CS 355 Lab #6: Transformation Hierarchies with OpenGL

Overview

In this lab, you will continue to use the OpenGL package and build off your completed code for Lab #5. Now that you are familiar with OpenGL and can setup the initial 3D projection, you can begin using some of the more powerful features.

In class, we have talked about the power of using transformations in designing, manipulating, and moving 3D models. In this lab, you will use transformations hierarchies to build a street of houses. Additionally, you will use transformation hierarchies to make animation of a car driving down the street, with wheels turning.

User Interface

The user interface should be identical to the one implemented in Lab #5. The same keys should cause the same movements. The only difference is that the H key returns to home position, but it also resets the time (see the Time section).

Scene

In addition to the house model that was provided in Lab #5, you also will be provided with two additional models: a car body model and a tire model. You will need these new models to complete the assignment.

The scene that you need to design for this lab is a street of houses with a car driving down the road. As the car moves, the tires should rotate appropriately. The location of the houses and car do not matter as long as the TAs are able to navigate the street easily and see that you are able to apply both translation and rotation matrices appropriately.

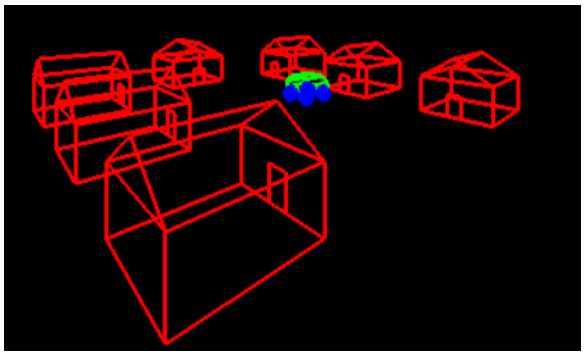

Figure 1: Possible Street Design

OpenGL Commands

As you saw in the last lab, we are using a very small subset of the available commands of OpenGL. The focus of this lab is using transformation hierarchies appropriately, so the two main functions you will need to add to your arsenal are `glPushMatrix()` and `glPopMatrix()`

glPushMatrix

OpenGL has the ability to keep a stack of matrices that it will apply to drawing operations. As you can imagine, this is extremely helpful to implement a transformation hierarchy. However, because OpenGL is imperative, the order of operations can be confusing.

The `glPushMatrix()` command must be called before the appropriate transformations are called. These transformations are then applied to the matrix that has been pushed on the stack. Each successive `glPushMatrix()` call clones the current matrix, places it on top of the stack, then applies the next transformation commands.

Here is some example code to illustrate the point:

```
glPushMatrix()           # Copies current matrix, places it on stack
glTranslatef(50,0,0)     # Applies translation to current matrix
glRotatef(10,0,1,0)      # Applies rotation to current matrix

drawSomething()          # Draws with transformations of current

glPushMatrix()           # Copies current matrix, places it on stack

glTranslate(0,25,0)      # Applies translation to current matrix

drawSomething()          # Draws with transformations of current

glPopMatrix()            # Returns the top matrix of the stack and
                         # sets it to the current matrix.
                         # Destructive Call
```

Also, remember that the matrix operations go right to left, so the last transformation called will be the first one applied to the object.

glPopMatrix

The `glPopMatrix()` command allows you to take transformations off the stack so that you can start from a previous state. Again, keep in mind that OpenGL is imperative, so `glPopMatrix()` does not return the top matrix on the stack. It simply sets the top matrix of stack to the current matrix. It also deletes that matrix from the stack. Keep this in mind as you design your code.

Time

In the previous lab, we had the display update anytime there was a key pressed, which in turn changed what needed to be displayed. However, since there is an animation in this lab (car moving, wheels rotating), you will need to keep track of time and update the display periodically.

The simplest way to do this is using OpenGL's `glutTimerFunc` command and place it right before `glutMainLoop`. You can look up documentation for this function at

http://pyopengl.sourceforge.net/documentation/manual-3.0/glutTimerFunc.html.

Submitting Your Lab

Your code should be contained inside a single .py file. To submit the lab, simply submit this file through Learning Suite. If for some reason you used multiple files, zip these files together before submission. If you need to add any special instruction, you can add them there in the notes when you submit.

Rubric

- Correct rendering of multiple houses along a street (30 points)
- Correct rendering of a car with tires in correct locations (30 points)
- Correct animation of car moving with tires rotating (30 points)
- Generally correct behavior otherwise (10 points)

  TOTAL: 100 points