CMSC 435 / 634 Introduction to Computer Graphics

Project Assignment 2: Viewing (Due Oct 1st 11 pm)

Goals of this project: Understand viewing matrix setup in computer graphics and OpenGL. Once this project is completed, you will be able to display 3D objects oriented in any way and viewed from any position.

Introduction

In order to visualize a complex 3-dimensional scene, thousands of tiny triangles must be drawn to the screen. It would be senseless to require the manual placement of each one of these triangles; instead, we usually define the scene in terms of the various primitives that compose it (your project 1). Even better, we allow for primitives to be grouped (e.g., through scene graph), and then we reference those groups as user-defined primitives. You can then move the primitives in the scene.

In this viewing project, you use results from your project 1 to build a driving simulator or a game environment where the character can move around. Each object will have certain properties needed for rendering (color, for instance), and a transformation that will move vertices and normals from object space into world space for the specific object.

This documentation is verbose and even redundant, but not complicated.

Reading Chapter 3 in the OpenGL Redbook will help this assignment tremendously.

Requirements:

--- Modeling:

Implement the scene drawn on the left using code from your project 1. The scene will include a character (you do this and can reuse the robot from project 1), a terrain (provided), and some mountains (using cones, provided). The example code will include a terrain and the mountains. It is your obligation to render the character using shaded primitives (in this case, you will need to do three things to implement the character:

- change your triangle primitives to surface rendering,
- enable depth test, and
- paint the character with colors.

Modeling will include two modes: wireframe (from project 1) and shaded (this project). Pressing “m” will toggle between these two modes.

--- Viewpoint manipulation:

You will modify your code to interactively drive together with the character over the terrain using keyboard and mouse controls. Motion will use four keys: holding 'w'

will accelerate you in whatever direction your car is currently pointing, 's' will accelerate you backwards (first slowing you down, then going into reverse). 'a' and 'd' will turn your character to the left and right. Moving the mouse left, right, up and down should rotate your view relative to your current driving direction, but not change your direction of motion. During any motion, you should remain a constant height above the landscape. All motions should be computed in terms of rates (per second or per millisecond), then scaled by the elapsed time between frames.

To figure out the height of your viewpoint, consider just the horizontal position of your vehicle. From that position, you will compute which triangle you are in, and your barycentric position within that triangle. Use barycentric interpolation of the vertex elevations to compute the surface height at your position, then add a constant offset to place you above the surface.

Make the landscape effectively infinite by rendering eight extra offset copies of the base terrain. Hide the repetition by turning on fog with glEnable(GL_FOG), and using gl_Fog() to set the fog parameters (the man glFog command will give you the options). Use the 'f' key to toggle the fog on and off (for debugging, and so we can check your work). If your character walks off any edge, move their position back to the opposite edge. This jump should not be visible when the fog is enabled, since the terrain edges match and the distant changes will be hidden.

Set your near clipping plane to just close enough to not visibly clip the terrain. Set your far clipping plane to just past where the landscape disappears in the fog.

634 only

Have two modes, toggled with the 'o' key. One that uses OpenGL matrix operations, and one where you do all of the matrix operations yourself, using only glLoadMatrix for both the GL_PERSPECTIVE and GL_MODELVIEW matrix, where you create all of the transformation matrices and do the matrix/matrix multiplies yourself.

Implement the routines using your OWN linear algebra package, to perform these matrix manipulations. The package should be capable of operations on matrices, vectors, and points.

Supporting code is provided for some of the operations.

Extra credit

For 5 points, tip yourself to always be aligned with the surface. You will need to interpolate the normals from each vertex using barycentric coordinates to find the normal within the triangle.

For 5 points, add gravity and drag. Gravity should not change the direction of yourself (or your car), but you should slow down going uphill and speed up going

downhill. With drag, your car should eventually slow to a stop if you are not actively accelerating.

Supporting code:

For 435, reuse the code from project 1.

For 634, please download the supporting code package for math operations.

You may find the following OpenGL calls helpful to complete the assignment:

- Reset the current matrix to a 4x4 identity matrix: glLoadIdentity()
- Set up perspective: glFrustum(left, right, bottom, top, near, far) or gluPerspective(fov,aspect,near,far)
- Look-at transform from point eye to center point, with up vector projecting vertically on the screen: gluLookAt(eyeX, eyeY, eyeZ, centerX, centerY, centerZ, upX, upY, upZ)
- Translate: glTranslatef(x,y,z)
- Rotate around an axis through the origin: glRotatef(degrees, axis_x, axis_y, axis_z)
- Apply an arbitrary 4x4 transformation matrix: glMultMatrix(matrix)
- Push and pop matrix state: glPushMatrix(), glPopMatrix()
- Replace the current matrix with one of your own devising: glLoadMatrix(matrix)

Extra credits

For 5 points, try implementing an orthographic camera (this is relatively simple).

For 8 points, write geometry handling routines that the robot can walk (arms and legs are moving in the appropriate reference frames). You will want to figure out the scene graph and how different parts of the body move together. For the transformation, refer to our lecture and the OpenGL Redbook Chapter 3 and example code for details. It is okay to let the robot walk in place (do not move in the world coordinate) or walk along the "FOO-axis" (x, y, or z).

For 10 points for 435 students, implement the 634 part of the assignment.

What to turn in

Source code only by email to TA. Please do not include any .o files. Please include:

- A README with your handin containing basic information about your design decisions and any known bugs or extra credit;
- How to compile and run your code as if you are telling a colleague that is to continue the development. This means you will need to submit **all code**.

- Please name your project directory as **02viewing_<your umbc name>** and put everything in that directory.

You only need to submit ONE tarball that compresses all files in the ***02viewing_<your umbc name>*** directory. To create the tarball, on the gl server, go one level above the 02viewing_<your umbc name> directory and then type in the following line.

tar cfv 02viewing_<your umbc name>.tar 02viewing_<your umbc name>

Email 02viewing_<your umbc name>.tar to the TA Alisa.