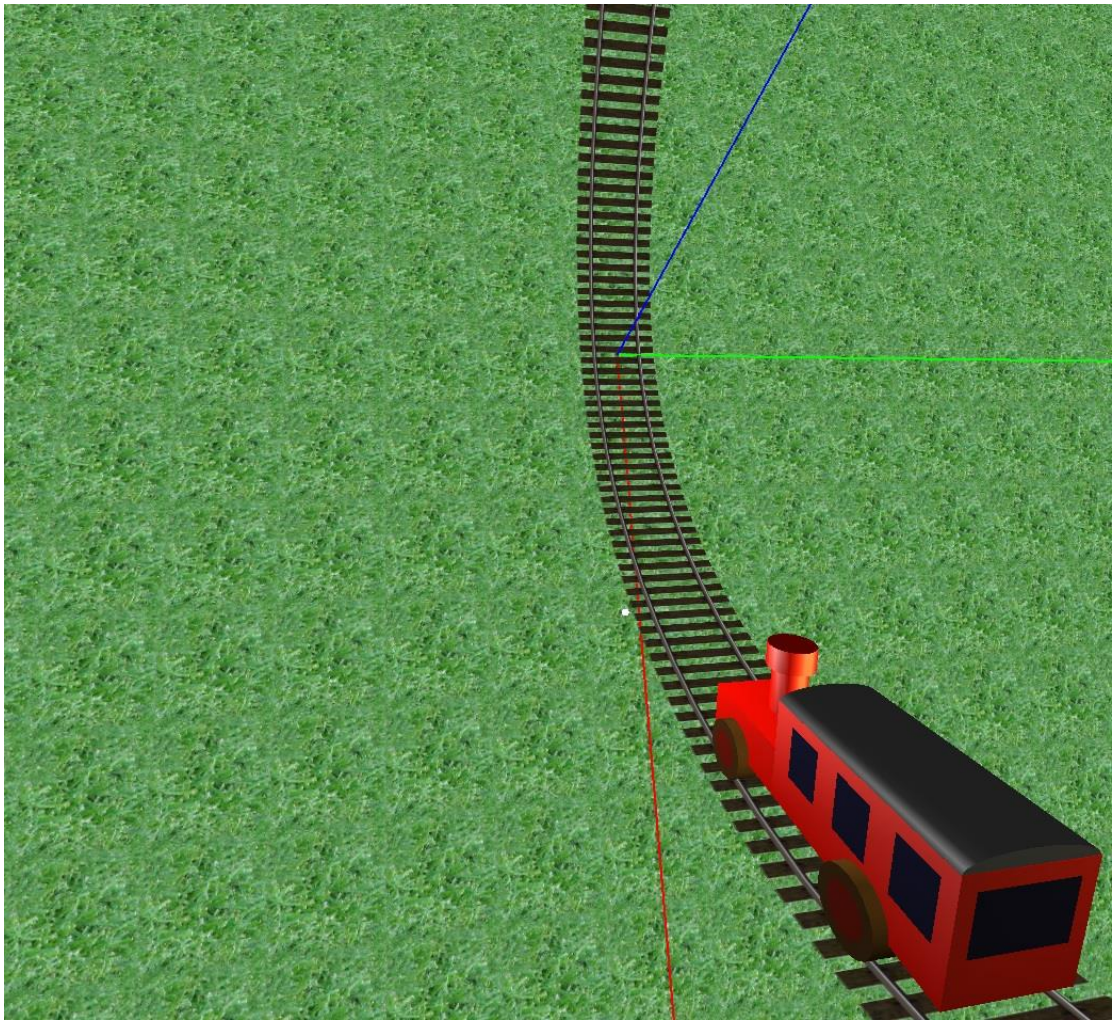
Exercise 4 – Animation



Overview

In this exercise you will implement an OpenGL 3D animation game that will consist of a locomotive moving on a track.

The implementation of this exercise will be an extension of ex3.

Usage

The same as ex3, plus the following:

- Pressing ‘c’ toggles the camera position (explained below).
- Pressing ‘l’ (Bonus) - turns on/off marking the light sources (little spheres will be displayed at the location of the light sources, which will have only an emittance attribute – i.e. not affected by the light source itself).
- Pressing the up and down arrows changes the locomotive's speed (note that the speed can be negative – in which case the locomotive will move backwards).
- Typing a number (greater than zero) and then pressing Enter changes the track.
- Pressing esc closes the application.

Requirements

1. General
    1.1. A locomotive should move on a track.
    1.2. The implementation of ex3 should be used as a starting point.
    1.3. Lighting and materials should be used (instead of glColor) – this is a Bonus.
    1.4. The track should be rendered above a textured plane (e.g. grass field).
    1.5. The track itself should be textured.
    1.6. The locomotive will face the correct direction.
2. View
    2.1. The camera will be placed in one of two positions (toggled by the ‘c’ key):
        2.1.1. As in ex3 – rotated/zoomed using the mouse.
        2.1.2. Behind the locomotive, giving the perception of moving along with the locomotive.
    2.2. In both cases, as before, a perspective projection will be used.
3. Bonus – adding lighting effects (10 points).

Modeling the track course

The track can be modeled by a parametric closed curve using cubic splines. You can design your own, but in the supplied TrackPoints.java file there are three examples. The x and y coordinates of each point should be between -1 and 1. The z coordinate of each point should be non-negative.

Given is a series of n points: $q_1, \dots, q_n$, to find the cubic splines for the track curve use the following constraints:

For each $i < n$:

1. $f_i(0) = q_i$
2. $f_i(1) = f_{i+1}(0)$
3. $f_i'(1) = f_{i+1}'(0)$
4. $f_i''(1) = f_{i+1}''(0)$

For $i = n$

1. $f_n(0) = q_n$
2. $f_n(1) = f_1(0)$
3. $f_n'(1) = f_1'(0)$
4. $f_n''(1) = f_1''(0)$

f_i is a curve of the form: $f_i: [0,1] \to \mathbb{R}^3$ ($f_i(t) = \big(x_i(t), y_i(t), z_i(t)\big)$). Each coordinate of $f_i(t)$ (i.e. $x_i(t), y_i(t), z_i(t)$) is a polynomial function of the form:

$$p(x) = ax^3 + bx^2 + cx + d$$

There are three different polynomials for each coordinate.

You need to solve three systems of linear equations; each system contains 4n variables and 4n equations. You can use the supplied solver Jama-1.0.3.jar file. Here is an example of using the solver:

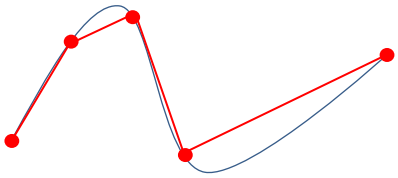
```
/*
 *Solving three variable linear equation system
 * 3x + 2y -  z =  1 ---&gt; Eqn(1)
 * 2x - 2y + 4z = -2 ---&gt; Eqn(2)
 * -x + y/2-  z =  0 ---&gt; Eqn(3)
 */
import Jama.Matrix;
import java.lang.Math.*;
public class Main {
    public Main() {
        //Creating  Arrays Representing Equations
        double[][] lhsArray = {{3, 2, -1}, {2, -2, 4}, {-1, 0.5, -1}};
        double[] rhsArray = {1, -2, 0};
        //Creating Matrix Objects with arrays
        Matrix lhs = new Matrix(lhsArray);
        Matrix rhs = new Matrix(rhsArray, 3);
        //Calculate Solved Matrix
        Matrix ans = lhs.solve(rhs);
        //Printing Answers
        System.out.println("x = " + Math.round(ans.get(0, 0)));
        System.out.println("y = " + Math.round(ans.get(1, 0)));
        System.out.println("z = " + Math.round(ans.get(2, 0)));
    }

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

Once you find the spline, use the following formula to calculate its length:

$$Length(f_i(t)) = \int_0^1 \|f_i'(t)\|dt = \int_0^1 \sqrt{\left(x_i'(t)\right)^2 + \left(y_i'(t)\right)^2 + \left(z_i'(t)\right)^2}\, dt$$

You can estimate the length by using Riemann sum.



The track itself should consist of triangles that would fully tessellate it. The track width should be 0.1. Here are some advices:

1. For every discrete $t \in [0,1)$ in each spline i, calculate $p_0 = f_i(t)$ and $p_1 = f_i(t + \Delta t)$, which are two consequent points on the curve (at the middle of the width of the track). Δt is the discretization step (use the spline length to measure Δt).
2. To calculate the curve's tangent (derivative) vector at each point use this formula:

   $tangent(f_i(t)) = f_i'(t) = (x_i'(t), y_i'(t), z_i'(t))$. Be aware that this vector is not normalized. You should normalize it.

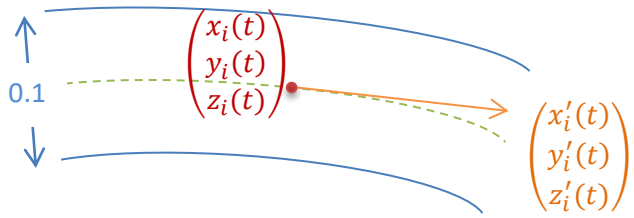
3. To calculate the normalized perpendicular normal at each point use this formula:

$$normal\big(f_i(t)\big) = \frac{\big(f_i'(t) \times f_i''(t)\big) \times f_i'(t)}{\left\|\big(f_i'(t) \times f_i''(t)\big) \times f_i'(t)\right\|}$$

4. You can use the normalized tangent and the normalized normal at each point to find the third axis (think how).

Making the locomotive face the correct direction

In order to rotate the locomotive so it faces the correct direction on the track, you should also use the derivatives of your curve. The locomotive's forward direction should be the normalized $tangent(f_i(t))$. The locomotive's roof (top) direction should be $normal\big(f_i(t)\big)$.



Tip: Maybe it would be easier to calculate the rotation matrix by hand, without explicitly calculating the rotation angle. (Given an orthonormal basis $\{v_1, v_2, v_3\}$, what matrix would rotate the vectors $\{(1,0,0)^T, (0,1,0)^T, (0,0,1)^T\}$ to be aligned with the basis?)

Tip: Make sure that whatever needs to be normalized is normalized.

Lighting (Bonus)

The scene should contain at least two positional light sources. Feel free to select materials as you see fit. You are required to demonstrate specular and diffusive lighting at least once each. You shouldn't use glColor in this exercise.

Lights should follow the locomotive (i.e. like they are attached to it).

Make sure front and back surface faces are what you expect them to be, and adjust normals accordingly. Remember that scaling transformations affects the normals. To handle this you can enable GL_NORMALIZE.

Remember that moving from wireframe to fully rasterized polygons requires you to use the depth buffer.

If your lighting seems to act strangely, check whether your front faces should be defined CW or CCW. Also check the directions of your normals.

Also, you should **display a sphere** at the position of each of the light sources. Their illumination should not be affected by the light sources (diffuse/specular set to 0), and they should be seen by emitting their own illumination. Pressing 'l' will hide these spheres.

Textures

Textures should be used for the track and landscape. Image files are supplied for your convenience (inside the textures folder), but you can use any other images you choose. Files should be placed in such a location that

```
File texFile = new File("texture.jpg" /* or .png */);
```

would find them. In Eclipse this means relative to the project folder. If you have many textures, you can use a subfolder:

```
File texFile906 = new File("tex/texture906.png");
```

Recommended Milestones

Write incrementally. We suggest the following implementation milestones:

1. Read the readme files in the supplied folders.
2. Implement the tracks curve functions and their derivatives and normal by using Cubic Spline method.
3. Tessellate the track with triangles. The track should be visible from the top and from the bottom (place each triangle twice clockwise and counterclockwise).
4. Place the locomotive at the first spline at $t = 0$. Scale it and make it face the right direction.
5. Implement Tack.setCamera(). Make sure that pressing 'c' shows the locomotive from behind.
6. Implement animation by returning true in Tack.isAnimated(), and advancing t inside Tack.render().

7. Increase the locomotive speed by clicking the up arrow key.
8. Decrease (or invert if the speed is zero) the locomotive speed by clicking the down arrow key.
9. The locomotive should move in a constant speed along the track (unless you change it with the arrows). Calculate each spline length to achieve that.
10. By clicking the number keys and then enter you can switch the tracks.
11. Do not build the splines of the track each time you render the track, do it only once and store the results in an appropriate data structure.
12. Compute the length of each spline only once and store the result as well.
13. Read the new Main.java file and understand its functionality.
14. Follow the TODOs instructions within the Track.java file.
15. Check that you didn't forget anything.

Submission

- Submission is in pairs
- Zip file should include
    - All the java source files, including the files you didn't change.
    - The jar files of my locomotive and the Jama.
- A short readme document where you can **briefly** discuss your implementation choices.
- Zip file should be submitted to Moodle.
- Name it:
    - <Ex##> <FirstName1> <FamilyName1> <ID1> <FirstName2> <FamilyName2> <ID2>