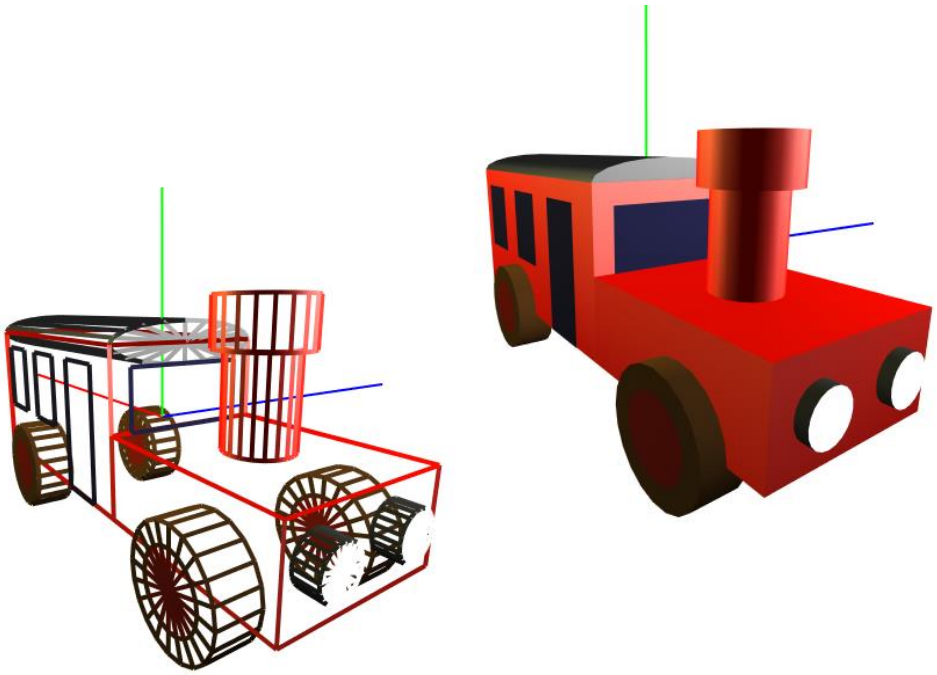
Exercise 3 – Model & Viewer



Overview

In this exercise you will practice basic OpenGL techniques in modeling, viewing and projecting. Your goal is to create an application that allows a user to view an OpenGL rendered model. You will have to create at least one model – a locomotive.

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

A reference executable can be downloaded from Moodle. Note that you will need to place it in a folder along with the JOGL dlls (Windows) or jnilibs (OS X).

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.
- Pressing 'a' turns the xyz axis on/off.
- The application receives no command line parameters.

Modeling

There are endless ways to model a locomotive. Here we require a minimum level of details. You may however embellish the model with as many additional details you see fit.

Building-Blocks

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

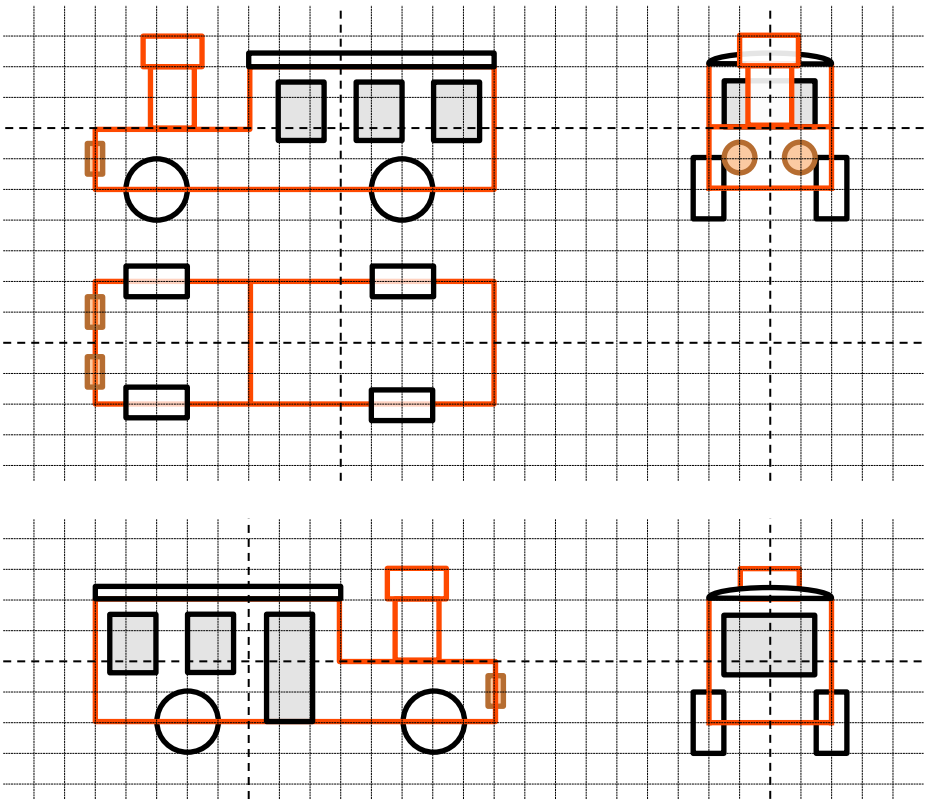
- At least one type of polygons (triangles, quads, polygons etc.)
- GLU quadrics
    - Cylinder
    - Disk

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

Structure

Your model must be modular and constructed in a recursive manner using OpenGL's matrix stack. Following is a list of the parts in our model. These are only a suggestion, but you do have to use the building blocks mentioned above.

- Locomotive – A chassis with a door, windows, wheels, front lights, and a chimney.
    - Chassis – The body of the locomotive. Filled polygons.
        - Door – Filled polygon.
        - Windows – Filled polygons.
    - Wheels
        - Wheel – Cylinder bounded by disks.
    - Lights
        - Front light – Cylinder bounded by disk.
    - Roof – Non-uniformly scaled cylinder bounded by disks.
    - Chimney – Two cylinders place on top of each other bounded by disks.



Symmetry

A front light is rendered twice – once on the left side and once on the right. A wheel is rendered four times. The left side of the chassis is almost identical to the right, mirrored (except that on the right there's a door and on the left there's a window). You should only model each of them once. Mirroring in rendering can be achieved using a negative scaling transformation.

Note that negative scaling can change the direction polygons are facing. You can solve this by alternating what is considered a front face using glFrontFace.

Alternatively, depending on your model, you might be able to achieve the same result by using simple rotations and translations.

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. Basically it involves projecting the mouse before dragging and after dragging positions on a sphere, and then computing the rotation needed to transform between the two. This transformation then should be performed on the model.

Zoom

Zoom should be achieved by moving the camera closer to the model. Note that this is possible only when a perspective projection is used.

Lighting

In this exercise you can use glColor to set a uniform color for each face or block. In the next exercise you will change this by adding light sources and material definitions that will make the locomotive's surface look prettier.

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 though alter the definition of the back side (GL_CW/GL_CCW).

Framework

The code you are given consists of the GUI and OpenGL initialization. You only need to implement trackball, OpenGL events and model.

Additional Models (Bonus)

In addition to the locomotive you can design as many models as you want. Bonus points will be given to impressive models. The only rule is that no external geometry files are allowed (e.g. mesh files).

Recommended Milestones

Write incrementally. We suggest the following implementation milestones:

1. Import and run the given code. Go through it. Read the TODOs.
2. Extend the Empty model by drawing a cube at the center of the axes (like RGB cube example from class). Make sure you see it.
3. Implement the trackball viewing (see Appendix A). Note that this is different from the method used in the RGB cube example.
4. Set perspective projection at reshape. Rotate the cube and make sure its back faces are smaller than the front ones (as consequence of perspective).
5. Activate back face culling and make the necessary corrections.

6. Time for art – model a locomotive! Add an element at a time. If you don't want to use the supplied sketch, try to build a model in your head+on paper first, and then find where approximately the vertices should be positioned. Fine tune your result.
7. Implement the rest of the control functions.
8. Read the assignment again and make sure you didn't forget anything.

Tips

- Write a method to draw each feature (e.g. drawChassis, drawWheel, etc.).
    - Before calling each of these push the current ModelView matrix.
    - i.e. each feature should adhere to a local coordinate system.
- IRenderable.control() can be used to pass user keystrokes to our model.
- During design, it might be easier to use an orthographic projection.
- Remember that polygons have to be convex to work with OpenGL.
- There are some places in the supplied code that are meant for ex6. You can ignore them for now.

Submission

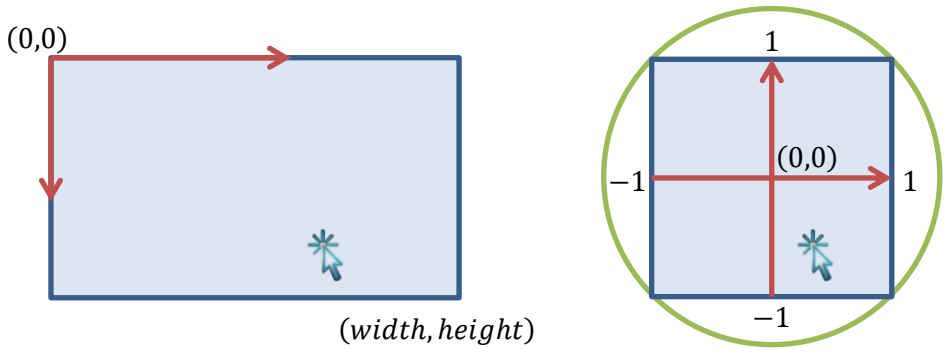
- Submission is in pairs
- Zip file should include
    - All the java source files, including the files you didn't change in a ZIP named "ex3-src.zip".
    - Compiled runnable JAR file named "ex3.jar"
        - This JAR should run after we place JOGL DLLs in its directory
        - Make sure the JAR 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>
- 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.

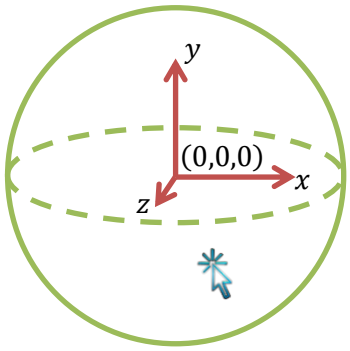
Step 1 – Transform Canvas Coordinates to View Plane

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 $\text{glRotate}(R_s)$).