CSCI 420 Programming Assignment 1: Height Fields Using Shaders

Assignment Due Friday Feb 28th 2020 by 11:59pm

An Overview

Height fields may be found in many applications of computer graphics. They are used to represent terrain in video games and simulations, and also often utilized to represent data in three dimensions. This assignment asks you to create a height field based on the data from an image which the user specifies at the command line, and to allow the user to manipulate the height field in three dimensions by rotating, translating, or scaling it. You also have to implement a vertex shader that performs smoothing of the geometry, and re-adjusts the geometry color. After the completion of your program, you will use it to create an animation. You will program the assignment using OpenGL's core profile.

This assignment is intended as a hands-on introduction to OpenGL and programming in three dimensions. It teaches the OpenGL's core profile and shader-based programming. The provided starter gives the functionality to initialize GLUT, read and write a JPEG image, handle mouse and keyboard input, and display one triangle to the screen. You must write code to handle camera transformations, transform the landscape (translate/rotate/scale), and render the heightfield. You must also write a shader to perform geometry smoothing and re-color the terrain accordingly. Please see the OpenGL Programming Guide for information, or OpenGL.org.

Background Information

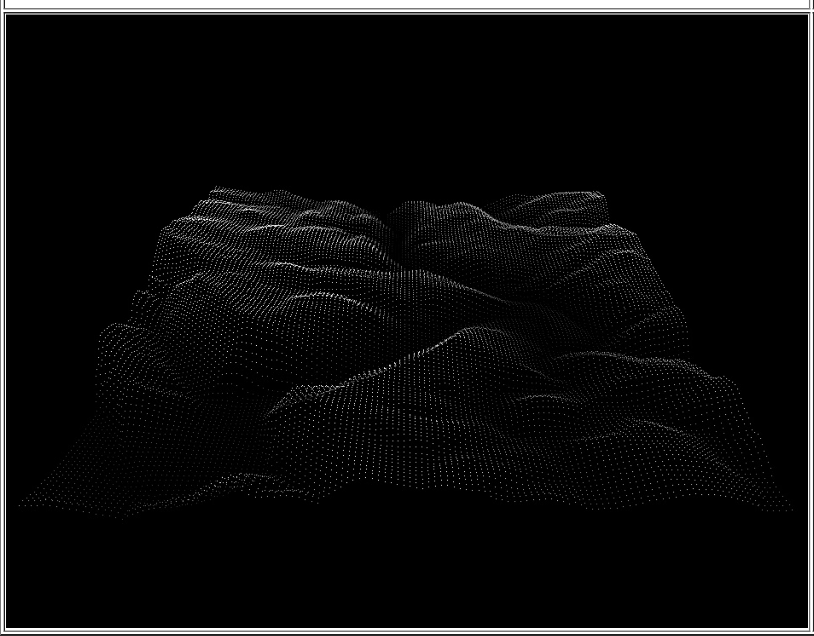
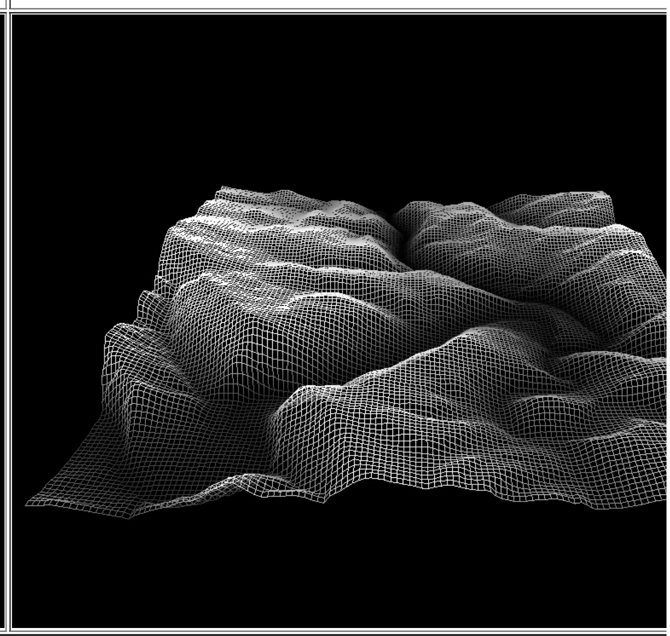
A height field is a visual representation of a function which takes as input a two-dimensional point and returns a scalar value ("height") as output.

Rendering a height field over arbitrary coordinates is somewhat tricky--we will simplify the problem by making our function piece-wise. Visually, the domain of our function is a two-dimensional grid of points, and a height value is defined at each point. We can render this data using only a point at each defined value, or use it to approximate a surface by connecting the points with triangles in 3D.

You will be using image data from a grayscale JPEG file to create your height field, such that the two dimensions of the grid correspond to the two dimensions of the image and the height value is a function of the image grayscale level. Since you will be working with grayscale image, the bytes per pixel (i.e., ImageIO::getBytesPerPixel) is always 1 and you don't have to worry about the case where the bytes per pixel is 3 (i.e., RGB images).

Render Points, Lines and Triangles

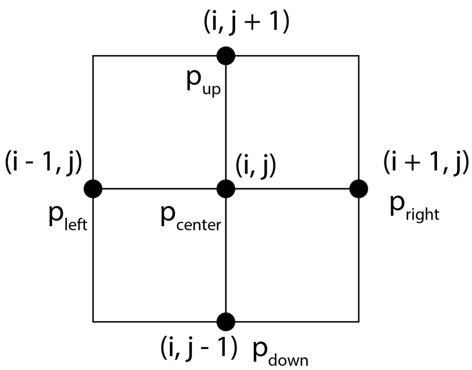
Your program needs to render the height field as points (when the key "1" is pressed on the keyboard), lines ("wireframe"; key "2"), or solid triangles (key "3"). The points, lines and solid triangles must be modeled using GL_POINTS, GL_LINES, GL_TRIANGLES, or their "LOOP" or "STRIP" variants. Usage of glPolygonMode (or similar) to achieve point or line rendering is not permitted. If in doubt, please ask the instructor/TA.

Points (key "1")	Lines (key "2")
	

Vertex Shader Requirement

You should write a vertex shader that provides two rendering modes. In the first mode, simply transform the vertex with the modelview and projection matrix. Leave the color unchanged. This vertex shader is already provided in the starter code. This mode should be used for point rendering (key "1"), line rendering (key "2"), and triangle rendering (key "3").

In the second mode (key "4"), change the vertex position to the average position of the four neighboring vertices **in the vertex shader**. Specifically, replace p_center with (p_left + p_right + p_down + p_up) / 4 (see image). This will have the effect of smoothening the terrain (the effect is most visible at low image resolutions, e.g., 128 x 128). Then transform the resulting vertex position with the modelview and projection matrix in the vertex shader as usual.



The positions of the four neighboring vertices should be passed into the vertex shader as additional attributes, in the same way as vertex position and color. In order to accommodate this, you should create additional VBOs. For example, one approach is to create 4 VBOs of 3-floats: position of the left vertex, right vertex, up vertex, down vertex. Note that these positions include the height of the vertex as one of its coordinates.

You should write **one** vertex shader that satisfies the above requirements. In order to switch between the two modes, you should create a uniform variable "uniform int mode" in the vertex shader, and set its value using the OpenGL function glUniform1i on the CPU, as follows. When the user presses keys "1", "2" or "3", mode should be set to 0, and when the user presses key "4", mode should be set to 1. When mode=0, the vertex shader should execute the first mode described above. When mode=1, the vertex shader should execute the second mode (smoothen the vertex position). You can achieve this using an "if" statement in the vertex shader. When mode=1, you should also correct the color in the vertex shader to correspond to the smoothened height, using the formula:

```
outputColor = max(inputColor, vec4(eps)) / max(inputHeight, eps) * smoothenedHeight,
```

where eps is is a small positive number such as 1e-5.

Starter Code

You can download the starter code here: Starter code.

For the Windows platform, we provide the Visual Studio 2017 and 2019 solution/project files in ./hw1-starterCode (same files for both 2017 and 2019).

On Linux, you need the libjpeg library, which can be obtained by "sudo apt-get install libjpeg62-dev libglew-dev". For Windows and Mac OS X, the starter code contains a precompiled jpeg library.

For Mac OS X, before you do any coding, you must install command-line utilities (make, gcc, etc.). Install XCode from the Mac app store, then go to XCode, and use "Preferences/Download" to install the command line tools. **Important:** If you are using Mac OS X Mojave, you need to update the OS to the latest version of Mojave. Otherwise, OpenGL does not work. Or, you can use Catalina. We tested the code both on the latest Mojave, and on Catalina 10.15.3.

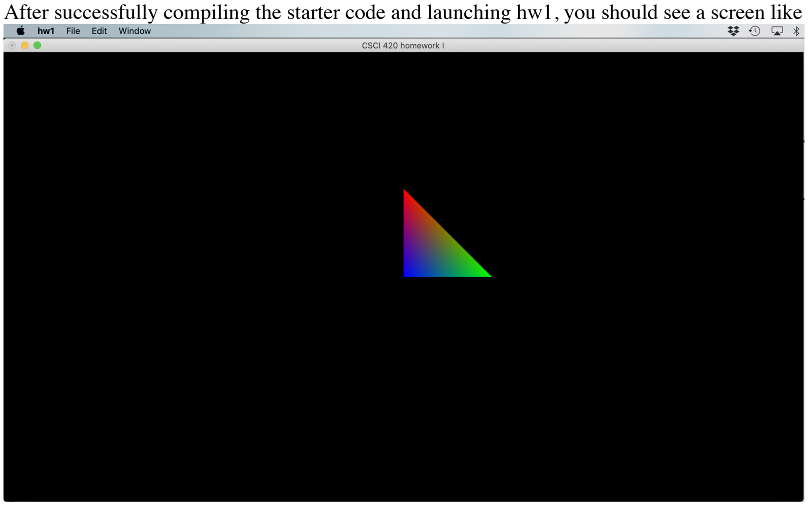
Here is a sample sequence of Ubuntu or Mac Terminal Shell commands that get everything compiled:

```
> unzip assign1_coreOpenGL_starterCode_MSVC2017_linux_mac.zip
> cd hw1-starterCode
> make
> ./hw1 heightmap/spiral.jpg
```

If your OpenGL version is too low, try updating your graphics card driver to a more recent version. Nvidia drivers have been supporting OpenGL 3.2 since 2009. On the Mac, OpenGL core profile 3.2 is supported since Mac OS X 10.7.5.

Please email the TA if you have trouble compiling the starter code.

After successfully compiling the starter code and launching hw1, you should see a screen like this:



Grading Criteria

Your program must:

- **Use the OpenGL core profile, version 3.2 or higher, and shader-based OpenGL.** The following are not allowed: compatibility profile, commands that were deprecated and/or removed in OpenGL version 3.2 or earlier, and the fixed-function pipeline. Exact specification is available here. If in doubt, please ask the instructor/TA. Submissions that do not follow these guidelines will receive zero points.
- Handle at least a 256x256 image for your height field at interactive frame rates (window size of 1280x720). Height field transformations should run smoothly.
- Be able to render the height field as points (key "1"), lines ("wireframe"; key "2"), or solid triangles (key "3"), as described in the "Render Points, Lines and Triangles" above.
- Render a smoothened height field (key "4"), colored with a smoothened color, as described in the "Vertex Shader Requirement" above.
- Render as a perspective view, utilizing GL's depth buffer for hidden surface removal.
- Use input from the mouse to rotate the heightfield using OpenGLMatrix::Rotate.
- Use input from the mouse to move the heightfield using OpenGLMatrix::Translate.
- Use input from the mouse to change the dimensions of the heightfield using OpenGLMatrix::Scale.
- Color the vertices using a color that is linearly proportional to the height (with higher values being brighter; color cannot be all black).
- Be reasonably commented and written in an understandable manner--we will read your code.
- Be submitted along with JPEG frames for the required animation (see below).
- Be submitted along with a **readme file** documenting your program's features, describing any extra credit you have done, and anything else that you may want to bring to our attention.

Animation Requirement

After finishing your program, you are required to submit a series of JPEG images which are screenshots from your program. Functionality to output a screenshot is included in the starter code, and assumes you are using a window size of 1280x720 -- your JPEG images must be this size. Please name your JPEG frames 000.jpg, 001.jpg, and so on, where 000.jpg is the first frame of your animation, and please do not exceed 300 frames. Expect a framerate of 15 frames per second, which corresponds to 20 seconds of animation running time maximum.

There is a large amount of room for creavitiy in terms of how you choose to show your results in the animation. You can use our provided input images, or modify them with any software you wish, or use your own input images. You may also use your animation to show off any extra features you choose to implement. Your animation will receive credit based on its artistic content, whether pretty, funny, or just interesting in some manner.

We will compile a video of all student submissions and show it in class. **Optional:** If you would like to convert your frames to a video by yourself, you can use Adobe Premiere, ffmpeg, QuickTime Pro, or Windows Movie Maker.

Submission

Please zip your code, readme and JPEG images into a single file and submit it to Blackboard. Include all files that were already in the starter code. Your submission should include the shader files, Makefiles, and all *.h and *.cpp files in all subfolders. Also include the compiled executable. The safest approach is to upload the entire homework folder. You can cut some space by not uploading the compiled object files (*.obj on Windows, *.o on Linux/Mac) and other intermediate files generated by the compiler such as *.pch files and similar. After submission, please verify that your zip file has been successfully uploaded. You may submit as many times as you like. If you submit the assignment multiple times, we will grade your LAST submission only. Your submission time is the time of your LAST submission; if this time is after the deadline, late policy will apply to it.

Tips

- Make sure the starter code works on your machine.
- Do not mix up modelview and projection matrices.
- The OpenGLMatrix::Rotate and OpenGLMatrix::Perspective routines take the input angle **in degrees**.
- Finish your program completely before worrying about the animation.
- You should use a separate VAO and VBO to render points, lines and triangles.
- To further optimize your program (extra credit), you can use GL_TRIANGLE_STRIP.
- Recent Mac OS X versions do not correctly trigger GLUT_ACTIVE_CTRL and GLUT_ACTIVE_CTRL when you Ctrl + mouse click or Alt + mouse click. If that is the case on your computer, you can change the key binding into whatever you want (please describe this in the readme file).
- Don't try to do this at the last minute. This assignment is supposed to be fun and relatively easy, but time pressure has a way of ruining that notion.
- For the OpenGLMatrix::Perspective function call, the near and far values for clipping plane have to be *positive*. You will see weird problems if they are zero or negative.
- Don't over-stretch the z-buffer. It has only finite precision. A good call to setup perspective is:

  ```
  matrix->Perspective(fovy, aspect, 0.01, 1000.0);
  ```

  A bad call would be:

  ```
  matrix->Perspective(fovy, aspect, 0.0001, 100000.0);
  ```

  or even worse:

  ```
  matrix->Perspective(fovy, aspect, 0.0, 100000.0);
  ```

  In the last two examples, the problem is that the ratio between the distance of the far clipping plane (=last parameter to Perspective), and the distance of the near clipping plane (=third parameter to Perspective) is way too large. Since the z-buffer has only finite precision (only a finite number of bits to store the z-value), it cannot represent such a large range. OpenGL will not warn you of this. Instead, you will get all sorts of strange artifacts on the screen and your scene will look nothing like what you intended it to be.

- **On JPEG Types:** The ImageIO library supports JPEG images with one, three, or four bytes per pixel (i.e. ImageIO::getBytesPerPixel can be 1,3, or 4). In other words, a single pixel may have either one, three, or four bytes for its intensity values. All JPEG images given in this assignments have one byte per pixel, and you don't need to worry about images with three or four bitplanes. If you have assumed one byte per pixel and happen to give a three- or fourbitplane JPEG file to your program, you will get incorrect results, at best. For your information, to convert images from RGB (ImageIO::getBytesPerPixel == 3) to grayscale (ImageIO::getBytesPerPixel == 1) in Windows or Mac, you can use Photoshop (Image->Mode->Grayscale). On all platforms, you can use the free GIMP software (Image->Mode->Grayscale).

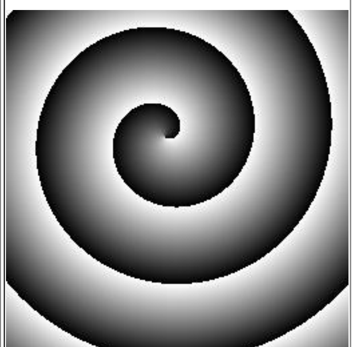
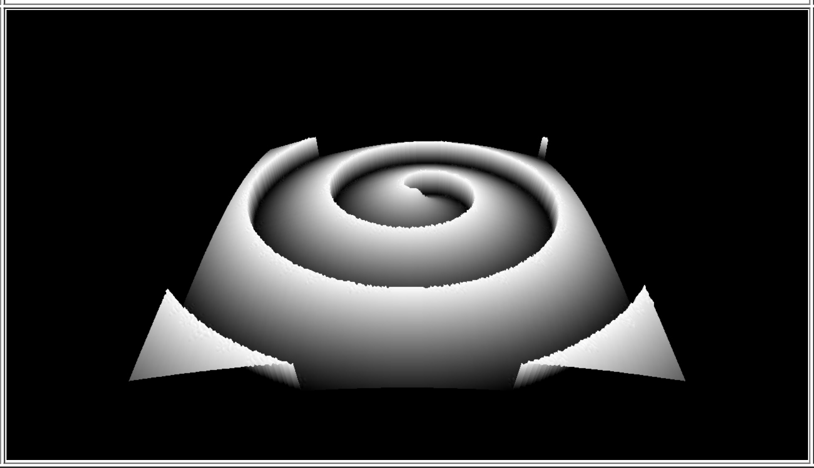
Extras

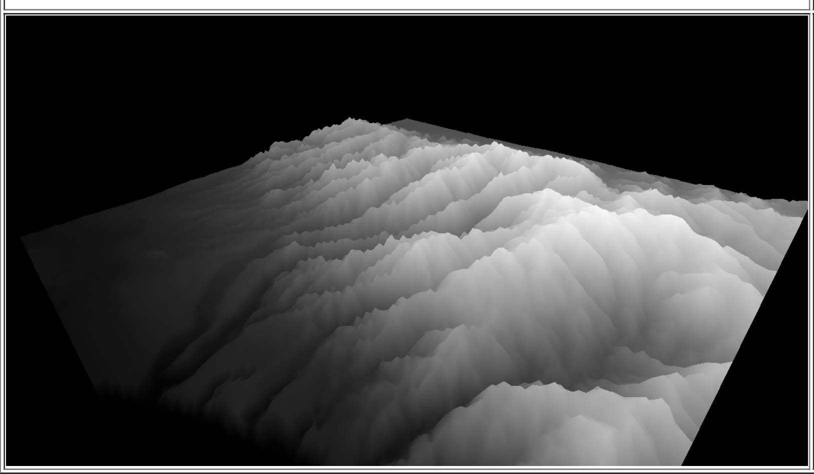
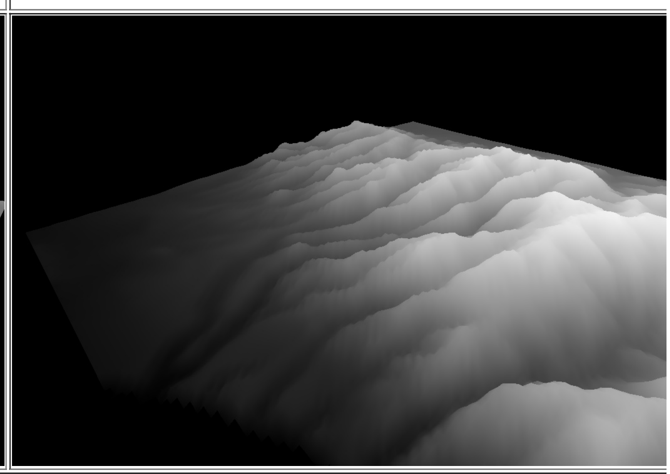
You may choose to implement any combination of the following for extra credit.

- Write a custom shader to further enhance the rendering. For example, you can experiment with material and lighting properties.
- Use element arrays and glDrawElements.
- Support color (ImageIO::getBytesPerPixel == 3) in input images.
- Render wireframe on top of solid triangles (use glPolygonOffset to avoid z-buffer fighting).
- Color the vertices based on color values taken from another image of equal size. However, your code should still also support the height-based coloring as per the core requirements.
- Texturemap the surface with an arbitrary image.
- We will also take into consideration other creative contributions.

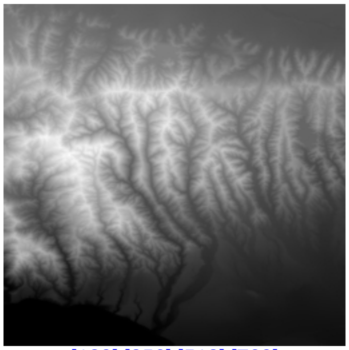
Please note that the amount of extra credit awarded will not exceed 10% of the assignment's total value.

Examples

Input (source image)	Output (height field), key "3" (triangles)	Output (height fiel
		

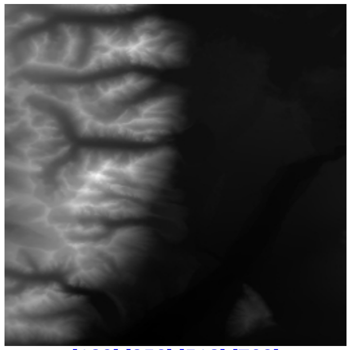
Key "3" (triangles)	Key "4" (smoothened triangles)
	

More inputs (real-world data):



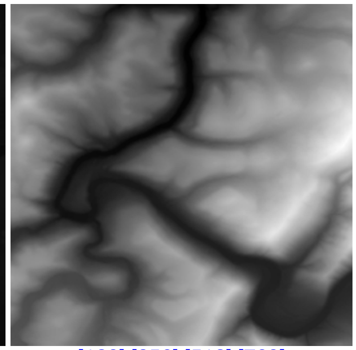
[128] [256] [512] [768]

Santa Monica Mountains
Min elevation: 0m / 0ft
Max elevation: 638m / 2093.17ft
Image size: 15.9km x 15.9km / 9.8miles x 9.8miles



[128] [256] [512] [768]

Grand Teton National Park
Min elevation: 1936m / 6351.71ft
Max elevation: 4200m / 13779.52ft
Image size: 27.8km x 27.8km / 17.2miles x 17.2miles



[128] [256] [512] [768]

Ohiopyle State Park
Min elevation: 326m / 1069.55ft
Max elevation: 712m / 2335.95ft
Image size: 7.1km x 7.1km / 4.4miles x 4.4miles

Data available from U.S. Geological Surv
Resources
Observation and Science (EROS) Center, S
SD.