CSCI 420: Homework 1 DUE DATE: 09/30/2022 Friday 23:59

Height Fields

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.

Motivation

This assignment is intended as a hands-on introduction to OpenGL and programming in three dimensions. The starter code we provide is minimal, giving only the functionality to read and write a JPEG image and handle mouse and keyboard input. You must write the code to create a window, handle camera transformations, perform rendering, and handle any other functionality you may desire. We use GLUT, please refer to class materials and example codes for basic usages. For more information, we recommend to refer to OpenGL's official website.
[https://www.opengl.org/] or [http://www.glprogramming.com/red/].

Background

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*.

Your Implementation

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. Pic->bpp) is always 1 and you don't have to worry about the case where bpp is 3 (i.e. RGB images) for now.

Starter code for *Visual Studio, Linux, and Mac* are appended. We provide a pic image library to handle image data structures (read-in, storage, and access, write-out…) . Please see the attachments. For VS2017 we have another image library, please look up the reference under ./Cimg-2.3.5/resources for image processing. If you got errors from CImg load() function, usually you'll need to install ImageMagick and set the path properly, since CImg requires ImageMagick's 'convert' program.

For Linux users, the pic image library which is used to read/write JPEG images. The *makefile* of the starter code assumes that the pic library locates one level above (i.e. if the starter code is in /home/tom/code/assign1, then the pic library should be in /home/tom/code/pic). You have to compile the pic library before the starter code. Make sure you have libjpeg before compiling pic.

The libjpeg can be obtained by "sudo apt-get install libjpeg62-dev". Here is a sample sequence of Ubuntu commands that get everything compiled:

```
> tar -xvf pic.tar.gz
> tar -xvf assign1_starterCode_linux.tar.gz
> cd pic
> make
> cd ..
> cd assign1
> make
> ./assign1 spiral.jpg
```

For Mac OS X, before any coding, you must install command-line utilities (make, gcc, etc.). Also, install XCode from the Mac AppStore, then go to XCode, and use "Preferences/Download" to install the command line tools. The makefile of the starter code assumes that the pic library locates one level above (i.e. if starter code == /Users/tom/code/assign1, then pic library should be in /Users/tom/code/pic). Please compile the pic library before compiling the starter code. Here is a sample sequence of commands that get everything compiled:

```
> unzip pic_MacOS.zip
> unzip assign1_starterCode_macOS.zip
> cd pic
> make
> cd ..
> cd assign1
> make
> ./assign1 spiral.jpg
```

Note: Under Mac OS X, if you get an error like this, you need to install Xquartz:
xpic.c:21:47: error: X11/Xlib.h: No such file or directory
xpic.c:22:23: error: X11/Xutil.h: No such file or directory
xpic.c:23:42: error: X11/Xos.h: No such file or directory

Note that the starter code includes an "exit" function call in the middle of the main function. On MS Visual Studio, you will see a console window appear briefly and then close right after that. Regardless of what operating system and platform you use to solve the assignment, you need to *remove the exit function call and replacewith proper OpenGL initialization*.

Grading Criteria

First,

Replace the header comment in assign1.cpp with
/*
CSCI 420 Computer Graphics
Assignment 1: Height Fields
<your name>
*/

(70 points) Basic Requirements. Your program **MUST**:
Handle at least a 256x256 image for your height field at interactive frame rates (window size of 640x480).
• Be able to render the height field as points, lines ("wireframe for triangles"), and solid triangles.
Bind the functions with keys, and users can switch between the three.
• Render as a perspective view, utilizing GL's depth buffer for hidden surface removal.
• Use input from the mouse to spin the heightfield around using glRotate.
• Use input from the mouse to move the heightfield around using glTranslate.
• Use input from the mouse to change the dimensions of the heightfield using glScale
• Color the vertices using some smooth gradient.
• Be reasonably commented and written in an understandable manner--*we will read your code*.
• Be submitted along with screenshots that capture your height field for **three required examples** under ./MoreData: *GrandTeton-256.jpg, OhioPyle-256.jpg, SantaMonicaMountains-256.jpg* (need to see your whole landscapes). Name your screenshots as “GrandTeton-256-HF.jpg”, “OhioPyle-256-HF.jpg”, “SantaMonicaMountains-256-HF.jpg”
• Be submitted along with an about **20-30** seconds video clip for animation (see below Animation Requirements).
• 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.

(30 points) Animation Requirements:
After finishing your program, you are required to submit a video clip about 20-30 seconds to show the functions you implement, based on the requirements and extra credits. We have provided functionality to output a screenshot, included in the starter code, and assume you are using a window size of 640x480. For animation, compile these screenshots into a video and expect a frame rate of 15 frames per second using [ffmpeg], [QuickTime Pro], or [Windows Movie Maker] or other softwares.

Screen recording softwares, such as OBS, are acceptable to use. Please make your video clean and precisely describe what you want to express. Do not include irrelevant contents in your video. Think of it as an artifact.

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 can also concatenate multiple small sequences into one to make your results look nicer. You can also add caption to assist watching. You may also use your animation to show off any extra features you chooseimplement. Your animation will receive credit based on its **artistic content**, whether pretty, funny,just interesting in some manner.

Previous example: https://www.youtube.com/watch?v=nf9dnhcD9ZE. This is just an example and doesn’t mean it’s the best video, and of course, you can present your results in a much different way.

Please properly compress your video if it's large. Usually the vide size is <100 Mb. Much larger files will result in latency for uploading to Blackboard.

Submission

Please zip your code, required results, video, and readme file into **a single file** and submit it to **Blackboard**. 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.
- Familiarize yourself with GL's viewing transformations before attempting to render the height field itself. Try rendering a simple object first. If the camera is set up correctly, you should see some contents rather than dark background.
- Do not mix up GL_MODELVIEW and GL_PROJECTION.
- For glFustrum and gluPerspective, the near and far values for clipping plane have to be positive. You will see weird problems if they are zero or negative.
- If you have issues of keyboard mapping (usually on MAC) for CTRL or ALT keys for g_ControlState switch, please use glutKeyboardFunc and use other keys (such as 'z', 'x', 'c' or others) to rebind the control.
- Finish your program completely before worrying about the animation.
- One way to optimize your program is to minimize the number of calls to glBegin(GL_TRIANGLES) and glVertex3f. For example, you should only call glBegin(GL_TRIANGLES) once at the beginning of your triangles and glEnd() at the end instead of calling glBegin(GL_TRIANGLES) for every single triangle. To further optimize your program, you can use GL_TRIANGLE_STRIP or GL_TRIANGLE_FAN.
- 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.
- Don't over-stretch the z-buffer. It has only finite precision. A good call to setup perspective is: gluPerspective(fovy, aspect, 0.01, 1000.0);
- A bad call would be gluPerspective(fovy, aspect, 0.0001, 100000.0); or gluPerspective(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 gluPerspective), and the distance of the near clipping plane (=third parameter to gluPerspective) 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: pic.h supports JPEG images with one, three, or four bitplanes (i.e. Pic->bpp 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 bytepixel 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 (bpp=3) to grayscale (bpp=1) in Windows or Mac, you can use Photoshop. In Linux, you can use GIMP.

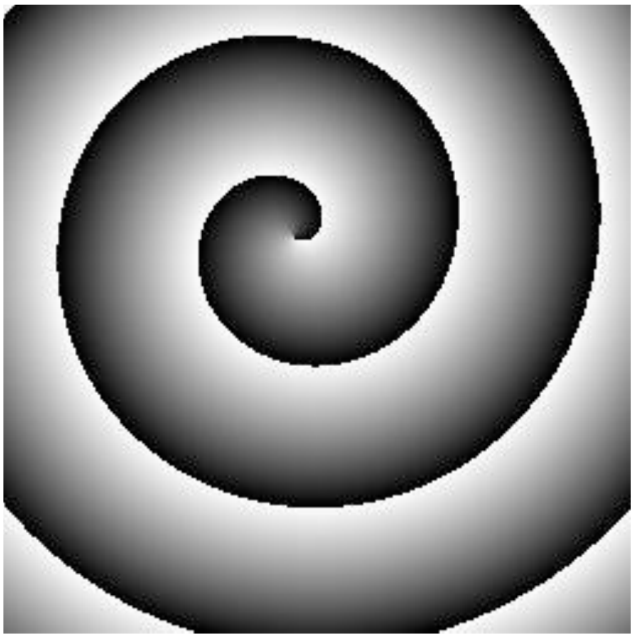
- Do not plagiarize. We have collected some codes from previous classes. No tolerance for plagiarism.

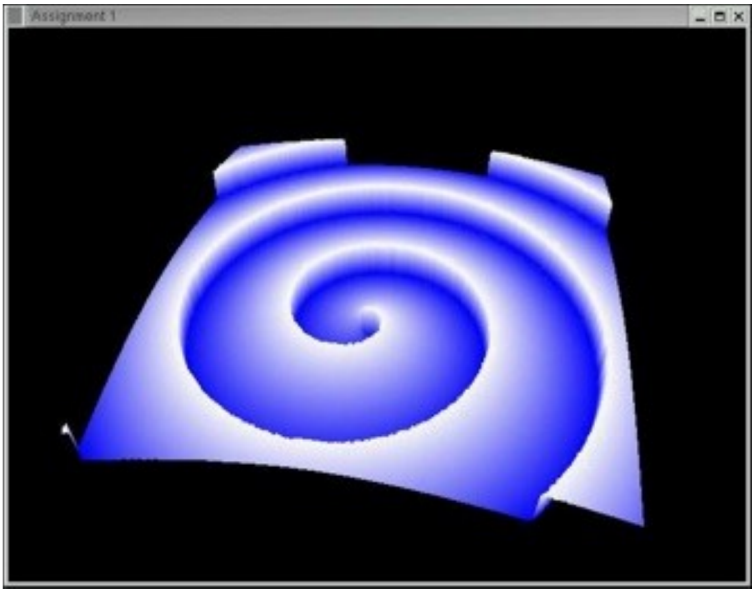
Extras

You may choose to implement any combination of the following for extra credit.
• Support color (bpp=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 support also smooth color gradients as per the core requirements.
• Texturemap the surface with an arbitrary image.
• Allow the user to interactively deform the landscape.

Each of them values 5 points. The amount of extra credit awarded will not exceed 10 points. The maximal credits you can get in this homework is 100 (basic + artistic part) + 10 (extra).

Examples (Input and Output)





Conducts of discussion

We encourage students to discuss with partners or on Piazza about *concepts, workflow, ideas, experiences, OpenGL references, or posts on StackOverflow or somewhere*. However, the discussion **should not include showing your implementation to other students**.

Similarly, do not bring your implementation to offices hours and ask why it doesn't work. **We would not debug your codes step by step**. You have the responsibility to maintain your codes and do all low-level details (coding, debugging, and run experiments) by yourself. We expect you have ideas about your implementation, such as *describe what your workflow is, how you set your cameras and objects, what the results look like from your workflow.*