CS 355 Lab #7: Implementing 3D Perspective

Overview

Now that you're familiar with the basic pipeline for 3D geometry, it's time to do it yourself. In this lab, you will implement the world-to-camera, projection, and viewport transformations that OpenGL did for you in the last lab.

We will be using a package called Pygame, which will allow us to create a viewing window and draw lines on the screen. As before, you will be given a static model of the house, car, and tire that you may use.

The goal of this lab is to implement all of the functionality of Labs #5 and #6 using your own transformation matrices.

User Interface

The user interface is identical to the interface introduced in Lab #5. However, this lab will omit orthographic projection, so you may also omit responding to the ”o” and ”p” keys.

Implementation Notes

Now that we are not using OpenGL, there are a few ideas that you will need to consider when implementing your code. It will be helpful to think in terms of a model-view-controller framework.

Model

For this lab, the 3D ”model” is static. We have provided the same house, car, and tire models that you saw in Lab #6. However, there models are described in our own `Line3D` and `Point3D` classes and stored in a simple array. It will be your job to figure out how to effectively tell Pygame to draw these lines after they have gone through the proper transformations.

Controller

Because we are using a Pygame window for display, we will also need to use Pygame's keyboard listener code. A sample is provided in the lab file. You will need to implement the same camera movements as in Lab #5. Note that OpenGL handles the z-direction in a unique way, so some of your movement code might make you move in the opposite direction than that which is intended. Adjust your code as needed to fit the original specs.

View

This part is the real heart of the lab. You should render the 3D model by using 2D line-drawing commands after first determining the projected 2D locations of each 3D line's endpoints. You should determine these projected 2D locations by implementing each of the pipeline stages we've talked about:

- Convert the 3D (X, Y, Z) coordinates to 4-element $(X, Y, Z, 1)$ homogenouse coordinates.
- Build a *single matrix* that converts from world to camera coordinates. As you did in Lab #5, this is the result of concatenating a translation matrix and a rotation matrix. *It is highly recommended that you use Numpy for storing and working with these matrices.*
- Apply this matrix to the 3D homogeneous world-space point to get a 3D homogeneous camera-space point.
- Build a clip matrix as discussed in class. Pick appropriate parameters for the $zoom_x$, $zoom_y$, near plane distance n, and far plane distance f.
- Apply this clip matrix to the 3D homogenous camera-space point to get 3D homogenous points in clip space.
- Apply the clipping tests described in class. Reject a line if *both* points fail the same view frustrum test **OR** if *either* endpoint fails the near plane test.
- Apply perspective by normalizing the 3D homogeneous clip-space coordinate to get the $(x/w, y/w)$ location of the point in canonical screen space.
- Apply a viewport transformation to map the canonical space to the screen space (512×512).
- Draw the line on the screen.

Scene

To receive full credit, you must render a street of houses and at least one car with tires located in the appropriate places. The locations 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.

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.

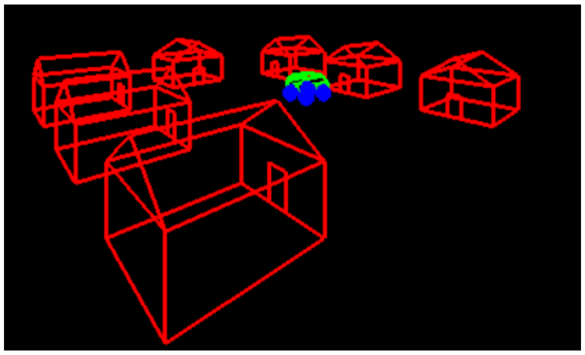

Figure 1: Possible Street Design

Rubric

- Correct rendering of the wireframe house with perspective (50 points)
- Correct navigation relative to current position and orientation (20 points)
- Correct clipping (20 points)
- Correct rendering of multiple houses along street (30 points)
- Correct rendering of a car along street (half credit for doing it without hierarchical transformations) (20 points)
- Generally correct behavior otherwise (10 points)

  TOTAL: 150 points

Extension

If you have your own .obj files that you would like to display (for fun), we have provided a loader for them. Simply call `loadOBJ(filename)` to load your .obj file as a list of `Line3D` objects. This should allow you to easily display your model in a similar fashion to the house, car, and tire models. Of course, this is not required to receive full credit on the lab.