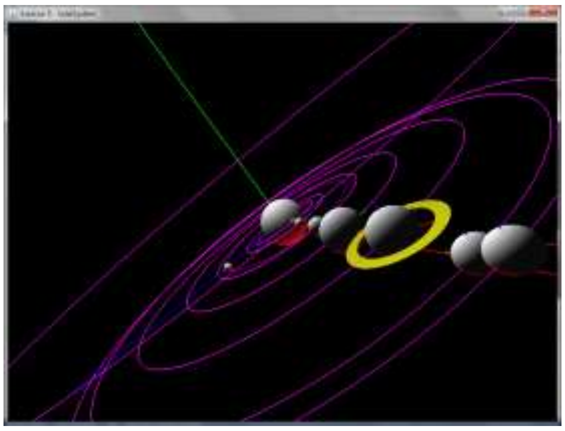
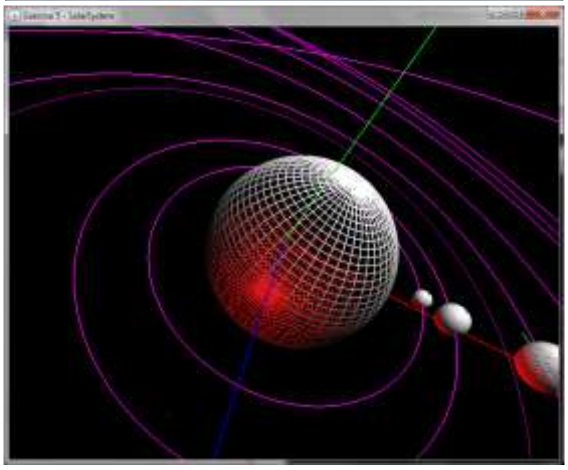
Exercise 5 – Model & Viewer Solar System





Contents

Overview

In this exercise you will practice basic OpenGL techniques in modeling, viewing and lighting. Your goal is to create an application that allows a user to view an OpenGL rendered model. You will have to create a solar system

The model you create here will serve you for the next exercise as well.

You'll get an example jar, which is a simplified version (planets locations and lighting should be a little more complicated). Note that you will need to place it in a folder along with the JOGL dlls.

Usage

- The user can rotate the view around the model by dragging the mouse over the canvas.
- Mouse wheel is used to zoom in and out.
- The user can change the window's size and alter its ratio, without distorting the image.
- Pressing 'p' toggles wireframe and filled polygon modes
- Pressing 'm' displays the next model (if you choose to model more than one).
- Pressing 'a' toggles the xyz axis on/off.
- Pressing 'l' turns on/off the light sources marking (when marking is on, 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).
- The application receives no command line parameters.

Framework

The code you are given consists of the GUI and OpenGL initialization. The App class takes care of the GUI and control keys, while the View class implements GLEventListener and takes care of camera mode and viewing options. All the control settings are written in the app class. Therefore attaching the desired events to the Viewer happens in app. The App also holds all your models. You have an interface class IRenderable which your model should implement.

You need to implement trackball, OpenGL events and create models. Display will be automatically called whenever needed, 30 times a second.

Modeling

You should implement a cube as a test model.

The main model is designing your own cool solar system. In the next exercise it will also be animated.

Building-Blocks

The minimum set of primitives you need to use for modeling are:

- polygons
- Lines (orbits and axis)
- Two GLU quadrics
    - Sphere
    - Disk

Everything in the image above can be obtained using these primitives and affine transformations. Again, you may use any additional drawing method you see fit.

The Solar System:

You should place the 9 planets, the Sun and the Moon. Saturn should be rendered with a ring surrounding it. Don't implement the sizes with the true physical proportions (orbit and plants' radiuses) the true proportions are hard to visualize, but make sure that what is bigger and farther, stays that way.

You should render circular orbits and place the planets somewhere on them. In the example jar the Earth and the Moon were shifted along the Earth's orbit, you are required to shift all planets to some arbitrary initial positions on their orbits.

The planets are tilted w.r.t the whole solar system plane (**orbital inclination)** and w.r.t their orbit plane (**axis inclination**). You should implement these inclinations. The physical parameters for the inclination are in Appendix B. Each planet should be rendered with its axis in the Axis mode (control key 'a'). This is how you can check if the orientation is correct.

Using OpenGL matrix stack for modeling

Your model must be modular and constructed in a recursive manner **using OpenGL's matrix stack**. Following is a list of the features in our model. You may choose to use different features, but they should use all the elements that we used.

Viewing

The user should be able to rotate the model using the mouse. The viewing method you should implement is called "Virtual Trackball" and is discussed in Appendix A of this document. Basically, it involves projecting the mouse position before dragging and after dragging onto a sphere, and then computing the rotation needed to transform between the two points. This transformation should then be performed on the model.

Code Flow Explanation of Viewing

We rotate the view by changing the modelview matrix- this is actually part of the view transformation: Remember the flow [projection*view*world*model]*object coordinates.

This happens in the beginning of the **display** method. This is a method of GLEventListener overrided by the Viewer. The method **setupCamera** (also a method of the Viewer class) is called in the beginning of display. **You are responsible for implementing it.**

Now we want to use mouse position to set the rotation value that we insert into the viewing matrix. Here you should use the trackball method (in setupCamera).

The GUI gives us the mouse position. The event we're dealing with for rotating the view is **mouseDragged.** This is implemented by adding a **MouseMotionListener** to the Viewer and this is happening in the app class. The mouseDragged(MouseEvent e) method is overridden and we get the mouse position through e.getPoint();

The position is passed to the storeTrackBall method of the Viewer class, which is also implemented for you and the position is saved to members of the Viewer class for your convenience.

Zoom

Zoom should be achieved by moving the camera closer to the model. Note that this is possible only when perspective projection is used. Think where should you insert it in the transformation flow.

The zoom is derived from the mouse wheel event in a similar way explained in the Viewing part:

- attaching MouseWheelListener in the app to the Viewer class
- overriding mouseWheelMoved(MouseWheelEvent e)
- calling the method zoom (implemented for you!) of the Viewer class with the rotation derived from the scrolling wheel - e.getWheelRotation()
- saving the amount of zooming we want (converting mouse wheel rotation to zoom units) to a member of the Viewer class.

Lighting

The scene should contain at least two positional light sources. Make the planets nice and colorful (the jar example is pretty dull). You are required to demonstrate both specular and diffusive lighting.

Make sure front and face surfaces are what you expect them to be, and adjust their normals accordingly. Remember that scaling transformations affect the normal direction. To handle this you can to enable GL_NORMALIZE.

Remember that moving from wireframe to fully rasterized polygons requires you to adjust how the depth buffer is handled.

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 themselves (set the diffuse/specular material property to 0), and they should be seen by emitting their own illumination. Pressing 'l' will show/hide these spheres.

Two Sided Lighting

As in this assignment you are required to activate back face culling (see below), there is no use in two sided lighting. However, note that this can help you for debugging purposes, before you activate culling. To achieve this you can enable the light model property GL_LIGHT_MODEL_TWO_SIDE. When this is enabled OpenGL calculates a different normal for the back side (i.e. the inverted front's normal).

Projection

You should use a perspective projection to render the scene. You should apply the required transformations for your model to be displayed in the center of the window, in an appropriate scale.

Additional requirements

Back face culling should remain enabled at all times. This will make polygons transparent from their back side. You may alter the definition of what is considered the back side during rendering by changing GL_CW/GL_CCW.

Additional Models (Bonus)

In addition to the solar system, you can design as many models as you want (cool polygonal asteroids, additional moons, multiple rings, spaceships, feel free to think creatively). Bonus points will be given to the most impressive models over all submissions. The only rule is that no external resource files are allowed (e.g. mesh files).

Recommended Milestones

Write incrementally. We suggest the following implementation milestones:

- Import and run the given code. Go through it. Read the TODOs.
- Drawing a cube at the center of the axes. Make sure you see it.
- Implement the trackball viewing (see Appendix A).
- Set perspective projection at reshape. Rotate the cube and make sure its back faces are smaller than the front ones (as consequence of perspective).
- Activate back face culling and make the necessary corrections.
- Time for art – model a solar system! Add an element at a time.
    - Start with one planet
    - Render its axes
    - Add its orbit
    - Add inclinations
    - Add a the Saturn ring
- Add lights and set materials
- Implement the rest of the control functions
- Read the assignment again and make sure you didn't forget anything

Tips

- Write a method to draw a planet
- IRenderable.control() can be used to pass user keystrokes to our model.
- <u>There are some places in the supplied code that are meant for ex6. You can ignore them for now.</u>

Submission

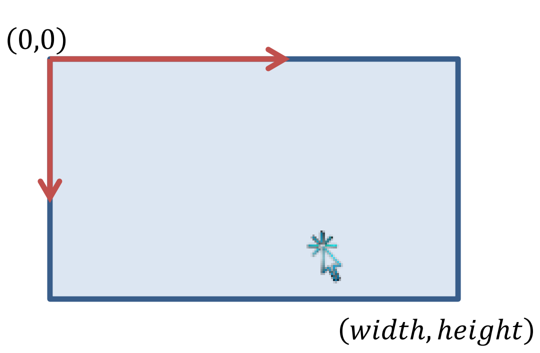
- Submission is in pairs
- Zip file should include
    - All the java source files, including the files you didn't change in a JAR named "ex5-src.jar".
    - Compiled runnable JAR file named "ex5.jar"
        - This JAR should run after we place JOGL DLLs in its directory
        - Make sure the JAR doesn't depend on absolute paths – test it on another machine before submitting
        - **Points will be taken off for any JAR that fails to run!**
- 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>
- Submission deadline is 24/05/2015
- Before submission be sure to check the discussion board for updates

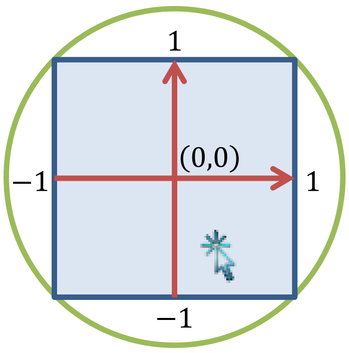
Appendix A – Virtual Trackball

The following is a short description of the trackball mechanism you need to implement.

Step 1 – Transform Canvas Coordinates to View Plane

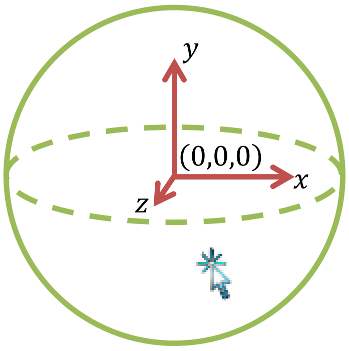
Given a 2D point on the canvas we need to find its projection on a sphere. First convert the 2D canvas point to a 2D point on a viewing plane contained in the sphere. This is accomplished by: $x = \frac{2p_x}{width} - 1, y = 1 - \frac{2p_y}{height}$





Step 2 – Project View Plane Coordinate onto Sphere

Given the view plane's 2D coordinates compute their spherical z-value by substituting them in the sphere's formula: $z = \sqrt{2 - x^2 - y^2}$ (make sure that $2 - x^2 - y^2 \geq 0$).



Step 3 – Compute Rotation

Given the current and previous 3D vectors we need to find a rotation transformation that rotates between the two. A rotation is defined using a rotation axis and rotation angle. Use vector calculus to obtain these. Note: pay attention to degrees/radians.

Step 4 – Rotate Model

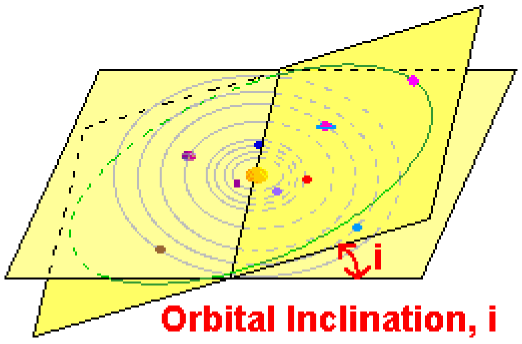
Rotate the world about the origin using the ModelView matrix. Note that the rotation is cumulative, so it would be easier for you to store the rotation matrix between redraws. Order of the rotations matters. Denoting the cumulated (stored) rotation R_s and the newly calculated rotation R_n, the new rotation matrix should be $R_n R_s$ (meaning that you should first call $\text{glRotate}(R_n)$, and then $\text{glRotate}(R_s)$).

Appendix B – Solar System Inclination Details

Orbital Inclination

http://www.allaboutspace.com/subjects/astronomy/glossary/Inclination.shtml

Inclination is a measure of how the plane of an orbit is tilted with respect to the plane of the ecliptic. An orbit that lies on the plane of the ecliptic would have an inclination of zero degrees; higher numbers indicate more inclined orbits.

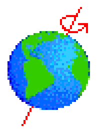


Pluto is the planet in our solar system with the most inclined orbit.

Orbital Inclination of the Planets in our Solar System

Planet	Orbital Inclination
Mercury	7°
Venus	3.39°
Earth	0°
Mars	1.85°
Jupiter	1.3°
Saturn	2.49°
Uranus	0.77°
Neptune	1.77°
Pluto	17.2°

Axial Tilt



http://www.allaboutspace.com/subjects/astronomy/glossary/index.shtml

Axial tilt, or obliquity to the ecliptic, is the tilt of a planet's axis from perpendicular to the plane of the ecliptic. Another way of looking at it is the angle between the plane of the planet's orbit and that of the equator.

Planet	**Axial Tilt (Obliquity to the Ecliptic)**
Mercury	2°
Venus	2°
Earth	23.45°
Mars	24°
Jupiter	3.1°
Saturn	26.7°
Uranus	97.9°
Neptune	28.8°
Pluto	57.5°