Programming Assignment #1
Transformations and Heightmaps in 3D

CMPSC 458 Computer Graphics, Fall 2018

Intermediate Project is Due on Tuesday September 11, 2018 at Midnight

Final Project is Due on Monday September 17, 2018 at Midnight

1 Introduction

Your first assignment will require you to learn some basic OpenGL functions. Part 1 of the project will require you to apply transformations to the boxes within the starter code. Part 2 of the project will be to create a skybox, a cube with scenery texture on the faces that moves with the camera. Lastly, in Part 3, you will create a surface mesh in which height is determined by pixel values in an image called the heightmap. You will use OpenGL to transform the boxes, create the sky box, and render the mesh. You will use the GLFW toolkit for interaction with the mouse (already provided) and keyboard (example provided in starter code) to give the user interactive control over motion and rotation of the camera, as well as the ability to interactively scale the scene along the X, Y, or Z axis.

For your final submission, you must submit **working code**, example output (including a video), and a readme file explaining whether or not you met each requirement and detailing any extra credit you added. The readme file is important! Your grade **will** suffer if you do not include one!

The submissions zip file need to be named based on the student's name in the form of 'Project1_$< Intermediate||Final >$_Lastname_Firstname.zip' depending on the submission.

You can record a video of your project with the OBS software in the computer lab though the link on the desktop. We will go over this in OpenGL Session II. This software is free and cross-platform so you can use it on other computers as well.

General list of requirements:

- Movement of the Boxes which can be altered by the keyboard.
- Textured Skybox even when you fly around the 3D space.
- Textured Heightmap

2 Interaction

The GLFW toolkit provides means of keyboard and mouse interaction through callbacks. The callback function which registers the interaction is already given to you within the

function 'void processInput' for some keys. Moving around the scene is already provided to you as an example. You **must** use the interactivity keys defined in Table 1.

The GLFW library commands are pretty simple. If you want to know if a key is pressed, you use the following command: glfwGetKey(window, GLFW_KEY_<KEY>) == GLFW_PRESS) where <KEY> is the key you want to press. For example, if you want to see if "W" is pressed, you would use glfwGetKey(window, GLFW_KEY_W) == GLFW_PRESS). This does not distinguish between upper and lower case. If you want to know if shift is pressed, than you can use 'glfwGetKey(window, GLFW_KEY_LEFT_SHIFT) || glfwGetKey(window, GLFW_KEY_RIGHT_SHIFT)'. Swap 'CONTROL' for 'SHIFT' if you want to know if ctrl was pressed. The mouse interactions have already been provided for you.

Initially, the boxes should not be moving. Pressing G will stop all rotation and reset scale and translation.

The interactions will apply to the boxes which are given to you within the starter code. First, you must make each box continuously rotate in place. Using the U, I, O, J, K, L keys, you will change the scale, the rotation rate, and will translate the position of the boxes. If Shift is pressed, the keys will change the scale of the boxes. If CTRL is pressed, the keys will translate the boxes. If only the letter key is pressed, it will change the rotation rate. The table below describes all of the keyboard commands which you have to program.

In addition,

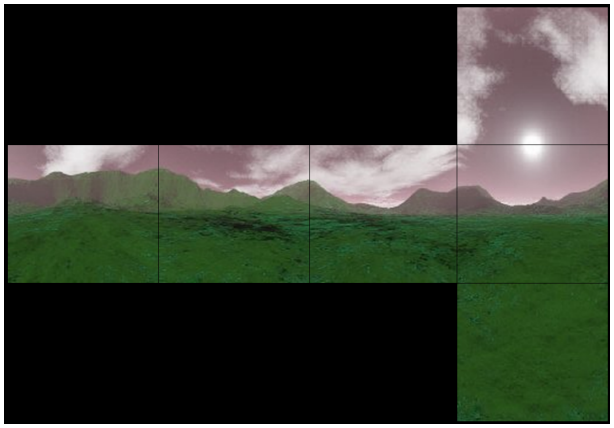
+	No Key	Shift	Ctrl
U	Increase Rotation Rate in X axis	Increase the Scale in X axis	Positive Translation in the X Axis
J	Decrease Rotation Rate in X axis	Decrease the Scale in X axis	Negative Translation in the X Axis
I	Increase Rotation Rate in Y axis	Increase the Scale in Y axis	Positive Translation in the Y Axis
K	Decrease Rotation Rate in Y axis	Decrease the Scale in Y axis	Negative Translation in the Y Axis
O	Increase Rotation Rate in Z axis	Increase the Scale in Z axis	Positive Translation in the Z Axis
L	Decrease Rotation Rate in Z axis	Decrease the Scale in Z axis	Negative Translation in the Z Axis

Table 1: The Keyboard Command that you need to program into the project. Each Command Consists of the column name + the row name. For example Ctrl+I causes a Positive Translation in the Y axis for all of the boxes.

3 Skybox

Your surface mesh will be contained in a skybox: a cube with textures of ground, horizon, and sky that fit together to appear as a seamless large environment (Figure 1). Six jpg files

Figure 1: A Set of Skybox Textures, Flattened



that form a skybox are included in the starter code. You will read each one of these and load it as an OpenGL texture. You will draw six quads to form the sides, each textured with one of the images provided. There are a few examples of loading a texture and drawing textured boxes is already in the starter code. To give your skybox a more realistic effect of being infinitely far away, you need to have it be unaffected by translation of the camera. Thus, no matter how far you move, the skybox will never get closer. I recommend against using a cubemap for the skybox since that is more complex than just making the skybox with 6 walls.

If you want to use a higher quality skybox, you may find one yourself or use one from http://www.custommapmakers.org/skyboxes.php

4 Heightmap

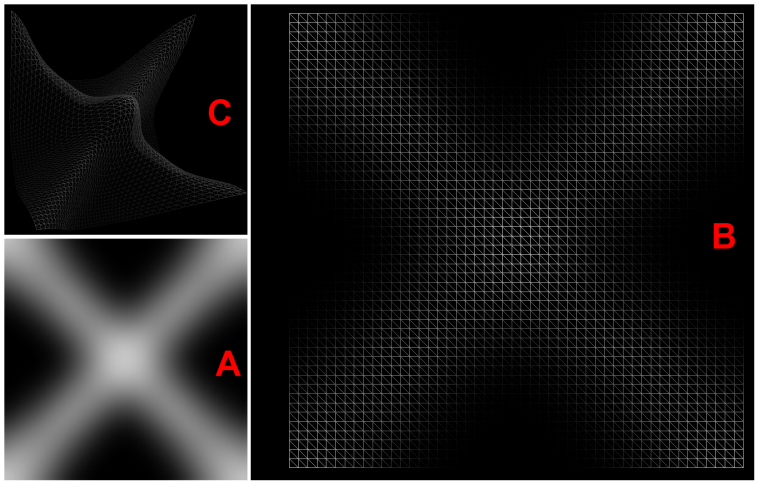
You will use the stb_image library to read a jpg file, such as in Figure 2A. The intensity (grayscale) values of pixels in this image will serve as height values for nodes in a triangle mesh (Figure 2B). This will create a detailed terrain. The heightmap will be applied to the bottom of the skybox. When rendered with OpenGL, this will look like a surface of a landscape (Figure 2C). The stb_image library will allow you to read RGB pixel values from a jpg file. There is an example of this provided in the starter code. There are also two images provided, hfhab4.jpg and spiral.jpg, that you may use as your heightmaps. The rendered heighmaps must be textured.

heightmap.hpp: This file defines the heightmap class which takes an image filename and creates and draws the heightmap. You will need to add to the following functions:

- void Draw(Shader shader, unsigned int textureID)

  Draw the data in the display loop.

Figure 2: An Example Heightmap and Surface Mesh.



- Vertex make_vertex(int x, int y)

  Create a single vertex from the image data.

- void create_heightmap()

  Created the heightmap vertexes

- void create_indices()

  Create the indices array for the EBO (corresponding to the vertexes created in create_heightmap)

- void setup_heightmap()

  Setup the vetexes/indices buffers (VAO, VBO, and EBO) and send to openGL.

5 Shaders

The 'shader.hpp' class will compile and link the shaders together for you. A vertex and fragment shader are already provide for you but you are free to write your own. The shader program code is located in the 'Project_1/Shaders' folder. The .vert file is the vertex shader and the .frag file is the fragment shader. I recommend against doing a cubemap shader for the skybox since that is more complex than just making the skybox with 6 walls.

6 camera.hpp

This is a class which keeps most of the camera parameters together in one place.

7 Extra Credit

You may earn up to 10 points of extra credit. Anything above the requirements will be considered for extra credit; the more impressive, the more points you will earn. Do not forget to describe your extra credit efforts in your readme file! Possible ideas include:

- Make the movement only along the height map (as if you are walking along the heightmap) and make the camera slightly swayed from side to side mimicking a walking movement. Provide a switch between free movement and the heightmap locked movement with the 'v' key.
- Create more spatial transformations beyond the basic set defined here (eg. shearing, rotation of all boxes around a point in space).
- Adding more interesting terrain such as making the height map heights correspond to texture (such as mountains should be tall and green should be lower).
- You can come up with something more to do as extra credit, you just need to justify why you think it is worth some points (for example, just swapping out a texture isn't really worth any points)

8 Getting Started with this code

Below are the instructions on how to run the starter code for Visual Studio (tested for VS2015 and VS2017). 'path/to/files' is the location you have unzip the files. If you try to transfer the program to another computer, you probably will have to go through these steps again (but it will be optimized for whatever computer you are using). This works for Linux and Apple systems (tested on Ubuntu 14.04 and Sierra) as long you has the proper drivers and a video card capable of at least OpenGL 3.3.

1. Unzip the code
2. CMAKE (Configuring the Project).
   (a) Open cmake-gui (already installed on lab machines or install it on your personal computer if you are using that)
   (b) Set source code to : 'path/to/files/CMPSC458_Projects'
   (c) Set build the binaries to : 'path/to/files/CMPSC458_Projects/Build
   (d) Press Configure twice (or until no more red entries)
   (e) Press Generate
3. for Visual Studio
   (a) Open the new solution file at 'path/to/files/CMPSC458_Projects/Build/CMPSC458_Projects.sl
   and open the solution file

(b) In VS, right click on Solution called 'CMPSC458_Projects' in VS and select project 'Project_1' for the single startup project.

(c) In VS, build the project (you can just click on Local Windows Debugger and it will build it). This takes some time the first time but is much faster every other time.

4. For others (Linux and Apple) from terminal.

   (a) cd into the 'path/to/files/CMPSC458_Projects/Build/'

   (b) Only for mac: turn off the 'ASSIMP_BUILD_GLTF_IMPORTER' cmake variable

   (c) Run 'make -j4' - takes some time, replace the 4 with the number of cores on you machine.

   (d) From the 'Build' directory, Run './Project_1/Project_1' to run the project. You can edit the source files with whatever editor you like though if you want to add more files, you will have to run cmake again (which you can from the command line with 'cmake ..' from the 'Build' folder and the files will automatically be added if they are in the same locations as the starter code).

The files are located in 'path/to/files/CMPSC458_Projects/Project_1/':

- .cpp source files: 'path/to/files/CMPSC458_Projects/Project_1/Sources'
- .hpp header files: 'path/to/files/CMPSC458_Projects/Project_1/Headers'
- .vert and .frag shader files: 'path/to/files/CMPSC458_Projects/Project_1/Shaders'
- Image files: 'path/to/files/CMPSC458_Projects/Project_1/Media'

9 Intermediate Project Submission

Your submission should be named 'Project1_Intermediate_Lastname_Firstname.zip' and has to contain (the files should be really small):

- Readme file containing the current state of the project (what you have done) and any issues which you are struggling on at the moment. It can also contain any questions you have on the project.
- Headers, Sources, Shader, and Media folder for the Project_1. Do **NOT** include the Vendor folder!
- 'Project_1.exe' from 'path/to/files/CMPSC458_Projects/Build/ Project_1/$< Release || Debug >$/Project_1.exe'
  Place this in the 'path/to/files/CMPSC458_Projects/Project_1' folder. Do NOT change the name of this folder or your executable will not work. If you are working on a different OS, you must create an exe file for your submission.

For your intermediate submission, we expect you to have the movement of the boxes and the interactivity mostly done.

10 Final Project Submission

Your final submission should be named 'Project1_Final_Lastname_Firstname.zip' and has to contain (the files should be really small):

- Headers, Sources, Shader, and Media folder for the 'Project_1' folder. Do **NOT** include the Vendor folder or other folders outside of the Project_1 folder!
- A README file containing:
  - The instructions on how to run the project
  - File names and Line Numbers stating where each part of your project is located
  - A list of everything contained in the submission
- A folder containing the .exe files which can run on the lab-W205 computers. Only put what is necessary to run the program.
- The zip file submission much be organized as:
  - CMPSC458_Projects
    - Shaders
    - Media
    - Sources
    - Headers
    - 'Project_1.exe' from 'path/to/files/CMPSC458_Projects/Build/ Project_1/$< Release||Debug >$/Project_1.exe' Do NOT change the name of this folder or your executable will not work. If you are working on a different OS, you must create an exe file for your submission.
  - Readme.md (md is a markdown file)
  - Project1_Lastname_Firstname.$< mp4||avi >$
- A video showing all of the requirements of the project and any extra credit work you have done in MP4 or AVI format (don't make this very large, no more than 50MB). Do not send a link to an online video (e.g. Youtube) and do not take it with your phone. Do not wait until right before submission to learn how to record the video.

11 Acknowledgments

Thanks to learnopengl.com for provide the 'shader.h' and 'camera.h' file and some of the starter code for the project. The 'glitter' github at https://github.com/Polytonic/Glitter/tree/master/Glitter which provide the shell for the starter code.

12 FAQ

- Why are there 5 coordinates for each box?

  The first three are the X, Y, and Z coordinates for each point. The last two are the texture coordinates corresponding to the 2d image which you are referencing. The texture coordinates have a range of [0,1] where 0 and 1 correspond to the edges of the image.

  The easiest way of doing this is to draw out what you want to do on a piece of paper. For the walls of the cube, one dimension in the 3d coordinates is always the same so you can look at it like a 2d problem.