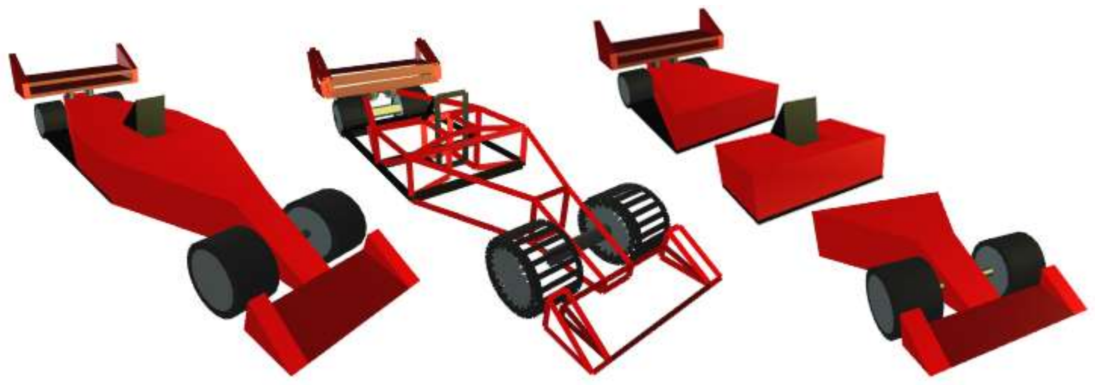
Exercise 5 – Model & Viewer



Overview

Professor Speed from the **International Driving Center (IDC)** asked Mr. Lazy to model a racing car for him. The car is expected to compete in the next Formula 1 racing competition. Mr. Lazy started working on the assignment, but he got lazy and needs your help finishing the job, otherwise he will be fired from his.

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 a model of a racing car.

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. Make sure to use the wireframe mode - it helps better understand the parts that make each car part.
- 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 F1 car. 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:

- **SkewedBox** : You will need to implement this primitive. It will be our main building block, you need to understand how it works, and how each skewedbox is rendered. Notice how the base of the skewed-box is always on the xz-plane.
- **GLU quadrics -** more information about GLU can be found in the recitations.
    - 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. You need to use the building blocks mentioned above, in order to obtain the same results.

- F1Car – This is the whole car we want to render. It consists of the following main three parts:
    - Back.java – The back body of the car. This part of the car is implemented, and can be used for reference when working other parts (note that this part renders a PairOfWheels, which still need to be implemented by you). This part is built from:
        - Spolier.java – A spoiler for the car which consists of rods and skewed boxes (**Implemented**).
        - PairOfWheels.java – Two wheels connected with a rod **(not implemented)**.
    - Center.java - The center of the car body - It is built from skewed boxes only. Notice that some of the skewed-boxes need to be rotated in order to obtain the desired model.
    - Front.java - The front of the car body - It is built from:
        - PairOfWheels.java - Two wheel connected with a rod
        - Skewed boxes - The hood is split into two parts which are skewed-boxes connected to each other. The front also has a bumper, which consists of three skewed boxes.

To make life easier, Mr. Lazy wrote down a list of parameters that can be used when you model the car. These parameters are part of the Specification.java class, and it contains details about the Height, Depth, and Width of each element in the car. The naming convention used is as follow:

- Length is measured relative to the x-axis (Red axis)

- Height is measured relative to the y-axis (Green axis)
- Depth is measured relative to the z-axis (Blue axis)

Make sure that you create the minimum amount of objects required to render each part. For example; you don’t need two Wheel objects in order to render the wheels of the front car. Use one Wheel object along with Model transformation.

Symmetry

Our model is symmetric along the z-axis, meaning the car is symmetric about the xy plane. Make sure to model your car such that all elements are symmetric about the xy-plane. For example, when you render a skew-box, the depth along the z-axis and the negative z-axis should be equal - this will make life easier for you.

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 car’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.

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 SkewedBox at the center of the axes. The SkewedBox should be placed on the xz-plane (this will make life easier for you when you want to later model the whole car). An example of a SkewedBox can be found in the provided jar.
3. Implement the trackball viewing (see Appendix A).
4. Set perspective projection at reshape. Rotate the SkewedBox 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 F1Car! Add an element at a time. If you don't want to use the supplied parameters, 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. Read the assignment again and make sure you didn't forget anything.

Tips

- Use Bottom up approach. First render basic elements (Skewed box, a wheel). Then start implementing complex models such as pair of wheels. Keep rendering complex models up until rendering the final car.
- 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

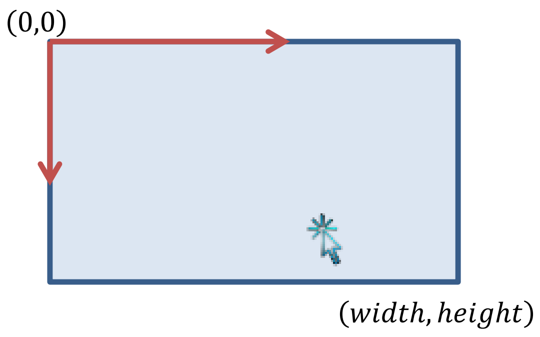
- Submission is in pairs
- Zip file should include
  - All the java source files, including the files you didn't change in a ZIP named "ex5-src.zip".
  - Compiled runnable JAR file named "ex5.jar"
    - This JAR should run after we place JOGL DLLs in its directory
    - Make sure the JAR doesn't depend 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>
- Before submission be sure to check for updates on moodle.

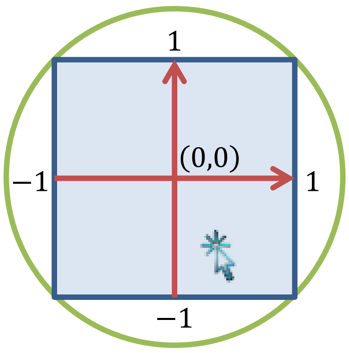
Appendix A – Virtual Trackball

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

Step 1 – Transform Canvas Coordinates to View Plane

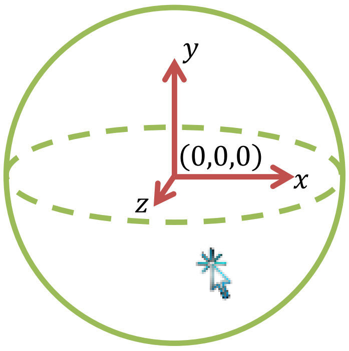
Given a 2D point on the canvas we need to find its projection on a sphere. First convert the 2D canvas point to a 2D point on a viewing plane contained in the sphere. This is accomplished by: $x = \frac{2p_x}{width} - 1, y = 1 - \frac{2p_y}{height}$





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 glRotate(R_n), and then glRotate(R_s)).