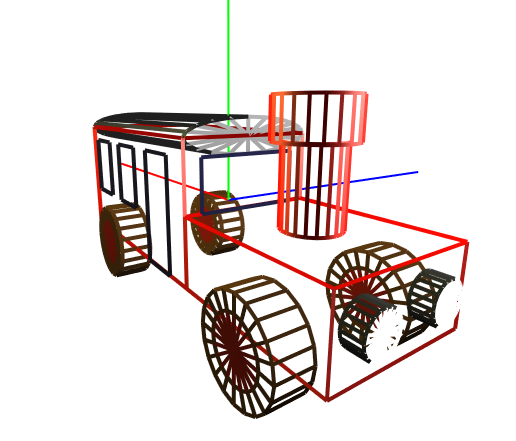
****Exercise 5 – Modeling with OpenGL

# 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 is available on Moodle.

# 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. Zooming is not relevant when the projection matrix used is the orthographic projection (see projection section for more details)
* The user can change the window’s size and alter its ratio, without distorting the image.
* Pressing ‘w’ 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.

To run the provided jar, simply download it and click on it (for non-mac users). For mac users, you need to run the jar file through the command line:

$$: java -XstartOnFirstThread -jar EX5-2021-macOS.jar

# 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, or alternatively, you could build a 3D car model that is at least as impressive as our Locomotive model.

## Building-Blocks

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

* A Box: a 3D box centered at the origin.
* GLU shapes
  + Cylinder
  + Disk

Our model is basically built from these 3D primitives, along with the necessary affine transformations.

## 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.
  + The body of the locomotive is divided into two components. A front body and a back body.
    - Front Body: The front body is composed of a chassis, two wheels, two car lights and a chimney.
    - Back Body: The back body of the locomotive is composed of a chassis, a roof, two wheels, windows and doors
  + Wheel: a wheel is a cylinder bounded by disks. Note that each wheel face is composed of two disks, a disk that represents the Wheel Rim, and a disk that represent the wheel tire.
  + Front Lights: The front lights are composed of a cylinder bounded by disks. Note that we use different colors for the car light case and the light “glass”.
  + Roof – Non-uniformly scaled cylinder bounded by disks.
  + Chimney – Two cylinders placed on top of each other bounded by disks (implemented).
  + Windows and door: these objects can be simply modeled as a quadratic polygon.

### Model Specifications and Unit-measure

Inside Specifcation.java we include the unit values we used to render the model. The specification of each part of the model is provided there. Specifically, we use the keyword “DEPTH” to represent lengths along the Z-axis, “WIDTH” to represent lengths along the x-axis and “HEIGHT” to represent lengths along the y-axis. All lengths are based on the field called BASE\_UNIT, which represents a square in the sketch you see above.

### Symmetry

You can use the symmetric property of the model when building different parts of the locomotive. More specifically, 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).

# Viewing

The user should be able to rotate the model using the mouse (specifically, dragging the mouse while clicking it). The viewing module involves projecting the positions of the mouse before and after the click on a sphere, and then computing the rotation needed to transform between the two. This transformation is then performed on the model. This module is already implemented. You can find further details about the implementation in the appendix.

## Zoom

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

# Surface Color

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 more realistic. We recommend calling glColor via static methods defined in the Materials.java class. In the next exercise we will modify these methods to define the material properties of each surface. Remember, the current color OpenGL uses remains the same until invoking glColor. Therefore, before rendering any component, make sure you set the current color of OpenGL to your desired color.

# 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. The projection happens in the reshape function inside Viewer.java. In the initial code we provided, an orthographic projection is used. Note that in orthographic projections, zooming does not make sense – so you should avoid playing with the mouse scrolling wheel when orthographic projection is used. In the final submission you should change the projection to perspective projection.

# Additional requirements

Back face culling should remain enabled at all times. This will make polygons transparent from their back side. You may disable back face culling while working on your model. However, for the final submission you must re-enable back face culling and make sure that all of your surfaces are defined in such a way that the front sides are those that are visible to the viewer. If you find that one of the faces you defined is backfacing the viewer, then you can alternate the order of the vertices inside the relevant glBegin, glEnd block. We recommend keeping the face culling enabled all the time and to be consistent during your implementation.

# Framework

The code you are given consists of the GUI and OpenGL initialization. You only need to implement drawing related code-blocks. See TODO’s in the code for further guidance.

# Improve the appearance of the Model (up to 5 points bonus)

In addition to the basic locomotive model, you can design additional features (such as back lights, exhausts) or anything else you see fit. There will be up-to 5 points bonuses, for creative additional features.

# Recommended Milestones

Design the 3D model in a bottom-up fashion. We suggest the following implementation milestones:

1. Read the TODO’s and make sure you understand the code. We numbered the TODOs in a way that is closely related to the recommended milestones below.
2. Complete the Implementation of the Box class. You can also re-implement from scratch if it easier for you. This class will be used to render the Front and back chassis of the locomotive model.
3. Complete the Wheel 3D model.
4. Complete the front body of the locomotive model:
   1. The front body is composed of two front lights, two wheels, a chimney, and a Box that represents the front chassis.
   2. The chimney model is already implemented. Please understand its units in order to combine it with other models.
5. Complete the back body of the locomotive model:
   1. Complete the roof model.
   2. Combine the roof, along with the box model and two wheels to render the back body (excluding the windows and doors)
   3. Add the windows and the doors to the model (including front, back and side windows). It is recommended to place the windows with a little offset relative to the back chassis. This can be achieved by adding the offset Specification.EPS to the position of the window (see tips below).

## Tips

* During design, it might be easier to use an orthographic projection.
* Remember that polygons have to be convex to work with OpenGL.
* Pay attention to the order in which you present the polygons vertices.
* There are some places in the supplied code that are meant for ex6. You can ignore them for now.
* The windows are placed on the back-body chassis. Placing the window exactly on the chassis surface could lead to unwanted results, because points on both the window and chassis will be at the same distance from the camera, and the Z-Buffer algorithm may not render the window surface. To avoid this, add Specification.EPS offset to the window's position relative to the chassis:



* Use the wireframe mode (by pressing “W”) to see how different parts are modeled in the provided jar file.
* GLU Shapes are available inside the package edu.cg.util.glu. See Recitation 8 for more details about the different GLU shapes.
* GLU helper functions such as gluLookAt and gluPerspective are defined as static methods inside edu.cg.util.glu.Project.java.

# Submission

* Submission is in pairs
* Zip file should include
  + Your project source folder only, excluding (lib) folder.
* A short readme document if you decide to model the car differently from the instructions – or if you decide to implement the bonus section.
* 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 [Optional Self-reading]

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:



### 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: (make sure that ).



### 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. The order of rotations matters. Denoting the cumulated (stored) rotation and the newly calculated rotation , the new rotation matrix should be (meaning that you should first call , and then ).