Computer Graphics Workshop 2 - Preworkshop

1 Introduction

In the previous assignments we have looked at particles, objects, movement and lighting using OpenGL 1.x and glut. In the coming workshop we will be looking at terrain rendering using a newer version of OpenGL and SDL (instead of glut).

1.1 New OpenGL

In the previous assignments we were using the old version of OpenGL which used the functions `glBegin` and `glEnd` to push vertices to the screen. Newer versions of OpenGL use a different method. Vertices are put into large buffer objects and processed by shader programs. This is explained in more detail in further sections.

1.2 SDL

Simple DirectMedia Layer(SDL) is a cross-platform development library designed to provide low level access to audio, keyboard, mouse, joystick, and graphics hardware via OpenGL and Direct3D. It is used by video playback software, emulators, and popular games including Valve's award winning catalog and many Humble Bundle games.

2 Source code

Provided is a program that does the basic rendering of a height map using newer OpenGL techniques. Please make sure that you have compiled this source code (using the included Makefile) in room 302/VM/titan on Linux. We also expect you to read through the source code thoroughly. A short explanation of the source files follows.

2.1 InputState

The InputState struct is a collection of two boolean arrays that contain information about which keys are pressed.

2.2 SDLApp

The SDLApp class handles all initialization and set up a loop. Unlike glut, SDL requires you to make your own loop (e.g. no `glutMainLoop()`) and make your own calls to functions that render, handle input, etcetera. Please carefully look at `SDLApp::run()` and try to understand how the main loop works. Also look at the `SDLApp::createWindow()` function to see how

initialization is handled in SDL. The SDLApp also has its own event handler that creates and updates a InputState object.

2.3 ShaderProgram

The ShaderProgram class is responsible for loading the vertex and fragment shaders in memory and provide the shader parameter to the `glUseProgram()` call in Workshop2.cpp

2.4 Workshop2

The Workshop2 class handles the actual work for the coming workshop. It has a rendering function that passes the relevant data to the shaders and draws it to the screen. The `Workshop2::update()` function handles all changes in the environment. During the workshop you will be adding several controls here. `Workshop2` also has various functions that convert the heightmap.raw input file to a array of position and normal vertices. Please study `Workshop2::loadTerrain()` and its called functions very carefully.

3 Shaders

Shaders are small pieces of programming code that are loaded and compiled during runtime. The syntax used in these shaders is very similar to C++ syntax. The vertex shader, `terrain.vs`, takes the position and normal vector and translates(projects) it to screen space. The fragment shader, `terrain.fs`, takes a fragment (pixel) and decides which color it should be. In a simplified way you could say the vertex shader handles positions and the vertex shader handles lighting/shading. Please make sure you understand what happens in these shader files.

4 Tasks

Your tasks for this preworkshop are:

1. Compile and run the source code on Linux in room 302 or your VM or titan.
   Note: If compiled correctly you will only see a flat green surface. We will insert the correct height map calculations during the workshop.
2. Thoroughly read through all of the source files, paying special attention to the functions mentioned in the above sections
3. Optionally, you can play around with the starting point. Try to load in a different heightmap or see what happens when you adjust the shader files.

Remember, the more familiar you are with this code, the easier the workshop will be.