CPSC 478/578: Computer Graphics	Jan. 15, 2020
Spring 2020, Assignment 1	

Due: Wednesday, January 29, 2020 at 11:59 PM

Homework 1: Drawing from JavaScript and C++

For the interactive, real-time assignments in this class, we will be using JavaScript and WebGL, which should run on any modern browser. For the non-interactive assignments, we will be using C++ and outputting directly to an image file. In this assignment, we will get the basics for both of these scenarios up and running.

1 JavaScript and WebGL

Read Chapter 3 of Marschner and Shirley, and Chapter 2 of Ganovelli.

1.1 Generating Random Noise

In the subdirectory `./q1.1`, there is an example HTML file `example.html` that calls the JavaScript file `js/onmycanvas.js` and generates a colored image by blending the red, green, and blue channels in each pixel.

Using the file `randomdot.html`, modify the JavaScript file `js/randomdot.js` so that it generates an image that is a random mixture of two specific colors. Use the probability entered in via the HTML form to determine the probability of using one color over another (see Fig. 1).

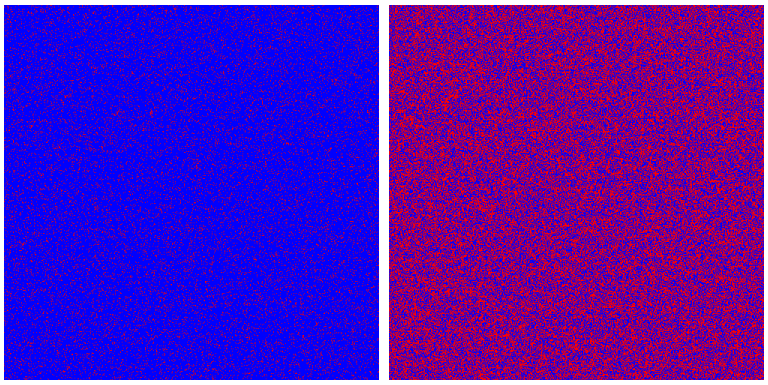

Figure 1: **Left:** An image generated with 10% chance of selecting red. **Right:** 50% chance of selecting red.

1.2 Reading and Writing an Image

You may find Shirley and Marschner, Chapter 9.1 - 9.4 to be helpful (particularly 9.4), as well as Ganovelli Chapter 10.1. We will be going over convolution and signal processing in more detail later in the term, so mastery of this material is not yet needed to complete this question.

1.2.1 Starting the Server

In the subdirectory `./q2.1`, there is a file `example.html` that you can use to load an image file. In order to do so, you first need to start a local server. From the subdirectory, call

```
python3 -m http.server
```

You should see

```
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

Now if you point your browser to `http://0.0.0.0:8000/`, it will bring up a list of the files in the subdirectory, and you can click on `example.html`. If you *do not* do this and just try to open `example.html` directly, the image loading will not work. When you click "Make the Image", nothing will happen, and an error will appear in your JavaScript console.

1.2.2 Filtering the Image

Modify the example so that the output is an image where each pixel is the average of a 5×5 neighborhood centered around each pixel in the original image, including the center pixel. If a pixel does not have a full 5×5 neighborhood (e.g. it is a corner pixel), then assume that the missing pixels are all black, and that its alpha channel value is 255. If everything is working correctly, you should see something resembling Figure 2

Figure 2: Image filtering working

1.3 Drawing Triangles

In the subdirectory `./q1.3`, there is a file `rendering-variation.html` that shows how to draw two colored triangles using the WebGL code in `./q1.3/js/rendering-variation.js`. Modify this code so that it instead draws a fan of triangles in the shape of a circle (Figure 3). Your code should support two different color schemes: a "circus tent" scheme (Fig. 3, left) and a "ramp" scheme (Fig. 3, right).

Using the code from 1.1 as an example, modify the HTML and JavaScript so that a user can enter the number of triangles they want in the circle, and toggle between the circus tent and ramp color schemes.

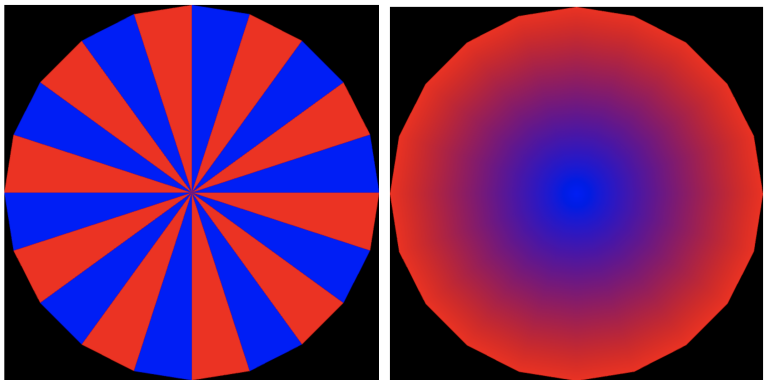

Figure 3: **Left:** Circus tent coloring **Right:** Ramp coloring

Note: The JavaScript console, `console.log` and the `Math` library are all your friends, so take advantage of them.

The SpiderGL site is also now `http://vcg.isti.cnr.it/spidergl/`, **it is not** the `http://spidergl.org` site listed in Chapter 2.4 of the Ganovelli book. The old site now looks like it will give your computer a virus.

2 Unix and C++

2.1 Generating Random Noise (Redux)

In the subdirectory `./q2.1`, there is a file `main.cpp` that you can build by calling `make`. It will generate a file `run` that you can execute by calling `./run`. Initially, it will generate an all-red 500×500 image file, `random.ppm`.

Modify `main.cpp` so that it generates random noise, just like for question 1.1. Instead of accepting inputs from an HTML file, you should parse them from the command line in the following format:

```
run <probability> <red1> <green1> <blue1> <red2> <green2> <blue2>
```

For example, the images from Fig. 1 should be generated by the following calls:

```
./run 0.1 255 0 0 0 0 255
```

```
./run 0.5 255 0 0 0 0 255
```

Write the final file out to `noise.ppm`.

2.1.1 Make a Checkerboard (578 Only)

Create a new subdirectory `./q2.1.1` and use the starter code from `./q2.1` to generate a checkerboard with two colors. You will need to accept an additional command line argument that determines how many squares are in the checkerboard. For example,

```
./run 255 0 0 0 0 255 7
```

should generate a 7×7 checkerboard pattern, with the squares colored red and blue. Note that the probability is no longer an argument. Write the final file out to `checkerboard.ppm`.

2.2 Reading and Writing an Image (Redux)

The starter code in `./q2.2` reads in a PPM file, zeros out all the color channels except for red, and then writes out the file `red.ppm`.

Use the starter code, read in a PPM and write out a blur-filtered version of that file. For example,

```
./run bunny.ppm
```

should read in `bunny.ppm` and output `filtered.ppm`. As in the WebGL version, the filtered file should be the average color value of a 5×5 neighborhood centered around each pixel, including the center pixel. Again, if a pixel does not have a full 5×5 neighborhood (e.g. it is a corner pixel), then assume that the missing pixels are all black. Note that PPM files do not have an alpha channel.

An example of a correct input / output pair is shown in Figure 4.

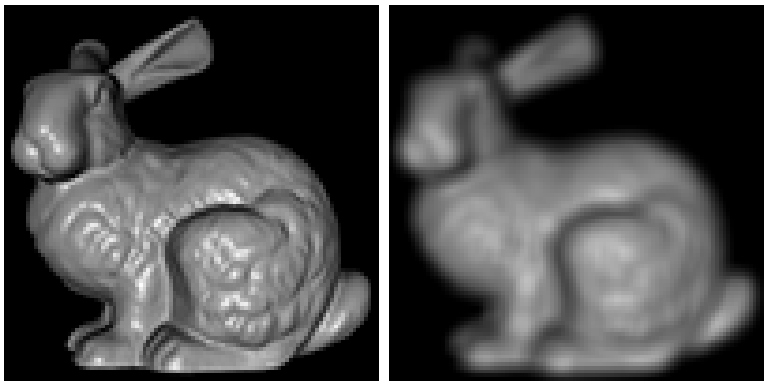

Figure 4: **Left:** The original `bunny.ppm`. **Right:** The correct `filtered.ppm` that results from averaging each 5×5 neighborhood.

2.3 Line Rasterization

Read Chapter 5.1 of Ganovelli and Chapter 8.1 in Marschner and Shirley.

Implement the *Bresenham* line rasterization algorithm, assuming that the output image is 500×500 pixels. The command line call should look like:

```
./run 25 35 400 425.
```

Here we are drawing a solid black line on a white background from pixel $(25, 35)$ to pixel $(400, 425)$. Write the final file out to `line.ppm`.

2.3.1 Triangle Wireframes (578 Only)

Using the code from the previous question, accept a third coordinate, such as:

```
./run 25 35 400 425 125 250
```

This should draw the wireframe of a triangle with vertices at pixels $(25, 35)$, $(400, 425)$, and $(125, 250)$. Make each edge of the triangle a different color, e.g. red, green, and blue. Write the final file out to `triangle.ppm`.

3 Formatting Instructions

- Turn in a ZIP file named `LastName_FirstName_Assignment_1.zip`.
- Unzipping produces subdirectories for each question: `q1.1`, `q1.2`, `q1.3`, etc.
- If necessary, each subdirectory can contain a `readme.txt` that answers any necessary questions, or explains how some pieces of your answer work while others do not.
- Follow problem-specific formatting instructions described above (input parameters, output names, etc.).

4 Points Breakdown

The assignment will be graded *on a Zoo lab machine*. Make sure your C++ code builds and runs there, and that your JavaScript runs fine on Google Chrome.

- **(15 points)** Generate random noise using WebGL.
- **(15 points)** Generate random noise using C++. For 578, the checkerboard component is worth 5 out of the 15 points.
- **(15 points)** Read and write an image using WebGL.
- **(15 points)** Read and write an image using C++.
- **(15 points)** Draw triangles using WebGL.
- **(15 points)** Rasterize lines using C++. For 578, the triangle component is worth 5 out of the 15 points.
- **(8 points)** Follow the formatting instructions correctly.
- **(2 points)** At the top level directory, include a `HOURS.txt` that lists approximately how many hours you spent on this assignment. There is no right answer; we will only be analyzing this data anonymously.