

# CSCI 420 Programming Assignment 1: Height Fields

**Milestone Due Monday Feb 11th 2019 by 11:59pm**

**Complete Assignment Due Tuesday Feb 19th 2019 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. After the completion of your program, you will use it to create an animation. You will program the assignment using OpenGL's core profile.

## Why?

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 starter code we provide is minimal, giving only the functionality to initialize GLUT, read and write a JPEG image and handle mouse and keyboard input. You must write the code to handle camera transformations, transform the landscape (translate/rotate/scale), perform rendering, and handle any other functionality you may desire. Please see the OpenGL Programming Guide for information, or, [OpenGL.org](http://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. In other words, a function  $f$  takes  $x$  and  $y$  coordinates and returns a  $z$  coordinate.

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).

## Starter code

You can download the starter code here: [Windows \(Visual Studio 2017\)](#), [Linux](#) and [Mac](#).

For the Windows platform, we provide the Visual Studio 2017 solution/project files in `./hw1-starterCode`.

On Linux, you need the libjpeg library, which can be obtained by "sudo apt-get install libjpeg62-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.). On Mac OS X Lion or newer, 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. Otherwise, OpenGL does not work.

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.

## 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 manipulations should run smoothly.
- Be able to render the height field as points, lines ("wireframe"), or solid triangles (with keys for the user to switch between the three). Usage of `glPolygonMode` (or similar) to achieve point or line rendering is not permitted. The points, lines and solid triangles must be modeled using `GL_POINTS`, `GL_LINES`, `GL_TRIANGLES`, or their "LOOP" or "STRIP" variants. If in doubt, please ask the instructor/TA.
- 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 some smooth gradient. Grayscale is acceptable, but the grayscale value must vary smoothly across the heightfield.
- 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 creativity 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 and JPEG images into a single file and submit it to Blackboard. 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.

## Important: Milestone deadline

By the milestone deadline, you must be able to render a single triangle on the screen. The triangle vertices are: (0.0,-1) (red color), (1.0,-1) (green color), (0.1,-1) (blue color). The triangles must be viewed with a camera positioned at (0, 0, zStudent), where  $zStudent = 3 + \lfloor 10\text{-digit USC student ID} \rfloor / 10,000,000,000$ . Example: if your student ID is 4483471708, then  $zStudent = 3.4483471708$ . The camera must be pointing in the negative-z direction, and use the perspective view: aspect ratio=1280:720, field of view = 45 degrees. Please upload a screenshot image (1280x720) of your result to the blackboard, to "Assignment 1 Milestone". No code submission is needed. **The screenshot image should show the following: your first and last name, and your zStudent value.** You can add this text using GIMP or Photoshop.

**Milestone grading:** we will deduct 10 points (from the final hw1 score) for a missing or late milestone submission. Incomplete or incorrect submissions will receive a deduction of 0-10 points. There are no late days for the milestone.

## Tips

- Make sure the starter code works on your machine.
- Familiarize yourself with GL's viewing transformations before attempting to render the height field itself. **You should first try to render a simple object, such as a single triangle.**
- 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.
- 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 four-bitplane 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](#)).

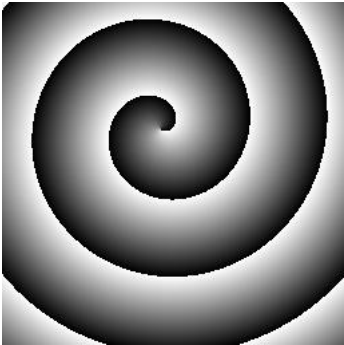
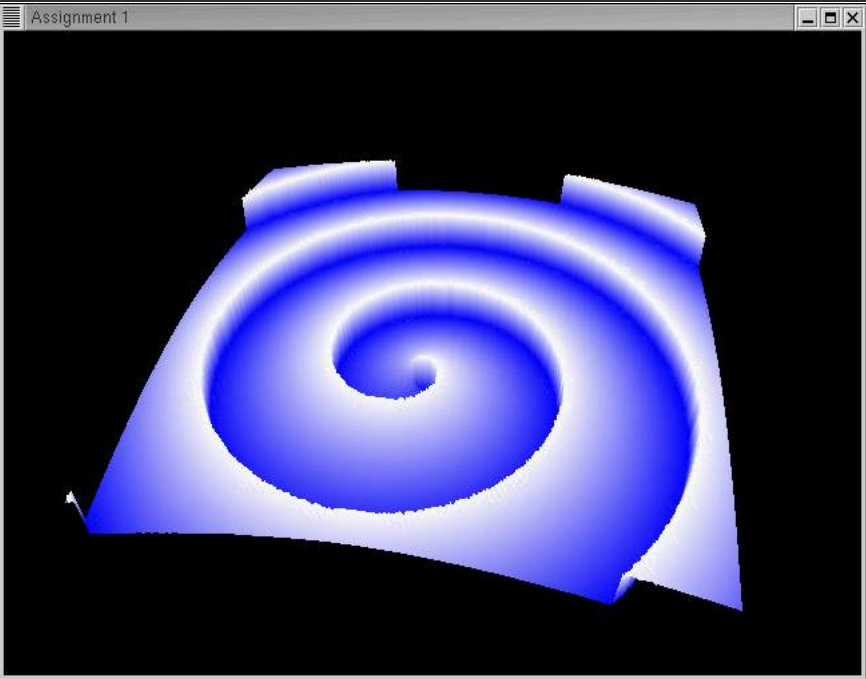
## Extras

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

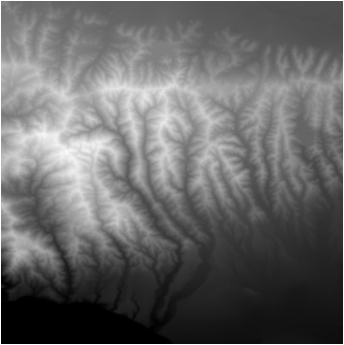
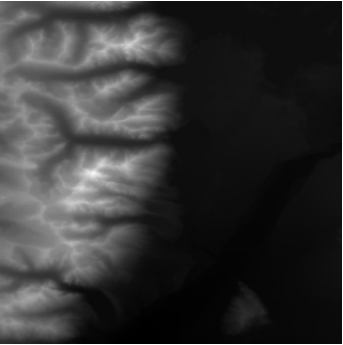
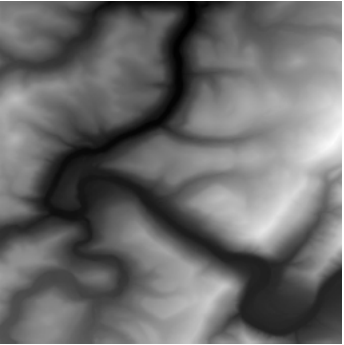

- Write a custom shader to enhance the rendering. For example, you can experiment with material and lighting properties.
- Use element arrays and `glDrawElements`.
- Support color (`ImageIO::getBytesPerPixel == 3`) in input images.
- Render wireframe on top of solid triangles (use `glPolygonOffset` to 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 smooth gradients as per the core requirements.
- Texturemap the surface with an arbitrary image.
- Allow the user to interactively deform the landscape (on the CPU, or in the shader).
- 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 and Inputs

Input (Source Image)	Output (Height Field)
	

### More inputs (real-world data):

			
<a href="#">[128]</a> <a href="#">[256]</a> <a href="#">[512]</a> <a href="#">[768]</a>	<a href="#">[128]</a> <a href="#">[256]</a> <a href="#">[512]</a> <a href="#">[768]</a>	<a href="#">[128]</a> <a href="#">[256]</a> <a href="#">[512]</a> <a href="#">[768]</a>	
<b>Santa Monica Mountains</b> Min elevation: 0m / 0ft Max elevation: 638m / 2093.17ft Image size: 15.9km x 15.9km / 9.8miles x 9.8miles	<b>Grand Teton National Park</b> Min elevation: 1936m / 6351.71ft Max elevation: 4200m / 13779.52ft Image size: 27.8km x 27.8km / 17.2miles x 17.2miles	<b>Ohiopele State Park</b> 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 Survey Resources Observation and Science (EROS) Center, Sioux Falls, SD.