Documentation for the Model Program system

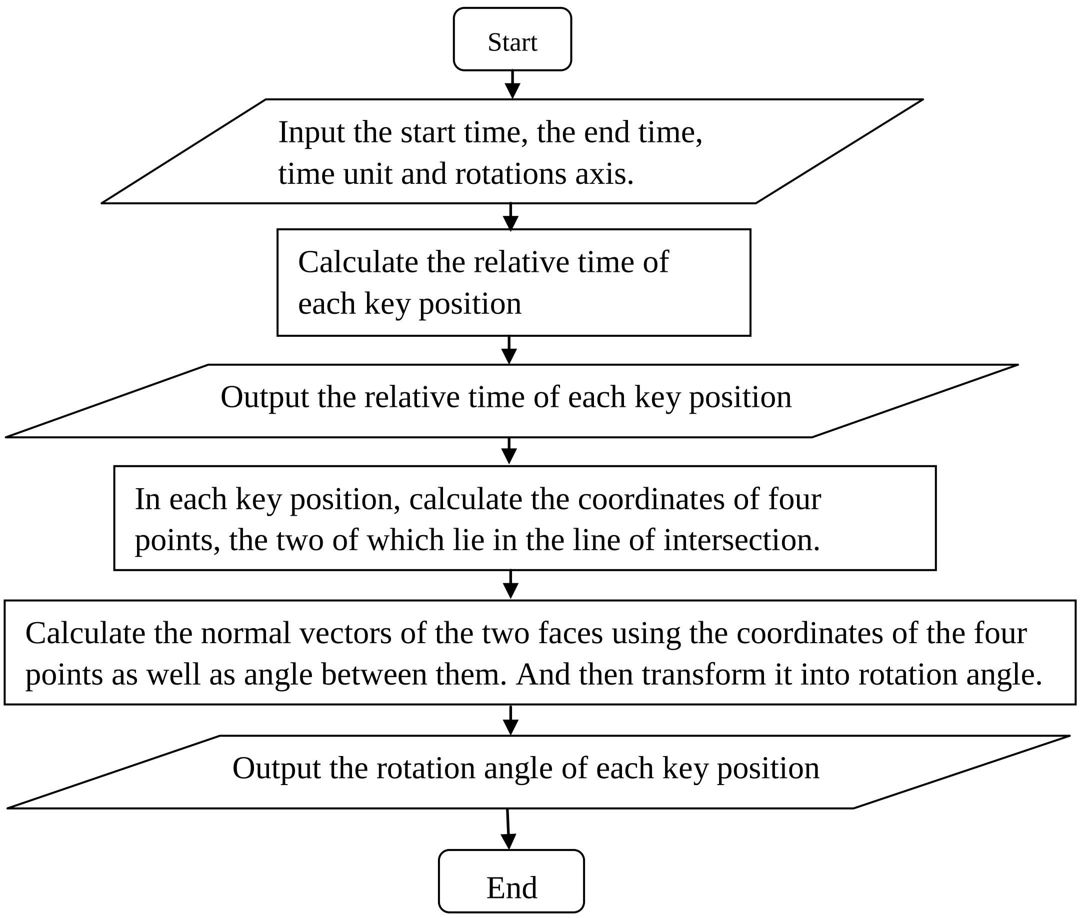
1 Information about the Model program:

Author: Jindong Gu
Contact E-mail: ostdong@gmail.com
Date: January 1. 2016
Type of work: Bachelor Thesis in Wuppertal University
Version: 1.7
Implementation Language: Java
Runtime Environment: Windows 7, IDE Netbeans 8.1

2 Background of the Model Program:

A large number of parameters describing key positions are necessary, when a non-uniform rotation should be described in X3D scene. This program is developed to produce parameters for key positions of animation based on the interpolators in X3D. A key position can be described by a rotation axis (i.e. a vector of three numbers) and a rotation angle (an angle in radian). Generally, the rotation axis stays unchanged, which can be easy found. And the value can be assigned to a corresponding variable of the program directly. Specially, this program is developed to calculate rotation angles for each key position in the "OrientationInterpolator" node of X3D scenes.

The idea of the program is to calculate the angle between two faces in each key position and transform the angles into parameters we need. The program accepts the start time and the end time of a rotation, a time unit (i.e. the time interval between two key positions) and rotations axis as parameters. The process of the calculation as follows:



3 Content of the Model Program:

The project (i.e. the file “Yoshizawa_Randlett_System”) of this program contains two classes. One of them is an abstract class (origami.class), which encapsulates an interface and some functions that have already been implemented. Besides, the program also presents an implementation class that demonstrates how to inherit the abstract class and realize its functionalities.

Class: Origami

Attribute:
public String axis // Store the value of rotation axis.

Interface:
//The interface returns the coordinates of four points. It should be implemented in Implementation class.
public abstract double[] coordinates (double t, double start_time, double end_time);

Function:
//Calculate the normal vectors, given the coordinates of the three points that lie in a same plane.
public double[] noraml_vektor(double x1, double y1, double z1, double x2, double y2, double z2, double x3, double y3, double z3){}

//Given the coordinates of four points, calculate the acute angle between the two rotation faces.
public double faces_angle(double[] coordinates){}

//Select the instant, which is closest to the π /2.
public double time_threshold(double start_time, double end_time, double time_unit){}

//Transform the angles into arguments we need with the help of the time threshold.
public double rotation_angle(double angle, double t, double t_threshold){}

//Output the parameters for key positions.
public void output_arguments(double start_time, double end_time, double time_unit)

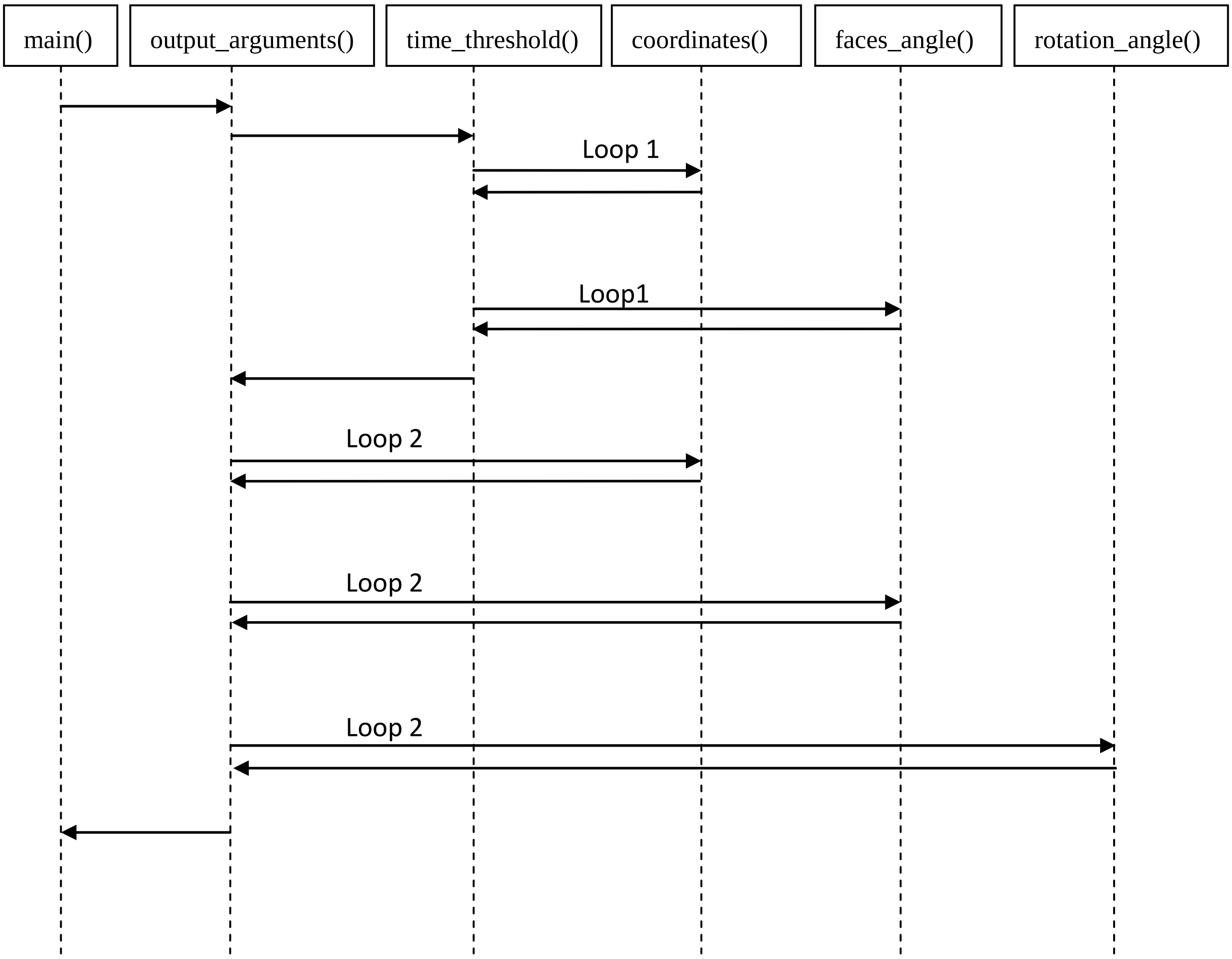
extends (inherits)

Class: Inside_Reverse_Fold:

Function:
//Start the program by invoking the “output_arguments” function and assign the value of rotation axis.
public static void main(String[] args)

//Implement the interface according to a concrete rotation and return the 12 coordinates of four points.
public double[] coordinates (double t, double start_time, double end_time);

4 Flow charts of the Program:



The sequence diagram shows the time order of the invocation of all the functions. Therein, “Loop 1” means that the invocations of the function “coordinates()” and the function “faces_angle()” are in a same loop body; and also the two functions are invoked many times by the function “time_threshold()” with the run of the loop body. Similarly, “Loop 2” means that the invocations of the function “coordinates()”, the function “faces_angle()” and “rotation_angle()” are in a same loop body. The number of the invocations depends on the parameters (i.e. start time, end time and time unit).

5 Usage and Extension of the model program

Users can create a new class that inherits the abstract class “Origami”. The new class must implement the interface “coordinates()” [1] of the abstract class. The implementation function [2] accepts the start time and end time of the rotation and current time as arguments. It must return 12 coordinates (of four points), which is stored into an array. The coordinates of the four points may be influenced by the arguments. It depends on the rotation of origami model. The creator must give extra information (e.g. geometric relationships of the four points) into the implementation function according to the rotation whose parameter should be calculated. Then create a main function in the new class to start the program.[3]

In addition, user can also further develop the model program to provide more information for the animation in X3D. The key functions are already given in the program. Users can defined more interfaces and allow program to accept more arguments to create a complete X3D scene.

[1] Q.v. Source Code, 6.1 Origami.java, Line 16

[2] Q.v. Source Code, 6.2 Inside_Reverse_Fold.java, Lines 14-59

[3] Q.v. Source Code, 6.2 Inside_Reverse_Fold.java, Lines 60-65

6 Source code

6.1 Origami.java

```
/* @author: Jindong Gu's Bachelor Thesis in Wuppertal University.**
** This program is developed to calculate the angles of rotation in the OrientationInterpolator node of X3D file.**
** This program includes an abstract class, which encapsulates an interface and some functions that have already been implemented.**
** Besides, the program presents an implementation class that demonstrates how to inherit the abstract class and realize its functionalities.*/
package yoshizawa_randlett_system;
import static java.lang.Math.acos;
import static java.lang.Math.sqrt;
import static java.lang.Math.abs;
/* An abstract class that includes an interface and some functions is defined as follows.**
** This abstract class must be inherited later.*/
public abstract class Origami {
	//A rotation axis is defined, which can be changed in the implementation class.
	public String axis = "0 0 0 ";
	// The purpose of this abstract function is to calculate all the point coordinates in instant t.
	// This interface must be also implemented in accordance with the concrete rotation of a portion of a paper model.
	public abstract double[] coordinates(double t, double start_time, double end_time);
	// If the only interface is implemented correctly, user can call this function with corresponding arguments to get results.
	public void output_arguments(double start_time, double end_time, double time_unit) {
		//The time is time threshold, namely, the instant when the biggest angle (about π /2) between the two faces occurs.
		double t_threshold = 0;
		//Define array to store the coordinates of four points.
		//The two of them lie in the line of the intersection between two faces.
		double[] coordinates = new double[12];
		//Define a varible to store the value of the angle in various situations.
		double angle;
```

```
        //Output arguments as  relative time in the "key" field of OrientationInterpolator node
        System.out.print("<OrientationInterpolator DEF=\"name\" key=\"");
        for (double t = start_time; t < end_time + time_unit; t = t + time_unit) {
            System.out.format("%.4f  ", t);
        }
        System.out.format("\"  \n");
        //Figure out the time thresold.
        t_threshold = time_threshold(start_time, end_time, time_unit);
        //Output arguments in the "keyvalue" field of OrientationInterpolator node
        //The rotation axis can be given directly. The angle of rotation angle in any instant can be calculated and then transformed to arguments.
        System.out.print("keyvalue=\"");
        for (double t = start_time; t < end_time + time_unit; t = t + time_unit) {
            //Calculate the coordiantes of the four points.
            coordinates = coordinates(t, start_time, end_time);
            //Calculate the acute angle between the two faces of rotation.
            angle = faces_angle(coordinates);
            //Transform the angles into arguments we need.
            angle = rotation_angle(angle, t, t_threshold);
            // output rotation axis and angle arguments
            System.out.format(axis + "%.4f  ", angle);
        }
        System.out.print("\"/>");
    }
    //Slect the instant, which is closest to the π /2.
    public double time_threshold(double start_time, double end_time, double time_unit) {
        //Micrify the time unit
        time_unit = time_unit / 10;
        //Define the time threshold, in which the value of anlge is π /2
        double t_threshold = 0;
        //Define and initial the array to store the two consequent angles
        double[] angle = new double[2];
        double[] coordinates = new double[12];
        angle[0] = 0.0;
```

```
        angle[1] = 0.0;
        for (double t = start_time; t < end_time + time_unit; t = t + time_unit) {
            coordinates = coordinates(t, start_time, end_time);
            angle[1] = faces_angle(coordinates);
            if (angle[1] >= angle[0]) {
                angle[0] = angle[1];
            } else {
                t_threshold = t;
                break;
            }
        }
        return t_threshold;
    }
    //Transform the angles into arguments we need with the help of the time threshold
    public double rotation_angle(double angle, double t, double t_threshold) {
        if (t > t_threshold) {
            angle = 3.14159256 - angle;
        }
        return angle;
    }
    //Given the coordinates of four points, calculate the acute angle between the two faces of rotation.
    public double faces_angle(double[] coordinates) {
        double cosA, angle;
        //Define two normal vectors of the two faces of rotation respectively.
        double[] vektor1 = new double[3];
        double[] vektor2 = new double[3];
        //Calculate the normal vectors
        vektor1 = noraml_vektor(coordinates[0], coordinates[1], coordinates[2], coordinates[3], coordinates[4], coordinates[5], coordinates[6], coordinates[7], coordinates[8]);
        vektor2 = noraml_vektor(coordinates[3], coordinates[4], coordinates[5], coordinates[6], coordinates[7], coordinates[8], coordinates[9], coordinates[10], coordinates[11]);
        //Calcute the angle between the two vectors.
```

```
        cosA = (vektor1[0] * vektor2[0] + vektor1[1] * vektor2[1] + vektor1[2] * vektor2[2]) / (sqrt(vektor1[0] * vektor1[0] + vektor1[1] * vektor1[1] + vektor1[2] * vektor1[2]) * sqrt(vektor2[0] * vektor2[0] + vektor2[1] * vektor2[1] + vektor2[2] * vektor2[2]));
        angle = acos(cosA);
        //Transform the angle between two normal vectors into the acute angle between the two faces of rotation.
        if (angle >= 1.57079628) {
            angle = 3.14159625 - angle;
        }
        return angle;
    }
    //Calculate  the normal vectors, given coordinates of the three points that lie in a same plane.
    public double[] noraml_vektor(double x1, double y1, double z1, double x2, double y2, double z2, double x3, double y3, double z3) {
        //Calculate the two vectors lying on the plane.
        double a, b, c, d, e, f;
        a = x1 - x2;
        b = y1 - y2;
        c = z1 - z2;
        d = x1 - x3;
        e = y1 - y3;
        f = z1 - z3;
        //Define and calculate the noraml vectors.
        double[] vektor = new double[3];
        vektor[2] = (a * e - b * d) / (c * d - f * a);
        vektor[1] = 1;
        vektor[0] = (c * e - f * b) / (f * a - c * d);
        if (abs(vektor[0] / vektor[1]) > 10000) {
            vektor[2] = (b * d - e * a) / (e * c - b * f);
            vektor[1] = (f * a - c * d) / (c * e - f * b);
            vektor[0] = 1;
            if (abs(vektor[0] / vektor[2]) > 10000) {
                vektor[2] = 1;
                vektor[0] = (b * f - e * c) / (e * a - b * d);
```

```
                vektor[1] = (a * f - d * c) / (b * d - a * e);
            }
        }
        return vektor;
    }
}
```

6.2 Inside_Reverse_Fold.java

```
/* @author: Jindong Gu's Bachelor Thesis in Wuppertal University.**
** This implementation class inherits the abstract "Origami" class and implements the interface.**
** The interface to calculate coordinates of the four points is implemented according to the motion*/
package yoshizawa_randlett_system;
import static java.lang.Math.cos;
import static java.lang.Math.sin;
import static java.lang.Math.sqrt;
/*This is an instance. It takes the motion of "Inside Reverse Fold" as an example.**
**1.Define an implementation class to inherit the abstract class**
**2.Implements the interface. In other words, calculate the coordinates of four points in a certain instant according to the mathemetic relationships.**
**3.Define the main function to set up the value of the rotation axis and then call the "output_arguments" function.*/
public class Inside_Reverse_Fold extends Origami {
    @Override
    //Implement the interface.
    public double[] coordinates(double t, double start_time, double end_time) {
        //Define the coordinates of four points and an array to store them later.
        double x1, y1, z1, x2, y2, z2, x3, y3, z3, x4, y4, z4;
        double[] coordinates = new double[12];
        //Assign coordinates of already known point.
        double A, A0 = 3.14, R1 = 2, R2 = 0.585786;
        //Figure out the coordinates of three of the points by tansformming of the coordinate system.
        //Point E
        A = (t - start_time) / (end_time - start_time) * A0 / 2;
        x1 = 0;
```

```
y1 = -R1 * sin(A);
z1 = -R1 * cos(A);
//Point B
x2 = 0;
y2 = 0;
z2 = 0;
//Point C
x3 = 1.414213;
y3 = -R2 * sin(A);
z3 = -R2 * cos(A);
/* Calculate the coordiante of the point A acoording to three relationships.
The point A(x, y, z) lies in Z=0 plane because of symmetry (i.e. z=0).
The distance from the point A to the point B is 2 (i.e. x2 +y2 +z2 =4).
The distance from the point A to the point C is 0.8284, which means the equation (x-1.4142) 2 + (y+R2×sin(β)) 2 + (z+ R2×cos(β)) 2 =0.6863.
These equations are solvable manually thanks to the symmetry in origami. Man can also resolve these equations with MATLAB.
*/
//Solve the equations manually.
y4 = y3 * 5.65685 / (2 + y3 * y3);
x4 = sqrt(4 - y4 * y4);
z4 = 0;
//Store the coordinates of the four points in an array.
coordinates[0] = x1;
coordinates[1] = y1;
coordinates[2] = z1;
coordinates[3] = x2;
coordinates[4] = y2;
coordinates[5] = z2;
coordinates[6] = x3;
coordinates[7] = y3;
coordinates[8] = z3;
coordinates[9] = x4;
coordinates[10] = y4;
```

```
        coordinates[11] = z4;
        return coordinates;
    }
    //The main function assign the value of rotation axis and calls the output_arguments function to get cooresponding arguments.
    public static void main(String[] args) {
        Inside_Reverse_Fold test = new Inside_Reverse_Fold();
        test.axis = "-1.414213 0 -0.585786 ";
        test.output_arguments(0.5, 0.8, 0.02);
    }
}
```