CSC 251: Graphics - Spring 2013:

Assignment 3: Build Your Own OpenGL Pipeline

Due: Feb 26th, 7 pm.

1 The Problem

Till now, you have used the standard implementation of OpenGL to generate 3D models and to project them on to the screen. In this assignment, you will create a simplified OpenGL (sgl) engine that will take care of the Modelling, Viewing, and Projection transformations.

2 The sgl Library

Your system should have 4 different matrices to hold the current modelling, viewing, projection, and viewport transformations. To make matters easy, we will have different commands for each. Your Simplified OpenGL library should have the following functions:

- Modelling Functions: The modelling functions modify the modelling matrix mentioned before. Each of the functions right multiplies the current Modelling Matrix with a matrix corresponding to the transform. (Except, obviously, the last three.) The Modelling Matrix transforms ORC to WC.
  1. sglModRotate(angle, axisx, axisy, axisz)
  2. sglModTranslate(tx, ty, tz)
  3. sglModScale(sx, sy, sz)
  4. sglModMatrix(float mat[16])
  5. sglModLoadIdentity()
  6. sglModPushMatrix()
  7. sglModPopMatrix()
- Viewing Functions: The viewing commands affect the camera position/parameters. The first 3 commands right multiply the current Viewing Matrix by a matrix corresponding to the command. The last one loads identity matrix. This corresponds to the VRC and WC being aligned. As with OpenGL, the camera is at the origin of VRC and looks into -Z direction. The Viewing Matrix transforms from WC to VRC.
  1. sglViewRotate(angle, axisx, axisy, axisz)
  2. sglViewTranslate(tx, ty, tz)
  3. sglLookAt(cx, cy, cz, lx, ly, lz, upx, upy, upz)
  4. sglViewMatrix(float mat[16])
  5. sglViewLoadIdentity()
- Projection Functions: Projection commands define the camera: its projection model and parameters. These commands assign the Projection Matrix to that corresponding to the command. (i.e., no right multiplication!). The Ortho and Frustum commands can define asymmetric view volumes.

  The Projection Matrix transforms from VRC to NC. At the end, the normalized screen coordinates in the range [-1 .. 1] are obtained by dividing the first two components by the fourth one.
  1. sglProjOrtho(left, right, bottom, top, near, far)
  2. sglProjFrustum(left, right, bottiom, top, near, far)
- Viewport Commands: Viewport commands map the normalized view window to the actual one in use.
  1. sglViewport(llx, lly, width, height)

  The Viewport Matrix is defined by this command. (i.e, no right multiplication.)

2.1 Suported Primitives

We will use only the triangle primitive for this assignment. The following functions are to be supported by the system with the usual meanings.

1. sglBegin(SGL_TRIANGLES) // Draw a triangle for every triplet of vertices
2. sglEnd()
3. sglColour(r, g, b)
4. sglClear(r, g, b)
5. sglVertex(x, y, z)
6. sglShow() // Makes the scene appear; write to the PLY file

sglClear clears the framebuffer to the given colour. sglColour defines the current colour. We only support triangles for this assignment. You may use OpenGL calls to implement the above drawing functions. However, you should be coding the modeling, view and projection transformations in your own C/C++ code.

2.2 The Task

The real task is to write the above functions. When the system starts up, initialize all matrices to identity matrices. The above commands thereafter modify the matrices.

Geometry is really defined only by the sglVertex() commands. When the required number of vertices for a primitive is available, they are transformed from ORC to the screen coordinates. Form the primitives in canonical coordinate system. Perform perspective division after projection to get (x, y, z) coordinates in canonical frame. Multiply (x, y) by the viewport translformation to get the screen coordinates. Generate triangles in screen coordinates and add to your own data structure with x, y, z, colour, etc.

The scene is built up in the memory array corresponding to the frame buffer. When a sglShow() call is received, render the data structure built usign OpenGL as described below. You can use SDL/GLUT commands to open and setup the window etc.

For rendering, use OpenGL and a 3D orthographic coordinate system. Use glOrtho(scrL, scrR, scrB, scrT, -1, 1) where scrL, scrR, scrB, scrT are the left, right, bottom, top planes in screen coords. The z-values should provide proper visibility. Thus, you are not doing clipping, rasterization, etc., yourself. That is being done by the orthographic projection of OpenGL.

3 Program Structure and Compilation

Place all your sgl-API functions in one file, which is compiled to an sgl.o file. Put your main function, the OpenGL drawing functions and other associated functions in another file. They can use SDL/GLUT to open the window and to handle keyboard etc.

Build a ”Scene” data structure at the end of the drawing, which is available globally.

Demonstrate the program working using your own main file that does something ”interesting”. You may use the model that you created (say Assignment 2) by replacing all Modelling, Viewing, Projection and Viewport transformations to point to your sdl library.

4 Submission

You submissions should include your source code, a makefile and a compiled executable. You need to include a readme file that describes any additional information that is needed in compiling/executing you code.

The submission will be due on Feb 26th, 7 pm.