation
                    c = d.charAt(k);
                }


        }

        s = s +c;
    }
    if (printMutation3)
    {
        System.out.println("In =  " + d);
        System.out.println("Out = " + s);
    }
    return s;
}


//Individuals pass under mutation
//Cases: position of fingers 2, 3 and 4.
private static void Mutation3()
{
    // We consider only one type of mutation:
    // substitution of one char by another of the same type:
    //Number by number, sign by sign, period by period.




    for(int i = 0; i < nIndiv; i++)
    {
        if (printMutation3)
        {
```

```
            System.out.println("Incoming individ "
                    + Individual[i]);
        }
        String v = Individual[i];
        //Three numbers suffer mutation
        // d = substring of number one
        String d = Individual[i].substring(0,7);
        //Mutant
        String w = mutateDouble(d);
        String mutant = "";
        //Concatenation
        mutant = mutant + w;
        //Number 2
        d = Individual[i].substring(7,14);
        w = mutateDouble(d);
        mutant = mutant + w;
        //Number 3
        d = Individual[i].substring(14,21);
        w = mutateDouble(d);
        mutant = mutant + w;

        Individual[i] = mutant;
        if (printMutation3)
        {
            System.out.println("Mutated          "
                    + Individual[i]);
        }

    }
}

//Individual l is recombined to another
//that is chosen at random
private static String recInd(int l)
{
    int length = Individual[l].length();
    //Random site for recombination
    int site = r.nextInt(length);
    //Take a part from l
    String s = Individual[l].substring(0, site);
```

```
    //Partner at random
    int k = r.nextInt(nIndiv);
    //Another part from k
    String t = Individual[k].substring(site);
    //Return concatenation
    return s+t;
}

//ALl individuals pass under recombination
//according to recRate
private static void Recombination()
{
    for(int i = 0; i < nIndiv; i++)
    {
        if (printRecombination)
            System.out.println("Incoming individ "
                    + Individual[i]);
        // Recombine?
        double  p = r.nextDouble();
        if (p < recRate)
            Individual[i] = recInd(i);
        if (printRecombination)
            System.out.println("Recombinant      "
                    + Individual[i]);
    }
}

//Report of best exponents
private static void report( )
{
    /*
    for(int i = 0; i < nIndiv; i++)
    {
       //if (Fitness[i] > 0.999999)
        if   (Distance[i] < 2)
            System.out.println("Gen = " + lastGeneration +
                "  Chromosome =  " +   Individual[i]
                + " Distance = " + Distance[i]);
    }
    */
```

```
System.out.println("Gen = " + lastGeneration
        + " Best Chromosome =  "
        +   Individual[bestIndividual]
        + " Distance = " + leastDistance);

double boxX ;
int boxY ;
int boxZ;

//Measures of the model box
if (landMark.equals("stageOne"))
{
    decode1(Individual[bestIndividual]);
    boxX = d1;
    boxY = int1;
    boxZ = int2;
    //Boxes get their   measures
    setMeasures(ph00,  boxX,  boxY,  boxZ);
    setMeasures(ph01,  boxX,  boxY,  boxZ);
    setMeasures(ph02,  boxX,  boxY,  boxZ);
    setMeasures(ph10,  boxX,  boxY,  boxZ);
    setMeasures(ph11,  boxX,  boxY,  boxZ);
    setMeasures(ph12,  boxX,  boxY,  boxZ);
    setMeasures(ph20,  boxX,  boxY,  boxZ);
    setMeasures(ph21,  boxX,  boxY,  boxZ);
    setMeasures(ph22,  boxX,  boxY,  boxZ);
    setMeasures(ph30,  boxX,  boxY,  boxZ);
    setMeasures(ph31,  boxX,  boxY,  boxZ);
    setMeasures(ph32,  boxX,  boxY,  boxZ);
}




if (landMark.equals("stageOneContinued"))
{
    decode1c(Individual[bestIndividual]);

    ph00X = d1;
```

```
        ph00Z = (double) int1;
        ph00Y = (double) int2;
        if (printMoreReport)
        {
            System.out.println(
                  " ph00X = " + ph00X
                + " ph00Y = " + ph00Y
                + " ph00Z = " + ph00Z);
        }
        MoveBoxTo(ph00,  ph00X, ph00Y, ph00Z) ;

    }



    //Position of phalanx 01 an 02
    if (landMark.equals("stageTwo"))
    {
        decode2(Individual[bestIndividual]);

        ph01Z = (double) int1;
        ph02Z = (double) int2;
        if (printMoreReport)
        {
            System.out.println(
                  " ph00X = " + ph00X
                + " ph00Y = " + ph00Y
                + " ph01Z = " + ph00Z);
        }
        MoveBoxTo(ph01,  ph00X, ph00Y, ph01Z) ;
        MoveBoxTo(ph02,  ph00X, ph00Y, ph02Z) ;

    }

    //X-Position of fingers 2, 3 and 4
    //Target = (15.625, -15.625, -46.875)
    if (landMark.equals("stageThree"))
    {
        decode3(Individual[bestIndividual]);
        double ph10X = d1;
```

```
        double ph20X = d2;
        double ph30X = d3;
        if (printMoreReport)
        {
            System.out.println(
                  " ph10X = " + ph10X
                + " ph20X = " + ph20X
                + " ph30X = " + ph30X);
        }

        MoveBoxTo(ph10,  ph10X, ph00Y, ph00Z) ;
        MoveBoxTo(ph11,  ph10X, ph00Y, ph01Z) ;
        MoveBoxTo(ph12,  ph10X, ph00Y, ph02Z) ;
        MoveBoxTo(ph20,  ph20X, ph00Y, ph00Z) ;
        MoveBoxTo(ph21,  ph20X, ph00Y, ph01Z) ;
        MoveBoxTo(ph22,  ph20X, ph00Y, ph02Z) ;
        MoveBoxTo(ph30,  ph30X, ph00Y, ph00Z) ;
        MoveBoxTo(ph31,  ph30X, ph00Y, ph01Z) ;
        MoveBoxTo(ph32,  ph30X, ph00Y, ph02Z) ;
    }


}

//Measures are given to box
private static void  setMeasures( Box box,
                        double X, int Y, int Z)
{
    Duration drtn =   Duration.millis(5);
    ScaleTransition scaleTransition
            = new ScaleTransition(drtn, box);

    scaleTransition.setToX(X);
    scaleTransition.setToY(Y);
    scaleTransition.setToZ(Z);
    animation.getChildren().add(scaleTransition);
}


//Box is translated to (X,Y,Z)
```

```
private static void  MoveBoxTo(Box box,
                        double X, double Y, double Z)
{
    if (printMoveBoxTo)
    {
        System.out.println(
              " X = " + X
            + " Y = " + Y
            + " Z = " + Z);
    }

    Duration drtn =   Duration.millis(5);
    TranslateTransition translateTransition =
             new TranslateTransition(drtn, box);
    //Jump is executed
    translateTransition.setToX(X);
    translateTransition.setToY(Y);
    translateTransition.setToZ(Z);
    //Number of repetitions
    translateTransition.setCycleCount(1);
    //Booking
    animation.getChildren().add(translateTransition);

}



//One generation of evolution
private static void oneGeneration()
{
    report();
    //The probability of cloning is proportional to fitness
    Reproduction();
    //The new population is subjected to mutation
    Mutation();
    //The new population is subjected to recombination
    Recombination();
}

//The measures of the model box are chasen:
```

```
//(31.25,20,40)
/*
The first measure (x) is encoded
by a subtring with 5 chars with a period included,
the second and third measures (x and y)
by substrings 2 chars long.
*/
private static void  stageOne()
{
    System.out.println("\nSTAGE ONE: Measures of model box");
    System.out.println("Target = (31.25,20,40)");
    lastGeneration = 0;
    numberOfChars = 8;//The dot is inserted a posteriori
    leastDistance = 1000;
    InsertionPeriod = 2;
    Initialization( );
    landMark = "stageOne";
    //Evolutionary loop
    while ( leastDistance  > toleratedError)
    {
        //Fitness of each individual is assessed
        assessFitness1();
        oneGeneration();
        lastGeneration++;
    }
}


//The position of the first phalanx is fitted.
//It is the equivalent of:
/*
    ph00.setTranslateX(15.625*4 -15.625 = 46.875);
    ph00.setTranslateZ(360 );
    ph00.setTranslateY(20);
*/
//TranslateX is encoded in 6 chars.
//TranslateY is encoded in 3 chars.
//TranslateZ is encoded in 2 chars.
//Total 11 chars
```

```
private static void  stageOneContinued()
{
    System.out.println("\nSTAGE ONE CONTINUED:"
            + " position of first phalanx");
    System.out.println("Target = (46.875, 360, 20)");
    lastGeneration = 0;
    numberOfChars = 10; //The period is inserted a posteriori
    leastDistance = 10000;
    InsertionPeriod = 2;
    Initialization( );
    landMark = "stageOneContinued";
    //Evolutionary loop
    while ( leastDistance  > toleratedError)
    {
        //Fitness of each individual is assessed
        assessFitness1c();
        oneGeneration();
        lastGeneration++;
    }
}

//Second and third phalanges of first finger are chasen.
//Measures are all the same.
//TranslateX and TranslateY coincide with those
//of the first phalanx.
//TranslateZ: 400 for phalanx2, 440 for phalanx3
private static void  stageTwo()
{
    System.out.println("\nSTAGE TWO: Z-position of second"
            + " and third pahalanges");
    System.out.println("Target = (400, 440)");
    lastGeneration = 0;
    numberOfChars = 6; //The period is inserted a posteriori
    leastDistance = 10000;
    InsertionPeriod = -1; //No decimal period
    Initialization( );
    landMark = "stageTwo";
    //Evolutionary loop
    while ( leastDistance  > toleratedError)
    {
```

```
        //Fitness of each individual is assessed
        assessFitness2();
        oneGeneration();
        lastGeneration++;
    }
}


//Second,  third and fourth fingers.
//An X-translation of first finger is
//enough to create another one.
//For the second finger setTranslateX(15.625*2 -15.625);
//For the third finger setTranslateX(15.625*0 -15.625)
//For thefourth finger setTranslateX(15.625*(-2) -15.625)
private static void  stageThree()
{
    System.out.println("\nSTAGE THREE: "
            + "X-position of other 3 fingers");
    System.out.println("Target = (15.625, -15.625, -46.875)");
    lastGeneration = 0;

    leastDistance = 10000;

    Initialization3( );
    landMark = "stageThree";
    //Evolutionary loop
    while ( leastDistance  > toleratedError)
    {
        //Fitness of each individual is assessed
        assessFitness3();
        //Best exponents are reported
        report();
        //The probability of cloning is proportional to fitness
        Reproduction();
        //The new population is subjected to mutation
        Mutation3();
        //The new population is subjected to recombination
        Recombination();
        lastGeneration++;
    }
```

```
}


//Genetic algorithm to fit a target string
//that represents the measures and positions
//of an ensemble of boxes that must form a hand
public static void managerOfEvolution()
{
    //Stage one:
    //The measures of the model box are synthesized
    stageOne();
    //Generation 0 also counts
    costOfAlgorithm = lastGeneration;
    //Stage one continued:
    //The position of the first phalanx is synthesized
    stageOneContinued();
    costOfAlgorithm = costOfAlgorithm + lastGeneration;
    //Second and third phalanges
    stageTwo();
    costOfAlgorithm = costOfAlgorithm + lastGeneration;
    //Second,  third and fourth fingers
    stageThree();
    costOfAlgorithm = costOfAlgorithm + lastGeneration;
    System.out.println("Total number of lastGenerations = "
            + costOfAlgorithm);
    System.out.println("Number of individual in "
            + "each lastGeneration = " + nIndiv);
    System.out.println("Tolerated error = "
            + toleratedError);
    System.out.println("Total cost of algorithm = "
            + costOfAlgorithm* nIndiv + " individuals");
    animation.play();
}


//=======================================
//==============3D module===============
//=======================================

//Make the drawing
```

```
private Node create3dContent()
{
    //The scene can be rotated
    XformWithPivot sceneRoot = new XformWithPivot();
    sceneRoot.rx.setAngle(45);
    sceneRoot.ry.setAngle(110);
    getColors();
    //The constructor of hand is invoked.
    Hand  h =  new Hand();
    sceneRoot.getChildren().add(h);
    return new Group(sceneRoot);
}

//Colors with a name in JavaFX are put into an array
static private Color[] getColors()
{
    //Colors with a name in JavaFX
    final Field[] fields = Color.class.getFields();


    int N = 0;
    //Colors are put into an array
    for (final Field field : fields)
    {
        if (field.getType() == Color.class)
        {
            try
            {
                final Color color = (Color) field.get(null);
                //Array to keep colors
                JavaFXColors[N] = color;
                /*
                //The name of the color
                final String colorName = field.getName();
                System.out.println("N = " + N + "  "
                        + color.toString() + " " + colorName);
                */
                N++;
            } catch (IllegalArgumentException
                    | IllegalAccessException ex)
```

```
            {
                Logger.getLogger(FossilRecord5.class
                    .getName()).log(Level.SEVERE, null, ex);
            }
        }
    }
    return JavaFXColors;
}


//A random color is synthesized
 private static Color randomColor()
 {
    //A JavaFX named color is picked
    //but the first, Transparent, is ignored
    int n = r.nextInt(JavaFXColors.length-1)+1;
    Color c = JavaFXColors[n];
    //Some light colors are not visualized so,
    //they might be darkened:
    //c = c.darker();
    return c;
 }



//=====================================================
//=============Inner class XformWithPivot============
//=====================================================



//Kit of available families of transformations
private class XformWithPivot extends Group
{
    //Declaration and initialization
    public Rotate rx = new Rotate();
    { rx.setAxis(Rotate.X_AXIS); }
    public Rotate ry = new Rotate();
    { ry.setAxis(Rotate.Y_AXIS); }
    public Rotate rz = new Rotate();
```

```
    { rz.setAxis(Rotate.Z_AXIS); }
    //Rotation = angle (one coordinate)
    //+ pivot (3 coordinates) + axis.
    //This rotation is around X_AXIS
    public Rotate rwpx = new Rotate(0,0,0,0, X_AXIS);
    //This rotation is around Y_AXIS
    public Rotate rwpy = new Rotate(0,0,0,0, Y_AXIS);
    //This rotation is around Z_AXIS
    public Rotate rwpz = new Rotate(0,0,0,0, Z_AXIS);

    public XformWithPivot()
    {
        super();
        getTransforms().addAll( rx, ry, rz, rwpx, rwpy, rwpz );
    }
}//End of class XformWithPivot



  //=======================================
  //============Inner class Hand===========
  //=======================================



private class Hand extends Group
{

   //A set of boxes are organized as a hand


Hand()
{
    Color c ;
    PhongMaterial m;
    //Arm is constructed
    Box arm = new Box(100,20,400);
    c = randomColor();
    m = new PhongMaterial(c);
    arm.setMaterial(m);
```

```
arm.setTranslateX(0);
arm.setTranslateZ(0);
arm.setTranslateY(20);
//Palm is constructed
Box palm = new Box(125,20,140);
c = randomColor();
m = new PhongMaterial(c);
palm.setMaterial(m);
palm.setTranslateX(0);
palm.setTranslateZ(270);
palm.setTranslateY(20);

//Finger 0 and its phalanges:
//Phalanx 0
c = randomColor();
m = new PhongMaterial(c);
ph00.setMaterial(m);
//Phalanx 1
c = randomColor();
m = new PhongMaterial(c);
ph01.setMaterial(m);
//Phalanx 2
c = randomColor();
m = new PhongMaterial(c);
ph02.setMaterial(m);


//Finger 1 and its phalanges:
//Phalanx 0
c = randomColor();
m = new PhongMaterial(c);
ph10.setMaterial(m);
//Phalanx 1
c = randomColor();
m = new PhongMaterial(c);
ph11.setMaterial(m);
//Phalanx 2
c = randomColor();
m = new PhongMaterial(c);
ph12.setMaterial(m);
```

```
//Finger 2 and its phalanges:
//Phalanx 0
c = randomColor();
m = new PhongMaterial(c);
ph20.setMaterial(m);
//Phalanx 1
c = randomColor();
m = new PhongMaterial(c);
ph21.setMaterial(m);
//Phalanx 2
c = randomColor();
m = new PhongMaterial(c);
ph22.setMaterial(m);


//Finger 3 and its phalanges:
//Phalanx 0
c = randomColor();
m = new PhongMaterial(c);
ph30.setMaterial(m);
//Phalanx 1
c = randomColor();
m = new PhongMaterial(c);
ph31.setMaterial(m);
//Phalanx 2
c = randomColor();
m = new PhongMaterial(c);
ph32.setMaterial(m);


//The first phalanx is made into a group
firstPhalanx.getChildren().add(ph00);
//Phalanges are grouped into fingers
finger0.getChildren().addAll(     ph01, ph02);
finger1.getChildren().addAll(ph10, ph11, ph12);
finger2.getChildren().addAll(ph20, ph21, ph22);
finger3.getChildren().addAll(ph30, ph31, ph32);
//Booking
```

```
    getChildren().addAll(arm, palm
            , firstPhalanx
            , finger0
            , finger1
            , finger2
            , finger3);
    managerOfEvolution();
}

}
}//End of main class M48 FossilRecord5
```

49 ***Exercise.*** *Run the previous program and play with the code.*

50 ***Exercise.*** *Let us define the* ***cost of a genetic algorithm*** *as the number of needed generations times the population size. In regard with the program developed in the previous exercise: find the functional dependency (polynomial, exponential, other) of the cost of the algorithm on the tolerated error. Discuss your results.* Answer

51 ***Our verdict.***

Program M41 FossilRecord2 made an ensemble of boxes into a hand that eventually can bend its fingers into a hook. Our motivation was to pose the following question: Can this hand be explained by Directed Darwinian Evolution if the selection function includes its description in terms of measures of boxes and their positions?

Our answer is a firm yes: by looking at the graphical show of the evolutionary process, one has no other choice but to accept that this problem is too easy for evolution. There is a problem with the quest for perfection but the show does not seem to imply that it could be important in our case.

We also can discuss the role of an additional constraint: Is our stand robust enough to support the fact that no other fossil was found in within the hand of type paddle and that of type hook?

Yes, it is. The problem is so easy and the algorithmic cost so low that destruction + sampling effects of fossils could perfectly explain the absence of intermediate fossils.

We say that this problem is easy because very good approximations are found immediately, usually in less than a second. In comparison, a difficult problem

requires hours of CPU time to produce crude rough solutions. The prototype of a difficult problem is to find the perfect matching of a tree to a distance matrix. The difficulty of this problem is the result of an extreme degree of **pleitropy** that results when every gene affects every observable character (see Vols X and XI).

3.3 Natural Evolution under trial

We argued in the previous section that the problem of explaining the fossil record of a hooked hand beginning with a paddled one without intermediate stages can be explained by Directed Evolution if measures of boxes and their positions are plugged into the selection function. This decision invites us to consider a stronger question: can the same transition be explained by Natural Evolution?

52 ***Reusing Directed Evolution.***

Since we already accepted that Directed Evolution can explain a fossil record, the first step to accept that Natural Evolution also can do it is to replace the inbuilt selection function by a stable natural factor or by a self organizing phenomenon.

The idea could be as the following: as a teacher, I use to assign scores to the works of my students. The purpose is to award critical reading, problem solving and discipline. My score system pretends to direct the evolution of personal techniques of study. This selection function is appreciated by the students and so, they tend to generate a social value that awards the same value. If it gets sufficiently strong, I can die but my principles will survive: directed evolution would pass away given place to a self reinforcing social phenomenon.

53 ***An example from the literature.***

I find very beautiful the proposal given by Young ([16], 2003) on why and how a self organized and self reinforcing phenomenon can be brought forward as the agent that fixed the human hand in hominid populations.

The core of the idea is that this hand can be used to grab clubs, to throw projectiles and also to build tools. We get a self-organized phenomenon because specimens with more developed hands would naturally left more offspring. It is also self-reinforcing because more developed hands are expected to be accompanied with more developed brains and so sexual selection would be more strong in favor of the best hands.

54 ***Proposing a theory.***

Following the argument of Young, our hand whose fingers can be bent into a hook also would provide carriers with a selective advantages over paddled hands: displacement over trees would be possible together with the capability to reach and grab things, say, food, clubs or projectiles. This self organized selective function is stable and generates a trend towards complexity and perfection.

From this we decide that it makes sense to theorize that our hooked hand arouse in our hypothetic world by Natural Evolution.

Now, our work is to compute the predictions of our theory to compare them with available data in regard with the fossil record. Data read: hooked hands prevailed over paddled ones, and apart from these two types of hands there was no other fossil in within concerned strata. Now, we have proven the feasibility of the transition of hands from paddled to hooked but we lack and explanation of the absence of other types of hands. That is why we need to compute the predictions of our theory. We will need some steps to that.

55 ***Predictions of the theory, step 1.***

Our theory is indeed simple: we had a paddled hand together with a genetic machinery to produce mutant boxes and to concatenate them. Let us be more specific in regard with the concatenating mechanism: let us imagine that a a child box can appear when a totipotent cell in some wall of an extant box gives rise to a new box. So, the child can appear at any place of any wall of its mother box and they, child and mother, get automatically glued. We expect that running this machinery must produce an expedite route to a hooked hand.

Our great problem is our difficulty to understand and implement the purposelessness of Natural Evolution. Our opinion is that the lack of purposes means that

1. There is no purpose imposed from the outside.

2. There is no purpose from inside at the start.

3. Differential success in reproduction can be seen a posteriori as the emergence of a self-organized purpose. Say, if a hooked hand is formed by mere chance and if we see that hooked hands prevail over other variants, then we, the beholders, would say in hindsight that a purpose emerged and it was to evolve a hooked hand to grab things. Our great trouble is that we are unable to imagine which function of a specific structural arrangement of boxes,

in a posteriori human form of speaking, could get rewarded by differential reproduction.

Our first step in our calculation program is just to generate new boxes at the walls of the palm of our paddled hand. The code appears in program *M55 FossilRecord6* and can be found in accompanying files.

56 ***Exercise.*** *Run previous program M55 FossilRecord6 and play with the code.*

Our previous program generates boxes at random, with random measures and random positions. As we can see, boxes usually intersect one with another. So, we need to detect intersections to declare them as undesirable.

57 ***Intersections in JavaFX 2D.***

JavaFX 2D offers the possibility to detect when two shapes intersect, such as it is shown by the next program *M57 BoundsPlayground* taken from the literature. You can find it in the adjoint file with programs.

58 ***Exercise.*** *Run the previous program and play with the code.*

59 ***Intersections of boxes in JavaFX 3D.***

Previous program *M57 BoundsPlayground* was adapted to detect intersection of boxes in 3D. The resultant code appears in program *M59 FossilRecord7*

60 ***Exercise.*** *Run the previous program and play with the code.*

61 ***Exercise.*** *Program M59 FossilRecord7 adds new boxes to the wall of the palm of a paddled hand. Furnish it with the possibility to detect and rule out new boxes that intersect old ones. You can compare your answer with ours in program M61 FossilRecord8 in the attached file of programs.*

62 ***Exercise.*** *Program M61 FossilRecord8 adds new not intersecting boxes to the walls of the palm of a paddled hand. Furnish it with the possibility to add recursively boxes to children and grandchildren ...You can compare your answer with ours in program M62 FossilRecord9 in the attached file of programs.*

63 ***Our final model.*** *Previous program adds recursively children boxes to the wall of extant boxes. New boxes were generated at random, with random measures. This situation corresponds with a very high mutation rate that surely would damage the genetic information of the carrier organism. Let us devise an alternative mechanism in which mutation also could be soft and one can regulate the proportion of soft else completely random mutations. The code follows:*

```
/*
    Program M63 FossilRecordd10
    This program adds children boxes
    to the walls of the palm
    of a paddled hand (just arm and palm).
    Children boxes are a mutated version of their mother.
    Soft mutation produces a variation of the mother,
    hard mutation produces a  new box.
    Mutation is a mixture of these two types
    and the proportion can be regulated with a slider.
    Boxes do not intersect one with another because
    intersecting baby boxes are purged.
 */

package fossilrecordd10;

import java.lang.reflect.Field;
import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.event.Event;
import javafx.geometry.Insets;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.PerspectiveCamera;
import javafx.scene.PointLight;
import javafx.scene.Scene;
import javafx.scene.SubScene;
import javafx.scene.control.Button;
import javafx.scene.control.ComboBox;
import javafx.scene.control.Label;
import javafx.scene.control.Slider;
```

```
import javafx.scene.input.PickResult;
import javafx.scene.layout.GridPane;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.Box;
import javafx.scene.transform.Rotate;
import static javafx.scene.transform.Rotate.X_AXIS;
import static javafx.scene.transform.Rotate.Y_AXIS;
import static javafx.scene.transform.Rotate.Z_AXIS;
import javafx.stage.Stage;



public class FossilRecordd10 extends Application
{


    // Turn on  the random generator
    private static final Random r  = new Random();

    //Boolean flags for printing
    private static final boolean    printDragBox = false;

    //Number of generations that
    //the mother box (the palm) reproduces
    private static int nIterations = 7;
    //Number of children of each mother box.
    //Children go over the walls of the mother.
    //Children that cause interference are killed
    private static int nChildren = 15;

    //========DRAWING MODULE==========


    //Slider for zooming
    private Slider zSlider;
    //Position of graphic along the z-axis,
    //normal to the screen.
    //It is a double property because
    //it will get involved into a binding
```

```
private final DoubleProperty zPos
        = new SimpleDoubleProperty(2000);

//Slider to regulate the proportion of
//soft vs hard  random  mutation
//from mother box to child
private Slider softVsHardMutationSlider;

 //This parameter takes values from 0 to 1 and
//regulates the proportion of softw vs hard mutation:
//0: mutation is always soft,
//taking care for inheritance.
//1: mutation is alwyas hard, no care for inheritance.
private final DoubleProperty softVsHardMutationProperty
        = new SimpleDoubleProperty(0.5);

private final Button newTryButton = new Button();


//For listener of mouse
private double anchorAngleX = 0;
private double anchorAngleY = 0;
private double anchorX, anchorY;
//Angles of rotation of the scene
private final DoubleProperty angleX
        = new SimpleDoubleProperty(0);
private final DoubleProperty angleY
        = new SimpleDoubleProperty(0);

PerspectiveCamera camera
        = new PerspectiveCamera(false);

//An array with all colors that are named in JavaFX
private static final Color JavaFXColors[]
                            = new Color[148];



private static final String title
        =  "The palm begets children on its walls";
```

```
    //Boxes are put in a list
    final ObservableList<Box> boxes
            = FXCollections.observableArrayList();
    //List of intersections of boxes by pairs
    final ObservableList<BoxPair> intersections
            = FXCollections.observableArrayList();

    //The constructor of hand is invoked.
    Hand  h;

    private static final boolean printTestIntersections
            = false;

    //The palm gets children, grandchildren,
    //grandgrandchildren...
    private static final ComboBox nIterationsCB = new ComboBox();
    private static final ComboBox nChildrenCB = new ComboBox();

public void init(Stage primaryStage)
{
    primaryStage.setTitle(title);

    //The whole drawing  is created
    //as a JavaFX space-like object
    final Group parent = new Group(create3dContent());


    //Initial position
    parent.setTranslateX(450);
    parent.setTranslateY(400);
    parent.setTranslateZ(0);
    Rotate xRotate;
    Rotate yRotate;
    //Initialization of rotation
    parent.getTransforms().setAll
    (
      xRotate = new Rotate(0, Rotate.X_AXIS),
      yRotate = new Rotate(0, Rotate.Y_AXIS)
```

```
);

//Graphic is attached to root
final Group root = new Group(parent);
//Window of the scene
final Scene scene = new Scene(root, 900, 900, true);
//Background
scene.setFill(Color.SKYBLUE);

//Listener of mousePressed
scene.setOnMousePressed
(
    event ->
    {
        anchorX = event.getSceneX();
        anchorY = event.getSceneY();
        anchorAngleX = angleX.get();
        anchorAngleY = angleY.get();
    }
);


//Listener of mouseDragged
scene.setOnMouseDragged
(
    event ->
    {
        angleX.set(anchorAngleX
                - (anchorY - event.getSceneY()));
        angleY.set(anchorAngleY
                + anchorX - event.getSceneX());
    }
);

//Rotation of the scene gets bound to
//the angle determined by  mouseDragged
xRotate.angleProperty().bind(angleX);
yRotate.angleProperty().bind(angleY);

//Illumination
```

```
PointLight pointLight = new PointLight(Color.WHITE);
pointLight.setTranslateX(100);
pointLight.setTranslateY(200);
pointLight.setTranslateZ(-2000);
root.getChildren().add(pointLight);

//Camera
scene.setCamera(camera);
primaryStage.setScene(scene);
PerspectiveCamera subSceneCamera
        = new PerspectiveCamera(false);
scene.setCamera(subSceneCamera);

//To zoom down the graphic,
//it is translated away from  the beholder,
//along the Z-direction (normal to the screen).
//Change zPos to change initial setting.
zSlider = new Slider();
zSlider.setMax(5000);
zSlider.setMin(1);


//Bind zPos to the reading of the slider
zSlider.valueProperty().bindBidirectional(zPos);
//Execute negative zoom
parent.translateZProperty().bind(zPos);

softVsHardMutationSlider = new Slider();
softVsHardMutationSlider.setMax(1);
softVsHardMutationSlider.setMin(0);
softVsHardMutationSlider.setValue(0.5);
softVsHardMutationSlider.valueProperty()
        .bindBidirectional(softVsHardMutationProperty);

//A new hand is built and shown
newTryButton.setText("New Try");
newTryButton.setOnAction((ActionEvent event) ->
{

    parent.visibleProperty().set(false);
```

```
    boxes.clear();
    intersections.clear();
    init(primaryStage) ;

});

//ComboBox nIterationsCB specifications
nIterationsCB.setValue(nIterations);
nIterationsCB.getItems().addAll("1",  "2", "3", "4", "5","6",
                "7", "8", "9", "10");
 nIterationsCB.setEditable(false);
nIterationsCB.setOnAction((Event ev) ->
{
    String s =
            nIterationsCB.getSelectionModel()
            .getSelectedItem().toString();
    nIterations  = Integer.parseInt(s);
    //System.out.println(nIterations);
});

//ComboBox nChildrenCB specifications
nChildrenCB.setValue(nChildren);
nChildrenCB.setValue(nChildren);
nChildrenCB.getItems().addAll("1",  "2", "3", "4", "5","6",
                "7", "8", "9", "10", "11",
                "12", "13", "14", "15");
nChildrenCB.setEditable(false);
nChildrenCB.setOnAction((Event ev) ->
{
    String s =
            nChildrenCB.getSelectionModel()
            .getSelectedItem().toString();
    nChildren  = Integer.parseInt(s);
    //System.out.println(nChildren);
});

//Other JavaFX controls, labels and containers
GridPane gridPane = new GridPane();
gridPane.setVgap(4);
gridPane.setHgap(10);
```

```
gridPane.setPadding(new Insets(5, 5, 5, 5));
//Column 0, Row   0
gridPane.add(new Label("Zoom: "), 0, 0);
gridPane.add(zSlider, 1, 0);
gridPane.add(new Label("Soft Mutation: "), 0, 1);
gridPane.add(softVsHardMutationSlider, 1, 1);
gridPane.add(new Label(": Hard Mutation "), 2, 1);
gridPane.add(new Label("Zoom: "), 0, 0);
gridPane.add(newTryButton, 4, 0);
gridPane.add(new Label("Number of iterations: "), 0, 2);
gridPane.add(nIterationsCB, 1, 2);
gridPane.add(new Label("Number of children: "), 3, 2);
gridPane.add(nChildrenCB, 4, 2);



SubScene subScene
        = new SubScene(gridPane, 880, 200);
subScene.setLayoutX(10);
subScene.setLayoutY(800);
root.getChildren().add(subScene);

primaryStage.show();

//The sense of touch is here
parent.setOnMouseClicked((event)->
{
    //Information that can be retrieved
    PickResult res = event.getPickResult();
    //Which node
    Node node = res.getIntersectedNode();
    //Identity label of node (null if not declared)
    String nodeId  = event.getPickResult()
            .getIntersectedNode().getId();
    //which face of the node (negative if is not enabled
    int face = event.getPickResult().getIntersectedFace();
    System.out.println("res = " + res);
    System.out.println("node = " + node);
    System.out.println("nodeID = " + nodeId);
```

```
        System.out.println("face" + face);
        //Change color to red, rotate around Z_AXIS
        if (node instanceof Box)
        {
            ((Box) node).setMaterial(
                    new PhongMaterial( Color.RED));

        }
    });
}



@Override
public void start(Stage primaryStage)  throws Exception
{

    init(primaryStage);
    primaryStage.show();
}


public static void main(String[] args)
{

    launch(args);
}

//=======================================
//==============3D module===============
//=======================================

//Make the drawing
private Node create3dContent()
{
    //The scene can be rotated
    XformWithPivot sceneRoot = new XformWithPivot();
    sceneRoot.rx.setAngle(45);
    sceneRoot.ry.setAngle(110);
    getColors();
```

```
    //The constructor of hand is invoked.
    h =  new Hand();
    sceneRoot.getChildren().add(h);
    return new Group(sceneRoot);
}

//Colors with a name in JavaFX are put into an array
static private Color[] getColors()
{
    //Colors with a name in JavaFX
    final Field[] fields = Color.class.getFields();


    int N = 0;
    //Colors are put into an array
    for (final Field field : fields)
    {
        if (field.getType() == Color.class)
        {
            try
            {
                final Color color = (Color) field.get(null);
                //Array to keep colors
                JavaFXColors[N] = color;
                /*
                //The name of the color
                final String colorName = field.getName();
                System.out.println("N = " + N + "  "
                        + color.toString() + " " + colorName);
                */
                N++;
            } catch (IllegalArgumentException
                    | IllegalAccessException ex)
            {
                Logger.getLogger(FossilRecordd10.class
                    .getName()).log(Level.SEVERE, null, ex);
            }
       }
    }
    return JavaFXColors;
```

```
}


//A random color is synthesized
 private static Color randomColor()
 {
    //A JavaFX named color is picked
    //but the first, Transparent, is ignored
    int n = r.nextInt(JavaFXColors.length-1)+1;
    Color c = JavaFXColors[n];
    //Some light colors are not visualized so,
    //they might be darkened:
    //c = c.darker();
    return c;
 }


//A box is synthesized with random measures
private static Box generateRandomBox()
{
    int bound = r.nextInt(200) +1;
    Box box = new Box();
    Color c ;
    PhongMaterial m;
    c = randomColor();
    m = new PhongMaterial(c);
    box.setMaterial(m);
    int d = r.nextInt(bound);
    box.setDepth(d);
    int h = r.nextInt(bound);
    box.setHeight(h);
    int w = r.nextInt(bound);
    box.setWidth(w);

    return box;
}

//Mother box undergoes mutation of measures
private static Box mutateBox(Box motherT)
{
```

```
    double d = motherT.getDepth();
    double h = motherT.getHeight();
    double w = motherT.getWidth();

    Box child = new Box();
    Color c ;
    PhongMaterial m;
    c = randomColor();
    m = new PhongMaterial(c);
    child.setMaterial(m);
    int bound = 50;

    int dd = r.nextInt(bound);
    //Measure goes up else down
    int f = r.nextInt(2);
    if (f == 0) child.setDepth(d + dd);
    else
    {
        if (d-dd > 0) child.setDepth(d - dd);
    }

    int hh = r.nextInt(bound);
    f = r.nextInt(2);
    if (f == 0) child.setHeight(h + hh);
    else
    {
        if (h-hh > 0) child.setHeight(h - hh);
    }

    int ww = r.nextInt(bound);
    f = r.nextInt(2);
    if (f == 0) child.setWidth(w + ww);
    else
    {
        if (w - ww > 0) child.setWidth(w - ww);
    }

    return child;
}
```

```
//Child box is dragged to a random place
// upon a face of its motherT.
//To understand how faces are numbered, let us imagine
//that we have a unitary cube with one corner at (0,0,0)
//and oppossed one at (1,1,1). Faces are numbered as:
//Face 0 : x = 0;
//Face 1 : x = 1;
//Face 2 : y = 0;
//Face 2 : y = 1;
//Face 4 : z = 0;
//Face 5 : z = 1;
private static Box dragBox(Box mother, Box child)
{

    //A face is chosen at random
    int face = r.nextInt(6);
    double motherD = mother.getDepth();
    double motherH = mother.getHeight();
    double motherW = mother.getWidth();
    double motherTX = mother.getTranslateX();
    double motherTY = mother.getTranslateY();
    double motherTZ = mother.getTranslateZ();
    if (printDragBox)
    {
    System.out.println("Mother: ");
    System.out.println("Measures: " + motherW + "  "
            + motherH + "  " +  motherD);
    System.out.println("Position " +  motherTX + "  "
            + motherTY + "  " +  motherTZ);
    }
    //Input is read
    double childD = child.getDepth();
    double childH = child.getHeight();
    double childW = child.getWidth();
    double childTX = 0;
    double childTY = 0;
    double childTZ = 0;
    //Sign1 is assigned
    int k = r.nextInt(2);
```

```
   int sign1;
 if ( k == 0) sign1 = 1;
 else sign1 = -1;
 //sign2
 k = r.nextInt(2);
   int sign2;
 if ( k == 0) sign2 = 1;
 else sign2 = -1;
//New box is posited at a face of the mother box
 switch (face)
 {
     case 0: childTX = motherTX  - motherW/2 - childW/2;
             childTY= motherTY
                     + sign1 * motherH*r.nextDouble()/2;
             childTZ = motherTZ
                     + sign2 * motherD*r.nextDouble()/2;
          break;
     case 1: childTX = motherTX + motherW/2 + childW/2;
             childTY = motherTY
                     + sign1 * motherH*r.nextDouble()/2;
             childTZ = motherTZ
                     + sign2 * motherD*r.nextDouble()/2;
         break;
     case 2: childTX = motherTX
             + sign1 * motherW*r.nextDouble()/2;
             childTY = motherTY - motherH/2 - childH/2;
             childTZ = motherTZ
                     + sign2 * motherD*r.nextDouble()/2;
         break;
     case 3: childTX = motherTX
             + sign1 * motherW*r.nextDouble()/2;
             childTY = motherTY + motherH/2 + childH/2;
             childTZ = motherTZ
                     + sign2 * motherD*r.nextDouble()/2;
         break;
     case 4: childTX = motherTX
             + sign1 *  motherW*r.nextDouble()/2;
             childTY = motherTY
                     + sign2 *  motherH*r.nextDouble()/2;
             childTZ = motherTZ - motherD/2 - childD/2;
```

```
            break;
        case 5: childTX = motherTX
                + sign1 * motherW*r.nextDouble()/2;
                childTY = motherTY
                        + sign2 * motherH*r.nextDouble()/2;
                childTZ = motherTZ +  motherD/2 + childD/2;
            break;
    }

    //New box is posited at a face of the mother box
    child.setTranslateX(childTX);
    child.setTranslateY(childTY);
    child.setTranslateZ(childTZ);
    if (printDragBox)
    {
        System.out.println("Child: ");
        System.out.println( "Measures: " + childW + "  "
                + childH + "  " +  childD);
        System.out.println("Position" +  childTX + "  "
                + childTY + "  " +  childTZ);
    }

    return child;
}


//=================================================
//=============Inner class XformWithPivot===========
//=================================================



//Kit of available families of transformations
private static class XformWithPivot extends Group
{
    //Declaration and initialization
    public Rotate rx = new Rotate();
    { rx.setAxis(Rotate.X_AXIS); }
    public Rotate ry = new Rotate();
    { ry.setAxis(Rotate.Y_AXIS); }
```

```
    public Rotate rz = new Rotate();
    { rz.setAxis(Rotate.Z_AXIS); }
    //Rotation = angle (one coordinate)
    //+ pivot (3 coordinates) + axis.
    //This rotation is around X_AXIS
    public Rotate rwpx = new Rotate(0,0,0,0, X_AXIS);
    //This rotation is around Y_AXIS
    public Rotate rwpy = new Rotate(0,0,0,0, Y_AXIS);
    //This rotation is around Z_AXIS
    public Rotate rwpz = new Rotate(0,0,0,0, Z_AXIS);

    public XformWithPivot()
    {
        super();
        getTransforms().addAll( rx, ry, rz, rwpx, rwpy, rwpz );
    }
}//End of class XformWithPivot



  //=======================================
  //============Inner class Hand===========
  //=======================================



private class Hand extends Group
{

   //A set of boxes are organized as a hand


Hand()
{
    Color c ;
    PhongMaterial m;
    //Arm is constructed
    Box arm = new Box(100,20,400);
    c = randomColor();
    m = new PhongMaterial(c);
```

```
arm.setMaterial(m);
arm.setTranslateX(0);
arm.setTranslateY(20);
arm.setTranslateZ(0);

//Palm is constructed
Box palm = new Box(125,20,140);
c = randomColor();
m = new PhongMaterial(c);
palm.setMaterial(m);
palm.setTranslateX(0);
palm.setTranslateY(20);
palm.setTranslateZ(270);


final Group bigGroup = new Group(arm, palm);
boxes.add( arm);
boxes.add(arm);
ObservableList<Box> OldMothers
        = FXCollections.observableArrayList();
OldMothers.add(palm);
//Recursion over the generation of offspring
for(int iterations = 0; iterations < nIterations; iterations++)
{
    ObservableList<Box> NewMothers
                = FXCollections.observableArrayList();
    for(Box mother: OldMothers)
    {
        //Loop for trying new boxes
        //on the walls of a given mother box
        for(int i = 0; i < 15; i++)
        {

            Box child;
            double p = r.nextDouble();
            if (p < softVsHardMutationProperty.getValue())
                child = mutateBox(mother);
            else child = generateRandomBox();

            child = dragBox(mother, child);
```

```
                bigGroup.getChildren().add(child);
                boxes.add( child);
                NewMothers.add(child);
                //Intersections are caught and reported
                testIntersections();
                //Purging
                if (intersections.size() != 0)
                {
                    //If the new box causes intersections,
                    //it is deleted.
                    bigGroup.getChildren().remove(child);
                    boxes.remove(child);
                    NewMothers.remove(child);
                }
            }

        }
        //Reinitialization:
        //New mothers are ready to get offspring
        OldMothers.clear();
        OldMothers.addAll(NewMothers);
    }

   //Booking
   getChildren().addAll(bigGroup);

}


//Intersections re caugth and listed
private void testIntersections()
{
    intersections.clear();
    //All pairs of boxes are tested for intersections.
    boxes.stream().forEach((src) ->
    {
        for (Box dest : boxes)
        {
            BoxPair pair = new BoxPair(src, dest);
            if (!intersections.contains(pair)
```

```
                    && pair.intersects())
            {
                intersections.add(pair);
            }
        }
    });
    //Report
    if (printTestIntersections)
    System.out.println("intersections = "
            + intersections.toString());
}

}//End of class Hand

//Inner class
//Records a pair of (possibly) intersecting boxes.
class BoxPair
{
    private final Box a, b;
    public BoxPair(Box src, Box dest)
    {
        this.a = src; this.b = dest;
    }

    //Boolean function
    public boolean intersects()
    {
        if (a == b) return false;
        a.intersects(b.getBoundsInLocal());
        return a.getBoundsInParent()
                .intersects(b.getBoundsInParent());
    }


    //A pair is translated to string
    @Override public String toString()
    {
        return a.getId() + " : " + b.getId();
    }
}//End of class BoxPair
```

```
}//End of main class M63 FossilRecordd10
```

64 ***Exercise.*** *Run the previous program and play with the code. Play enough with your new toy to agree else disagree with the claim of the Author: mutant hands have chained structures than eventually might evolve into a finger. Other hands might be used as weapons, say, axes, awls, hammers and maces. There are some mutant hands that are mechanically unfit because they have a heavy head that is sustained by a weak neck.*

65 ***Our verdict.***

Selection, of whatever type, functions as filter of mutants. A mutational machinery was simulated in the previous program with the possibility to recursively add boxes to the walls of extant boxes. By design, this machinery can produce complex structures similar to composed tails or fingers. Nevertheless, the Author considers that resulting drawings naturally inspire the beholder to see structures that might be used as weapons: axes, awls, hammers and maces. Therefore, a prediction immediately springs: a trend towards sophistication of hands that are weapons by themselves must had been arisen with a clear track in the fossil record. This prediction makes sense in the presence of a high doses of intelligence, both to understand that a hand can be used as a weapon and to exercise sexual selection.

Let us contrast our mandatory prediction with data of our virtual world. These say that from paddled hands to hooked hands there is no other fossil while our mandatory prediction claims that diversity of fossils must exist. So, our answer to posed question is negative: hooked hands did not appear in our imagined world by natural selection from paddled ones.

Warning: our discussion involves a high load of anthropomorphism that allows us to rapidly compose a discourse about natural selection: after seen something similar to a hammer we were driven to imagine that natural selection also sees a hammer. This is no more that an humanization of natural selection. The Reader is invited to consider the next parable to try to understand our complaint:

There are many games with cards, say, poker. A simple form is the following:

1. Fix a number as the target, say, 21.
2. Give a value to each card.
3. Shuffle cards, to introduce randomness.
4. Take as many cards as wanted, add together their values.

5. The winner is the first that reaches to the target number or that is closes to it but under it. Say, 22 loses irremediably while 20 wins over 18.

This simple game is very entertaining because it is like approaching the abyss but without getting into it. An expert player is one that is a usual winner in spite of the chance that is immanent to the game. Now, when one sees players occupied with cards, one usually tends to think in terms of the games one knows, most of which challenge as the intelligence as the courage. Now, this is like Directed Evolution: you have a goal and a lot of intelligence and resources to achieve it. By contrast, natural selection is very different, it is like the next game that is played in Africa:

1. Cards are flushed and the same number of cards is given to all players.
2. Each player puts one of its cards over the ground in the center of the group.
3. Who is the winner? It is that person whose card has been visited by a fly before everyone else.

This parable is directed to illustrate the fact that the application to natural selection of our preconceptions about functions might be no more than mythology. Now, what shall we expect if we free Natural Evolution from our anthropomorphisms? In all cases, it is expected too much variability. In our case, this is plainly contradicted by data that is accepted in our fictitious world: in within paddled and hooked hands there is no other fossil.

In conclusion: we have no reason to believe that our hooked hand arouse by Natural Evolution in our fictitious world from a paddled one. Notice that the appearance of fingers is unavoidable and that eventually a hooked hand could had appeared naturally. So, why do we reject Natural Evolution? The reason is that we have a mandatory prediction: were Natural Evolution the cause of the transition, from paddled to hooked hands, then a great diversity of fossils should had been also observed, say, many forms of weapons. But this fact is not supported by the fossil record of our fictitious world.

66 ***Challenge.*** *We studied the problem of explaining one transition, to pass from a paddled hand to a hooked one, by Directed and Natural Evolutions. Choose another transition of the posed fossil record and carry out the same study.*

3.4 Conclusion

We prepared a virtual world in which we posed a fossil record of a hand, which was just an ensemble of boxes observing an appropriate distribution. We choose

one transition, to pass from a paddled hand (arm + palm) to a hooked one that has fingers and that presumingly can bend them to form a hook. We found no trouble in explained this transition by Directed Evolution, in which the observer plugged the information encoding for the target hand. The problem was so easy for this type of evolution that transition fossils were expected to be few in number and then discarded by sampled effects of the fossil quest. So, this type of evolution is consistent with data of our imagined world: in within a paddle hand and a hooked one there is no other fossil. This success animated us to pose a most difficult question: can the same transition be explained by Natural Evolution? So, we plugged no more target structures and just developed a mutational machinery that allows the reproduction of boxes whose children appear automatically over the walls of the mother. Structures than eventually would evolve into fingers are easily found. Other structures were promises for every sort of weapons: axes, awls, hammers and maces. Therefore, a prediction immediately sprung: a trend towards sophistication of hands that are weapons by themselves must had been arisen with a clear track in the fossil record. This prediction makes sense in the presence of a high doses of intelligence, both to understand that a hand can be used as a weapon and to exercise sexual selection. If no such intelligence is available, the mandatory prediction is that a high variability of fossils must exist. Both predictions are in contradiction with the field data of our virtual world, so we rejected the idea that posed transition can be explained by Natural Evolution.

Warning: our discussion involved at some moment a high load of anthropomorphism that allowed us to rapidly compose a discourse about natural selection: after seen something similar to a hammer we were driven to imagine that natural selection also saw a hammer. This is no more that an humanization of natural selection. The Reader is invited to consider the next parable to try to understand our complaint:

There are many games with cards, say, poker. A simple form is the following:

1. Fix a number as the target, say, 21.
2. Give a value to each card.
3. Shuffle cards, to introduce randomness.
4. Take as many cards as wanted, add together their values.
5. The winner is the first that reaches to the target number or that is closes to it but under it. Say, 22 loses irremediably while 20 wins over 18.

This simple game is very entertaining because it is like approaching the abyss but without getting into it. An expert player is one that is a usual winner in spite

of the chance that is immanent to the game. Now, when one sees players occupied with cards, one usually tends to think in terms of the games one knows, most of which challenge as the intelligence as the courage. Now, this is like Directed Evolution: you have a goal and a lot of intelligence together with resources to achieve it. By contrast, natural selection is very different, it is like the next game that is played in Africa:

1. Cards are flushed and the same number of cards is given to all players.
2. Each player puts one of its cards over the ground in the center of the group.
3. Who is the winner? It is that person whose card has been visited by a fly before everyone else.

This parable is directed to illustrate the fact that the application to natural selection of our preconceptions about functions might be no more than mythology. Now, what shall we expect if we free Natural Evolution from our anthropomorphisms? In all cases, it is expected too much variability. In our case, this is plainly contradicted by data that is accepted in our fictitious world: in within paddled and hooked hands there is no other fossil.

Theoretically, we find the instantiation of a simple principle: complexity can be created in too many forms and problems can be solved in too many ways and therefore variability is a mandatory prediction when evolution is invoked to explain a complex object.

Chapter 4

3D from 2D panes

Sheets create space

67 ***Introduction and objective.*** *We have seen how to use spheres, boxes and cylinders to create 3D shapes that resemble objects of this world. One might imagine that to augment the perfection of imitations, one can use those primitives in varying size with a tendency to miniaturization. Nevertheless this approach is extremely expensive. So, let us explore others way as JavaFX allows the creation of personalized 3D shapes. We will see in present chapter how to glue 2D panes to form 3D shapes. This is a usual strategy in the world of construction, say, of buildings.*

68 ***A cube and a pyramid***

One can make three dimensional objects by gluing together two dimensional panes. One can do the same in JavaFX.

69 ***A cube is assembled from 2D rectangles:***

```
/*
//Program M69 CubeFrom2D
//2D Rectangles are manipulated to
//form a cube
 */
package cubefrom2d;

import javafx.animation.Animation;
import javafx.animation.KeyFrame;
```

```
import javafx.animation.KeyValue;
import javafx.animation.Timeline;
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.PerspectiveCamera;
import javafx.scene.Scene;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.scene.shape.RectangleBuilder;
import javafx.scene.transform.Rotate;
import javafx.stage.Stage;
import javafx.util.Duration;

/**
 *
 * @author Saravanababu
 * http://www.javafxapps.in/tutorial/Creating-3D-Cube-in-javafx.html
 */
public class CubeFrom2D extends Application {

    private Timeline animation;

    @Override
    public void start(Stage primaryStage) {

        Group cube = new Group();
        //size of the cube
        double size = 75;
        //set color for the cube
        Color color = Color.DARKCYAN;
        cube.getChildren().addAll(
                RectangleBuilder.create() // back face
                .width(size).height(size)
                .fill(color.deriveColor(0.0, 1.0, (1 - 0.5 * 1), 1.0))
                .translateX(-0.5 * size)
                .translateY(-0.5 * size)
                .translateZ(0.5 * size)
                .build(),
                RectangleBuilder.create() // bottom face
                .width(size).height(size)
```

```
            .fill(color.deriveColor(0.0, 1.0, (1 - 0.4 * 1), 1.0))
            .translateX(-0.5 * size)
            .translateY(0)
            .rotationAxis(Rotate.X_AXIS)
            .rotate(90)
            .build(),
            RectangleBuilder.create() // right face
            .width(size).height(size)
            .fill(color.deriveColor(0.0, 1.0, (1 - 0.3 * 1), 1.0))
            .translateX(-1 * size)
            .translateY(-0.5 * size)
            .rotationAxis(Rotate.Y_AXIS)
            .rotate(90)
            .build(),
            RectangleBuilder.create() // left face
            .width(size).height(size)
            .fill(color.deriveColor(0.0, 1.0, (1 - 0.2 * 1), 1.0))
            .translateX(0)
            .translateY(-0.5 * size)
            .rotationAxis(Rotate.Y_AXIS)
            .rotate(90)
            .build(),
            RectangleBuilder.create() // top face
            .width(size).height(size)
            .fill(color.deriveColor(0.0, 1.0, (1 - 0.1 * 1), 1.0))
            .translateX(-0.5 * size)
            .translateY(-1 * size)
            .rotationAxis(Rotate.X_AXIS)
            .rotate(90)
            .build(),
            RectangleBuilder.create() // front face
            .width(size).height(size)
            .fill(color)
            .translateX(-0.5 * size)
            .translateY(-0.5 * size)
            .translateZ(-0.5 * size)
            .build());
cube.getTransforms().addAll(new Rotate(25, Rotate.Z_AXIS),new Rotate(7
StackPane root = new StackPane();
root.getChildren().add(cube);
```

```
        animation = new Timeline();
        animation.getKeyFrames().addAll(
                new KeyFrame(Duration.ZERO,
                new KeyValue(cube.rotationAxisProperty(), Rotate.X_AXI
                new KeyValue(cube.rotateProperty(), 0d)),
                new KeyFrame(Duration.seconds(5),
                new KeyValue(cube.rotationAxisProperty(), Rotate.X_AXI
                new KeyValue(cube.rotateProperty(), 360d)));
        animation.setCycleCount(Animation.INDEFINITE);
        animation.play();
        Scene scene = new Scene(root, 300, 250, true);
        scene.setCamera(new PerspectiveCamera());
        primaryStage.setResizable(false);
        primaryStage.setTitle("Cube Demo");
        primaryStage.setScene(scene);
        primaryStage.show();
    }


    public static void main(String[] args) {
        launch(args);
    }
}//End of main class M69 CubeFrom2D
```

70 ***Exercise .*** *Run the previous program and play with the code.*

71 ***To be a Java object means power to create. The cube, as object, as class, is assembled from 2D rectangles:***

```
//Program M71 CubeFrom2D2
//2D Rectangles are manipulated to
//form a cube that is made into a class, an object.

/*
 * Copyright (c) 2008, 2012 Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
```

```
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle Corporation nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */
package cubefrom2d2;

import javafx.animation.Animation;
import javafx.animation.KeyFrame;
import javafx.animation.KeyValue;
import javafx.animation.Timeline;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.scene.DepthTest;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.PerspectiveCamera;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
```

```
import javafx.scene.shape.RectangleBuilder;
import javafx.scene.transform.Rotate;
import javafx.scene.transform.Translate;
import javafx.stage.Stage;
import javafx.util.Duration;


public class CubeFrom2D2 extends Application {

    private Timeline animation;

private void init(Stage primaryStage) {
    Group root = new Group();
    root.setDepthTest(DepthTest.ENABLE);
    primaryStage.setResizable(false);
    primaryStage.setScene(new Scene(root, 400, 150, true));
    primaryStage.getScene().setCamera(new PerspectiveCamera());
    root.getTransforms().addAll(
        new Translate(400 / 2, 150 / 2),
        new Rotate(180, Rotate.X_AXIS)
    );
    root.getChildren().add(create3dContent());
}

public Node create3dContent() {
    Cube c = new Cube(50,Color.RED,1);
    c.rx.setAngle(45);
    c.ry.setAngle(45);
    Cube c2 = new Cube(50,Color.GREEN,1);
    c2.setTranslateX(100);
    c2.rx.setAngle(45);
    c2.ry.setAngle(45);
    Cube c3 = new Cube(50,Color.ORANGE,1);
    c3.setTranslateX(-100);
    c3.rx.setAngle(45);
    c3.ry.setAngle(45);

    animation = new Timeline();
    animation.getKeyFrames().addAll(
            new KeyFrame(Duration.ZERO,
```

```
                    new KeyValue(c.ry.angleProperty(), 0d),
                    new KeyValue(c2.rx.angleProperty(), 0d),
                    new KeyValue(c3.rz.angleProperty(), 0d)
            ),
            new KeyFrame(Duration.seconds(1),
                    new KeyValue(c.ry.angleProperty(), 360d),
                    new KeyValue(c2.rx.angleProperty(), 360d),
                    new KeyValue(c3.rz.angleProperty(), 360d)
            ));
    animation.setCycleCount(Animation.INDEFINITE);

    return new Group(c,c2,c3);
}

public void play() {
    animation.play();
}

@Override public void stop() {
    animation.pause();
}

public class Cube extends Group {
final Rotate rx = new Rotate(0,Rotate.X_AXIS);
final Rotate ry = new Rotate(0,Rotate.Y_AXIS);
final Rotate rz = new Rotate(0,Rotate.Z_AXIS);
public Cube(double size, Color color, double shade) {
    getTransforms().addAll(rz, ry, rx);
    getChildren().addAll(
        RectangleBuilder.create() // back face
            .width(size).height(size)
            .fill(color.deriveColor(0.0, 1.0, (1 - 0.5*shade), 1.0))
            .translateX(-0.5*size)
            .translateY(-0.5*size)
            .translateZ(0.5*size)
            .build(),
        RectangleBuilder.create() // bottom face
            .width(size).height(size)
            .fill(color.deriveColor(0.0, 1.0, (1 - 0.4*shade), 1.0))
            .translateX(-0.5*size)
```

```
            .translateY(0)
            .rotationAxis(Rotate.X_AXIS)
            .rotate(90)
            .build(),
        RectangleBuilder.create() // right face
            .width(size).height(size)
            .fill(color.deriveColor(0.0, 1.0, (1 - 0.3*shade), 1.0))
            .translateX(-1*size)
            .translateY(-0.5*size)
            .rotationAxis(Rotate.Y_AXIS)
            .rotate(90)
            .build(),
        RectangleBuilder.create() // left face
            .width(size).height(size)
            .fill(color.deriveColor(0.0, 1.0, (1 - 0.2*shade), 1.0))
            .translateX(0)
            .translateY(-0.5*size)
            .rotationAxis(Rotate.Y_AXIS)
            .rotate(90)
            .build(),
        RectangleBuilder.create() // top face
            .width(size).height(size)
            .fill(color.deriveColor(0.0, 1.0, (1 - 0.1*shade), 1.0))
            .translateX(-0.5*size)
            .translateY(-1*size)
            .rotationAxis(Rotate.X_AXIS)
            .rotate(90)
            .build(),
        RectangleBuilder.create() // top face
            .width(size).height(size)
            .fill(color)
            .translateX(-0.5*size)
            .translateY(-0.5*size)
            .translateZ(-0.5*size)
            .build()
    );
}
}

@Override public void start(Stage primaryStage) throws Exception {
```

```
    init(primaryStage);
    primaryStage.show();
    play();
}


public static void main(String[] args) {
    launch(args);
}
}//End of main class M71 CubeFrom2D2
```

72 ***Exercise .*** *Run the previous program and play with the code.*

73 ***The object cube shows the power of being a class by exhibiting a complex planetary system :***

```
//Program M73 CubeSystem3D
//A complex planetary system is made of cubes


/*
 * Copyright (c) 2008, 2012 Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle Corporation nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
```

```
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FO
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON AN
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE US
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE
 */
package cubesystem3d;

import javafx.animation.KeyFrame;
import javafx.animation.KeyValue;
import javafx.animation.Timeline;
import javafx.application.Application;
import javafx.scene.DepthTest;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.PerspectiveCamera;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.RectangleBuilder;
import javafx.scene.transform.Rotate;
import javafx.scene.transform.Translate;
import javafx.stage.Stage;
import javafx.util.Duration;

/**
 * A sample that demonstrates a system of animated 3D cubes. When the
 * application runs in standalone mode, the scene must be constructed w
 * the depthBuffer argument set to true, and the root node must have de
 * set to true.
 *
 * @see javafx.scene.transform.Rotate
 * @see javafx.scene.paint.Color
 * @see javafx.scene.shape.RectangleBuilder
```

```
 */
public class CubeSystem3D extends Application {

    private Timeline animation;

private void init(Stage primaryStage) {
    Group root = new Group();
    root.setDepthTest(DepthTest.ENABLE);
    primaryStage.setResizable(false);
    primaryStage.setScene(new Scene(root, 500, 500, true));
    primaryStage.getScene().setCamera(new PerspectiveCamera());
    root.getTransforms().addAll(
        new Translate(500 / 2, 500 / 2),
        new Rotate(180, Rotate.X_AXIS)
    );
    root.getChildren().add(create3dContent());
}

public Node create3dContent() {
    Xform sceneRoot = new Xform();

    Xform cube1X = new Xform();
    Cube cube1  = new Cube(40, new Color(1.0, 0.9, 0.0, 1.0), 1.0);

    Xform cube1_1X = new Xform();
    Xform cube1_2X = new Xform();
    Xform cube1_3X = new Xform();
    Xform cube1_4X = new Xform();
    Xform cube1_5X = new Xform();
    Xform cube1_6X = new Xform();
    Xform cube1_7X = new Xform();
    Xform cube1_8X = new Xform();
    Xform cube1_9X = new Xform();

    Cube cube1_1 = new Cube(4, Color.RED, 1.0);
    Cube cube1_2 = new Cube(5, Color.ORANGE, 1.0);
    Cube cube1_3 = new Cube(6, Color.CORNFLOWERBLUE, 1.0);
    Cube cube1_4 = new Cube(7, Color.DARKGREEN, 1.0);
    Cube cube1_5 = new Cube(8, Color.BLUE, 1.0);
    Cube cube1_6 = new Cube(9, Color.PURPLE, 1.0);
```

```
Cube cube1_7 = new Cube(10, Color.BLUEVIOLET, 1.0);
Cube cube1_8 = new Cube(11, Color.DARKGOLDENROD, 1.0);
Cube cube1_9 = new Cube(12, Color.KHAKI, 1.0);

sceneRoot.getChildren().add(cube1X);

cube1X.getChildren().add(cube1);
cube1X.getChildren().add(cube1_1X);
cube1X.getChildren().add(cube1_2X);
cube1X.getChildren().add(cube1_3X);
cube1X.getChildren().add(cube1_4X);
cube1X.getChildren().add(cube1_5X);
cube1X.getChildren().add(cube1_6X);
cube1X.getChildren().add(cube1_7X);
cube1X.getChildren().add(cube1_8X);
cube1X.getChildren().add(cube1_9X);

cube1_1X.getChildren().add(cube1_1);
cube1_2X.getChildren().add(cube1_2);
cube1_3X.getChildren().add(cube1_3);
cube1_4X.getChildren().add(cube1_4);
cube1_5X.getChildren().add(cube1_5);
cube1_6X.getChildren().add(cube1_6);
cube1_7X.getChildren().add(cube1_7);
cube1_8X.getChildren().add(cube1_8);
cube1_9X.getChildren().add(cube1_9);

cube1_1.setTranslateX(40.0);
cube1_2.setTranslateX(60.0);
cube1_3.setTranslateX(80.0);
cube1_4.setTranslateX(100.0);
cube1_5.setTranslateX(120.0);
cube1_6.setTranslateX(140.0);
cube1_7.setTranslateX(160.0);
cube1_8.setTranslateX(180.0);
cube1_9.setTranslateX(200.0);

cube1_1X.rx.setAngle(30.0);
cube1_2X.rz.setAngle(10.0);
cube1_3X.rz.setAngle(50.0);
```

```
cube1_4X.rz.setAngle(170.0);
cube1_5X.rz.setAngle(60.0);
cube1_6X.rz.setAngle(30.0);
cube1_7X.rz.setAngle(120.0);
cube1_8X.rz.setAngle(40.0);
cube1_9X.rz.setAngle(-60.0);

double endTime = 4000.0;

// Animate
animation = new Timeline();
animation.getKeyFrames().addAll
(new KeyFrame(Duration.ZERO,
              new KeyValue(cube1X.ry.angleProperty(), 0.0)),
 new KeyFrame(new Duration(endTime),
              new KeyValue(cube1X.ry.angleProperty(), 360.0)),
 new KeyFrame(Duration.ZERO,
              new KeyValue(cube1X.rx.angleProperty(), 0.0)),
 new KeyFrame(new Duration(endTime),
              new KeyValue(cube1X.rx.angleProperty(), 360.0)),
 new KeyFrame(Duration.ZERO,
              new KeyValue(cube1_1X.ry.angleProperty(), 0.0)),
 new KeyFrame(new Duration(endTime),
              new KeyValue(cube1_1X.ry.angleProperty(), -2880.0)),
 new KeyFrame(Duration.ZERO,
              new KeyValue(cube1_2X.ry.angleProperty(), 0.0)),
 new KeyFrame(new Duration(endTime),
              new KeyValue(cube1_2X.ry.angleProperty(), -1440.0)),
 new KeyFrame(Duration.ZERO,
              new KeyValue(cube1_3X.ry.angleProperty(), 0.0)),
 new KeyFrame(new Duration(endTime),
              new KeyValue(cube1_3X.ry.angleProperty(), -1080.0)),
 new KeyFrame(Duration.ZERO,
              new KeyValue(cube1_4X.ry.angleProperty(), 0.0)),
 new KeyFrame(new Duration(endTime),
              new KeyValue(cube1_4X.ry.angleProperty(), -720.0)),
 new KeyFrame(Duration.ZERO,
              new KeyValue(cube1_5X.ry.angleProperty(), 0.0)),
 new KeyFrame(new Duration(endTime),
              new KeyValue(cube1_5X.ry.angleProperty(), 1440.0)),
```

```
new KeyFrame(Duration.ZERO,
             new KeyValue(cube1_6X.ry.angleProperty(), 0.0)),
new KeyFrame(new Duration(endTime),
             new KeyValue(cube1_6X.ry.angleProperty(), 1080.0)),
new KeyFrame(Duration.ZERO,
             new KeyValue(cube1_7X.ry.angleProperty(), 0.0)),
new KeyFrame(new Duration(endTime),
             new KeyValue(cube1_7X.ry.angleProperty(), -360.0)),
new KeyFrame(Duration.ZERO,
             new KeyValue(cube1_8X.ry.angleProperty(), 0.0)),
new KeyFrame(new Duration(endTime),
             new KeyValue(cube1_8X.ry.angleProperty(), -720.0)),
new KeyFrame(Duration.ZERO,
             new KeyValue(cube1_9X.ry.angleProperty(), 0.0)),
new KeyFrame(new Duration(endTime),
             new KeyValue(cube1_9X.ry.angleProperty(), -1080.0)),
new KeyFrame(Duration.ZERO,
             new KeyValue(cube1_1.rx.angleProperty(), 0.0)),
new KeyFrame(new Duration(endTime),
             new KeyValue(cube1_1.rx.angleProperty(), 7200.0)),
new KeyFrame(Duration.ZERO,
             new KeyValue(cube1_2.rx.angleProperty(), 0.0)),
new KeyFrame(new Duration(endTime),
             new KeyValue(cube1_2.rx.angleProperty(), -7200.0)),
new KeyFrame(Duration.ZERO,
             new KeyValue(cube1_3.rx.angleProperty(), 0.0)),
new KeyFrame(new Duration(endTime),
             new KeyValue(cube1_3.rx.angleProperty(), 7200.0)),
new KeyFrame(Duration.ZERO,
             new KeyValue(cube1_4.rx.angleProperty(), 0.0)),
new KeyFrame(new Duration(endTime),
             new KeyValue(cube1_4.rx.angleProperty(), -7200.0)),
new KeyFrame(Duration.ZERO,
             new KeyValue(cube1_5.rx.angleProperty(), 0.0)),
new KeyFrame(new Duration(endTime),
             new KeyValue(cube1_5.rx.angleProperty(), 7200.0)),
new KeyFrame(Duration.ZERO,
             new KeyValue(cube1_6.rx.angleProperty(), 0.0)),
new KeyFrame(new Duration(endTime),
             new KeyValue(cube1_6.rx.angleProperty(), -7200.0)),
```

```
     new KeyFrame(Duration.ZERO,
                  new KeyValue(cube1_7.rx.angleProperty(), 0.0)),
     new KeyFrame(new Duration(endTime),
                  new KeyValue(cube1_7.rx.angleProperty(), 7200.0)),
     new KeyFrame(Duration.ZERO,
                  new KeyValue(cube1_8.rx.angleProperty(), 0.0)),
     new KeyFrame(new Duration(endTime),
                  new KeyValue(cube1_8.rx.angleProperty(), -7200.0)),
     new KeyFrame(Duration.ZERO,
                  new KeyValue(cube1_9.rx.angleProperty(), 0.0)),
     new KeyFrame(new Duration(endTime),
                  new KeyValue(cube1_9.rx.angleProperty(), 7200.0))
     );
    animation.setCycleCount(Timeline.INDEFINITE);
    animation.play();

    return sceneRoot;
}

public void play() {
    animation.play();
}

@Override public void stop() {
    animation.pause();
}

class Xform extends Group {
    final Rotate rx = new Rotate(0, Rotate.X_AXIS);
    final Rotate ry = new Rotate(0, Rotate.Y_AXIS);
    final Rotate rz = new Rotate(0, Rotate.Z_AXIS);
    public Xform() {
        super();
        getTransforms().addAll(rz, ry, rx);
    }
}

public class Cube extends Group {
final Rotate rx = new Rotate(0,Rotate.X_AXIS);
final Rotate ry = new Rotate(0,Rotate.Y_AXIS);
```

```
final Rotate rz = new Rotate(0,Rotate.Z_AXIS);
public Cube(double size, Color color, double shade) {
getTransforms().addAll(rz, ry, rx);
getChildren().addAll(
    RectangleBuilder.create() // back face
        .width(size).height(size)
        .fill(color.deriveColor(0.0, 1.0, (1 - 0.5*shade), 1.0))
        .translateX(-0.5*size)
        .translateY(-0.5*size)
        .translateZ(0.5*size)
        .build(),
    RectangleBuilder.create() // bottom face
        .width(size).height(size)
        .fill(color.deriveColor(0.0, 1.0, (1 - 0.4*shade), 1.0))
        .translateX(-0.5*size)
        .translateY(0)
        .rotationAxis(Rotate.X_AXIS)
        .rotate(90)
        .build(),
    RectangleBuilder.create() // right face
        .width(size).height(size)
        .fill(color.deriveColor(0.0, 1.0, (1 - 0.3*shade), 1.0))
        .translateX(-1*size)
        .translateY(-0.5*size)
        .rotationAxis(Rotate.Y_AXIS)
        .rotate(90)
        .build(),
    RectangleBuilder.create() // left face
        .width(size).height(size)
        .fill(color.deriveColor(0.0, 1.0, (1 - 0.2*shade), 1.0))
        .translateX(0)
        .translateY(-0.5*size)
        .rotationAxis(Rotate.Y_AXIS)
        .rotate(90)
        .build(),
    RectangleBuilder.create() // top face
        .width(size).height(size)
        .fill(color.deriveColor(0.0, 1.0, (1 - 0.1*shade), 1.0))
        .translateX(-0.5*size)
        .translateY(-1*size)
```

```
            .rotationAxis(Rotate.X_AXIS)
            .rotate(90)
            .build(),
        RectangleBuilder.create() // top face
            .width(size).height(size)
            .fill(color)
            .translateX(-0.5*size)
            .translateY(-0.5*size)
            .translateZ(-0.5*size)
            .build()
);
    }
}

@Override public void start(Stage primaryStage) throws Exception {
    init(primaryStage);
    primaryStage.show();
    play();
}


public static void main(String[] args) {
    launch(args);
}
}//End of main class M73 CubeSystem3D
```

74 ***Exercise .*** *Run the previous program and play with the code.*

75 ***Exercise .*** *Use 2D squares to assemble a 3D brick that must be made into a class. Cure your code from any deprecated command.* Answer

76 ***Exercise .*** *Use 2D polygons to assemble a pyramid.* Answer

4.1 Interactive graphics

The joy of 3D graphic is that they can be rotated at will by the beholder. How this is achieved is illustrated by the next code.

77 ***An interactive molecule: it can be rotated with the mouse and translated forth or back with the help of the arrow keys of the keyboard. The code follows:***

```
/Interactive graphics

//program M77 MoleculeSampleApp
/*
 * Copyright (c) 2011, 2013 Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distri
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FO
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON AN
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE US
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE
 */

package moleculesampleapp;

import javafx.animation.Timeline;
import javafx.scene.PerspectiveCamera;
```

```
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.stage.Stage;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.Box;
import javafx.scene.shape.Cylinder;
import javafx.scene.shape.Sphere;
import javafx.scene.transform.Rotate;

import javafx.event.EventHandler;
import static javafx.scene.input.KeyCode.*;
import javafx.scene.input.KeyEvent;
import javafx.scene.input.MouseEvent;
import javafx.util.Duration;
import javafx.scene.Node;
import javafx.scene.transform.Scale;
import javafx.scene.transform.Translate;



/**
 * MoleculeSampleApp
 */




class Xform extends Group {

    public enum RotateOrder {
        XYZ, XZY, YXZ, YZX, ZXY, ZYX
    }

    public Translate t  = new Translate();
    public Translate p  = new Translate();
    public Translate ip = new Translate();
```

```
    public Rotate rx = new Rotate();
    { rx.setAxis(Rotate.X_AXIS); }
    public Rotate ry = new Rotate();
    { ry.setAxis(Rotate.Y_AXIS); }
    public Rotate rz = new Rotate();
    { rz.setAxis(Rotate.Z_AXIS); }
    public Scale s = new Scale();

    public Xform() {
        super();
        getTransforms().addAll(t, rz, ry, rx, s);
    }

    public Xform(RotateOrder rotateOrder) {
        super();
        // choose the order of rotations
        //based on the rotateOrder
        switch (rotateOrder) {
        case XYZ:
            getTransforms().addAll(t, p, rz, ry, rx, s, ip);
            break;
        case XZY:
            getTransforms().addAll(t, p, ry, rz, rx, s, ip);
            break;
        case YXZ:
            getTransforms().addAll(t, p, rz, rx, ry, s, ip);
            break;
        case YZX:
            getTransforms().addAll(t, p, rx, rz, ry, s, ip);  // For Ca
            break;
        case ZXY:
            getTransforms().addAll(t, p, ry, rx, rz, s, ip);
            break;
        case ZYX:
            getTransforms().addAll(t, p, rx, ry, rz, s, ip);
            break;
        }
    }

    public void setTranslate(double x, double y, double z) {
```

```
        t.setX(x);
        t.setY(y);
        t.setZ(z);
    }

    public void setTranslate(double x, double y) {
        t.setX(x);
        t.setY(y);
    }

    // Cannot override these methods as they are final:
    // public void setTranslateX(double x) { t.setX(x); }
    // public void setTranslateY(double y) { t.setY(y); }
    // public void setTranslateZ(double z) { t.setZ(z); }
    // Use these methods instead:
    public void setTx(double x) { t.setX(x); }
    public void setTy(double y) { t.setY(y); }
    public void setTz(double z) { t.setZ(z); }

    public void setRotate(double x, double y, double z) {
        rx.setAngle(x);
        ry.setAngle(y);
        rz.setAngle(z);
    }

    public void setRotateX(double x) { rx.setAngle(x); }
    public void setRotateY(double y) { ry.setAngle(y); }
    public void setRotateZ(double z) { rz.setAngle(z); }
    public void setRx(double x) { rx.setAngle(x); }
    public void setRy(double y) { ry.setAngle(y); }
    public void setRz(double z) { rz.setAngle(z); }

    public void setScale(double scaleFactor) {
        s.setX(scaleFactor);
        s.setY(scaleFactor);
        s.setZ(scaleFactor);
    }

    public void setScale(double x, double y, double z) {
        s.setX(x);
```

```
        s.setY(y);
        s.setZ(z);
    }

    // Cannot override these methods as they are final:
    // public void setScaleX(double x) { s.setX(x); }
    // public void setScaleY(double y) { s.setY(y); }
    // public void setScaleZ(double z) { s.setZ(z); }
    // Use these methods instead:
    public void setSx(double x) { s.setX(x); }
    public void setSy(double y) { s.setY(y); }
    public void setSz(double z) { s.setZ(z); }

    public void setPivot(double x, double y, double z) {
        p.setX(x);
        p.setY(y);
        p.setZ(z);
        ip.setX(-x);
        ip.setY(-y);
        ip.setZ(-z);
    }

    public void reset() {
        t.setX(0.0);
        t.setY(0.0);
        t.setZ(0.0);
        rx.setAngle(0.0);
        ry.setAngle(0.0);
        rz.setAngle(0.0);
        s.setX(1.0);
        s.setY(1.0);
        s.setZ(1.0);
        p.setX(0.0);
        p.setY(0.0);
        p.setZ(0.0);
        ip.setX(0.0);
        ip.setY(0.0);
        ip.setZ(0.0);
    }
```

```
    public void resetTSP() {
        t.setX(0.0);
        t.setY(0.0);
        t.setZ(0.0);
        s.setX(1.0);
        s.setY(1.0);
        s.setZ(1.0);
        p.setX(0.0);
        p.setY(0.0);
        p.setZ(0.0);
        ip.setX(0.0);
        ip.setY(0.0);
        ip.setZ(0.0);
    }
}


public class MoleculeSampleApp extends Application
{




    final Group root = new Group();
    final Group axisGroup = new Group();
    final Xform world = new Xform();
    final PerspectiveCamera camera
            = new PerspectiveCamera(true);
    final Xform cameraXform = new Xform();
    final Xform cameraXform2 = new Xform();
    final Xform cameraXform3 = new Xform();

    final double cameraDistance = 450;
    final Xform moleculeGroup = new Xform();



    private Timeline timeline;
    boolean timelinePlaying = false;
    double ONE_FRAME = 1.0/24.0;
```

```
double DELTA_MULTIPLIER = 200.0;
double CONTROL_MULTIPLIER = 0.1;
double SHIFT_MULTIPLIER = 0.1;
double ALT_MULTIPLIER = 0.5;

double mousePosX;
double mousePosY;
double mouseOldX;
double mouseOldY;
double mouseDeltaX;
double mouseDeltaY;



private void buildScene() {
    System.out.println("buildScene");
    root.getChildren().add(world);
}

 private void buildCamera() {
    root.getChildren().add(cameraXform);
    cameraXform.getChildren().add(cameraXform2);
    cameraXform2.getChildren().add(cameraXform3);
    cameraXform3.getChildren().add(camera);
    cameraXform3.setRotateZ(180.0);

    camera.setNearClip(0.1);
    camera.setFarClip(10000.0);
    camera.setTranslateZ(-cameraDistance);
    cameraXform.ry.setAngle(320.0);
    cameraXform.rx.setAngle(40);
}



private void buildAxes() {
    System.out.println("buildAxes()");
    final PhongMaterial redMaterial
            = new PhongMaterial();
```

```
        redMaterial.setDiffuseColor(Color.DARKRED);
        redMaterial.setSpecularColor(Color.RED);

        final PhongMaterial greenMaterial
                = new PhongMaterial();
        greenMaterial.setDiffuseColor(Color.DARKGREEN);
        greenMaterial.setSpecularColor(Color.GREEN);

        final PhongMaterial blueMaterial
                = new PhongMaterial();
        blueMaterial.setDiffuseColor(Color.DARKBLUE);
        blueMaterial.setSpecularColor(Color.BLUE);

        final Box xAxis = new Box(240.0, 1, 1);
        final Box yAxis = new Box(1, 240.0, 1);
        final Box zAxis = new Box(1, 1, 240.0);

        xAxis.setMaterial(redMaterial);
        yAxis.setMaterial(greenMaterial);
        zAxis.setMaterial(blueMaterial);

        axisGroup.getChildren().addAll(xAxis, yAxis, zAxis);
        world.getChildren().addAll(axisGroup);
    }



private void buildMolecule() {

        final PhongMaterial redMaterial
                = new PhongMaterial();
        redMaterial.setDiffuseColor(Color.DARKRED);
        redMaterial.setSpecularColor(Color.RED);

        final PhongMaterial whiteMaterial
                = new PhongMaterial();
        whiteMaterial.setDiffuseColor(Color.WHITE);
        whiteMaterial.setSpecularColor(Color.LIGHTBLUE);

        final PhongMaterial greyMaterial
```

```
        = new PhongMaterial();
greyMaterial.setDiffuseColor(Color.DARKGREY);
greyMaterial.setSpecularColor(Color.GREY);

// Molecule Hierarchy
// [*] moleculeXform
//     [*] oxygenXform
//         [*] oxygenSphere
//     [*] hydrogen1SideXform
//         [*] hydrogen1Xform
//             [*] hydrogen1Sphere
//         [*] bond1Cylinder
//     [*] hydrogen2SideXform
//         [*] hydrogen2Xform
//             [*] hydrogen2Sphere
//         [*] bond2Cylinder

Xform moleculeXform = new Xform();
Xform oxygenXform = new Xform();
Xform hydrogen1SideXform = new Xform();
Xform hydrogen1Xform = new Xform();
Xform hydrogen2SideXform = new Xform();
Xform hydrogen2Xform = new Xform();

Sphere oxygenSphere = new Sphere(40.0);
oxygenSphere.setMaterial(redMaterial);

Sphere hydrogen1Sphere = new Sphere(30.0);
hydrogen1Sphere.setMaterial(whiteMaterial);
hydrogen1Sphere.setTranslateX(0.0);

Sphere hydrogen2Sphere = new Sphere(30.0);
hydrogen2Sphere.setMaterial(whiteMaterial);
hydrogen2Sphere.setTranslateZ(0.0);

Cylinder bond1Cylinder = new Cylinder(5, 100);
bond1Cylinder.setMaterial(greyMaterial);
bond1Cylinder.setTranslateX(50.0);
bond1Cylinder.setRotationAxis(Rotate.Z_AXIS);
bond1Cylinder.setRotate(90.0);
```

```
        Cylinder bond2Cylinder = new Cylinder(5, 100);
        bond2Cylinder.setMaterial(greyMaterial);
        bond2Cylinder.setTranslateX(50.0);
        bond2Cylinder.setRotationAxis(Rotate.Z_AXIS);
        bond2Cylinder.setRotate(90.0);

        moleculeXform.getChildren().add(oxygenXform);
        moleculeXform.getChildren().add(hydrogen1SideXform);
        moleculeXform.getChildren().add(hydrogen2SideXform);
        oxygenXform.getChildren().add(oxygenSphere);
        hydrogen1SideXform.getChildren().add(hydrogen1Xform);
        hydrogen2SideXform.getChildren().add(hydrogen2Xform);
        hydrogen1Xform.getChildren().add(hydrogen1Sphere);
        hydrogen2Xform.getChildren().add(hydrogen2Sphere);
        hydrogen1SideXform.getChildren().add(bond1Cylinder);
        hydrogen2SideXform.getChildren().add(bond2Cylinder);

        hydrogen1Xform.setTx(100.0);
        hydrogen2Xform.setTx(100.0);
        hydrogen2SideXform.setRotateY(104.5);

        moleculeGroup.getChildren().add(moleculeXform);

        world.getChildren().addAll(moleculeGroup);
    }


//
// The handleCameraViews file contains
//the handleMouse() and handleKeyboard()
// methods that are used in the
//MoleculeSampleApp application to handle the
// different 3D camera views.
//These methods are used in the Getting Started with
// JavaFX 3D Graphics tutorial.
//

private void handleMouse(Scene scene, final Node root) {
        scene.setOnMousePressed(new EventHandler<MouseEvent>() {
```

```
        @Override public void handle(MouseEvent me) {
            mousePosX = me.getSceneX();
            mousePosY = me.getSceneY();
            mouseOldX = me.getSceneX();
            mouseOldY = me.getSceneY();
        }
    });
    scene.setOnMouseDragged(new EventHandler<MouseEvent>()
    {
        @Override public void handle(MouseEvent me) {
            mouseOldX = mousePosX;
            mouseOldY = mousePosY;
            mousePosX = me.getSceneX();
            mousePosY = me.getSceneY();
            mouseDeltaX = (mousePosX - mouseOldX);
            mouseDeltaY = (mousePosY - mouseOldY);

            double modifier = 1.0;
            double modifierFactor = 0.1;

            if (me.isControlDown()) {
                modifier = 0.1;
            }
            if (me.isShiftDown()) {
                modifier = 10.0;
            }
            if (me.isPrimaryButtonDown()) {
                cameraXform.ry.setAngle(cameraXform
                .ry.getAngle()
                    - mouseDeltaX*modifierFactor
                            *modifier*2.0);  // +
                cameraXform.rx.setAngle(cameraXform
                .rx.getAngle()
                + mouseDeltaY*modifierFactor
                        *modifier*2.0);  // -
            }
            else if (me.isSecondaryButtonDown()) {
                double z = camera.getTranslateZ();
                double newZ = z
                    + mouseDeltaX*modifierFactor*modifier;
```

```
                    camera.setTranslateZ(newZ);
                }
                else if (me.isMiddleButtonDown()) {
                    cameraXform2.t.setX(cameraXform2.t.getX()
                    + mouseDeltaX*modifierFactor*modifier*0.3);  // -
                    cameraXform2.t.setY(cameraXform2.t.getY()
                    + mouseDeltaY*modifierFactor*modifier*0.3);  // -
                }
            }
        });
    }

private void handleKeyboard(Scene scene, final Node root)
{
    final boolean moveCamera = true;
    scene.setOnKeyPressed(new EventHandler<KeyEvent>()
    {
        @Override
        public void handle(KeyEvent event) {
            Duration currentTime;
            switch (event.getCode()) {
                case Z:
                    if (event.isShiftDown()) {
                        cameraXform.ry.setAngle(0.0);
                        cameraXform.rx.setAngle(0.0);
                        camera.setTranslateZ(-300.0);
                    }
                    cameraXform2.t.setX(0.0);
                    cameraXform2.t.setY(0.0);
                    break;
                case X:
                    if (event.isControlDown()) {
                        if (axisGroup.isVisible()) {
                            System.out.println(
                                    "setVisible(false)");
                            axisGroup.setVisible(false);
                        }
                        else {
                            System.out.println(
                                    "setVisible(true)");
```

```
                axisGroup.setVisible(true);
            }
        }
        break;
    case S:
        if (event.isControlDown()) {
            if (moleculeGroup.isVisible()) {
                moleculeGroup.setVisible(false);
            }
            else {
                moleculeGroup.setVisible(true);
            }
        }
        break;
    case SPACE:
        if (timelinePlaying) {
            timeline.pause();
            timelinePlaying = false;
        }
        else {
            timeline.play();
            timelinePlaying = true;
        }
        break;
    case UP:
        if (event.isControlDown()
                && event.isShiftDown()) {
            cameraXform2.t
            .setY(cameraXform2.t.getY()
                    - 10.0*CONTROL_MULTIPLIER);
        }
        else if (event.isAltDown()
                && event.isShiftDown()) {
            cameraXform.rx.setAngle(cameraXform
                .rx.getAngle()
                    - 10.0*ALT_MULTIPLIER);
        }
        else if (event.isControlDown()) {
            cameraXform2.t.setY(
            cameraXform2.t.getY()
```

```
                    - 1.0*CONTROL_MULTIPLIER);
                }
                else if (event.isAltDown()) {
                    cameraXform.rx.setAngle(
                            cameraXform.rx.getAngle()
                                    - 2.0*ALT_MULTIPLIER);
                }
                else if (event.isShiftDown()) {
                    double z = camera.getTranslateZ();
                    double newZ = z
                            + 5.0*SHIFT_MULTIPLIER;
                    camera.setTranslateZ(newZ);
                }
                break;
            case DOWN:
                if (event.isControlDown() && event
                        .isShiftDown()) {
                    cameraXform2.t.setY(
                            cameraXform2.t.getY()
                            + 10.0*CONTROL_MULTIPLIER);
                }
                else if (event.isAltDown()
                        && event.isShiftDown()) {
                    cameraXform.rx.setAngle(
                            cameraXform.rx.getAngle()
                            + 10.0*ALT_MULTIPLIER);
                }
                else if (event.isControlDown())
                {
                    cameraXform2.t.setY(cameraXform2
                    .t.getY() + 1.0*CONTROL_MULTIPLIER);
                }
                else if (event.isAltDown()) {
                    cameraXform.rx.setAngle(
                        cameraXform.rx.getAngle()
                        + 2.0*ALT_MULTIPLIER);
                }
                else if (event.isShiftDown()) {
                    double z = camera.getTranslateZ();
                    double newZ = z
```

```
                        - 5.0*SHIFT_MULTIPLIER;
                camera.setTranslateZ(newZ);
            }
            break;
        case RIGHT:
            if (event.isControlDown() && event
                    .isShiftDown()) {
                cameraXform2.t.setX(cameraXform2
                .t.getX() + 10.0*CONTROL_MULTIPLIER);
            }
            else if (event.isAltDown()
                    && event.isShiftDown()) {
                cameraXform.ry.setAngle(cameraXform
                .ry.getAngle()
                        - 10.0*ALT_MULTIPLIER);
            }
            else if (event.isControlDown()) {
                cameraXform2.t.setX(cameraXform2
                .t.getX() + 1.0*CONTROL_MULTIPLIER);
            }
            else if (event.isAltDown()) {
                cameraXform.ry.setAngle(cameraXform
                .ry.getAngle() - 2.0*ALT_MULTIPLIER);
            }
            break;
        case LEFT:
            if (event.isControlDown()
                    && event.isShiftDown()) {
                cameraXform2.t.setX(cameraXform2
                .t.getX() - 10.0*CONTROL_MULTIPLIER);
            }
            else if (event.isAltDown()
                    && event.isShiftDown()) {
                cameraXform.ry.setAngle(cameraXform
                .ry.getAngle() + 10.0*ALT_MULTIPLIER);  // -
            }
            else if (event.isControlDown()) {
                cameraXform2.t.setX(cameraXform2
                    .t.getX() - 1.0*CONTROL_MULTIPLIER);
            }
```

```
                    else if (event.isAltDown()) {
                        cameraXform.ry.setAngle(cameraXform
                        .ry.getAngle() + 2.0*ALT_MULTIPLIER);  // -
                    }
                    break;
                }
            }
        });
    }


    @Override
    public void start(Stage primaryStage) {
        System.out.println("start");
        //buildScene();

        Scene scene = new Scene(root, 1024, 768, true);
        scene.setFill(Color.GREY);
                handleKeyboard(scene, world);
        handleMouse(scene, world);

        primaryStage.setTitle("Drag the mouse to rotate "
            + "and hold shift + arrow keys "
                + "to get closer or farther");
        primaryStage.setScene(scene);
        primaryStage.show();
        buildScene();
        buildCamera();
        buildAxes();
        buildMolecule();
        primaryStage.show();
        scene.setCamera(camera);

    }


    public static void main(String[] args) {
        System.setProperty("prism.dirtyopts", "false");
        launch(args);
```

```
    }
}//End of main class M77 MoleculeSampleApp
```

78 ***Exercise .*** *Run the previous program and play with the code.*

79 ***Antialliasing.*** *The next code shows an interactive graphics to illustrate the technique of antialiasing to smooth unintended crappy curves.*

```
//Program L79 MSAAApp
//Interactive graphics that shows
//anti-aliasing technique to smotth
//unintended crappy curves.


/*
* Copyright (c) 2013, 2014 Oracle and/or its affiliates.
* All rights reserved. Use is subject to license terms.
*
* This file is available and licensed under the following license:
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
* are met:
*
* - Redistributions of source code must retain the above copyright
*
notice, this list of conditions and the following disclaimer.
* - Redistributions in binary form must reproduce the above copyright
*
notice, this list of conditions and the following disclaimer in
*
the documentation and/or other materials provided with the distributio
* - Neither the name of Oracle nor the names of its
*
contributors may be used to endorse or promote products derived
*
from this software without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
```

```
* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
* OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
* DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
* THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/


package msaaapp;


import javafx.application.Application;
import javafx.application.ConditionalFeature;
import javafx.application.Platform;
import javafx.geometry.Point3D;
import javafx.scene.AmbientLight;
import javafx.scene.Camera;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.Parent;
import javafx.scene.PerspectiveCamera;
import javafx.scene.PointLight;
import javafx.scene.Scene;
import javafx.scene.SceneAntialiasing;
import javafx.scene.SubScene;
import javafx.scene.control.Slider;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.Cylinder;
import javafx.scene.text.Font;
import javafx.scene.text.Text;
import javafx.stage.Stage;
```

```
public class MSAAApp extends Application {
@Override
public void start(Stage stage) {
    if (!Platform.isSupported(ConditionalFeature.SCENE3D)) {
    throw new RuntimeException("*** ERROR: common "
            + "conditional SCENE3D is not supported");
    }
    stage.setTitle("JavaFX MSAA demo");
    Group root = new Group();
    Scene scene = new Scene(root, 1000, 800);
    scene.setFill(Color.color(0.2, 0.2, 0.2, 1.0));
    HBox hbox = new HBox();
    hbox.setLayoutX(75);
    hbox.setLayoutY(200);
    PhongMaterial phongMaterial
            = new PhongMaterial(Color.color(1.0, 0.7,
    0.8));
    Cylinder cylinder1 = new Cylinder(100, 200);
    cylinder1.setMaterial(phongMaterial);
    SubScene noMsaa = createSubScene(
            "MSAA = false", cylinder1,  Color.TRANSPARENT,
    new PerspectiveCamera(), false);
    hbox.getChildren().add(noMsaa);
    Cylinder cylinder2 = new Cylinder(100, 200);
    cylinder2.setMaterial(phongMaterial);
    SubScene msaa = createSubScene("MSAA = true",
            cylinder2,  Color.TRANSPARENT,
    new PerspectiveCamera(), true);
    hbox.getChildren().add(msaa);
    Slider slider = new Slider(0, 360, 0);
    slider.setBlockIncrement(1);
    slider.setTranslateX(425);
    slider.setTranslateY(625);
    cylinder1.rotateProperty().bind(slider.valueProperty());
    cylinder2.rotateProperty().bind(slider.valueProperty());
    root.getChildren().addAll(hbox, slider);
    stage.setScene(scene);
```

```
    stage.show();
}


private static Parent setTitle(String str) {
    final VBox vbox = new VBox();
    final Text text = new Text(str);
    text.setFont(Font.font("Times New Roman", 24));
    text.setFill(Color.WHEAT);
    vbox.getChildren().add(text);
    return vbox;
}


private static SubScene createSubScene(String title,
        Node node, Color fillPaint, Camera camera,
        boolean msaa) {
    Group root = new Group();
    PointLight light = new PointLight(Color.WHITE);
    light.setTranslateX(50);
    light.setTranslateY(-300);
    light.setTranslateZ(-400);
    PointLight light2 = new PointLight(
            Color.color(0.6, 0.3, 0.4));
    light2.setTranslateX(400);
    light2.setTranslateY(0);
    light2.setTranslateZ(-400);
    AmbientLight ambientLight = new
            AmbientLight(Color.color(0.2, 0.2, 0.2));
    node.setRotationAxis(new Point3D(2, 1, 0).normalize());
    node.setTranslateX(180);
    node.setTranslateY(180);
    root.getChildren().addAll(setTitle(title), ambientLight,
    light, light2, node);
    SubScene subScene = new SubScene(root, 500, 400, true,
    msaa ? SceneAntialiasing.BALANCED :
            SceneAntialiasing.DISABLED);
    subScene.setFill((javafx.scene.paint.Paint) fillPaint);
    subScene.setCamera(camera);
    return subScene;
```

```
}
/**
* @param args the command line arguments
*/
public static void main(String[] args) {
launch(args);
}
}//End of main class M79 MSAAApp
```

80 ***Exercise .*** *Run the previous program and play with the code.*

4.2 Conclusion

JavaFX 3D graphics facility is wonderful, rich and complex. Nevertheless, the strategy of making 3D shapes using 2D panes as raw material is not productive. Something else must be invented.

Chapter 5

Meshes

`Personalized, detailed 3D shapes`

81 ***Introduction.*** *Using 2D polygons as covering panes to build 3D shapes has proven to be an extremely cumbersome strategy. So, a question arises: Is there something to diminish the burden of creating personalized 3D shapes without killing the possibility of professionalism?*

82 ***Objective.*** *JavaFX invites every one to build personalized 3D shapes with a sophisticated tool:* ***meshes*** *that are approximation to 3D shapes by means of triangles. So, our purpose is to learn here how to use them to build 3D graphics.*

As we will see, professionalism is an imprinting of the implemented approach. This means that learning is difficult but we try our best to alleviate the ensuing burden. So, we begin with a cube.

5.1 A box as a mesh

A mesh in JavaFX is a concept for professionals, so, we will need some steps to understand it. To begin with, we will work a mesh for a brick.

83 ***The idea.*** *A* ***mesh*** *is a set of triangles that conform to a triangulation (an approximation by triangles) of a 3D shape.*

84 ***Let us show how the mesh for a brick is encoded in JavaFX.***

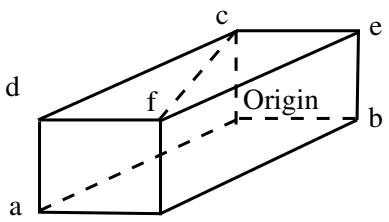


Figure 5.0. To define a mesh, we declare in first place the vertexes. Each vertex is a point with 3 coordinates. A triangle is just a set of three vertexes, say, the upper face is composed of two triangles. One is fcd and the other is fec.

To specify the set of vertexes, we give just a list of their coordinates. So, a vertex will be known by its place in that list (lets keep in mind that Java enumerates entries beginning from 0 not from 1). Now, it is customary to make the center of the object coincide with the center of the system of coordinates.

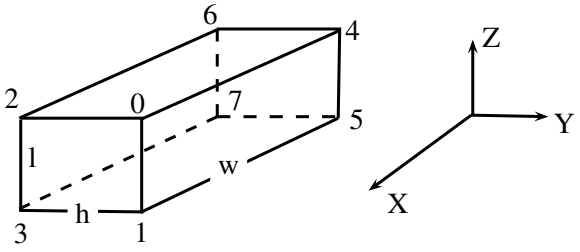


Figure 5.1. We see a box with measures w (x-coordinate), h (y-coordinate) and l (z-coordinate). The name of a vertex will be its place in the list of vertexes. If the center of coordinates coincides with that of the box, the coordinates of point 0 will be (w/2,h/2,l/2), those of vertex 3 will be (w/2,-h/2, -l/2), those of vertex 4 will be (-w/2, h/2, l/2) and so on.

The following code shows us a meshed box under JavaFX:

```
//Program M84 MeshBrick
//A mesh is made by JavaFX into a cube.

/*
* Copyright (c) 2013, Pro JavaFX Authors
```

```
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
* 1. Redistributions of source code must retain the above copyright notice,
* this list of conditions and the following disclaimer.
* 2. Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
* 3. Neither the name of JFXtras nor the names of its contributors may be used
* to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
* POSSIBILITY OF SUCH DAMAGE.
*
* MeshCubePickDemo.fx - A simple example of JavaFX 3D
*
* Developed 2013 by James L. Weaver jim.weaver [at] javafxpert.com
* as a JavaFX 8 example for the Pro JavaFX 8 book.

*/

//Adapted by JR (EvolJavaCommunity), 2014.

package meshbrick;

import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;
```

```
import javafx.geometry.Insets;
import javafx.scene.*;

import javafx.scene.control.Label;
import javafx.scene.control.Slider;
import javafx.scene.layout.HBox;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.MeshView;
import javafx.scene.shape.TriangleMesh;
import javafx.scene.transform.Rotate;
import javafx.stage.Stage;


public class MeshBrick extends Application {

//For mouse listener
double anchorX, anchorY;

//Length of edge of a cube
private static final float EDGE_LENGTH = 380;

//For listener of mouse
private double anchorAngleX = 0;
private double anchorAngleY = 0;

//Angle of rotation
private final DoubleProperty angleX
        = new SimpleDoubleProperty(25);
private final DoubleProperty angleY
        = new SimpleDoubleProperty(40);

PerspectiveCamera camera = new PerspectiveCamera(false);

//Slider for zooming
private Slider zSlider;
//Position of cube along the z-axis, normal to the screen.
//It is a double property because
//it will get involved into a binding
private final DoubleProperty zPos
```

```
        = new SimpleDoubleProperty(-200);


//Outputs a box with measures w, h and l.
static TriangleMesh createMesh(float w, float h, float l)
{


    //Halves of measures.
    //Type double is not accepted,
    //instead float or int are allowed
    float hw = w / 2;
    float hh = h / 2;
    float hl = l / 2f; //f stands for type float (optional)


    //The mesh is declared and instantiated
    TriangleMesh mesh = new TriangleMesh();
    //Declaration of the points of the mesh.
    //Each point has 3 coordinates.
    //There is neither a tag for  points
    //or a special separator in within points!
    mesh.getPoints().addAll
    (
        hw, hh, hl, //point 0
        hw, hh, -hl, //point 1
        hw, -hh, hl, //point 2
        hw, -hh, -hl, //point 3
        -hw, hh, hl, //point 4
        -hw, hh, -hl, //point 5
        -hw, -hh, hl, //point 6
        -hw, -hh, -hl //point 7
    );

    //This is for future use
    mesh.getTexCoords().addAll(0,0);

    /* List of triangles:

    //We have 12 triangles, two for each face of the box.
```

```
//Each triangle is defined by a triade of points:
//The upper face has two triangles:
//The first is conformed by points 0, 4 and 2.
//The second by points 2, 4, 6.

0,   2,   1, //Frontal face
2,   3,   1,
4,   5,   6, //Back face
6,   5,   7,
0,   1,   4, //Right face
4,   1,   5,
2,   6,   3, //Left face
3,   6,   7,
0,   4,   2, //Upper face
2,   4,   6,
1,   3,   5, //Bottom face
5,   3,   7

*/

//The official form as  the list of triangles
//must be declared for a mesh in JavaFX
//demands additional information that we just fill
//with a 0 intercalated after each vertex.
mesh.getFaces().addAll
(
    //Frontal face with vertexes 0,2, 1.
    0, 0, 2, 0, 1, 0, //Frontal face
    2, 0, 3, 0, 1, 0,
    4, 0, 5, 0, 6, 0, //Back face
    6, 0, 5, 0, 7, 0,
    0, 0, 1, 0, 4, 0, //Right face
    4, 0, 1, 0, 5, 0,
    2, 0, 6, 0, 3, 0, //Left face
    3, 0, 6, 0, 7, 0,
    0, 0, 4, 0, 2, 0, //Upper face
    2, 0, 4, 0, 6, 0,
    1, 0, 3, 0, 5, 0, //Bottom face
    5, 0, 3, 0, 7, 0
);
```

```
  return mesh;
}


@Override
public void start(Stage primaryStage)
{
    primaryStage.setTitle("MeshBrick: drag the mouse to rotate");



    //A box is declared and instantiated as  a cube
    MeshView cube =
      new MeshView(createMesh(EDGE_LENGTH
               , EDGE_LENGTH, EDGE_LENGTH));
    PhongMaterial earthMaterial
            = new PhongMaterial(Color.GOLDENROD);
    cube.setMaterial(earthMaterial);

    //Initialization
    final Group parent = new Group(cube);
    parent.setTranslateX(450);
    parent.setTranslateY(400);
    parent.setTranslateZ(0);
    Rotate xRotate;
    Rotate yRotate;
    parent.getTransforms().setAll
    (
      xRotate = new Rotate(0, Rotate.X_AXIS),
      yRotate = new Rotate(0, Rotate.Y_AXIS)
    );
    xRotate.angleProperty().bind(angleX);
    yRotate.angleProperty().bind(angleY);
    final Group root = new Group(parent);
    final Scene scene = new Scene(root, 900, 900, true);
    scene.setFill(Color.WHITE);

    //Listener of mousePressed
    scene.setOnMousePressed
```

```
(
    event ->
    {
      anchorX = event.getSceneX();
      anchorY = event.getSceneY();
      anchorAngleX = angleX.get();
      anchorAngleY = angleY.get();

    }
);

//Listener of mouseDragged
scene.setOnMouseDragged
(
    event ->
    {
        angleX.set(anchorAngleX
                - (anchorY - event.getSceneY()));
        angleY.set(anchorAngleY
                + anchorX - event.getSceneX());
    }
);

//Illumination
PointLight pointLight = new PointLight(Color.WHITE);
pointLight.setTranslateX(300);
pointLight.setTranslateY(200);
pointLight.setTranslateZ(-2000);
root.getChildren().add(pointLight);
//Camera
scene.setCamera(camera);
primaryStage.setScene(scene);
// SubScene code
PerspectiveCamera subSceneCamera
        = new PerspectiveCamera(false);
scene.setCamera(subSceneCamera);
Label zLabel = new Label("Zoom");
zSlider = new Slider(-200, 3000, -200);

//Box for the slider and its label
```

```
    HBox subSceneGroup = new HBox(70,
            new HBox(10, zLabel, zSlider));
    subSceneGroup.setPadding(new Insets(100, 100, 0, 100));
    SubScene subScene
            = new SubScene(subSceneGroup, 880, 200);
    subScene.setLayoutX(10);
    subScene.setLayoutY(720);

    root.getChildren().add(subScene);

    //To zoom down the brick,
    //it is translated away from  the beholder,
    //along the Z-direction (normal to the screen).
    //Change zPos to change initial setting.
    zSlider.valueProperty().bindBidirectional(zPos);
    parent.translateZProperty().bind(zPos);

    primaryStage.show();
  }

  /**
* @param args the command line arguments
*/
  public static void main(String[] args)
  {
    launch(args);
  }
}//End of main class M84 MeshBrick
```

85 ***Exercise .*** *Run the previous program and play with the code.*

5.2 Wallpapers

We have learned how to draw a JavaFX 3D graphic by resorting to a (triangular) mesh. Every face of a mesh can be given the color one desires. Besides this, JavaFX has another tool to paint shapes: they can be covered with wallpapers that can be engraved with whatever image. Let us see.

86 ***How to make a wallpaper.*** *Just take an image, whatever.*

87 ***How to give a coordinate system to a wallpaper.*** *All wallpapers have one and only one system of coordinates. It is 2D and to each point it gives two coordinates. But beware: the width of the image in the JavaFX system of coordinates for wallpapers is 1 as also its height. So, every wallpaper is seen as a square. Therefore, points in the wallpaper has coordinates in within 0 and 1.*

88 ***How to assign a winding covering for you 3D graphic.*** *There are infinitely many procedures to wind a wallpaper around a 3D shape. We will illustrate one and after this one probably will imagine many variants.*

The simplest procedure to define a winding of a wallpaper for a 3D shape is the following:

1. Unfold your 3D shape as a 2D graphic. This new 2D graphic must be an unfolded copy of the mesh.
2. Shrink or expand the unfolded 2D graphic until occupying the unitary square.
3. Mark some points over the wallpaper as to match the points given by the mesh. Use the appropriate formalism to pack this correspondence.

In relation with the cube, this procedure is illustrated with the following graphics. We have in first place the unitary square with its marked points:

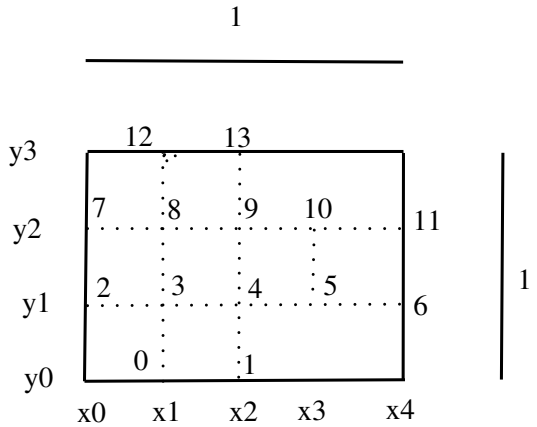


Figure 5.2. Instructions to wind a squared wallpaper to cover a 3d shape that here is an unfolded cube. We see the unit square that represents how the wallpaper is seen by our system of coordinates. There is inside it an unfolded cube with its

6 faces forming a cross. Every vertex is given a number as a name. This number defines the order as the coordinates of the vertexes must be declared. For instance, the coordinates of point 0 are (x0,y0). Those of vertex 8 are (x1, y2), those of vertex 11 are (x4, y2) and so on. The value of xi is $i/4$. That of yi is $i/3$.

We also have the same unfolded cube but with a copy of the mesh:

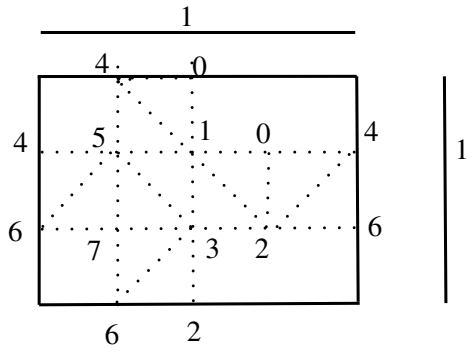


Figure 5.3. The unfolded cube with the triangular mesh. If you wind this figure, you get the original cube with the correct name of its vertexes. We see that vertex 7 of the cube matches vertex 3 of the unitary cube. This is why, to define the mesh, vertex 3 is followed by vertex 7, the first is the vertex of the 3D graphic, the second is the vertex of the wallpaper or texture. By the same token, vertex 0 of the cube corresponds with vertex 10 in the frontal face, but with vertex 13 of the right face. So, in the list of vertexes of the frontal face, 0 is followed by 10, while in the right face, 0 is followed by 13. And so on.

89 ***A cube with wallpaper, whose corresponding file has been attached to the project, is encoded as follows:***

```
//Program M89 MeshCubePickDemo
//A mesh is made by JavaFX into a cube.
//The cube is covered with wallpaper.
```

```
/*
* Copyright (c) 2013, Pro JavaFX Authors
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions ar
* 1. Redistributions of source code must retain the above copyright not
* this list of conditions and the following disclaimer.
* 2. Redistributions in binary form must reproduce the above copyright
* this list of conditions and the following disclaimer in the documenta
* and/or other materials provided with the distribution.
* 3. Neither the name of JFXtras nor the names of its contributors may
* to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PU
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSIN
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWIS
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED O
* POSSIBILITY OF SUCH DAMAGE.
*
* MeshCubePickDemo.fx - A simple example of JavaFX 3D
*
* Developed 2013 by James L. Weaver jim.weaver [at] javafxpert.com
* as a JavaFX 8 example for the Pro JavaFX 8 book.

*/

//Adapted by JR (evoljava.com), 2014.
```

```
package meshcubepickdemo;

import javafx.application.Application;
import javafx.beans.binding.When;
import javafx.beans.property.BooleanProperty;
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleBooleanProperty;
import javafx.beans.property.SimpleDoubleProperty;
import javafx.geometry.Insets;
import javafx.scene.*;
import javafx.scene.control.CheckBox;
import javafx.scene.control.Label;
import javafx.scene.control.Slider;
import javafx.scene.image.Image;

import javafx.scene.input.PickResult;
import javafx.scene.layout.HBox;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.DrawMode;
import javafx.scene.shape.MeshView;

import javafx.scene.shape.TriangleMesh;
import javafx.scene.transform.Rotate;
import javafx.stage.Stage;

/**
*
* @author of original program: Jim, javafxpert.com
*/

public class MeshCubePickDemo extends Application
{

//For mouse listener
double anchorX, anchorY;

//Length of edge of a cube
private static final float EDGE_LENGTH = 380;
```

```
//For listener of mouse
private double anchorAngleX = 0;
private double anchorAngleY = 0;

//Angle of rotation
private final DoubleProperty angleX
        = new SimpleDoubleProperty(25);
private final DoubleProperty angleY
        = new SimpleDoubleProperty(40);

PerspectiveCamera camera
        = new PerspectiveCamera(false);

//Slider for zooming
private Slider zSlider;
//Position of cube along the z-axis, normal to the screen.
//It is a double property because
//it will get involved into a binding
private final DoubleProperty zPos
        = new SimpleDoubleProperty(-200);



private CheckBox meshCheckBox;
//Show mesh?
private final BooleanProperty showWireframe
        = new SimpleBooleanProperty(false);

private Label facePickedLabel = new Label();

  static TriangleMesh createMesh(float w, float h, float l)
  {

    if (w * h * l == 0)
    {
      return null;
    }

    //Halves of measures.
    //Type double is not accepted,
```

```
//instead float or int are allowed
float hw = w / 2;
float hh = h / 2;
float hl = l / 2f; //f stands for type float (optional)



//The mesh is declared and instantiated
TriangleMesh mesh = new TriangleMesh();
//Declaration of the points of the mesh.
//Each point has 3 coordinates.
//There is neither a tag for  points
//or a special separator in within points!
mesh.getPoints().addAll
    (
        hw, hh, hl, //point 0
        hw, hh, -hl, //point 1
        hw, -hh, hl, //point 2
        hw, -hh, -hl, //point 3
        -hw, hh, hl, //point 4
        -hw, hh, -hl, //point 5
        -hw, -hh, hl, //point 6
        -hw, -hh, -hl //point 7
    );

//The cube will be covered by wallpaper,
//which is marked with points.
//Each point of the wallpaper has two coordinates.
//Coordinates go from 0 to 1 in both dimensions.
//So, wallpaper is a rectangular image
//but it is seen here  as a square.

//Coordinates for the marking points of wallpaper.
//These points serve to form triangles
//that are next used to cover triangles of the mesh.
float x0 = 0f;
float x1 = 1f / 4f;//this means x1 = 1/4 but with decimals
float x2 = 2f / 4f;
float x3 = 3f / 4f;
float x4 = 1f;
```

```
float y0 = 0f;
float y1 = 1f / 3f;
float y2 = 2f / 3f;
float y3 = 1f;


//Points in wallpaper are distingushed by the order
//as they appear in the following list.
//Say, point 0 has coordinates (x1,y0)
//point 3 has coordinates (x1,y1) and so on.
//Order is the only separator of points.
mesh.getTexCoords().addAll(
  x1, y0, //Point 0 of wallpaper
  x2, y0,
  x0, y1,
  x1, y1,//Point 3 of wallpaper
  x2, y1,
  x3, y1,
  x4, y1,
  x0, y2,
  x1, y2,
  x2, y2,
  x3, y2,
  x4, y2,
  x1, y3,
  x2, y3////Point 13 of wallpaper
);

/* List of triangles:

//We have 12 triangles, two for each face of the box.
//Each triangle is defined by a triade of points:
//The upper face has two triangles:
//The first is conformed by points 0, 4 and 2.
//The second by points 2, 4, 6.

0,   2,   1, //Frontal face
2,   3,   1,
4,   5,   6, //Back face
6,   5,   7,
```

```
0,   1,   4, //Right face
4,   1,   5,
2,   6,   3, //Left face
3,   6,   7,
0,   4,   2, //Upper face
2,   4,   6,
1,   3,   5, //Bottom face
5,   3,   7

*/

//The official form as  the list of triangles
//must be declared for a mesh in JavaFX
//demands additional information in regard
//as the form as the covering wallpaper
//must be glued to the cube. To that aim,
//after listing a vertex of the cube,
//a vertex of the wallpaper is also attached.
//So,
//0, 10, 2, 5, 1, 9, //Frontal face
//means:
//Triangle 0, 2, 1 of the mesh
//must be covered with triangle
//10, 5, 9 of the wallpaper,
//in the cited order.

mesh.getFaces().addAll(
    //Frontal face with vertexes 0,2, 1.
    0, 10, 2, 5, 1, 9, //Frontal face
    2, 5, 3, 4, 1, 9,
    4, 7, 5, 8, 6, 2, //Back face
    6, 2, 5, 8, 7, 3,
    0, 13, 1, 9, 4, 12, //Right face
    4, 12, 1, 9, 5, 8,
    2, 1, 6, 0, 3, 4, //Left face
    3, 4, 6, 0, 7, 3,
    0, 10, 4, 11, 2, 5, //Upper face
    2, 5, 4, 11, 6, 6,
    1, 9, 3, 4, 5, 8, //Bottom face
    5, 8, 3, 4, 7, 3
```

```
    );



    /*
    //The following grouping renders the faces of the cube
    //with one normal:
    //the sun is reflected homogeneously
    //along the two triangles.
    int smoothingGroups[]
            = { 0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 5, 5};
    */

    //This grouping renders triangles
    //of the same face of the cube
    //with slightly different normals:
    //the sun is reflected inhomogeneously.
    int smoothingGroups[]
            = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11};


    mesh.getFaceSmoothingGroups().addAll(smoothingGroups);

    return mesh;
  }

@Override
public void start(Stage primaryStage)
{
    primaryStage.setTitle("MeshCubePickDemo");

    MeshView cube =
      new MeshView(createMesh(EDGE_LENGTH,
              EDGE_LENGTH, EDGE_LENGTH));




    //Wallpaper is gotten
    Image wallpaper = new Image(getClass()
```

```
        .getResourceAsStream("Beijing.png"));
//Declaration of material for painting the cube
PhongMaterial material = new PhongMaterial();
//Sun
material.setSpecularColor(Color.WHEAT);
//Wallpaper is declared as covering material
material.setDiffuseMap(wallpaper);
//Cube is covered with wallpaper
cube.setMaterial(material);

//The cube as JavaFX spacelike object
final Group parent = new Group(cube);
parent.setTranslateX(450);
parent.setTranslateY(400);
parent.setTranslateZ(0);
Rotate xRotate;
Rotate yRotate;
parent.getTransforms().setAll(
  xRotate = new Rotate(0, Rotate.X_AXIS),
  yRotate = new Rotate(0, Rotate.Y_AXIS)
);
xRotate.angleProperty().bind(angleX);
yRotate.angleProperty().bind(angleY);
//Cube is attached to root
final Group root = new Group(parent);
final Scene scene = new Scene(root, 900, 900, true);
//Background
scene.setFill(Color.WHITE);

//Listener of mousePressed
scene.setOnMousePressed(event -> {
  anchorX = event.getSceneX();
  anchorY = event.getSceneY();
  anchorAngleX = angleX.get();
  anchorAngleY = angleY.get();

  PickResult pr = event.getPickResult();

  facePickedLabel.setText("Face picked: "
      + pr.getIntersectedFace());
```

```
});

//Listener of mouseDragged
scene.setOnMouseDragged
(
    event ->
    {
        angleX.set(anchorAngleX
                - (anchorY - event.getSceneY()));
        angleY.set(anchorAngleY
                + anchorX - event.getSceneX());
    }
);

//Illumination
PointLight pointLight = new PointLight(Color.WHITE);
pointLight.setTranslateX(300);
pointLight.setTranslateY(200);
pointLight.setTranslateZ(-2000);

root.getChildren().add(pointLight);
//Camera
scene.setCamera(camera);
primaryStage.setScene(scene);
PerspectiveCamera subSceneCamera
        = new PerspectiveCamera(false);
scene.setCamera(subSceneCamera);

//To zoom down the brick,
//it is translated away from  the beholder,
//along the Z-direction (normal to the screen).
//Change zPos to change initial setting.
zSlider = new Slider(-200, 3000, -200);
//Bind zPos to the reading of the slider
zSlider.valueProperty().bindBidirectional(zPos);
//Execute zoom
parent.translateZProperty().bind(zPos);

//Decide what to show: filled cube else mesh
cube.drawModeProperty().bind(new When(showWireframe)
```

```
            .then(DrawMode.LINE).otherwise(DrawMode.FILL));

    //Other JavaFX controls, labels and containers
    meshCheckBox = new CheckBox("Wireframe");
    Label zLabel = new Label("Zoom");
    meshCheckBox.selectedProperty()
            .bindBidirectional(showWireframe);
    HBox subSceneGroup = new HBox(70,
        new HBox(10, zLabel, zSlider)
            , meshCheckBox, facePickedLabel);
    subSceneGroup.setPadding(new Insets(100, 100, 0, 100));
    SubScene subScene
            = new SubScene(subSceneGroup, 880, 200);
    subScene.setLayoutX(10);
    subScene.setLayoutY(720);
    root.getChildren().add(subScene);

    primaryStage.show();
  }

  /**
* @param args the command line arguments
*/
  public static void main(String[] args)
  {
    launch(args);
  }
}//End of main class M89 MeshCubePickDemo
```

90 ***Exercise .*** *Run the previous program and play with the code.*

5.3 Orientation

Three points define a triangle of a mesh but they must be listed observing certain rule. Otherwise one might get not what one desires. Let us see.

91 ***Triangles are bifaced in JavaFX.*** *A triangular face in JavaFX is always a surface with two sides. The next code explains what it means.*

```
//Program M91 BifacedTriangle
//A JavaFX 3D triangle has two faces.


package bifacedtriangle;

import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;
import javafx.event.EventHandler;
import javafx.geometry.Insets;
import javafx.scene.Group;
import javafx.scene.PerspectiveCamera;
import javafx.scene.PointLight;
import javafx.scene.Scene;
import javafx.scene.SubScene;
import javafx.scene.control.Label;
import javafx.scene.control.Slider;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.HBox;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.MeshView;
import javafx.scene.shape.TriangleMesh;
import javafx.scene.transform.Rotate;
import javafx.stage.Stage;


public class BifacedTriangle extends Application
{


//For mouse listener
double anchorX, anchorY, anchorAngle;

//Length of edge of a triangle
private static final float EDGE_LENGTH = 200;

//For listener of mouse
```

```
private double anchorAngleX = 0;
private double anchorAngleY = 0;

//Angle of rotation
private final DoubleProperty angleX
        = new SimpleDoubleProperty(25);
private final DoubleProperty angleY
        = new SimpleDoubleProperty(40);

PerspectiveCamera camera = new PerspectiveCamera(false);

//Slider for zooming
private Slider zSlider;
//Position of triangle along the z-axis, normal to the screen.
//It is a double property because
//it will get involved into a binding
private final DoubleProperty zPos
        = new SimpleDoubleProperty(-100);


//Outputs a triangle in 3D
static TriangleMesh createMesh(float w, float h, float l)
{


    //Halves of measures.
    //Type double is not accepted,
    //instead float or int are allowed
    float hw = w / 2;
    float hh = h / 2;
    float hl = l / 2f; //f stands for type float (optional)


    //The mesh is declared and instantiated
    TriangleMesh mesh = new TriangleMesh();
    //Declaration of the points of the mesh.
    //Each point has 3 coordinates.
    //There is neither a tag for  points
    //or a special separator in within points!
```

```
mesh.getPoints().addAll
(

    -hw, hh, hl, //point 0
    hw, hh, hl, //point 1
    0, h, -hl //point 2

);



mesh.getTexCoords().addAll(0,0);



/*
//This is the brilliant face alone
mesh.getFaces().addAll
(
    //Frontal face with vertexes 0,2, 1.
    0, 0, 2, 0, 1, 0
);
*/



/*
//This is the dark face alone
mesh.getFaces().addAll
(

    //Rear face with vertexes 0, 1, 2.
    0, 0, 1, 0, 2, 0
);

*/
```

```
    //This is a bifaced triangle
    mesh.getFaces().addAll
    (
        //Frontal face with vertexes 0,2, 1.
        //It is brilliant
        0, 0, 2, 0, 1, 0,
        //Rear face with vertexes 0, 1, 2.
        //It is dark
        0, 0, 1, 0, 2, 0
    );


  return mesh;
}


@Override
public void start(Stage primaryStage)
{
    primaryStage.setTitle("Click and drag the mouse");



    //A triangle is declared and instantiated
    MeshView triangle =
      new MeshView(createMesh(EDGE_LENGTH
               , EDGE_LENGTH, EDGE_LENGTH));
    PhongMaterial earthMaterial
            = new PhongMaterial(Color.GOLDENROD);
    triangle.setMaterial(earthMaterial);

    //Initialization
    final Group parent = new Group(triangle);
    parent.setTranslateX(150);
    parent.setTranslateY(100);
    parent.setTranslateZ(0);
    Rotate xRotate;
    Rotate yRotate;
    parent.getTransforms().setAll
```

```
    (
      xRotate = new Rotate(0, Rotate.X_AXIS),
      yRotate = new Rotate(0, Rotate.Y_AXIS)
    );
    xRotate.angleProperty().bind(angleX);
    yRotate.angleProperty().bind(angleY);
    final Group root = new Group(parent);
    final Scene scene = new Scene(root, 400, 400, true);
    scene.setFill(Color.WHITE);

   scene.setOnMousePressed(new EventHandler<MouseEvent>()
   {
            @Override public void handle(MouseEvent event)
            {
                anchorX = event.getSceneX();
                anchorY = event.getSceneY();
                anchorAngle = triangle.getRotate();
            }
    });

    scene.setOnMouseDragged(new EventHandler<MouseEvent>()
    {
        @Override public void handle(MouseEvent event)
        {
            triangle.setRotate(anchorAngle
                    + anchorX - event.getSceneX());
        }
    });

    //Illumination
    PointLight pointLight = new PointLight(Color.WHITE);
    pointLight.setTranslateX(300);
    pointLight.setTranslateY(200);
    pointLight.setTranslateZ(-2000);
    root.getChildren().add(pointLight);
    //Camera
    scene.setCamera(camera);
    primaryStage.setScene(scene);
    // SubScene code
    PerspectiveCamera subSceneCamera
```

```
            = new PerspectiveCamera(false);
    scene.setCamera(subSceneCamera);
    Label zLabel = new Label("Zoom");
    zSlider = new Slider(-200, 3000, -200);

    //Box for the slider and its label
    HBox subSceneGroup = new HBox(70,
            new HBox(10, zLabel, zSlider));
    subSceneGroup.setPadding(new Insets(100, 100, 0, 100));
    SubScene subScene
            = new SubScene(subSceneGroup, 880, 200);
    subScene.setLayoutX(10);
    subScene.setLayoutY(720);

    root.getChildren().add(subScene);

    //To zoom down the brick,
    //it is translated away from  the beholder,
    //along the Z-direction (normal to the screen).
    //Change zPos to change initial setting.
    zSlider.valueProperty().bindBidirectional(zPos);
    parent.translateZProperty().bind(zPos);

    primaryStage.show();
  }

  /**
* @param args the command line arguments
*/
  public static void main(String[] args)
  {
    launch(args);
  }
}//End of main class M91 BifacedTriangle
```

92 ***Exercise .*** *Run the previous program and play with the code in which a triangle in 3D is shown. Notice that every triangle is bi-faced: it has frontal and rear faces. There is three options to define the mesh. Activate each one of them. One is conformed by the frontal face of the triangle, the second by the rear face, and the third has both faces. The difference between frontal and rear faces is the order as*

the vertexes are listed. If a face is not listed, it is invisible.

The order as one lists the vertexes of a triangle is denominated **orientation**.

93 ***Rules to orientate triangles.***

There are two rules that determine how to list the vertexes of triangles in an order that is adequate for JavaFX 3D:

Rule of the watch over the 3D shape: imagine that you see your 3D graphic with its triangulation. The problem is how to list vertexes that resultant triangles could be seen exactly as you have planed. So, imagine that you posit a watch with handles over the 3D shape and inside the triangle you want to encode. Now, pick up a vertex of your triangle, whatever, and list it , list next the other two vertexes following a direction that is counterclockwise. Repeat the same procedure over every vertex. Your shape will be a collection of one-faced triangles with the correct orientation.

Rule for the 2D unfolded shape: While the 3D shape presents a look that is visible from its outside, the unfolded 2D figure may present a look from inside or from outside. Choose one point of view and adapt your orientation consequently.

94 ***Exercise.*** *Over the cube of this chapter, check the fairness of the aforementioned two rules for orientation.*

5.4 Exercises

We need to practice what we have learned. So, let us solve some exercises.

95 ***Exercise.*** *Develop ab initio your own version of a brick as a Java object. Exhibit the correspondence between vertexes of the 3D shape and the marked points of the wallpaper. Include appropriate graphics.* Answer

96 ***Exercise.*** *Develop a pyramid from a mesh as a Java object. Make suitable drawings to easy reading by third parts.* Answer

97 ***Let us develop an application that draws a function in two variables, say,*** $f(x,y) = x^2 + y^2$.

The code that does it follows:

```
//Program M97 XyToF
//Draws a real function in two variables, say
//z = f(x,y) = x^2 + y^2.
//The function is evaluated
//over a square of its domain = R^2
//of edge 2l.
//Its  graph is allocated inside a cube.
package xytof;




import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;
import javafx.geometry.Insets;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.PerspectiveCamera;
import javafx.scene.PointLight;
import javafx.scene.Scene;
import javafx.scene.SubScene;
import javafx.scene.control.Label;
import javafx.scene.control.Slider;
import javafx.scene.layout.HBox;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.CullFace;
import javafx.scene.shape.DrawMode;
import javafx.scene.shape.MeshView;
import javafx.scene.shape.TriangleMesh;
import javafx.scene.transform.Rotate;
import javafx.stage.Stage;



public class XyToF extends Application
```

```
{

    //Function to be graphicated.
    //It has two independent variables x and y.
    static private float function(float x, float y)
    {
        //Paraboloid
        float function = x*x + y*y;

        //Plane
        //float function = x + 3*y;

        //Saddle
        //float function = x*x - y*y;

        return function;
    }


    //Edge of the cube of the presentation
    private static float halfEdgeCube = 300;
    //The function is evaluated over a square
    //in the plane XY, centered at (0,0)
    //and of edge 2*l.
    final static float l = 200;

    //The maximal value of the function
    //determines the scale in the z-direction
    final static float maxValueOfFunction = 0;

    //Slider for zooming
    private Slider zSlider;
    //Position of brick along the z-axis, normal to the screen.
    //It is a double property because
    //it will get involved into a binding
    private final DoubleProperty zPos
            = new SimpleDoubleProperty(2000);

    //For listener of mouse
    private double anchorAngleX = 0;
```

```
    private double anchorAngleY = 0;
    private double anchorX, anchorY;
    //Angles of rotation
    private final DoubleProperty angleX
            = new SimpleDoubleProperty(0);
    private final DoubleProperty angleY
            = new SimpleDoubleProperty(0);

    PerspectiveCamera camera
            = new PerspectiveCamera(false);




private void init(Stage primaryStage)
{
    primaryStage.setTitle("Function graphicator");
    //The whole drawing  is created
    //as a JavaFX space-like object
    final Group parent = new Group(create3dContent());

    //Initial position
    parent.setTranslateX(450);
    parent.setTranslateY(400);
    parent.setTranslateZ(0);
    Rotate xRotate;
    Rotate yRotate;
    //Initialization of rotation
    parent.getTransforms().setAll
    (
      xRotate = new Rotate(0, Rotate.X_AXIS),
      yRotate = new Rotate(0, Rotate.Y_AXIS)
    );

    //Brick is attached to root
    final Group root = new Group(parent);
    //Window of the scene
    final Scene scene = new Scene(root, 900, 900, true);
    //Background
```

```
scene.setFill(Color.SKYBLUE);

//Listener of mousePressed
scene.setOnMousePressed
(
    event ->
    {
        anchorX = event.getSceneX();
        anchorY = event.getSceneY();
        anchorAngleX = angleX.get();
        anchorAngleY = angleY.get();
    }
);


//Listener of mouseDragged
scene.setOnMouseDragged
(
    event ->
    {
        angleX.set(anchorAngleX
                - (anchorY - event.getSceneY()));
        angleY.set(anchorAngleY
                + anchorX - event.getSceneX());
    }
);

//Rotation gets bound to
//the angle determined by  mouseDragged
xRotate.angleProperty().bind(angleX);
yRotate.angleProperty().bind(angleY);

//Illumination
PointLight pointLight = new PointLight(Color.WHITE);
pointLight.setTranslateX(100);
pointLight.setTranslateY(200);
pointLight.setTranslateZ(-2000);
root.getChildren().add(pointLight);

//Camera
```

```
    scene.setCamera(camera);
    primaryStage.setScene(scene);
    PerspectiveCamera subSceneCamera
            = new PerspectiveCamera(false);
    scene.setCamera(subSceneCamera);

    //To zoom down the brick,
    //it is translated away from  the beholder,
    //along the Z-direction (normal to the screen).
    //Change zPos to change initial setting.
    zSlider = new Slider();
    zSlider.setMax(5000);
    zSlider.setMin(1);


    //Bind zPos to the reading of the slider
    zSlider.valueProperty().bindBidirectional(zPos);
    //Execute negative zoom
    parent.translateZProperty().bind(zPos);
    System.out.println("zPos = " + zPos);


    //Other JavaFX controls, labels and containers
    Label zLabel = new Label("Zoom");
    HBox subSceneGroup = new HBox(70,
        new HBox(10, zLabel, zSlider));

    subSceneGroup.setPadding(new Insets(10, 100, 0, 100));
    SubScene subScene
            = new SubScene(subSceneGroup, 880, 200);
    subScene.setLayoutX(10);
    subScene.setLayoutY(800);
    root.getChildren().add(subScene);

    primaryStage.show();
}


//Manager of graphic
public Node create3dContent()
```

```
{
    //Cube with enclosed grid of floor
    //and diffuse graph of function.
    //Edge = 2*halfEdgeCube.
    Graphic b = new Graphic(2*halfEdgeCube
            ,2*halfEdgeCube,2*halfEdgeCube);
    b.rx.setAngle(45);
    b.ry.setAngle(110);

    return new Group(b);
}

public void play()
{

}

@Override public void stop()
{

}

//This class encloses a function with a cube.
public class Graphic extends Group
{
final Rotate rx = new Rotate(0,Rotate.X_AXIS);
final Rotate ry = new Rotate(0,Rotate.Y_AXIS);
final Rotate rz = new Rotate(0,Rotate.Z_AXIS);
//The ambient of the graphic is a box
//of measures a, b, and c.
public  Graphic(float a, float b, float c)
  {
    getTransforms().addAll(rz, ry, rx);
    if ((a * b * c == 0) || (a < 0) || (b < 0) ||(c< 0))
    {
      return;
    }

    //Halves of measures.
    //Type double is not accepted,
```

```
//instead float or int are allowed
float ha = a / 2;
float hb = b / 2;
float hc = c / 2f; //f stands for type float (optional)

//The mesh is declared and instantiated
TriangleMesh mesh = new TriangleMesh();
//Declaration of the points of the mesh.
//Each point has 3 coordinates.
//There is neither a tag for  points
//or a special separator in within points!
mesh.getPoints().addAll
    (
        ha, hb, hc, //point 0
        ha, hb, -hc, //point 1
        ha, -hb, -hc, //point 2
        ha, -hb, hc, //point 3
        -ha, hb, hc, //point 4
        -ha, hb, -hc, //point 5
        -ha, -hb, -hc, //point 6
        -ha, -hb, hc //point 7
    );




//Points in wallpaper are distingushed by the order
//as they appear in the following list.
//Say, point 0 has coordinates (x1,y0)
//point 3 has coordinates (x1,y1) and so on.
//Order is the only separator of points.
mesh.getTexCoords().addAll(0,0);


/* List of triangles:

//We have 12 triangles, two for each face of the box.
//Each triangle is defined by a triade of points:
5,6,4, //Rear
```

```
7,4,6,
7,6,3,//Left
2,3,6,
4,7,0,//Upper
3,0,7,
5,4,1, //Right
0,1,4,
0,3,1, //Front
2,1,3,
1,2,5, //Bottom
6,5,2


*/

//Mesh with symbolic texture points
mesh.getFaces().addAll(
//Rear face with vertexes 5, 6, 4
//must be covered with triangle
//13, 12, 10 of the wallpaper,
//in the cited order.
5, 0, 6, 0, 4, 0, //Rear
7, 0, 4, 0, 6, 0,
7, 0, 6, 0, 3, 0, //Left
2, 0, 3, 0, 6, 0,
4, 0, 7, 0, 0, 0, //Upper
3, 0, 0, 0, 7, 0,
5, 0, 4, 0, 1, 0,//Right
0, 0, 1, 0, 4, 0,
0, 0, 3, 0, 1, 0, //Front
2, 0, 1, 0, 3, 0,
1, 0, 2, 0, 5, 0,//Bottom
6, 0, 5, 0, 2, 0
);



//The following grouping renders the
//triangles of same face
```

```
//with one normal:
//the sun is reflected homogeneously
//along the two triangles.
int smoothingGroups[]
        = { 0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 5, 5};



mesh.getFaceSmoothingGroups().addAll(smoothingGroups);

MeshView brick = new MeshView(mesh);


//The brick as JavaFX spacelike object
final Group parent = new Group(brick);
getChildren().addAll(parent);


//The brick is shown as line
brick.drawModeProperty().set(DrawMode.LINE);


//TriangleMesh of the floor
TriangleMesh meshFloor
        = buildTriangleMesh(1, 10, halfEdgeCube);
PhongMaterial material = new PhongMaterial();
material.setDiffuseColor(Color.FUCHSIA);
//material.setSpecularColor(Color.rgb(30, 30, 30));
MeshView meshView = new MeshView(meshFloor);
meshView.setTranslateZ(0);
meshView.setMaterial(material);
//Set Wireframe mode
meshView.setDrawMode(DrawMode.LINE);
meshView.setCullFace(CullFace.NONE);

//Mesh of the graph of the function
TriangleMesh meshFunction
        = buildMeshFunction(1, 10, halfEdgeCube);
PhongMaterial materialF  = new PhongMaterial();
```

```
    materialF.setDiffuseColor(Color.BLUEVIOLET);
    MeshView  meshViewFunction = new MeshView(meshFunction);
    meshViewFunction.setMaterial(materialF);
    meshViewFunction.setCullFace(CullFace.NONE);
    meshViewFunction.setTranslateZ(0);

    //Illumination
    PointLight pointLight;
    pointLight = new PointLight(Color.ANTIQUEWHITE);
    pointLight.setTranslateX(150);
    pointLight.setTranslateY(-100);
    pointLight.setTranslateZ(-1000);


    getChildren().addAll(meshView,meshViewFunction,pointLight  );

  }
}//End of class Graphic




//The mesh of the floor = domain of the function:
//it is square of edge 2l and with n subdivisions
//Adpated from a code by Oracle.
static TriangleMesh buildTriangleMesh(float l,
                    int n,  float halfEdgeCube)
{
    //The domain can change but the cube not.
    //This line makes the adjustment.
    float xyScale = halfEdgeCube / l;

    //Array with coordinates of points.
    //Each point has 3 coordinates
    float points[] = new float[3* (n+1) * (n +1)];
    //Edge of each tiny rectangle over the floor
    float delta = 2*l/n;

    //Each rectangle has 2 triangles
    int nFaces = 2*n*n;
```

```
//Each face has 6 coordinates,
//3 for the points plus
//3 for the marked points of the wallpaper
int faces[] = new int[nFaces * 6];

// Create vertexes

for (int j = 0; j <= n; j++)
{
    //y goes from -l to l by increments of delta
    float yj = -l + j * delta;

    for (int i = 0; i <= n; i++)
    {
        //x goes from -l to l by increments of delta
        float xi = -l + i * delta;
        //Coordinates of vertexes are numbered.
        //(n+1)* 3 coordinates for each row
        int index = j * (n+1)* 3 + (i * 3);
        //x-coordinate in graphic space of vertex
        points[index] = (float) xi * xyScale;
        //y-coordinate in graphic space of vertex
        points[index + 1] = (float) yj * xyScale;
        //z-coordinate of the floor = 0
        points[index + 2] = 0.0f;
    }
}


//Faces are built.
//Two triangles for each rectangle.
//Each face or triangle is composed of three points.
//These are determined  by pointing
//to their position in the vector points[].
//The declaration of no wallpaper is done
//by associating a zero to each vertex.
for (int j = 0; j < n; j++)
{
    for (int i = 0; i < n; i++)
    {
```

```
            //Rectangle ij has four vertexes
            //whose indexes in vector points[] are
            //p00, p01, p10, p11.
            //Points at row j
            int p00 = j * (n+1) + i;
            int p01 = p00 + 1;
            //Points one row above
            int p10 = p00 + (n+1);
            int p11 = p10 + 1;
            //No texture, no wallpaper
            int tc00 = 0;
            int tc01 = 0;
            int tc10 = 0;
            int tc11 = 0;
            //First triangle
            int index = (j * n * 6 + (i * 6)) * 2;
            faces[index + 0] = p00;
            faces[index + 1] = tc00;
            faces[index + 2] = p10;
            faces[index + 3] = tc10;
            faces[index + 4] = p11;
            faces[index + 5] = tc11;
            //Second triangle
            index = index + 6;
            faces[index + 0] = p11;
            faces[index + 1] = tc11;
            faces[index + 2] = p01;
            faces[index + 3] = tc01;
            faces[index + 4] = p00;
            faces[index + 5] = tc00;
        }
    }


    TriangleMesh mesh = new TriangleMesh();
    mesh.getPoints().addAll(points);

    mesh.getTexCoords().addAll(0,0);
    mesh.getFaces().addAll(faces);
```

```
    return mesh;
}

//The function is evaluated over every vertex
//of the grid of the domain
//and its maximal value is catched and returned.
//Domain is a square of size 2l and has n subdivisions
//on each side.
static float findMaxValueFunction(float l, int n )
{
    //Length of side of tiny rectangle
    float delta = 2*l/n;

    float max = 0;
    float function;
    //Evaluation at every vertex
    for (int j = 0; j <= n; j++)
    {
        float yj = -l + j * delta;

        for (int i = 0; i <= n; i++)
        {
            float xi = -l + i * delta;
            function = function(xi,yj);
            //Max is catched
            if ( max < Math.abs(function)) max = function;
        }
    }
    return max;
}
```

```
//The mesh of the graph of the function
//is a vertical deformation of the mesh
//of the square of the floor.
static TriangleMesh buildMeshFunction(float l, int n,  float halfEdgeCube)
```

```
{

    //The graph must fit the cube. So,
    //the z scale is calculated
    float zMax = findMaxValueFunction(l, n);
    float zScale =   halfEdgeCube/zMax;
    float xyScale =  halfEdgeCube / l;

    //Array with coordinates of points of
    //the mesh of the floor, each one with 3 coordinates
    float points[] = new float[3* (n+1) * (n +1)];
    //Size of tiny rect
    float delta = 2*l/n;

    //There are n*n tiny rectangles.
    //Each rectangle has 2 triangles
    int nFaces = 2*n*n;
    //Each point of each face has 6 coordinates,
    //3 for the vertex and
    //3 for the marked points of the wallpaper
    int faces[] = new int[nFaces * 6];

    // Create points
    for (int j = 0; j <= n; j++)
    {
        float yj = -l + j * delta;

        for (int i = 0; i <= n; i++)
        {
            float xi = -l + i * delta;

            int index = j * (n+1)* 3 + (i * 3);
            points[index] = (float) xi * xyScale;
            points[index + 1] = (float) yj * xyScale;
            points[index + 2] = function(xi,yj)*zScale;
        }
    }


    //Faces are built.
```

```
//two triangles for each rectangle
for (int j = 0; j < n; j++)
{
    for (int i = 0; i < n; i++)
    {
        int p00 = j * (n+1) + i;
        int p01 = p00 + 1;
        int p10 = p00 + (n+1);
        int p11 = p10 + 1;
        int tc00 = 0;
        int tc01 = 0;
        int tc10 = 0;
        int tc11 = 0;
        //First triangle
        int index = (j * n * 6 + (i * 6)) * 2;
        faces[index + 0] = p00;
        faces[index + 1] = tc00;
        faces[index + 2] = p10;
        faces[index + 3] = tc10;
        faces[index + 4] = p11;
        faces[index + 5] = tc11;
        //second triangle?
        index = index + 6;
        faces[index + 0] = p11;
        faces[index + 1] = tc11;
        faces[index + 2] = p01;
        faces[index + 3] = tc01;
        faces[index + 4] = p00;
        faces[index + 5] = tc00;
    }
}


TriangleMesh mesh = new TriangleMesh();
mesh.getPoints().addAll(points);

mesh.getTexCoords().addAll(0,0);
mesh.getFaces().addAll(faces);

return mesh;
```

```
}

@Override public void start(Stage primaryStage)
        throws Exception
{
    init(primaryStage);
    primaryStage.show();
    play();
}


public static void main(String[] args)
{
    launch(args);
}
}//End of main class M97 XyToF
```

98 ***Exercise .*** *Run the program and play with the code. The previous program draws the graph of a function over a squared region that is centered at zero. Extend this capability to allow the region to be center around whatever point of its domain.*
Answer

99 ***Challenge.*** *To understand what perfection is and how much it costs, make and effort to convert the previous program into a professional application. This is a tacit challenge for all our programs.*

5.5 Conclusion

We have learned how to build our own JavaFX 3d objects. This personalization is done through meshes, ensembles of triangles. A JavaFX triangle is set of three points that is given an orientation. So, triangles are bifaced, meaning that to see one of its faces, one must declare it. The other face results from swapping the order of two points whatever. Meshes can be covered with wallpapers or textures. The specification of the covering must be done triangle per triangle so, a great potential of personalization exists.

Chapter 6

Mologrames

`Meshes that encode for 3D photographies`

100 ***Motivation and purpose.*** *3D-meshes were introduced in the previous chapter to personalize pictures. The problem is that it is rather impossible to define a mesh by hand if only it is a bit complicated. Nevertheless, automatic procedures to do this have been invented. One of such methods is to make a 3D-photography and compute an associated* ***mologram****, a mesh that encodes for a a 3D image of an object. Here we will learn how to download and work with mologrames from Internet.*

6.1 Downloading mologrames

Three dimensional animation and photography has now turned into a big industry with very high standards which is mainly fueled by the industry of 3D movies. In regard with personal use, by 2015, a good set of popular mologrames may cost 100. A 3D printing of a single mologram, as a printed 3D portrait, may cost 50 euros. Nevertheless, many stores also offer some models for free. Scientists on their side also have tried to popularized for free as methods as models. Thanks to this effort, it will be possible for us to learn here how to use a bank of mologrames of the male human body. Precisely, we will download the bones of a hand and next we will learn how to make the skeleton of a hand out of them.

This section is guaranteed because it is a prerequisite for further and essential work: if given instructions do not function, write me (jose@evoljava.com) to negotiate a description of truly functional procedures.

101 ***Target website.***

We are interesting in the project *BodyParts3D* of the Faculty of Engineering of The University Of Tokyo [4].

102 ***Exercise.*** *Link to cited website and scrabble about it, in particular, pay attention to videos, licenses and warnings.*

103 ***Downloading.***

We need the bones of the left hand. To get the corresponding files:

1. Connect to BodyParts3D/Anatomography at [3]
2. Look for numbers in orange colors that mark special windows, whose borders can be shifted. You shall visualize window number 1 (else, look for it with the help of great blue arrows).
3. Choose **Data version** (upper left). An old version will suffice for us: 4.0
4. In the **window 1**, marked in orange color, check **HAS PART TREE** and select the tab: **Tree**. A tree with body parts must pop up. Go to **Left upper limb** and expand it. Select **Left free upper limb**.
5. Some images must appear on window 2. Select the image **Left hand**.
6. To the right of window 2 there is another window with various tabs. Select **Bodypart3D**. Click over **[download obj files]**. A new window must pop up with the list of 19 objects, phalanges and metacarpals. Activate downloading by selecting the box **check all** at the upper left. **C**lick the Download icon, on the bottom right. Save your file to an appropriate place.
7. On window 2 check the image of the Left wrist. Repeat previous procedure to download a file with all the 8 carpals of the wrist.
8. Save and unzip files to an appropriate folder.

6.2 Displaying mologrames

We have the files that contain mologrames of the hand but, how to display them? This is what we will learn in this section. The crucial point is how to read, interpret and import the meshes. An option that seems the simplest one to some people is to reuse an app developed by *InteractiveMesh.org* [10]:

That is not our option. We have chosen to reinvent the wheel. This promises to be difficult but our point is that mesh technology is a must for everyone in our community. Therefore, it is worth to become well prepared. So, let us begin.

104 ***Exercise.*** *Open one of the files with the information for the mologram of some bone of the hand. Whatever. These files are written in plain text so, they can be read by a word processor, say,* ***Text Editor*** *of Ubuntu. Their names have* ***.obj*** *as suffix. This format is an invention of* `Wavefront Technologies`*, [17], a computer graphics company that developed and sold animation software used in Hollywood motion pictures. The company declared the .obj format [14], [15] as open. Thanks a lot.*

Verify that the official format is not the one used here but a dialect. The author made the following inference: a comment is marked with `#`*, a vertex with* `v`*, the normal to a face by* `vn`*, the vertexes that conform a face by*

`f + double back slashes and repetitions.`

Study that document to make your own inferences and agree else disagree with the Author.

105 ***Let us test the inference made in the last exercise: a preliminary program to import the meshes encoded by the downloaded files follows.***

```
/*
    Program M105 MlgrmLeftHand
    This program imports a mesh that is encoded
    in a file whose name ends with suffix .obj.
    Content is shown in a textArea.
    A preliminary drawing is made in line mode.
    We have two troubles:
    1.  The file contains too many faces,
        more than are allowed by JavaFX 3D.
    2.  In JavaFX 3D all triangles must be
        correctly oriented, with normals pointing outwards,
        but imported meshes seemingly do not care about that.


    Tested only with some  files of the BodyParts3D project.
    Those files are licensed:

    The license for this database is specified in the
```

```
    Creative Commons Attribution-Share Alike 2.1 Japan.
    If you use data from this database, please be sure
    attribute this database as follows: "BodyParts3D, (c)
    The Database Center for Life Science licensed under
    CC Attribution-Share Alike 2.1 Japan".
    http://dbarchive.biosciencedbc.jp/en/bodyparts3d/lic.html

 */

package mlgrmlefthand;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;
import javafx.event.ActionEvent;
import javafx.geometry.Insets;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.PerspectiveCamera;
import javafx.scene.PointLight;
import javafx.scene.Scene;
import javafx.scene.SubScene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.ScrollPane;
import javafx.scene.control.Slider;
import javafx.scene.control.TextArea;
import javafx.scene.input.PickResult;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
```

```
import javafx.scene.shape.Box;
import javafx.scene.shape.DrawMode;
import javafx.scene.shape.MeshView;
import javafx.scene.shape.TriangleMesh;
import javafx.scene.transform.Rotate;
import javafx.stage.FileChooser;
import javafx.stage.Stage;



public class MlgrmLeftHand extends Application
{




    //Slider for zooming
    private Slider zSlider;
    //Position of graphic along the z-axis,
    //normal to the screen.
    //It is a double property because
    //it will get involved into a binding
    private final DoubleProperty zPos
            = new SimpleDoubleProperty(0);




    //For listener of mouse
    private double anchorAngleX = 0;
    private double anchorAngleY = 0;
    private double anchorX, anchorY;
    //Angles of rotation of the scene
    private final DoubleProperty angleX
            = new SimpleDoubleProperty(0);
    private final DoubleProperty angleY
            = new SimpleDoubleProperty(0);

    PerspectiveCamera camera
```

```
            = new PerspectiveCamera(false);



    private static final String title
            =  "Importer of meshes in format .obj";

    //Tool to read files
    BufferedReader br ;
    private static  String textOfFile = "";
    //The mesh is declared and instantiated
    private static final  TriangleMesh mesh  = new TriangleMesh();



public void init(Stage primaryStage)
{
    primaryStage.setTitle(title);


    //======File loader ======

    /*Adapted from Java-Buddy:
    http://java-buddy.blogspot.com/2012/05/read-text-file-with-javafx-
    */

    //Content of file will be shown in textArea
    final TextArea textArea = new TextArea();
    textArea.prefWidth(400);
    textArea.setWrapText(true);

    //Container
    ScrollPane scrollPane = new ScrollPane();
    scrollPane.getStyleClass().add("noborder-scroll-pane");
    scrollPane.setContent(textArea);
    scrollPane.setFitToWidth(true);
    scrollPane.setPrefWidth(400);
    scrollPane.setPrefHeight(180);
```

```
//Button to load file
Button buttonLoad = new Button("Load file");
//Listener
buttonLoad.setOnAction((ActionEvent arg0) ->
{
    FileChooser fileChooser = new FileChooser();

    //Name of file with suffix .obj
    FileChooser.ExtensionFilter extFilter
            = new FileChooser
                    .ExtensionFilter("TXT files (*.obj)"
                            , "*.obj");
    fileChooser.getExtensionFilters().add(extFilter);

    //Show open file dialog
    File file = fileChooser.showOpenDialog(primaryStage);
    try {
            textOfFile = readFile(file);
        } catch (IOException ex) {
            Logger.getLogger(MlgrmLeftHand
                    .class.getName())
                    .log(Level.SEVERE, null, ex);
        }
    //Text copied to textArea
    textArea.setText(textOfFile);


});



VBox vBox = new VBox();
vBox.getChildren().addAll(scrollPane);
//====== End of File loader

MeshView bone =
  new MeshView(mesh);
PhongMaterial goldenMaterial
        = new PhongMaterial(Color.GOLDENROD);
bone.setMaterial(goldenMaterial);
```

```
bone.scaleXProperty().setValue(15);
bone.scaleYProperty().setValue(15);
bone.scaleZProperty().setValue(15);

bone.drawModeProperty().set(DrawMode.LINE);


//The whole drawing  is created
//as a JavaFX space-like object
final Group parent = new Group( bone);


//Initial position
parent.setTranslateX(450);
parent.setTranslateY(400);
parent.setTranslateZ(0);
parent.visibleProperty().set(true);


Rotate xRotate;
Rotate yRotate;
//Initialization of rotation
parent.getTransforms().setAll
(
  xRotate = new Rotate(0, Rotate.X_AXIS),
  yRotate = new Rotate(0, Rotate.Y_AXIS)
);

//Graphic is attached to root
final Group root = new Group(parent);
//Window of the scene
final Scene scene = new Scene(root, 900, 900, true);
//Background
scene.setFill(Color.SKYBLUE);

//Listener of mousePressed
scene.setOnMousePressed
(
    event ->
```

```
        {
            anchorX = event.getSceneX();
            anchorY = event.getSceneY();
            anchorAngleX = angleX.get();
            anchorAngleY = angleY.get();
        }
    );


    //Listener of mouseDragged
    scene.setOnMouseDragged
    (
        event ->
        {
            angleX.set(anchorAngleX
                    - (anchorY - event.getSceneY()));
            angleY.set(anchorAngleY
                    + anchorX - event.getSceneX());
        }
    );

    //Rotation of the scene gets bound to
    //the angle determined by  mouseDragged
    xRotate.angleProperty().bind(angleX);
    yRotate.angleProperty().bind(angleY);

    //Illumination
    PointLight pointLight = new PointLight(Color.WHITE);
    pointLight.setTranslateX(100);
    pointLight.setTranslateY(200);
    pointLight.setTranslateZ(-2000);
    root.getChildren().add(pointLight);

    //Camera
    scene.setCamera(camera);
    primaryStage.setScene(scene);
    PerspectiveCamera subSceneCamera
            = new PerspectiveCamera(false);
    scene.setCamera(subSceneCamera);
```

```
//To zoom down the graphic,
//it is translated away from  the beholder,
//along the Z-direction (normal to the screen).
//Change zPos to change initial setting.
zSlider = new Slider();
zSlider.setMax(5000);
zSlider.setMin(-3000);


//Bind zPos to the reading of the slider
zSlider.valueProperty().bindBidirectional(zPos);
//Execute negative zoom
parent.translateZProperty().bind(zPos);


Button buttonHide = new Button("Hide textArea");
buttonHide.setOnAction((ActionEvent arg0) ->
{
  vBox.visibleProperty().set(false);
});


//Other JavaFX controls, labels and containers
GridPane gridPane = new GridPane();
gridPane.setVgap(4);
gridPane.setHgap(10);
gridPane.setPadding(new Insets(5, 5, 5, 5));
//Column 0, Row   0
gridPane.add(buttonLoad, 0,0);
//Column 1, Row   0
gridPane.add(buttonHide, 1,0);
//Row   1
gridPane.add(new Label("Zoom: "), 0, 1);
gridPane.add(zSlider, 1, 1);



// Subscene to place controls
```

```
    SubScene subScene
            = new SubScene(gridPane, 880, 200);
    subScene.setLayoutX(10);
    subScene.setLayoutY(800);
    root.getChildren().addAll(vBox, subScene);

    primaryStage.show();

    //The sense of touch is here
    parent.setOnMouseClicked((event)->
    {
        //Information that can be retrieved
        PickResult res = event.getPickResult();
        //Which node
        Node node = res.getIntersectedNode();
        //Identity label of node (null if not declared)
        String nodeId  = event.getPickResult()
                .getIntersectedNode().getId();
        //which face of the node (negative if is not enabled
        int face = event.getPickResult().getIntersectedFace();
        System.out.println("res = " + res);
        System.out.println("node = " + node);
        System.out.println("nodeID = " + nodeId);
        System.out.println("face" + face);
        //Change color to red, rotate around Z_AXIS
        if (node instanceof Box)
        {
            ((Box) node).setMaterial(
                    new PhongMaterial( Color.RED));

        }
    });
}


private String readFile(File file)
        throws FileNotFoundException, IOException
{
        StringBuilder stringBuffer = new StringBuilder();
        BufferedReader bufferedReader = null;
```

```
        int counterOfFaces = 0;
        int counterOfVertexes = 0;
        int counterOfNormals = 0;

        //Mesh has no predefined texture
mesh.getTexCoords().addAll(0,0);


try {

bufferedReader = new BufferedReader(new FileReader(file));

String text;

while ((text = bufferedReader.readLine()) != null)
{
    //Text for the textArea
    stringBuffer.append(text);

    //Vertexes are read, 3 per line
    if ((text.charAt(0) == 'v') && (text.charAt(1) == ' '))
    {
        System.out.println("Vertex  = " + counterOfVertexes
                + " = "+text);
        counterOfVertexes++;
        text = text.substring(2);
        int j = 0;
        for(int i = 0; i < 3; i++)
        {

            double d ;
            String s = "";
            Boolean go = true;

            while (go)
            {
                char thisChar = text.charAt(j);
                s = s + thisChar;
                //j++;
                char nextChar = ' ';
```

```
                if (++j < text.length())
                 nextChar = text.charAt(j);
                if ((nextChar == ' ') ||
                        (j >= text.length()))
                        go = false;

            }
            d = Double.parseDouble(s);

            float df = (float) d;
            System.out.println("Float number = " + df);
            mesh.getPoints().addAll(df);

        }
    }

    //Face is read, one face per line.
    //A face has 3 vertexes
    //intercalated with 3 texture coordinates that are zeroed

    if (text.charAt(0) == 'f')
    {

        System.out.println("Face  = " + counterOfFaces
                + " = "+text);
        counterOfFaces++;
        text = text.substring(2);
        int j = 0;
        int vertex[] = new int[3];
        for(int i = 0; i < 3; i++)
        {

            double d ;
            String s = "";
            Boolean go1 = true;


            while (go1)
            {
                if (j >= text.length()-2)
```

```
                    break;

                char thisChar = text.charAt(j);
                s = s + thisChar;
                j++;
                char nextChar = ' ';
                if (j < text.length())
                 nextChar = text.charAt(j);
                if (nextChar == '/' )
                {
                    go1 = false;
                    //Pass over "//" + duplication
                    j = j + 3 + s.length();
                }

            }
            d = Double.parseDouble(s);
            int v = (int) d;
            vertex[i] = v;
            System.out.println("Int number = " + v);
        }



    mesh.getFaces().addAll(vertex[0], 0, vertex[1], 0,
            vertex[2], 0);
    for(int i = 0; i < 3; i++)
        System.out.print(vertex[i] + " . ");
    System.out.println();
    }

    System.out.println("text = " + text);
     if ((text.charAt(0) == 'v') && (text.charAt(1) == 'n'))
    {
        counterOfNormals++;
    }
}

} catch (FileNotFoundException ex) {
```

```
Logger.getLogger(MlgrmLeftHand
        .class.getName()).log(Level.SEVERE, null, ex);
} catch (IOException ex) {
Logger.getLogger(MlgrmLeftHand
        .class.getName()).log(Level.SEVERE, null, ex);
} finally {
try {
    bufferedReader.close();
} catch (IOException ex) {
    Logger.getLogger(MlgrmLeftHand
            .class.getName()).log(Level.SEVERE, null, ex);
}
}


System.out.println("Number of points = "
    + mesh.getPoints().size() + "\n"
    + "Number of faces = "
    + mesh.getFaces().size() + "\n" );
int[] dest = new int[mesh.getFaces().size()];
mesh.getFaces().toArray(dest);

int counter = 0;
for(int j = 0; j <  dest.length/6; j++)
{
    System.out.print(j + "   " );
    for(int k = 0; k < 6; k++)
    {

        System.out.print(counter + "  "
                + dest[j*6 + k] + " " );

        counter++;
    }
    System.out.println();
}
System.out.println("counterOfPoints = "
        + counterOfVertexes );
System.out.println("counterOfFaces = " + counterOfFaces );
System.out.println("counterOfNormals = " + counterOfNormals);
```

```
System.out.println("mesh.getPoints().size() = "
        + mesh.getPoints().size() );
System.out.println("mesh.getFaces().size() = "
        + mesh.getFaces().size() );
int boundToNumberOfFaces
        = 6* (mesh.getPoints().size()/3 -1) + 474;
System.out.println("boundToNumberOfFaces = "
        + boundToNumberOfFaces);
System.out.println("getPointElementSize = "
        + mesh.getPointElementSize() );
System.out.println("getfaceElementSize = "
        + mesh.getFaceElementSize() );
mesh.getFaces().resize(boundToNumberOfFaces);
    return stringBuffer.toString();
}



@Override
public void start(Stage primaryStage)  throws Exception
{

    init(primaryStage);
    primaryStage.show();
}


public static void main(String[] args)
{

    launch(args);
}

}//End of main class M105 MlgrmLeftHand
```

106 ***Exercise.*** *Run previous program and play with the code.*

Our diagnostic is that previous program fails because imported files contain

too many faces, more than are allowed by JavaFX 3D. And, moreover, all triangles must be correctly oriented in JavaFX 3D, with normals pointing outwards, but imported meshes seemingly do not care about that. Let us try to battle these troubles.

We tested in first place the possibility that the imported mesh might have repeated faces. That was not the case. In second place, and to remedy the unbalance among the number of vertexes and that of faces, the three vertexes of the first face were repeatedly added to the mesh as many times as necessary. We need some understanding to test the orientation of faces.

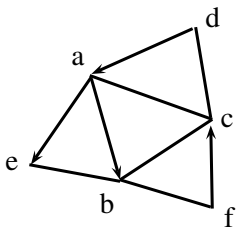


Figure 6.0. All this triangles follow the same orientation (imagine that water circulates in whirls for each triangle in the direction indicated by the arrows: all whirls follow a counterclockwise rotation.)

As it illustrated by the previous figure, we get the following enumeration of vertexes of faces that have the same orientation:

```
abc bca cab line 1
aeb eba bae line 2
cbf bfc fcb line 3
acd cda dac line 4
```

if we change e by c in line 2, f by a in line 3, d by b in line 4, we get:

```
abc bca cab line 1
aeb eba bae line 2 acb cba bac
cbf bfc fcb line 3 cba bac acb
acd cda dac line 4 acb cba bac
```

What did we get? We got the original face abc oriented in contrary form. Thus, to check that two faces have the same orientation, we check firstly if they are neighbor and if they are, then they must share two vertexes. If we change the new vertex of the second face by the lacking one in from first face and we get the original face with contrary orientation, then the two faces have the same

orientation. All these ideas were tested with the next program but no error was found. At last, the Author realized that the problem was very simple: counting begins in Java with 0 but in the .obj format with 1. So, an innocuous point was added as point 0-th and this was all to it.

107 ***Improved code of an importer of meshes follows.***

```
/*
    Program M107 MlgrmLeftHand2
    This program imports a mesh that is encoded
    in a file in the .obj format.
    Content is shown in a textArea and
    encoded mesh is drawn


    Tested only with some  files of the BodyParts3D project.
    http://lifesciencedb.jp/bp3d/
    Those files are licensed:

    The license for this database is specified in the
    Creative Commons Attribution-Share Alike 2.1 Japan.
    If you use data from this database, please be sure
    attribute this database as follows: "BodyParts3D, (c)
    The Database Center for Life Science licensed under
    CC Attribution-Share Alike 2.1 Japan".
    http://dbarchive.biosciencedbc.jp/en/bodyparts3d/lic.html

 */

package mlgrmlefthand2;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.application.Application;
```

```
import static javafx.application.Application.launch;
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;
import javafx.event.ActionEvent;
import javafx.geometry.Insets;
import javafx.scene.Group;
import javafx.scene.PerspectiveCamera;
import javafx.scene.PointLight;
import javafx.scene.Scene;
import javafx.scene.SubScene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.ScrollPane;
import javafx.scene.control.Slider;
import javafx.scene.control.TextArea;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.MeshView;
import javafx.scene.shape.TriangleMesh;
import javafx.scene.transform.Rotate;
import javafx.stage.FileChooser;
import javafx.stage.Stage;




public class MlgrmLeftHand2 extends Application
{
    //Slider for zooming
    private Slider zSlider;
    //Position of graphic along the z-axis,
    //normal to the screen.
    //It is a double property because
    //it will get involved into a binding
    private final DoubleProperty zPos
            = new SimpleDoubleProperty(2000);
```

```
    //For listener of mouse
    private double anchorAngleX = 0;
    private double anchorAngleY = 0;
    private double anchorX, anchorY;
    //Angles of rotation of the scene
    private final DoubleProperty angleX
            = new SimpleDoubleProperty(0);
    private final DoubleProperty angleY
            = new SimpleDoubleProperty(0);

    PerspectiveCamera camera
            = new PerspectiveCamera(false);

    private static final String title
            =  "Importer of meshes in format .obj";

    //Tool to read files
    BufferedReader br ;
    private static  String textOfFile = "";
    //The mesh is declared and instantiated
    private static final  TriangleMesh mesh  = new TriangleMesh();
    private static int counterOfFaces = 0;
    private static int counterOfVertexes = 0;
    private static int counterOfNormals = 0;

    private static int[] destFaces ;



public void init(Stage primaryStage)
{
    primaryStage.setTitle(title);


    //======File loader ======

    /*Adapted from Java-Buddy:
    http://java-buddy.blogspot.com/2012/05/read-text-file-with-javafx-
    */
```

```
//Content of file will be shown in textArea
final TextArea textArea = new TextArea();
textArea.prefWidth(400);
textArea.setWrapText(true);

//Container
ScrollPane scrollPane = new ScrollPane();
scrollPane.getStyleClass().add("noborder-scroll-pane");
scrollPane.setContent(textArea);
scrollPane.setFitToWidth(true);
scrollPane.setPrefWidth(400);
scrollPane.setPrefHeight(180);
//Button to load file
Button buttonLoad = new Button("Load file");
//Listener
buttonLoad.setOnAction((ActionEvent arg0) ->
{
    FileChooser fileChooser = new FileChooser();

    //Name of file with suffix .obj
    FileChooser.ExtensionFilter extFilter
            = new FileChooser
                    .ExtensionFilter("TXT files (*.obj)"
                            , "*.obj");
    fileChooser.getExtensionFilters().add(extFilter);

    //Show open file dialog
    File file = fileChooser.showOpenDialog(primaryStage);
    try {
            textOfFile = readFile(file);
        } catch (IOException ex) {
            Logger.getLogger(MlgrmLeftHand2
                    .class.getName())
                    .log(Level.SEVERE, null, ex);
        }
    //Text copied to textArea
    textArea.setText(textOfFile);
```

```
});


VBox vBox = new VBox();
vBox.getChildren().addAll(scrollPane);
//====== End of File loader

//======Mesh preparation
MeshView bone =
  new MeshView(mesh);
PhongMaterial goldenMaterial
        = new PhongMaterial(Color.GOLDENROD);
bone.setMaterial(goldenMaterial);

int scaleFactor = 3;
bone.scaleXProperty().setValue(scaleFactor);
bone.scaleYProperty().setValue(scaleFactor);
bone.scaleZProperty().setValue(scaleFactor);

//bone.drawModeProperty().set(DrawMode.LINE);


//The whole drawing  is created
//as a JavaFX space-like object
final Group parent = new Group( bone);


//Initial position
parent.setTranslateX(450);
parent.setTranslateY(400);
parent.setTranslateZ(0);
parent.visibleProperty().set(true);


Rotate xRotate;
Rotate yRotate;
//Initialization of rotation
parent.getTransforms().setAll
(
```

```
      xRotate = new Rotate(0, Rotate.X_AXIS),
      yRotate = new Rotate(0, Rotate.Y_AXIS)
    );

    //Graphic is attached to root
    final Group root = new Group(parent);
    //Window of the scene
    final Scene scene = new Scene(root, 900, 900, true);
    //Background
    scene.setFill(Color.SKYBLUE);

    //Listener of mousePressed
    scene.setOnMousePressed
    (
        event ->
        {
            anchorX = event.getSceneX();
            anchorY = event.getSceneY();
            anchorAngleX = angleX.get();
            anchorAngleY = angleY.get();
        }
    );


    //Listener of mouseDragged
    scene.setOnMouseDragged
    (
        event ->
        {
            angleX.set(anchorAngleX
                    - (anchorY - event.getSceneY()));
            angleY.set(anchorAngleY
                    + anchorX - event.getSceneX());
        }
    );

    //Rotation of the scene gets bound to
    //the angle determined by  mouseDragged
    xRotate.angleProperty().bind(angleX);
    yRotate.angleProperty().bind(angleY);
```

```
//Illumination
PointLight pointLight = new PointLight(Color.WHITE);
pointLight.setTranslateX(100);
pointLight.setTranslateY(200);
pointLight.setTranslateZ(-2000);
root.getChildren().add(pointLight);

//Camera
scene.setCamera(camera);
primaryStage.setScene(scene);
PerspectiveCamera subSceneCamera
        = new PerspectiveCamera(false);
scene.setCamera(subSceneCamera);



//To zoom down the graphic,
//it is translated away from  the beholder,
//along the Z-direction (normal to the screen).
//Change zPos to change initial setting.
zSlider = new Slider();
zSlider.setMax(5000);
zSlider.setMin(1);


//Bind zPos to the reading of the slider
zSlider.valueProperty().bindBidirectional(zPos);
//Execute negative zoom
parent.translateZProperty().bind(zPos);


Button buttonHide = new Button("Hide textArea");
buttonHide.setOnAction((ActionEvent arg0) ->
{
  vBox.visibleProperty().set(false);
});


//Other JavaFX controls, labels and containers
```

```
    GridPane gridPane = new GridPane();
    gridPane.setVgap(4);
    gridPane.setHgap(10);
    gridPane.setPadding(new Insets(5, 5, 5, 5));
    //Column 0, Row   0
    gridPane.add(buttonLoad, 0,0);
    //Column 1, Row   0
    gridPane.add(buttonHide, 1,0);
    //Row   1
    gridPane.add(new Label("Zoom: "), 0, 1);
    gridPane.add(zSlider, 1, 1);



    // Subscene to place controls
    SubScene subScene
            = new SubScene(gridPane, 880, 200);
    subScene.setLayoutX(10);
    subScene.setLayoutY(800);
    root.getChildren().addAll(vBox, subScene);

    primaryStage.show();
}


//File is read and its content is returned as a string.
//The associated mesh is recovered
//and correct orientation fo faces is provided.
//The mesh is a global variable.
private String readFile(File file)
        throws FileNotFoundException, IOException
{
    StringBuilder stringBuffer = new StringBuilder();
    BufferedReader bufferedReader = null;

    //The mesh has no predefined texture
    mesh.getTexCoords().addAll(0,0);
    //In the .obj format, counting begins with 1
    //but in Java, we begin with 0.
    //So, an additional inocuos point is added.
```

```
    mesh.getPoints().addAll(0,0,0);

    System.out.println("File content: ");

try
{

bufferedReader = new BufferedReader(new FileReader(file));

String text;

while ((text = bufferedReader.readLine()) != null)
{
    //Text for the textArea
    stringBuffer.append(text);

    //Vertexes are read
    if ((text.charAt(0) == 'v') && (text.charAt(1) == ' '))
    {
        System.out.println("Vertex  = " + counterOfVertexes
                + " = "+text);
        counterOfVertexes++;
        text = text.substring(2);
        int j = 0;
        //Three vertexes are read
        for(int i = 0; i < 3; i++)
        {
            double d ;
            String s = "";
            Boolean go = true;

            while (go)
            {
                char thisChar = text.charAt(j);
                s = s + thisChar;
                char nextChar = ' ';
                //++j means add one to j
                //and then evaluate expression with <
                if (++j < text.length())
                 nextChar = text.charAt(j);
```

```
            if ((nextChar == ' ') ||
                    (j >= text.length()))
                    go = false;

        }
        //string to double
        d = Double.parseDouble(s);
        float df = (float) d;
        System.out.println("Float number = " + df);
        mesh.getPoints().addAll(df);

    }
}



//Face is read, one face per line.
//A face has 3 vertexes
//intercalated with 3 texture coordinates that are zeroed

if (text.charAt(0) == 'f')
{

    System.out.println("Face  = " + counterOfFaces
            + " = "+text);
    counterOfFaces++;
    text = text.substring(2);
    int j = 0;
    int vertex[] = new int[3];
    for(int i = 0; i < 3; i++)
    {

        double d ;
        String s = "";
        Boolean go1 = true;


        while (go1)
        {
```

```
                if (j >= text.length()-2)
                    break;

                char thisChar = text.charAt(j);
                s = s + thisChar;
                j++;
                char nextChar = ' ';
                if (j < text.length())
                 nextChar = text.charAt(j);
                if (nextChar == '/' )
                {
                    go1 = false;
                    //Pass over "//" + duplication
                    j = j + 3 + s.length();
                }

            }
            d = Double.parseDouble(s);
            int v = (int) d;
            vertex[i] = v;
            System.out.println("Int number = " + v);
        }



mesh.getFaces().addAll(vertex[0], 0, vertex[1], 0,
            vertex[2], 0);
for(int i = 0; i < 3; i++)
    System.out.print(vertex[i] + " . ");
System.out.println();
}


//Normal vectors are counted and displayed
if ((text.charAt(0) == 'v') && (text.charAt(1) == 'n'))
{

    System.out.println("Normal " + counterOfNormals
            + " = " + text);
```

```
        counterOfNormals++;
    }
}

} //End of retrieving try
    catch (FileNotFoundException ex)
    {
        Logger.getLogger(MlgrmLeftHand2
        .class.getName()).log(Level.SEVERE, null, ex);
    }
    catch (IOException ex)
    {
        Logger.getLogger(MlgrmLeftHand2
        .class.getName()).log(Level.SEVERE, null, ex);
    }
    finally
    {
        try
        {
                bufferedReader.close();
        } catch (IOException ex)
        {
            Logger.getLogger(MlgrmLeftHand2
            .class.getName()).log(Level.SEVERE, null, ex);
        }
    }
    System.out.println("End of file content. ");
    //Other indicators are reported
    report();

    //Meshes for JavaFX 3D have particular requirements.
    corrections();
    return stringBuffer.toString();
}


//Indicators of retrieved data are printed
private void report()
{
    float[] destPoints = new float[mesh.getPoints().size()];
```

```
mesh.getPoints().toArray(destPoints);
System.out.println("POINTS");
for(int j = 0; j <  destPoints.length/3; j++)
{
    System.out.print(j + "   " );
    for(int k = 0; k < 3; k++)
    {
        System.out.print( destPoints[j*3 + k] + "   " );
    }
    System.out.println();
}

destFaces = new int[mesh.getFaces().size()];
//Faces go to a matrix
mesh.getFaces().toArray(destFaces);
System.out.println("FACES WITH TEXTURE COORDINATES");
for(int j = 0; j <  destFaces.length/6; j++)
{
    System.out.print(j + "   " );
    for(int k = 0; k < 6; k++)
    {
        System.out.print( destFaces[j*6 + k] + "   " );
    }
    System.out.println();
}

System.out.println("counterOfPoints = "
        + counterOfVertexes );
System.out.println("counterOfFaces = "
        + counterOfFaces );
System.out.println("counterOfNormals = "
        + counterOfNormals);
System.out.println("mesh.getPoints().size() = "
        + mesh.getPoints().size() );
System.out.println("mesh.getFaces().size() = "
        + mesh.getFaces().size() );

System.out.println("getPointElementSize = "
        + mesh.getPointElementSize() );
System.out.println("getfaceElementSize = "
```

```
            + mesh.getFaceElementSize() );

}

//Meshes must comply with additional constraints
//to be correctly displayed by JavaFX 3D. So,
//additional points are added to satisfy compiler directives
//in regard with a surpass of faces relative to
//the number of points.
//The faces of the mesh are correctly oriented.
private void corrections()
{
    int boundToGetFacesSize
            = 6* (mesh.getPoints().size()/3 -1);
    float[] destPoints = new float[mesh.getPoints().size()];
    mesh.getPoints().toArray(destPoints);
    int deficit
        = (mesh.getFaces().size() - boundToGetFacesSize)/6;
    System.out.println("boundToGetFacesSize = "
            + boundToGetFacesSize
            + "\nNumber of added points = " + deficit);
    //Repeated points are added to go around Java compiler:
    //number of points and of meshes
    //must observe certain relation.
    for(int l = 0; l < deficit; l++)
            mesh.getPoints().addAll(destPoints[0]
            , destPoints[1]
            , destPoints[2]);

    //Faces get correct orientation
    destFaces = new int[mesh.getFaces().size()];
    mesh.getFaces().toArray(destFaces);

    for(int I = 0; I <  destFaces.length/6; I++)
    {
        for(int J = I; J <  destFaces.length/6; J++)
        {

            if (areNeighbors (I,J))
            {
```

```
                System.out.println(" Faces " + I + " and " + J +
                " are neighbors and " + haveSameorientation(I, J)
                        + " have the same orientation " );
            }


        }
    }

    listNeighbors();
}


//Are faces I and J neighbors?
//They are when they share two vertexes.
private boolean areNeighbors(int I, int J)
{
    ArrayList<Integer> faceI = new ArrayList<>();
    faceI.add(destFaces[I*6]);
    faceI.add(destFaces[I*6+ 2]);
    faceI.add(destFaces[I*6 + 4]);
    int counter = 0;
    if (faceI.contains(destFaces[J*6])) counter++;
    if (faceI.contains(destFaces[J*6 + 2])) counter++;
    if (faceI.contains(destFaces[J*6 + 4])) counter++;
    return counter > 1;
}

//Are faces I and J equal as set?
//They are when they have the same set of vertexes
//possibly in different order.
private void listNeighbors()
{
    for(int I = 0; I <  destFaces.length/6; I++)
    {
        System.out.println(" Face " + I + " has neighbors ");
        for(int J = 0; J <  destFaces.length/6; J++)
        {
            if (areNeighbors (I,J))
            {
```

```
                    System.out.println( J + "  " );

            }
        }
    }

}


//Are faces I and J equal as set?
//They are when they have the same set of vertexes
//possibly in different order.
private boolean areEqual(int I, int J)
{

    ArrayList<Integer> faceI = new ArrayList<>();
    faceI.add(destFaces[I*6]);
    faceI.add(destFaces[I*6+ 2]);
    faceI.add(destFaces[I*6 + 4]);
    int counter = 0;
    if (faceI.contains(destFaces[J*6])) counter++;
    if (faceI.contains(destFaces[J*6 + 2])) counter++;
    if (faceI.contains(destFaces[J*6 + 4])) counter++;
    return counter > 2;
}

//If faces I and J are neighbors,
//Do they have equal orientation?
private boolean haveSameorientation(int I, int J)
{
    //LackingVertex:
    //Vertex that is in faceI but not in faceJ
    Integer lackingVertex = 0;
    int indexLackingVertex = 0;
    int indexHereLackingVertex = 0;
    ArrayList<Integer> faceJ = new ArrayList<>();
    faceJ.add(destFaces[J*6]);
    faceJ.add(destFaces[J*6+ 2]);
    faceJ.add(destFaces[J*6 + 4]);
    if (faceJ.contains(destFaces[I*6]));
```

```
else
{
    //This node is in faceI but not in faceJ
    lackingVertex = destFaces[I*6];
    indexLackingVertex = I*6;
    indexHereLackingVertex = 0;

}

if (faceJ.contains(destFaces[I*6 + 2]));
else
{
    //This node is in faceI but not in faceJ
    lackingVertex = destFaces[I*6 + 2];
    indexLackingVertex = I*6 + 2;
    indexHereLackingVertex = 1;
}

if (faceJ.contains(destFaces[I*6 + 4]));
else
{
    //This node is in faceI but not in faceJ
    lackingVertex = destFaces[I*6 + 4];
    indexLackingVertex = I*6 + 4;
    indexHereLackingVertex = 2;
}



//New vertex
//Vertex that is in faceJ but not in faceI

//Integer newVertex;
//int indexNewVertex = 0;
int indexHereNewVertex = 0;
ArrayList<Integer> faceI = new ArrayList<>();
faceI.add(destFaces[I*6]);
faceI.add(destFaces[I*6+ 2]);
faceI.add(destFaces[I*6 + 4]);
if (faceI.contains(destFaces[J*6]));
```

```
else
{
    //This node is in faceJ but not in faceI
    //newVertex = destFaces[J*6];
    //indexNewVertex = J*6;
    indexHereNewVertex = 0;
}

if (faceI.contains(destFaces[J*6 + 2]));
else
{
    //This node is in faceJ but not in faceI
    //newVertex = destFaces[J*6 + 2];
    //indexNewVertex = J*6 + 2;
    indexHereNewVertex = 1;
}

if (faceI.contains(destFaces[J*6 + 4]));
else
{
    //This node is in faceJ but not in faceI
    //newVertex = destFaces[J*6 + 4];
    //indexNewVertex = J*6 + 4;
    indexHereNewVertex = 2;
}


//NewVertex is replaced in faceJ by lackingVertex:
//Newvertex is deleted
faceJ.remove(indexHereNewVertex);
//Lacking vertex is added at indexHereNewVertex
faceJ.add(indexHereNewVertex, lackingVertex);

//The three instances of the same orientation
//of  faceI is built
//Face abc
ArrayList<Integer> faceIa = new ArrayList<>();
faceIa.add(destFaces[I*6]);
faceIa.add(destFaces[I*6+ 2]);
faceIa.add(destFaces[I*6 + 4]);
```

```
    //Face bca
    ArrayList<Integer> faceIb = new ArrayList<>();
    faceIb.add(destFaces[I*6+ 2]);
    faceIb.add(destFaces[I*6 + 4]);
    faceIb.add(destFaces[I*6]);
    //face cab
    ArrayList<Integer> faceIc = new ArrayList<>();
    faceIc.add(destFaces[I*6 + 4]);
    faceIc.add(destFaces[I*6]);
    faceIc.add(destFaces[I*6+ 2]);


    //Comparison of lists according to  membership and order.
    //If faceI and remodelled faceJ
    //have the same orientation,
    //faceI and faceJ have not the same orientation
    boolean sameOrientation = true;
    if (faceIa.equals(faceJ)) sameOrientation = false;
    else if (faceIb.equals(faceJ)) sameOrientation = false;
    else if (faceIc.equals(faceJ)) sameOrientation = false;

    return sameOrientation;
}



@Override
public void start(Stage primaryStage)  throws Exception
{

    init(primaryStage);
    primaryStage.show();
}


public static void main(String[] args)
{

    launch(args);
}
```

```
}//End of main class M107 MlgrmLeftHand2
```

108 ***Exercise.*** *Run the previous program and play with the code.*

6.3 The skeleton of the hand

We know how to load one mesh from a file. So, we can visualize all bones one by one. The hand nevertheless is an ensemble with around 30 bones. Let us learn how to visualize all of then at once.

109 ***The next code includes multiple loading of bones.***

```
/*
    Program M109 MlgrmLeftHand3
    This program imports a mesh that is encoded
    in a file in the .obj format.
    Content is shown in a textArea and
    encoded mesh is drawn


    Tested only with some  files of the BodyParts3D project.
    http://lifesciencedb.jp/bp3d/
    Those files are licensed:

    The license for this database is specified in the
    Creative Commons Attribution-Share Alike 2.1 Japan.
    If you use data from this database, please be sure
    attribute this database as follows: "BodyParts3D, (c)
    The Database Center for Life Science licensed under
    CC Attribution-Share Alike 2.1 Japan".
    http://dbguidehive.biosciencedbc.jp/en/bodyparts3d/lic.html

 */

package mlgrmlefthand3;

import java.io.BufferedReader;
import java.io.File;
```

```
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;
import javafx.event.ActionEvent;
import javafx.scene.Group;
import javafx.scene.PerspectiveCamera;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Menu;
import javafx.scene.control.MenuBar;
import javafx.scene.control.MenuItem;
import javafx.scene.control.ScrollPane;
import javafx.scene.control.TextArea;
import javafx.scene.input.KeyEvent;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.Box;
import javafx.scene.shape.MeshView;;
import javafx.scene.shape.TriangleMesh;
import javafx.scene.transform.Rotate;
import javafx.scene.transform.Scale;
import javafx.scene.transform.Translate;
import javafx.stage.FileChooser;
import javafx.stage.Stage;



public class MlgrmLeftHand3 extends Application
{

    //Position of graphic along the z-axis,
```

```
//normal to the screen.
//It is a double property because
//it will get involved into a binding
private final DoubleProperty zPos
        = new SimpleDoubleProperty(0);

private static final String title
        =  "Importer of meshes in format .obj";

//Tool to read files
BufferedReader br ;
private static  String textOfFile = "";

private static int counterOfFaces = 0;
private static int counterOfVertexes = 0;
private static int counterOfNormals = 0;

private static int[] destFaces ;


//*****************

final Group root = new Group();

final Group axisGroup = new Group();

final PerspectiveCamera camera
        = new PerspectiveCamera(true);
final Xform cameraXform = new Xform();
final Xform cameraXform2 = new Xform();
final Xform cameraXform3 = new Xform();

final double cameraDistance = 1450;//450
final Xform world3DXform = new Xform();
final Xform bonesXform = new Xform();



boolean timelinePlaying = false;
double ONE_FRAME = 1.0/24.0;
```

```
    double DELTA_MULTIPLIER = 200.0;
    double CONTROL_MULTIPLIER = 0.1;
    double SHIFT_MULTIPLIER = 0.1;
    double ALT_MULTIPLIER = 0.5;

    double mousePosX;
    double mousePosY;
    double mouseOldX;
    double mouseOldY;
    double mouseDeltaX;
    double mouseDeltaY;

    final static boolean printReadFile = false;

   final Xform guidesXform = new Xform();
   //Menubar declaration and initialization
    MenuBar menuBar = new MenuBar();


     private void buildCamera() {
        root.getChildren().add(cameraXform);
        cameraXform.getChildren().add(cameraXform2);
        cameraXform2.getChildren().add(cameraXform3);
        cameraXform3.getChildren().add(camera);
        cameraXform3.setRotateZ(180.0);

        camera.setNearClip(0.1);
        camera.setFarClip(10000.0);
        camera.setTranslateZ(-cameraDistance);
        cameraXform.ry.setAngle(320.0);
        cameraXform.rx.setAngle(40);
    }



    private void buildAxes() {
        System.out.println("buildAxes()");
        final PhongMaterial redMaterial
                = new PhongMaterial();
        redMaterial.setDiffuseColor(Color.DARKRED);
```

```
        redMaterial.setSpecularColor(Color.RED);

        final PhongMaterial greenMaterial
                = new PhongMaterial();
        greenMaterial.setDiffuseColor(Color.DARKGREEN);
        greenMaterial.setSpecularColor(Color.GREEN);

        final PhongMaterial blueMaterial
                = new PhongMaterial();
        blueMaterial.setDiffuseColor(Color.DARKBLUE);
        blueMaterial.setSpecularColor(Color.BLUE);

        final Box xAxis = new Box(640.0, 1, 1);
        final Box yAxis = new Box(1, 640.0, 1);
        final Box zAxis = new Box(1, 1, 640.0);

        xAxis.setMaterial(redMaterial);
        yAxis.setMaterial(greenMaterial);
        zAxis.setMaterial(blueMaterial);

        axisGroup.getChildren().addAll(xAxis, yAxis, zAxis);
        world3DXform.getChildren().addAll(axisGroup);
    }



//
// The handleCameraViews file contains
//the handleMouse() and handleKeyboard()
// methods that are used in the
//MoleculeSampleApp application to handle the
// different 3D camera views.
//These methods are used in the Getting Started with
// JavaFX 3D Graphics tutorial.
//

private void handleMouse(Scene scene)
{
    scene.setOnMousePressed((MouseEvent me) ->
    {
```

```
        mousePosX = me.getSceneX();
        mousePosY = me.getSceneY();
        mouseOldX = me.getSceneX();
        mouseOldY = me.getSceneY();
    });
    scene.setOnMouseDragged((MouseEvent me) ->
    {
        mouseOldX = mousePosX;
        mouseOldY = mousePosY;
        mousePosX = me.getSceneX();
        mousePosY = me.getSceneY();
        mouseDeltaX = (mousePosX - mouseOldX);
        mouseDeltaY = (mousePosY - mouseOldY);

        double modifier = 1.0;
        double modifierFactor = 0.1;

        if (me.isControlDown()) {
            modifier = 0.1;
        }
        if (me.isShiftDown()) {
            modifier = 10.0;
        }
        if (me.isPrimaryButtonDown()) {
            cameraXform.ry.setAngle(cameraXform
                    .ry.getAngle()
                    - mouseDeltaX*modifierFactor
                            *modifier*2.0);  // +
            cameraXform.rx.setAngle(cameraXform
                    .rx.getAngle()
                    + mouseDeltaY*modifierFactor
                            *modifier*2.0);  // -
        }
        else if (me.isSecondaryButtonDown()) {
            double z = camera.getTranslateZ();
            double newZ = z
            + mouseDeltaX*modifierFactor*modifier;
            camera.setTranslateZ(newZ);
        }
        else if (me.isMiddleButtonDown()) {
```

```
            cameraXform2.t.setX(cameraXform2.t.getX()
                + mouseDeltaX*modifierFactor*modifier*0.3);
            cameraXform2.t.setY(cameraXform2.t.getY()
                + mouseDeltaY*modifierFactor*modifier*0.3);
        }
        });
}

private void handleKeyboard(Scene scene)
{

    scene.setOnKeyPressed((KeyEvent event) -> {

        switch (event.getCode()) {
            case A: //Hide axis
                if (event.isControlDown()) {
                    if (axisGroup.isVisible()) {
                        System.out.println(
                                "setVisible(false)");
                        axisGroup.setVisible(false);
                    }
                    else {
                        System.out.println(
                                "setVisible(true)");
                        axisGroup.setVisible(true);
                    }
                }
                break;
        }
    });
}



@Override
public void start(Stage primaryStage)
{
    System.out.println("start");

    Scene scene = new Scene(root, 1200, 1000, true);
```

```
/*
   Multiple file loader
*/

//Content of files will be shown in textArea
final TextArea textArea = new TextArea();
textArea.prefWidth(400);
textArea.setWrapText(true);
textArea.setText("");
//Container  for textArea.
ScrollPane scrollPane = new ScrollPane();
scrollPane.getStyleClass().add("noborder-scroll-pane");
scrollPane.setContent(textArea);
scrollPane.setFitToWidth(true);
scrollPane.setPrefWidth(400);
scrollPane.setPrefHeight(180);
VBox vBox = new VBox();
vBox.visibleProperty().setValue(false);
vBox.getChildren().addAll(scrollPane);
root.getChildren().addAll(vBox);


// --- Menu files
Menu filesMenu = new Menu("Files");
//Menu to load multiple files
MenuItem loadFiles = new MenuItem("Load + process multiple files "
//MenuItem added to menu
filesMenu.getItems().addAll(loadFiles);
menuBar.getMenus().add(filesMenu);
root.getChildren().add(menuBar);

//Listener
loadFiles.setOnAction((ActionEvent t) ->
{
    FileChooser fileChooser = new FileChooser();

    //Name of file with extension .obj
```

```
        FileChooser.ExtensionFilter extFilter
                = new FileChooser
                        .ExtensionFilter("TXT files (*.obj)"
                                , "*.obj");
        fileChooser.getExtensionFilters().add(extFilter);
        //Files will be kept in a list
        List<File> files = new FileChooser()
                .showOpenMultipleDialog(primaryStage);
        //Each file is processed
        files.stream().forEach((File file) -> { try
        {
            //Text of file is read and mesh is extracted
            textOfFile = readFile(file);
            //Text copied to textArea
            textArea.appendText(textOfFile);

            } catch (IOException ex)
            {
                Logger.getLogger(MlgrmLeftHand3
                        .class.getName())
                        .log(Level.SEVERE, null, ex);
            }
        });

    //To remedy a not understood bug
    //( = menuBar is added to the graphic )
    root.getChildren().remove(menuBar);

    vBox.visibleProperty().set(true);
    });

    //====== End of File loader

    primaryStage.setTitle(title
            + ". Drag the mouse to rotate. "
        + " Ctrl + A = Axis visibility"
            );
    scene.setFill(Color.BLUE);
    primaryStage.setScene(scene);
    primaryStage.show();
```

```
    buildAxes();
    handleKeyboard(scene);
    handleMouse(scene);

    //Hide textArea
    Button hideTextArea = new Button();
    hideTextArea.setText("View graphic");
    //Listener
    hideTextArea.setOnAction((ActionEvent t) ->
    {
        vBox.visibleProperty().set(false);
        buildCamera();
        double scaleFactor = 5;
        bonesXform.setScale(scaleFactor);
        //Centering: found by trial and error.
        double xMax = bonesXform.getBoundsInLocal().getMaxX();
        double yMax = bonesXform.getBoundsInLocal().getMaxY();
        double zMax = bonesXform.getBoundsInLocal().getMaxZ();
        double xMin = bonesXform.getBoundsInLocal().getMinX();
        double yMin = bonesXform.getBoundsInLocal().getMinY();
        double zMin = bonesXform.getBoundsInLocal().getMinZ();
        double x = scaleFactor *  0.82 * (xMax + xMin) ;
        double y = scaleFactor * 0.78 * (yMax + yMin);
        double z = scaleFactor * 0.9 * (zMax + zMin) ;
        bonesXform.setTranslate(-x, -y, -z);
        world3DXform.getChildren().add(bonesXform);
        root.getChildren().add(world3DXform);
        scene.setCamera(camera);
    });

    vBox.getChildren().addAll(hideTextArea);
    primaryStage.show();
}



//File is read and its content is returned as a string.
//The associated mesh is recovered
//and correct orientation fo faces is provided.
```

```
//The mesh is a global variable.

private String readFile(File file)
        throws FileNotFoundException, IOException
{
    //Name of file is printed
    String name = file.getName();
    System.out.println(name);
    //Buffer to keep the content of file
    //StringBuilder stringBuffer = new StringBuilder();
    StringBuilder stringBuffer  = new StringBuilder();
    //Tool to read file
    BufferedReader bufferedReader = null;

    //A mesh is declared and instantiated
    TriangleMesh mesh  = new TriangleMesh();
    //The mesh has no predefined texture
    mesh.getTexCoords().addAll(0,0);
    //In the .obj format, counting begins with 1
    //but in Java, we begin with 0.
    //So, an additional inocuos point is added.
    mesh.getPoints().addAll(0,0,0);

    if (printReadFile)
        System.out.println("File content: ");
//File is read
try
{
//
bufferedReader = new BufferedReader(new FileReader(file));

String text;

while ((text = bufferedReader.readLine()) != null)
{
    //Text for the textArea
    stringBuffer.append(text);

    //Vertexes are read
    if ((text.charAt(0) == 'v') && (text.charAt(1) == ' '))
```

```
{
    if (printReadFile)
        System.out.println("Vertex  = "
                + counterOfVertexes + " = "+text);
    counterOfVertexes++;
    text = text.substring(2);
    int j = 0;
    //Three vertexes are read
    for(int i = 0; i < 3; i++)
    {
        double d ;
        String s = "";
        Boolean go = true;

        while (go)
        {
            char thisChar = text.charAt(j);
            s = s + thisChar;
            char nextChar = ' ';
            //++j means add one to j
            //and then evaluate expression with <
            if (++j < text.length())
             nextChar = text.charAt(j);
            if ((nextChar == ' ') ||
                    (j >= text.length()))
                    go = false;

        }
        //string to double
        d = Double.parseDouble(s);
        float df = (float) d;
        if (printReadFile)
            System.out.println("Float number = " + df);
        mesh.getPoints().addAll(df);

    }
}
```

```
//Face is read, one face per line.
//A face has 3 vertexes
//intercalated with 3 texture coordinates that are zeroed

if (text.charAt(0) == 'f')
{

    if (printReadFile)
        System.out.println("Face  = " + counterOfFaces
            + " = "+text);
    counterOfFaces++;
    text = text.substring(2);
    int j = 0;
    int vertex[] = new int[3];
    for(int i = 0; i < 3; i++)
    {

        double d ;
        String s = "";
        Boolean go1 = true;


        while (go1)
        {
            if (j >= text.length()-2)
                break;

            char thisChar = text.charAt(j);
            s = s + thisChar;
            j++;
            char nextChar = ' ';
            if (j < text.length())
             nextChar = text.charAt(j);
            if (nextChar == '/' )
            {
                go1 = false;
                //Pass over "//" + duplication
                j = j + 3 + s.length();
            }
```

```
            }
            d = Double.parseDouble(s);
            int v = (int) d;
            vertex[i] = v;
            if (printReadFile)
                System.out.println("Int number = " + v);
        }



    mesh.getFaces().addAll(vertex[0], 0, vertex[1], 0,
            vertex[2], 0);
    if (printReadFile)
    {
        for(int i = 0; i < 3; i++)
            System.out.print(vertex[i] + " . ");
        System.out.println();
    }
    }


    //Normal vectors are counted and displayed
    if ((text.charAt(0) == 'v') && (text.charAt(1) == 'n'))
    {

        if (printReadFile)
            System.out.println("Normal " + counterOfNormals
                + " = " + text);
        counterOfNormals++;
    }
}

} //End of  try.
    //Exception management
    catch (FileNotFoundException ex)
    {
        Logger.getLogger(MlgrmLeftHand3
        .class.getName()).log(Level.SEVERE, null, ex);
```

```
}
catch (IOException ex)
{
    Logger.getLogger(MlgrmLeftHand3
    .class.getName()).log(Level.SEVERE, null, ex);
}
finally
{
    try
    {
        bufferedReader.close();
    } catch (IOException ex)
    {
        Logger.getLogger(MlgrmLeftHand3
        .class.getName()).log(Level.SEVERE, null, ex);
    }
}

if (printReadFile)
{
    System.out.println("End of file content. ");
    //Other indicators are reported
    report(mesh);
}


//======Mesh preparation

MeshView boneMesh =
        new MeshView(mesh);

PhongMaterial goldenMaterial
        = new PhongMaterial(Color.GOLDENROD);
boneMesh.setMaterial(goldenMaterial);
Group boneGroup = new Group( boneMesh);
//Mesh is added to bonesXform
bonesXform.getChildren().add(boneGroup);


return stringBuffer.toString();
```

```
}


//Indicators of retrieved data are printed
private void report(TriangleMesh mesh)
{
    float[] destPoints = new float[mesh.getPoints().size()];
    mesh.getPoints().toArray(destPoints);
    System.out.println("POINTS");
    for(int j = 0; j <  destPoints.length/3; j++)
    {
        System.out.print(j + "   " );
        for(int k = 0; k < 3; k++)
        {
            System.out.print( destPoints[j*3 + k] + "   " );
        }
        System.out.println();
    }

    destFaces = new int[mesh.getFaces().size()];
    //Faces go to a matrix
    mesh.getFaces().toArray(destFaces);
    System.out.println("FACES WITH TEXTURE COORDINATES");
    for(int j = 0; j <  destFaces.length/6; j++)
    {
        System.out.print(j + "   " );
        for(int k = 0; k < 6; k++)
        {
            System.out.print( destFaces[j*6 + k] + "   " );
        }
        System.out.println();
    }

    System.out.println("counterOfPoints = "
            + counterOfVertexes );
    System.out.println("counterOfFaces = "
            + counterOfFaces );
    System.out.println("counterOfNormals = "
            + counterOfNormals);
```

```
    System.out.println("mesh.getPoints().size() = "
            + mesh.getPoints().size() );
    System.out.println("mesh.getFaces().size() = "
            + mesh.getFaces().size() );

    System.out.println("getPointElementSize = "
            + mesh.getPointElementSize() );
    System.out.println("getfaceElementSize = "
            + mesh.getFaceElementSize() );

}


public static void main(String[] args)
{

    launch(args);
}

}//End of main class M109 MlgrmLeftHand3


//Outer class Xform.
//Machinery that enables rotations in vivo of the scene
//and the conformation of groups of control.
class Xform extends Group {

    public enum RotateOrder {
        XYZ, XZY, YXZ, YZX, ZXY, ZYX
    }

    public Translate t  = new Translate();
    public Translate p  = new Translate();
    public Translate ip = new Translate();
    public Rotate rx = new Rotate();
    { rx.setAxis(Rotate.X_AXIS); }
    public Rotate ry = new Rotate();
    { ry.setAxis(Rotate.Y_AXIS); }
    public Rotate rz = new Rotate();
    { rz.setAxis(Rotate.Z_AXIS); }
```

```
    public Scale s = new Scale();

    public Xform() {
        super();
        getTransforms().addAll(t, rz, ry, rx, s);
    }

    public Xform(RotateOrder rotateOrder) {
        super();
        // choose the order of rotations
        //based on the rotateOrder
        switch (rotateOrder) {
        case XYZ:
            getTransforms().addAll(t, p, rz, ry, rx, s, ip);
            break;
        case XZY:
            getTransforms().addAll(t, p, ry, rz, rx, s, ip);
            break;
        case YXZ:
            getTransforms().addAll(t, p, rz, rx, ry, s, ip);
            break;
        case YZX:
            getTransforms().addAll(t, p, rx, rz, ry, s, ip);
            break;
        case ZXY:
            getTransforms().addAll(t, p, ry, rx, rz, s, ip);
            break;
        case ZYX:
            getTransforms().addAll(t, p, rx, ry, rz, s, ip);
            break;
        }
    }

    public void setTranslate(double x, double y, double z) {
        t.setX(x);
        t.setY(y);
        t.setZ(z);
    }

    public void setTranslate(double x, double y) {
```

```
        t.setX(x);
        t.setY(y);
    }

    // Cannot override these methods as they are final:
    // public void setTranslateX(double x) { t.setX(x); }
    // public void setTranslateY(double y) { t.setY(y); }
    // public void setTranslateZ(double z) { t.setZ(z); }
    // Use these methods instead:
    public void setTx(double x) { t.setX(x); }
    public void setTy(double y) { t.setY(y); }
    public void setTz(double z) { t.setZ(z); }

    public void setRotate(double x, double y, double z) {
        rx.setAngle(x);
        ry.setAngle(y);
        rz.setAngle(z);
    }

    public void setRotateX(double x) { rx.setAngle(x); }
    public void setRotateY(double y) { ry.setAngle(y); }
    public void setRotateZ(double z) { rz.setAngle(z); }
    public void setRx(double x) { rx.setAngle(x); }
    public void setRy(double y) { ry.setAngle(y); }
    public void setRz(double z) { rz.setAngle(z); }

    public void setScale(double scaleFactor) {
        s.setX(scaleFactor);
        s.setY(scaleFactor);
        s.setZ(scaleFactor);
    }

    public void setScale(double x, double y, double z) {
        s.setX(x);
        s.setY(y);
        s.setZ(z);
    }

    // Cannot override these methods as they are final:
    // public void setScaleX(double x) { s.setX(x); }
```

```
    // public void setScaleY(double y) { s.setY(y); }
    // public void setScaleZ(double z) { s.setZ(z); }
    // Use these methods instead:
    public void setSx(double x) { s.setX(x); }
    public void setSy(double y) { s.setY(y); }
    public void setSz(double z) { s.setZ(z); }

    public void setPivot(double x, double y, double z) {
        p.setX(x);
        p.setY(y);
        p.setZ(z);
        ip.setX(-x);
        ip.setY(-y);
        ip.setZ(-z);
    }

    public void reset() {
        t.setX(0.0);
        t.setY(0.0);
        t.setZ(0.0);
        rx.setAngle(0.0);
        ry.setAngle(0.0);
        rz.setAngle(0.0);
        s.setX(1.0);
        s.setY(1.0);
        s.setZ(1.0);
        p.setX(0.0);
        p.setY(0.0);
        p.setZ(0.0);
        ip.setX(0.0);
        ip.setY(0.0);
        ip.setZ(0.0);
    }

    public void resetTSP() {
        t.setX(0.0);
        t.setY(0.0);
        t.setZ(0.0);
        s.setX(1.0);
        s.setY(1.0);
```

```
        s.setZ(1.0);
        p.setX(0.0);
        p.setY(0.0);
        p.setZ(0.0);
        ip.setX(0.0);
        ip.setY(0.0);
        ip.setZ(0.0);
    }
}//End of class Xform; End of program M109 MlgrmLeftHand3
```

110 ***Exercise.*** *Run the previous program and play with the code. Use the freedom to load multiple files to see the complete hand or some parts of it. You can place in a given folder the specific files you want to load and see them all.*

111 ***Research and challenge.*** *Persons usually have supernumerary bones in the hand: these are the sesamoid bones. Inquire about them and their function. Add ellipsoids to the meshes of the hand to mimic them.*

6.4 Skinned hand

The web is full in mologrames or 3D models and of applications to deal with them [7], some of which are free and can be downloaded without registration. Since there are various formats, it is a good idea to use a free application to see mologrames before attempting anything with Java. The Author uses MeshLab [11] and Blender[2] that come packed with Ubuntu but that exist also for Windows and Mac. Let us learn how to use another important for us example.

112 ***Exercise.*** *Go to the site [6] and download a file with a mesh of a hand. Extract it to a suitable folder. Verify that our previous program fails over this file. So, open the file with a text editor. Revise that it is written in a dialect a bit different than the one used above. Adapt previous program to read this dialect, run the resultant program and admire the wonderful mesh. Answer*

6.5 Conclusion

We have learned how to import and draw meshes from a file in the .obj format. Actually, we only have managed two dialects. The first uses triangle meshes and is used by the authors of the *BodyParts3D Project* from Japan. The second uses

rectangles or a mix of triangles and rectangles. The result has been very gratifying in spite of found troubles.

Chapter 7

Evolution vs. our hand

`Testing a hull hypothesis.`

113 ***Objective.*** *We explore here the problem of deciding whether or not Evolution is a good explanation of our hand beginning from that of a common ancestor with the chimp.*

7.1 The hand as a marvel

Our hand is not very complex. Instead, it is marvel.

114 ***Exercise.*** *Enjoy the video by BBC(2014), [1], about the human hand and agree else disagree with the appreciation of the Author: the hand is a marvel that is more clever than complex.*

To proceed, we need a prerequisite from JavaFX.

7.2 JavaFX: layers

Our purpose is to produce an animation in which the hand of a chimp is transformed into a human hand. To that aim, we fit each hand by a curve and next deform the curve that represents the hand of a chimp into the hand of a human. It seems to us that this work is best done if we use different layers, one for the original drawing of both hands and other for the fitted curves. So, the next program shows how to deal with layers in CanvasFX.

115 ***JavaFX Code that shows the use of layers.***

```
 /*
 *  Program M115 Layers
    This program shows how to use various layers
    in a graphic application.

    We follow Oracle.
 */
package layers;

import javafx.application.Application;
import javafx.beans.value.ObservableValue;
import javafx.collections.FXCollections;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.control.ChoiceBox;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.stage.Stage;


public class Layers extends Application

{

    Canvas layer1 = new Canvas(300, 250);
    GraphicsContext gc1;
    Canvas layer2 = new Canvas(300, 250);
    GraphicsContext gc2 ;
    BorderPane borderPane;
    ChoiceBox cb ;

    @Override
    public void start(Stage primaryStage)
    {
```

```
Group root = new Group();
// Obtain Graphics Contexts
gc1 = layer1.getGraphicsContext2D();
gc1.setFill(Color.GREEN);
gc1.fillOval(50,50,20,20);
gc2 = layer2.getGraphicsContext2D();
gc2.setFill(Color.BLUE);
gc2.fillOval(100,100,20,20);

//Booking
borderPane = new BorderPane();
borderPane.setTop(cb);
Pane pane = new Pane();
pane.getChildren().add(layer1);
pane.getChildren().add(layer2);
layer1.toFront();
borderPane.setCenter(pane);
root.getChildren().add(borderPane);



// Handler for Layer 1
layer1.addEventHandler(MouseEvent
    .MOUSE_PRESSED, (MouseEvent e) ->
    {
        gc1.fillOval(e.getX(),e.getY(),20,20);
    });
// Handler for Layer 2
layer2.addEventHandler(MouseEvent
    .MOUSE_PRESSED, (MouseEvent e) ->
    {
        gc2.fillOval(e.getX(),e.getY(),20,20);
    });

//ChoiceBox for controlling
//which layer is the working one
cb = new ChoiceBox();
cb.setItems(FXCollections.observableArrayList(
"Layer 1 is GREEN", "Layer 2 is BLUE"));
```

```
        cb.getSelectionModel().selectedItemProperty().
        addListener((ObservableValue o,
                Object o1, Object o2) -> {
            if(o2.toString().equals("Layer 1 is GREEN"))
            {
                layer1.toFront();
                gc1.setFill(Color.GREEN);
            }else if(o2.toString().equals("Layer 2 is BLUE"))
            {
                layer2.toFront();
                gc2.setFill(Color.BLUE);
            }
        });
        cb.setValue("Layer 1 is GREEN");
        root.getChildren().add(cb);

        Scene scene = new Scene(root, 300, 250);
        primaryStage.setScene(scene);
        primaryStage.show();
    }



    public static void main(String[] args)
    {
        launch(args);
    }

}//End of main class Program M115 Layers
```

116 ***Exercise.*** *Run the previous program and play with the code.*

7.3 Evolution as a null hypothesis

Human beings are extremely powerful for extracting the essential information. This work can be made automatically as by the eye or after research as in compression of digital files. The hand is particularly suitable for this.

117 ***Exercise.*** *Download the cartoon of the hands of a human and of a chimp*

posted by Denise Morgan (2015), [8]. Click over the image and select `save image as`*. Choose the format and click* `Save`*.*

Our purpose is to produce an animation in which the hand of a chimp is transformed into a human hand. To that aim, we fit each drawing by a broken line in a single stroke and next deform the line that represents the hand of a chimp into that of the hand of a human.

118 ***Use the following code to associate a polyline to each hand. To prevent problems with directories, paste a copy of the required image to the same folder of this program.***

```
/*
 *  Program M118 HandChimpHuman
    Use this program to associate a polyline to
    a hand of a chimp and another to the
    hand of a human.
    Output = coordinates of  polylines.

    You need a drawing with the hand of a chimp
    and the hand of a human:
http://www.eurekalert.org/multimedia/pub/50921.php?from=228578

    Or, prompt Google with: chimp hand.
    It is also possible to make own drawing.

    Save the image as "ChimpHumanHand.jpg"
    to  the same package of this program.
 */

package handchimphuman;


import java.io.IOException;
import javafx.application.Application;
import javafx.beans.value.ObservableValue;
import javafx.collections.FXCollections;

import javafx.scene.Group;
import javafx.scene.Scene;
```

```
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.control.ChoiceBox;
import javafx.scene.effect.DropShadow;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.Pane;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.stage.Stage;



public class HandChimpHuman extends Application
{
//Layer for the image with the hands of chimp nd human
private static final Canvas layer1 = new Canvas(500, 450);
private static GraphicsContext gc1;
//Layer for the broken line of the hand of chimp
private static final Canvas layer2 = new Canvas(500, 450);
private static GraphicsContext gc2 ;
//Layer for the broken line of the hand of the human
private static final Canvas layer3 = new Canvas(500, 450);
private static GraphicsContext gc3 ;
private static BorderPane borderPane  = new BorderPane();
private static final Pane pane = new Pane();
private static ChoiceBox cb ;
//Polyline that represents the hand of the chimp
private static final double[] XChimp = new double[200];
private static final double[] YChimp = new double[200];
private static int nPointsChimp = 0;
//Polyline that represents the hand of the chimp
private static final double[] XHuman = new double[200];
private static final double[] YHuman = new double[200];
private static int nPointsHuman = 0;
//Line that represents the hand of a human
```

```
@Override
public void start(Stage stage) throws IOException
{


    final VBox vb = new VBox();
    DropShadow shadow = new DropShadow();

    Group root = new Group();
    Scene scene = new Scene(root, 500, 450);
    scene.setFill(Color.BLACK);
    stage.setScene(scene);
    stage.setTitle("The chimp-to-human gap");


    //Instantiation of Graphics Contexts
    gc1 = layer1.getGraphicsContext2D();
    gc2 = layer2.getGraphicsContext2D();
    gc2.setFill(Color.RED);
    gc2.fillOval(120,380,20,20);
    gc2.setLineWidth(3);
    gc2.moveTo(130,390);
    gc3 = layer3.getGraphicsContext2D();
    gc3.setFill(Color.BLUE);
    gc3.fillOval(380,350,20,20);
    gc3.moveTo(390,360);

    //Booking
    borderPane = new BorderPane();
    pane.getChildren().add(layer1);
    pane.getChildren().add(layer2);
    pane.getChildren().add(layer3);

    layer1.toFront();
    borderPane.setCenter(pane);
    root.getChildren().add(borderPane);

    // Handler for Layer 1
    layer1.addEventHandler(MouseEvent
        .MOUSE_PRESSED, (MouseEvent e) ->
```

```
        {
            gc1.strokePolyline(XChimp,YChimp, nPointsChimp);
            gc1.strokePolyline(XHuman,YHuman, nPointsHuman);
            System.out.println("Points of chimp = "
                    + nPointsChimp);
            System.out.println("Points of human = "
                    + nPointsHuman);
        });


    // Handler for Layer 2
    layer2.addEventHandler(MouseEvent
        .MOUSE_PRESSED, (MouseEvent e) ->
        {
            double x =  e.getX();
            double y = e.getY();
            gc2.fillOval(x-3,y-3,6,6);
            XChimp[nPointsChimp] = x;
            YChimp[nPointsChimp] = y;
            nPointsChimp++;
        });
    // Handler for Layer 3
    layer3.addEventHandler(MouseEvent
        .MOUSE_PRESSED, (MouseEvent e) ->
        {
            double x =  e.getX();

            double y = e.getY();
            gc3.fillOval(x-3,y-3,6,6);
            XHuman[nPointsHuman] = x;
            YHuman[nPointsHuman] = y;
            nPointsHuman++;
        });

    //ChoiceBox for controlling
    //which layer is the working one
    cb = new ChoiceBox();
    cb.setItems(FXCollections.observableArrayList(
        "Image and lines",
        "Draw a single polyline: red is activated",
```

```
        "Draw a single polyline: blue is activated",
        "Polylines alone"));
    cb.getSelectionModel().selectedItemProperty().
    addListener((ObservableValue o,
        Object o1, Object o2) ->
        {
            switch (o2.toString())
            {
                case "Image and lines":
                    layer1.toBack();
                    layer1.visibleProperty().setValue(true);
                    layer2.visibleProperty().setValue(true);
                    layer3.visibleProperty().setValue(true);
                    gc1.strokePolyline(XChimp,YChimp, nPointsChimp);
                    gc1.strokePolyline(XHuman,YHuman, nPointsHuman);
                    break;
                case "Draw a single polyline: red is activated":
                    layer2.toFront();
                    layer1.visibleProperty().setValue(true);
                    layer2.visibleProperty().setValue(true);
                    layer3.visibleProperty().setValue(false);
                    gc2.setFill(Color.RED);
                    break;
                case "Draw a single polyline: blue is activated":
                    layer3.toFront();
                    layer1.visibleProperty().setValue(true);
                    layer2.visibleProperty().setValue(false);
                    layer3.visibleProperty().setValue(true);
                    gc2.setFill(Color.BLUE);
                    break;
                case "Polylines alone":
                    scene.setFill(Color.YELLOW);
                    layer1.visibleProperty().setValue(false);
                    layer2.visibleProperty().setValue(true);
                    layer3.visibleProperty().setValue(true);
                    gc2.strokePolyline(XChimp,YChimp, nPointsChimp);
                    gc3.strokePolyline(XHuman,YHuman, nPointsHuman);
                    //Captured coordinates of points  are reported
                    report();
                    break;
```

```
            }
        });
    cb.setValue("Draw a single polyline: red is activated");
    borderPane.setTop(cb);
    root.getChildren().add(cb);



    vb.setLayoutX(5);
    vb.setLayoutY(50);
    vb.setSpacing(100);

    root.getChildren().addAll(vb);

    //Image is loaded
    Image image  = new Image(getClass()
                .getResourceAsStream("ChimpHumanHand.jpg"));
    final ImageView imageView  = new ImageView(image);
    imageView.setEffect(shadow);
    gc1.drawImage(image, 50, 100);

    stage.show();

}



    static private void report()
    {
        System.out.println("Chimp: x-coordinates");
        for (int i = 0; i < nPointsChimp; i++)
            System.out.print((int) XChimp[i] + ", ");
        System.out.println();
        System.out.println("Chimp: y-coordinates");
        for (int i = 0; i < nPointsChimp; i++)
            System.out.print((int) YChimp[i] + ", ");
        System.out.println();
        System.out.println("Human: x-coordinates");
        for (int i = 0; i < nPointsHuman; i++)
            System.out.print((int) XHuman[i] + ", ");
```

```
        System.out.println();
        System.out.println("Human: y-coordinates");
        for (int i = 0; i < nPointsHuman; i++)
            System.out.print((int) YHuman[i] + ", ");
        System.out.println();
    }


    public static void main(String[] args)
    {
        launch(args);
    }

}//End of main class M118 HandChimpHuman
```

119 ***Exercise.*** *Run the previous program and play with the code.*

Experimenting with the previous program allowed the Author to imagine a better application as the next program shows.

120 ***The following code allows to draw one polyline for the hand of the chimp and another for the hand of the human but observing as much homology as possible.***

```
/*
 *  Program M120 HandChimpHuman2
    Use this program to associate a polyline to
    a hand of a chimp and another to the
    hand of a human.
    Observe as much homology as possible:
    the two polylines must have the same number of points,
    and points in the same entry must correspond
    with homologous positions in the hands.
    Output = coordinates of  polylines.

    You need a drawing with the hand of a chimp
    and the hand of a human:
http://www.eurekalert.org/multimedia/pub/50921.php?from=228578
```

```
    Or, prompt Google with: chimp hand.
    It is also possible to make own drawing.

    Save the image as "ChimpHumanHand.jpg"
    to  the same package of this program.
 */

package handchimphuman2;

import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.beans.value.ObservableValue;
import javafx.collections.FXCollections;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.control.ChoiceBox;
import javafx.scene.effect.DropShadow;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.Pane;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.stage.Stage;


public class HandChimpHuman2 extends Application
{
//Layer for the image with the hands of chimp nd human
private static final Canvas layer1 = new Canvas(500, 450);
private static GraphicsContext gc1;
//Layer for the broken line of the hand of chimp
private static final Canvas layer2 = new Canvas(500, 450);
private static GraphicsContext gc2 ;
//Layer for the broken line of the hand of the human
private static final Canvas layer3 = new Canvas(500, 450);
private static GraphicsContext gc3 ;
```

```
private static BorderPane borderPane  = new BorderPane();
private static final Pane pane = new Pane();
private static ChoiceBox cb ;
//Polyline that represents the hand of the chimp
private static final double[] XChimp = new double[200];
private static final double[] YChimp = new double[200];
private static int nPointsChimp = 0;
//Polyline that represents the hand of the chimp
private static final double[] XHuman = new double[200];
private static final double[] YHuman = new double[200];
private static int nPointsHuman = 0;
//Line that represents the hand of a human


@Override
public void start(Stage stage)
{


    final VBox vb = new VBox();
    DropShadow shadow = new DropShadow();

    Group root = new Group();
    Scene scene = new Scene(root, 500, 450);
    scene.setFill(Color.BLACK);
    stage.setScene(scene);
    stage.setTitle("The chimp-to-human gap");


    //Instantiation of Graphics Contexts
    gc1 = layer1.getGraphicsContext2D();
    gc2 = layer2.getGraphicsContext2D();
    gc2.setFill(Color.RED);
    gc2.fillOval(120,380,20,20);
    gc2.setLineWidth(3);
    gc2.moveTo(130,390);
    gc3 = layer3.getGraphicsContext2D();
    gc3.setFill(Color.BLUE);
    gc3.fillOval(380,350,20,20);
    gc3.moveTo(390,360);
```

```
//Booking
borderPane = new BorderPane();
pane.getChildren().add(layer1);
pane.getChildren().add(layer2);
pane.getChildren().add(layer3);

layer1.toFront();
borderPane.setCenter(pane);
root.getChildren().add(borderPane);

// Handler for Layer 1
layer1.addEventHandler(MouseEvent
    .MOUSE_PRESSED, (MouseEvent e) ->
    {
        gc1.strokePolyline(XChimp,YChimp, nPointsChimp);
        gc1.strokePolyline(XHuman,YHuman, nPointsHuman);
        System.out.println("Points of chimp = "
                + nPointsChimp);
        System.out.println("Points of human = "
                + nPointsHuman);
    });


// Handler for Layer 2
layer2.addEventHandler(MouseEvent
    .MOUSE_PRESSED, (MouseEvent e) ->
    {
        double x =  e.getX();
        double y = e.getY();
        gc2.fillOval(x-3,y-3,6,6);
        XChimp[nPointsChimp] = x;
        YChimp[nPointsChimp] = y;
        nPointsChimp++;
    });
// Handler for Layer 3
layer3.addEventHandler(MouseEvent
    .MOUSE_PRESSED, (MouseEvent e) ->
    {
        double x =  e.getX();
```

```
            double y = e.getY();
            gc3.fillOval(x-3,y-3,6,6);
            XHuman[nPointsHuman] = x;
            YHuman[nPointsHuman] = y;
           nPointsHuman++;
            gc3.setFill(Color.GREEN);
            double xChimp =  XChimp[nPointsHuman];
            double yChimp = YChimp[nPointsHuman];
            gc3.fillOval(xChimp-3,yChimp-3,6,6);
            gc3.setFill(Color.BLUE);


        });

    //ChoiceBox for controlling
    //which layer is the working one
    cb = new ChoiceBox();
    cb.setItems(FXCollections.observableArrayList(
        "Image and lines",
        "Draw the chimp polyline: red is activated",
        "Draw the human polyline: blue is activated",
        "Polylines alone"));
    cb.getSelectionModel().selectedItemProperty().
    addListener((ObservableValue o,
        Object o1, Object o2) ->
        {
            switch (o2.toString())
            {
                case "Image and lines":
                    layer1.toBack();
                    layer1.visibleProperty().setValue(true);
                    layer2.visibleProperty().setValue(true);
                    layer3.visibleProperty().setValue(true);
                    gc1.strokePolyline(XChimp,YChimp, nPointsChimp);
                    gc1.strokePolyline(XHuman,YHuman, nPointsHuman);
                    break;
                case "Draw the chimp polyline: red is activated":
                    layer2.toFront();
                    layer1.visibleProperty().setValue(true);
```

```
                    layer2.visibleProperty().setValue(true);
                    layer3.visibleProperty().setValue(true);
                    gc2.setFill(Color.RED);
                    break;
                case "Draw the human polyline: blue is activated":
                    layer3.toFront();
                    layer1.visibleProperty().setValue(true);
                    layer2.visibleProperty().setValue(true);
                    layer3.visibleProperty().setValue(true);


                    gc3.setFill(Color.GREEN);
            double xChimp =  XChimp[0];
            double yChimp = YChimp[0];
            gc3.fillOval(xChimp-3,yChimp-3,6,6);
            gc3.setFill(Color.BLUE);
                    break;
                case "Polylines alone":
                    scene.setFill(Color.YELLOW);
                    layer1.visibleProperty().setValue(false);
                    layer2.visibleProperty().setValue(true);
                    layer3.visibleProperty().setValue(true);
                    gc2.strokePolyline(XChimp,YChimp, nPointsChimp);
                    gc3.strokePolyline(XHuman,YHuman, nPointsHuman);
                    //captured coordinates of points  are reported
                    report();
                    break;

            }
        });
    cb.setValue("Draw the chimp polyline: red is activated");
    borderPane.setTop(cb);
    root.getChildren().add(cb);



    vb.setLayoutX(5);
    vb.setLayoutY(50);
    vb.setSpacing(100);
```

```
    root.getChildren().addAll(vb);

    //mage is loaded
    /*
    Image image  = new Image(getClass()
                .getResourceAsStream("ChimpHumanHand.jpg"));
    */

    Image image  = new Image(getClass()
                .getResourceAsStream("ChimpHumanHand.jpg"));

    final ImageView imageView  = new ImageView(image);
    imageView.setEffect(shadow);
    gc1.drawImage(image, 50, 100);

    stage.show();

}

    static private void report()
    {
        System.out.println("Chimp: x-coordinates");
        for (int i = 0; i < nPointsChimp; i++)
            System.out.print((int) XChimp[i] + ", ");
        System.out.println();
        System.out.println("Chimp: y-coordinates");
        for (int i = 0; i < nPointsChimp; i++)
            System.out.print((int) YChimp[i] + ", ");
        System.out.println();
        System.out.println("Human: x-coordinates");
        for (int i = 0; i < nPointsHuman; i++)
            System.out.print((int) XHuman[i] + ", ");
        System.out.println();
        System.out.println("Human: y-coordinates");
        for (int i = 0; i < nPointsHuman; i++)
            System.out.print((int) YHuman[i] + ", ");
        System.out.println();
        /*
        Output by the Author:
        Chimp: x-coordinates
```

```
        125, 141, 161, 131, 115, 108,  99, 103,  97, 101,
        109, 118, 124, 126, 122, 127, 140, 146, 148, 151,
        154, 161, 173, 180, 178, 181, 185, 192, 201, 208,
        204, 208, 207, 205, 201, 204, 211, 210, 214, 213,
        218, 226, 230, 231, 235, 230, 228, 219, 201, 188,
        180, 182, 190, 200, 203,
        Chimp: y-coordinates
        362, 362, 394, 411, 404, 394, 296, 277, 192, 179,
        180, 192, 250, 261, 155, 138, 136, 147, 239, 250,
        147, 130, 128, 141, 245, 253, 174, 162, 160, 174,
        263, 277, 302, 311, 321, 330, 341, 330, 322, 311,
        301, 295, 296, 307, 320, 335, 357, 385, 409, 417,
        411, 378, 357, 354, 345,
        Human: x-coordinates
        323, 319, 314, 310, 302, 299, 290, 293, 293, 299,
        308, 310, 313, 319, 320, 325, 337, 340, 340, 346,
        347, 355, 361, 369, 368, 374, 376, 383, 394, 395,
        393, 394, 394, 394, 395, 395, 392, 398, 407, 405,
        410, 418, 429, 427, 430, 426, 416, 397, 377, 362,
        333, 345, 368, 377, 387,
        Human: y-coordinates
        408, 409, 409, 409, 409, 395, 315, 303, 238, 229,
        230, 237, 291, 286, 214, 199, 199, 212, 287, 286,
        198, 187, 190, 195, 286, 289, 211, 202, 205, 218,
        291, 298, 310, 315, 319, 324, 340, 321, 313, 298,
        285, 278, 280, 295, 313, 325, 350, 383, 405, 409,
        396, 364, 338, 331, 325,

        */
    }


    public static void main(String[] args)
    {
        launch(args);
    }

}//End of main class M120 HandChimpHuman2
```

121 ***Exercise.*** *Run the previous program and play with the code.*

Our plan for the animation is to have each segment of the polyline of the hand of the chimp moving and transforming into the corresponding homologous segment in the hand of the human. So, let learn how to transform a stroke.

122 ***The following code produces an animation in which a line is transformed into other one.***

```
/*
 *  Program M122 HandChimpHuman3
    An animated line is shown.
 */

package handchimphuman3;


import javafx.animation.KeyFrame;
import javafx.animation.KeyValue;
import javafx.animation.Timeline;
import javafx.application.Application;
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.shape.Line;
import javafx.stage.Stage;


public class HandChimpHuman3 extends Application
{


@Override
public void start(Stage stage)
{

    Group root = new Group();

    //Line goes from (xo,yo) to (x1,y1);
```

```
    DoubleProperty xo = new SimpleDoubleProperty( 0 );
    DoubleProperty yo = new SimpleDoubleProperty( 0 );
    DoubleProperty x1 = new SimpleDoubleProperty( 0 );
    DoubleProperty y1 = new SimpleDoubleProperty( 0 );
    Line l = new Line(xo.getValue(),yo.getValue(),
                x1.getValue(),y1.getValue());
    //End points get bound to  varying parameters
    l.startXProperty().bind(xo);
    l.startYProperty().bind(yo);
    l.endXProperty().bind(x1);
    l.endYProperty().bind(y1);
    root.getChildren().add(l);

    //Animation of parameters
    Timeline timeline = new Timeline
    (
        //Initial states
        new KeyFrame( javafx.util.Duration.ZERO,
            new KeyValue( xo, 400 ),
            new KeyValue( x1, 40 ),
        new KeyValue( yo, 400 ),
            new KeyValue( y1, 40 )),
        //Final states
        new KeyFrame( new javafx.util.Duration(2000),
            new KeyValue( xo, 23 ),
            new KeyValue( x1, 300),
            new KeyValue( yo, 500 ),
            new KeyValue( y1, 200 ) )

    );
    timeline.setCycleCount( Timeline.INDEFINITE );
    timeline.setAutoReverse( true );
    timeline.play();

    Scene scene = new Scene(root, 600, 600);
    stage.setScene(scene);
    stage.setTitle("Moving line");
    stage.show();
}
```

```
public static void main(String[] args)
{
    launch(args);
}

}//End of main class M122 HandChimpHuman3
```

123 ***Exercise.*** *Run the previous program and play with the code.*

We can now produce an animation using the information on polylines captured by a previous program and that encodes for the shapes of two hands, one of a chimp and another of a human.

124 ***Exercise. Develop a program to produce an animation in which the hand of the chimp evolves into that of a human.*** ***Answer***

Our animation invites us to formalize what everybody thinks:

125 ***Formulation of the null hypothesis.*** *The animation presented by the previous program awakes the feeling that Evolution is quite simple and indeed natural. Formally, the null hypothesis of science reads: the human hand together with that of the chimp come by Evolution from a common ancestor. Thanks to our animation, we can force this hypothesis into the following one: the human hand together with that of the chimp come by Gradual Evolution from a common ancestor.*

126 ***Exercise.*** *Express you agreement else discomfort with our formulation of the null hypothesis of science.*

Now, it is obvious that this null hypothesis is false. Let us show why.

7.4 Computing the null hypothesis.

The null hypothesis of science reads: the human and chimp hands came by gradual Evolution from the hand of a common ancestor. We must calculate all the predictions that spring from this hypothesis to compare them with nature. Or, we can just formulate one prediction that shows that this hypothesis is obviously false, as it indeed is. To see that, it is enough to consider not the two dimensional images of mentioned hands but their 3D versions. To see a human hand, just look to your own hand. Now, the Author was unable to get a free 3D model of the hand of a chimp so, we must get conformed with 2D images.

127 ***Exercise.*** *We can appreciate important 2D images of the hand of the chimp at the gallery Eskeletons, [5]. Enjoy wandering over this wonderful site and then go to the menu*

```
Comparative Anatomy
```

Select:

```
Bone: Manual Intermediate phalanx 3
View: medial.
```

Phalanges of a human, chimp and orangutan must appear. Now, agree else disagree with the appreciation of the Author:

The palm side of the phalanx of the orangutan is more curved than that of the chimp that is more curved than that of man that is anyway curved.

128 ***Exercise.*** *Assume that the chimp, the bonobo and the human are direct descendants from a common ancestor. Formulate and sustain a mandatory, falsifiable and false prediction relating the size and curvature of the phalanges of the chimp to the shape of the phalanges of the hand of humans.* *Answer*

We want now to use a 3D mesh to illustrate our mandatory prediction developed in the last exercise. The problem is that we have no 3D-mesh of a phalange with a pronounced curvature. So, let us learn how to make such a 3D-mesh from 2D portraits.

7.5 From silhouettes to 3D-meshes

We already know how to associate a polylone, a silhouette, to a 2D image of a bone. The following purpose is to compose a 3D mesh that represents a bone beginning from two images that represent mutually perpendicular views and their corresponding silhouettes. So, you will need the dorsal and medial views of the image of the *Manual Intermediate Phalanx 3 of the thumb of the chimp* from Eskeletons, l.c., [5].

Let us solve posed task step by step. We need, of course, a program to associate a polyline to a 2D view. To that aim we can use program M118 HandChimpHuman. But the problem with it is that the input file with the image must be in the same package as the program. So, let us add to this program the capability to read the file with the image from whatever folder or directory.

129 ***Use the following code to associate a polyline to a 2D image. The polyline will capture a representation of its silhouette. The 2D image may be placed at whatever folder or directory.***

```
/*
 *  Program M130 HandChimpHuman5
    Use this program to associate a closed polyline to
    a silhuoette of a 2D image, say of a bone.
    Output = coordinates of  polylines.


    You need the dorsal and medial  views of the image of the
    Manual Intermediate Phalanx 3 of the thumb of the chimp from
    http://www.eskeletons.org/

    Save the image to a convenient directory.
 */

package handchimphuman5;


import java.io.File;
import java.io.IOException;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.beans.value.ObservableValue;
import javafx.collections.FXCollections;

import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.control.ChoiceBox;
import javafx.scene.effect.DropShadow;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.Pane;
```

```
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.stage.FileChooser;
import javafx.stage.Stage;



public class HandChimpHuman5 extends Application
{
//Layer for the image with the hands of chimp nd human
private static final Canvas layer1 = new Canvas(500, 450);
private static GraphicsContext gc1;
//Layer for the broken line of the hand of chimp
private static final Canvas layer2 = new Canvas(500, 450);
private static GraphicsContext gc2 ;
//Layer for the broken line of the hand of the human
private static final Canvas layer3 = new Canvas(500, 450);
private static GraphicsContext gc3 ;
private static BorderPane borderPane  = new BorderPane();
private static final Pane pane = new Pane();
private static ChoiceBox cb ;
//Polyline that represents the hand of the chimp
private static final double[] XChimp = new double[200];
private static final double[] YChimp = new double[200];
private static int nPointsChimp = 0;
//Polyline that represents the hand of the chimp
private static final double[] XHuman = new double[200];
private static final double[] YHuman = new double[200];
private static int nPointsHuman = 0;
//Line that represents the hand of a human


@Override
public void start(Stage stage) throws IOException
{


    final VBox vb = new VBox();
    DropShadow shadow = new DropShadow();
```

```
Group root = new Group();
Scene scene = new Scene(root, 500, 450);
scene.setFill(Color.BLACK);
stage.setScene(scene);
stage.setTitle("The chimp-to-human gap");


//Instantiation of Graphics Contexts
gc1 = layer1.getGraphicsContext2D();
gc2 = layer2.getGraphicsContext2D();
gc2.setFill(Color.RED);
gc2.fillOval(120,380,20,20);
gc2.setLineWidth(3);
gc2.moveTo(130,390);
gc3 = layer3.getGraphicsContext2D();
gc3.setFill(Color.BLUE);
gc3.fillOval(380,350,20,20);
gc3.moveTo(390,360);

//Booking
borderPane = new BorderPane();
pane.getChildren().add(layer1);
pane.getChildren().add(layer2);
pane.getChildren().add(layer3);

layer1.toFront();
borderPane.setCenter(pane);
root.getChildren().add(borderPane);

// Handler for Layer 1
layer1.addEventHandler(MouseEvent
    .MOUSE_PRESSED, (MouseEvent e) ->
    {
        gc1.strokePolyline(XChimp,YChimp, nPointsChimp);
        gc1.strokePolyline(XHuman,YHuman, nPointsHuman);
        System.out.println("Points of chimp = "
                + nPointsChimp);
        System.out.println("Points of human = "
                + nPointsHuman);
    });
```

```
// Handler for Layer 2
layer2.addEventHandler(MouseEvent
    .MOUSE_PRESSED, (MouseEvent e) ->
    {
        double x =  e.getX();
        double y = e.getY();
        gc2.fillOval(x-3,y-3,6,6);
        XChimp[nPointsChimp] = x;
        YChimp[nPointsChimp] = y;
        nPointsChimp++;
    });
// Handler for Layer 3
layer3.addEventHandler(MouseEvent
    .MOUSE_PRESSED, (MouseEvent e) ->
    {
        double x =  e.getX();

        double y = e.getY();
        gc3.fillOval(x-3,y-3,6,6);
        XHuman[nPointsHuman] = x;
        YHuman[nPointsHuman] = y;
        nPointsHuman++;
    });

//ChoiceBox for controlling
//which layer is the working one
cb = new ChoiceBox();
cb.setItems(FXCollections.observableArrayList(
    "Image and lines",
    "Draw a single polyline: red is activated",
    "Draw a single polyline: blue is activated",
    "Polylines alone"));
cb.getSelectionModel().selectedItemProperty().
addListener((ObservableValue o,
    Object o1, Object o2) ->
    {
        switch (o2.toString())
        {
```

```
                case "Image and lines":
                    layer1.toBack();
                    layer1.visibleProperty().setValue(true);
                    layer2.visibleProperty().setValue(true);
                    layer3.visibleProperty().setValue(true);
                    gc1.strokePolyline(XChimp,YChimp, nPointsChimp);
                    gc1.strokePolyline(XHuman,YHuman, nPointsHuman);
                    break;
                case "Draw a single polyline: red is activated":
                    layer2.toFront();
                    layer1.visibleProperty().setValue(true);
                    layer2.visibleProperty().setValue(true);
                    layer3.visibleProperty().setValue(false);
                    gc2.setFill(Color.RED);
                    break;
                case "Draw a single polyline: blue is activated":
                    layer3.toFront();
                    layer1.visibleProperty().setValue(true);
                    layer2.visibleProperty().setValue(false);
                    layer3.visibleProperty().setValue(true);
                    gc2.setFill(Color.BLUE);
                    break;
                case "Polylines alone":
                    scene.setFill(Color.YELLOW);
                    layer1.visibleProperty().setValue(false);
                    layer2.visibleProperty().setValue(true);
                    layer3.visibleProperty().setValue(true);
                    gc2.strokePolyline(XChimp,YChimp, nPointsChimp);
                    gc3.strokePolyline(XHuman,YHuman, nPointsHuman);
                    //Captured coordinates of points  are reported
                    report();
                    break;

            }
        });
    cb.setValue("Draw a single polyline: red is activated");
    borderPane.setTop(cb);
    root.getChildren().add(cb);
```

```
    vb.setLayoutX(5);
    vb.setLayoutY(50);
    vb.setSpacing(100);

    root.getChildren().addAll(vb);

    //Image is loaded
    //Image image  = getImage();
    Image image  = getImage(stage);
    final ImageView imageView  = new ImageView(image);
    imageView.setEffect(shadow);
    gc1.drawImage(image, 50, 100);

    stage.show();

}


//This method gets the image from the posted directory
static private Image getImage()
{
    //Path to folder with image
    File file = new File("/home/jose/jose/AAjoser"
            + "/BodyParts3D/ChimpHumanHand"
            + "/chimpanzee_hands_manual_intermediate"
            + "_phalanx_3_Medial_thumb.jpg");
    //Image is decoded from file
    Image image = new Image(file.toURI().toString());
    return image;
}


//This method gets an image using the fileChooser
static private Image getImage(Stage stage)
{
    FileChooser fileChooser = new FileChooser();
    //Set extension filter
    FileChooser.ExtensionFilter extFilterJPG
            = new FileChooser.ExtensionFilter(
```

```
                    "JPG files (*.jpg)", "*.jpg");
    FileChooser.ExtensionFilter extFilterPNG
            = new FileChooser.ExtensionFilter(
                    "PNG files (*.png)", "*.png");
    fileChooser.getExtensionFilters()
      .addAll(extFilterJPG, extFilterPNG);
    //Show open file dialog
    File file = fileChooser.showOpenDialog(stage);
    //Image is decoded from file
    Image image = new Image(file.toURI().toString());
    return image;
}


static private void report()
{
    System.out.println("Chimp: x-coordinates");
    for (int i = 0; i < nPointsChimp; i++)
        System.out.print((int) XChimp[i] + ", ");
    System.out.println();
    System.out.println("Chimp: y-coordinates");
    for (int i = 0; i < nPointsChimp; i++)
        System.out.print((int) YChimp[i] + ", ");
    System.out.println();
    System.out.println("Human: x-coordinates");
    for (int i = 0; i < nPointsHuman; i++)
        System.out.print((int) XHuman[i] + ", ");
    System.out.println();
    System.out.println("Human: y-coordinates");
    for (int i = 0; i < nPointsHuman; i++)
        System.out.print((int) YHuman[i] + ", ");
    System.out.println();
    //Output gotten by the Author:
    /*
    x-coordinates
    106,  99,  97,  96,  90,  86,  85,  83,  91,  98, 105, 111,
    114, 114, 109, 105, 103, 104, 105, 109, 114, 111, 106,
    y-coordinates
     99, 102, 108, 114, 128, 147, 185, 193, 201, 197, 195, 197,
    193, 190, 180, 166, 152, 137, 126, 117, 109, 104,  99,
```

```
            */
}


    public static void main(String[] args)
    {
        launch(args);
    }

}//End of main class M130 HandChimpHuman5
```

130 ***Exercise.*** *Run the previous program and play with the code. Report the coordinates of your polylines. Answer*

131 ***To proceed further, we will use program XyToF2 (in the attached file) as primer.***

We can now turn to compute the 3D-mesh generated by the dorsal and medial views of the chosen bone. Let us learn how to do that in various steps. To begin with, let us compute the 3D mesh associated to a cone, i.e., when the polyline consists of just one stroke.

132 ***The following code computes the mesh of a truncated cone generated by a stroke that spins around the z-axis.***

```
//Program M132 HandChimpHuman6
//This program calculates and draws the 3D mesh
//of the truncated cone
//generated by a stroke that orbits around
//the z-axis.


package handchimphuman6;



import javafx.application.Application;
import static javafx.application.Application.launch;
```

```
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.PerspectiveCamera;
import javafx.scene.PointLight;
import javafx.scene.Scene;
import javafx.scene.control.MenuBar;
import javafx.scene.input.KeyEvent;
import javafx.scene.input.MouseEvent;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.Box;
import javafx.scene.shape.CullFace;
import javafx.scene.shape.DrawMode;
import javafx.scene.shape.MeshView;;
import javafx.scene.shape.TriangleMesh;
import javafx.scene.transform.Rotate;
import javafx.scene.transform.Scale;
import javafx.scene.transform.Translate;
import javafx.stage.Stage;



public class HandChimpHuman6 extends Application
{

    //Function to be graphicated.
    //It is radius as a linear dependence on z.
    static private float radius(float z)
    {
        //stroke in 2D: input z, output = radius;
        float radius = 3*z + 100;
        return radius;
    }


    final Group root = new Group();

    final Group axisGroup = new Group();

    final PerspectiveCamera camera
```

```
        = new PerspectiveCamera(true);
 final Xform cameraXform = new Xform();
 final Xform cameraXform2 = new Xform();
 final Xform cameraXform3 = new Xform();

 //Parameter to regulate the size of graphic
 final double cameraDistance = 7850;//450
 final Xform world3DXform = new Xform();
 final Xform bonesXform = new Xform();


 double mousePosX;
 double mousePosY;
 double mouseOldX;
 double mouseOldY;
 double mouseDeltaX;
 double mouseDeltaY;



final Xform guidesXform = new Xform();
//Menubar declaration and initialization
 MenuBar menuBar = new MenuBar();


  private void buildCamera() {
     root.getChildren().add(cameraXform);
     cameraXform.getChildren().add(cameraXform2);
     cameraXform2.getChildren().add(cameraXform3);
     cameraXform3.getChildren().add(camera);
     cameraXform3.setRotateZ(180.0);

     camera.setNearClip(0.1);
     camera.setFarClip(10000.0);
     camera.setTranslateZ(-cameraDistance);
     cameraXform.ry.setAngle(320.0);
     cameraXform.rx.setAngle(40);
 }
```

```
    private void buildAxes() {
        System.out.println("buildAxes()");
        final PhongMaterial redMaterial
                = new PhongMaterial();
        redMaterial.setDiffuseColor(Color.DARKRED);
        redMaterial.setSpecularColor(Color.RED);

        final PhongMaterial greenMaterial
                = new PhongMaterial();
        greenMaterial.setDiffuseColor(Color.DARKGREEN);
        greenMaterial.setSpecularColor(Color.GREEN);

        final PhongMaterial blueMaterial
                = new PhongMaterial();
        blueMaterial.setDiffuseColor(Color.DARKBLUE);
        blueMaterial.setSpecularColor(Color.BLUE);

        final Box xAxis = new Box(640.0, 1, 1);
        final Box yAxis = new Box(1, 640.0, 1);
        final Box zAxis = new Box(1, 1, 640.0);

        xAxis.setMaterial(redMaterial);
        yAxis.setMaterial(greenMaterial);
        zAxis.setMaterial(blueMaterial);

        axisGroup.getChildren().addAll(xAxis, yAxis, zAxis);
        axisGroup.setVisible(false);
        world3DXform.getChildren().addAll(axisGroup);
    }



private void handleMouse(Scene scene)
{
    scene.setOnMousePressed((MouseEvent me) ->
    {
        mousePosX = me.getSceneX();
        mousePosY = me.getSceneY();
        mouseOldX = me.getSceneX();
```

```
        mouseOldY = me.getSceneY();
    });
    scene.setOnMouseDragged((MouseEvent me) ->
    {
        mouseOldX = mousePosX;
        mouseOldY = mousePosY;
        mousePosX = me.getSceneX();
        mousePosY = me.getSceneY();
        mouseDeltaX = (mousePosX - mouseOldX);
        mouseDeltaY = (mousePosY - mouseOldY);

        double modifier = 1.0;
        double modifierFactor = 0.1;

        if (me.isControlDown()) {
            modifier = 0.1;
        }
        if (me.isShiftDown()) {
            modifier = 10.0;
        }
        if (me.isPrimaryButtonDown()) {
            cameraXform.ry.setAngle(cameraXform
                    .ry.getAngle()
                    - mouseDeltaX*modifierFactor
                            *modifier*2.0);  // +
            cameraXform.rx.setAngle(cameraXform
                    .rx.getAngle()
                    + mouseDeltaY*modifierFactor
                            *modifier*2.0);  // -
        }
        else if (me.isSecondaryButtonDown()) {
            double z = camera.getTranslateZ();
            double newZ = z
            + mouseDeltaX*modifierFactor*modifier;
            camera.setTranslateZ(newZ);
        }
        else if (me.isMiddleButtonDown()) {
            cameraXform2.t.setX(cameraXform2.t.getX()
                + mouseDeltaX*modifierFactor*modifier*0.3);
            cameraXform2.t.setY(cameraXform2.t.getY()
```

```
                + mouseDeltaY*modifierFactor*modifier*0.3);
        }
        });
}

private void handleKeyboard(Scene scene)
{

    scene.setOnKeyPressed((KeyEvent event) -> {

        switch (event.getCode()) {
            case A: //Hide axis
                if (event.isControlDown()) {
                    if (axisGroup.isVisible()) {
                        System.out.println(
                                "setVisible(false)");
                        axisGroup.setVisible(false);
                    }
                    else {
                        System.out.println(
                                "setVisible(true)");
                        axisGroup.setVisible(true);
                    }
                }
                break;
        }
    });
}



@Override
public void start(Stage primaryStage)
{

    primaryStage.setTitle("Function graphicator. "
            + "Axes = ctrl+A");
    //The whole drawing  is created
    //as a JavaFX space-like object
    final Group parent = new Group(create3dContent());
```

```
    //Window of the scene
    final Scene scene = new Scene(root, 900, 900, true);
    //Background
    scene.setFill(Color.SKYBLUE);
    buildCamera();
    primaryStage.setScene(scene);
    primaryStage.show();
    buildAxes();
    handleKeyboard(scene);
    handleMouse(scene);
    scene.setCamera(camera);
    root.getChildren().addAll(parent);

}

//Manager of graphic
public Node create3dContent()
{
    Graphic b = new Graphic();
    b.getChildren().add( world3DXform);
    b.rx.setAngle(45);
    b.ry.setAngle(110);

    return new Group(b);
}


//A mesh is coverted into a meshView.
public class Graphic extends Group
{
final Rotate rx = new Rotate(0,Rotate.X_AXIS);
final Rotate ry = new Rotate(0,Rotate.Y_AXIS);
final Rotate rz = new Rotate(0,Rotate.Z_AXIS);

//The ambient of the graphic is a box
//of measures a, b, and c.
public  Graphic()
  {
    getTransforms().addAll(rz, ry, rx);
```

```
    //Mesh of the truncated cone
    TriangleMesh meshFunction
            = buildMeshFunction();
    PhongMaterial materialF  = new PhongMaterial();
    materialF.setDiffuseColor(Color.FUCHSIA);
    MeshView  meshViewFunction = new MeshView(meshFunction);
    meshViewFunction.setMaterial(materialF);
    meshViewFunction.setDrawMode(DrawMode.LINE);
    //meshViewFunction.setDrawMode(DrawMode.FILL);
    meshViewFunction.setCullFace(CullFace.NONE);

    //Illumination
    PointLight pointLight;
    pointLight = new PointLight(Color.ANTIQUEWHITE);
    pointLight.setTranslateX(150);
    pointLight.setTranslateY(-100);
    pointLight.setTranslateZ(-1000);

    getChildren().addAll(meshViewFunction,pointLight  );
  }
}//End of class Graphic



//The mesh of the truncated cone
static TriangleMesh buildMeshFunction()
{

    //Scales
    float zScale =   2f;
    float xyScale =  1;
    float zo = 100;
    //Increment along Z
    float deltaZ = 10;
    //Numbers of subdivisions along z
    int n = 50;

    double PI =  Math.PI;
    //Numbers of subdivisions along theta angle
```

```
int m = 66 ;
//Angle increment
double deltaTheta =  (2*PI/m) ;

//Array with coordinates of points of
//the mesh, each one with 3 coordinates
float points[] = new float[3* (n+1) * (m +1)];

//There are n*m tiny rectangles.
//Each rectangle has 2 triangles
int nFaces = 2*n*m;
//Each point of each face has 6 coordinates,
//3 for the vertex and
//3 for the marked points of the wallpaper (texture)
int faces[] = new int[nFaces * 6];


// Create points
//z-variable
for (int j = 0; j <= n; j++)
{
    //Z begins with zo and is incremented by deltaZ
    float zj = zo + deltaZ*j;
    float r = radius(zj);

    //Angle theta
    for (int i = 0; i <= m; i++)
    {
        double thetai = i*deltaTheta;
        int index = j * (m+1)* 3 + (i * 3);
        points[index] =   r
            * ( (float) Math.cos(thetai)) * xyScale;
        points[index + 1]  =   r
            * ( (float) Math.sin(thetai)) * xyScale;
        points[index + 2] = zj*zScale;

    }
}
```

```
//Faces are built.
//Two triangles for each rectangle.
for (int j = 0; j < n; j++)
{
    for (int i = 0; i < m; i++)
    {
        int p00 = j * (m+1) + i;
        int p01 = p00 + 1;
        int p10 = p00 + (m+1);
        int p11 = p10 + 1;
        int tc00 = 0;
        int tc01 = 0;
        int tc10 = 0;
        int tc11 = 0;
        //First triangle
        int index = (j * m * 6 + (i * 6)) * 2;
        faces[index + 0] = p00;
        faces[index + 1] = tc00;
        faces[index + 2] = p10;
        faces[index + 3] = tc10;
        faces[index + 4] = p11;
        faces[index + 5] = tc11;
        //second triangle
        index = index + 6;
        faces[index + 0] = p11;
        faces[index + 1] = tc11;
        faces[index + 2] = p01;
        faces[index + 3] = tc01;
        faces[index + 4] = p00;
        faces[index + 5] = tc00;
    }
}


TriangleMesh mesh = new TriangleMesh();
mesh.getPoints().addAll(points);

mesh.getTexCoords().addAll(0,0);
mesh.getFaces().addAll(faces);
```

```
    return mesh;
}

public static void main(String[] args)
{

    launch(args);
}

}//End of main class M112 MlgrmLeftHand4


//Outer class Xform.
//Machinery that enables rotations in vivo of the scene
//and the conformation of groups of control.
class Xform extends Group {

    public enum RotateOrder {
        XYZ, XZY, YXZ, YZX, ZXY, ZYX
    }

    public Translate t  = new Translate();
    public Translate p  = new Translate();
    public Translate ip = new Translate();
    public Rotate rx = new Rotate();
    { rx.setAxis(Rotate.X_AXIS); }
    public Rotate ry = new Rotate();
    { ry.setAxis(Rotate.Y_AXIS); }
    public Rotate rz = new Rotate();
    { rz.setAxis(Rotate.Z_AXIS); }
    public Scale s = new Scale();

    public Xform() {
        super();
        getTransforms().addAll(t, rz, ry, rx, s);
    }

    public Xform(RotateOrder rotateOrder) {
        super();
        // choose the order of rotations
```

```
        //based on the rotateOrder
        switch (rotateOrder) {
        case XYZ:
            getTransforms().addAll(t, p, rz, ry, rx, s, ip);
            break;
        case XZY:
            getTransforms().addAll(t, p, ry, rz, rx, s, ip);
            break;
        case YXZ:
            getTransforms().addAll(t, p, rz, rx, ry, s, ip);
            break;
        case YZX:
            getTransforms().addAll(t, p, rx, rz, ry, s, ip);
            break;
        case ZXY:
            getTransforms().addAll(t, p, ry, rx, rz, s, ip);
            break;
        case ZYX:
            getTransforms().addAll(t, p, rx, ry, rz, s, ip);
            break;
        }
    }

    public void setTranslate(double x, double y, double z) {
        t.setX(x);
        t.setY(y);
        t.setZ(z);
    }

    public void setTranslate(double x, double y) {
        t.setX(x);
        t.setY(y);
    }

    // Cannot override these methods as they are final:
    // public void setTranslateX(double x) { t.setX(x); }
    // public void setTranslateY(double y) { t.setY(y); }
    // public void setTranslateZ(double z) { t.setZ(z); }
    // Use these methods instead:
    public void setTx(double x) { t.setX(x); }
```

```
    public void setTy(double y) { t.setY(y); }
    public void setTz(double z) { t.setZ(z); }

    public void setRotate(double x, double y, double z) {
        rx.setAngle(x);
        ry.setAngle(y);
        rz.setAngle(z);
    }

    public void setRotateX(double x) { rx.setAngle(x); }
    public void setRotateY(double y) { ry.setAngle(y); }
    public void setRotateZ(double z) { rz.setAngle(z); }
    public void setRx(double x) { rx.setAngle(x); }
    public void setRy(double y) { ry.setAngle(y); }
    public void setRz(double z) { rz.setAngle(z); }

    public void setScale(double scaleFactor) {
        s.setX(scaleFactor);
        s.setY(scaleFactor);
        s.setZ(scaleFactor);
    }

    public void setScale(double x, double y, double z) {
        s.setX(x);
        s.setY(y);
        s.setZ(z);
    }

    // Cannot override these methods as they are final:
    // public void setScaleX(double x) { s.setX(x); }
    // public void setScaleY(double y) { s.setY(y); }
    // public void setScaleZ(double z) { s.setZ(z); }
    // Use these methods instead:
    public void setSx(double x) { s.setX(x); }
    public void setSy(double y) { s.setY(y); }
    public void setSz(double z) { s.setZ(z); }

    public void setPivot(double x, double y, double z) {
        p.setX(x);
        p.setY(y);
```

```
        p.setZ(z);
        ip.setX(-x);
        ip.setY(-y);
        ip.setZ(-z);
    }

    public void reset() {
        t.setX(0.0);
        t.setY(0.0);
        t.setZ(0.0);
        rx.setAngle(0.0);
        ry.setAngle(0.0);
        rz.setAngle(0.0);
        s.setX(1.0);
        s.setY(1.0);
        s.setZ(1.0);
        p.setX(0.0);
        p.setY(0.0);
        p.setZ(0.0);
        ip.setX(0.0);
        ip.setY(0.0);
        ip.setZ(0.0);
    }

    public void resetTSP() {
        t.setX(0.0);
        t.setY(0.0);
        t.setZ(0.0);
        s.setX(1.0);
        s.setY(1.0);
        s.setZ(1.0);
        p.setX(0.0);
        p.setY(0.0);
        p.setZ(0.0);
        ip.setX(0.0);
        ip.setY(0.0);
        ip.setZ(0.0);
    }
}//End of class Xform; End of program M132 MlgrmLeftHand6
```

133 ***Exercise.*** *Run the previous program and play with the code.*

In our next step to go from a silhouette, that is a 2D-curve, to a 3D-mesh, we assume that we have a polyline and that the generated mesh has cylindrical symmetry, i-e-. that it looks the same if it is rotated around the z-axis.

134 ***Exercise. Develop a program to draw a polycone generated by a polyline that orbits around the z-axis.*** *Answer*

135 ***Exercise: ellipsoidal cone.*** *An ellipse results from enlarging a unitary circle,* `a` *units in the x-direction and* `b` *units in the y-direction. Modify a previous program to calculate the mesh of such an elliptical cone.* *Answer*

136 ***Polar interpolation .*** *We know how to calculate the mesh of a cone whose section can be cylindrical or ellipsoidal. In this case, there are circles or ellipses generated by the same z-value. To get more complex silhouettes we need to calculate meshes that walk over an elliptical cone but that connect points whose z-value are different. How to do this is shown by the next program:*

```
//Program M136 HandChimpHuman8
//Polar interpolation.
//Points A is over the XZ-plane,
//point B is over the YZ-plane.
//We   connect points A and B  by an ellipsoidal arc.
//The same will be done with points C and D.
//The mesh in within the two arcs
//will be calculated and drawn.

package handchimphuman8;

import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.PerspectiveCamera;
import javafx.scene.PointLight;
import javafx.scene.Scene;
import javafx.scene.control.MenuBar;
import javafx.scene.input.KeyEvent;
import javafx.scene.input.MouseEvent;
```

```
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.Box;
import javafx.scene.shape.CullFace;
import javafx.scene.shape.DrawMode;
import javafx.scene.shape.MeshView;;
import javafx.scene.shape.TriangleMesh;
import javafx.scene.transform.Rotate;
import javafx.scene.transform.Scale;
import javafx.scene.transform.Translate;
import javafx.stage.Stage;



public class HandChimpHuman8 extends Application
{

    //Coordinates of 4 points, each one with 3 coordinates
    private static float[][]  P = new float[][]
    {
        {50, 0, 100}, //In xz-plane
        {0, 200, 280}, //In yz-plane
        {250, 0, 300}, //In xz-plane
        {0, 700, 500} //In yz-plane
    };


    /*
    //Test: 1/4 of a cylinder
    //Coordinates of 4 points, each one with 3 coordinates
    private static float[][]  P = new float[][]
    {
        {200, 0, 100}, //In xz-plane
        {0, 200, 100}, //In yz-plane
        {200, 0, 500}, //In xz-plane
        {0, 200, 500} //In yz-plane
    };
    */
```

```
final Group root = new Group();

final Group axisGroup = new Group();

final PerspectiveCamera camera
        = new PerspectiveCamera(true);
final Xform cameraXform = new Xform();
final Xform cameraXform2 = new Xform();
final Xform cameraXform3 = new Xform();

//Parameter to regulate the size of graphic
final double cameraDistance = 7850;
final Xform world3DXform = new Xform();
final Xform bonesXform = new Xform();


double mousePosX;
double mousePosY;
double mouseOldX;
double mouseOldY;
double mouseDeltaX;
double mouseDeltaY;



final Xform guidesXform = new Xform();
//Menubar declaration and initialization
 MenuBar menuBar = new MenuBar();


  private void buildCamera() {
     root.getChildren().add(cameraXform);
     cameraXform.getChildren().add(cameraXform2);
     cameraXform2.getChildren().add(cameraXform3);
     cameraXform3.getChildren().add(camera);
     cameraXform3.setRotateZ(180.0);

     camera.setNearClip(0.1);
     camera.setFarClip(10000.0);
     camera.setTranslateZ(-cameraDistance);
```

```
        cameraXform.ry.setAngle(320.0);
        cameraXform.rx.setAngle(40);
    }



    private void buildAxes() {
        System.out.println("buildAxes()");
        final PhongMaterial redMaterial
                = new PhongMaterial();
        redMaterial.setDiffuseColor(Color.DARKRED);
        redMaterial.setSpecularColor(Color.RED);

        final PhongMaterial greenMaterial
                = new PhongMaterial();
        greenMaterial.setDiffuseColor(Color.DARKGREEN);
        greenMaterial.setSpecularColor(Color.GREEN);

        final PhongMaterial blueMaterial
                = new PhongMaterial();
        blueMaterial.setDiffuseColor(Color.DARKBLUE);
        blueMaterial.setSpecularColor(Color.BLUE);

        final Box xAxis = new Box(640.0, 1, 1);
        final Box yAxis = new Box(1, 640.0, 1);
        final Box zAxis = new Box(1, 1, 640.0);

        xAxis.setMaterial(redMaterial);
        yAxis.setMaterial(greenMaterial);
        zAxis.setMaterial(blueMaterial);

        axisGroup.getChildren().addAll(xAxis, yAxis, zAxis);
        axisGroup.setVisible(false);
        world3DXform.getChildren().addAll(axisGroup);
    }



private void handleMouse(Scene scene)
{
```

```
scene.setOnMousePressed((MouseEvent me) ->
{
    mousePosX = me.getSceneX();
    mousePosY = me.getSceneY();
    mouseOldX = me.getSceneX();
    mouseOldY = me.getSceneY();
});
scene.setOnMouseDragged((MouseEvent me) ->
{
    mouseOldX = mousePosX;
    mouseOldY = mousePosY;
    mousePosX = me.getSceneX();
    mousePosY = me.getSceneY();
    mouseDeltaX = (mousePosX - mouseOldX);
    mouseDeltaY = (mousePosY - mouseOldY);

    double modifier = 1.0;
    double modifierFactor = 0.1;

    if (me.isControlDown()) {
        modifier = 0.1;
    }
    if (me.isShiftDown()) {
        modifier = 10.0;
    }
    if (me.isPrimaryButtonDown()) {
        cameraXform.ry.setAngle(cameraXform
                .ry.getAngle()
                - mouseDeltaX*modifierFactor
                        *modifier*2.0);  // +
        cameraXform.rx.setAngle(cameraXform
                .rx.getAngle()
                + mouseDeltaY*modifierFactor
                        *modifier*2.0);  // -
    }
    else if (me.isSecondaryButtonDown()) {
        double z = camera.getTranslateZ();
        double newZ = z
        + mouseDeltaX*modifierFactor*modifier;
        camera.setTranslateZ(newZ);
```

```
        }
        else if (me.isMiddleButtonDown()) {
            cameraXform2.t.setX(cameraXform2.t.getX()
                + mouseDeltaX*modifierFactor*modifier*0.3);
            cameraXform2.t.setY(cameraXform2.t.getY()
                + mouseDeltaY*modifierFactor*modifier*0.3);
        }
        });
}

private void handleKeyboard(Scene scene)
{

    scene.setOnKeyPressed((KeyEvent event) -> {

        switch (event.getCode()) {
            case A: //Hide axis
                if (event.isControlDown()) {
                    if (axisGroup.isVisible()) {
                        System.out.println(
                                "setVisible(false)");
                        axisGroup.setVisible(false);
                    }
                    else {
                        System.out.println(
                                "setVisible(true)");
                        axisGroup.setVisible(true);
                    }
                }
                break;
        }
    });
}



@Override
public void start(Stage primaryStage)
{
```

```
    primaryStage.setTitle("Function graphicator. "
            + "Axes = ctrl+A");
    //The whole drawing  is created
    //as a JavaFX space-like object
    final Group parent = new Group(create3dContent());

    //Window of the scene
    final Scene scene = new Scene(root, 900, 900, true);
    //Background
    scene.setFill(Color.SKYBLUE);
    buildCamera();
    primaryStage.setScene(scene);
    primaryStage.show();
    buildAxes();
    handleKeyboard(scene);
    handleMouse(scene);
    scene.setCamera(camera);
    root.getChildren().addAll(parent);

}

//Manager of graphic
public Node create3dContent()
{
    Graphic b = new Graphic();
    b.getChildren().add( world3DXform);
    b.rx.setAngle(45);
    b.ry.setAngle(110);

    return new Group(b);
}


//A mesh is coverted into a meshView.
public class Graphic extends Group
{
final Rotate rx = new Rotate(0,Rotate.X_AXIS);
final Rotate ry = new Rotate(0,Rotate.Y_AXIS);
```

```
final Rotate rz = new Rotate(0,Rotate.Z_AXIS);

//The ambient of the graphic is a box
//of measures a, b, and c.
public  Graphic()
  {
    getTransforms().addAll(rz, ry, rx);

    //Mesh of the truncated cone
    TriangleMesh meshFunction
            = buildMeshFunction();
    PhongMaterial materialF  = new PhongMaterial();
    materialF.setDiffuseColor(Color.FUCHSIA);
    MeshView  meshViewFunction = new MeshView(meshFunction);
    meshViewFunction.setMaterial(materialF);
    meshViewFunction.setDrawMode(DrawMode.LINE);
    //meshViewFunction.setDrawMode(DrawMode.FILL);
    meshViewFunction.setCullFace(CullFace.NONE);

    //Illumination
    PointLight pointLight;
    pointLight = new PointLight(Color.ANTIQUEWHITE);
    pointLight.setTranslateX(150);
    pointLight.setTranslateY(-100);
    pointLight.setTranslateZ(-1000);

    getChildren().addAll(meshViewFunction,pointLight  );
  }
}//End of class Graphic



//The mesh of the truncated cone
static TriangleMesh buildMeshFunction()
{

    //Scales
    float zScale =   1f;
    float xyScale =  1;
```

```
double PIHalf =  Math.PI/2;
//Numbers of subdivisions along theta angle
int m = 18 ;
//Angle increment
double deltaTheta =  (PIHalf/ m) ;

//Array with coordinates of points of
//the mesh, each one with 3 coordinates
float points[] = new float[3 * 2*(m +1)];

//There are nm tiny rectangles.
//Each rectangle has 2 triangles
int nFaces = 2*m;
//Each point of each face has 6 coordinates,
//3 for the vertex and
//3 for the marked points of the wallpaper (texture)
int faces[] = new int[nFaces * 6];


// Create points
//z-variable




for (int j = 0; j < 2; j++)
{
    //Points that connect P[2*j] to P[2*j + 1]
    //Z begins with zo and is incremented by deltaZ
    float zo = P[2*j][2];
    float deltaZ = ( P[2*j + 1][2] -  P[2*j][2])/m;

    //Angle theta
    for (int i = 0; i <= m; i++)
    {
        float zi = zo + deltaZ*i;
        float ra = P[2*j][0];
```

```
            float rb = P[2*j+1][1];
            double thetai = -i*deltaTheta;
            int index = j * (m+1)* 3 + (i * 3);
            points[index] =   ra
                * ( (float) Math.cos(thetai)) * xyScale;
            points[index + 1]  =   rb
                * ( (float) Math.sin(thetai)) * xyScale;
            points[index + 2] = zi*zScale;

        }
    }


    //Faces are built.
    //Two triangles for each rectangle.
    for (int j = 0; j < 1; j++)
    {
        for (int i = 0; i < m; i++)
        {
            int p00 = j * (m+1) + i;
            int p01 = p00 + 1;
            int p10 = p00 + (m+1);
            int p11 = p10 + 1;
            int tc00 = 0;
            int tc01 = 0;
            int tc10 = 0;
            int tc11 = 0;
            //First triangle
            int index = (j * m * 6 + (i * 6)) * 2;
            faces[index + 0] = p00;
            faces[index + 1] = tc00;
            faces[index + 2] = p10;
            faces[index + 3] = tc10;
            faces[index + 4] = p11;
            faces[index + 5] = tc11;
            //second triangle
            index = index + 6;
            faces[index + 0] = p11;
            faces[index + 1] = tc11;
            faces[index + 2] = p01;
```

```
            faces[index + 3] = tc01;
            faces[index + 4] = p00;
            faces[index + 5] = tc00;
        }
    }


    TriangleMesh mesh = new TriangleMesh();
    mesh.getPoints().addAll(points);

    mesh.getTexCoords().addAll(0,0);
    mesh.getFaces().addAll(faces);

    return mesh;
}

public static void main(String[] args)
{

    launch(args);
}

}//End of main class M136 MlgrmLeftHand8


//Outer class Xform.
//Machinery that enables rotations in vivo of the scene
//and the conformation of groups of control.
class Xform extends Group {

    public enum RotateOrder {
        XYZ, XZY, YXZ, YZX, ZXY, ZYX
    }

    public Translate t  = new Translate();
    public Translate p  = new Translate();
    public Translate ip = new Translate();
    public Rotate rx = new Rotate();
    { rx.setAxis(Rotate.X_AXIS); }
    public Rotate ry = new Rotate();
```

```
    { ry.setAxis(Rotate.Y_AXIS); }
    public Rotate rz = new Rotate();
    { rz.setAxis(Rotate.Z_AXIS); }
    public Scale s = new Scale();

    public Xform() {
        super();
        getTransforms().addAll(t, rz, ry, rx, s);
    }

    public Xform(RotateOrder rotateOrder) {
        super();
        // choose the order of rotations
        //based on the rotateOrder
        switch (rotateOrder) {
        case XYZ:
            getTransforms().addAll(t, p, rz, ry, rx, s, ip);
            break;
        case XZY:
            getTransforms().addAll(t, p, ry, rz, rx, s, ip);
            break;
        case YXZ:
            getTransforms().addAll(t, p, rz, rx, ry, s, ip);
            break;
        case YZX:
            getTransforms().addAll(t, p, rx, rz, ry, s, ip);
            break;
        case ZXY:
            getTransforms().addAll(t, p, ry, rx, rz, s, ip);
            break;
        case ZYX:
            getTransforms().addAll(t, p, rx, ry, rz, s, ip);
            break;
        }
    }

    public void setTranslate(double x, double y, double z) {
        t.setX(x);
        t.setY(y);
        t.setZ(z);
```

```
    }

    public void setTranslate(double x, double y) {
        t.setX(x);
        t.setY(y);
    }

    // Cannot override these methods as they are final:
    // public void setTranslateX(double x) { t.setX(x); }
    // public void setTranslateY(double y) { t.setY(y); }
    // public void setTranslateZ(double z) { t.setZ(z); }
    // Use these methods instead:
    public void setTx(double x) { t.setX(x); }
    public void setTy(double y) { t.setY(y); }
    public void setTz(double z) { t.setZ(z); }

    public void setRotate(double x, double y, double z) {
        rx.setAngle(x);
        ry.setAngle(y);
        rz.setAngle(z);
    }

    public void setRotateX(double x) { rx.setAngle(x); }
    public void setRotateY(double y) { ry.setAngle(y); }
    public void setRotateZ(double z) { rz.setAngle(z); }
    public void setRx(double x) { rx.setAngle(x); }
    public void setRy(double y) { ry.setAngle(y); }
    public void setRz(double z) { rz.setAngle(z); }

    public void setScale(double scaleFactor) {
        s.setX(scaleFactor);
        s.setY(scaleFactor);
        s.setZ(scaleFactor);
    }

    public void setScale(double x, double y, double z) {
        s.setX(x);
        s.setY(y);
        s.setZ(z);
    }
```

```
    // Cannot override these methods as they are final:
    // public void setScaleX(double x) { s.setX(x); }
    // public void setScaleY(double y) { s.setY(y); }
    // public void setScaleZ(double z) { s.setZ(z); }
    // Use these methods instead:
    public void setSx(double x) { s.setX(x); }
    public void setSy(double y) { s.setY(y); }
    public void setSz(double z) { s.setZ(z); }

    public void setPivot(double x, double y, double z) {
        p.setX(x);
        p.setY(y);
        p.setZ(z);
        ip.setX(-x);
        ip.setY(-y);
        ip.setZ(-z);
    }

    public void reset() {
        t.setX(0.0);
        t.setY(0.0);
        t.setZ(0.0);
        rx.setAngle(0.0);
        ry.setAngle(0.0);
        rz.setAngle(0.0);
        s.setX(1.0);
        s.setY(1.0);
        s.setZ(1.0);
        p.setX(0.0);
        p.setY(0.0);
        p.setZ(0.0);
        ip.setX(0.0);
        ip.setY(0.0);
        ip.setZ(0.0);
    }

    public void resetTSP() {
        t.setX(0.0);
        t.setY(0.0);
```

```
        t.setZ(0.0);
        s.setX(1.0);
        s.setY(1.0);
        s.setZ(1.0);
        p.setX(0.0);
        p.setY(0.0);
        p.setZ(0.0);
        ip.setX(0.0);
        ip.setY(0.0);
        ip.setZ(0.0);
    }
}//End of class Xform; End of program M136 MlgrmLeftHand8
```

137 ***Exercise.*** *Run the previous program and play with the code.*

We can use the idea of the previous program to devise a mesh generated by two polylines if one of them represents, say, a side of the lateral view of a 3D body and the other the corresponding side of a perpendicular view.

138 ***Exercise.*** *Use program* `M130 HandChimpHuman5` *to associate two open polylines in the same direction to the medial view of bone* `chimpanzee hands manual intermediate phalanx 3`*. Each polyline shall represent a lateral part of the silhouette of the bone and together conform the whole silhouette. Do the same with the dorsal view of the same bone. At the end you must have 4 polylines, i.e., 8 arrays. Report your polylines. Invitation: the polylines shall observe as similar z-partitions as possible* Answer

7.6 Our prediction as a 3D-mesh

The assumption that the chimp, the bonobo and the homo are directly related by a common ancestor leads to the following mandatory prediction: apart from ordinary homo, you and me, many more variants must exist, in particular, one whose phalanges of his hand must be curved on the inner side to accommodate thicker tendons and bigger muscles with more space to get contact with opposite parts. This will make them more competitive for foraging and in struggles both with nude nails and with clubs.

We are ready now to learn how to make a 3D illustration of this prediction beginning with polylines related to 2D portraits.

139 ***Let us develop a program to compose a 3D mesh using the four polylines captured in the previous exercise and that describes the silhouettes of two complementary views of the same bone.***

```
/*
Program M139 HandChimpHuman9
This program  draws a 3D mesh
generated by four polylines that correspond
with two mutually orthogonal views of a bone of chimp.

Data constitute  an edited version of the output of program
M130 HandChimpHuman5 over two views of a bone of
the hand of a chimp.

This represents a mandatory prediction of the supposition
that the human hand came by Natural Evolution
from a specimen whose direct descendants
are the chimp and the homo:
apart from ordinary homo, you and me,
many more variants must exist, in particular,
one whose  phalanges of his
hand must be curved on the inner side
to accommodate thicker tendons and bigger muscles
to get more competitive for foraging and in struggles
both with  nude nails and with clubs.
 */

package handchimphuman9;

import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.PerspectiveCamera;
import javafx.scene.PointLight;
import javafx.scene.Scene;
import javafx.scene.control.MenuBar;
import javafx.scene.input.KeyEvent;
import javafx.scene.input.MouseEvent;
import javafx.scene.paint.Color;
```

```
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.Box;
import javafx.scene.shape.CullFace;
import javafx.scene.shape.DrawMode;
import javafx.scene.shape.MeshView;;
import javafx.scene.shape.TriangleMesh;
import javafx.scene.transform.Rotate;
import javafx.scene.transform.Scale;
import javafx.scene.transform.Translate;
import javafx.stage.Stage;



public class HandChimpHuman9 extends Application
{


//Four edited polylones representing 4 lateral strokes of the
//two views that are mutually perpendicular:

//Polyline :
//chimpanzee_hands_manual_intermediate_phalanx_3_Medial_thumb
//x-coordinates, left side
private static float[] xMedialViewLeftSide  = new float[]
{
    105,  98,  95,  95,  90,  87,  87,  87,  85,  84,  91, 101
};
//z-coordinates, left side
private static float[] yMedialViewLeftSide  = new float[]
{
    101, 104, 109, 113, 126, 142, 161, 178, 187, 191, 200, 197
};

//chimpanzee_hands_manual_intermediate_phalanx_3_Dorsal_thumb
//x-coordinates, right side
private static float[] xMedialViewRightSide  = new float[]
{
    105, 112, 111, 107, 105, 102, 104, 108, 112, 114, 109, 101
};
//z-coordinates, right side
```

```
private static float[] yMedialViewRightSide  = new float[]
{
    101, 106, 110, 118, 128, 142, 161, 176, 185, 190, 199, 197
};

//Complementary Polylines :
//chimpanzee_hands_manual_intermediate_phalanx_3_Medial_thumb
//x-coordinates of left side
private static float[] xDorsalViewLeftSide  = new float[]
{
    101,  98,  92,  89,  89,  90,  87,  87,  82,  80,  86, 100
};
//z-coordinates of left side
private static float[] yDorsalViewLeftSide  = new float[]
{
    102, 101, 102, 112, 128, 145, 156, 168, 180, 188, 198, 200
};

//chimpanzee_hands_manual_intermediate_phalanx_3_Dorsal_thumb
//x-coordinates of right side
private static float[] xDorsalViewRightSide  = new float[]
{
    101, 109, 111, 114, 113, 114, 116, 118, 117, 120, 113, 100
};
//y-coordinates of right side
private static float[] yDorsalViewRightSide  = new float[]
{
    102, 100, 106, 111, 129, 146, 160, 171, 179, 186, 198, 200
};
```

```
    final Group root = new Group();

    final Group axisGroup = new Group();

    final PerspectiveCamera camera
            = new PerspectiveCamera(true);
    final Xform cameraXform = new Xform();
```

```
final Xform cameraXform2 = new Xform();
final Xform cameraXform3 = new Xform();

//Parameter to regulate the size of graphic
final double cameraDistance = 1450;
final Xform world3DXform = new Xform();
final Xform bonesXform = new Xform();


double mousePosX;
double mousePosY;
double mouseOldX;
double mouseOldY;
double mouseDeltaX;
double mouseDeltaY;



final Xform guidesXform = new Xform();
//Menubar declaration and initialization
 MenuBar menuBar = new MenuBar();


  private void buildCamera() {
     root.getChildren().add(cameraXform);
     cameraXform.getChildren().add(cameraXform2);
     cameraXform2.getChildren().add(cameraXform3);
     cameraXform3.getChildren().add(camera);
     cameraXform3.setRotateZ(180.0);

     camera.setNearClip(0.1);
     camera.setFarClip(10000.0);
     camera.setTranslateZ(-cameraDistance);
     cameraXform.ry.setAngle(320.0);
     cameraXform.rx.setAngle(40);
 }



 private void buildAxes() {
```

```
        System.out.println("buildAxes()");
        final PhongMaterial redMaterial
                = new PhongMaterial();
        redMaterial.setDiffuseColor(Color.DARKRED);
        redMaterial.setSpecularColor(Color.RED);

        final PhongMaterial greenMaterial
                = new PhongMaterial();
        greenMaterial.setDiffuseColor(Color.DARKGREEN);
        greenMaterial.setSpecularColor(Color.GREEN);

        final PhongMaterial blueMaterial
                = new PhongMaterial();
        blueMaterial.setDiffuseColor(Color.DARKBLUE);
        blueMaterial.setSpecularColor(Color.BLUE);

        final Box xAxis = new Box(640.0, 1, 1);
        final Box yAxis = new Box(1, 640.0, 1);
        final Box zAxis = new Box(1, 1, 640.0);

        xAxis.setMaterial(redMaterial);
        yAxis.setMaterial(greenMaterial);
        zAxis.setMaterial(blueMaterial);

        axisGroup.getChildren().addAll(xAxis, yAxis, zAxis);
        axisGroup.setVisible(false);
        world3DXform.getChildren().addAll(axisGroup);
    }



private void handleMouse(Scene scene)
{
    scene.setOnMousePressed((MouseEvent me) ->
    {
        mousePosX = me.getSceneX();
        mousePosY = me.getSceneY();
        mouseOldX = me.getSceneX();
        mouseOldY = me.getSceneY();
    });
```

```
scene.setOnMouseDragged((MouseEvent me) ->
{
    mouseOldX = mousePosX;
    mouseOldY = mousePosY;
    mousePosX = me.getSceneX();
    mousePosY = me.getSceneY();
    mouseDeltaX = (mousePosX - mouseOldX);
    mouseDeltaY = (mousePosY - mouseOldY);

    double modifier = 1.0;
    double modifierFactor = 0.1;

    if (me.isControlDown()) {
        modifier = 0.1;
    }
    if (me.isShiftDown()) {
        modifier = 10.0;
    }
    if (me.isPrimaryButtonDown()) {
        cameraXform.ry.setAngle(cameraXform
                .ry.getAngle()
                - mouseDeltaX*modifierFactor
                        *modifier*2.0);  // +
        cameraXform.rx.setAngle(cameraXform
                .rx.getAngle()
                + mouseDeltaY*modifierFactor
                        *modifier*2.0);  // -
    }
    else if (me.isSecondaryButtonDown()) {
        double z = camera.getTranslateZ();
        double newZ = z
        + mouseDeltaX*modifierFactor*modifier;
        camera.setTranslateZ(newZ);
    }
    else if (me.isMiddleButtonDown()) {
        cameraXform2.t.setX(cameraXform2.t.getX()
            + mouseDeltaX*modifierFactor*modifier*0.3);
        cameraXform2.t.setY(cameraXform2.t.getY()
            + mouseDeltaY*modifierFactor*modifier*0.3);
    }
```

```
        });
}

private void handleKeyboard(Scene scene)
{

    scene.setOnKeyPressed((KeyEvent event) -> {

        switch (event.getCode()) {
            case A: //Hide axis
                if (event.isControlDown()) {
                    if (axisGroup.isVisible()) {
                        System.out.println(
                                "setVisible(false)");
                        axisGroup.setVisible(false);
                    }
                    else {
                        System.out.println(
                                "setVisible(true)");
                        axisGroup.setVisible(true);
                    }
                }
                break;
        }
    });
}



@Override
public void start(Stage primaryStage)
{
    primaryStage.setTitle("Evolutionary prediction of the form of  phalages. "
            + "Axes = ctrl+A");
    //The whole drawing  is created
    //as a JavaFX space-like object
    final Group parent = new Group(create3dContent());

    //Window of the scene
    final Scene scene = new Scene(root, 900, 900, true);
```

```
    //Background
    scene.setFill(Color.SKYBLUE);
    buildCamera();
    primaryStage.setScene(scene);
    primaryStage.show();
    buildAxes();
    handleKeyboard(scene);
    handleMouse(scene);
    scene.setCamera(camera);
    root.getChildren().addAll(parent);

}

//Manager of graphic
public Node create3dContent()
{
    Graphic b = new Graphic();
    b.getChildren().add( world3DXform);
    b.rx.setAngle(45);
    b.ry.setAngle(110);

    return new Group(b);
}


//A mesh is coverted into a meshView.
public class Graphic extends Group
{
final Rotate rx = new Rotate(0,Rotate.X_AXIS);
final Rotate ry = new Rotate(0,Rotate.Y_AXIS);
final Rotate rz = new Rotate(0,Rotate.Z_AXIS);

//The ambient of the graphic is a box
//of measures a, b, and c.
public  Graphic()
  {
    getTransforms().addAll(rz, ry, rx);

    //Mesh of quadrant 0
    Xform meshXform0 = MeshQuadrant(
```

```
        xDorsalViewLeftSide, yDorsalViewLeftSide,
        xMedialViewRightSide, yMedialViewRightSide,
        0*Math.PI,
        Color.LEMONCHIFFON,
        //Color.GOLDENROD,
        +1);

    //Mesh of quadrant 1
    Xform meshXform1 = MeshQuadrant(
        xDorsalViewRightSide, yDorsalViewRightSide,
        xMedialViewRightSide, yMedialViewRightSide,
        0.5*Math.PI,
        Color.CYAN,
        //Color.GOLDENROD,
        -1);


    //Mesh of quadrant 2
    Xform meshXform2 = MeshQuadrant(
        xDorsalViewRightSide, yDorsalViewRightSide,
        xMedialViewLeftSide, yMedialViewLeftSide,
        1*Math.PI,
        Color.GREEN,
        //Color.GOLDENROD,
        1);


    //Mesh of quadrant 3
    Xform meshXform3 = MeshQuadrant(
        xDorsalViewLeftSide, yDorsalViewLeftSide,
        xMedialViewLeftSide, yMedialViewLeftSide,
        1.5*Math.PI,
        Color.RED,
        //Color.GOLDENROD,
         -1);
    //Four parts are united into a single bone
    Xform bone = new Xform();
    bone.getChildren().addAll(meshXform0, meshXform1,
            meshXform2,  meshXform3 );
    //Contraction of bone
```

```
    bone.setScale(0.1);

    //Illumination
    PointLight pointLight;
    pointLight = new PointLight(Color.ANTIQUEWHITE);
    pointLight.setTranslateX(0);
    pointLight.setTranslateY(0);
    pointLight.setTranslateZ(0);


    //Booking
    getChildren().addAll(bone, pointLight  );

  }
}//End of class Graphic




//The mesh of a quadrant generated
//by two complementary polylines 1 and 2.
//The mesh must be rotated angle= phase
//and the orientation shall be adjusted.
static Xform MeshQuadrant(float[] xPolyline1,
        float[] yPolyline1, float[] xPolyline2,
        float[] yPolyline2, double phase,
        Color color,
        int orientation)
{


    int n = xPolyline1.length;
    //Scales
    float zScale =   20f;
    float xyScale =  20f;

    //Theta angle opened by a quadrant
    double PIHalf =  Math.PI/2;
    //Numbers of subdivisions along theta angle
    int m = 18 ;
```

```
//Angle increment
double deltaTheta =  (PIHalf/ m) ;

//Array with coordinates of points of
//the mesh, each one with 3 coordinates
float points[] = new float[3 * (n+ 1)*(m +1)];

//There are nm tiny rectangles.
//Each rectangle has 2 triangles
int nFaces = 2*n*m;
//Each point of each face has 6 coordinates,
//3 for the vertex and
//3 for the marked points of the wallpaper (texture)
int faces[] = new int[nFaces * 6];


// Create points
//z-variable


float deltaZ = 0;
//First quadrant
//Connection from polyline1 to polyline2
for (int j = 0; j < n ; j++)
{

    deltaZ = (yPolyline2[j] - yPolyline1[j])/m;

    float ra =  (xPolyline1[j] - xPolyline1[0]);

    float rb =  (xPolyline2[j] - xPolyline2[0]);

    //Angle theta
    for (int i = 0; i <= m; i++)
    {
        float zi = yPolyline1[j] + deltaZ*i -yPolyline1[0];

        double thetai =  -i*deltaTheta;
        if (i == m) thetai = -PIHalf;
        int index = j * (m+1)* 3 + (i * 3);
```

```
            points[index] =   ra
                * ( (float) Math.cos(thetai)) * xyScale;
            points[index + 1]  =  rb
                * ( (float) Math.sin(thetai)) * xyScale;
            points[index + 2] = zi*zScale;

        }
    }

  if (orientation == 1)
  {
    //Faces are built.
    //Two triangles for each rectangle.
    for (int j = 0; j <  n-1; j++)
    {
        for (int i = 0; i < m; i++)
        {
            int p00 = j * (m+1) + i;
            int p01 = p00 + 1;
            int p10 = p00 + (m+1);
            int p11 = p10 + 1;
            int tc00 = 0;
            int tc01 = 0;
            int tc10 = 0;
            int tc11 = 0;
            //First triangle
            int index = (j * m * 6 + (i * 6)) * 2;
            faces[index + 0] = p00;
            faces[index + 1] = tc00;
            faces[index + 2] = p10;
            faces[index + 3] = tc10;
            faces[index + 4] = p11;
            faces[index + 5] = tc11;
            //second triangle
            index = index + 6;
            faces[index + 0] = p11;
            faces[index + 1] = tc11;
            faces[index + 2] = p01;
            faces[index + 3] = tc01;
            faces[index + 4] = p00;
```

```
            faces[index + 5] = tc00;
        }
    }
}


 if (orientation == -1)
{
    //Faces are built.
    //Two triangles for each rectangle.
    for (int j = 0; j <  n-1; j++)
    {
        for (int i = 0; i < m; i++)
        {
            int p00 = j * (m+1) + i;
            int p01 = p00 + 1;
            int p10 = p00 + (m+1);
            int p11 = p10 + 1;
            int tc00 = 0;
            int tc01 = 0;
            int tc10 = 0;
            int tc11 = 0;
            //First triangle
            int index = (j * m * 6 + (i * 6)) * 2;
            faces[index + 0] = p00;
            faces[index + 1] = tc00;
            faces[index + 2] = p11;
            faces[index + 3] = tc11;
            faces[index + 4] = p10;
            faces[index + 5] = tc10;
            //second triangle
            index = index + 6;
            faces[index + 0] = p11;
            faces[index + 1] = tc11;
            faces[index + 2] = p00;
            faces[index + 3] = tc00;
            faces[index + 4] = p01;
            faces[index + 5] = tc01;
        }
    }
```

```
  }


    TriangleMesh mesh = new TriangleMesh();
    mesh.getPoints().addAll(points);

    mesh.getTexCoords().addAll(0,0);
    mesh.getFaces().addAll(faces);


    PhongMaterial materialF  = new PhongMaterial();
    materialF.setDiffuseColor(color);
    MeshView  meshView = new MeshView(mesh);
    meshView.setMaterial(materialF);
    meshView.setDrawMode(DrawMode.LINE);
    meshView.setDrawMode(DrawMode.FILL);
    meshView.setCullFace(CullFace.NONE);

    final Xform meshViewXform = new Xform();
    meshViewXform.getChildren().add(meshView);
    meshViewXform.rz.angleProperty().set(phase);

    return meshViewXform;
}



public static void main(String[] args)
{

    launch(args);
}

}//End of main class M139 MlgrmLeftHand9


//Outer class Xform.
//Machinery that enables rotations in vivo of the scene
```

```
//and the conformation of groups of control.
class Xform extends Group {

    public enum RotateOrder {
        XYZ, XZY, YXZ, YZX, ZXY, ZYX
    }

    public Translate t  = new Translate();
    public Translate p  = new Translate();
    public Translate ip = new Translate();
    public Rotate rx = new Rotate();
    { rx.setAxis(Rotate.X_AXIS); }
    public Rotate ry = new Rotate();
    { ry.setAxis(Rotate.Y_AXIS); }
    public Rotate rz = new Rotate();
    { rz.setAxis(Rotate.Z_AXIS); }
    public Scale s = new Scale();

    public Xform() {
        super();
        getTransforms().addAll(t, p, ip, rx, ry, rz, s);
    }

    public Xform(RotateOrder rotateOrder) {
        super();
        // choose the order of rotations
        //based on the rotateOrder
        switch (rotateOrder) {
        case XYZ:
            getTransforms().addAll(t, p, rz, ry, rx, s, ip);
            break;
        case XZY:
            getTransforms().addAll(t, p, ry, rz, rx, s, ip);
            break;
        case YXZ:
            getTransforms().addAll(t, p, rz, rx, ry, s, ip);
            break;
        case YZX:
            getTransforms().addAll(t, p, rx, rz, ry, s, ip);
            break;
```

```
        case ZXY:
            getTransforms().addAll(t, p, ry, rx, rz, s, ip);
            break;
        case ZYX:
            getTransforms().addAll(t, p, rx, ry, rz, s, ip);
            break;
        }
    }

    public void setTranslate(double x, double y, double z) {
        t.setX(x);
        t.setY(y);
        t.setZ(z);
    }

    public void setTranslate(double x, double y) {
        t.setX(x);
        t.setY(y);
    }

    // Cannot override these methods as they are final:
    // public void setTranslateX(double x) { t.setX(x); }
    // public void setTranslateY(double y) { t.setY(y); }
    // public void setTranslateZ(double z) { t.setZ(z); }
    // Use these methods instead:
    public void setTx(double x) { t.setX(x); }
    public void setTy(double y) { t.setY(y); }
    public void setTz(double z) { t.setZ(z); }

    public void setRotate(double x, double y, double z) {
        rx.setAngle(x);
        ry.setAngle(y);
        rz.setAngle(z);
    }

    public void setRotateX(double x) { rx.setAngle(x); }
    public void setRotateY(double y) { ry.setAngle(y); }
    public void setRotateZ(double z) { rz.setAngle(z); }
    public void setRx(double x) { rx.setAngle(x); }
    public void setRy(double y) { ry.setAngle(y); }
```

```
    public void setRz(double z) { rz.setAngle(z); }

    public void setScale(double scaleFactor) {
        s.setX(scaleFactor);
        s.setY(scaleFactor);
        s.setZ(scaleFactor);
    }

    public void setScale(double x, double y, double z) {
        s.setX(x);
        s.setY(y);
        s.setZ(z);
    }

    // Cannot override these methods as they are final:
    // public void setScaleX(double x) { s.setX(x); }
    // public void setScaleY(double y) { s.setY(y); }
    // public void setScaleZ(double z) { s.setZ(z); }
    // Use these methods instead:
    public void setSx(double x) { s.setX(x); }
    public void setSy(double y) { s.setY(y); }
    public void setSz(double z) { s.setZ(z); }

    public void setPivot(double x, double y, double z) {
        p.setX(x);
        p.setY(y);
        p.setZ(z);
        ip.setX(-x);
        ip.setY(-y);
        ip.setZ(-z);
    }

    public void reset() {
        t.setX(0.0);
        t.setY(0.0);
        t.setZ(0.0);
        rx.setAngle(0.0);
        ry.setAngle(0.0);
        rz.setAngle(0.0);
        s.setX(1.0);
```

```
        s.setY(1.0);
        s.setZ(1.0);
        p.setX(0.0);
        p.setY(0.0);
        p.setZ(0.0);
        ip.setX(0.0);
        ip.setY(0.0);
        ip.setZ(0.0);
    }

    public void resetTSP() {
        t.setX(0.0);
        t.setY(0.0);
        t.setZ(0.0);
        s.setX(1.0);
        s.setY(1.0);
        s.setZ(1.0);
        p.setX(0.0);
        p.setY(0.0);
        p.setZ(0.0);
        ip.setX(0.0);
        ip.setY(0.0);
        ip.setZ(0.0);
    }
}//End of class Xform; End of program M139 MlgrmLeftHand9
```

140 ***Exercise.*** *Run the previous program and play with the code.*

141 ***Challenge.*** *Compose and scene that compares the phalanx developed in the previous program with that found in homo.*

7.7 Conclusion

The human hand is a marvel that is more clever than complex. Nevertheless, anyone that has ever seen the hand of a chimp or at least its photography shall propose that these two hands are related by evolution. So, the null hypothesis of science reads: the human and chimp hands came by Natural and Gradual Evolution from the hand of a common ancestor.

Now, the human, the chimp and the bonobo are very violent and also by their very nature they are killing machines. In fact, chimps are blamed by having caused

severe damage to many persons. To that purpose they used their teeth and nails. By the same token, bonobos conform small groups to chase chimps. On his side, the man loves to use his whole body, intelligence and weapons to harm others. So, the proposed evolutionary hypothesis must be complemented by assuming that violence was an important molding factor of the evolutionary branch leading to modern chimp, bonobo and man.

We must calculate all the predictions that spring from this null hypothesis to compare them with nature. Or, we can just formulate one prediction that shows that this hypothesis is obviously false, as it indeed is.

To formulate a mandatory, falsifiable and false prediction of The Evolutionary Theory we follow the following pattern: *Did you say that you appeared here after running 14km in an hour? You are here, OK but, where is your perspiration?* So, our falsification is as follows:

The assumption that the chimp, the bonobo and the homo are directly related by a common ancestor leads to the following mandatory prediction: apart from ordinary homo, you and me, many more variants must exist, in particular, one whose phalanges of his hand must be curved on the inner side. The proof follows:

Curvature of phalanges serves the purpose of making room to accommodate thicker tendons and bigger muscles with more space to get contact with opposite parts. Muscles and tendons over not curved phalanges would greatly interfere with the fitting of hand grip to objects. Now, the curvature of fingers of the chimp is adjusted to make of it a champion in arboreal mobility that is nevertheless not as spectacular as that of the orangutan. On the other hand, man can get trained to be good at arm swinging but this activity is not the one he usually prefers because his strong mode of locomotion is bipedal. So, how could curved phalanges help the man? They would allow for more powerful muscles than on its turn would demand more mineralized bones. Specimens like these are predicted to be present as in the fossil record as in modern populations because they would have an advantage in the use of nails, very important for foraging, and also for the management of clubs, very important as for defense as for attack.

Is gradualism used in our argument? Yes, it has been used everywhere: to evolve from a hand that is perfect for hanging to a multi-grip hand (Alan Hedge, 2013) is not that easy and one must suppose that everything can softly change to make evolution reliable. Moreover, suppose that originally the homo branch was packed without curved phalanges. In that case and thanks to gradualism and by the same reasons that maintain it in a population, curvature would be a inescapably acquired by evolution. In the official jargon: acquiring curvature is under gradualism an evolutionary invading, stable strategy. That is why our prediction is mandatory. And it is also false: predicted specimens exist neither in extant population nor in the fossil record.

Answers to exercises

9, page 9. Colors, axes of rotation and direction of movement of sphere, cylinder and box are decided by randomness.

```
/*  Program M9 PredefinedShapes4

    Randomness directs cylinders, spheres and boxes
    that move around.
 */


package predefinedshapes4;

import java.util.Random;
import javafx.animation.KeyFrame;
import javafx.animation.KeyValue;
import javafx.animation.Timeline;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.geometry.Point3D;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.Box;
import javafx.scene.shape.Cylinder;
import javafx.scene.shape.Sphere;
import javafx.util.Duration;
```

```
public class PredefinedShapes4 extends Application
{
    private final Timeline animation  = new Timeline();
    //Generator of pseudo random numbers
    private static final Random rd = new Random();




 //A random color is synthesized
 private static Color randomColor()
 {
    //Color is defined as a random
    //mix of red, green and blue
    double r,g,b;
    r = rd.nextDouble();
    g = rd.nextDouble();
    b = rd.nextDouble();
    //The color is prepared
    Color c = Color.color(r,g,b);
    return c;
 }

     //A random 3D direction is synthesized
 private static Point3D randomDirection()
 {

    Double theta1 = Math.PI * rd.nextDouble();
    Double cosTheta1 = Math.cos(theta1);
    Double theta2 = Math.PI * rd.nextDouble();
    Double cosTheta2 = Math.cos(theta2);
    Double theta3 = Math.PI * rd.nextDouble();
    Double cosTheta3 = Math.cos(theta3);
    return new Point3D(cosTheta1,cosTheta2, cosTheta3);
 }

@Override
```

```
public void start(Stage stage)
{

    stage.setTitle("Predefined shapes in movement");
    Group root = new Group();
    Scene scene = new Scene(root, 1000, 800);
    //Color of Background
    scene.setFill(Color.color(0.2, 0.2, 0.2, 1.0));

    //========Initialization============

    //Sphere
    double radius = 50;
    Sphere sphere = new Sphere(radius);

    PhongMaterial materialSphere
            = new PhongMaterial(randomColor());
    sphere.setMaterial(materialSphere);
    //Position
    sphere.setLayoutX(100);
    sphere.setLayoutY(200);

    //Cylinder(radius, height)
    Cylinder cylinder = new Cylinder(50, 100);
    //Material for skinning shapes
    PhongMaterial materialCyl
            = new PhongMaterial(randomColor());
    //Material is used as skin of cylinder
    cylinder.setMaterial(materialCyl);
    //Position
    cylinder.setLayoutX(300);
    cylinder.setLayoutY(200);

    //Box(xLength,yLength,zLength)
    Box box = new Box(50,70,100);
    PhongMaterial materialBox
            = new PhongMaterial(randomColor());
    box.setMaterial(materialBox);
    box.setLayoutX(500);
    box.setLayoutY(200);
```

```
//Booking
root.getChildren().addAll(cylinder, box, sphere);

//==========Animation============


//Axis of rotation of box
box.setRotationAxis(randomDirection());
//Axis of rotation of box
cylinder.setRotationAxis(randomDirection());

//Duration of animation
double endTime = 4000.0;


animation.getKeyFrames().addAll
(
    ////Sphere
    //X translation: initial position at initial time
    new KeyFrame(Duration.ZERO,
        new KeyValue(sphere.translateXProperty(), 0.0)),
    // X Translation: final position at final time
    new KeyFrame(new Duration(endTime),
        new KeyValue(sphere.translateXProperty(), 500)),
    //Y translation: initial position at initial time
    new KeyFrame(Duration.ZERO,
        new KeyValue(sphere.translateYProperty(), 0.0)),
    // y Translation: final position at final time
    new KeyFrame(new Duration(endTime),
        new KeyValue(sphere.translateYProperty()
                , rd.nextInt(500))),

    ////Cylinder
    //Rotation: initial rotate at initial time
    new KeyFrame(Duration.ZERO,
        new KeyValue(cylinder.rotateProperty(), 0.0)),
    //Rotation: final rotate at final time
    new KeyFrame(new Duration(endTime),
```

```
            new KeyValue(cylinder.rotateProperty(), 360)),

        ////Box
        //Y Translation: initial position at initial time
        new KeyFrame(Duration.ZERO,
            new KeyValue(box.translateYProperty(), 0.0)),
        //Y Translation: final position at final time
        new KeyFrame(new Duration(endTime),
            new KeyValue(box.translateYProperty(), 500)),

        //X Translation: initial position at initial time
        new KeyFrame(Duration.ZERO,
            new KeyValue(box.translateXProperty(), 0.0)),
        //X Translation: final position at final time
        new KeyFrame(new Duration(endTime),
            new KeyValue(box.translateXProperty()
                    , 500-rd.nextInt(1000))),

        //Rotation: initial rotate at initial time
        new KeyFrame(Duration.ZERO,
            new KeyValue(box.rotateProperty(), 0.0)),
        //Rotation: final rotate at final time
        new KeyFrame(new Duration(endTime),
            new KeyValue(box.rotateProperty(), 360))

    );
    animation.play();
    stage.setScene(scene);
    stage.show();
}



public static void main(String[] args)
{
    launch(args);
}

}//End of main class M9 PredefinedShapes4
```

16, page 34. The control of the rotation slider is extended to the wrist.

```
/*
 * Program M16 HandOnlyBoxes3.
 * It is shown how to make an ensemble of boxes
 * into a 3D closable hand.
 * A slider controls rotation of wrist and phalanges.
 */


package handonlyboxes3;

import java.lang.reflect.Field;
import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;
import javafx.geometry.Insets;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.PerspectiveCamera;
import javafx.scene.PointLight;
import javafx.scene.Scene;
import javafx.scene.SubScene;
import javafx.scene.control.Label;
import javafx.scene.control.Slider;
import javafx.scene.layout.HBox;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.Box;
import javafx.scene.transform.Rotate;
import static javafx.scene.transform.Rotate.X_AXIS;
import javafx.stage.Stage;



public class HandOnlyBoxes3 extends Application
```

```
{

    //Slider for zooming
    private Slider zSlider;
    //Position of graphic along the z-axis,
    //normal to the screen.
    //It is a double property because
    //it will get involved into a binding
    private final DoubleProperty zPos
            = new SimpleDoubleProperty(2000);

    //Slider for angle of rotation of fingers
    private Slider angleSlider;
    //Position of graphic along the z-axis,
    //normal to the screen.
    //It is a double property because
    //it will get involved into a binding
    private final DoubleProperty angleValue
            = new SimpleDoubleProperty(0);

    //For listener of mouse
    private double anchorAngleX = 0;
    private double anchorAngleY = 0;
    private double anchorX, anchorY;
    //Angles of rotation
    private final DoubleProperty angleX
            = new SimpleDoubleProperty(0);
    private final DoubleProperty angleY
            = new SimpleDoubleProperty(0);

    PerspectiveCamera camera
            = new PerspectiveCamera(false);

    //The application is instantiated
    static HandOnlyBoxes3 p = new HandOnlyBoxes3();



    //An array with all colors that are named in JavaFX
    private static final Color JavaFXColors[]
```

```
                                = new Color[148];

    //Generator of pseudo random numbers
    private static final Random rd = new Random();


    //Initalization of number of cubes in each finger
    private final int nPhalanges = 3;

public void init(Stage primaryStage)
{
    primaryStage.setTitle("An ensemble of boxes"
            + " act coherently as a thumbless hand");
    //The whole drawing  is created
    //as a JavaFX space-like object
    final Group parent = new Group(create3dContent());

    //Initial position
    parent.setTranslateX(450);
    parent.setTranslateY(400);
    parent.setTranslateZ(0);
    Rotate xRotate;
    Rotate yRotate;
    //Initialization of rotation
    parent.getTransforms().setAll
    (
      xRotate = new Rotate(0, Rotate.X_AXIS),
      yRotate = new Rotate(0, Rotate.Y_AXIS)
    );

    //Graphic is attached to root
    final Group root = new Group(parent);
    //Window of the scene
    final Scene scene = new Scene(root, 900, 900, true);
    //Background
    scene.setFill(Color.SKYBLUE);

    //Listener of mousePressed
    scene.setOnMousePressed
    (
```

```
    event ->
    {
        anchorX = event.getSceneX();
        anchorY = event.getSceneY();
        anchorAngleX = angleX.get();
        anchorAngleY = angleY.get();
    }
);


//Listener of mouseDragged
scene.setOnMouseDragged
(
    event ->
    {
        angleX.set(anchorAngleX
                - (anchorY - event.getSceneY()));
        angleY.set(anchorAngleY
                + anchorX - event.getSceneX());
    }
);

//Rotation of the scene gets bound to
//the angle determined by  mouseDragged
xRotate.angleProperty().bind(angleX);
yRotate.angleProperty().bind(angleY);

//Illumination
PointLight pointLight = new PointLight(Color.WHITE);
pointLight.setTranslateX(100);
pointLight.setTranslateY(200);
pointLight.setTranslateZ(-2000);
root.getChildren().add(pointLight);

//Camera
scene.setCamera(camera);
primaryStage.setScene(scene);
PerspectiveCamera subSceneCamera
        = new PerspectiveCamera(false);
scene.setCamera(subSceneCamera);
```

```
//To zoom down the graphic,
//it is translated away from  the beholder,
//along the Z-direction (normal to the screen).
//Change zPos to change initial setting.
zSlider = new Slider();
zSlider.setMax(5000);
zSlider.setMin(1);


//Bind zPos to the reading of the slider
zSlider.valueProperty().bindBidirectional(zPos);
//Execute negative zoom
parent.translateZProperty().bind(zPos);
System.out.println("zPos = " + zPos);


//This slider   controls the closing of the hand.
angleSlider = new Slider();
angleSlider.setMax(90);
angleSlider.setMin(0);
angleSlider.setValue(10);

//To close the hand, an angle of rotation is enabled.
//Bind angleValue to the reading of the slider
angleSlider.valueProperty().bindBidirectional(angleValue);



//Other JavaFX controls, labels and containers
Label zLabel = new Label("Zoom");
Label angleLabel = new Label("Close hand");
HBox subSceneGroup = new HBox(70,
    new HBox(10, zLabel, zSlider,
            angleLabel, angleSlider));

subSceneGroup.setPadding(new Insets(10, 100, 0, 100));
SubScene subScene
        = new SubScene(subSceneGroup, 880, 200);
subScene.setLayoutX(10);
```

```
    subScene.setLayoutY(800);
    root.getChildren().add(subScene);

    primaryStage.show();
}


public void play()
{

}

@Override public void stop()
{

}

@Override
public void start(Stage primaryStage)  throws Exception
{
    init(primaryStage);
    primaryStage.show();
    play();
}


public static void main(String[] args)
{
    launch(args);
}



private Node create3dContent()
{
    XformWithPivot sceneRoot = new XformWithPivot();
    sceneRoot.rx.setAngle(45);
    sceneRoot.ry.setAngle(110);
    getColors();
    //The constructor of hand is invoked.
```

```
    Hand  h =  new Hand();
    sceneRoot.getChildren().add(h);
    return new Group(sceneRoot);
}

//Colors with a name in JavaFX are put into an array
static private Color[] getColors()
{
    //Colors with a name in JavaFX
    final Field[] fields = Color.class.getFields();


    int N = 0;
    //Colors are put into an array
    for (final Field field : fields)
    {
        if (field.getType() == Color.class)
        {
            try
            {
                final Color color = (Color) field.get(null);
                //Array to keep colors
                JavaFXColors[N] = color;
                /*
                //The name of the color
                final String colorName = field.getName();
                System.out.println("N = " + N + "  "
                        + color.toString() + " " + colorName);
                */
                N++;
            } catch (IllegalArgumentException
                    | IllegalAccessException ex)
            {
                Logger.getLogger(HandOnlyBoxes3.class
                    .getName()).log(Level.SEVERE, null, ex);
            }
       }
    }

    return JavaFXColors;
```

```
}


//A random color is synthesized
 private static Color randomColor()
 {
    //A JavaFX named color is picked
    //but the first, Transparent, is ignored
    int n = rd.nextInt(JavaFXColors.length-1)+1;
    Color c = JavaFXColors[n];
    //Some light colors are not visualized so,
    //they might be darkened:
    //c = c.darker();
    return c;
 }


  //=======================================================
  //=============Beginning of Inner class Hand============
  //=======================================================



//Kit of available families of transformations
private class XformWithPivot extends Group
{
    public Rotate rx = new Rotate();
    { rx.setAxis(Rotate.X_AXIS); }
    public Rotate ry = new Rotate();
    { ry.setAxis(Rotate.Y_AXIS); }
    public Rotate rz = new Rotate();
    { rz.setAxis(Rotate.Z_AXIS); }
    //Rotation = angle + pivot + axis.
    //Every rotation is around X_AXIS
    public Rotate rwpx = new Rotate(0,0,0,0, X_AXIS);

    public XformWithPivot()
    {
        super();
        getTransforms().addAll( rx, ry, rz, rwpx );
```

```
    }
}//End of class XformWithPivot



private class Hand extends Group
{

    //Rotations about the center
    final Rotate rx = new Rotate(0,Rotate.X_AXIS);
    final Rotate ry = new Rotate(0,Rotate.Y_AXIS);
    final Rotate rz = new Rotate(0,Rotate.Z_AXIS);
    //A node with all elements
    XformWithPivot handGroup = new XformWithPivot();

    //Nested definition of   nodes is necesary
    //so that, say,  the palm drags the fingers:

    //Group with phalanges 0 of all fingers + fingers1Group
    XformWithPivot fingers0Group = new XformWithPivot();
    //Group with phalanges 1 of all fingers + fingers2Group
    XformWithPivot fingers1Group = new XformWithPivot();
    ///Group with phalanges 2 of all fingers
    XformWithPivot fingers2Group = new XformWithPivot();
    //Group with all phalanges of all fingers
    XformWithPivot fingersGroup = new XformWithPivot();
    //Group with all phalanges of all fingers
    XformWithPivot palmGroup = new XformWithPivot();
    Hand()
    {

        //Arm is constructed
        Box arm = new Box(100,20,400);
        Color c = randomColor();
        PhongMaterial materialArm = new PhongMaterial(c);
        arm.setMaterial(materialArm);
        arm.setTranslateY(20);
        //Palm is constructed
        Box palm = new Box(125,20,200);
        Color c2 = randomColor();
```

```
PhongMaterial materialPalm = new PhongMaterial(c2);
palm.setMaterial(materialPalm);
palm.setTranslateZ(300);
palm.setTranslateY(20);


//5 fingers on the same line.
//No opposable thumb
Finger Fingers[] = new Finger[5];

for(int i = 0; i < 5; i++)
{
    //Each finger has 3 phalanges
    Fingers[i] = new Finger(3);
    for(int j =0; j < nPhalanges; j++)
    {
        Box ph = Fingers[i].Phalanges[j];
        ph.setTranslateX(12.5 * (5-2*(i))-12.5);
        ph.setTranslateZ(420 + 40*j);
        ph.setTranslateY(20);

        if (j == 0) fingers0Group.getChildren().add(ph);
        if (j == 1) fingers1Group.getChildren().add(ph);
        if (j == 2) fingers2Group.getChildren().add(ph);

    }
}
//Groups are nested
fingers1Group.getChildren().addAll( fingers2Group);
fingers0Group.getChildren().addAll( fingers1Group);
fingersGroup.getChildren().addAll(fingers0Group);
palmGroup.getChildren().addAll(palm, fingersGroup );
handGroup.getChildren().addAll(arm, palmGroup );

//The palm must rotate around
//the end of the arm that is parallel to the X_AXIS
//This is the pivot of rotation
palmGroup.rwpx.pivotZProperty().setValue(200.0);
palmGroup.rwpx.pivotYProperty().setValue(10.0);
//The angle of rotation is controlled by a slider
```

```
        palmGroup.rwpx.angleProperty()
                .bindBidirectional(angleValue);



        //The group of 5 fingers must rotate around
        //the end of the palm that is parallel to the X_AXIS
        //This is the pivot of rotation
        fingersGroup.rwpx.pivotZProperty().setValue(400.0);
        fingersGroup.rwpx.pivotYProperty().setValue(10.0);
        //The angle of rotation is controlled by a slider
        fingersGroup.rwpx.angleProperty()
                .bindBidirectional(angleValue);



        //The group of  fingers past phalanx 0
        //must rotate around
        //the end of the 0-th phalanx.
        //This is the pivot of rotation
        fingers1Group.rwpx.pivotZProperty().setValue(440.0);
        fingers1Group.rwpx.pivotYProperty().setValue(10.0);
        //The angle of rotation is controlled by a slider
        fingers1Group.rwpx.angleProperty()
                .bindBidirectional(angleValue);


        //The group of  fingers past phalanx 1
        //must rotate around
        //the end of the 1st phalanx.
        //This is the pivot of rotation
        fingers2Group.rwpx.pivotZProperty().setValue(480.0);
        fingers2Group.rwpx.pivotYProperty().setValue(10.0);
        //The angle of rotation is controlled by a slider
        fingers2Group.rwpx.angleProperty()
                .bindBidirectional(angleValue);

        getChildren().addAll(handGroup );
    }//End of constructor of class Hand
```

```
}//End of inner class Hand


//A finger is an ensemble of phalanges
private class Finger extends Group
{
    final Rotate rx = new Rotate(0,Rotate.X_AXIS);
    final Rotate ry = new Rotate(0,Rotate.Y_AXIS);
    final Rotate rz = new Rotate(0,Rotate.Z_AXIS);
    //Our finger is a composition of 3 boxes or phalanges
    private   Box[] Phalanges    = new Box [nPhalanges];


    //Constructor
    Finger(int nPhalanges)
    {
        for(int i =0; i < nPhalanges; i++)
        {
             final Box p = new Box(25,20, 40);
             Color c = randomColor();
             PhongMaterial materialFinger
                     = new PhongMaterial(c);
             p.setMaterial(materialFinger);
             Phalanges[i] = p;
        }
    }
}//End of inner class Finger



}//En of main class M16 HandOnlyBoxes3
```

17, page 34. To implement a closable hand whose fingers can also be moved transversally, we have a restriction: a JavaFX group cannot be added to more than a single node. This forces us to finely particularize groups.

```
/*
```

```
 * Program M17 HandOnlyBoxes4.
 * It is shown how to make an ensemble of boxes
 * into a 3D closable hand
 * whose fingers can be moved transversally.
 * The hand lacks an opposable thumb.

 * We copy the anatomical structure:
 * phalanges form fingers that together
 * with the palm conform the hand.
 * It is found that JavaFX is highly compatible
 * with the anatomical organization.
 */

package handonlyboxes4;

import java.lang.reflect.Field;
import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.beans.binding.NumberBinding;
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;
import javafx.geometry.Insets;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.PerspectiveCamera;
import javafx.scene.PointLight;
import javafx.scene.Scene;
import javafx.scene.SubScene;
import javafx.scene.control.Label;
import javafx.scene.control.Slider;
import javafx.scene.layout.HBox;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.Box;
import javafx.scene.transform.Rotate;
import static javafx.scene.transform.Rotate.X_AXIS;
import static javafx.scene.transform.Rotate.Y_AXIS;
```

```
import javafx.stage.Stage;



public class HandOnlyBoxes4 extends Application
{

    //Slider for zooming
    private Slider zSlider;
    //Position of graphic along the z-axis,
    //normal to the screen.
    //It is a double property because
    //it will get involved into a binding
    private final DoubleProperty zPos
            = new SimpleDoubleProperty(2000);

    //Slider for angle of rotation of fingers
    private Slider closeHandSlider;
    //To close the hand,
    //each phalange suffers a rotation.
    //It is a double property because
    //it will get involved into a binding
    private final DoubleProperty closeProperty
            = new SimpleDoubleProperty(0);

    //Slider for opening fingers thanks to
    //their transversal rotation
    private Slider openingFingersSlider;
    //To open the fingers,
    //each phalange suffers a transversal rotation.
    //It is a double property because
    //it will get involved into a binding
    private final DoubleProperty openingFingersProperty
            = new SimpleDoubleProperty(0);
    private final DoubleProperty openingFingersProperty3
            = new SimpleDoubleProperty(0);


    //For listener of mouse
    private double anchorAngleX = 0;
```

```
    private double anchorAngleY = 0;
    private double anchorX, anchorY;
    //Angles of rotation of the scene
    private final DoubleProperty angleX
            = new SimpleDoubleProperty(0);
    private final DoubleProperty angleY
            = new SimpleDoubleProperty(0);

    PerspectiveCamera camera
            = new PerspectiveCamera(false);

    //The application is instantiated
    static HandOnlyBoxes4 p = new HandOnlyBoxes4();



    //An array with all colors that are named in JavaFX
    private static final Color JavaFXColors[]
                                = new Color[148];

    //Generator of pseudo random numbers
    private static final Random rd = new Random();

    //Number of fingers
    private final int nFingers = 5;

    //Number of phalanges of each finger
    private final int nPhalanges = 3;

public void init(Stage primaryStage)
{
    primaryStage.setTitle("An ensemble of boxes"
            + " act coherently as a thumbless hand");
    //The whole drawing  is created
    //as a JavaFX space-like object
    final Group parent = new Group(create3dContent());

    //Initial position
    parent.setTranslateX(450);
    parent.setTranslateY(400);
```

```
parent.setTranslateZ(0);
Rotate xRotate;
Rotate yRotate;
//Initialization of rotation
parent.getTransforms().setAll
(
  xRotate = new Rotate(0, Rotate.X_AXIS),
  yRotate = new Rotate(0, Rotate.Y_AXIS)
);

//Graphic is attached to root
final Group root = new Group(parent);
//Window of the scene
final Scene scene = new Scene(root, 900, 900, true);
//Background
scene.setFill(Color.SKYBLUE);

//Listener of mousePressed
scene.setOnMousePressed
(
    event ->
    {
        anchorX = event.getSceneX();
        anchorY = event.getSceneY();
        anchorAngleX = angleX.get();
        anchorAngleY = angleY.get();
    }
);


//Listener of mouseDragged
scene.setOnMouseDragged
(
    event ->
    {
        angleX.set(anchorAngleX
                - (anchorY - event.getSceneY()));
        angleY.set(anchorAngleY
                + anchorX - event.getSceneX());
    }
```

```
);

//Rotation of the scene gets bound to
//the angle determined by  mouseDragged
xRotate.angleProperty().bind(angleX);
yRotate.angleProperty().bind(angleY);

//Illumination
PointLight pointLight = new PointLight(Color.WHITE);
pointLight.setTranslateX(100);
pointLight.setTranslateY(200);
pointLight.setTranslateZ(-2000);
root.getChildren().add(pointLight);

//Camera
scene.setCamera(camera);
primaryStage.setScene(scene);
PerspectiveCamera subSceneCamera
        = new PerspectiveCamera(false);
scene.setCamera(subSceneCamera);

//To zoom down the graphic,
//it is translated away from  the beholder,
//along the Z-direction (normal to the screen).
//Change zPos to change initial setting.
zSlider = new Slider();
zSlider.setMax(5000);
zSlider.setMin(1);


//Bind zPos to the reading of the slider
zSlider.valueProperty().bindBidirectional(zPos);
//Execute negative zoom
parent.translateZProperty().bind(zPos);
System.out.println("zPos = " + zPos);


//This slider   controls the closing of the hand.
closeHandSlider = new Slider();
closeHandSlider.setMax(90);
```

```
    closeHandSlider.setMin(0);
    closeHandSlider.setValue(10);

    //To close the hand, an angle of rotation is enabled.
    //Bind angleProperty to the reading of the slider
    closeHandSlider.valueProperty()
            .bindBidirectional(closeProperty);


    //This slider controls the openning of fingers.
    openingFingersSlider = new Slider();
    openingFingersSlider.setMax(15);
    openingFingersSlider.setMin(0);
    openingFingersSlider.setValue(10);

    //To open fingers, an angle of rotation is enabled.
    //Bind openingFingersProperty to the reading of the slider
    openingFingersSlider.valueProperty()
            .bindBidirectional(openingFingersProperty);


    //Other JavaFX controls, labels and containers
    Label zLabel = new Label("Zoom");
    Label angleLabel = new Label("Close hand");
    HBox subSceneGroup = new HBox(70,
        new HBox(10, zLabel, zSlider,
                angleLabel, closeHandSlider, openingFingersSlider));

    subSceneGroup.setPadding(new Insets(10, 100, 0, 100));
    SubScene subScene
            = new SubScene(subSceneGroup, 880, 200);
    subScene.setLayoutX(10);
    subScene.setLayoutY(800);
    root.getChildren().add(subScene);

    primaryStage.show();
}


public void play()
```

```
{

}

@Override public void stop()
{

}

@Override
public void start(Stage primaryStage)  throws Exception
{
    init(primaryStage);
    primaryStage.show();
    play();
}


public static void main(String[] args)
{
    launch(args);
}


//Make the drawing
private Node create3dContent()
{
    //The scene can be rotated
    XformWithPivot sceneRoot = new XformWithPivot();
    sceneRoot.rx.setAngle(45);
    sceneRoot.ry.setAngle(110);
    getColors();
    //The constructor of hand is invoked.
    Hand  h =  new Hand();
    sceneRoot.getChildren().add(h);
    return new Group(sceneRoot);
}

//Colors with a name in JavaFX are put into an array
static private Color[] getColors()
```

```
{
    //Colors with a name in JavaFX
    final Field[] fields = Color.class.getFields();


    int N = 0;
    //Colors are put into an array
    for (final Field field : fields)
    {
        if (field.getType() == Color.class)
        {
            try
            {
                final Color color = (Color) field.get(null);
                //Array to keep colors
                JavaFXColors[N] = color;
                /*
                //The name of the color
                final String colorName = field.getName();
                System.out.println("N = " + N + "   "
                        + color.toString() + " " + colorName);
                */
                N++;
            } catch (IllegalArgumentException
                    | IllegalAccessException ex)
            {
                Logger.getLogger(HandOnlyBoxes4.class
                    .getName()).log(Level.SEVERE, null, ex);
            }
       }
    }

    return JavaFXColors;
}


//A random color is synthesized
 private static Color randomColor()
 {
    //A JavaFX named color is picked
```

```
    //but the first, Transparent, is ignored
    int n = rd.nextInt(JavaFXColors.length-1)+1;
    Color c = JavaFXColors[n];
    //Some light colors are not visualized so,
    //they might be darkened:
    //c = c.darker();
    return c;
 }


  //=====================================================
  //=============Inner class XformWithPivot============
  //=====================================================



//Kit of available families of transformations
private class XformWithPivot extends Group
{
    //Declaration and initialization
    public Rotate rx = new Rotate();
    { rx.setAxis(Rotate.X_AXIS); }
    public Rotate ry = new Rotate();
    { ry.setAxis(Rotate.Y_AXIS); }
    public Rotate rz = new Rotate();
    { rz.setAxis(Rotate.Z_AXIS); }
    //Rotation = angle (one coordinate)
    //+ pivot (3 coordinates) + axis.
    //This rotation is around X_AXIS
    public Rotate rwpx = new Rotate(0,0,0,0, X_AXIS);
    //This rotation is around Y_AXIS
    public Rotate rwpy = new Rotate(0,0,0,0, Y_AXIS);

    public XformWithPivot()
    {
        super();
        getTransforms().addAll( rx, ry, rz, rwpx, rwpy );
    }
}//End of class XformWithPivot
```

```
  //=======================================
  //=============Inner class Hand===========
  //=======================================


private class Hand extends Group
{


    //Nested definition of   nodes is implemented
    //to have, say,  the palm dragging the fingers,
    //and a phalanx dragging the rest of its the finger.


    //Each finger generates various natural groups:
    //Group with the extreme phalanx of finger0
    //Symbolic initialization
    XformWithPivot finger0Ph2Group = new XformWithPivot();
    //Group with the two extreme phalanges
    XformWithPivot finger0Ph21Group = new XformWithPivot();
    //Group with all three phalanges. It is the finger
    XformWithPivot finger0Ph210Group = new XformWithPivot();
    //Finger1
    XformWithPivot finger1Ph2Group = new XformWithPivot();
    XformWithPivot finger1Ph21Group = new XformWithPivot();
    XformWithPivot finger1Ph210Group = new XformWithPivot();
    //Finger2
    XformWithPivot finger2Ph2Group = new XformWithPivot();
    XformWithPivot finger2Ph21Group = new XformWithPivot();
    XformWithPivot finger2Ph210Group = new XformWithPivot();
    //Finger3
    XformWithPivot finger3Ph2Group = new XformWithPivot();
    XformWithPivot finger3Ph21Group = new XformWithPivot();
    XformWithPivot finger3Ph210Group = new XformWithPivot();
    //Finger4
    XformWithPivot finger4Ph2Group = new XformWithPivot();
```

```
XformWithPivot finger4Ph21Group = new XformWithPivot();
XformWithPivot finger4Ph210Group = new XformWithPivot();



//Group with all  fingers
//XformWithPivot allFingersgroup = new XformWithPivot();
//Group with palm and fingers
XformWithPivot palmGroup = new XformWithPivot();
//Node with all elements
XformWithPivot handGroup = new XformWithPivot();

Hand()
{

    //Arm is constructed
    Box arm = new Box(100,20,400);
    Color c = randomColor();
    PhongMaterial materialArm = new PhongMaterial(c);
    arm.setMaterial(materialArm);
    arm.setTranslateY(20);
    //Palm is constructed
    Box palm = new Box(125,20,140);
    Color c2 = randomColor();
    PhongMaterial materialPalm = new PhongMaterial(c2);
    palm.setMaterial(materialPalm);
    palm.setTranslateZ(270);
    palm.setTranslateY(20);



    //5 fingers on the same line.
    //No opposable thumb

    Finger Fingers[] = new Finger[nFingers];

    //Groups are constructed
    for(int i = 0; i < nFingers; i++)
    {
        //Each finger has nPhalanges phalanges
```

```
Fingers[i] = new Finger(nPhalanges);
for(int j =0; j < nPhalanges; j++)
{
    Box ph = Fingers[i].Phalanges[j].box;
    ph.setTranslateX(12.5 * (5-2*(i))-12.5);
    ph.setTranslateZ(360 + 40*j);
    ph.setTranslateY(20);

    //Natural groups are instantiated
    //Finger 0 and its subgroups
    if (i == 0)
    {
        if (j == 2) finger0Ph2Group.getChildren().add(ph);
        if (j == 1) finger0Ph21Group
                .getChildren().addAll(finger0Ph2Group, ph);
        if (j == 0) finger0Ph210Group
                .getChildren().addAll(finger0Ph21Group, ph);
    }

    //Finger 1  and its subgroups
    if (i == 1)
    {
        if (j == 2) finger1Ph2Group.getChildren().add(ph);
        if (j == 1) finger1Ph21Group
                .getChildren().addAll(finger1Ph2Group, ph);
        if (j == 0) finger1Ph210Group
                .getChildren().addAll(finger1Ph21Group, ph);
    }


    //Finger 2  and its subgroups
    if (i == 2)
    {
        if (j == 2) finger2Ph2Group.getChildren().add(ph);
        if (j == 1) finger2Ph21Group
                .getChildren().addAll(finger2Ph2Group, ph);
        if (j == 0) finger2Ph210Group
                .getChildren().addAll(finger2Ph21Group, ph);
    }
```

```
            //Finger 3 and its subgroups
            if (i == 3)
            {
                if (j == 2) finger3Ph2Group.getChildren().add(ph);
                if (j == 1) finger3Ph21Group
                        .getChildren().addAll(finger3Ph2Group, ph)
                if (j == 0) finger3Ph210Group
                        .getChildren().addAll(finger3Ph21Group, ph)
            }

            //Finger 4 and its subgroups
            if (i == 4)
            {
                if (j == 2) finger4Ph2Group.getChildren().add(ph);
                if (j == 1) finger4Ph21Group
                        .getChildren().addAll(finger4Ph2Group, ph)
                if (j == 0) finger4Ph210Group
                        .getChildren().addAll(finger4Ph21Group, ph)
            }

        }
    }

    //Fingers are attached to the palm
    palmGroup.getChildren().addAll(palm, finger0Ph210Group,
            finger1Ph210Group,
            finger2Ph210Group,
            finger3Ph210Group,
            finger4Ph210Group );
    //Palm and arm are attached to hand
    handGroup.getChildren().addAll(arm, palmGroup );



    //========OPENING/CL0SING OF THE HAND============

    NumberBinding closePropertyHalf
            = closeProperty.multiply(0.5);
```

```
//The slider that closes the hand extends its
//control to the palm:
//it must rotate around
//the end of the arm that is parallel to the X_AXIS
//This is the pivot of rotation
palmGroup.rwpx.pivotZProperty().setValue(200.0);
palmGroup.rwpx.pivotYProperty().setValue(10.0);
//The angle of rotation is controlled by a slider
palmGroup.rwpx.angleProperty()
        .bind(closePropertyHalf);



//Specifications of movement for subgroups of finger 0
finger0Ph2Group.rwpx.pivotZProperty().setValue(420.0);
finger0Ph2Group.rwpx.pivotYProperty().setValue(10.0);
finger0Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger0Ph21Group.rwpx.pivotZProperty().setValue(380.0);
finger0Ph21Group.rwpx.pivotYProperty().setValue(10.0);
finger0Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger0Ph210Group.rwpx.pivotZProperty().setValue(340.0);
finger0Ph210Group.rwpx.pivotYProperty().setValue(10.0);
finger0Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

//Specifications of movement for subgroups of finger 1
finger1Ph2Group.rwpx.pivotZProperty().setValue(420.0);
finger1Ph2Group.rwpx.pivotYProperty().setValue(10.0);
finger1Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger1Ph21Group.rwpx.pivotZProperty().setValue(380.0);
finger1Ph21Group.rwpx.pivotYProperty().setValue(10.0);
finger1Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);
```

```
finger1Ph210Group.rwpx.pivotZProperty().setValue(340.0);
finger1Ph210Group.rwpx.pivotYProperty().setValue(10.0);
finger1Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

//Specifications of movement for subgroups of finger 2
finger2Ph2Group.rwpx.pivotZProperty().setValue(420.0);
finger2Ph2Group.rwpx.pivotYProperty().setValue(10.0);
finger2Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger2Ph21Group.rwpx.pivotZProperty().setValue(380.0);
finger2Ph21Group.rwpx.pivotYProperty().setValue(10.0);
finger2Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger2Ph210Group.rwpx.pivotZProperty().setValue(340.0);
finger2Ph210Group.rwpx.pivotYProperty().setValue(10.0);
finger2Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

//Specifications of movement for subgroups of finger 3
finger3Ph2Group.rwpx.pivotZProperty().setValue(420.0);
finger3Ph2Group.rwpx.pivotYProperty().setValue(10.0);
finger3Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger3Ph21Group.rwpx.pivotZProperty().setValue(380.0);
finger3Ph21Group.rwpx.pivotYProperty().setValue(10.0);
finger3Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger3Ph210Group.rwpx.pivotZProperty().setValue(340.0);
finger3Ph210Group.rwpx.pivotYProperty().setValue(10.0);
finger3Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

//Specifications of movement for subgroups of finger 4
finger4Ph2Group.rwpx.pivotZProperty().setValue(420.0);
finger4Ph2Group.rwpx.pivotYProperty().setValue(10.0);
```

```
finger4Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger4Ph21Group.rwpx.pivotZProperty().setValue(380.0);
finger4Ph21Group.rwpx.pivotYProperty().setValue(10.0);
finger4Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger4Ph210Group.rwpx.pivotZProperty().setValue(340.0);
finger4Ph210Group.rwpx.pivotYProperty().setValue(10.0);
finger4Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

//======OPENING/CLOSING OF FINGERS========

//Fingers that are at the border must
//open twice as those
//near the center.
NumberBinding openingFingersPropertyTwo
        = openingFingersProperty.multiply(2);
//Specifications of movement for subgroups of finger 0
finger0Ph210Group.rwpy.pivotXProperty().setValue(0.0);
finger0Ph210Group.rwpy.pivotYProperty().setValue(10.0);
finger0Ph210Group.rwpy.pivotZProperty().setValue(340.0);
finger0Ph210Group.rwpy.angleProperty()
        .bind(openingFingersPropertyTwo);


//Specifications of movement for subgroups of finger 1
finger1Ph210Group.rwpy.pivotXProperty().setValue(0.0);
finger1Ph210Group.rwpy.pivotYProperty().setValue(30.0);
finger1Ph210Group.rwpy.pivotZProperty().setValue(340.0);

finger1Ph210Group.rwpy.angleProperty()
        .bind(openingFingersProperty);

//Finger two remains invariant under opening of fingers

//Opening in the contrary direction of fingers 3 and 4
NumberBinding openingFingersPropertyMinus
```

```
                = openingFingersProperty.multiply(-1);

        //Specifications of movement for finger 3
        finger3Ph210Group.rwpy.pivotXProperty().setValue(0.0);
        finger3Ph210Group.rwpy.pivotYProperty().setValue(30.0);
        finger3Ph210Group.rwpy.pivotZProperty().setValue(340.0);
        finger3Ph210Group.rwpy.angleProperty()
                .bind(openingFingersPropertyMinus);

        NumberBinding openingFingersPropertyMinusTwo
                = openingFingersProperty.multiply(-2);

        //Specifications of movement for finger 4
        finger4Ph210Group.rwpy.pivotXProperty().setValue(0.0);
        finger4Ph210Group.rwpy.pivotYProperty().setValue(10.0);
        finger4Ph210Group.rwpy.pivotZProperty().setValue(340.0);
        finger4Ph210Group.rwpy.angleProperty()
                .bind(openingFingersPropertyMinusTwo);

        getChildren().addAll(handGroup );
    }//End of constructor of class Hand


}//End of inner class Hand



//A finger is an ensemble of phalanges
private class Finger extends Group
{

    Phalanx[] Phalanges    = new Phalanx [nPhalanges];

    //Constructor
    Finger(int nPhalanges)
    {
        for(int i =0; i < nPhalanges; i++)
        {
            Phalanx ph = new Phalanx(25,20,40);
```

```
            Phalanges[i] = ph;
        }
    }
}//End of inner class Finger



//A phalanx is a JavaFX 3D box
private class Phalanx extends Group
{
    Box box;

    //Constructor
    Phalanx(int width, int length, int height)
    {
        box = new Box(width, length, height);
        Color c = randomColor();
        PhongMaterial materialFinger
                = new PhongMaterial(c);
        box.setMaterial(materialFinger);
        //ph.getChildren().add(box);
    }
}//End of inner class phalanx


}//En of main class M17 HandOnlyBoxes4
```

22, page 76. The next code implements a hand with the function of a pair of pliers. We claim that our solution is the cheapest one given the hands of previous programs, i.e., beginning with a given hand, it employs the minimum number of changes to get at the desired hand: some few adaptations together with turning off of some genes and deleting of others (here gene means a line of executable code that ends with semicolon). Our solution classifies as mediocre because pliers get open when the hand is closed. To heal this defect, we need animations.

```
/*
 * Program M22 HandOnlyBoxes7.
 * It is shown how to make an ensemble of boxes
```

```
 * into a 3D closable hand
 * with the function of a pair of pliers.
 * The hand can react to touching with the mouse:
 * touched phalanx changes its color to red.

 * We copy the anatomical structure:
 * phalanges form fingers that together
 * with the palm conform the hand.
 * It is found that JavaFX is highly compatible
 * with the anatomical organization.
 */

package handonlyboxes7;

import java.lang.reflect.Field;
import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.beans.binding.NumberBinding;
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;
import javafx.geometry.Insets;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.PerspectiveCamera;
import javafx.scene.PointLight;
import javafx.scene.Scene;
import javafx.scene.SubScene;
import javafx.scene.control.Label;
import javafx.scene.control.Slider;
import javafx.scene.input.PickResult;
import javafx.scene.layout.HBox;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.Box;
import javafx.scene.transform.Rotate;
import static javafx.scene.transform.Rotate.X_AXIS;
import static javafx.scene.transform.Rotate.Y_AXIS;
```

```
import static javafx.scene.transform.Rotate.Z_AXIS;
import javafx.stage.Stage;



public class HandOnlyBoxes7 extends Application
{
    //Node with all elements
    XformWithPivot handGroup = new XformWithPivot();
    XformWithPivot finger4Ph2Group = new XformWithPivot();
    //Slider for zooming
    private Slider zSlider;
    //Position of graphic along the z-axis,
    //normal to the screen.
    //It is a double property because
    //it will get involved into a binding
    private final DoubleProperty zPos
            = new SimpleDoubleProperty(2000);

    //Slider for angle of rotation of fingers
    private Slider closeHandSlider;
    //To close the hand,
    //each phalange suffers a rotation.
    //It is a double property because
    //it will get involved into a binding
    private final DoubleProperty closeProperty
            = new SimpleDoubleProperty(0);

    //Slider for opening fingers thanks to
    //their transversal rotation
    private Slider openingFingersSlider;
    //To open the fingers,
    //each phalange suffers a transversal rotation.
    //It is a double property because
    //it will get involved into a binding
    private final DoubleProperty openingFingersProperty
            = new SimpleDoubleProperty(0);
    private final DoubleProperty openingFingersProperty3
            = new SimpleDoubleProperty(0);
```

```
    //For listener of mouse
    private double anchorAngleX = 0;
    private double anchorAngleY = 0;
    private double anchorX, anchorY;
    //Angles of rotation of the scene
    private final DoubleProperty angleX
            = new SimpleDoubleProperty(0);
    private final DoubleProperty angleY
            = new SimpleDoubleProperty(0);

    PerspectiveCamera camera
            = new PerspectiveCamera(false);

    //The application is instantiated
    static HandOnlyBoxes7 p = new HandOnlyBoxes7();



    //An array with all colors that are named in JavaFX
    private static final Color JavaFXColors[]
                                = new Color[148];

    //Generator of pseudo random numbers
    private static final Random rd = new Random();

    //Number of fingers
    private final int nFingers = 5;

    //Number of phalanges of each finger
    private final int nPhalanges = 3;

public void init(Stage primaryStage)
{
    primaryStage.setTitle("An ensemble of boxes"
            + " act coherently as a thumbless hand");
    //The whole drawing  is created
    //as a JavaFX space-like object
    final Group parent = new Group(create3dContent());
```

```
//Initial position
parent.setTranslateX(450);
parent.setTranslateY(400);
parent.setTranslateZ(0);
Rotate xRotate;
Rotate yRotate;
//Initialization of rotation
parent.getTransforms().setAll
(
  xRotate = new Rotate(0, Rotate.X_AXIS),
  yRotate = new Rotate(0, Rotate.Y_AXIS)
);

//Graphic is attached to root
final Group root = new Group(parent);
//Window of the scene
final Scene scene = new Scene(root, 900, 900, true);
//Background
scene.setFill(Color.SKYBLUE);

//Listener of mousePressed
scene.setOnMousePressed
(
    event ->
    {
        anchorX = event.getSceneX();
        anchorY = event.getSceneY();
        anchorAngleX = angleX.get();
        anchorAngleY = angleY.get();
    }
);


//Listener of mouseDragged
scene.setOnMouseDragged
(
    event ->
    {
        angleX.set(anchorAngleX
                - (anchorY - event.getSceneY()));
```

```
            angleY.set(anchorAngleY
                    + anchorX - event.getSceneX());
        }
    );

    //Rotation of the scene gets bound to
    //the angle determined by  mouseDragged
    xRotate.angleProperty().bind(angleX);
    yRotate.angleProperty().bind(angleY);

    //Illumination
    PointLight pointLight = new PointLight(Color.WHITE);
    pointLight.setTranslateX(100);
    pointLight.setTranslateY(200);
    pointLight.setTranslateZ(-2000);
    root.getChildren().add(pointLight);

    //Camera
    scene.setCamera(camera);
    primaryStage.setScene(scene);
    PerspectiveCamera subSceneCamera
            = new PerspectiveCamera(false);
    scene.setCamera(subSceneCamera);

    //To zoom down the graphic,
    //it is translated away from  the beholder,
    //along the Z-direction (normal to the screen).
    //Change zPos to change initial setting.
    zSlider = new Slider();
    zSlider.setMax(5000);
    zSlider.setMin(1);


    //Bind zPos to the reading of the slider
    zSlider.valueProperty().bindBidirectional(zPos);
    //Execute negative zoom
    parent.translateZProperty().bind(zPos);
```

```
//This slider   controls the closing of the hand.
closeHandSlider = new Slider();
closeHandSlider.setMax(90);
closeHandSlider.setMin(0);
closeHandSlider.setValue(10);

//To close the hand, an angle of rotation is enabled.
//Bind angleProperty to the reading of the slider
closeHandSlider.valueProperty()
        .bindBidirectional(closeProperty);


//This slider controls the openning of fingers.
openingFingersSlider = new Slider();
openingFingersSlider.setMax(15);
openingFingersSlider.setMin(0);
openingFingersSlider.setValue(10);

//To open fingers, an angle of rotation is enabled.
//Bind openingFingersProperty to the reading of the slider
openingFingersSlider.valueProperty()
        .bindBidirectional(openingFingersProperty);


//Other JavaFX controls, labels and containers
Label zLabel = new Label("Zoom");
Label angleLabel = new Label("Close hand");
HBox subSceneGroup = new HBox(70,
    new HBox(10, zLabel, zSlider,
             angleLabel, closeHandSlider, openingFingersSlider));

subSceneGroup.setPadding(new Insets(10, 100, 0, 100));
SubScene subScene
        = new SubScene(subSceneGroup, 880, 200);
subScene.setLayoutX(10);
subScene.setLayoutY(800);
root.getChildren().add(subScene);

primaryStage.show();
```

```
    //The sense of touch is here
    parent.setOnMouseClicked((event)->
    {
        //Information that can be retrieved
        PickResult res = event.getPickResult();
        //Which node
        Node node = res.getIntersectedNode();
        //Id of node (null if not declared)
        String nodeId  = event.getPickResult()
                .getIntersectedNode().getId();
        //which face of the node (negative if is not enabled
        int face = event.getPickResult().getIntersectedFace();
        System.out.println("res = " + res);
        System.out.println("node = " + node);
        System.out.println("nodeID = " + nodeId);
        System.out.println("face" + face);
        //Change color to red, rotate around Z_AXIS
        if (node instanceof Box)
            {
                ((Box) node).setMaterial(
                        new PhongMaterial( Color.RED));
                handGroup.rotationAxisProperty().setValue(Z_AXIS);
                int val = rd.nextInt(91);
                handGroup.rotateProperty().setValue(val);
                finger4Ph2Group.rotateProperty().setValue(val);
            }




    });

}


public void play()
{

}
```

```
@Override public void stop()
{

}

@Override
public void start(Stage primaryStage)  throws Exception
{
    init(primaryStage);
    primaryStage.show();
    play();
}


public static void main(String[] args)
{
    launch(args);
}


//Make the drawing
private Node create3dContent()
{
    //The scene can be rotated
    XformWithPivot sceneRoot = new XformWithPivot();
    sceneRoot.rx.setAngle(45);
    sceneRoot.ry.setAngle(110);
    getColors();
    //The constructor of hand is invoked.
    Hand  h =  new Hand();
    sceneRoot.getChildren().add(h);
    return new Group(sceneRoot);
}

//Colors with a name in JavaFX are put into an array
static private Color[] getColors()
{
    //Colors with a name in JavaFX
    final Field[] fields = Color.class.getFields();
```

```
    int N = 0;
    //Colors are put into an array
    for (final Field field : fields)
    {
        if (field.getType() == Color.class)
        {
            try
            {
                final Color color = (Color) field.get(null);
                //Array to keep colors
                JavaFXColors[N] = color;
                /*
                //The name of the color
                final String colorName = field.getName();
                System.out.println("N = " + N + "  "
                        + color.toString() + " " + colorName);
                */
                N++;
            } catch (IllegalArgumentException
                    | IllegalAccessException ex)
            {
                Logger.getLogger(HandOnlyBoxes7.class
                    .getName()).log(Level.SEVERE, null, ex);
            }
       }
    }
    return JavaFXColors;
}


//A random color is synthesized
 private static Color randomColor()
 {
    //A JavaFX named color is picked
    //but the first, Transparent, is ignored
    int n = rd.nextInt(JavaFXColors.length-1)+1;
    Color c = JavaFXColors[n];
    //Some light colors are not visualized so,
    //they might be darkened:
```

```
    //c = c.darker();
    return c;
 }


  //================================================
  //=============Inner class XformWithPivot===========
  //================================================



//Kit of available families of transformations
private class XformWithPivot extends Group
{
    //Declaration and initialization
    public Rotate rx = new Rotate();
    { rx.setAxis(Rotate.X_AXIS); }
    public Rotate ry = new Rotate();
    { ry.setAxis(Rotate.Y_AXIS); }
    public Rotate rz = new Rotate();
    { rz.setAxis(Rotate.Z_AXIS); }
    //Rotation = angle (one coordinate)
    //+ pivot (3 coordinates) + axis.
    //This rotation is around X_AXIS
    public Rotate rwpx = new Rotate(0,0,0,0, X_AXIS);
    //This rotation is around Y_AXIS
    public Rotate rwpy = new Rotate(0,0,0,0, Y_AXIS);

    public XformWithPivot()
    {
        super();
        getTransforms().addAll( rx, ry, rz, rwpx, rwpy );
    }
}//End of class XformWithPivot



  //=========================================
  //=============Inner class Hand===========
  //=========================================
```

```
private class Hand extends Group
{


    //Nested definition of   nodes is implemented
    //to have, say,  the palm dragging the fingers,
    //and a phalanx dragging the rest of its the finger.


    //Each finger generates various natural groups:
    //Group with the extreme phalanx of finger0
    //Symbolic initialization
    XformWithPivot finger0Ph2Group = new XformWithPivot();
    //Group with the two extreme phalanges
    XformWithPivot finger0Ph21Group = new XformWithPivot();
    //Group with all three phalanges. It is the finger
    XformWithPivot finger0Ph210Group = new XformWithPivot();
    //Finger1
    XformWithPivot finger1Ph2Group = new XformWithPivot();
    XformWithPivot finger1Ph21Group = new XformWithPivot();
    XformWithPivot finger1Ph210Group = new XformWithPivot();
    //Finger2
    XformWithPivot finger2Ph2Group = new XformWithPivot();
    XformWithPivot finger2Ph21Group = new XformWithPivot();
    XformWithPivot finger2Ph210Group = new XformWithPivot();
    //Finger3
    XformWithPivot finger3Ph2Group = new XformWithPivot();
    XformWithPivot finger3Ph21Group = new XformWithPivot();
    XformWithPivot finger3Ph210Group = new XformWithPivot();
    //Finger4

    XformWithPivot finger4Ph21Group = new XformWithPivot();
    XformWithPivot finger4Ph210Group = new XformWithPivot();
```

```
    //Group with all  fingers
    //XformWithPivot allFingersgroup = new XformWithPivot();
    //Group with palm and fingers
    XformWithPivot palmGroup = new XformWithPivot();


Hand()
{

    //Arm is constructed
    Box arm = new Box(100,20,400);
    Color c = randomColor();
    PhongMaterial materialArm = new PhongMaterial(c);
    arm.setMaterial(materialArm);
    arm.setTranslateY(20);
    //Palm is constructed
    Box palm = new Box(125,20,140);
    Color c2 = randomColor();
    PhongMaterial materialPalm = new PhongMaterial(c2);
    palm.setMaterial(materialPalm);
    palm.setTranslateZ(270);
    palm.setTranslateY(20);



    //5 fingers on the same line.
    //No opposable thumb

    Finger Fingers[] = new Finger[nFingers];

    //Groups are constructed
    for(int i = 0; i < nFingers; i++)
    {
        //Each finger has nPhalanges phalanges
        Fingers[i] = new Finger(nPhalanges);
        for(int j =0; j < nPhalanges; j++)
        {
            Box ph = Fingers[i].Phalanges[j].box;

            ph.setTranslateX(12.5 * (5-2*(i))-12.5);
```

```
ph.setTranslateZ(360 + 40*j);
ph.setTranslateY(20);

//Natural groups are instantiated
//Finger 0 and its subgroups
if (i == 0)
{
    if (j == 2) finger0Ph2Group.getChildren().add(ph);

    if (j == 1) finger0Ph21Group
        .getChildren().addAll(finger0Ph2Group, ph);
    if (j == 0) finger0Ph210Group
        .getChildren().addAll(finger0Ph21Group, ph);
}

//Finger 1  and its subgroups
if (i == 1)
{
    if (j == 2) finger1Ph2Group.getChildren().add(ph);
    if (j == 1) finger1Ph21Group
        .getChildren().addAll(finger1Ph2Group, ph);
    if (j == 0) finger1Ph210Group
        .getChildren().addAll(finger1Ph21Group, ph);
}


//Finger 2  and its subgroups
if (i == 2)
{
    if (j == 2) finger2Ph2Group.getChildren().add(ph);
    if (j == 1) finger2Ph21Group
        .getChildren().addAll(finger2Ph2Group, ph);
    if (j == 0) finger2Ph210Group
        .getChildren().addAll(finger2Ph21Group, ph);
}


//Finger 3 and its subgroups
if (i == 3)
{
```

```
            if (j == 2) finger3Ph2Group.getChildren().add(ph);
            if (j == 1) finger3Ph21Group
                .getChildren().addAll(finger3Ph2Group, ph);
            if (j == 0) finger3Ph210Group
                .getChildren().addAll(finger3Ph21Group, ph);
        }

        //Finger 4 and its subgroups
        if (i == 4)
        {
            if (j == 2) finger4Ph2Group.getChildren().add(ph);
            if (j == 1) finger4Ph21Group
                .getChildren().addAll(finger4Ph2Group, ph);
            if (j == 0) finger4Ph210Group
                .getChildren().addAll(finger4Ph21Group, ph);
        }

    }
}

//Fingers are attached to the palm
palmGroup.getChildren().addAll(palm, finger0Ph210Group,
        //finger1Ph210Group,
        //finger2Ph210Group,
        //finger3Ph210Group,
        finger4Ph210Group );
//Palm and arm are attached to hand
handGroup.getChildren().addAll(arm, palmGroup );



//========OPENING/CL0SING OF THE HAND============

NumberBinding closePropertyHalf
        = closeProperty.multiply(0.5);

//The slider that closes the hand extends its
//control to the palm:
//it must rotate around
//the end of the arm that is parallel to the X_AXIS
```

```
//This is the pivot of rotation
palmGroup.rwpx.pivotZProperty().setValue(200.0);
palmGroup.rwpx.pivotYProperty().setValue(10.0);
//The angle of rotation is controlled by a slider
palmGroup.rwpx.angleProperty()
        .bind(closePropertyHalf);



//Specifications of movement for subgroups of finger 0
finger0Ph2Group.rwpx.pivotZProperty().setValue(420.0);
finger0Ph2Group.rwpx.pivotYProperty().setValue(10.0);
finger0Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger0Ph21Group.rwpx.pivotZProperty().setValue(380.0);
finger0Ph21Group.rwpx.pivotYProperty().setValue(10.0);
finger0Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger0Ph210Group.rwpx.pivotZProperty().setValue(340.0);
finger0Ph210Group.rwpx.pivotYProperty().setValue(10.0);
finger0Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

//Specifications of movement for subgroups of finger 1
finger1Ph2Group.rwpx.pivotZProperty().setValue(420.0);
finger1Ph2Group.rwpx.pivotYProperty().setValue(10.0);
finger1Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger1Ph21Group.rwpx.pivotZProperty().setValue(380.0);
finger1Ph21Group.rwpx.pivotYProperty().setValue(10.0);
finger1Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger1Ph210Group.rwpx.pivotZProperty().setValue(340.0);
finger1Ph210Group.rwpx.pivotYProperty().setValue(10.0);
finger1Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);
```

```
//Specifications of movement for subgroups of finger 2
finger2Ph2Group.rwpx.pivotZProperty().setValue(420.0);
finger2Ph2Group.rwpx.pivotYProperty().setValue(10.0);
finger2Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger2Ph21Group.rwpx.pivotZProperty().setValue(380.0);
finger2Ph21Group.rwpx.pivotYProperty().setValue(10.0);
finger2Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger2Ph210Group.rwpx.pivotZProperty().setValue(340.0);
finger2Ph210Group.rwpx.pivotYProperty().setValue(10.0);
finger2Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

//Specifications of movement for subgroups of finger 3
finger3Ph2Group.rwpx.pivotZProperty().setValue(420.0);
finger3Ph2Group.rwpx.pivotYProperty().setValue(10.0);
finger3Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger3Ph21Group.rwpx.pivotZProperty().setValue(380.0);
finger3Ph21Group.rwpx.pivotYProperty().setValue(10.0);
finger3Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger3Ph210Group.rwpx.pivotZProperty().setValue(340.0);
finger3Ph210Group.rwpx.pivotYProperty().setValue(10.0);
finger3Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

//Specifications of movement for subgroups of finger 4
finger4Ph2Group.rwpx.pivotZProperty().setValue(420.0);
finger4Ph2Group.rwpx.pivotYProperty().setValue(10.0);
finger4Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger4Ph21Group.rwpx.pivotZProperty().setValue(380.0);
```

```
    finger4Ph21Group.rwpx.pivotYProperty().setValue(10.0);
    finger4Ph21Group.rwpx.angleProperty()
            .bindBidirectional(closeProperty);

    finger4Ph210Group.rwpx.pivotZProperty().setValue(340.0);
    finger4Ph210Group.rwpx.pivotYProperty().setValue(10.0);
    finger4Ph210Group.rwpx.angleProperty()
            .bindBidirectional(closeProperty);

    //======OPENING/CLOSING OF FINGERS========

    //One fingers rotates towards one side
    //while the opposite one to the opposite side.
    NumberBinding openingFingersPropertyPlus
            = openingFingersProperty.multiply(1.3);
    NumberBinding openingFingersPropertyMinus
            = openingFingersProperty.multiply(-1.3);

    //Specifications of movement for subgroups of finger 0
    finger0Ph210Group.rwpy.pivotXProperty().setValue(50.);
    finger0Ph210Group.rwpy.pivotYProperty().setValue(10.0);
    finger0Ph210Group.rwpy.pivotZProperty().setValue(340.0);
    finger0Ph210Group.rwpy.angleProperty()
            .bind(openingFingersPropertyMinus);

    //Control instructions for middle fingers are deleted.

    //Specifications of movement for finger 4
    finger4Ph210Group.rwpy.pivotXProperty().setValue(-50.0);
    finger4Ph210Group.rwpy.pivotYProperty().setValue(10.0);
    finger4Ph210Group.rwpy.pivotZProperty().setValue(340.0);
    finger4Ph210Group.rwpy.angleProperty()
            .bind(openingFingersPropertyPlus);

    getChildren().addAll(handGroup );

}//End of constructor of class Hand


}//End of inner class Hand
```

```
//A finger is an ensemble of phalanges
private class Finger extends Group
{

    Phalanx[] Phalanges    = new Phalanx [nPhalanges];

    //Constructor
    Finger(int nPhalanges)
    {
        for(int i =0; i < nPhalanges; i++)
        {
            Phalanx ph = new Phalanx(25,20,40);
            Phalanges[i] = ph;
        }
    }
}//End of inner class Finger



//A phalanx is a JavaFX 3D box
private class Phalanx extends Group
{
    Box box;

    //Constructor
    Phalanx(int width, int length, int height)
    {
        box = new Box(width, length, height);
        Color c = randomColor();
        PhongMaterial materialFinger
                = new PhongMaterial(c);
        box.setMaterial(materialFinger);
        //ph.getChildren().add(box);
    }
}//End of inner class phalanx
```

```
}//En of main class M22 HandOnlyBoxes7
```

23, page 76. The next code implements a hand with opposite thumb.

```
package handonlyboxes8;

/*
 * Program M23 HandOnlyBoxes8.
 * It is shown how to make an ensemble of boxes
 * into a 3D closable hand
 * whose fingers can be moved transversally.
 * The hand has an opposite thumb.
 * The hand can react to touching with the mouse:
 * touched phalanx changes its color to red.

 * As a hand, it is horrible.
 * But a little change can transform it into
 * a marvelous limb of a bird.

 * We copy the anatomical structure:
 * phalanges form fingers that together
 * with the palm conform the hand.
 * It is found that JavaFX is highly compatible
 * with the anatomical organization.
 */



import java.lang.reflect.Field;
import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.beans.binding.NumberBinding;
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;
```

```
import javafx.geometry.Insets;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.PerspectiveCamera;
import javafx.scene.PointLight;
import javafx.scene.Scene;
import javafx.scene.SubScene;
import javafx.scene.control.Label;
import javafx.scene.control.Slider;
import javafx.scene.input.PickResult;
import javafx.scene.layout.HBox;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.Box;
import javafx.scene.transform.Rotate;
import static javafx.scene.transform.Rotate.X_AXIS;
import static javafx.scene.transform.Rotate.Y_AXIS;
import static javafx.scene.transform.Rotate.Z_AXIS;
import javafx.stage.Stage;



public class HandOnlyBoxes8 extends Application
{
    //Node with all elements
    XformWithPivot handGroup = new XformWithPivot();
    XformWithPivot finger4Ph2Group = new XformWithPivot();
    //Slider for zooming
    private Slider zSlider;
    //Position of graphic along the z-axis,
    //normal to the screen.
    //It is a double property because
    //it will get involved into a binding
    private final DoubleProperty zPos
            = new SimpleDoubleProperty(2000);

    //Slider for angle of rotation of fingers
    private Slider closeHandSlider;
    //To close the hand,
    //each phalange suffers a rotation.
```

```
//It is a double property because
//it will get involved into a binding
private final DoubleProperty closeProperty
        = new SimpleDoubleProperty(0);

//Slider for opening fingers thanks to
//their transversal rotation
private Slider openingFingersSlider;
//To open the fingers,
//each phalange suffers a transversal rotation.
//It is a double property because
//it will get involved into a binding
private final DoubleProperty openingFingersProperty
        = new SimpleDoubleProperty(0);

//Slider for opening closing the thumb
private Slider closeThumbSlider;
//To close the thumb we need a rotation.
//It is a double property because
//it will get involved into a binding
private final DoubleProperty closeThumbProperty
        = new SimpleDoubleProperty(0);

//For listener of mouse
private double anchorAngleX = 0;
private double anchorAngleY = 0;
private double anchorX, anchorY;
//Angles of rotation of the scene
private final DoubleProperty angleX
        = new SimpleDoubleProperty(0);
private final DoubleProperty angleY
        = new SimpleDoubleProperty(0);

PerspectiveCamera camera
        = new PerspectiveCamera(false);

//The application is instantiated
static HandOnlyBoxes8 p = new HandOnlyBoxes8();
```

```
    //An array with all colors that are named in JavaFX
    private static final Color JavaFXColors[]
                                = new Color[148];

    //Generator of pseudo random numbers
    private static final Random rd = new Random();

    //Number of fingers
    private final int nFingers = 5;

    //Number of phalanges of each finger
    private final int nPhalanges = 3;

public void init(Stage primaryStage)
{
    primaryStage.setTitle("An ensemble of boxes"
            + " act coherently as a  hand");
    //The whole drawing  is created
    //as a JavaFX space-like object
    final Group parent = new Group(create3dContent());

    //Initial position
    parent.setTranslateX(450);
    parent.setTranslateY(400);
    parent.setTranslateZ(0);
    Rotate xRotate;
    Rotate yRotate;
    //Initialization of rotation
    parent.getTransforms().setAll
    (
      xRotate = new Rotate(0, Rotate.X_AXIS),
      yRotate = new Rotate(0, Rotate.Y_AXIS)
    );

    //Graphic is attached to root
    final Group root = new Group(parent);
    //Window of the scene
    final Scene scene = new Scene(root, 900, 900, true);
    //Background
```

```
scene.setFill(Color.SKYBLUE);

//Listener of mousePressed
scene.setOnMousePressed
(
    event ->
    {
        anchorX = event.getSceneX();
        anchorY = event.getSceneY();
        anchorAngleX = angleX.get();
        anchorAngleY = angleY.get();
    }
);


//Listener of mouseDragged
scene.setOnMouseDragged
(
    event ->
    {
        angleX.set(anchorAngleX
                - (anchorY - event.getSceneY()));
        angleY.set(anchorAngleY
                + anchorX - event.getSceneX());
    }
);

//Rotation of the scene gets bound to
//the angle determined by  mouseDragged
xRotate.angleProperty().bind(angleX);
yRotate.angleProperty().bind(angleY);

//Illumination
PointLight pointLight = new PointLight(Color.WHITE);
pointLight.setTranslateX(100);
pointLight.setTranslateY(200);
pointLight.setTranslateZ(-2000);
root.getChildren().add(pointLight);

//Camera
```

```
scene.setCamera(camera);
primaryStage.setScene(scene);
PerspectiveCamera subSceneCamera
        = new PerspectiveCamera(false);
scene.setCamera(subSceneCamera);

//To zoom down the graphic,
//it is translated away from  the beholder,
//along the Z-direction (normal to the screen).
//Change zPos to change initial setting.
zSlider = new Slider();
zSlider.setMax(5000);
zSlider.setMin(1);


//Bind zPos to the reading of the slider
zSlider.valueProperty().bindBidirectional(zPos);
//Execute negative zoom
parent.translateZProperty().bind(zPos);



//This slider   controls the closing of the hand.
closeHandSlider = new Slider();
closeHandSlider.setMax(90);
closeHandSlider.setMin(0);
closeHandSlider.setValue(10);

//To close the hand, an angle of rotation is enabled.
//Bind angleProperty to the reading of the slider
closeHandSlider.valueProperty()
        .bindBidirectional(closeProperty);


//This slider controls the openning of fingers.
openingFingersSlider = new Slider();
openingFingersSlider.setMax(15);
openingFingersSlider.setMin(0);
openingFingersSlider.setValue(10);
```

```
//To open fingers, an angle of rotation is enabled.
//Bind openingFingersProperty to the reading of the slider
openingFingersSlider.valueProperty()
        .bindBidirectional(openingFingersProperty);


//Slider to open-close the thumb
closeThumbSlider = new Slider();
closeThumbSlider.setMax(110);
closeThumbSlider.setMin(40);
closeThumbSlider.setValue(90);
closeThumbSlider.adjustValue(closeThumbSlider.getValue());

//To open the thumb, an angle of rotation is enabled.
//Bind closeThumbProperty to the reading of the slider
closeThumbSlider.valueProperty()
        .bindBidirectional(closeThumbProperty);

//Other JavaFX controls, labels and containers
Label zLabel = new Label("Zoom");
Label angleLabel = new Label("Close hand");
HBox subSceneGroup = new HBox(70,
    new HBox(10, zLabel, zSlider
             , angleLabel, closeHandSlider
             , openingFingersSlider
             , closeThumbSlider));



subSceneGroup.setPadding(new Insets(10, 100, 0, 100));
SubScene subScene
        = new SubScene(subSceneGroup, 880, 200);
subScene.setLayoutX(10);
subScene.setLayoutY(800);
root.getChildren().add(subScene);

primaryStage.show();

//The sense of touch is here
parent.setOnMouseClicked((event)->
```

```
    {
        //Information that can be retrieved
        PickResult res = event.getPickResult();
        //Which node
        Node node = res.getIntersectedNode();
        //Id of node (null if not declared)
        String nodeId  = event.getPickResult()
                .getIntersectedNode().getId();
        //which face of the node (negative if is not enabled
        int face = event.getPickResult().getIntersectedFace();
        System.out.println("res = " + res);
        System.out.println("node = " + node);
        System.out.println("nodeID = " + nodeId);
        System.out.println("face" + face);
        //Change color to red, rotate around Z_AXIS
        if (node instanceof Box)
            {
                ((Box) node).setMaterial(
                        new PhongMaterial( Color.RED));
                handGroup.rotationAxisProperty()
                        .setValue(Z_AXIS);
                int val = rd.nextInt(91);
                handGroup.rotateProperty().setValue(val);
                finger4Ph2Group.rotateProperty().setValue(val);
            }




    });

}


public void play()
{

}
```

```
@Override public void stop()
{

}

@Override
public void start(Stage primaryStage)  throws Exception
{
    init(primaryStage);
    primaryStage.show();
    play();
}


public static void main(String[] args)
{
    launch(args);
}


//Make the drawing
private Node create3dContent()
{
    //The scene can be rotated
    XformWithPivot sceneRoot = new XformWithPivot();
    sceneRoot.rx.setAngle(45);
    sceneRoot.ry.setAngle(110);
    getColors();
    //The constructor of hand is invoked.
    Hand  h =  new Hand();
    sceneRoot.getChildren().add(h);
    return new Group(sceneRoot);
}

//Colors with a name in JavaFX are put into an array
static private Color[] getColors()
{
    //Colors with a name in JavaFX
    final Field[] fields = Color.class.getFields();
```

```
    int N = 0;
    //Colors are put into an array
    for (final Field field : fields)
    {
        if (field.getType() == Color.class)
        {
            try
            {
                final Color color = (Color) field.get(null);
                //Array to keep colors
                JavaFXColors[N] = color;
                /*
                //The name of the color
                final String colorName = field.getName();
                System.out.println("N = " + N + "  "
                        + color.toString() + " " + colorName);
                */
                N++;
            } catch (IllegalArgumentException
                    | IllegalAccessException ex)
            {
                Logger.getLogger(HandOnlyBoxes8.class
                    .getName()).log(Level.SEVERE, null, ex);
            }
       }
    }
    return JavaFXColors;
}


//A random color is synthesized
 private static Color randomColor()
 {
    //A JavaFX named color is picked
    //but the first, Transparent, is ignored
    int n = rd.nextInt(JavaFXColors.length-1)+1;
    Color c = JavaFXColors[n];
    //Some light colors are not visualized so,
    //they might be darkened:
```

```
    //c = c.darker();
    return c;
 }


  //=================================================
  //=============Inner class XformWithPivot===========
  //=================================================



//Kit of available families of transformations
private class XformWithPivot extends Group
{
    //Declaration and initialization
    public Rotate rx = new Rotate();
    { rx.setAxis(Rotate.X_AXIS); }
    public Rotate ry = new Rotate();
    { ry.setAxis(Rotate.Y_AXIS); }
    public Rotate rz = new Rotate();
    { rz.setAxis(Rotate.Z_AXIS); }
    //Rotation = angle (one coordinate)
    //+ pivot (3 coordinates) + axis.
    //This rotation is around X_AXIS
    public Rotate rwpx = new Rotate(0,0,0,0, X_AXIS);
    //This rotation is around Y_AXIS
    public Rotate rwpy = new Rotate(0,0,0,0, Y_AXIS);

    public XformWithPivot()
    {
        super();
        getTransforms().addAll( rx, ry, rz, rwpx, rwpy );
    }
}//End of class XformWithPivot



  //=========================================
  //=============Inner class Hand============
  //=========================================
```

```
private class Hand extends Group
{


    //Nested definition of   nodes is implemented
    //to have, say,  the palm dragging the fingers,
    //and a phalanx dragging the rest of its the finger.


    //Each finger generates various natural groups:
    //Group with the extreme phalanx of finger0
    //Symbolic initialization
    XformWithPivot finger0Ph2Group = new XformWithPivot();
    //Group with the two extreme phalanges
    XformWithPivot finger0Ph21Group = new XformWithPivot();
    //Group with all three phalanges. It is the finger
    XformWithPivot finger0Ph210Group = new XformWithPivot();
    //Finger1
    XformWithPivot finger1Ph2Group = new XformWithPivot();
    XformWithPivot finger1Ph21Group = new XformWithPivot();
    XformWithPivot finger1Ph210Group = new XformWithPivot();
    //Finger2
    XformWithPivot finger2Ph2Group = new XformWithPivot();
    XformWithPivot finger2Ph21Group = new XformWithPivot();
    XformWithPivot finger2Ph210Group = new XformWithPivot();
    //Finger3
    XformWithPivot finger3Ph2Group = new XformWithPivot();
    XformWithPivot finger3Ph21Group = new XformWithPivot();
    XformWithPivot finger3Ph210Group = new XformWithPivot();
    //Finger4
    XformWithPivot finger4Ph2Group = new XformWithPivot();
    XformWithPivot finger4Ph21Group = new XformWithPivot();
    XformWithPivot finger4Ph210Group = new XformWithPivot();
    //Thumb
    XformWithPivot thumbPh2Group = new XformWithPivot();
    XformWithPivot thumbPh21Group = new XformWithPivot();
```

```
    XformWithPivot thumbPh210Group = new XformWithPivot();


    //Group with all  fingers
    //XformWithPivot allFingersgroup = new XformWithPivot();
    //Group with palm and fingers
    XformWithPivot palmGroup = new XformWithPivot();


Hand()
{

    //Arm is constructed
    Box arm = new Box(100,20,400);
    Color c = randomColor();
    PhongMaterial materialArm = new PhongMaterial(c);
    arm.setMaterial(materialArm);
    arm.setTranslateY(20);
    //Palm is constructed
    Box palm = new Box(125,20,140);
    Color c2 = randomColor();
    PhongMaterial materialPalm = new PhongMaterial(c2);
    palm.setMaterial(materialPalm);
    palm.setTranslateZ(270);
    palm.setTranslateY(20);



    //5 fingers on the same line.
    //No opposable thumb

    Finger Fingers[] = new Finger[nFingers];

    //Groups are constructed
    for(int i = 0; i < nFingers; i++)
    {
        //Each finger has nPhalanges phalanges
        Fingers[i] = new Finger(nPhalanges);
        for(int j =0; j < nPhalanges; j++)
        {
```

```
Box ph = Fingers[i].Phalanges[j].box;

ph.setTranslateX(12.5 * (5-2*(i))-12.5);
ph.setTranslateZ(360 + 40*j);
ph.setTranslateY(20);

//Natural groups are instantiated
//Finger 0 and its subgroups
if (i == 0)
{
    if (j == 2) finger0Ph2Group.getChildren()
            .add(ph);

    if (j == 1) finger0Ph21Group
            .getChildren()
            .addAll(finger0Ph2Group, ph);
    if (j == 0) finger0Ph210Group
            .getChildren()
            .addAll(finger0Ph21Group, ph);
}

//Finger 1  and its subgroups
if (i == 1)
{
    if (j == 2) finger1Ph2Group
            .getChildren().add(ph);
    if (j == 1) finger1Ph21Group
            .getChildren()
            .addAll(finger1Ph2Group, ph);
    if (j == 0) finger1Ph210Group
            .getChildren()
            .addAll(finger1Ph21Group, ph);
}


//Finger 2  and its subgroups
if (i == 2)
{
    if (j == 2) finger2Ph2Group
            .getChildren().add(ph);
```

```
                if (j == 1) finger2Ph21Group
                        .getChildren()
                        .addAll(finger2Ph2Group, ph);
                if (j == 0) finger2Ph210Group
                        .getChildren()
                        .addAll(finger2Ph21Group, ph);
            }


            //Finger 3 and its subgroups
            if (i == 3)
            {
                if (j == 2) finger3Ph2Group.getChildren().add(ph);
                if (j == 1) finger3Ph21Group
                        .getChildren()
                        .addAll(finger3Ph2Group, ph);
                if (j == 0) finger3Ph210Group
                        .getChildren()
                        .addAll(finger3Ph21Group, ph);
            }

            //Finger 4 and its subgroups
            if (i == 4)
            {
                if (j == 2) finger4Ph2Group
                        .getChildren().add(ph);
                if (j == 1) finger4Ph21Group
                        .getChildren()
                        .addAll(finger4Ph2Group, ph);
                if (j == 0) finger4Ph210Group
                        .getChildren()
                        .addAll(finger4Ph21Group, ph);
            }

        }
    }

    //The thumb also has 3 phalanges
    Finger thumb = new Finger(3);
    for(int j = 0; j < nPhalanges; j++)
```

```
    {
        Box ph = thumb.Phalanges[j].box;
        ph.setTranslateX(-50);
        ph.setTranslateZ(250 + 40*j);
    }

thumbPh2Group.getChildren().add(thumb
        .Phalanges[2].box);
thumbPh21Group.getChildren()
        .addAll(thumbPh2Group
                , thumb.Phalanges[1].box);
thumbPh210Group.getChildren()
        .addAll(thumbPh21Group
                , thumb.Phalanges[0].box);

//Fingers are attached to the palm
palmGroup.getChildren().addAll(palm
        , finger0Ph210Group,
        finger1Ph210Group,
        finger2Ph210Group,
        finger3Ph210Group,
        finger4Ph210Group,
        thumbPh210Group);
//Palm and arm are attached to hand
handGroup.getChildren().addAll(arm, palmGroup );



//========OPENING/CL0SING OF THE HAND============

NumberBinding closePropertyHalf
        = closeProperty.multiply(0.5);

//The slider that closes the hand extends its
//control to the palm:
//it must rotate around
//the end of the arm that is parallel to the X_AXIS
//This is the pivot of rotation
palmGroup.rwpx.pivotZProperty().setValue(200.0);
palmGroup.rwpx.pivotYProperty().setValue(10.0);
```

```
//The angle of rotation is controlled by a slider
palmGroup.rwpx.angleProperty()
        .bind(closePropertyHalf);



//Specifications of movement for subgroups of finger 0
finger0Ph2Group.rwpx.pivotZProperty().setValue(420.0);
finger0Ph2Group.rwpx.pivotYProperty().setValue(10.0);
finger0Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger0Ph21Group.rwpx.pivotZProperty().setValue(380.0);
finger0Ph21Group.rwpx.pivotYProperty().setValue(10.0);
finger0Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger0Ph210Group.rwpx.pivotZProperty().setValue(340.0);
finger0Ph210Group.rwpx.pivotYProperty().setValue(10.0);
finger0Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

//Specifications of movement for subgroups of finger 1
finger1Ph2Group.rwpx.pivotZProperty().setValue(420.0);
finger1Ph2Group.rwpx.pivotYProperty().setValue(10.0);
finger1Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger1Ph21Group.rwpx.pivotZProperty().setValue(380.0);
finger1Ph21Group.rwpx.pivotYProperty().setValue(10.0);
finger1Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger1Ph210Group.rwpx.pivotZProperty().setValue(340.0);
finger1Ph210Group.rwpx.pivotYProperty().setValue(10.0);
finger1Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

//Specifications of movement for subgroups of finger 2
finger2Ph2Group.rwpx.pivotZProperty().setValue(420.0);
```

```
finger2Ph2Group.rwpx.pivotYProperty().setValue(10.0);
finger2Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger2Ph21Group.rwpx.pivotZProperty().setValue(380.0);
finger2Ph21Group.rwpx.pivotYProperty().setValue(10.0);
finger2Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger2Ph210Group.rwpx.pivotZProperty().setValue(340.0);
finger2Ph210Group.rwpx.pivotYProperty().setValue(10.0);
finger2Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

//Specifications of movement for subgroups of finger 3
finger3Ph2Group.rwpx.pivotZProperty().setValue(420.0);
finger3Ph2Group.rwpx.pivotYProperty().setValue(10.0);
finger3Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger3Ph21Group.rwpx.pivotZProperty().setValue(380.0);
finger3Ph21Group.rwpx.pivotYProperty().setValue(10.0);
finger3Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger3Ph210Group.rwpx.pivotZProperty().setValue(340.0);
finger3Ph210Group.rwpx.pivotYProperty().setValue(10.0);
finger3Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

//Specifications of movement for subgroups of finger 4
finger4Ph2Group.rwpx.pivotZProperty().setValue(420.0);
finger4Ph2Group.rwpx.pivotYProperty().setValue(10.0);
finger4Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger4Ph21Group.rwpx.pivotZProperty().setValue(380.0);
finger4Ph21Group.rwpx.pivotYProperty().setValue(10.0);
finger4Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);
```

```
finger4Ph210Group.rwpx.pivotZProperty().setValue(340.0);
finger4Ph210Group.rwpx.pivotYProperty().setValue(10.0);
finger4Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

//======OPENING/CLOSING OF FINGERS========

//Fingers that are at the border must
//open twice as those
//near the center.
NumberBinding openingFingersPropertyTwo
        = openingFingersProperty.multiply(2);
//Specifications of movement for subgroups of finger 0
finger0Ph210Group.rwpy.pivotXProperty().setValue(0.0);
finger0Ph210Group.rwpy.pivotYProperty().setValue(10.0);
finger0Ph210Group.rwpy.pivotZProperty().setValue(340.0);
finger0Ph210Group.rwpy.angleProperty()
        .bind(openingFingersPropertyTwo);


//Specifications of movement for subgroups of finger 1
finger1Ph210Group.rwpy.pivotXProperty().setValue(0.0);
finger1Ph210Group.rwpy.pivotYProperty().setValue(30.0);
finger1Ph210Group.rwpy.pivotZProperty().setValue(340.0);

finger1Ph210Group.rwpy.angleProperty()
        .bind(openingFingersProperty);

//Finger two remains invariant under opening of fingers

//Opening in the contrary direction of fingers 3 and 4
NumberBinding openingFingersPropertyMinus
        = openingFingersProperty.multiply(-1);

//Specifications of movement for finger 3
finger3Ph210Group.rwpy.pivotXProperty().setValue(0.0);
finger3Ph210Group.rwpy.pivotYProperty().setValue(30.0);
finger3Ph210Group.rwpy.pivotZProperty().setValue(340.0);
finger3Ph210Group.rwpy.angleProperty()
```

```
        .bind(openingFingersPropertyMinus);

NumberBinding openingFingersPropertyMinusTwo
        = openingFingersProperty.multiply(-2);

//Specifications of movement for finger 4
finger4Ph210Group.rwpy.pivotXProperty().setValue(0.0);
finger4Ph210Group.rwpy.pivotYProperty().setValue(10.0);
finger4Ph210Group.rwpy.pivotZProperty().setValue(340.0);
finger4Ph210Group.rwpy.angleProperty()
        .bind(openingFingersPropertyMinusTwo);

//Specifications of movement for the thumb

NumberBinding closeThumbPh1Property
        = closeThumbProperty.multiply(1).add(-90);

//Phalanx 2
thumbPh2Group.rwpx.pivotXProperty().setValue(0.0);
thumbPh2Group.rwpx.pivotYProperty().setValue(0.0);
thumbPh2Group.rwpx.pivotZProperty().setValue(320.0);
thumbPh2Group.rwpx.angleProperty()
        .bind(closeThumbPh1Property);
//Phalanges 2 and 1
thumbPh21Group.rwpx.pivotXProperty().setValue(0.0);
thumbPh21Group.rwpx.pivotYProperty().setValue(0.0);
thumbPh21Group.rwpx.pivotZProperty().setValue(280.0);
thumbPh21Group.rwpx.angleProperty()
        .bind(closeThumbPh1Property);
//Phalanges 2, 1 and 0
thumbPh210Group.rwpx.pivotXProperty().setValue(0.0);
thumbPh210Group.rwpx.pivotYProperty().setValue(0.0);
thumbPh210Group.rwpx.pivotZProperty().setValue(240.0);
thumbPh210Group.rwpx.angleProperty()
        .bind(closeThumbProperty);



getChildren().addAll(handGroup );
```

```
}//End of constructor of class Hand


}//End of inner class Hand



//A finger is an ensemble of phalanges
private class Finger extends Group
{

    Phalanx[] Phalanges    = new Phalanx [nPhalanges];

    //Constructor
    Finger(int nPhalanges)
    {
        for(int i =0; i < nPhalanges; i++)
        {
            Phalanx ph = new Phalanx(25,20,40);
            Phalanges[i] = ph;
        }
    }
}//End of inner class Finger



//A phalanx is a JavaFX 3D box
private class Phalanx extends Group
{
    Box box;

    //Constructor
    Phalanx(int width, int length, int height)
    {
        box = new Box(width, length, height);
        Color c = randomColor();
        PhongMaterial materialFinger
                = new PhongMaterial(c);
        box.setMaterial(materialFinger);
```

```
    }
}//End of inner class phalanx

}//En of main class M23 HandOnlyBoxes8
```

26, page 98. The next code implements a hand with opposable thumb.

```
/*
 * Program M26 HandOnlyBoxess10.
 * It is shown how to make an ensemble of boxes
 * into a 3D closable hand
 * whose fingers can be moved transversally.
 * The hand has an opposable thumb, i.e.,
 * its thumb is aligned with the other fingers
 * but after a complex movement opposites them
 * to form a strong grip upon closing.
 * The thumb can also get transversal movements.
 * The hand can react to touching with the mouse:
 * touched phalanx changes its color to red.

 * We try to copy the anatomical structure:
 * phalanges form fingers that together
 * with the palm conform the hand.
 * The thumb is initially rotated
 * around the Z_AXIS and around the X_AXIS,
 * just as our hands.

 * It is found that JavaFX is highly compatible
 * with the anatomical organization.
 */

package handonlyboxess10;



import java.lang.reflect.Field;
import java.util.Random;
import java.util.logging.Level;
```

```
import java.util.logging.Logger;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.beans.binding.NumberBinding;
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;
import javafx.geometry.Insets;

import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.PerspectiveCamera;
import javafx.scene.PointLight;
import javafx.scene.Scene;
import javafx.scene.SubScene;
import javafx.scene.control.Label;
import javafx.scene.control.Slider;
import javafx.scene.input.PickResult;
import javafx.scene.layout.GridPane;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.Box;
import javafx.scene.transform.Rotate;
import static javafx.scene.transform.Rotate.X_AXIS;
import static javafx.scene.transform.Rotate.Y_AXIS;
import static javafx.scene.transform.Rotate.Z_AXIS;
import javafx.stage.Stage;



public class HandOnlyBoxess10 extends Application
{
    //Node with all elements
    XformWithPivot armGroup = new XformWithPivot();
    XformWithPivot finger4Ph2Group = new XformWithPivot();
    //Slider for zooming
    private Slider zSlider;
    //Position of graphic along the z-axis,
    //normal to the screen.
    //It is a double property because
    //it will get involved into a binding
```

```
private final DoubleProperty zPos
        = new SimpleDoubleProperty(2000);

//Slider for angle of rotation of fingers
private Slider closeHandSlider;
//To close the hand,
//each phalange suffers a rotation.
//It is a double property because
//it will get involved into a binding
private final DoubleProperty closeProperty
        = new SimpleDoubleProperty(0);

//Slider for opening fingers thanks to
//their transversal rotation
private Slider openFingersSlider;
//To open the fingers,
//each phalange suffers a transversal rotation.
//It is a double property because
//it will get involved into a binding
private final DoubleProperty openFingersProperty
        = new SimpleDoubleProperty(0);

//Slider for opening/closing the thumb
private Slider openThumbSlider;
//To close the thumb we need a rotation.
//It is a double property because
//it will get involved into a binding
private final DoubleProperty openThumbProperty
        = new SimpleDoubleProperty(0);

//Slider for rotating the thumb to form the grip
private Slider rotateThumbSlider;
//To form the grip we need a rotation of the thumb.
//It is a double property because
//it will get involved into a binding
private final DoubleProperty rotateThumbProperty
        = new SimpleDoubleProperty(0);

//For listener of mouse
private double anchorAngleX = 0;
```

```
    private double anchorAngleY = 0;
    private double anchorX, anchorY;
    //Angles of rotation of the scene
    private final DoubleProperty angleX
            = new SimpleDoubleProperty(0);
    private final DoubleProperty angleY
            = new SimpleDoubleProperty(0);

    PerspectiveCamera camera
            = new PerspectiveCamera(false);

    //The application is instantiated
    static HandOnlyBoxess10 p = new HandOnlyBoxess10();



    //An array with all colors that are named in JavaFX
    private static final Color JavaFXColors[]
                                = new Color[148];

    //Generator of pseudo random numbers
    private static final Random rd = new Random();

    //Number of fingers
    private final int nFingers = 4;

    //Number of phalanges of each finger
    private final int nPhalanges = 3;

public void init(Stage primaryStage)
{
    primaryStage.setTitle("An ensemble of boxes"
            + " act coherently as a   hand");
    //The whole drawing  is created
    //as a JavaFX space-like object
    final Group parent = new Group(create3dContent());

    //Initial position
    parent.setTranslateX(450);
    parent.setTranslateY(400);
```

```
parent.setTranslateZ(0);
Rotate xRotate;
Rotate yRotate;
//Initialization of rotation
parent.getTransforms().setAll
(
  xRotate = new Rotate(0, Rotate.X_AXIS),
  yRotate = new Rotate(0, Rotate.Y_AXIS)
);

//Graphic is attached to root
final Group root = new Group(parent);
//Window of the scene
final Scene scene = new Scene(root, 900, 900, true);
//Background
scene.setFill(Color.SKYBLUE);

//Listener of mousePressed
scene.setOnMousePressed
(
    event ->
    {
        anchorX = event.getSceneX();
        anchorY = event.getSceneY();
        anchorAngleX = angleX.get();
        anchorAngleY = angleY.get();
    }
);


//Listener of mouseDragged
scene.setOnMouseDragged
(
    event ->
    {
        angleX.set(anchorAngleX
                - (anchorY - event.getSceneY()));
        angleY.set(anchorAngleY
                + anchorX - event.getSceneX());
    }
```

```
);

//Rotation of the scene gets bound to
//the angle determined by  mouseDragged
xRotate.angleProperty().bind(angleX);
yRotate.angleProperty().bind(angleY);

//Illumination
PointLight pointLight = new PointLight(Color.WHITE);
pointLight.setTranslateX(100);
pointLight.setTranslateY(200);
pointLight.setTranslateZ(-2000);
root.getChildren().add(pointLight);

//Camera
scene.setCamera(camera);
primaryStage.setScene(scene);
PerspectiveCamera subSceneCamera
        = new PerspectiveCamera(false);
scene.setCamera(subSceneCamera);

//To zoom down the graphic,
//it is translated away from  the beholder,
//along the Z-direction (normal to the screen).
//Change zPos to change initial setting.
zSlider = new Slider();
zSlider.setMax(5000);
zSlider.setMin(1);


//Bind zPos to the reading of the slider
zSlider.valueProperty().bindBidirectional(zPos);
//Execute negative zoom
parent.translateZProperty().bind(zPos);



//This slider   controls the closing of the hand.
closeHandSlider = new Slider();
closeHandSlider.setMax(90);
```

```
closeHandSlider.setMin(0);
closeHandSlider.setValue(10);

//To close the hand, an angle of rotation is enabled.
//Bind angleProperty to the reading of the slider
closeHandSlider.valueProperty()
        .bindBidirectional(closeProperty);


//This slider controls the openning of fingers.
openFingersSlider = new Slider();
openFingersSlider.setMax(15);
openFingersSlider.setMin(0);
openFingersSlider.setValue(10);

//To open fingers, an angle of rotation is enabled.
//Bind openFingersProperty to the reading of the slider
openFingersSlider.valueProperty()
        .bindBidirectional(openFingersProperty);


//Slider to open  the thumb
openThumbSlider = new Slider();
openThumbSlider.setMax(55);
openThumbSlider.setMin(0);
openThumbSlider.setValue(0);
openThumbSlider.adjustValue(openThumbSlider.getValue());

//To open the thumb, an angle of rotation is enabled.
//Bind closeThumbProperty to the reading of the slider
openThumbSlider.valueProperty().bindBidirectional(openThumbProperty);


//Slider to rotate  the thumb
rotateThumbSlider = new Slider();
rotateThumbSlider.setMax(60);
rotateThumbSlider.setMin(0);
rotateThumbSlider.setValue(0);
rotateThumbSlider.adjustValue(rotateThumbSlider.getValue());
```

```
//To rotate the thumb, an angle of rotation is enabled.
//Bind closeThumbProperty to the reading of the slider
rotateThumbSlider.valueProperty()
        .bindBidirectional(rotateThumbProperty);



//Other JavaFX controls, labels and containers
GridPane gridPane = new GridPane();
gridPane.setVgap(4);
gridPane.setHgap(10);
gridPane.setPadding(new Insets(5, 5, 5, 5));
//Column 0, Row   0
gridPane.add(new Label("Zoom: "), 0, 0);
gridPane.add(zSlider, 1, 0);
gridPane.add(new Label("Close hand: "), 2, 0);
gridPane.add(closeHandSlider, 3, 0);
gridPane.add(new Label("Open fingers: "), 0, 1);
gridPane.add(openFingersSlider, 1, 1);
gridPane.add(new Label("Open thumb: "), 0, 2);
gridPane.add(openThumbSlider, 1, 2);
gridPane.add(new Label("Rotate thumb: "), 2, 2);
gridPane.add(rotateThumbSlider, 3,2);



 SubScene subScene
        = new SubScene(gridPane, 880, 200);
subScene.setLayoutX(10);
subScene.setLayoutY(800);
root.getChildren().add(subScene);

primaryStage.show();

//The sense of touch is here
parent.setOnMouseClicked((event)->
{
    //Information that can be retrieved
    PickResult res = event.getPickResult();
    //Which node
```

```
        Node node = res.getIntersectedNode();
        //Id of node (null if not declared)
        String nodeId  = event.getPickResult()
                .getIntersectedNode().getId();
        //which face of the node (negative if is not enabled
        int face = event.getPickResult().getIntersectedFace();
        System.out.println("res = " + res);
        System.out.println("node = " + node);
        System.out.println("nodeID = " + nodeId);
        System.out.println("face" + face);
        //Change color to red, rotate around Z_AXIS
        if (node instanceof Box)
            {
                ((Box) node).setMaterial(
                        new PhongMaterial( Color.RED));
                armGroup.rotationAxisProperty()
                        .setValue(Z_AXIS);
                int val = rd.nextInt(91);
                armGroup.rotateProperty().setValue(val);
                finger4Ph2Group.rotateProperty().setValue(val);
            }




    });

}


public void play()
{

}

@Override public void stop()
{

}
```

```
@Override
public void start(Stage primaryStage)  throws Exception
{
    init(primaryStage);
    primaryStage.show();
    play();
}


public static void main(String[] args)
{
    launch(args);
}


//Make the drawing
private Node create3dContent()
{
    //The scene can be rotated
    XformWithPivot sceneRoot = new XformWithPivot();
    sceneRoot.rx.setAngle(45);
    sceneRoot.ry.setAngle(110);
    getColors();
    //The constructor of hand is invoked.
    Hand  h =  new Hand();
    sceneRoot.getChildren().add(h);
    return new Group(sceneRoot);
}

//Colors with a name in JavaFX are put into an array
static private Color[] getColors()
{
    //Colors with a name in JavaFX
    final Field[] fields = Color.class.getFields();


    int N = 0;
    //Colors are put into an array
    for (final Field field : fields)
```

```
    {
        if (field.getType() == Color.class)
        {
            try
            {
                final Color color = (Color) field.get(null);
                //Array to keep colors
                JavaFXColors[N] = color;
                /*
                //The name of the color
                final String colorName = field.getName();
                System.out.println("N = " + N + "  "
                        + color.toString() + " " + colorName);
                */
                N++;
            } catch (IllegalArgumentException
                    | IllegalAccessException ex)
            {
                Logger.getLogger(HandOnlyBoxess10.class
                    .getName()).log(Level.SEVERE, null, ex);
            }
       }
    }
    return JavaFXColors;
}


//A random color is synthesized
 private static Color randomColor()
 {
    //A JavaFX named color is picked
    //but the first, Transparent, is ignored
    int n = rd.nextInt(JavaFXColors.length-1)+1;
    Color c = JavaFXColors[n];
    //Some light colors are not visualized so,
    //they might be darkened:
    //c = c.darker();
    return c;
 }
```

```
  //=================================================
  //============Inner class XformWithPivot===========
  //=================================================



//Kit of available families of transformations
private class XformWithPivot extends Group
{
    //Declaration and initialization
    public Rotate rx = new Rotate();
    { rx.setAxis(Rotate.X_AXIS); }
    public Rotate ry = new Rotate();
    { ry.setAxis(Rotate.Y_AXIS); }
    public Rotate rz = new Rotate();
    { rz.setAxis(Rotate.Z_AXIS); }
    //Rotation = angle (one coordinate)
    //+ pivot (3 coordinates) + axis.
    //This rotation is around X_AXIS
    public Rotate rwpx = new Rotate(0,0,0,0, X_AXIS);
    //This rotation is around Y_AXIS
    public Rotate rwpy = new Rotate(0,0,0,0, Y_AXIS);
    //This rotation is around Z_AXIS
    public Rotate rwpz = new Rotate(0,0,0,0, Z_AXIS);

    public XformWithPivot()
    {
        super();
        getTransforms().addAll( rx, ry, rz, rwpx, rwpy, rwpz );
    }
}//End of class XformWithPivot



  //=========================================
  //============Inner class Hand============
  //=========================================
```

```
private class Hand extends Group
{


    //Nested definition of   nodes is implemented
    //to have, say,  the palm dragging the fingers,
    //and a phalanx dragging the rest of its the finger.


    //Each finger generates various natural groups:
    //Group with the extreme phalanx of finger0
    //Symbolic initialization
    XformWithPivot finger0Ph2Group = new XformWithPivot();
    //Group with the two extreme phalanges
    XformWithPivot finger0Ph21Group = new XformWithPivot();
    //Group with all three phalanges. It is the finger
    XformWithPivot finger0Ph210Group = new XformWithPivot();
    //Finger1
    XformWithPivot finger1Ph2Group = new XformWithPivot();
    XformWithPivot finger1Ph21Group = new XformWithPivot();
    XformWithPivot finger1Ph210Group = new XformWithPivot();
    //Finger2
    XformWithPivot finger2Ph2Group = new XformWithPivot();
    XformWithPivot finger2Ph21Group = new XformWithPivot();
    XformWithPivot finger2Ph210Group = new XformWithPivot();
    //Finger3
    XformWithPivot finger3Ph2Group = new XformWithPivot();
    XformWithPivot finger3Ph21Group = new XformWithPivot();
    XformWithPivot finger3Ph210Group = new XformWithPivot();
    //Finger4
    XformWithPivot finger4Ph2Group = new XformWithPivot();
    XformWithPivot finger4Ph21Group = new XformWithPivot();
    XformWithPivot finger4Ph210Group = new XformWithPivot();
    //Thumb
    XformWithPivot thumbPh2Group = new XformWithPivot();
    XformWithPivot thumbPh21Group = new XformWithPivot();
    XformWithPivot thumbPh210Group = new XformWithPivot();
```

```
//Group with palm and fingers
XformWithPivot palmGroup = new XformWithPivot();


Hand()
{

    //Arm is constructed
    Box arm = new Box(100,20,400);
    Color c = randomColor();
    PhongMaterial materialArm = new PhongMaterial(c);
    arm.setMaterial(materialArm);
    arm.setTranslateY(20);
    //Palm is constructed
    Box palm = new Box(125,20,140);
    Color c2 = randomColor();
    PhongMaterial materialPalm = new PhongMaterial(c2);
    palm.setMaterial(materialPalm);
    palm.setTranslateZ(270);
    palm.setTranslateY(20);



    //4 fingers on the same line.

    Finger Fingers[] = new Finger[nFingers];

    //Groups are constructed
    for(int i = 0; i < nFingers; i++)
    {
        //Each finger has nPhalanges phalanges
        Fingers[i] = new Finger(nPhalanges);
        for(int j =0; j < nPhalanges; j++)
        {
            Box ph = Fingers[i].Phalanges[j].box;

            ph.setTranslateX(15.625 * (4-2*(i))-15.625);
            ph.setTranslateZ(360 + 40*j);
            ph.setTranslateY(20);
```

```
//Natural groups are instantiated
//Finger 0 and its subgroups
if (i == 0)
{
    if (j == 2) finger0Ph2Group.getChildren()
            .add(ph);

    if (j == 1) finger0Ph21Group
            .getChildren()
            .addAll(finger0Ph2Group, ph);
    if (j == 0) finger0Ph210Group
            .getChildren()
            .addAll(finger0Ph21Group, ph);
}

//Finger 1  and its subgroups
if (i == 1)
{
    if (j == 2) finger1Ph2Group
            .getChildren().add(ph);
    if (j == 1) finger1Ph21Group
            .getChildren()
            .addAll(finger1Ph2Group, ph);
    if (j == 0) finger1Ph210Group
            .getChildren()
            .addAll(finger1Ph21Group, ph);
}


//Finger 2  and its subgroups
if (i == 2)
{
    if (j == 2) finger2Ph2Group
            .getChildren().add(ph);
    if (j == 1) finger2Ph21Group
            .getChildren()
            .addAll(finger2Ph2Group, ph);
    if (j == 0) finger2Ph210Group
            .getChildren()
```

```
                        .addAll(finger2Ph21Group, ph);
            }


            //Finger 3 and its subgroups
            if (i == 3)
            {
                if (j == 2) finger3Ph2Group.getChildren().add(ph);
                if (j == 1) finger3Ph21Group
                        .getChildren()
                        .addAll(finger3Ph2Group, ph);
                if (j == 0) finger3Ph210Group
                        .getChildren()
                        .addAll(finger3Ph21Group, ph);
            }

        }
    }

    //Our thumb  has 3 phalanges
    Finger thumb = new Finger(3);
    for(int j = 0; j < nPhalanges; j++)
        {
            Box ph = thumb.Phalanges[j].box;
            //When the thumb is aligned with the hand
            //it has a rotation around its vertical axis
            ph.setRotationAxis(Z_AXIS);
            ph.setRotate(45);
            //Initial position
            ph.setTranslateX(-78.125);
            ph.setTranslateY(15);
            ph.setTranslateZ(250 + 40*j);



        }
    //Natural groups of control
    thumbPh2Group.getChildren().add(thumb
            .Phalanges[2].box);
    thumbPh21Group.getChildren()
```

```
        .addAll(thumbPh2Group
                , thumb.Phalanges[1].box);
thumbPh210Group.getChildren()
        .addAll(thumbPh21Group
                , thumb.Phalanges[0].box);
//Original inclination of the thumb
//in within the   plane of the palm
thumbPh210Group.getTransforms()
        .add(new Rotate(-55, //Angle
                -78.125, 15, 250, //Pivot
                Y_AXIS)); //Axis

//Fingers are attached to the palm
palmGroup.getChildren().addAll(palm
        , finger0Ph210Group,
        finger1Ph210Group,
        finger2Ph210Group,
        finger3Ph210Group,
        thumbPh210Group);
//Palm and arm are attached to armGroup
armGroup.getChildren().addAll(arm, palmGroup );



//========OPENING/CL0SING OF THE HAND============

NumberBinding closePropertyHalf
        = closeProperty.multiply(0.5);

//The slider that closes the hand extends its
//control to the palm:
//it must rotate around
//the end of the arm that is parallel to the X_AXIS
//This is the pivot of rotation
palmGroup.rwpx.pivotZProperty().setValue(200.0);
palmGroup.rwpx.pivotYProperty().setValue(10.0);
//The angle of rotation is controlled by a slider
palmGroup.rwpx.angleProperty()
        .bind(closePropertyHalf);
```

```
//Specifications of movement for subgroups of finger 0
finger0Ph2Group.rwpx.pivotZProperty().setValue(420.0);
finger0Ph2Group.rwpx.pivotYProperty().setValue(10.0);
finger0Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger0Ph21Group.rwpx.pivotZProperty().setValue(380.0);
finger0Ph21Group.rwpx.pivotYProperty().setValue(10.0);
finger0Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger0Ph210Group.rwpx.pivotZProperty().setValue(340.0);
finger0Ph210Group.rwpx.pivotYProperty().setValue(10.0);
finger0Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

//Specifications of movement for subgroups of finger 1
finger1Ph2Group.rwpx.pivotZProperty().setValue(420.0);
finger1Ph2Group.rwpx.pivotYProperty().setValue(10.0);
finger1Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger1Ph21Group.rwpx.pivotZProperty().setValue(380.0);
finger1Ph21Group.rwpx.pivotYProperty().setValue(10.0);
finger1Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger1Ph210Group.rwpx.pivotZProperty().setValue(340.0);
finger1Ph210Group.rwpx.pivotYProperty().setValue(10.0);
finger1Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

//Specifications of movement for subgroups of finger 2
finger2Ph2Group.rwpx.pivotZProperty().setValue(420.0);
finger2Ph2Group.rwpx.pivotYProperty().setValue(10.0);
finger2Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);
```

```
finger2Ph21Group.rwpx.pivotZProperty().setValue(380.0);
finger2Ph21Group.rwpx.pivotYProperty().setValue(10.0);
finger2Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger2Ph210Group.rwpx.pivotZProperty().setValue(340.0);
finger2Ph210Group.rwpx.pivotYProperty().setValue(10.0);
finger2Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

//Specifications of movement for subgroups of finger 3
finger3Ph2Group.rwpx.pivotZProperty().setValue(420.0);
finger3Ph2Group.rwpx.pivotYProperty().setValue(10.0);
finger3Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger3Ph21Group.rwpx.pivotZProperty().setValue(380.0);
finger3Ph21Group.rwpx.pivotYProperty().setValue(10.0);
finger3Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);

finger3Ph210Group.rwpx.pivotZProperty().setValue(340.0);
finger3Ph210Group.rwpx.pivotYProperty().setValue(10.0);
finger3Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeProperty);



//======OPENING/CLOSING OF FINGERS========

//Fingers that are at the border must
//open twice faster as those
//near the center.
NumberBinding openFingersPropertyTwo
        = openFingersProperty.multiply(2);
//Specifications of movement for subgroups of finger 0
finger0Ph210Group.rwpy.pivotXProperty().setValue(0.0);
finger0Ph210Group.rwpy.pivotYProperty().setValue(10.0);
finger0Ph210Group.rwpy.pivotZProperty().setValue(340.0);
finger0Ph210Group.rwpy.angleProperty()
```

```
        .bind(openFingersPropertyTwo);


//Specifications of movement for subgroups of finger 1
finger1Ph210Group.rwpy.pivotXProperty().setValue(0.0);
finger1Ph210Group.rwpy.pivotYProperty().setValue(30.0);
finger1Ph210Group.rwpy.pivotZProperty().setValue(340.0);

finger1Ph210Group.rwpy.angleProperty()
        .bind(openFingersProperty);

//Finger two remains invariant under opening of fingers

//Opening in the contrary direction of fingers 3 and 4
NumberBinding openFingersPropertyMinus
        = openFingersProperty.multiply(-1);

//Specifications of movement for finger 3
finger3Ph210Group.rwpy.pivotXProperty().setValue(0.0);
finger3Ph210Group.rwpy.pivotYProperty().setValue(30.0);
finger3Ph210Group.rwpy.pivotZProperty().setValue(340.0);
finger3Ph210Group.rwpy.angleProperty()
        .bind(openFingersPropertyMinus);

NumberBinding openFingersPropertyMinusTwo
        = openFingersProperty.multiply(-2);

//Specifications of movement for finger 4
finger4Ph210Group.rwpy.pivotXProperty().setValue(0.0);
finger4Ph210Group.rwpy.pivotYProperty().setValue(10.0);
finger4Ph210Group.rwpy.pivotZProperty().setValue(340.0);
finger4Ph210Group.rwpy.angleProperty()
        .bind(openFingersPropertyMinusTwo);

//====== The thumb ========
//Specifications of movement for the thumb


NumberBinding rotateThumbPropertyDown
        = rotateThumbProperty.multiply(0.2);
```

```
NumberBinding rotateThumbPropertyUp
        = rotateThumbProperty.multiply(2);


//Tranversal movement
thumbPh210Group.rwpy.pivotXProperty().setValue(-78.125);
thumbPh210Group.rwpy.pivotZProperty().setValue(240.0);
thumbPh210Group.rwpy.angleProperty().bind(openThumbProperty);


//Rotations of phalanges 2 and 1 to make the grip.
//Rotations around X_AXIS and Y_AXIS at the same time
//Phalanx 2
thumbPh2Group.rwpx.pivotXProperty().setValue(-70.125);
thumbPh2Group.rwpx.pivotYProperty().setValue(15);
thumbPh2Group.rwpx.pivotZProperty().setValue(310.0);
thumbPh2Group.rwpx.angleProperty()
        .bind(rotateThumbPropertyDown);
//Phalanges 2 and 1
thumbPh21Group.rwpx.pivotXProperty().setValue(-70.125);
thumbPh21Group.rwpx.pivotYProperty().setValue(15);
thumbPh21Group.rwpx.pivotZProperty().setValue(290.0);
thumbPh21Group.rwpx.angleProperty()
        .bind(rotateThumbPropertyDown);

//Phalanx 2
thumbPh2Group.rwpy.pivotXProperty().setValue(-70.125);
thumbPh2Group.rwpy.pivotYProperty().setValue(15);
thumbPh2Group.rwpy.pivotZProperty().setValue(310.0);
thumbPh2Group.rwpy.angleProperty()
        .bind(rotateThumbPropertyDown);
//Phalanges 2 and 1
thumbPh21Group.rwpy.pivotXProperty().setValue(-70.125);
thumbPh21Group.rwpy.pivotYProperty().setValue(15);
thumbPh21Group.rwpy.pivotZProperty().setValue(290.0);
thumbPh21Group.rwpy.angleProperty()
        .bind(rotateThumbPropertyDown);

//Rotation of the whole thumb
```

```
        //to get opposite to other fingers
        thumbPh210Group.rwpz.pivotXProperty().setValue(-70.125);
        thumbPh210Group.rwpz.pivotYProperty().setValue(15);
        thumbPh210Group.rwpz.pivotZProperty().setValue(250.0);
        thumbPh210Group.rwpz.angleProperty()
                .bind(rotateThumbPropertyUp);

        getChildren().addAll(armGroup );

    }//End of constructor of class Hand


}//End of inner class Hand



//A finger is an ensemble of phalanges
private class Finger extends Group
{

    Phalanx[] Phalanges    = new Phalanx [nPhalanges];

    //Constructor
    Finger(int nPhalanges)
    {
        for(int i =0; i < nPhalanges; i++)
        {
            Phalanx ph = new Phalanx(31.25,20,40);
            Phalanges[i] = ph;
        }
    }
}//End of inner class Finger



//A phalanx is a JavaFX 3D box
private class Phalanx extends Group
{
    Box box;
```

```
    //Constructor
    Phalanx(double width, int length, int height)
    {
        box = new Box(width, length, height);
        Color c = randomColor();
        PhongMaterial materialFinger
                = new PhongMaterial(c);
        box.setMaterial(materialFinger);
    }
}//End of inner class phalanx

}//En of main class M26 HandOnlyBoxess10
```

29, page 122. The following code shows an animated hand that also can be controlled by sliders

```
/*
 * Program M29 HandOnlyBoxess12.
 * It is shown how to make an ensemble of boxes
 * into a 3D closable hand
 * whose fingers can be moved transversally.
 * The hand has an opposable thumb, i.e.,
 * its thumb is aligned with the other fingers
 * but after a complex movement opposites them
 * to form a strong grip upon closing.
 * The thumb can also get transversal movements.
 * The hand can react to touching with the mouse:
 * touched phalanx changes its color to red.

 * A complete animated show is included.

 * We try to copy the anatomical structure:
 * phalanges form fingers that together
 * with the palm conform the hand.
 * The thumb is initially rotated
 * around the Z_AXIS and around the X_AXIS,
 * just as our hands.
```

```
 * It is found that JavaFX is highly compatible
 * with the anatomical organization.
 */

package handonlyboxess12;



import java.lang.reflect.Field;
import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.animation.KeyFrame;
import javafx.animation.KeyValue;
import javafx.animation.SequentialTransition;
import javafx.animation.Timeline;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.beans.binding.NumberBinding;
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;
import javafx.geometry.Insets;

import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.PerspectiveCamera;
import javafx.scene.PointLight;
import javafx.scene.Scene;
import javafx.scene.SubScene;
import javafx.scene.control.Label;
import javafx.scene.control.Slider;
import javafx.scene.input.PickResult;
import javafx.scene.layout.GridPane;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.Box;
import javafx.scene.transform.Rotate;
import static javafx.scene.transform.Rotate.X_AXIS;
import static javafx.scene.transform.Rotate.Y_AXIS;
```

```
import static javafx.scene.transform.Rotate.Z_AXIS;
import javafx.stage.Stage;
import javafx.util.Duration;



public class HandOnlyBoxess12 extends Application
{
    //Node with all elements
    XformWithPivot armGroup = new XformWithPivot();
    XformWithPivot finger4Ph2Group = new XformWithPivot();
    //Slider for zooming
    private Slider zSlider;
    //Position of graphic along the z-axis,
    //normal to the screen.
    //It is a double property because
    //it will get involved into a binding
    private final DoubleProperty zPos
            = new SimpleDoubleProperty(2000);

    //Slider for angle of rotation of fingers
    private Slider closeHandSlider;
    //To close the hand,
    //each phalange suffers a rotation.
    //It is a double property because
    //it will get involved into a binding
    private final DoubleProperty closeHandProperty
            = new SimpleDoubleProperty(0);

    //Slider for opening fingers thanks to
    //their transversal rotation
    private Slider openFingersSlider;
    //To open the fingers,
    //each phalange suffers a transversal rotation.
    //It is a double property because
    //it will get involved into a binding
    private final DoubleProperty openFingersProperty
            = new SimpleDoubleProperty(0);

    //Slider for opening/closing the thumb
```

```
private Slider closeThumbSlider;
//To close the thumb we need a rotation.
//It is a double property because
//it will get involved into a binding
private final DoubleProperty closeThumbProperty
        = new SimpleDoubleProperty(0);

//Slider for rotating the thumb to form the grip
private Slider rotateThumbSlider;
//To form the grip we need a rotation of the thumb.
//It is a double property because
//it will get involved into a binding
private final DoubleProperty rotateThumbProperty
        = new SimpleDoubleProperty(0);

//For listener of mouse
private double anchorAngleX = 0;
private double anchorAngleY = 0;
private double anchorX, anchorY;
//Angles of rotation of the scene
private final DoubleProperty angleX
        = new SimpleDoubleProperty(0);
private final DoubleProperty angleY
        = new SimpleDoubleProperty(0);

PerspectiveCamera camera
        = new PerspectiveCamera(false);

//The application is instantiated
static HandOnlyBoxess12 p = new HandOnlyBoxess12();



//An array with all colors that are named in JavaFX
private static final Color JavaFXColors[]
                            = new Color[148];

//Generator of pseudo random numbers
private static final Random rd = new Random();
```

```
    //Number of fingers
    private final int nFingers = 4;

    //Number of phalanges of each finger
    private final int nPhalanges = 3;

    //Overall JavaFX animation
    private static  SequentialTransition allST
                    = new SequentialTransition();
    private final Timeline animation  = new Timeline();

public void init(Stage primaryStage)
{
    primaryStage.setTitle("An ensemble of boxes"
            + " act coherently as a   hand");
    //The whole drawing  is created
    //as a JavaFX space-like object
    final Group parent = new Group(create3dContent());

    //Initial position
    parent.setTranslateX(450);
    parent.setTranslateY(400);
    parent.setTranslateZ(0);
    Rotate xRotate;
    Rotate yRotate;
    //Initialization of rotation
    parent.getTransforms().setAll
    (
      xRotate = new Rotate(0, Rotate.X_AXIS),
      yRotate = new Rotate(0, Rotate.Y_AXIS)
    );

    //Graphic is attached to root
    final Group root = new Group(parent);
    //Window of the scene
    final Scene scene = new Scene(root, 900, 900, true);
    //Background
    scene.setFill(Color.SKYBLUE);

    //Listener of mousePressed
```

```
scene.setOnMousePressed
(
    event ->
    {
        anchorX = event.getSceneX();
        anchorY = event.getSceneY();
        anchorAngleX = angleX.get();
        anchorAngleY = angleY.get();
    }
);


//Listener of mouseDragged
scene.setOnMouseDragged
(
    event ->
    {
        angleX.set(anchorAngleX
                - (anchorY - event.getSceneY()));
        angleY.set(anchorAngleY
                + anchorX - event.getSceneX());
    }
);

//Rotation of the scene gets bound to
//the angle determined by  mouseDragged
xRotate.angleProperty().bind(angleX);
yRotate.angleProperty().bind(angleY);

//Illumination
PointLight pointLight = new PointLight(Color.WHITE);
pointLight.setTranslateX(100);
pointLight.setTranslateY(200);
pointLight.setTranslateZ(-2000);
root.getChildren().add(pointLight);

//Camera
scene.setCamera(camera);
primaryStage.setScene(scene);
PerspectiveCamera subSceneCamera
```

```
        = new PerspectiveCamera(false);
scene.setCamera(subSceneCamera);

//To zoom down the graphic,
//it is translated away from  the beholder,
//along the Z-direction (normal to the screen).
//Change zPos to change initial setting.
zSlider = new Slider();
zSlider.setMax(5000);
zSlider.setMin(1);


//Bind zPos to the reading of the slider
zSlider.valueProperty().bindBidirectional(zPos);
//Execute negative zoom
parent.translateZProperty().bind(zPos);



//This slider   controls the closing of the hand.
closeHandSlider = new Slider();
closeHandSlider.setMax(90);
closeHandSlider.setMin(0);
closeHandSlider.setValue(10);

//To close the hand, an angle of rotation is enabled.
//Bind angleProperty to the reading of the slider
closeHandSlider.valueProperty()
        .bindBidirectional(closeHandProperty);


//This slider controls the openning of fingers.
openFingersSlider = new Slider();
openFingersSlider.setMax(15);
openFingersSlider.setMin(0);
openFingersSlider.setValue(10);

//To open fingers, an angle of rotation is enabled.
//Bind openFingersProperty to the reading of the slider
openFingersSlider.valueProperty()
```

```
        .bindBidirectional(openFingersProperty);


//Slider to open  the thumb
closeThumbSlider = new Slider();
closeThumbSlider.setMax(55);
closeThumbSlider.setMin(0);
closeThumbSlider.setValue(0);
closeThumbSlider.adjustValue(closeThumbSlider.getValue());

//To open the thumb, an angle of rotation is enabled.
//Bind closeThumbProperty to the reading of the slider
closeThumbSlider.valueProperty().bindBidirectional(closeThumbPrope


//Slider to rotate  the thumb
rotateThumbSlider = new Slider();
rotateThumbSlider.setMax(50);
rotateThumbSlider.setMin(0);
rotateThumbSlider.setValue(0);
rotateThumbSlider.adjustValue(rotateThumbSlider.getValue());

//To rotate the thumb, an angle of rotation is enabled.
//Bind closeThumbProperty to the reading of the slider
rotateThumbSlider.valueProperty()
        .bindBidirectional(rotateThumbProperty);



//Other JavaFX controls, labels and containers
GridPane gridPane = new GridPane();
gridPane.setVgap(4);
gridPane.setHgap(10);
gridPane.setPadding(new Insets(5, 5, 5, 5));
//Column 0, Row   0
gridPane.add(new Label("Zoom: "), 0, 0);
gridPane.add(zSlider, 1, 0);
gridPane.add(new Label("Close hand: "), 2, 0);
gridPane.add(closeHandSlider, 3, 0);
gridPane.add(new Label("Open fingers: "), 0, 1);
```

```
gridPane.add(openFingersSlider, 1, 1);
gridPane.add(new Label("Close thumb: "), 0, 2);
gridPane.add(closeThumbSlider, 1, 2);
gridPane.add(new Label("Rotate thumb: "), 2, 2);
gridPane.add(rotateThumbSlider, 3,2);



SubScene subScene
        = new SubScene(gridPane, 880, 200);
subScene.setLayoutX(10);
subScene.setLayoutY(800);
root.getChildren().add(subScene);

primaryStage.show();

//The sense of touch is here
parent.setOnMouseClicked((event)->
{
    //Information that can be retrieved
    PickResult res = event.getPickResult();
    //Which node
    Node node = res.getIntersectedNode();
    //Id of node (null if not declared)
    String nodeId  = event.getPickResult()
            .getIntersectedNode().getId();
    //which face of the node (negative if is not enabled
    int face = event.getPickResult().getIntersectedFace();
    System.out.println("res = " + res);
    System.out.println("node = " + node);
    System.out.println("nodeID = " + nodeId);
    System.out.println("face" + face);
    //Change color to red, rotate around Z_AXIS
    if (node instanceof Box)
        {
            ((Box) node).setMaterial(
                    new PhongMaterial( Color.RED));
            armGroup.rotationAxisProperty()
                    .setValue(Z_AXIS);
            int val = rd.nextInt(91);
```

```
                armGroup.rotateProperty().setValue(val);
                finger4Ph2Group.rotateProperty().setValue(val);
            }




    });

    SequentialTransition a = new SequentialTransition();

}


public void play()
{

}

@Override public void stop()
{

}

@Override
public void start(Stage primaryStage)  throws Exception
{
    init(primaryStage);
    primaryStage.show();

    play();
}


public static void main(String[] args)
{
    launch(args);
}
```

```
//Make the drawing
private Node create3dContent()
{
    //The scene can be rotated
    XformWithPivot sceneRoot = new XformWithPivot();
    sceneRoot.rx.setAngle(45);
    sceneRoot.ry.setAngle(110);
    getColors();
    //The constructor of hand is invoked.
    Hand  h =  new Hand();
    sceneRoot.getChildren().add(h);
    return new Group(sceneRoot);
}

//Colors with a name in JavaFX are put into an array
static private Color[] getColors()
{
    //Colors with a name in JavaFX
    final Field[] fields = Color.class.getFields();


    int N = 0;
    //Colors are put into an array
    for (final Field field : fields)
    {
        if (field.getType() == Color.class)
        {
            try
            {
                final Color color = (Color) field.get(null);
                //Array to keep colors
                JavaFXColors[N] = color;
                /*
                //The name of the color
                final String colorName = field.getName();
                System.out.println("N = " + N + "  "
                        + color.toString() + " " + colorName);
                */
                N++;
```

```
            } catch (IllegalArgumentException
                    | IllegalAccessException ex)
            {
                Logger.getLogger(HandOnlyBoxess12.class
                    .getName()).log(Level.SEVERE, null, ex);
            }
        }
    }
    return JavaFXColors;
}


//A random color is synthesized
 private static Color randomColor()
 {
    //A JavaFX named color is picked
    //but the first, Transparent, is ignored
    int n = rd.nextInt(JavaFXColors.length-1)+1;
    Color c = JavaFXColors[n];
    //Some light colors are not visualized so,
    //they might be darkened:
    //c = c.darker();
    return c;
 }


  //=================================================
  //=============Inner class XformWithPivot===========
  //=================================================



//Kit of available families of transformations
private class XformWithPivot extends Group
{
    //Declaration and initialization
    public Rotate rx = new Rotate();
    { rx.setAxis(Rotate.X_AXIS); }
    public Rotate ry = new Rotate();
    { ry.setAxis(Rotate.Y_AXIS); }
```

```
    public Rotate rz = new Rotate();
    { rz.setAxis(Rotate.Z_AXIS); }
    //Rotation = angle (one coordinate)
    //+ pivot (3 coordinates) + axis.
    //This rotation is around X_AXIS
    public Rotate rwpx = new Rotate(0,0,0,0, X_AXIS);
    //This rotation is around Y_AXIS
    public Rotate rwpy = new Rotate(0,0,0,0, Y_AXIS);
    //This rotation is around Z_AXIS
    public Rotate rwpz = new Rotate(0,0,0,0, Z_AXIS);

    public XformWithPivot()
    {
        super();
        getTransforms().addAll( rx, ry, rz, rwpx, rwpy, rwpz );
    }
}//End of class XformWithPivot



  //=========================================
  //=============Inner class Hand============
  //=========================================



private class Hand extends Group
{



    //Nested definition of   nodes is implemented
    //to have, say,  the palm dragging the fingers,
    //and a phalanx dragging the rest of its the finger.


    //Each finger generates various natural groups:
    //Group with the extreme phalanx of finger0
    //Symbolic initialization
    XformWithPivot finger0Ph2Group = new XformWithPivot();
```

```
    //Group with the two extreme phalanges
    XformWithPivot finger0Ph21Group = new XformWithPivot();
    //Group with all three phalanges. It is the finger
    XformWithPivot finger0Ph210Group = new XformWithPivot();
    //Finger1
    XformWithPivot finger1Ph2Group = new XformWithPivot();
    XformWithPivot finger1Ph21Group = new XformWithPivot();
    XformWithPivot finger1Ph210Group = new XformWithPivot();
    //Finger2
    XformWithPivot finger2Ph2Group = new XformWithPivot();
    XformWithPivot finger2Ph21Group = new XformWithPivot();
    XformWithPivot finger2Ph210Group = new XformWithPivot();
    //Finger3
    XformWithPivot finger3Ph2Group = new XformWithPivot();
    XformWithPivot finger3Ph21Group = new XformWithPivot();
    XformWithPivot finger3Ph210Group = new XformWithPivot();
    //Finger4
    XformWithPivot finger4Ph2Group = new XformWithPivot();
    XformWithPivot finger4Ph21Group = new XformWithPivot();
    XformWithPivot finger4Ph210Group = new XformWithPivot();
    //Thumb
    XformWithPivot thumbPh2Group = new XformWithPivot();
    XformWithPivot thumbPh21Group = new XformWithPivot();
    XformWithPivot thumbPh210Group = new XformWithPivot();


    //Group with palm and fingers
    XformWithPivot palmGroup = new XformWithPivot();


Hand()
{

    //Arm is constructed
    Box arm = new Box(100,20,400);
    Color c = randomColor();
    PhongMaterial materialArm = new PhongMaterial(c);
    arm.setMaterial(materialArm);
    arm.setTranslateY(20);
    //Palm is constructed
```

```
Box palm = new Box(125,20,140);
Color c2 = randomColor();
PhongMaterial materialPalm = new PhongMaterial(c2);
palm.setMaterial(materialPalm);
palm.setTranslateZ(270);
palm.setTranslateY(20);



//4 fingers on the same line.

Finger Fingers[] = new Finger[nFingers];

//Groups are constructed
for(int i = 0; i < nFingers; i++)
{
    //Each finger has nPhalanges phalanges
    Fingers[i] = new Finger(nPhalanges);
    for(int j =0; j < nPhalanges; j++)
    {
        Box ph = Fingers[i].Phalanges[j].box;

        ph.setTranslateX(15.625 * (4-2*(i))-15.625);
        ph.setTranslateZ(360 + 40*j);
        ph.setTranslateY(20);

        //Natural groups are instantiated
        //Finger 0 and its subgroups
        if (i == 0)
        {
            if (j == 2) finger0Ph2Group.getChildren()
                    .add(ph);

            if (j == 1) finger0Ph21Group
                    .getChildren()
                    .addAll(finger0Ph2Group, ph);
            if (j == 0) finger0Ph210Group
                    .getChildren()
                    .addAll(finger0Ph21Group, ph);
        }
```

```
//Finger 1  and its subgroups
if (i == 1)
{
    if (j == 2) finger1Ph2Group
            .getChildren().add(ph);
    if (j == 1) finger1Ph21Group
            .getChildren()
            .addAll(finger1Ph2Group, ph);
    if (j == 0) finger1Ph210Group
            .getChildren()
            .addAll(finger1Ph21Group, ph);
}


//Finger 2  and its subgroups
if (i == 2)
{
    if (j == 2) finger2Ph2Group
            .getChildren().add(ph);
    if (j == 1) finger2Ph21Group
            .getChildren()
            .addAll(finger2Ph2Group, ph);
    if (j == 0) finger2Ph210Group
            .getChildren()
            .addAll(finger2Ph21Group, ph);
}


//Finger 3 and its subgroups
if (i == 3)
{
    if (j == 2) finger3Ph2Group.getChildren().add(ph);
    if (j == 1) finger3Ph21Group
            .getChildren()
            .addAll(finger3Ph2Group, ph);
    if (j == 0) finger3Ph210Group
            .getChildren()
            .addAll(finger3Ph21Group, ph);
}
```

```
    }
}

//Our thumb  has 3 phalanges
Finger thumb = new Finger(3);
for(int j = 0; j < nPhalanges; j++)
    {
        Box ph = thumb.Phalanges[j].box;
        //When the thumb is aligned with the hand
        //it has a rotation around its vertical axis
        ph.setRotationAxis(Z_AXIS);
        ph.setRotate(45);
        //Initial position
        ph.setTranslateX(-78.125);
        ph.setTranslateY(15);
        ph.setTranslateZ(250 + 40*j);



    }
//Natural groups of control
thumbPh2Group.getChildren().add(thumb
        .Phalanges[2].box);
thumbPh21Group.getChildren()
        .addAll(thumbPh2Group
                , thumb.Phalanges[1].box);
thumbPh210Group.getChildren()
        .addAll(thumbPh21Group
                , thumb.Phalanges[0].box);
//Original inclination of the thumb
//in within the   plane of the palm
thumbPh210Group.getTransforms()
        .add(new Rotate(-55, //Angle
                -78.125, 15, 250, //Pivot
                Y_AXIS)); //Axis

//Fingers are attached to the palm
palmGroup.getChildren().addAll(palm
        , finger0Ph210Group,
```

```
        finger1Ph210Group,
        finger2Ph210Group,
        finger3Ph210Group,
        thumbPh210Group);
//Palm and arm are attached to armGroup
armGroup.getChildren().addAll(arm, palmGroup );



//========OPENING/CL0SING OF THE HAND============

NumberBinding closeHandPropertyHalf
        = closeHandProperty.multiply(0.5);

//The slider that closes the hand extends its
//control to the palm:
//it must rotate around
//the end of the arm that is parallel to the X_AXIS
//This is the pivot of rotation
palmGroup.rwpx.pivotZProperty().setValue(200.0);
palmGroup.rwpx.pivotYProperty().setValue(10.0);
//The angle of rotation is controlled by a slider
palmGroup.rwpx.angleProperty()
        .bind(closeHandPropertyHalf);



//Specifications of movement for subgroups of finger 0
finger0Ph2Group.rwpx.pivotZProperty().setValue(420.0);
finger0Ph2Group.rwpx.pivotYProperty().setValue(10.0);
finger0Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

finger0Ph21Group.rwpx.pivotZProperty().setValue(380.0);
finger0Ph21Group.rwpx.pivotYProperty().setValue(10.0);
finger0Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

finger0Ph210Group.rwpx.pivotZProperty().setValue(340.0);
finger0Ph210Group.rwpx.pivotYProperty().setValue(10.0);
```

```
finger0Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

//Specifications of movement for subgroups of finger 1
finger1Ph2Group.rwpx.pivotZProperty().setValue(420.0);
finger1Ph2Group.rwpx.pivotYProperty().setValue(10.0);
finger1Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

finger1Ph21Group.rwpx.pivotZProperty().setValue(380.0);
finger1Ph21Group.rwpx.pivotYProperty().setValue(10.0);
finger1Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

finger1Ph210Group.rwpx.pivotZProperty().setValue(340.0);
finger1Ph210Group.rwpx.pivotYProperty().setValue(10.0);
finger1Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

//Specifications of movement for subgroups of finger 2
finger2Ph2Group.rwpx.pivotZProperty().setValue(420.0);
finger2Ph2Group.rwpx.pivotYProperty().setValue(10.0);
finger2Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

finger2Ph21Group.rwpx.pivotZProperty().setValue(380.0);
finger2Ph21Group.rwpx.pivotYProperty().setValue(10.0);
finger2Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

finger2Ph210Group.rwpx.pivotZProperty().setValue(340.0);
finger2Ph210Group.rwpx.pivotYProperty().setValue(10.0);
finger2Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

//Specifications of movement for subgroups of finger 3
finger3Ph2Group.rwpx.pivotZProperty().setValue(420.0);
finger3Ph2Group.rwpx.pivotYProperty().setValue(10.0);
finger3Ph2Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);
```

```
finger3Ph21Group.rwpx.pivotZProperty().setValue(380.0);
finger3Ph21Group.rwpx.pivotYProperty().setValue(10.0);
finger3Ph21Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);

finger3Ph210Group.rwpx.pivotZProperty().setValue(340.0);
finger3Ph210Group.rwpx.pivotYProperty().setValue(10.0);
finger3Ph210Group.rwpx.angleProperty()
        .bindBidirectional(closeHandProperty);



//======OPENING/CLOSING OF FINGERS========

//Fingers that are at the border must
//open twice faster as those
//near the center.
NumberBinding openFingersPropertyTwo
        = openFingersProperty.multiply(2);
//Specifications of movement for subgroups of finger 0
finger0Ph210Group.rwpy.pivotXProperty().setValue(0.0);
finger0Ph210Group.rwpy.pivotYProperty().setValue(10.0);
finger0Ph210Group.rwpy.pivotZProperty().setValue(340.0);
finger0Ph210Group.rwpy.angleProperty()
        .bind(openFingersPropertyTwo);


//Specifications of movement for subgroups of finger 1
finger1Ph210Group.rwpy.pivotXProperty().setValue(0.0);
finger1Ph210Group.rwpy.pivotYProperty().setValue(30.0);
finger1Ph210Group.rwpy.pivotZProperty().setValue(340.0);

finger1Ph210Group.rwpy.angleProperty()
        .bind(openFingersProperty);

//Finger two remains invariant under opening of fingers

//Opening in the contrary direction of fingers 3 and 4
NumberBinding openFingersPropertyMinus
```

```
        = openFingersProperty.multiply(-1);

//Specifications of movement for finger 3
finger3Ph210Group.rwpy.pivotXProperty().setValue(0.0);
finger3Ph210Group.rwpy.pivotYProperty().setValue(30.0);
finger3Ph210Group.rwpy.pivotZProperty().setValue(340.0);
finger3Ph210Group.rwpy.angleProperty()
        .bind(openFingersPropertyMinus);

NumberBinding openFingersPropertyMinusTwo
        = openFingersProperty.multiply(-2);

//Specifications of movement for finger 4
finger4Ph210Group.rwpy.pivotXProperty().setValue(0.0);
finger4Ph210Group.rwpy.pivotYProperty().setValue(10.0);
finger4Ph210Group.rwpy.pivotZProperty().setValue(340.0);
finger4Ph210Group.rwpy.angleProperty()
        .bind(openFingersPropertyMinusTwo);

//====== The thumb ========
//Specifications of movement for the thumb


NumberBinding rotateThumbPropertyDown
        = rotateThumbProperty.multiply(0.2);

NumberBinding rotateThumbPropertyUp
        = rotateThumbProperty.multiply(2);


//Tranversal movement
thumbPh210Group.rwpy.pivotXProperty().setValue(-78.125);
thumbPh210Group.rwpy.pivotZProperty().setValue(240.0);
thumbPh210Group.rwpy.angleProperty().bind(closeThumbProperty);


//Rotations of phalanges 2 and 1 to make the grip.
//Rotations around X_AXIS and Y_AXIS at the same time
//Phalanx 2
thumbPh2Group.rwpx.pivotXProperty().setValue(-70.125);
```

```
thumbPh2Group.rwpx.pivotYProperty().setValue(15);
thumbPh2Group.rwpx.pivotZProperty().setValue(310.0);
thumbPh2Group.rwpx.angleProperty()
        .bind(rotateThumbPropertyDown);
//Phalanges 2 and 1
thumbPh21Group.rwpx.pivotXProperty().setValue(-70.125);
thumbPh21Group.rwpx.pivotYProperty().setValue(15);
thumbPh21Group.rwpx.pivotZProperty().setValue(290.0);
thumbPh21Group.rwpx.angleProperty()
        .bind(rotateThumbPropertyDown);

//Phalanx 2
thumbPh2Group.rwpy.pivotXProperty().setValue(-70.125);
thumbPh2Group.rwpy.pivotYProperty().setValue(15);
thumbPh2Group.rwpy.pivotZProperty().setValue(310.0);
thumbPh2Group.rwpy.angleProperty()
        .bind(rotateThumbPropertyDown);
//Phalanges 2 and 1
thumbPh21Group.rwpy.pivotXProperty().setValue(-70.125);
thumbPh21Group.rwpy.pivotYProperty().setValue(15);
thumbPh21Group.rwpy.pivotZProperty().setValue(290.0);
thumbPh21Group.rwpy.angleProperty()
        .bind(rotateThumbPropertyDown);

//Rotation of the whole thumb
//to get opposite to other fingers
thumbPh210Group.rwpz.pivotXProperty().setValue(-70.125);
thumbPh210Group.rwpz.pivotYProperty().setValue(15);
thumbPh210Group.rwpz.pivotZProperty().setValue(250.0);
thumbPh210Group.rwpz.angleProperty()
  .bind(rotateThumbProperty.multiply(2));

getChildren().addAll(armGroup );

//===========Animation=========

double endTime1 = 4000.0; ////1000 = 1 second
double endTime2 = 2 * endTime1;
double endTime3 = 3 * endTime1;
double endTime4 = 4 * endTime1;
```

```
double endTime5 = 5 * endTime1;
double endTime6 = 6 * endTime1;
double endTime7 = 7 * endTime1;
double endTime8 = 8 * endTime1;



animation.getKeyFrames().addAll
(

    //The hand gets closed
    //closeHandProperty: initial position at initial time
    new KeyFrame(Duration.ZERO,
        //Permanent condition
        new KeyValue(closeHandProperty, 0)),
    //closeHandProperty: final position at final time
    new KeyFrame(new Duration(endTime1),
        new KeyValue(closeHandProperty, 90)),


    //The thumb closes the grip
    //rotateThumbProperty: initial position at initial time
    new KeyFrame(new Duration(endTime1),
        //Permanent condition
        new KeyValue(rotateThumbProperty, 0)),
    //rotateThumbProperty: final position at final time
    new KeyFrame(new Duration(endTime2),
        new KeyValue(rotateThumbProperty, 50)),

    //The thumb gets reopened
    //rotateThumbProperty: initial position at initial time
    new KeyFrame(new Duration(endTime2),
        //Permanent condition
        new KeyValue(rotateThumbProperty, 50)),
    //rotateThumbProperty: final position at final time
    new KeyFrame(new Duration(endTime3),
        new KeyValue(rotateThumbProperty, 0)),

    //The hand get reopened
```

```
//closeHandProperty: initial position at initial time
new KeyFrame(new Duration(endTime3),
    //Permanent condition
    new KeyValue(closeHandProperty, 90)),
//closeHandProperty: final position at final time
new KeyFrame(new Duration(endTime4),
    new KeyValue(closeHandProperty, 0)),

//Fingers get reopened
//openFingersProperty: initial position at initial time
new KeyFrame(new Duration(endTime4),
//Permanent condition
new KeyValue(openFingersProperty, 0)),
//closeHandProperty: final position at final time
new KeyFrame(new Duration(endTime5),
    new KeyValue(openFingersProperty, 15)),

//Fingers get closed
//openFingersProperty: initial position at initial time
new KeyFrame(new Duration(endTime5),
//Permanent condition
new KeyValue(openFingersProperty, 15)),
//closeHandProperty: final position at final time
new KeyFrame(new Duration(endTime6),
    new KeyValue(openFingersProperty, 0)),

//The thumb gets closed
//rotateThumbProperty: initial position at initial time
new KeyFrame(new Duration(endTime6),
    //Permanent condition
    new KeyValue(closeThumbProperty, 55)),
//rotateThumbProperty: final position at final time
new KeyFrame(new Duration(endTime7),
    new KeyValue(closeThumbProperty, 0)),

//The thumb gets reopen
//rotateThumbProperty: initial position at initial time
new KeyFrame(new Duration(endTime7),
    //Permanent condition
    new KeyValue(closeThumbProperty, 0)),
```

```
        //rotateThumbProperty: final position at final time
        new KeyFrame(new Duration(endTime8),
            new KeyValue(closeThumbProperty, 55))

    );




animation.play();

}//End of constructor of class Hand


}//End of inner class Hand



//A finger is an ensemble of phalanges
private class Finger extends Group
{

    Phalanx[] Phalanges    = new Phalanx [nPhalanges];

    //Constructor
    Finger(int nPhalanges)
    {
        for(int i =0; i < nPhalanges; i++)
        {
            Phalanx ph = new Phalanx(31.25,20,40);
            Phalanges[i] = ph;
        }
    }
}//End of inner class Finger



//A phalanx is a JavaFX 3D box
```

```
private class Phalanx extends Group
{
    Box box;

    //Constructor
    Phalanx(double width, int length, int height)
    {
        box = new Box(width, length, height);
        Color c = randomColor();
        PhongMaterial materialFinger
                = new PhongMaterial(c);
        box.setMaterial(materialFinger);
    }
}//End of inner class phalanx

}//End of main class M29 HandOnlyBoxess12
```

33, page 138. We have devised a human-like hand. To help us, we implemented a solution to the following problem: devise a human driven evolutionary process of software development that produces a human hand in JavaFX beginning from simplicity and risen up the ladder of complexity in the most accelerated but realizable pace. Our process had the following steps:

We began with a hand that has all its 5 fingers on the same line and that can bend fingers to form a grip. Next we got a hand that can move its fingers transversally. Next we implemented a hand that can react to being punched with the mouse. We also got another that can grasp things using two fingers as a pair of pliers. Then we found a hand with an opposite thumb, i.e., a hand that has a thumb that is anatomically opposed to the other four fingers, like the foot of a bird. At last we encountered a hand with opposable thumb, which is a plane hand that has a thumb that after a complex movement can get opposed to other fingers to form a grip.

Our work awakes in us a feeling of wonder in regard with such a clever design of the opposable thumb. At the same time, we feel that we can make science, that our work induces an eager desire to compare our solutions with those found in nature and with those proposed by the Evolutionary Theory.

37, page 170. The report reads as follows:

Summary of the fossil record found at district D123. This district is important because it allows us to reconstruct the evolution of the hand. In this regard, and despite the variability of fossils, each one of them fits in one of the following

categories that are located as expected, simpler fossils below complex ones:

1. A paddle that later became the palm. Used to swing much as in a canoe. Swimming increases the possibility to get food.
2. A hand with 4 fingers. This hand could had been used as a hook to hang from branches of trees.
3. A hand with a extra finger that next became a thumb. This is a preadaptation that was preserved by mere chance.
4. The thumb becomes opposable and was rewarded by natural selection because it allowed to grab clubs for both defense and attack.
5. The thumb gets armed with muscles and nerves that allow the fine control that is necessary for the construction of tools to build weapons for hunting.
6. At last, the possibility to separate fingers was enabled, a property that is needed to play the piano. Our ancestors used this property to easily devise accurate tools, such as a flint ax.

47, page 201. The following code chases in three stages the known parameters of a hand made of boxes:

```
//  Program  M47 FossilRecord4

/*  A genetic algorithm synthesizes the known
    parameters of a hand that functions as a hook
    and that is made of boxes:
    all boxes have the same measures but
    the position is specific to each box.
    First stage:
    the measures of the model box are synthesized.
    First stage continued:
    The position of the first box as a phalanx is chasen.
    Second stage:
    The other two phalanges of the first finger is pursued.
    Third stage;
    The other three fingers are synthesized.

    This algorithm is not good to find the target exactly
```

```
    but is  good to find a nice approximation.
*/


package fossilrecord4;
import java.util.Random;

public class FossilRecord4
{
    //Number of chromosomes = evolving strings
    private static final int nIndiv = 200 ;
    //Tolerated error: if distance to target gets less
    //than this, the algorithm stops
    private static final  double  toleratedError = 1.1;
    //Mutation rate per site  per generation
    private static final double mutationRate = 0.05;
    //Mutation rate per chromosome  per generation
    private static final double recRate = 0.1;
    //Chromosomes
    private static String Individual[ ] = new String[nIndiv];
    private static final String Individual2[ ] = new String[nIndiv];
    //Distances to target
    private static final Double Distance[] = new Double[nIndiv];
    //Fitness of each individual
    private static final Double Fitness[] = new Double[nIndiv];
    //The least distance to target
    private static double leastDistance;
    //Actual generation
    private static int generation = 0;
    //Each string encodes for three numbers
    //= three lengths of target box
    private static double d1 = 0;
    private static int int1 = 0;
    private static int int2 = 0;
    private static double d2 = 0;
    private static double d3 = 0;

    private static int lengthOfChr;

    // Turn on of the random generator
```

```
    private static final Random r  = new Random();
    // place of insertion of the decimal period
    //if negative, there is no period
    private static  int InsertionPeriod  = 2;

    //Boolean flags for printing
    private static final boolean print = false;
    private static final boolean printInitialization3 = false;
    private static final boolean printDecode = false;
    private static final boolean printSacelFitnesses = false;
    private static final boolean printReproduction = false;
    private static final boolean printMutation = false;
    private static final boolean printMutation3 = false;
    private static final boolean printRecombination = false;

//Random integer as char
private  static  Character  randomChar()

// This method generates random chars
// that represent digits
{
 return  ( char ) ( r.nextInt( 10 ) + 48  );
}


//Population is initialized.
    //Each chromosome must fit the pattern given by
    //"31.252040" which encodes for numbers
    //"three numbers, say (31.25, 20, 40)
private static void Initialization( )
{
    /* We generate nIndiv individuals (strings)
    9  = 5 + 2 +2 characters long.
    Sequences are completely random */

    for(int i = 0; i <  nIndiv; i++)
    {
        //Initialization in blank.
        Individual[i]="";
```

```
        //An individual is assembled char by char
        for(int j = 0; j< lengthOfChr; j++)
        {
            // We generate a random numeric char
            Character c =  randomChar();
            //System.out.println( "char = " + c);
            // Char c is converted into a string s
            String s = "" + c;
            // s is concatenated to Individual[i]
            Individual[i] = Individual[i]+s;

        }
        if (InsertionPeriod > 0)
        //A period is inserted at place InsertionPeriod
        Individual[i]
            = Individual[i].substring(0, InsertionPeriod)
                 + "." + Individual[i].substring(InsertionPeriod);
    }
    //Test:
    //Individual[0] = "31.252040";
}

//Population is initialized.
//Each chromosome must fit the pattern given by three
//numbers of type double with their signs
//(+15.625, -15.625, -46.875)

private static void Initialization3( )
{


    for(int i = 0; i <  nIndiv; i++)
    {
        //Initialization in blank.
        Individual[i]="";
        String s = "";
        //Three numbers of type double are composed
        for(int j = 0; j < 3; j++)
        {
            s = "";
```

```
            //Sign is invented
            int l = r.nextInt(2);
            if ( l == 0) s = "+";
            else  s = "-";


            //A numeric substring is assembled char by char
            for(int k = 0; k <  5; k++)
            {
                // We generate a random numeric char
                Character c =  randomChar();
                //System.out.println( "char = " + c);
                // Char c is converted into a string s
                s = s + c;
                // s is concatenated to Individual[i]
            }

            //A period is inserted at place 2,
            s  = s.substring(0, 3)
                     + "." + s.substring(3);
            Individual[i] = Individual[i] + s;
        }
        if (printInitialization3)
            System.out.println(Individual[i]);
    }
}


// This method transforms each string into
//three  numbers, one double and two ints.
//The incoming string is the genotype.
//The set with three numbers is the phenotype.
//Case: measures of the model box
 private static void decode1(String s)
{
    //Double is retrieved
    String s1 = s.substring(0, 5);
    d1 = Double.parseDouble(s1);
    //First int is retrieved
    String s2 = s.substring(5,7);
```

```
    Double d2 = Double.parseDouble(s2);
    int1 = (int) Math.floor(d2);
    //Second int is retrieved
    String s3 = s.substring(7,9);
    Double d3 = Double.parseDouble(s3);
    int2 = (int) Math.floor(d3);
    if (printDecode)
    {
        System.out.println("string = " + s
        + "\nSubstring s1 = " + s1 + " double1 = " + d1
        + "\nSubstring s2 = " + s2 + " int1 = " + int1
        + "\nSubstring s3 = " + s3 + " int2 = " + int2);
    }
}

// This method transforms each string into
//three  numbers, one double and two ints.
//The incoming string is the genotype.
//The set with three numbers is the phenotype.
//Case: Position of the model box
 private static void decode1c(String s)
{
    //Double is retrieved
    String s1 = s.substring(0, 6);
    d1 = Double.parseDouble(s1);
    //First int is retrieved
    String s2 = s.substring(6,9);
    Double d2 = Double.parseDouble(s2);
    int1 = (int) Math.floor(d2);
    //Second int is retrieved
    String s3 = s.substring(9,11);
    Double d3 = Double.parseDouble(s3);
    int2 = (int) Math.floor(d3);
    if (printDecode)
    {
        System.out.println("string = " + s
        + "\nSubstring s1 = " + s1 + " double1 = " + d1
        + "\nSubstring s2 = " + s2 + " int1 = " + int1
        + "\nSubstring s3 = " + s3 + " int2 = " + int2);
    }
```

```
}

//This method transforms each string into
//three  numbers, one double and two ints.
//The incoming string is the genotype.
//The set with three numbers is the phenotype.
//Case: Position of phalanges 2 and 3 of first finger
 private static void decode2(String s)
{

    //First int is retrieved
    String s2 = s.substring(0,3);
    Double d2 = Double.parseDouble(s2);
    int1 = (int) Math.floor(d2);
    //Second int is retrieved
    String s3 = s.substring(3,6);
    Double d3 = Double.parseDouble(s3);
    int2 = (int) Math.floor(d3);
    if (printDecode)
    {
        System.out.println("string = " + s
        + "\nSubstring s2 = " + s2 + " int1 = " + int1
        + "\nSubstring s3 = " + s3 + " int2 = " + int2);
    }
}

//This method transforms each string into
//three  numbers, one double and two ints.
//The incoming string is the genotype.
//The set with three numbers is the phenotype.
//Case: Position of fingers 2, 3 and 4
 private static void decode3(String s)
{
    //First int is retrieved
    String s1 = s.substring(0,7);
    d1 = Double.parseDouble(s1);
    //First int is retrieved
    String s2 = s.substring(7,14);
    d2 = Double.parseDouble(s2);
```

```
    //Second int is retrieved
    String s3 = s.substring(14,21);
    d3 = Double.parseDouble(s3);

    if (printDecode)
    {
        System.out.println("string = " + s
                + "\nSubstring s1 = " + s1 + " d1 = " + d1
        + "\nSubstring s2 = " + s2 + " d2 = " + d2
        + "\nSubstring s3 = " + s3 + " d3 = " + d3);
    }
}


//The distance function upon chromosome s
//measures the squared euclidean distance
 //of its phenotype = three numbers to
//the target(31.25,20,40).
private static Double distance1(String s)
{
    //d1, int1 and int2 are calculated
    decode1(s);
    //The squared distance to target is measured
    Double distance = (31.25 - d1)*(31.25 - d1)
            + (20 - int1)*(20 - int1)
            + (40 - int2)*(40 - int2);
    return distance;
}

//The distance function upon chromosome s
//measures the squared euclidean distance
 //of its phenotype = three numbers to
//the target(46.875, 360, 20).

private static Double distance1c(String s)
{
    //d1, int1 and int2 are calculated
    decode1c(s);
    //The squared distance to target is measured
    Double distance = (46.875 - d1)*(46.875 - d1)
```

```
            + (360 - int1)*(360 - int1)
            + (20 - int2)*(20 - int2);
    return distance;
}

//The distance function upon chromosome s
//measures the squared euclidean distance
 //of its phenotype = three numbers to
//the target(400, 440).
private static Double distance2(String s)
{
    //int1 and int2 are calculated
    decode2(s);
    //The squared distance to target is measured
    Double distance = 0.0 +
            (400 - int1)*(400 - int1)
            + (440 - int2)*(440 - int2);
    return distance;
}


//The distance function upon chromosome s
//measures the squared euclidean distance
 //of its phenotype = three numbers to
//the target(+15.625, -15.625, -46.875).
private static Double distance3(String s)
{
    //int1 and int2 are calculated
    decode3(s);
    //The squared distance to target is measured
    Double distance = (15.625 - d1)*(15.625 - d1) +
            (-15.625 - d2)*(-15.625 - d2)
            + (-46.875 - d3)*(-46.875 - d3);
    return distance;
}


//Fitness = rescaled distance that takes values
//in within 0(the worst) and 1(the best)
private static void calculateFitness()
```

```
{
    if (printSacelFitnesses)
        System.out.println(  "\n====Fitnesses are rescaled: " +
                   "0(the worst) and 1(the best)");
    //Min and max distances are found
    double maxD = Distance[0];
    double minD = Distance[0];
    double b;
    for(int i = 1; i < nIndiv; i++)
    {
        if ( Distance[i] > maxD ) maxD =  Distance[i];
        if ( Distance[i] < minD )
        {
            minD =  Distance[i];
            leastDistance = Distance[i];
        }
    }
    //Fitness is calculated
    if (minD == maxD) //All   equally fit
    {
      for(int i = 0; i < nIndiv; i++)
            Fitness[i] = 1.0;
    }
    else
    {
        //Distance to target determine fitness:
        //fitness[maxD] = 0;    fitness[minD] = 1;
        for(int i = 0; i < nIndiv; i++)
        {
            b = 1;
            b = b/(minD - maxD);
            double a = -b*maxD;
            Fitness[i] = a + b * Distance[i];
            //Selection is sharpened
            Fitness[i] = Math.pow(Fitness[i],7) ;
        }
    }
    //Report
    if (printSacelFitnesses)
    {
```

```
        System.out.println("Individual, distance and fitness");
        for(int i = 0; i < nIndiv; i++)
        {
            System.out.println(
                 i + "\t" + Individual[i] + "\t" +
             Distance[i] + "\t" + Fitness[i]);
        }
    }
}


//The fitness of each individual is evaluated.
//Case: Measures of model box
private static void assessFitness1()
{
    //Distance to target is calculated
    for(int i = 0; i < nIndiv; i++)
    {
        Distance[i] = distance1(Individual[i]);
    }
    //Fitness is an inversion of the distance,
    //The less far, the more fit
    calculateFitness();
}

//The fitness of each individual is evaluated.
//Case: Position of model box
private static void assessFitness1c()
{
    //Distance to target is calculated
    for(int i = 0; i < nIndiv; i++)
    {
        Distance[i] = distance1c(Individual[i]);
    }
    //Fitness is an inversion of the distance,
    //The less far, the more fit
    calculateFitness();
}

//The fitness of each individual is evaluated.
```

```
//Case: Position of model box
private static void assessFitness2()
{
    //Distance to target is calculated
    for(int i = 0; i < nIndiv; i++)
    {
        Distance[i] = distance2(Individual[i]);
    }
    //Fitness is an inversion of the distance,
    //The less far, the more fit
    calculateFitness();
}

//The fitness of each individual is evaluated.
//Case: Position of model box
private static void assessFitness3()
{
    //Distance to target is calculated
    for(int i = 0; i < nIndiv; i++)
    {
        Distance[i] = distance3(Individual[i]);
    }
    //Fitness is an inversion of the distance,
    //The less far, the more fit
    calculateFitness();
}

//Probability of cloning is proportional to fitness.
private static void Reproduction()
{
    //Numbers of clones of each individual
    int count[] = new int[nIndiv];
    //The probability of reproduction
    //is proportional to fitness.
    //Cumulative distribution
    double cumulate[] = new double[nIndiv];
    cumulate[0] =   Fitness[0];
    for (int i = 1; i < nIndiv; i++)
        cumulate[i]= cumulate[i-1] + Fitness[i];
```

```
if (printReproduction)
{
    System.out.println( "i \tfitness cumulate ");
    for (int i = 0; i < nIndiv; i++)
    {
        System.out.println( i + "\t" + Fitness[i] + " \t "
                + cumulate[i]);
    }
}

//Maximal value
double max = cumulate[nIndiv-1];
//Counter of chosen individuals
int W = 0;

//System.out.println("Generated numbers: ");
for (int i = 0; i < nIndiv;i++)
{
    //A random number less than max is generated
    //with a uniform distribution
    double number = max * r.nextFloat();
    if (printReproduction)
        System.out.println("Random number =  " + number);
    //Probability of cloning is made
    //in proportion to fitness.
    //Is indiv 0 to be chosen?
    if (number < cumulate[0])
    {
        if (printReproduction)
            System.out.println(
                    "chosen individual  = 0 ");
        count[0]++;
        //New population
        Individual2[W] = Individual[0];
        W++;
    }
    //Is worm l to be chosen?
    for (int j=0; j < nIndiv-1;j++)
    {
```

```
            int l = j+1;
            if ( ( cumulate[j] <= number) & (number < cumulate[l]) )
            {
                if (printReproduction)
                    System.out.println(
                        "Chosen individual = " + l);
                count[l]++;
                //New population
                Individual2[W] = Individual[l];
                W++;
            }
        }

    }

   //Output
    if (printReproduction)
    {
    System.out.println("\nEvents and their observed "
            + "and expected frequencies: ");
    int sum1 = 0;
    Double sum2 = 0.0;
    for (int i=0; i < nIndiv;i++)
    {
        System.out.println(i + "\t" + count[i] + "\t"
                +  Fitness[i]*nIndiv/cumulate[nIndiv-1]);
        sum1 = sum1 + count[i];
        sum2 = sum2 + Fitness[i]*nIndiv/cumulate[nIndiv-1];

    }
    System.out.println("Totals  " + sum1 + "\t"
                +  sum2);
    }
    //New population becomes current one
    Individual = Individual2;
}
```

```
//Individuals pass under mutation
//Cases: one, one continued and two
private static void Mutation()
{
    // We consider only one type of mutation:
    // substitution of one char by another one.


    Character c;
     String s;

    for(int i = 0; i < nIndiv; i++)
    {
        if (printMutation)
            System.out.println("Incoming individ " + Individual[i]);
        String v = Individual[i];
        // w = mutated individual.
        String w = "";
        int l = v.length();
        //Mutation char by char according to mutationRate
        for(int k = 0; k < l; k++)
        {
            //Touch not the period
            if ( k == InsertionPeriod) c = Individual[i].charAt(2);
            else
            {
                // Change digit k?
                double  p = r.nextDouble();
                if (p < mutationRate)    c =  randomChar();
                else
                    { //no mutation
                        c = Individual[i].charAt(k);
                    }


            }
            // Char c is converted into a string s
            s = c.toString();
            // s is concatenated to w
            w = w+s;
```

```
        }

        Individual[i] = w;
        if (printMutation)
        System.out.println("Mutated          " + Individual[i]);

    }
}


//Mutation of a string that represents a double
//as -15.625
private static String mutateDouble(String d)
{

    String s = "";

    char c = d.charAt(0);
    String cs = ""+ c;
    double  p = r.nextDouble();
    // Change sign?
    if (p < mutationRate)
    {
        if (cs.contentEquals("-")) cs = "" + '+';
        else cs = "" + '-';

    }
    s = s + cs;
    //Mutation char by char according to mutationRate
    for(int k = 1; k < 7; k++)
    {
        //Touch not the period
        if ( k == 3) c = '.';
        else
        {
            // Change digit k?
            p = r.nextDouble();
            if (p < mutationRate)    c =  randomChar();
            else
                { //no mutation
```

```
                    c = d.charAt(k);
                }


            }

            s = s +c;
        }
        if (printMutation3)
        {
            System.out.println("In =  " + d);
            System.out.println("Out = " + s);
        }
        return s;
    }


    //Individuals pass under mutation
    //Cases: position of fingers 2, 3 and 4.
    private static void Mutation3()
    {
        // We consider only one type of mutation:
        // substitution of one char by another of the same type:
        //Number by number, sign by sign, period by period.


        Character c;


        for(int i = 0; i < nIndiv; i++)
        {
            if (printMutation3)
            {
                System.out.println("Incoming individ "
                        + Individual[i]);
            }
            String v = Individual[i];
            //Three numbers suffer mutation
            // d = substring of number one
            String d = Individual[i].substring(0,7);
```

```
        //Mutant
        String w = mutateDouble(d);
        String mutant = "";
        //Concatenation
        mutant = mutant + w;
        //Number 2
        d = Individual[i].substring(7,14);
        w = mutateDouble(d);
        mutant = mutant + w;
        //Number 3
        d = Individual[i].substring(14,21);
        w = mutateDouble(d);
        mutant = mutant + w;

        Individual[i] = mutant;
        if (printMutation3)
        {
            System.out.println("Mutated          "
                    + Individual[i]);
        }

    }
}

//Individual l is recombined to another
//that is chosen at random
private static String recInd(int l)
{
    int length = Individual[l].length();
    //Random site for recombination
    int site = r.nextInt(length);
    //Take a part from l
    String s = Individual[l].substring(0, site);
    //Partner at random
    int k = r.nextInt(nIndiv);
    //Another part from k
    String t = Individual[k].substring(site);
    //Return concatenation
    return s+t;
}
```

```
//ALl individuals pass under recombination
//according to recRate
private static void Recombination()
{
    for(int i = 0; i < nIndiv; i++)
    {
        if (printRecombination)
            System.out.println("Incoming individ " + Individual[i]);
        // Recombine?
        double  p = r.nextDouble();
        if (p < recRate)
            Individual[i] = recInd(i);
        if (printRecombination)
            System.out.println("Recombinant      " + Individual[i]);
    }
}

//Report of best exponents
private static void report( )
{


    for(int i = 0; i < nIndiv; i++)
    {
       //if (Fitness[i] > 0.999999)
        if   (Distance[i] < 2)
            System.out.println("Gen = " + generation +
                "  Chromosome =  " +   Individual[i]
                + " Distance = " + Distance[i]);
    }
}

//One generation of evolution
private static void oneGeneration()
{
    report();
    //The probability of cloning is proportional to fitness
    Reproduction();
    //The new population is subjected to mutation
```

```
    Mutation();
    //The new population is subjected to recombination
    Recombination();
}

//The measures of the model box are chasen:
//(31.25,20,40)
/*
The first measure (x) is encoded
by a subtring with 5 chars with a period included,
the second and third measures (x and y)
by substrings 2 chars long.
*/
private static void  stageOne()
{
    System.out.println("Stage one: Measures of model box");
    System.out.println("Target = (31.25,20,40)");
    generation = 0;
    lengthOfChr = 8;//The dot is inserted a posteriori
    leastDistance = 1000;
    InsertionPeriod = 2;
    Initialization( );
    while ( leastDistance  > toleratedError)
    {
        //Fitness of each individual is assessed
        assessFitness1();
        oneGeneration();
        generation++;
    }
}


//The position of the first phalanx is fitted.
//It is the equivalent of:
/*
    ph00.setTranslateX(15.625*4 -15.625 = 46.875);
    ph00.setTranslateZ(360 );
    ph00.setTranslateY(20);
*/
//TranslateX is encoded in 6 chars.
```

```
//TranslateY is encoded in 3 chars.
//TranslateZ is encoded in 2 chars.
//Total 11 chars

private static void  stageOneContinued()
{
    System.out.println("Stage one continued:"
            + " position of first phalanx");
    System.out.println("Target = (46.875, 360, 20)");
    generation = 0;
    lengthOfChr = 10; //The period is inserted a posteriori
    leastDistance = 10000;
    InsertionPeriod = 2;
    Initialization( );
    while ( leastDistance  > toleratedError)
    {
        //Fitness of each individual is assessed
        assessFitness1c();
        oneGeneration();
        generation++;
    }
}

//Second and third phalanges of first finger are chasen.
//Measures are all the same.
//TranslateX an TranslateY are
//the same of the first phalanx.
//TranslateZ vary: 400 for phalanx2, 440 for phalanx3
private static void  stageTwo()
{
    System.out.println("Stage two: Z-position of second"
            + " and third pahalanges");
    System.out.println("Target = (400, 440)");
    generation = 0;
    lengthOfChr = 6; //The period is inserted a posteriori
    leastDistance = 10000;
    InsertionPeriod = -1; //No decimal period
    Initialization( );
    while ( leastDistance  > toleratedError)
    {
```

```
        //Fitness of each individual is assessed
        assessFitness2();
        oneGeneration();
        generation++;
    }
}


//Second,  third and fourth fingers.
//An X-translation of first finger is
//enough to create another one.
//For the second finger setTranslateX(15.625*2 -15.625);
//For the third finger setTranslateX(15.625*0 -15.625)
//For thefourth finger setTranslateX(15.625*(-2) -15.625)
private static void  stageThree()
{
    System.out.println("Stage three: "
            + "X-position of other 3 fingers");
    System.out.println("Target = (15.625, -15.625, -46.875)");
    generation = 0;

    leastDistance = 10000;

    Initialization3( );

    while ( leastDistance  > toleratedError)
    {
        //Fitness of each individual is assessed
        assessFitness3();
        report();
        //The probability of cloning is proportional to fitness
        Reproduction();
        //The new population is subjected to mutation
        Mutation3();
        //The new population is subjected to recombination
        Recombination();
        generation++;
    }
}
```

```
//Genetic algorithm to fit a target string
//that represents the measures of a box
public static void main(String[] args)
{



    //Stage one:
    //The measures of the model box are synthesized
    stageOne();
    //Stage one continued:
    //The position of the first phalanx is synthesized
    stageOneContinued();
    //Second and third phalanges
    stageTwo();
    //Second,  third and fourth fingers
    stageThree();

}
}//End of main class M47 FossilRecord4
```

50, page 240. With a mutation rate per site per generation of 0.05; and a recombination rate of 0.1 per chromosome per generation, the next pairs of data for (tolerated error, cost of algorithm) were processed:

```
10.0 38200
 9.0 38000
 7.0 89200
 5.0 48000
 3.0 116000
 1.0 1190800
 0.9 2496800
 0.7 746400
 0.5 1302600
 0.3 1695400
 0.1 1441000
 0.07 1964000
 0.03 3673000
 0.01 7560200
```

```
0.005 7160400
```

I used OpenOffice to make a graphic and an hyperbole of the form $y = 1/x$ appeared. That means that the cost of the algorithm and the tolerated error are in inverse proportional relation. If one takes logarithms, one gets $lny = -lnx$. Thus, natural logarithms were found and package R was used to calculate the coefficients of the regression line:

```
#Univariate linear regression
#Clean memory
rm(list = ls())
#Graphic mode
par(mfrow=c(5,5), pch=16)
#
#Part 1

lnOfError <- c(
2.302585093,
2.1972245773,
1.9459101491,
1.6094379124,
1.0986122887,
0,
-0.1053605157,
-0.3566749439,
-0.6931471806,
-1.2039728043,
-2.302585093,
-2.6592600369,
-3.5065578973,
-4.605170186,
-5.2983173665)

lnOfCost <- c(
10.5505907946,
10.5453414387,
11.3986363186,
10.7789562899,
11.6613454701,
13.9901359081,
```

```
14.7305204699,
13.5230169285,
14.0798728251,
14.3434292592,
14.180847875,
14.4904937679,
15.1165193248,
15.8384082028,
15.7840764033
)

#Regression = linear model that shows
#the dependency of lnOfCost on lnOfError
y<-lm(lnOfCost ~ lnOfError)
#List regression coefficients
y$coefficients
```

The output was:

```
(Intercept)   lnOfError
 12.8618229  -0.6983377
```

Thus, we have that
$lnOfCost = 12.86 - 0.698 lnOfError$
$Cost = e^{12.86}/Error^{0.698}$

This looks like

$Cost = k/\sqrt{Error}$

For values of Error less than 1, $Cost = k/\sqrt{Error} < k/Error$.

The use of the GUI of RKWard of Ubuntu allowed the Author to see that although a linear regression was acceptable, a parabolic repression might do it better:

```
#Polynomic regression or order 2
#Clean memory
rm(list = ls())
#Graphic mode
par(mfrow=c(3,2), pch=16)
```

```
#Data
lnOfError <- c(
2.302585093,
2.1972245773,
1.9459101491,
1.6094379124,
1.0986122887,
0,
-0.1053605157,
-0.3566749439,
-0.6931471806,
-1.2039728043,
-2.302585093,
-2.6592600369,
-3.5065578973,
-4.605170186,
-5.2983173665)

lnOfCost <- c(
10.5505907946,
10.5453414387,
11.3986363186,
10.7789562899,
11.6613454701,
13.9901359081,
14.7305204699,
13.5230169285,
14.0798728251,
14.3434292592,
14.180847875,
14.4904937679,
15.1165193248,
15.8384082028,
15.7840764033
)


//Table with data
tablaxd <- data.frame(lnOfError,lnOfCost)
```

```
#Linear model for the polynomic regresssion or order 2
poli2 <- lm(lnOfCost ~ poly(lnOfError, 2, raw = TRUE), data = tablaxd)
summary(poli2)
```

The out was:

```
Call:
lm(formula = lnOfCost ~ poly(lnOfError, 2, raw = TRUE), data = tablaxd)

Residuals:
     Min       1Q   Median       3Q      Max
-0.77755 -0.37358 -0.01061  0.28297  1.33397

Coefficients:
                                Estimate Std. Error t value Pr(>|t|)
(Intercept)                     13.29708    0.20864  63.732  < 2e-16 ***
poly(lnOfError, 2, raw = TRUE)1 -0.95484    0.10018  -9.531    6e-07 ***
poly(lnOfError, 2, raw = TRUE)2 -0.10134    0.02981  -3.399  0.00527 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.6068 on 12 degrees of freedom
Multiple R-squared:  0.9117,Adjusted R-squared:  0.8969
F-statistic: 61.93 on 2 and 12 DF,  p-value: 4.75e-07
```

This means that we have the following relation between the logarithms of the cost and the tolerated error:

$lnOfCost = 13.297 - 0.955 lnOfError - 0.101 (lnOfError)^2$

taking antilogs, we get:

$$Cost = \frac{e^{13.297}}{Error^{0.955} \times e^{0.101 (lnOfError)^2}}$$

Found relations were next tested with the following Java program:

```
//Program M50 ExplorationOfRegressionModel
//Two regression models are converted into
//maps that can be proven over desired values.

package explorationofregressionmodel;
```

```
public class ExplorationOfRegressionModel
{

    public static void main(String[] args)
    {
        double error = 0.01;
        //Linear model over logarithms
        double cost1 = Math.pow(Math.E, 12.86);
        double den1 = Math.pow(error, 0.698);
        cost1 = cost1/ den1;
        //Order-2 repression  model over logarithms
        double cost2 = Math.pow(Math.E, 13.297);
        double den2 = Math.pow(error, 0.955) *
                Math.pow(Math.E, 0.101
                        * Math.log(error)*Math.log(error));
        cost2 = cost2 /den2;
        System.out.println("cost (linear regression)    = "
                + cost1);
        System.out.println("cost (Regression of order 2) = "
                + cost2);
    }
}//End of main class M50 ExplorationOfRegressionModel
```

Grosso modo, both relations appear to be good. Nevertheless, we have a powerful random component due to what geneticists call drift that is caused by sampling effects. Warning: the second relation is not good for extrapolations beyond studied range of error from 5 to 0.005.

75, page 283. 2D rectangles are used to assemble a brick. All faces are constructed beginning from 2D rectangles

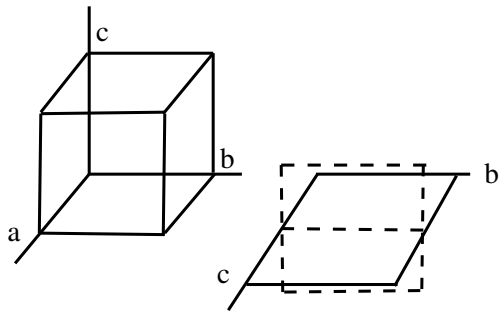


Figure 7.0. A brick, with edges a, b, c, is made from 2D rectangles that are created as rectangles over the floor. The bottom face is just rectangle ab. Suppose now that we want to build face bc. So, we make rectangle cb over the floor, rotate it about an axis parallel to axis b but that passes trough c/2 along the a axis. Next we translate the vertical face backwards c/2 and we rise it c/2: we end with face bc in its right place. And so on.

```
//Program M75 BrickFrom2D
//2D Rectangles are manipulated to
//form a brick that is made into a class, an object.


package brickfrom2d;

import java.lang.reflect.Field;
import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.animation.Animation;
import javafx.animation.KeyFrame;
import javafx.animation.KeyValue;
import javafx.animation.Timeline;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.scene.DepthTest;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.PerspectiveCamera;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;

import javafx.scene.transform.Rotate;
import static javafx.scene.transform.Rotate.X_AXIS;
import static javafx.scene.transform.Rotate.Y_AXIS;
import javafx.scene.transform.Translate;
import javafx.stage.Stage;
```

```
import javafx.util.Duration;


public class BrickFrom2D extends Application
{

    //Animation is declared
    private Timeline animation;
    //An array with all colors that are named in JavaFX
    private static final Color JavaFXColors[] = new Color[148];
    //Generator of pseudo random numbers
    private static final Random rd = new Random();
    //Duration of a cycle of the animation
    private static final int endTime = 5;
    //Width of window
    private static final int width  = 800;
    //Width of window
    private static final int height  = 400;

private void init(Stage primaryStage)
{
    getColors();
    Group root = new Group();
    //Depth = z coordinate.
    //Useful for camera and memory managment.
    root.setDepthTest(DepthTest.ENABLE);
    //Window cannot be rezised (centered would be distorted)
    primaryStage.setResizable(false);
    //Measures of window
    primaryStage.setScene(new Scene(root, width, height, true));
    //Camera is set up:
    //Parallel lines along Z converge to a point.
    primaryStage.getScene().setCamera(new PerspectiveCamera());
    //The stage is centered and rotated
    root.getTransforms().addAll
    (
        new Translate(width / 2, height / 2),
        new Rotate(180, Rotate.X_AXIS)
    );
    root.getChildren().add(create3dContent());
```

```
}

//Initialization and content of the 3Danimation
public Node create3dContent()
{
    //Declaration and initialization.
    //Brick b is at the center
    Brick b = new Brick(40,30,20);
    //Initial rotation
    b.rx.setAngle(45);
    b.ry.setAngle(45);
    //Brick b2 is to the right
    Brick b2 = new Brick(30,80,90);
    b2.setTranslateX(200);
    b2.rx.setAngle(45);
    b2.ry.setAngle(45);
    //Brick b3 is to the left
    Brick b3 = new Brick(30,70,20);
    b3.setTranslateX(-200);
    b3.rx.setAngle(45);
    b3.ry.setAngle(45);
    //Animation is instantiated
    animation = new Timeline();
    //Animation is given a procedure
    animation.getKeyFrames().addAll
    (
        new KeyFrame
        (
           //At start all objects are given an initial position
            Duration.ZERO,
            new KeyValue(b.ry.angleProperty(), 0d),
            new KeyValue(b2.rx.angleProperty(), 0d),
            new KeyValue(b3.rz.angleProperty(), 0d)
        ),
        new KeyFrame
        (
            //At endTime all objects are given a final position
            Duration.seconds(endTime),
            new KeyValue(b.ry.angleProperty(), 360d),
            new KeyValue(b2.rx.angleProperty(), 360d),
```

```
            new KeyValue(b3.rz.angleProperty(), 360d)
        )
    );
    animation.setCycleCount(Animation.INDEFINITE);

    return new Group(b,b2,b3);
}

public void play()
{
    animation.play();
}

@Override public void stop()
{
    animation.pause();
}

private class Brick extends Group
{
//Rotations are declared and initialized
final Rotate rx = new Rotate(0,Rotate.X_AXIS);
final Rotate ry = new Rotate(0,Rotate.Y_AXIS);
final Rotate rz = new Rotate(0,Rotate.Z_AXIS);
private Brick(double a, double b, double c)
{
    getTransforms().addAll(rz, ry, rx);

    //The Author imagined a world as follows:
    //the floor is X-Y and Z is along
    //the normal direction.
    //JavaFX draws objects according to its
    //default orientation of space:
    //X: horizontal axis of the screen
    //Y: vertical axis, positive to the right
    //Z: perpendicular to the screen, positive inwards.

    //World according to the mind of the Author:
    //This face is on the floor: Z = 0.
    Rectangle bottomFace = new Rectangle (a,b);
```

```
bottomFace.setFill(randomColor());
getChildren().add(bottomFace);

//BottomFace is rised c units in the Z-direction
Rectangle lidFace = new Rectangle (a,b);
lidFace.setFill(randomColor());
lidFace.setTranslateZ(c);
getChildren().add(lidFace);

//Rectangle (a,c) is on the floor.
//It is rotated 90 degrees
//around an axis that is parallel to the X-axis
//and that passes for its middle point
//to have it vertical.
//It is then traslated to its correct place.
Rectangle backFace = new Rectangle (a,c);
backFace.setFill(randomColor());
backFace.setRotationAxis(X_AXIS);
backFace.setRotate(90);
backFace.setTranslateY(-c/2);
backFace.setTranslateZ(c/2);
getChildren().add(backFace);

//BackFace is translated b units in the Y-direction
Rectangle frontFace = new Rectangle (a,c);
frontFace.setFill(randomColor());
frontFace.setRotationAxis(X_AXIS);
frontFace.setRotate(90);
frontFace.setTranslateY(-c/2+b);
frontFace.setTranslateZ(c/2);
getChildren().add(frontFace);

//Rectangle (c,b) is on the floor.
//It is rotated 90 degrees
//around an axis that is parallel to the Y-axis
//and that passes for its middle point
//to have it vertical.
//It is then traslated to its correct place.
Rectangle rightFace = new Rectangle (c,b);
rightFace.setFill(randomColor());
```

```
    rightFace.setRotationAxis(Y_AXIS);
    rightFace.setRotate(90);
    rightFace.setTranslateX(-c/2);
    rightFace.setTranslateZ(c/2);
    getChildren().add(rightFace);

    //RightFace is translated a units in the X-direction
    Rectangle leftFace = new Rectangle (c,b);
    leftFace.setFill(randomColor());
    leftFace.setRotationAxis(Y_AXIS);
    leftFace.setRotate(90);
    leftFace.setTranslateX(-c/2+a);
    leftFace.setTranslateZ(c/2);
    getChildren().add(leftFace);

}
}

@Override public void start(Stage primaryStage) throws Exception {
    init(primaryStage);
    primaryStage.show();
    play();
}


 //A random JavaFX color is returned
 private static Color randomColor()
 {
    //A JavaFX named color is picked
    //but the first, Transparent, is ignored
    int n = rd.nextInt(JavaFXColors.length-1)+1;
    Color c = JavaFXColors[n];
    //Some light colors are not visualized so,
    //they might be darkened:
    //c = c.darker();
    return c;
 }

 //Colors with a name in JavaFX are put into an array
static private Color[] getColors()
```

```
{
    //Colors with a name in JavaFX
    final Field[] fields = Color.class.getFields();


    int N = 0;
    //Colors are put into an array
    for (final Field field : fields)
    {
        if (field.getType() == Color.class)
        {
            try
            {
                final Color color = (Color) field.get(null);
                //Array to keep colors
                JavaFXColors[N] = color;
                /*
                //The name of the color
                final String colorName = field.getName();
                System.out.println("N = " + N + "  "
                        + color.toString() + " " + colorName);
                */
                N++;
            } catch (IllegalArgumentException
                    | IllegalAccessException ex)
            {
                Logger.getLogger(BrickFrom2D.class.getName())
                        .log(Level.SEVERE, null, ex);
            }

       }
    }
    return JavaFXColors;
}

public static void main(String[] args)
{
    launch(args);
}
}//End of main class M75 BrickFrom2D
```

76, page 283. 2D rectangles are used to assemble a pyramid.
The following figure shows some important relations:

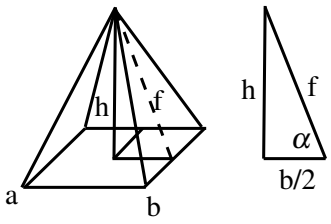


Figure 7.1. Because we are dealing with right triangles, we get $f = \sqrt{h^2 + \frac{a^2}{4}}$. *Moreover, the angle* α *that forms face bf is such that its sine is* h/f.

Suppose now that we want to build the back face, which is opposite to bf.

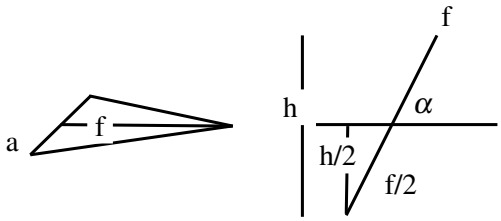


Figure 7.2. To build the back face, determined by the origin and vertex a, we begin with a triangle in the floor, its basis is a and hight f. After a rotation around an axis that is parallel to axis a and that passes by f/2, the face is underwater $h/2$ *while its b-coordinate is* $f/2$. *So, to get this face to its right place, we must translate it back* $f/2$ *along the b-axis and we must rise it* $h/2$.

We begin with a 2D triangle over the floor. Next, we rotate it the angle α around an axis that is parallel to the axis determined by vertex a, but that passes by the middle point of f. Next, we must translate it appropriately: $f/2$ back along the b-axis and we must rise it $h/2$ to get it in place. The complete code follows:

```
//Program M76 PyramidFrom2D
//2D polygons are manipulated to
//form a pyramid that is made into a class, an object.

//The base is a rectangle.
//Each lateral face is a triangle.
//Faces are declared at the origen,
//they are rotated and then translated
//to get them in place.


package pyramidfrom2d;

import java.lang.reflect.Field;
import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.animation.Animation;
import javafx.animation.KeyFrame;
import javafx.animation.KeyValue;
import javafx.animation.Timeline;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.scene.DepthTest;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.PerspectiveCamera;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Polygon;
import javafx.scene.shape.Rectangle;
import javafx.scene.transform.Rotate;
import static javafx.scene.transform.Rotate.X_AXIS;
import static javafx.scene.transform.Rotate.Y_AXIS;
import javafx.scene.transform.Translate;
import javafx.stage.Stage;
import javafx.util.Duration;


public class PyramidFrom2D extends Application {
```

```
    //Animation is declared
    private Timeline animation;
    //An array with all colors that are named in JavaFX
    private static final Color JavaFXColors[] = new Color[148];
    //Generator of pseudo random numbers
    private static final Random rd = new Random();
    //Duration of a cycle of the animation
    private static final int endTime = 5;
    //Width of window
    private static final int width  = 800;
    //Width of window
    private static final int height  = 400;

private void init(Stage primaryStage)
{
    getColors();
    Group root = new Group();
    //Depth = z coordinate.
    //Useful for camera and memory managment.
    root.setDepthTest(DepthTest.ENABLE);
    //Window cannot be rezised (centered would be distorted)
    primaryStage.setResizable(false);
    //Measures of window
    primaryStage.setScene(new Scene(root, width, height, true));
    //Camera is set up:
    //Parallel lines along Z converge to a point.
    primaryStage.getScene().setCamera(new PerspectiveCamera());
    //The stage is centered and rotated
    root.getTransforms().addAll
    (
        new Translate(width / 2, height / 2),
        new Rotate(180, Rotate.X_AXIS)
    );
    root.getChildren().add(create3dContent());
}

//Initialization and content of the 3Danimation
private Node create3dContent()
{
```

```
//The base of the pyramid s a rectangle (a,b)
double a = 60;
double b = 20;
//This is the hight of the pyramid
double h = 50;
//Pyramid p is at the center
Pyramid p = new Pyramid(a,a,h);
p.rx.setAngle(45);
p.ry.setAngle(45);
//Pyramid p2 is to the right
Pyramid p2 = new Pyramid(a,b,h);
p2.setTranslateX(200);
p2.rx.setAngle(45);
p2.ry.setAngle(45);
//Pyramid p3 is to the left
Pyramid p3 = new Pyramid(b,b,h);
p3.setTranslateX(-200);
p3.rx.setAngle(45);
p3.ry.setAngle(45);

//Animation is instantiated
animation = new Timeline();
//Animation is given a procedure
animation.getKeyFrames().addAll
(
    //At start all objects are given an initial position
    new KeyFrame(Duration.ZERO,
        new KeyValue(p.ry.angleProperty(), 0d),
        new KeyValue(p2.rx.angleProperty(), 0d),
        new KeyValue(p3.rz.angleProperty(), 0d)
    ),
    //At endTime all objects are given a final position
    new KeyFrame(Duration.seconds(5),
        new KeyValue(p.ry.angleProperty(), 360d),
        new KeyValue(p2.rx.angleProperty(), 360d),
        new KeyValue(p3.rz.angleProperty(), 360d)
    )
);
animation.setCycleCount(Animation.INDEFINITE);
//animation.setCycleCount(2);
```

```
    return new Group(p,p2,p3);
}

public void play()
{
    animation.play();
}

@Override public void stop()
{
    animation.pause();
}


 //A random color is synthesized
 private static Color randomColor()
 {
    //A JavaFX named color is picked
    //but the first, Transparent, is ignored
    int n = rd.nextInt(JavaFXColors.length-1)+1;
    Color c = JavaFXColors[n];
    //Some light colors are not visualized so,
    //they might be darkened:
    //c = c.darker();
    return c;
 }

 //Colors with a name in JavaFX are put into an array
static private Color[] getColors()
{
    //Colors with a name in JavaFX
    final Field[] fields = Color.class.getFields();


    int N = 0;
    //Colors are put into an array
    for (final Field field : fields)
    {
        if (field.getType() == Color.class)
        {
```

```
            try
            {
                final Color color = (Color) field.get(null);
                //Array to keep colors
                JavaFXColors[N] = color;
                /*
                //The name of the color
                final String colorName = field.getName();
                System.out.println("N = " + N + "  "
                        + color.toString() + " " + colorName);
                */
                N++;
            } catch (IllegalArgumentException
                    | IllegalAccessException ex)
            {
                Logger.getLogger(PyramidFrom2D.class.getName())
                        .log(Level.SEVERE, null, ex);
            }

        }
    }
    return JavaFXColors;
}

private class Pyramid extends Group
{
//Rotations are declared and initialized
final Rotate rx = new Rotate(0,Rotate.X_AXIS);
final Rotate ry = new Rotate(0,Rotate.Y_AXIS);
final Rotate rz = new Rotate(0,Rotate.Z_AXIS);

private Pyramid(double a, double b, double h)
{
    getTransforms().addAll(rz, ry, rx);

    //The Author imagined a world as follows:
    //the floor is X-Y and Z is along
    //the normal direction.
    //JavaFX draws objects according to its
    //default orientation of space:
```

```
//X: horizontal axis of the screen
//Y: vertical axis, positive to the right
//Z: perpendicular to the screen, positive inwards.

//World according to the mind of the Author:
//A pyramid with rectangular base.
//This face is on the floor: Z = 0.
Rectangle bottomFace = new Rectangle(a, b);
bottomFace.setFill(randomColor());

//Height of triangle that will become a face
double e = Math.sqrt(h*h + a*a/4);
//Other hight of another triangle
double f = Math.sqrt(h*h + b*b/4);


//BackFace
Polygon backFace = new Polygon();
//A 2D triangle is a polygon with three vertexes
backFace.getPoints().addAll(new Double[]{
0.0, 0.0, //x,y coordinates of vertix 1
a, 0.0, //x,y coordinates of vertix 2
a/2, f }); //x,y coordinates of vertix 3
backFace.setFill(randomColor());
backFace.setRotationAxis(X_AXIS);
//Angle of rotation for backFace
double angleBackFace =
        Math.asin(h/f) ;
backFace.setRotate(180 * angleBackFace/Math.PI);


backFace.setTranslateY(-f/2 );
backFace.setTranslateZ(h/2);


//FrontFace
Polygon frontFace = new Polygon();
//A triangle is a polygon with three vertexes
frontFace.getPoints().addAll(new Double[]{
0.0, 0.0, //x,y coordinates of vertix 1
```

```
a, 0.0, //x,y coordinates of vertix 2
a/2, f }); //x,y coordinates of vertix 3
frontFace.setFill(randomColor());
frontFace.setRotationAxis(X_AXIS);

frontFace.setRotate(180-180 * angleBackFace/Math.PI);


frontFace.setTranslateY(-f/2 + b);
frontFace.setTranslateZ(h/2);




Polygon rightFace = new Polygon();
rightFace.getPoints().addAll(new Double[]{
0.0, 0.0,
0.0, b,
 e, b/2  });
rightFace.setFill(randomColor());
rightFace.setRotationAxis(Y_AXIS);

//Angle of rotation for ightFace
double angleRightFace =
        Math.asin(-h/e) ;
rightFace.setRotate(180 * angleRightFace/Math.PI);


rightFace.setTranslateX(-e/2 );
rightFace.setTranslateZ(h/2);



    Polygon leftFace = new Polygon();
leftFace.getPoints().addAll(new Double[]{
0.0, 0.0,
0.0, b,
 e, b/2  });
leftFace.setFill(randomColor());
```

```
    leftFace.setRotationAxis(Y_AXIS);


    leftFace.setRotate(180- 180 * angleRightFace/Math.PI);


    leftFace.setTranslateX(-e/2 + a);
    leftFace.setTranslateZ(h/2);

    getChildren().addAll(bottomFace, backFace
            , frontFace, rightFace, leftFace);
    //getChildren().addAll(base, face1, face2, face3, face4);

}
}

@Override public void start(Stage primaryStage) throws Exception {
    init(primaryStage);
    primaryStage.show();
    play();
}


public static void main(String[] args) {
    launch(args);
}
}//End of main class M76 PyramidFrom2D
```

This exercise has been very difficult for the Author, and he does not know where this strange pyramid comes from. The conclusion is that this strategy, 3D from 2D, cannot be used for great projects. Something else must be invented.

95, page 332. This is how we see the brick:

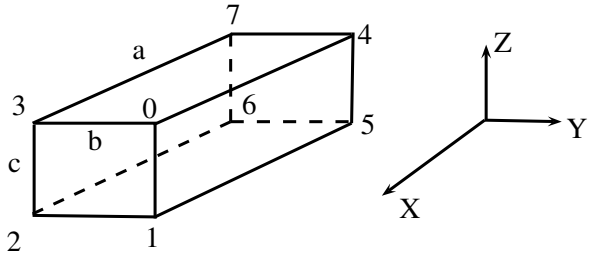


Figure 7.3. We see a brick with measures a (x-coordinate), b (y-coordinate) and c (z-coordinate). The name of a vertex will be its place in the list of vertexes. If the center of coordinates coincides with that of the box, the coordinates of point 0 will be (a/2,b/2,c/2), those of vertex 3 will be (a/2,-b/2, -c/2), those of vertex 4 will be (-a/2, b/2, c/2) and so on.

We unfold the brick:

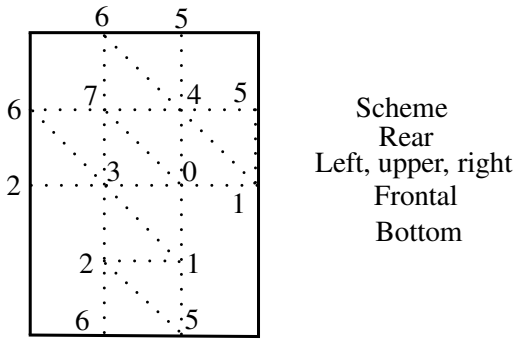


Figure 7.4. The unfolded brick with a triangular mesh. The vertexes encode the form to wind this 2D figure into the 3D brick. Nevertheless an incertitude remains: shall we wind this figure up or down? Answer: You must wind this figure to reproduce the attached scheme, which pictures the view of the 3D shape from the outside. Therefore, vertexes must be listed counterclockwise for each triangle.

Now, we dealt with the unite square that represents the wallpaper. We mark the points that we want to assign to the vertexes of the 3D shape. Our correspondence is natural but many variants exist, say, one can assign the very same triangle in the square to several triangles in the mesh.

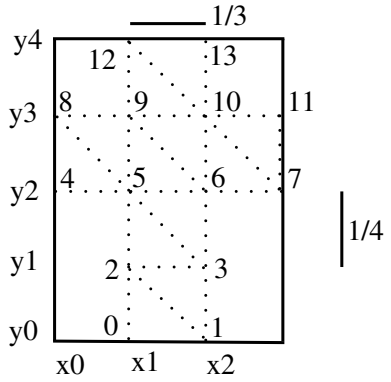


Figure 7.5. The unfolded brick is stretched to cover the unitary square. Equivalently, the wallpaper is coordinated as a square. Points corresponding to the vertexes of the brick are numbered. Point number 3 has coordinates (x2,y1) = (2/3, 1/4). Point 3 of the square is associated to vertex 1 of the brick. And so on.

The corresponding code follows:

```
//Program M95 BrickAsObject
//A brick, as a Java class, is built from a mesh

package brickasobject;

import javafx.animation.Animation;
import javafx.animation.KeyFrame;
import javafx.animation.KeyValue;
import javafx.animation.Timeline;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.scene.DepthTest;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.PerspectiveCamera;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.MeshView;
```

```
import javafx.scene.shape.TriangleMesh;
import javafx.scene.transform.Rotate;
import javafx.scene.transform.Translate;
import javafx.stage.Stage;
import javafx.util.Duration;


public class BrickAsObject extends Application
{

    private Timeline animation;

private void init(Stage primaryStage)
{
    Group root = new Group();
    root.setDepthTest(DepthTest.ENABLE);
    primaryStage.setResizable(false);
    primaryStage.setScene(new Scene(root, 400, 150, true));
    primaryStage.getScene().setCamera(new PerspectiveCamera());
    root.getTransforms().addAll
    (
        new Translate(400 / 2, 150 / 2),
        new Rotate(180, Rotate.X_AXIS)
    );
    root.getChildren().add(create3dContent());
}


//Manager of bricks
public Node create3dContent()
{
    //Central brick
    Brick b = new Brick(30,30,30, "UnfoldedBrick.png");
    b.rx.setAngle(45);
    b.ry.setAngle(45);
    //Brick at t he right
    Brick b2 = new Brick(30,20,30, "Beijing.png");
    b2.setTranslateX(100);
    b2.rx.setAngle(45);
```

```
    b2.ry.setAngle(45);
    //Brick at the left
    Brick b3 = new Brick(40,100,20, "Camping.JPG");
    b3.setTranslateX(-100);
    b3.rx.setAngle(45);
    b3.ry.setAngle(-45);

    //Bricks get rotated
    animation = new Timeline();
    animation.getKeyFrames().addAll
    (
        new KeyFrame//Initial states at initial time
        (
            Duration.ZERO,
            new KeyValue(b.ry.angleProperty(), 0d),
            new KeyValue(b2.rx.angleProperty(), 0d),
            new KeyValue(b3.rz.angleProperty(), 0d)
        ),
        new KeyFrame//Final states at final time
        (
            Duration.seconds(5),
            new KeyValue(b.ry.angleProperty(), 360d),
            new KeyValue(b2.rx.angleProperty(), 360d),
            new KeyValue(b3.rz.angleProperty(), 360d)
        )
    );
    animation.setCycleCount(Animation.INDEFINITE);

    return new Group(b,b2,b3);
}

public void play()
{
    animation.play();
}

@Override public void stop()
{
    animation.pause();
}
```

```
public class Brick extends Group
{
final Rotate rx = new Rotate(0,Rotate.X_AXIS);
final Rotate ry = new Rotate(0,Rotate.Y_AXIS);
final Rotate rz = new Rotate(0,Rotate.Z_AXIS);
public  Brick(float a, float b, float c, String image)
  {
    getTransforms().addAll(rz, ry, rx);
    if ((a * b * c == 0) || (a < 0) || (b < 0) ||(c< 0))
    {
      return;
    }

    //Halves of measures.
    //Type double is not accepted,
    //instead float or int are allowed
    float ha = a / 2;
    float hb = b / 2;
    float hc = c / 2f; //f stands for type float (optional)



    //The mesh is declared and instantiated
    TriangleMesh mesh = new TriangleMesh();
    //Declaration of the points of the mesh.
    //Each point has 3 coordinates.
    //There is neither a tag for  points
    //or a special separator in within points!
    mesh.getPoints().addAll
        (
            ha, hb, hc, //point 0
            ha, hb, -hc, //point 1
            ha, -hb, -hc, //point 2
            ha, -hb, hc, //point 3
            -ha, hb, hc, //point 4
            -ha, hb, -hc, //point 5
            -ha, -hb, -hc, //point 6
            -ha, -hb, hc //point 7
        );
```

```
//The brick will be covered by wallpaper,
//which is marked with points.
//Each point of the wallpaper has two coordinates.
//Coordinates go from 0 to 1 in both dimensions.
//So, wallpaper might be a rectangular image
//but it is seen here  as a square.

//Coordinates for the marking points of wallpaper.
//These points serve to form triangles
//that are next used to cover triangles of the mesh.
float x0 = 0f;
float x1 = 1f / 3f;//this means x1 = 1/4 but with decimals
float x2 = 2f / 3f;
float x3 = 3f / 3f;

float y0 = 0f;
float y1 = 1f / 4f;
float y2 = 2f / 4f;
float y3 = 3f / 4f;
float y4 = 1f;


/*
float y0 = 1f;
float y1 = 3f / 4f;
float y2 = 2f / 4f;
float y3 = 1f / 4f;
float y4 = 0f;
*/


//Points in wallpaper are distingushed by the order
//as they appear in the following list.
//Say, point 0 has coordinates (x1,y0)
//point 3 has coordinates (x1,y1) and so on.
//Order is the only separator of points.
mesh.getTexCoords().addAll(
  x1, y0, //Point 0 of wallpaper
  x2, y0,
```

```
  x1, y1,
  x2, y1,//Point 3 of wallpaper
  x0, y2,
  x1, y2,
  x2, y2,
  x3, y2,
  x0, y3,
  x1, y3,
  x2, y3,
  x3, y3,
  x1,y4,
  x2,y4//Point 13 of wallpaper

);

/* List of triangles:

//We have 12 triangles, two for each face of the box.
//Each triangle is defined by a triade of points:
5,6,4, //Rear
7,4,6,
7,6,3,//Left
2,3,6,
4,7,0,//Upper
3,0,7,
5,4,1, //Right
0,1,4,
0,3,1, //Front
2,1,3,
1,2,5, //Bottom
6,5,2


*/

//The official form as  the list of triangles
//must be declared for a mesh in JavaFX
//demands additional information in regard
//as the form as the covering wallpaper
//must be glued to the brick. To that aim,
```

```
//after listing a vertex of the brick,
//a vertex of the wallpaper is also attached.
mesh.getFaces().addAll(
//Rear face with vertexes 5, 6, 4
//must be covered with triangle
//13, 12, 10 of the wallpaper,
//in the cited order.
5, 13, 6, 12, 4, 10, //Rear
7, 9, 4, 10, 6, 12,
7, 9, 6, 8, 3, 5, //Left
2, 4, 3, 5, 6, 8,
4, 10, 7, 9, 0, 6, //Upper
3, 5, 0, 6, 7, 9,
5, 11, 4, 10, 1, 7,//Right
0, 6, 1, 7, 4, 10,
0, 6, 3, 5, 1, 3, //Front
2, 2, 1, 3, 3, 5,
1, 3, 2, 2, 5, 1,//Bottom
6, 0, 5, 1, 2, 2
);



//The following grouping renders the
//triangles of same face
//with one normal:
//the sun is reflected homogeneously
//along their two triangles.
int smoothingGroups[]
        = { 0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 5, 5};


/*
//This grouping renders triangles
//of the same face of the brick
//with slightly different normals:
//the sun is reflected inhomogeneously.
int smoothingGroups[]
        = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11};
```

```
    */

    mesh.getFaceSmoothingGroups().addAll(smoothingGroups);

    MeshView brick =
      new MeshView(mesh);

    //Wallpaper is gotten
    Image wallpaper = new Image(getClass()
            .getResourceAsStream(image));
    //Declaration of material for painting the brick
    PhongMaterial material = new PhongMaterial();
    //Sun
    material.setSpecularColor(Color.WHEAT);
    //Wallpaper is declared as covering material
    material.setDiffuseMap(wallpaper);
    //Brick is covered with wallpaper
    brick.setMaterial(material);
    //The brick as JavaFX spacelike object
    final Group parent = new Group(brick);
    getChildren().addAll(parent);
  }
}

@Override public void start(Stage primaryStage)
        throws Exception
{
    init(primaryStage);
    primaryStage.show();
    play();
}


public static void main(String[] args)
{
    launch(args);
}
}//End of main class M95 BrickAsObject
```

96, page 332. We choose to build a pyramid with a rectangular basis. This is how we see it:

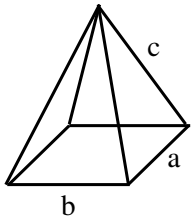


Figure 7.6. The pyramid has a rectangular basis and all its obliques edges have the same length, c.

The unfolded pyramid looks like this:

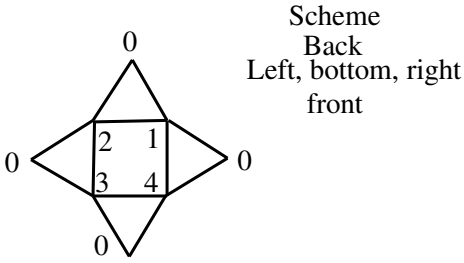


Figure 7.7. The unfolded pyramid as a star. In real scale it is irregular, but we impose a system of coordinates from which it looks regular. To reproduce the scheme, bend the arms of the star upwards. Therefore, vertexes must be listed clockwise. To see this, imagine you bending the frontal face with vertexes 3, 4 and 0. After bending, a counterclockwise listing will be 4, 0, 3 that corresponds with a clockwise listing over the shown figure.

We must superpose now the unfolded pyramid over the unitary square that represents the wallpaper:

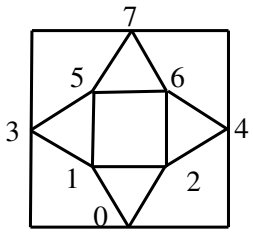


Figure 7.8. The unitary square that represents the wallpaper with marked points that match the vertexes of the unfolded pyramid.

We can put now all this into the code:

```
//Program M96 PyramidAsObject
//A pyramid, as a Java class, is built from a mesh.

package pyramidasobject;

import javafx.animation.Animation;
import javafx.animation.KeyFrame;
import javafx.animation.KeyValue;
import javafx.animation.Timeline;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.beans.property.BooleanProperty;
import javafx.beans.property.SimpleBooleanProperty;
import javafx.scene.DepthTest;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.PerspectiveCamera;
import javafx.scene.Scene;
import javafx.scene.control.CheckBox;
import javafx.scene.control.Label;
import javafx.scene.image.Image;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.MeshView;
import javafx.scene.shape.TriangleMesh;
import javafx.scene.transform.Rotate;
import javafx.scene.transform.Translate;
import javafx.stage.Stage;
import javafx.util.Duration;


public class PyramidAsObject extends Application
{
```

```
    private Timeline animation;
    private CheckBox meshCheckBox;
//Show mesh?
private final BooleanProperty showWireframe
        = new SimpleBooleanProperty(false);

private Label facePickedLabel = new Label();


private void init(Stage primaryStage)
{
    Group root = new Group();
    root.setDepthTest(DepthTest.ENABLE);
    primaryStage.setResizable(false);
    primaryStage.setScene(new Scene(root, 800, 400, true));
    primaryStage.getScene().setCamera(new PerspectiveCamera());
    root.getTransforms().addAll
    (
        new Translate(800 / 2, 400 / 2),
        new Rotate(180, Rotate.X_AXIS)
    );
    root.getChildren().add(create3dContent());
}


//Manager of pyramids
public Node create3dContent()
{
    //Central pyramid
    Pyramid p = new Pyramid(200,200,200, "UnfoldedBrick.png");
    p.rx.setAngle(-45);
    p.ry.setAngle(45);
    //Pyramid at the right
    Pyramid p2 = new Pyramid(30,20,70, "Beijing.png");
    p2.setTranslateX(300);
    p2.rx.setAngle(45);
    p2.ry.setAngle(45);
    //Pyramid at the left
    Pyramid p3 = new Pyramid(60,100,50, "Camping.JPG");
```

```
    p3.setTranslateX(-300);
    p3.rx.setAngle(45);
    p3.ry.setAngle(-45);

    //Pyramids get rotated
    animation = new Timeline();
    animation.getKeyFrames().addAll
    (
        new KeyFrame//Initial states at initial time
        (
            Duration.ZERO,
            new KeyValue(p.ry.angleProperty(), 0d),
            new KeyValue(p2.rx.angleProperty(), 0d),
            new KeyValue(p3.rz.angleProperty(), 0d)
        ),
        new KeyFrame//Final states at final time
        (
            Duration.seconds(5),
            new KeyValue(p.ry.angleProperty(), 360d),
            new KeyValue(p2.rx.angleProperty(), 360d),
            new KeyValue(p3.rz.angleProperty(), 360d)
        )
    );
    animation.setCycleCount(Animation.INDEFINITE);

    return new Group(p,p2,p3);
}

public void play()
{
    animation.play();
}

@Override public void stop()
{
    animation.pause();
}

public class Pyramid extends Group
{
```

```
final Rotate rx = new Rotate(0,Rotate.X_AXIS);
final Rotate ry = new Rotate(0,Rotate.Y_AXIS);
final Rotate rz = new Rotate(0,Rotate.Z_AXIS);
public  Pyramid(float a, float b, float c, String image)
  {
    getTransforms().addAll(rz, ry, rx);
    if ((a * b * c == 0) || (a < 0) || (b < 0) ||(c< 0))
    {
      return;
    }

    //Halves of measures.
    //Type double is not accepted,
    //instead float or int are allowed
    float ha = a / 2;
    float hb = b / 2;
    float hc = c / 2f; //f stands for type float (optional)



    //The mesh is declared and instantiated
    TriangleMesh mesh = new TriangleMesh();
    //Declaration of the points of the mesh.
    //Each point has 3 coordinates.
    //There is neither a tag for  points
    //or a special separator in within points!
    mesh.getPoints().addAll
        (
            0, 0, hc, //point 0
            -ha, hb, 0, //point 1
            -ha, -hb, 0, //point 2
            ha, -hb, 0, //point 3
            ha, hb, 0 //point 4
        );

    //The pyramid will be covered by wallpaper,
    //which is marked with points.
    //Each point of the wallpaper has two coordinates.
    //Coordinates go from 0 to 1 in both dimensions.
    //So, wallpaper might be a rectangular image
```

```
//but it is seen here  as a square.

//Coordinates for the marking points of wallpaper.
//These points serve to form triangles
//that are next used to cover triangles of the mesh.
float x0 = 0f;
float x1 = 1f / 4f;//this means x1 = 1/4 but with decimals
float x2 = 2f / 4f;
float x3 = 3f / 4f;
float x4 = 4f / 4f;

float y0 = 0f;
float y1 = 1f / 4f;
float y2 = 2f / 4f;
float y3 = 3f / 4f;




//Points in wallpaper are distingushed by the order
//as they appear in the following list.
//Say, point 0 has coordinates (x1,y0)
//point 3 has coordinates (x2,y0) and so on.
//Order is the only separator of points.
mesh.getTexCoords().addAll(
  x2, y0, //Point 0 of wallpaper
  x1, y1,
  x3, y1,
  x0, y2,//Point 3 of wallpaper
  x4, y2,
  x0, y3,
  x3, y3,
  x2, y2
);

/* List of triangles:

//We have 6 triangles, two for bottom face of the box.
//Each triangle is defined by a triade of points:
```

```
0,  1,  2,  //Rear
2,  3,  0,  //left;
2,  1,  3,  //Bootom
4,  3,  1,
4,  1,  0, //Right
0,  3,  4 //Front


*/

//The official form as  the list of triangles
//must be declared for a mesh in JavaFX
//demands additional information in regard
//as the form as the covering wallpaper
//must be glued to the pyramid. To that aim,
//after listing a vertex of the pyramid,
//a vertex of the wallpaper is also attached.
mesh.getFaces().addAll(
0, 7, 1, 6, 2, 5, //Rear
2, 5, 3, 1, 0, 3, //left;
2, 5, 1, 6, 3, 1, //Bootom
4, 2, 3, 1, 1, 6,
4, 2, 1, 6, 0, 4,//Right
0, 1, 3, 1, 4, 2 //Front
);



//The following grouping renders the
//triangles of same face
//with one normal:
//the sun is reflected homogeneously
//along the two triangles.
int smoothingGroups[]
        = { 0,1,2,3,4,5};
//Add smoothing groups
mesh.getFaceSmoothingGroups().addAll(smoothingGroups);

//Assemble pyramid from mesh
```

```
    MeshView pyramid = new MeshView(mesh);

    //Wallpaper is gotten
    Image wallpaper = new Image(getClass()
            .getResourceAsStream(image));
    //Declaration of material for painting the pyramid
    PhongMaterial material = new PhongMaterial();
    //Sun
    material.setSpecularColor(Color.WHEAT);
    //Wallpaper is declared as covering material
    material.setDiffuseMap(wallpaper);
    //Pyramid is covered with wallpaper
    pyramid.setMaterial(material);
    //The pyramid as JavaFX spacelike object
    final Group parent = new Group(pyramid);

    //Option: see wiring
    //pyramid.drawModeProperty().set(DrawMode.LINE);
    //Booking
    getChildren().addAll(parent);
  }
}

@Override public void start(Stage primaryStage)
        throws Exception
{
    init(primaryStage);
    primaryStage.show();
    play();
}


public static void main(String[] args)
{
    launch(args);
}
}//End of main class M96 PyramidAsObject
```

98, page 348. How to use a program that draws a function f around (0,0) to draw a function g over a region around a point (a,b)? We define $f(x,y) = g(x-a, y-b)$ and use the program to draw f. That is all to it.

112, page 405. The following code opens the requested file. The difference is that in the previously found dialect, the number of a vertex was duplicated in each face, but here we have that faces come with information about texture coordinates. We ignore such information and simply read the vertexes that conform each face.

```
/*
    Program M112 MlgrmLeftHand4
    This program imports a set  of meshes that are encoded
    in a file in the .obj format.
    Content is shown in a textArea and
    encoded meshes are drawn.

    One can select multiple files in the file chooser.


    Tested only with some  files,
    particularly  with those of the BodyParts3D project.
    http://lifesciencedb.jp/bp3d/
    Those files are licensed:

    The license for this database is specified in the
    Creative Commons Attribution-Share Alike 2.1 Japan.
    If you use data from this database, please be sure
    attribute this database as follows: "BodyParts3D, (c)
    The Database Center for Life Science licensed under
    CC Attribution-Share Alike 2.1 Japan".
    http://dbguidehive.biosciencedbc.jp/en/bodyparts3d/lic.html

 */

package mlgrmlefthand4;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
```

```
import java.io.FileReader;
import java.io.IOException;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.event.ActionEvent;
import javafx.scene.Group;
import javafx.scene.PerspectiveCamera;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Menu;
import javafx.scene.control.MenuBar;
import javafx.scene.control.MenuItem;
import javafx.scene.control.ScrollPane;
import javafx.scene.control.TextArea;
import javafx.scene.input.KeyEvent;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.Box;
import javafx.scene.shape.MeshView;;
import javafx.scene.shape.TriangleMesh;
import javafx.scene.transform.Rotate;
import javafx.scene.transform.Scale;
import javafx.scene.transform.Translate;
import javafx.stage.FileChooser;
import javafx.stage.Stage;



public class MlgrmLeftHand4 extends Application
{
    //Visibility of the mesh
    //might depend on the correct choice
    //of this factor. Move it in within 1 and 500.
    private static double scaleFactor = 7;
    //Printing for debugging
```

```
final static boolean printReadFile = false;
private static final String title
        =  "Importer of meshes in format .obj";

//Tool to read files
BufferedReader br ;
private static  String textOfFile = "";

private static int counterOfFaces = 0;
private static int counterOfVertexes = 0;
private static int counterOfNormals = 0;

private static int[] destFaces ;


//*****************

final Group root = new Group();

final Group axisGroup = new Group();

final PerspectiveCamera camera
        = new PerspectiveCamera(true);
final Xform cameraXform = new Xform();
final Xform cameraXform2 = new Xform();
final Xform cameraXform3 = new Xform();

final double cameraDistance = 1450;//450
final Xform world3DXform = new Xform();
final Xform bonesXform = new Xform();



boolean timelinePlaying = false;
double ONE_FRAME = 1.0/24.0;
double DELTA_MULTIPLIER = 200.0;
double CONTROL_MULTIPLIER = 0.1;
double SHIFT_MULTIPLIER = 0.1;
double ALT_MULTIPLIER = 0.5;
```

```
 double mousePosX;
 double mousePosY;
 double mouseOldX;
 double mouseOldY;
 double mouseDeltaX;
 double mouseDeltaY;



final Xform guidesXform = new Xform();
//Menubar declaration and initialization
 MenuBar menuBar = new MenuBar();


  private void buildCamera() {
     root.getChildren().add(cameraXform);
     cameraXform.getChildren().add(cameraXform2);
     cameraXform2.getChildren().add(cameraXform3);
     cameraXform3.getChildren().add(camera);
     cameraXform3.setRotateZ(180.0);

     camera.setNearClip(0.1);
     camera.setFarClip(10000.0);
     camera.setTranslateZ(-cameraDistance);
     cameraXform.ry.setAngle(320.0);
     cameraXform.rx.setAngle(40);
 }



 private void buildAxes() {
     System.out.println("buildAxes()");
     final PhongMaterial redMaterial
             = new PhongMaterial();
     redMaterial.setDiffuseColor(Color.DARKRED);
     redMaterial.setSpecularColor(Color.RED);

     final PhongMaterial greenMaterial
             = new PhongMaterial();
     greenMaterial.setDiffuseColor(Color.DARKGREEN);
```

```
        greenMaterial.setSpecularColor(Color.GREEN);

        final PhongMaterial blueMaterial
                = new PhongMaterial();
        blueMaterial.setDiffuseColor(Color.DARKBLUE);
        blueMaterial.setSpecularColor(Color.BLUE);

        final Box xAxis = new Box(640.0, 1, 1);
        final Box yAxis = new Box(1, 640.0, 1);
        final Box zAxis = new Box(1, 1, 640.0);

        xAxis.setMaterial(redMaterial);
        yAxis.setMaterial(greenMaterial);
        zAxis.setMaterial(blueMaterial);

        axisGroup.getChildren().addAll(xAxis, yAxis, zAxis);
        axisGroup.setVisible(false);
        world3DXform.getChildren().addAll(axisGroup);
    }



//
// The handleCameraViews file contains
//the handleMouse() and handleKeyboard()
// methods that are used in the
//MoleculeSampleApp application to handle the
// different 3D camera views.
//These methods are used in the Getting Started with
// JavaFX 3D Graphics tutorial.
//

private void handleMouse(Scene scene)
{
    scene.setOnMousePressed((MouseEvent me) ->
    {
        mousePosX = me.getSceneX();
        mousePosY = me.getSceneY();
        mouseOldX = me.getSceneX();
        mouseOldY = me.getSceneY();
```

```
});
scene.setOnMouseDragged((MouseEvent me) ->
{
    mouseOldX = mousePosX;
    mouseOldY = mousePosY;
    mousePosX = me.getSceneX();
    mousePosY = me.getSceneY();
    mouseDeltaX = (mousePosX - mouseOldX);
    mouseDeltaY = (mousePosY - mouseOldY);

    double modifier = 1.0;
    double modifierFactor = 0.1;

    if (me.isControlDown()) {
        modifier = 0.1;
    }
    if (me.isShiftDown()) {
        modifier = 10.0;
    }
    if (me.isPrimaryButtonDown()) {
        cameraXform.ry.setAngle(cameraXform
                .ry.getAngle()
                - mouseDeltaX*modifierFactor
                        *modifier*2.0);  // +
        cameraXform.rx.setAngle(cameraXform
                .rx.getAngle()
                + mouseDeltaY*modifierFactor
                        *modifier*2.0);  // -
    }
    else if (me.isSecondaryButtonDown()) {
        double z = camera.getTranslateZ();
        double newZ = z
        + mouseDeltaX*modifierFactor*modifier;
        camera.setTranslateZ(newZ);
    }
    else if (me.isMiddleButtonDown()) {
        cameraXform2.t.setX(cameraXform2.t.getX()
            + mouseDeltaX*modifierFactor*modifier*0.3);
        cameraXform2.t.setY(cameraXform2.t.getY()
            + mouseDeltaY*modifierFactor*modifier*0.3);
```

```
        }
        });
}

private void handleKeyboard(Scene scene)
{

    scene.setOnKeyPressed((KeyEvent event) -> {

        switch (event.getCode()) {
            case A: //Hide axis
                if (event.isControlDown()) {
                    if (axisGroup.isVisible()) {
                        System.out.println(
                                "setVisible(false)");
                        axisGroup.setVisible(false);
                    }
                    else {
                        System.out.println(
                                "setVisible(true)");
                        axisGroup.setVisible(true);
                    }
                }
                break;
        }
    });
}



@Override
public void start(Stage primaryStage)
{
    System.out.println("start");

    Scene scene = new Scene(root, 1200, 1000, true);



    /*
```

```
   Multiple file loader
*/

//Content of files will be shown in textArea
final TextArea textArea = new TextArea();
textArea.prefWidth(400);
textArea.setWrapText(true);
textArea.setText("");
//Container  for textArea.
ScrollPane scrollPane = new ScrollPane();
scrollPane.getStyleClass().add("noborder-scroll-pane");
scrollPane.setContent(textArea);
scrollPane.setFitToWidth(true);
scrollPane.setPrefWidth(400);
scrollPane.setPrefHeight(180);
VBox vBox = new VBox();
vBox.visibleProperty().setValue(false);
vBox.getChildren().addAll(scrollPane);
root.getChildren().addAll(vBox);


// --- Menu files
Menu filesMenu = new Menu("Files");
//Menu to load multiple files
MenuItem loadFiles
        = new MenuItem("Load + process multiple files ");
//MenuItem added to menu
filesMenu.getItems().addAll(loadFiles);
menuBar.getMenus().add(filesMenu);
root.getChildren().add(menuBar);

//Listener
loadFiles.setOnAction((ActionEvent t) ->
{
    FileChooser fileChooser = new FileChooser();

    //Name of file with extension .obj
    FileChooser.ExtensionFilter extFilter
            = new FileChooser
                    .ExtensionFilter("TXT files (*.obj)"
```

```
                                , "*.obj");
        fileChooser.getExtensionFilters().add(extFilter);
        //Files will be kept in a list
        List<File> files = new FileChooser()
                .showOpenMultipleDialog(primaryStage);
        //Each file is processed
        files.stream().forEach((File file) -> { try
        {
            //Text of file is read and mesh is extracted
            textOfFile = readFile(file);
            //Text copied to textArea
            textArea.appendText(textOfFile);

            } catch (IOException ex)
            {
                Logger.getLogger(MlgrmLeftHand4
                        .class.getName())
                        .log(Level.SEVERE, null, ex);
            }
        });

    //To remedy a not understood bug
    //( = menuBar is added to the graphic )
    root.getChildren().remove(menuBar);

    vBox.visibleProperty().set(true);
    });

    //====== End of File loader

    primaryStage.setTitle(title
            + ". Drag the mouse to rotate. "
        + " Ctrl + A = Axis visibility"
            );
    scene.setFill(Color.BLUE);
    primaryStage.setScene(scene);
    primaryStage.show();
    buildAxes();
    handleKeyboard(scene);
    handleMouse(scene);
```

```
    //Hide textArea
    Button hideTextArea = new Button();
    hideTextArea.setText("View graphic");
    //Listener
    hideTextArea.setOnAction((ActionEvent t) ->
    {
        vBox.visibleProperty().set(false);
        buildCamera();
        bonesXform.setScale(scaleFactor);
        //Centering: found by trial and error.
        double xMax = bonesXform.getBoundsInLocal().getMaxX();
        double yMax = bonesXform.getBoundsInLocal().getMaxY();
        double zMax = bonesXform.getBoundsInLocal().getMaxZ();
        double xMin = bonesXform.getBoundsInLocal().getMinX();
        double yMin = bonesXform.getBoundsInLocal().getMinY();
        double zMin = bonesXform.getBoundsInLocal().getMinZ();
        double x = scaleFactor *  0.82 * (xMax + xMin) ;
        double y = scaleFactor * 0.78 * (yMax + yMin);
        double z = scaleFactor * 0.9 * (zMax + zMin) ;
        bonesXform.setTranslate(-x, -y, -z);
        world3DXform.getChildren().add(bonesXform);
        root.getChildren().add(world3DXform);
        scene.setCamera(camera);
    });

    vBox.getChildren().addAll(hideTextArea);
    primaryStage.show();
}



//File is read and its content is returned as a string.
//The associated mesh is recovered
//and correct orientation fo faces is provided.
//The mesh is a global variable.


private String readFile(File file)
```

```
        throws FileNotFoundException, IOException
{
    //Name of file is printed
    String name = file.getName();
    System.out.println(name);
    //Buffer to keep the content of file
    //StringBuilder stringBuffer = new StringBuilder();
    StringBuilder stringBuffer  = new StringBuilder();
    //Tool to read file
    BufferedReader bufferedReader = null;

    //A mesh is declared and instantiated
    TriangleMesh mesh  = new TriangleMesh();
    //The mesh has no predefined texture
    mesh.getTexCoords().addAll(0,0);
    //In the .obj format, counting begins with 1
    //but in Java, we begin with 0.
    //So, an additional inocuos point is added.
    mesh.getPoints().addAll(0,0,0);

    if (printReadFile)
        System.out.println("File content: ");
//File is read
try
{
//
bufferedReader = new BufferedReader(new FileReader(file));

String text;




while( (text = bufferedReader.readLine()) != null)
{
    //Text for the textArea
    stringBuffer.append(text);

    //Vertexes are read
    if (((text.length() > 0 ) && text.charAt(0) == 'v')
```

```
        && text.charAt(1) == ' ')
{
    if (printReadFile)
        System.out.println("Vertex  = "
                + counterOfVertexes + " = "+text);
    counterOfVertexes++;
    text = text.substring(2);
    int j = 0;
    //Three vertexes are read
    for(int i = 0; i < 3; i++)
    {
        double d ;
        String s = "";
        Boolean go = true;

        while (go)
        {
            char thisChar = text.charAt(j);
            s = s + thisChar;
            char nextChar = ' ';
            //++j means add one to j
            //and then evaluate expression with <
            if (++j < text.length())
             nextChar = text.charAt(j);
            if ((nextChar == ' ') ||
                    (j >= text.length()))
                    go = false;

        }
        //string to double
        d = Double.parseDouble(s);
        float df = (float) d;
        if (printReadFile)
            System.out.println("Float number = " + df);
        mesh.getPoints().addAll(df);

    }
}
```

```
//Normal vectors are counted and displayed
if ((text.length() > 0 ) && (text.charAt(0) == 'v')
        && (text.charAt(1) == 'n'))
{

    if (printReadFile)
        System.out.println("Normal " + counterOfNormals
            + " = " + text);
    counterOfNormals++;
}


//Face is read, one face per line.
//A face has 3 or more vertexes. These are extracted.
//Additional information is ignored
if ( (text.length() > 0 ) && (text.charAt(0) == 'f'))
{

    if (printReadFile)
        System.out.println("Face  = " + counterOfFaces
            + " = "+text);
    counterOfFaces++;
    text = text.substring(2);

    int vertex[] = new int[4];
    int j = 0;

    Boolean isNotBlank ;
    //Three vertexes are extracted
    for(int i = 0; i < 3; i++)
    {

        double d ;
        String s = "";
        Boolean go1 = true;


        while (go1)
        {
            if (j >= text.length()-2)
```

```
            break;

        char thisChar = text.charAt(j);
        s = s + thisChar;
        /*
        System.out.println(" j = "
                + j + "    s = " + s);
                */
        char nextChar = ' ';
        j++;
        if (j < text.length())
         nextChar = text.charAt(j);
        char nextNextChar = ' ';
        isNotBlank = true;
        if (nextChar == '/' )
        {
            go1 = false;
            //go until the following blank
            while ((isNotBlank)
                    && (j < text.length()-1))
            {
                j++;
                nextNextChar = text.charAt(j);
                if (nextNextChar == ' ' )
                    isNotBlank = false;

            }
        }

    }

    d = Double.parseDouble(s);
    int v = (int) d;
    vertex[i] = v;
    if (printReadFile)
        System.out.println("Int number = " + v);
}//End of cycle for 3 vertexes

//A face with 3 vertexes is added to mesh
mesh.getFaces().addAll(vertex[0], 0, vertex[1], 0,
```

```
        vertex[2], 0);

    if (printReadFile)
    {
        for(int i = 0; i < 3; i++)
         System.out.print(vertex[i] + " . ");
        System.out.println();
    }

    //Is there a four vertex in the face?
    if   ( (text.length() - j )> 3)
    {
        String s = "";
        Boolean go1 = true;
        while (go1)
        {


            char thisChar = text.charAt(j);
            s = s + thisChar;
            j++;
            char nextChar = ' ';
            if (j < text.length())
             nextChar = text.charAt(j);
            if (nextChar == '/' ) go1 = false;
        }
        if (printReadFile)
            System.out.println("4th = " + s);
        double d = Double.parseDouble(s);
        int v = (int) d;
        vertex[3] = v;

        //A polyface with 4 vertex, say, a,b,c,d,
        //can be covered by 2 triangles:
        //Triangle 1: a,b,c (Already added to mesh)
        //and triangle 2: a,c,d.
        //Triangle 2 with 3 vertexes is added to mesh
        mesh.getFaces().addAll(vertex[0], 0, vertex[2],
                0, vertex[3], 0);
    }
```

```
    }//End of face cycle



}

} //End of  try.
    //Exception management
    catch (FileNotFoundException ex)
    {
        Logger.getLogger(MlgrmLeftHand4
        .class.getName()).log(Level.SEVERE, null, ex);
    }
    catch (IOException ex)
    {
        Logger.getLogger(MlgrmLeftHand4
        .class.getName()).log(Level.SEVERE, null, ex);
    }
    finally
    {
        try
        {
            bufferedReader.close();
        } catch (IOException ex)
        {
            Logger.getLogger(MlgrmLeftHand4
            .class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    if (printReadFile)
    {
        System.out.println("End of file content. ");
        //Other indicators are reported
        report(mesh);
    }


    //======MeshWork=======
```

```
    MeshView boneMesh =
            new MeshView(mesh);

    PhongMaterial goldenMaterial
            = new PhongMaterial(Color.GOLDENROD);
    boneMesh.setMaterial(goldenMaterial);
    Group boneGroup = new Group( boneMesh);
    //Mesh is added to bonesXform
    bonesXform.getChildren().add(boneGroup);


    return stringBuffer.toString();
}
```

```
//Indicators of retrieved data are printed
private void report(TriangleMesh mesh)
{
    float[] destPoints = new float[mesh.getPoints().size()];
    mesh.getPoints().toArray(destPoints);
    System.out.println("POINTS");
    for(int j = 0; j <  destPoints.length/3; j++)
    {
        System.out.print(j + "   " );
        for(int k = 0; k < 3; k++)
        {
            System.out.print( destPoints[j*3 + k] + "   " );
        }
        System.out.println();
    }
```

```
    destFaces = new int[mesh.getFaces().size()];
    //Faces go to a matrix
    mesh.getFaces().toArray(destFaces);
    System.out.println("FACES WITH TEXTURE COORDINATES");
    for(int j = 0; j <  destFaces.length/6; j++)
    {
        System.out.print(j + "   " );
        for(int k = 0; k < 6; k++)
        {
            System.out.print( destFaces[j*6 + k] + "   " );
        }
        System.out.println();
    }

    System.out.println("counterOfPoints = "
            + counterOfVertexes );
    System.out.println("counterOfFaces = "
            + counterOfFaces );
    System.out.println("counterOfNormals = "
            + counterOfNormals);
    System.out.println("mesh.getPoints().size() = "
            + mesh.getPoints().size() );
    System.out.println("mesh.getFaces().size() = "
            + mesh.getFaces().size() );

    System.out.println("getPointElementSize = "
            + mesh.getPointElementSize() );
    System.out.println("getfaceElementSize = "
            + mesh.getFaceElementSize() );

}


public static void main(String[] args)
{

    launch(args);
}

}//End of main class M112 MlgrmLeftHand4
```

```
//Outer class Xform.
//Machinery that enables rotations in vivo of the scene
//and the conformation of groups of control.
class Xform extends Group {

    public enum RotateOrder {
        XYZ, XZY, YXZ, YZX, ZXY, ZYX
    }

    public Translate t  = new Translate();
    public Translate p  = new Translate();
    public Translate ip = new Translate();
    public Rotate rx = new Rotate();
    { rx.setAxis(Rotate.X_AXIS); }
    public Rotate ry = new Rotate();
    { ry.setAxis(Rotate.Y_AXIS); }
    public Rotate rz = new Rotate();
    { rz.setAxis(Rotate.Z_AXIS); }
    public Scale s = new Scale();

    public Xform() {
        super();
        getTransforms().addAll(t, rz, ry, rx, s);
    }

    public Xform(RotateOrder rotateOrder) {
        super();
        // choose the order of rotations
        //based on the rotateOrder
        switch (rotateOrder) {
        case XYZ:
            getTransforms().addAll(t, p, rz, ry, rx, s, ip);
            break;
        case XZY:
            getTransforms().addAll(t, p, ry, rz, rx, s, ip);
            break;
        case YXZ:
            getTransforms().addAll(t, p, rz, rx, ry, s, ip);
```

```
            break;
        case YZX:
            getTransforms().addAll(t, p, rx, rz, ry, s, ip);
            break;
        case ZXY:
            getTransforms().addAll(t, p, ry, rx, rz, s, ip);
            break;
        case ZYX:
            getTransforms().addAll(t, p, rx, ry, rz, s, ip);
            break;
        }
    }

    public void setTranslate(double x, double y, double z) {
        t.setX(x);
        t.setY(y);
        t.setZ(z);
    }

    public void setTranslate(double x, double y) {
        t.setX(x);
        t.setY(y);
    }

    // Cannot override these methods as they are final:
    // public void setTranslateX(double x) { t.setX(x); }
    // public void setTranslateY(double y) { t.setY(y); }
    // public void setTranslateZ(double z) { t.setZ(z); }
    // Use these methods instead:
    public void setTx(double x) { t.setX(x); }
    public void setTy(double y) { t.setY(y); }
    public void setTz(double z) { t.setZ(z); }

    public void setRotate(double x, double y, double z) {
        rx.setAngle(x);
        ry.setAngle(y);
        rz.setAngle(z);
    }

    public void setRotateX(double x) { rx.setAngle(x); }
```

```
    public void setRotateY(double y) { ry.setAngle(y); }
    public void setRotateZ(double z) { rz.setAngle(z); }
    public void setRx(double x) { rx.setAngle(x); }
    public void setRy(double y) { ry.setAngle(y); }
    public void setRz(double z) { rz.setAngle(z); }

    public void setScale(double scaleFactor) {
        s.setX(scaleFactor);
        s.setY(scaleFactor);
        s.setZ(scaleFactor);
    }

    public void setScale(double x, double y, double z) {
        s.setX(x);
        s.setY(y);
        s.setZ(z);
    }

    // Cannot override these methods as they are final:
    // public void setScaleX(double x) { s.setX(x); }
    // public void setScaleY(double y) { s.setY(y); }
    // public void setScaleZ(double z) { s.setZ(z); }
    // Use these methods instead:
    public void setSx(double x) { s.setX(x); }
    public void setSy(double y) { s.setY(y); }
    public void setSz(double z) { s.setZ(z); }

    public void setPivot(double x, double y, double z) {
        p.setX(x);
        p.setY(y);
        p.setZ(z);
        ip.setX(-x);
        ip.setY(-y);
        ip.setZ(-z);
    }

    public void reset() {
        t.setX(0.0);
        t.setY(0.0);
        t.setZ(0.0);
```

```
        rx.setAngle(0.0);
        ry.setAngle(0.0);
        rz.setAngle(0.0);
        s.setX(1.0);
        s.setY(1.0);
        s.setZ(1.0);
        p.setX(0.0);
        p.setY(0.0);
        p.setZ(0.0);
        ip.setX(0.0);
        ip.setY(0.0);
        ip.setZ(0.0);
    }

    public void resetTSP() {
        t.setX(0.0);
        t.setY(0.0);
        t.setZ(0.0);
        s.setX(1.0);
        s.setY(1.0);
        s.setZ(1.0);
        p.setX(0.0);
        p.setY(0.0);
        p.setZ(0.0);
        ip.setX(0.0);
        ip.setY(0.0);
        ip.setZ(0.0);
    }
}//End of class Xform; End of program M112 MlgrmLeftHand4
```

124, page 427.

```
 /*
 *  Program M124 HandChimpHuman4
    Animation that shows a hand of a chimp
    transforming into the hand of a human.
    Data were captured using program M120 HandChimpHuman2.

    Every one that has ever seen a human hand together
```

```
    with a drawing of the hand of a chimpanzee
    has the sacred duty of
    proposing that the first evolved from the other.
    This is our way of fulfilling that duty:
    Evolution is the null hypothesis of science.
*/
package handchimphuman4;



import javafx.animation.KeyFrame;
import javafx.animation.KeyValue;
import javafx.animation.Timeline;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Line;
import javafx.stage.Stage;


public class HandChimpHuman4 extends Application
{

    //Chimp: x-coordinates
    private static double[] xChimp  = new double[]{
        125, 141, 161, 131, 115, 108,  99, 103,  97, 101,
        109, 118, 124, 126, 122, 127, 140, 146, 148, 151,
        154, 161, 173, 180, 178, 181, 185, 192, 201, 208,
        204, 208, 207, 205, 201, 204, 211, 210, 214, 213,
        218, 226, 230, 231, 235, 230, 228, 219, 201, 188,
        180, 182, 190, 200, 203};
        //Chimp: y-coordinates
    private static double[] yChimp  = new double[]{
        362, 362, 394, 411, 404, 394, 296, 277, 192, 179,
        180, 192, 250, 261, 155, 138, 136, 147, 239, 250,
        147, 130, 128, 141, 245, 253, 174, 162, 160, 174,
```

```
        263, 277, 302, 311, 321, 330, 341, 330, 322, 311,
        301, 295, 296, 307, 320, 335, 357, 385, 409, 417,
        411, 378, 357, 354, 345};
        //Human: x-coordinates
    private static double[] xHuman  = new double[]{
        323, 319, 314, 310, 302, 299, 290, 293, 293, 299,
        308, 310, 313, 319, 320, 325, 337, 340, 340, 346,
        347, 355, 361, 369, 368, 374, 376, 383, 394, 395,
        393, 394, 394, 394, 395, 395, 392, 398, 407, 405,
        410, 418, 429, 427, 430, 426, 416, 397, 377, 362,
        333, 345, 368, 377, 387};
        //Human: y-coordinates
    private static double[] yHuman  = new double[]{
        408, 409, 409, 409, 409, 395, 315, 303, 238, 229,
        230, 237, 291, 286, 214, 199, 199, 212, 287, 286,
        198, 187, 190, 195, 286, 289, 211, 202, 205, 218,
        291, 298, 310, 315, 319, 324, 340, 321, 313, 298,
        285, 278, 280, 295, 313, 325, 350, 383, 405, 409,
        396, 364, 338, 331, 325};

@Override
public void start(Stage stage)
{

    Group root = new Group();
    //Both drawings have equal number of points
    int nPoints = xHuman.length;


    //Draw the hand of the chimp
    for(int i = 0; i < nPoints-1; i++)
    {
        Line l = new Line(xChimp[i], yChimp[i],
                          xChimp[i+1], yChimp[i+1]);
        l.setFill(Color.BLUE);
        root.getChildren().add(l);
    }
    //Draw the hand of the human
    for(int i = 0; i < nPoints-1; i++)
    {
```

```
        Line l = new Line(xHuman[i], yHuman[i],
                    xHuman[i+1], yHuman[i+1]);
        l.setFill(Color.BLUE);
        root.getChildren().add(l);
    }

    //Animation.
    //Each line is controlled by its bordering points
    DoubleProperty xo ;
    DoubleProperty yo ;
    DoubleProperty x1 ;
    DoubleProperty y1 ;
    for(int i = 0; i < nPoints-1; i++)
    {
        //Line goes from (xo,yo) to (x1,y1);
        xo = new SimpleDoubleProperty( xChimp[i] );
        yo = new SimpleDoubleProperty( yChimp[i] );
        x1 = new SimpleDoubleProperty( xChimp[i+1] );
        y1 = new SimpleDoubleProperty( yChimp[i+1] );
        Line l = new Line
        (
            xo.getValue(),yo.getValue(),
            x1.getValue(),y1.getValue()
        );
        //End points get bound to  varying parameters
        l.startXProperty().bind(xo);
        l.startYProperty().bind(yo);
        l.endXProperty().bind(x1);
        l.endYProperty().bind(y1);
        l.setFill(Color.BLUE);
        root.getChildren().add(l);

        //Dynamic events
        Timeline timeline = new Timeline();
        //Each stroke of the hand of the chimp
        //moves towards its homologue at the hand of human.
        timeline.getKeyFrames().addAll
        (
            //Initial states
            new KeyFrame( javafx.util.Duration.ZERO,
```

```
                new KeyValue( xo, xChimp[i] ),
                new KeyValue( yo, yChimp[i] ),
            new KeyValue( x1, xChimp[i+1] ),
                new KeyValue( y1, yChimp[i+1] )),
            //Final states
            new KeyFrame( new javafx.util.Duration(10000),
                new KeyValue( xo, xHuman[i] ),
                new KeyValue( yo, yHuman[i] ),
            new KeyValue( x1, xHuman[i+1] ),
                new KeyValue( y1, yHuman[i+1] ))

        );

    timeline.setCycleCount( Timeline.INDEFINITE );
    timeline.setAutoReverse( true );
    timeline.play();
    }

    Scene scene = new Scene(root, 500, 600);
    stage.setScene(scene);
    stage.setTitle("From the hand of chimp into "
            + "the hand of human");
    stage.show();
}


public static void main(String[] args)
{
    launch(args);
}

}//End of main class M124 HandChimpHuman4
```

128, page 428.

To formulate a mandatory, falsifiable and false prediction of The Evolutionary Theory we follow the following pattern: *Did you say that you appeared here after running 14km in an hour? You are here, OK but, where is your perspiration?* Our formulation is as follows:

The human, the chimp and the bonobo are very violent and also by their very nature they are killing machines. In fact, chimps are blamed by having caused severe damage to many persons. To that purpose they used their teeth and nails. By the same token, bonobos join in small groups to chase chimps. On his side, the man loves to use his whole body, intelligence and weapons to harm others. So, the proposed evolutionary hypothesis must be complemented by assuming that violence was an important molding factor of the evolutionary branch leading to modern chimp, bonobo and man.

Curvature of phalanges serves the purpose of making room to accommodate thicker tendons and bigger muscles with more place to get a contact point. Tendons and muscles over not curved phalanges would greatly interfere with the fitting of hand grip to objects. Now, the curvature of fingers of the chimp is adjusted to make of it a champion in arboreal mobility, that is nevertheless not as spectacular as that of the orangutan. On the other hand, man can get trained to be good at arm swinging but this activity is not the one he usually prefers because his strong mode of locomotion is bipedal. So, how could curved phalanges help the man? They would allow for more powerful muscles than on its turn would demand more mineralized bones. Specimens like these are predicted to be present as in the fossil record as in modern populations because they would have an advantage in the management of clubs, very important as for defense as for attack.

Is gradualism used in our argument? Yes, it has been used everywhere: to evolve from a hand that is perfect for hanging to a multi-grip hand (Alan Hedge, [9], 2013) is not that easy and one must suppose that everything can softly change to make evolution reliable. Moreover, suppose that originally the homo branch was packed without curved phalanges. In that case and thanks to gradualism, curvature would be an inescapably acquired by evolution. In the official jargon: acquiring curvature is under gradualism an evolutionary invading, stable strategy. That is why our prediction is mandatory. And it is also false: predicted specimens exist neither in extant population nor in the fossil record.

130, page 436. These are closed polylines associated to the medial and Dorsal views of the images of the `Manual Intermediate Phalanx 3` of the thumb:

```
//Output gotten by the Author:
    chimpanzee_hands_manual_intermediate_phalanx_3_Medial_thumb
   x-coordinates
   106,  99,  97,  96,  90,  86,  85,  83,  91,  98, 105, 111,
   114, 114, 109, 105, 103, 104, 105, 109, 114, 111, 106,
   y-coordinates
    99, 102, 108, 114, 128, 147, 185, 193, 201, 197, 195, 197,
```

```
    193, 190, 180, 166, 152, 137, 126, 117, 109, 104,  99,

    chimpanzee_hands_manual_intermediate_phalanx_3_Dorsal_thumb
    x-coordinates
    100,  97,  93,  90,  90,  90,  87,  87,  83,  80,  88,
    102, 112, 120, 120, 117, 118, 113, 112, 113, 110, 100,
    y-coordinates
    102,  99, 101, 109, 122, 145, 158, 168, 181, 188, 199,
    201, 198, 191, 183, 181, 173, 140, 126, 111, 102, 103,
```

134, page 450. The following code draws a polycone generated by a polyline that orbits around the z-axis.

```
//Program M134 HandChimpHuman7
//This program calculates and draws the 3D mesh
//of the truncated cone
//generated by a polyline that orbits around
//the z-axis.


package handchimphuman7;


import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.PerspectiveCamera;
import javafx.scene.PointLight;
import javafx.scene.Scene;
import javafx.scene.control.MenuBar;
import javafx.scene.input.KeyEvent;
import javafx.scene.input.MouseEvent;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.Box;
import javafx.scene.shape.CullFace;
import javafx.scene.shape.DrawMode;
import javafx.scene.shape.MeshView;;
```

```
import javafx.scene.shape.TriangleMesh;
import javafx.scene.transform.Rotate;
import javafx.scene.transform.Scale;
import javafx.scene.transform.Translate;
import javafx.stage.Stage;



public class HandChimpHuman7 extends Application
{
    //Polyline:
    //x-coordinates
    private static float[] xPolyline  = new float[]
    {
        106,  99,  97,  96,  90,  86,
        85,  83,  91,  98, 105
    };
    //y-coordinates
    private static float[] yPolyline  = new float[]
    {
         99, 102, 108, 114, 128, 147,
        185, 193, 201, 197, 195
    };




    final Group root = new Group();

    final Group axisGroup = new Group();

    final PerspectiveCamera camera
            = new PerspectiveCamera(true);
    final Xform cameraXform = new Xform();
    final Xform cameraXform2 = new Xform();
    final Xform cameraXform3 = new Xform();

    //Parameter to regulate the size of graphic
    final double cameraDistance = 7850;
```

```
 final Xform world3DXform = new Xform();
 final Xform bonesXform = new Xform();


 double mousePosX;
 double mousePosY;
 double mouseOldX;
 double mouseOldY;
 double mouseDeltaX;
 double mouseDeltaY;



final Xform guidesXform = new Xform();
//Menubar declaration and initialization
 MenuBar menuBar = new MenuBar();


  private void buildCamera() {
     root.getChildren().add(cameraXform);
     cameraXform.getChildren().add(cameraXform2);
     cameraXform2.getChildren().add(cameraXform3);
     cameraXform3.getChildren().add(camera);
     cameraXform3.setRotateZ(180.0);

     camera.setNearClip(0.1);
     camera.setFarClip(10000.0);
     camera.setTranslateZ(-cameraDistance);
     cameraXform.ry.setAngle(320.0);
     cameraXform.rx.setAngle(40);
 }



 private void buildAxes() {
     System.out.println("buildAxes()");
     final PhongMaterial redMaterial
             = new PhongMaterial();
     redMaterial.setDiffuseColor(Color.DARKRED);
     redMaterial.setSpecularColor(Color.RED);
```

```
        final PhongMaterial greenMaterial
                = new PhongMaterial();
        greenMaterial.setDiffuseColor(Color.DARKGREEN);
        greenMaterial.setSpecularColor(Color.GREEN);

        final PhongMaterial blueMaterial
                = new PhongMaterial();
        blueMaterial.setDiffuseColor(Color.DARKBLUE);
        blueMaterial.setSpecularColor(Color.BLUE);

        final Box xAxis = new Box(640.0, 1, 1);
        final Box yAxis = new Box(1, 640.0, 1);
        final Box zAxis = new Box(1, 1, 640.0);

        xAxis.setMaterial(redMaterial);
        yAxis.setMaterial(greenMaterial);
        zAxis.setMaterial(blueMaterial);

        axisGroup.getChildren().addAll(xAxis, yAxis, zAxis);
        axisGroup.setVisible(false);
        world3DXform.getChildren().addAll(axisGroup);
    }



private void handleMouse(Scene scene)
{
    scene.setOnMousePressed((MouseEvent me) ->
    {
        mousePosX = me.getSceneX();
        mousePosY = me.getSceneY();
        mouseOldX = me.getSceneX();
        mouseOldY = me.getSceneY();
    });
    scene.setOnMouseDragged((MouseEvent me) ->
    {
        mouseOldX = mousePosX;
        mouseOldY = mousePosY;
        mousePosX = me.getSceneX();
```

```
        mousePosY = me.getSceneY();
        mouseDeltaX = (mousePosX - mouseOldX);
        mouseDeltaY = (mousePosY - mouseOldY);

        double modifier = 1.0;
        double modifierFactor = 0.1;

        if (me.isControlDown()) {
            modifier = 0.1;
        }
        if (me.isShiftDown()) {
            modifier = 10.0;
        }
        if (me.isPrimaryButtonDown()) {
            cameraXform.ry.setAngle(cameraXform
                    .ry.getAngle()
                    - mouseDeltaX*modifierFactor
                            *modifier*2.0);  // +
            cameraXform.rx.setAngle(cameraXform
                    .rx.getAngle()
                    + mouseDeltaY*modifierFactor
                            *modifier*2.0);  // -
        }
        else if (me.isSecondaryButtonDown()) {
            double z = camera.getTranslateZ();
            double newZ = z
            + mouseDeltaX*modifierFactor*modifier;
            camera.setTranslateZ(newZ);
        }
        else if (me.isMiddleButtonDown()) {
            cameraXform2.t.setX(cameraXform2.t.getX()
                + mouseDeltaX*modifierFactor*modifier*0.3);
            cameraXform2.t.setY(cameraXform2.t.getY()
                + mouseDeltaY*modifierFactor*modifier*0.3);
        }
        });
}

private void handleKeyboard(Scene scene)
{
```

```
    scene.setOnKeyPressed((KeyEvent event) -> {

        switch (event.getCode()) {
            case A: //Hide axis
                if (event.isControlDown()) {
                    if (axisGroup.isVisible()) {
                        System.out.println(
                                "setVisible(false)");
                        axisGroup.setVisible(false);
                    }
                    else {
                        System.out.println(
                                "setVisible(true)");
                        axisGroup.setVisible(true);
                    }
                }
                break;
        }
    });
}



@Override
public void start(Stage primaryStage)
{

    primaryStage.setTitle("Function graphicator. "
            + "Axes = ctrl+A");
    //The whole drawing  is created
    //as a JavaFX space-like object
    final Group parent = new Group(create3dContent());

    //Window of the scene
    final Scene scene = new Scene(root, 900, 900, true);
    //Background
    scene.setFill(Color.SKYBLUE);
    buildCamera();
    primaryStage.setScene(scene);
```

```
    primaryStage.show();
    buildAxes();
    handleKeyboard(scene);
    handleMouse(scene);
    scene.setCamera(camera);
    root.getChildren().addAll(parent);

}

//Manager of graphic
public Node create3dContent()
{
    Graphic b = new Graphic();
    b.getChildren().add( world3DXform);
    b.rx.setAngle(45);
    b.ry.setAngle(110);

    return new Group(b);
}


//A mesh is coverted into a meshView.
public class Graphic extends Group
{
final Rotate rx = new Rotate(0,Rotate.X_AXIS);
final Rotate ry = new Rotate(0,Rotate.Y_AXIS);
final Rotate rz = new Rotate(0,Rotate.Z_AXIS);

//The ambient of the graphic is a box
//of measures a, b, and c.
public  Graphic()
  {
    getTransforms().addAll(rz, ry, rx);

    //Mesh of the truncated cone
    TriangleMesh meshFunction
            = buildMeshFunction();
    PhongMaterial materialF  = new PhongMaterial();
    materialF.setDiffuseColor(Color.FUCHSIA);
    MeshView  meshViewFunction = new MeshView(meshFunction);
```

```
    meshViewFunction.setMaterial(materialF);
    //meshViewFunction.setDrawMode(DrawMode.LINE);
    meshViewFunction.setDrawMode(DrawMode.FILL);
    meshViewFunction.setCullFace(CullFace.NONE);

    //Illumination
    PointLight pointLight;
    pointLight = new PointLight(Color.ANTIQUEWHITE);
    pointLight.setTranslateX(150);
    pointLight.setTranslateY(-100);
    pointLight.setTranslateZ(-1000);

    getChildren().addAll(meshViewFunction,pointLight  );
  }
}//End of class Graphic



//The mesh of the truncated cone
static TriangleMesh buildMeshFunction()
{

    //Scales
    float zScale =   5f;
    float xyScale =  5;


    //Numbers of subdivisions along z
    int n = yPolyline.length;

    double PI =  Math.PI;
    //Numbers of subdivisions along theta angle
    int m = 66 ;
    //Angle increment
    double deltaTheta =  (2*PI/m) ;

    //Array with coordinates of points of
    //the mesh, each one with 3 coordinates
    float points[] = new float[3* (n+1) * (m +1)];
```

```
//There are n*m tiny rectangles.
//Each rectangle has 2 triangles
int nFaces = 2*n*m;
//Each point of each face has 6 coordinates,
//3 for the vertex and
//3 for the marked points of the wallpaper (texture)
int faces[] = new int[nFaces * 6];


// Create points
//z-variable
for (int j = 0; j < n; j++)
{
    //X becomes the radius and y turns into z
    float zj = yPolyline[j];
    //float r = xPolyline[j];
    float r = xPolyline[j] - xPolyline[0];
    //float r = xPolyline[0]-(xPolyline[j] - xPolyline[0]);


    //Angle theta
    for (int i = 0; i <= m; i++)
    {
        double thetai = i*deltaTheta;
        int index = j * (m+1)* 3 + (i * 3);
        points[index] =
            r * ( (float) Math.cos(thetai)) * xyScale;
        points[index + 1]  =
            r * ( (float) Math.sin(thetai)) * xyScale;
        points[index + 2] = zj*zScale;
    }
}


//Faces are built.
//Two triangles for each rectangle.
for (int j = 0; j < n-1; j++)
{
    for (int i = 0; i < m; i++)
    {
```

```
            int p00 = j * (m+1) + i;
            int p01 = p00 + 1;
            int p10 = p00 + (m+1);
            int p11 = p10 + 1;
            int tc00 = 0;
            int tc01 = 0;
            int tc10 = 0;
            int tc11 = 0;
            //First triangle
            int index = (j * m * 6 + (i * 6)) * 2;
            faces[index + 0] = p00;
            faces[index + 1] = tc00;
            faces[index + 2] = p10;
            faces[index + 3] = tc10;
            faces[index + 4] = p11;
            faces[index + 5] = tc11;
            //second triangle
            index = index + 6;
            faces[index + 0] = p11;
            faces[index + 1] = tc11;
            faces[index + 2] = p01;
            faces[index + 3] = tc01;
            faces[index + 4] = p00;
            faces[index + 5] = tc00;
        }
    }


    TriangleMesh mesh = new TriangleMesh();
    mesh.getPoints().addAll(points);

    mesh.getTexCoords().addAll(0,0);
    mesh.getFaces().addAll(faces);

    return mesh;
}

public static void main(String[] args)
{
```

```
    launch(args);
}

}//End of main class M134 MlgrmLeftHand7


//Outer class Xform.
//Machinery that enables rotations in vivo of the scene
//and the conformation of groups of control.
class Xform extends Group {

    public enum RotateOrder {
        XYZ, XZY, YXZ, YZX, ZXY, ZYX
    }

    public Translate t  = new Translate();
    public Translate p  = new Translate();
    public Translate ip = new Translate();
    public Rotate rx = new Rotate();
    { rx.setAxis(Rotate.X_AXIS); }
    public Rotate ry = new Rotate();
    { ry.setAxis(Rotate.Y_AXIS); }
    public Rotate rz = new Rotate();
    { rz.setAxis(Rotate.Z_AXIS); }
    public Scale s = new Scale();

    public Xform() {
        super();
        getTransforms().addAll(t, rz, ry, rx, s);
    }

    public Xform(RotateOrder rotateOrder) {
        super();
        // choose the order of rotations
        //based on the rotateOrder
        switch (rotateOrder) {
        case XYZ:
            getTransforms().addAll(t, p, rz, ry, rx, s, ip);
            break;
        case XZY:
```

```
            getTransforms().addAll(t, p, ry, rz, rx, s, ip);
            break;
        case YXZ:
            getTransforms().addAll(t, p, rz, rx, ry, s, ip);
            break;
        case YZX:
            getTransforms().addAll(t, p, rx, rz, ry, s, ip);
            break;
        case ZXY:
            getTransforms().addAll(t, p, ry, rx, rz, s, ip);
            break;
        case ZYX:
            getTransforms().addAll(t, p, rx, ry, rz, s, ip);
            break;
        }
    }

    public void setTranslate(double x, double y, double z) {
        t.setX(x);
        t.setY(y);
        t.setZ(z);
    }

    public void setTranslate(double x, double y) {
        t.setX(x);
        t.setY(y);
    }

    // Cannot override these methods as they are final:
    // public void setTranslateX(double x) { t.setX(x); }
    // public void setTranslateY(double y) { t.setY(y); }
    // public void setTranslateZ(double z) { t.setZ(z); }
    // Use these methods instead:
    public void setTx(double x) { t.setX(x); }
    public void setTy(double y) { t.setY(y); }
    public void setTz(double z) { t.setZ(z); }

    public void setRotate(double x, double y, double z) {
        rx.setAngle(x);
        ry.setAngle(y);
```

```
    rz.setAngle(z);
}

public void setRotateX(double x) { rx.setAngle(x); }
public void setRotateY(double y) { ry.setAngle(y); }
public void setRotateZ(double z) { rz.setAngle(z); }
public void setRx(double x) { rx.setAngle(x); }
public void setRy(double y) { ry.setAngle(y); }
public void setRz(double z) { rz.setAngle(z); }

public void setScale(double scaleFactor) {
    s.setX(scaleFactor);
    s.setY(scaleFactor);
    s.setZ(scaleFactor);
}

public void setScale(double x, double y, double z) {
    s.setX(x);
    s.setY(y);
    s.setZ(z);
}

// Cannot override these methods as they are final:
// public void setScaleX(double x) { s.setX(x); }
// public void setScaleY(double y) { s.setY(y); }
// public void setScaleZ(double z) { s.setZ(z); }
// Use these methods instead:
public void setSx(double x) { s.setX(x); }
public void setSy(double y) { s.setY(y); }
public void setSz(double z) { s.setZ(z); }

public void setPivot(double x, double y, double z) {
    p.setX(x);
    p.setY(y);
    p.setZ(z);
    ip.setX(-x);
    ip.setY(-y);
    ip.setZ(-z);
}
```

```
    public void reset() {
        t.setX(0.0);
        t.setY(0.0);
        t.setZ(0.0);
        rx.setAngle(0.0);
        ry.setAngle(0.0);
        rz.setAngle(0.0);
        s.setX(1.0);
        s.setY(1.0);
        s.setZ(1.0);
        p.setX(0.0);
        p.setY(0.0);
        p.setZ(0.0);
        ip.setX(0.0);
        ip.setY(0.0);
        ip.setZ(0.0);
    }

    public void resetTSP() {
        t.setX(0.0);
        t.setY(0.0);
        t.setZ(0.0);
        s.setX(1.0);
        s.setY(1.0);
        s.setZ(1.0);
        p.setX(0.0);
        p.setY(0.0);
        p.setZ(0.0);
        ip.setX(0.0);
        ip.setY(0.0);
        ip.setZ(0.0);
    }
}//End of class Xform; End of program M134 MlgrmLeftHand7
```

135, page 450. We modified `Program M132 HandChimpHuman6`. We changed the code from line 314 to line 333 by the following piece of code:

```
 for (int j = 0; j <= n; j++)
    {
```

```
        //Z begins with zo and is incremented by deltaZ
        float zj = zo + deltaZ*j;
        float ra = 3*zj + 100;
        float rb = 1.0f*zj + 10;

        //Angle theta
        for (int i = 0; i <= m; i++)
        {
            double thetai = i*deltaTheta;
            int index = j * (m+1)* 3 + (i * 3);
            points[index] =   ra
                * ( (float) Math.cos(thetai)) * xyScale;
            points[index + 1]  =   rb
                * ( (float) Math.sin(thetai)) * xyScale;
            points[index + 2] = zj*zScale;

        }
    }
```

138, page 464. These are 4 polylines representing the silhouettes of two views of the bone `chimpanzee_hands_manual_intermediate_phalanx_3`

```
chimpanzee_hands_manual_intermediate_phalanx_3

Medial view, left side:
x-coordinates
105, 98, 95, 95, 90, 87, 87, 87, 85, 84, 91, 101,
y-coordinates
101, 104, 109, 113, 126, 142, 161, 178, 187, 191, 200, 197,
Medial view, right side:
x-coordinates
107, 112, 111, 107, 105, 102, 104, 108, 112, 114, 109, 100,
y-coordinates
101, 106, 110, 118, 128, 142, 161, 176, 185, 190, 199, 196,

Dorsal view, left side:
x-coordinates
101, 98, 92, 89, 89, 90, 87, 87, 82, 80, 86, 100,
y-coordinates
102, 101, 102, 112, 128, 145, 156, 168, 180, 188, 198, 200,
```

```
Dorsal view, right side:
x-coordinates
102, 109, 111, 114, 113, 114, 116, 118, 117, 120, 113, 102,
y-coordinates
103, 100, 106, 111, 129, 146, 160, 171, 179, 186, 198, 201,

/*Use the following code to see a polyline.
     Paste the code into a 2D JavaFX application.

    //Draw a polyline
    for(int i = 0; i < length-1; i++)
    {
        Line l = new Line(xPolyline[i], yPolyline[i],
                          xPolyline[i+1], yPolyline[i+1]);
        l.setFill(Color.BLUE);
        root.getChildren().add(l);
    }
    */
```

Bibliography

[1] BBC (2014) DISSECTED THE INCREDIBLE HUMAN 1OF2 HAND
https://www.youtube.com/watch?v=MIKjRtN5pSI
Verified 27/III/2015 407

[2] BLENDER
http://www.blender.org/
Verified 27/III/2015 405

[3] BODYPARTS3D
http://lifesciencedb.jp/bp3d/
Verified 27/III/2015 350

[4] "BODYPARTS3D, (C) THE DATABASE CENTER FOR LIFE SCIENCE LICENSED UNDER CC ATTRIBUTION-SHARE ALIKE 2.1 JAPAN".
http://dbarchive.biosciencedbc.jp/en/bodyparts3d/lic.html
Verified 27/III/2015 350

[5] ESKELETONS
http://www.eskeletons.org/
Verified 27/III/2015 428

[6] FASTSCAN3D
http://www.fastscan3d.com/download/samples/
Verified 27/III/2015 405

[7] FREE MOLOGRAMS
http://www.hongkiat.com/blog/60-excellent-free-3d-model-websites/
Verified 26/III/2015 405

[8] DENISE MORGAN (2015). CHIMP AND HUMAN HANDS (IMAGE)
http://www.eurekalert.org/multimedia/pub/50921.php?from=228578
Verified 27/III/2015 411

[9] ALAN HEDGE (2013). GRIPS
http://ergo.human.cornell.edu/studentdownloads/DEA3250pdfs/grips.pdf
Verified 27/III/2015 694

[10] INTERACTIVEMESH
http://www.interactivemesh.org/models/jfx3dimporter.html
Verified 27/III/2015 350

[11] MESHLAB
http://meshlab.sourceforge.net/
Verified 27/III/2015 405

[12] ORACLE 2013 Getting Started with JavaFX 3D Graphics.
http://docs.oracle.com/javafx/8/3d_graphics/jfxpub-3d_graphics.htm
Verified 26/III/2015 1

[13] ORACLE 2014 Class Rotate.
https://docs.oracle.com/javafx/2/api/javafx/scene/transform/Rotate.html
Verified 26/III/2015 21

[14] THE.OBJ FORMAT
http://en.wikipedia.org/wiki/Wavefront_.obj_file
Verified 27/III/2015 351

[15] THE.OBJ FORMAT 2
http://www.martinreddy.net/gfx/3d/OBJ.spec
Verified 27/III/2015 351

[16] YOUNG RW (2003). Evolution of the human hand: the role of throwing and clubbing. Journal of Anatomy 202(1):165-174. doi:10.1046/j.1469-7580.2003.00144.x.
http://www.ncbi.nlm.nih.gov/pmc/articles/PMC1571064/
Verified 26/III/2015 241

[17] WAVEFRONT TECHNOLOGIES
http://en.wikipedia.org/wiki/Wavefront_Technologies
Verified 27/III/2015 351