GPU Programming

Computer Graphics: Project 4

1 - Objective

The purpose of this assignment is to use the GPU (vertex and fragment processors) to carry out various rendering tasks. You will write three short fragment programs that carry out various rendering task. In addition, you will write one vertex program that will change the geometry of a simple surface. (Note that unlike previous assignments, this one is **not** divided into part A and part B.)

2 - Deadline

This project should be submitted on T-Square by 11:55PM on Monday, April 10, 2017.

3 - Process

3.1 Download the base source

Download and unzip the folder with the base code for this project.

3.2 Project Description

The code that you create will take the simple examples we provide and modify them to carry out a set of tasks. Here is a list of the four tasks:

Swiss Cheese

The first task is to take the translucent blue polygon and modify it so that the polygon has a number of circular holes. You will do this by modifying the alpha value on a per-fragment basis. In particular, you should create an array of holes on the polygon that are arranged in a grid of 3 by 3. Make sure that this polygon is the last one drawn, so the transparency will be handled correctly from all viewing directions.

Mandelbrot Set

The second task is to draw the fractal known as the Mandelbrot Set. You will take one of the squares from the example code and modify it so that you display a white Mandelbrot set on some colored background. The colors (and possibly color bands) for the background are for you to decide. Let z(i+1) = z(i)^2 + c, where z and c are both complex numbers. The Mandelbrot set is essentially a map of what happens when using different values of c (which correspond to different locations in the plane). Let z(0) = (0,0), and look at the values z(1)=z(0)^2+c, z(2)=z(1)^2+c, and so on. Plugging the result of a function back into itself is called iteration. If these iterated values stay near zero (never leave a circle of radius 2), then draw a white pixel at the location c. If the values do leave the circle, color them something else (e.g. red). Do this for all the values for values of c such that cx is in the range [-2.1,0.9] and cy is in [-1.5,1.5]. The result is the Mandelbrot Set. Use 20 iterations of the function to create your Mandelbrot set.

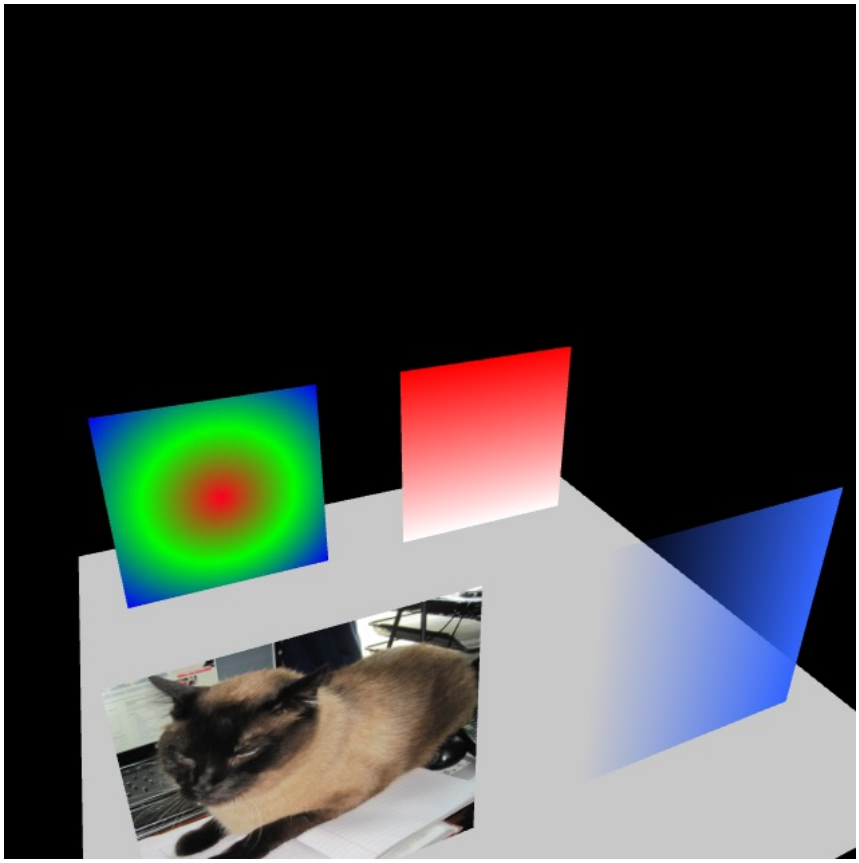
Edge Detection

The third task is to perform what is called "edge detection" in the image processing literature. You will write a GLSL fragment program to perform several texture lookups in order to find edges in images. We will provide the input images as a texture (a picture of a cat). Since we wish to do this for a grey-scale image, and because the input image is in color, you will first have to convert color pixels to grey-scale values. You can compute the intensity value for a pixel by performing a weighted average of the red, green and blue values of the pixels. Use weights 0.3, 0.6 and 0.1 for RGB, respectively. Then you will use what is known as a Laplacian filter to estimate the "edge-ness" of a pixel. The Laplacian filter simply takes the values of the four surrounding pixels, sums them, and subtracts four times the value of the middle pixel. You will have to map the resulting value to the range of grey-scale values that can be displayed. You should multiply by a scaling factor and shift the values so that the edges can be seen more easily.

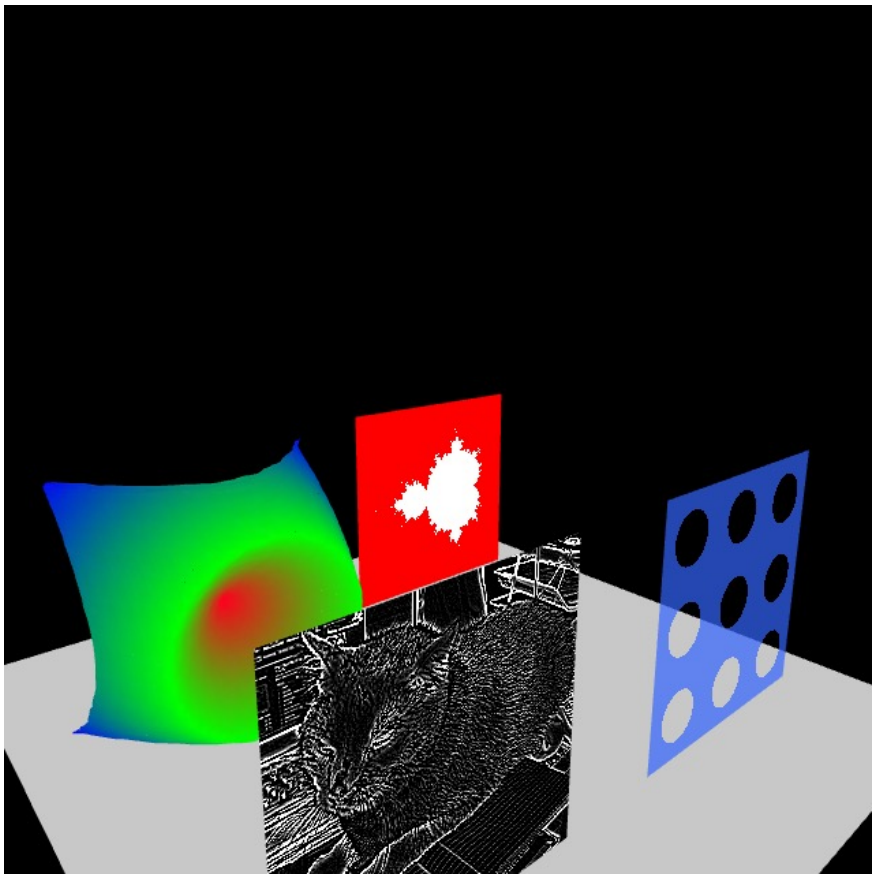
Mountain Generator

The fourth and final task is to write a vertex program to modify the geometry of a collection of polygons. Your task is to replace one of the quads with a "mountain generator". You will need to subdivide the original quad with many tiny quads by modifying the Processing code in P4.pyde. Then your vertex program will displace these vertices along the normal vector using the intensity of the closest texel to the vertex (again, convert to grey-scale as in the edge detection). Note that you will need to subdivide the quad in both x and y directions. Do *not* move the vertices out of the plane in the P4.pyde file -- this must be done in the vertex shader! You should pass a collection of equal area, planar quads into the shader. You should subdivide the quad into at least a 20x20 grid, but no more than a 40x40 grid. Also, be sure that your final geometry gets its color from the given texture.

The screen should look like the image below when you run the provided code:



Once you have finished all four of the parts of the assignment, the screen should look similar to the image below when you run your code:



3.3 Provided Code and Programming Suggestions

We provide example code for this project in an accompanying zip file. This example program draws four squares upright on a ground plane. Moving the mouse in the window rotates the view of this scene.

You should tackle this assignment by modifying the fragment and vertex code to carry out the tasks listed above. One by one, you should modify the simple polygons that we provide into much more beautiful polygons that show off the power of GLSL programming. The key to success in programming GLSL is to make small changes to working code, and verify that each of your changes does what you expect. Do not expect to write an entire GLSL program from scratch, replace the example code, and have it work beautifully the first time. Debugging GLSL code is an art. You can't single-step through GLSL code, you can't set breakpoints, and you can't print out intermediate values. Your only form of output from GLSL programs is the resulting image you see on the screen. To debug, you must make clever use of the framebuffer to display intermediate values as pixel colors. Don't forget that pixel colors must be in the range of zero to one.

There are five fragment programs, and they have names "shaderX.frag", where X is an integer from 1 to 4. These programs are in the sub-directory called "data", and must remain there for the Processing program to find them. You will need to modify some of these to create the necessary fragment shaders for the first three tasks. Processing does not know that these files are code, and so you cannot modify them from within the Processing development environment. You should edit them using any text editor that supports plain text files. We suggest that you start by modifying shader1.frag for the swiss cheese example because it already shows how to modify the

alpha channel (transparency). The file shader3.frag uses the image texture called “catTexture” (the file called “cat.png”). You should write your edge detection program based on this code.

There are four examples of vertex programs, and they are called shaderX.vert, where X is an integer from 1 to 4. These are also in the sub-directory “data”, and should stay there. Modify shader3.vert for the mountain generator task.

Please note that GLSL programs are not well supported by very old graphics cards. If you run the provided code and do not see a cat image and a translucent blue card, your graphics card is probably too old. If this turns out to be the case, seek another machine to use, such as those that are available in the library.

3.4 Programming Resources

You can find details about using shaders in Processing at the following web page:

https://processing.org/tutorials/pshader/

Here are some tutorials on GLSL:

http://nehe.gamedev.net/article/glsl_an_introduction/25007/
http://www.lighthouse3d.com/tutorials/glsl-tutorial/

3.5 Authorship Rules

The code that you turn in entirely your own. You are allowed to talk to other members of the class and to the Professor and the TA’s about general implementation of the assignment. It is also fine to seek the help of others for general Processing/Java programming questions. You may not, however, use code that anyone other than yourself has written. The only exception to this is that you should use the provided source code that is particular to this project. Code that is explicitly not allowed includes code taken from the Web, from books, from previous assignments or from any source other than yourself. You should not show your code to other students. Feel free to seek the help of the Professor and the TA's for suggestions about debugging your code.

3.6 Submission

In order to run the source code, it must be in a folder named after the main file. When submitting any assignment, leave it in this folder, compress it with zip and submit via T-square. Make sure that you include all of your modified shaders in the sub-directory "data", as well as your modified version of P4.pyde.