Working with Pixels on Images

The OpenGL framework we introduced in Lab 9 can also be used to process images. Unfortunately, OpenGL doesn't include any functions to read in standard image files (e.g. GIF, JPG, PNG). For this class we'll use a small, free, third-party library called lodepng. It is available here: http://lodev.org/lodepng/ . It will only read PNG images, so if you want to work on something that is not PNG it will have to be converted.

I've included the lodepng library and an additional class that makes it easier to load a .png image and retrieve the colors. It can be downloaded from the calendar page or from http://www.cse.uaa.alaska.edu/~afkjm/csce211/handouts/pngImageFiles.zip

To run the programs shown here:

1. Start with the OpenGL project setup from Lab 9.
2. Unzip the files from the pngImageFiles.zip folder and copy the files to your Visual Studio/projects/ProjectName/ProjectName folder. The files should be:
    a. BoysSanddune.png
    b. lodepng.cpp
    c. lodepng.h
    d. pngImage.cpp
    e. pngImage.h
3. Add the .cpp and .h files to your project solution

To use this library the first thing we have to do is include the pngImage.h library:

```
#include "pngImage.h"
```

Next we will create a PngImage object. In our case we'll make it a global variable. Although we have frowned upon global variables, the architecture of OpenGL makes this the easiest way to reference variables.

```
PngImage image;
```

To actually load the image call the loadImage function. This line of code goes somewhere before we need to reference the image, say, in main or in the init() function:

```
image.loadImage("BoysSanddune.png");
```

If you don't provide the full pathname to the file then Visual Studio will look for the file in the Visual Studio/projects/ProjectName/ProjectName folder.

To display the image we read in each pixel's red, green, and blue value and then plot it on the screen using glVertex2f. This all goes inside the display() function. The RGB values are each read in the range from 0-255.

Here is a complete display function to show the image that we read in:

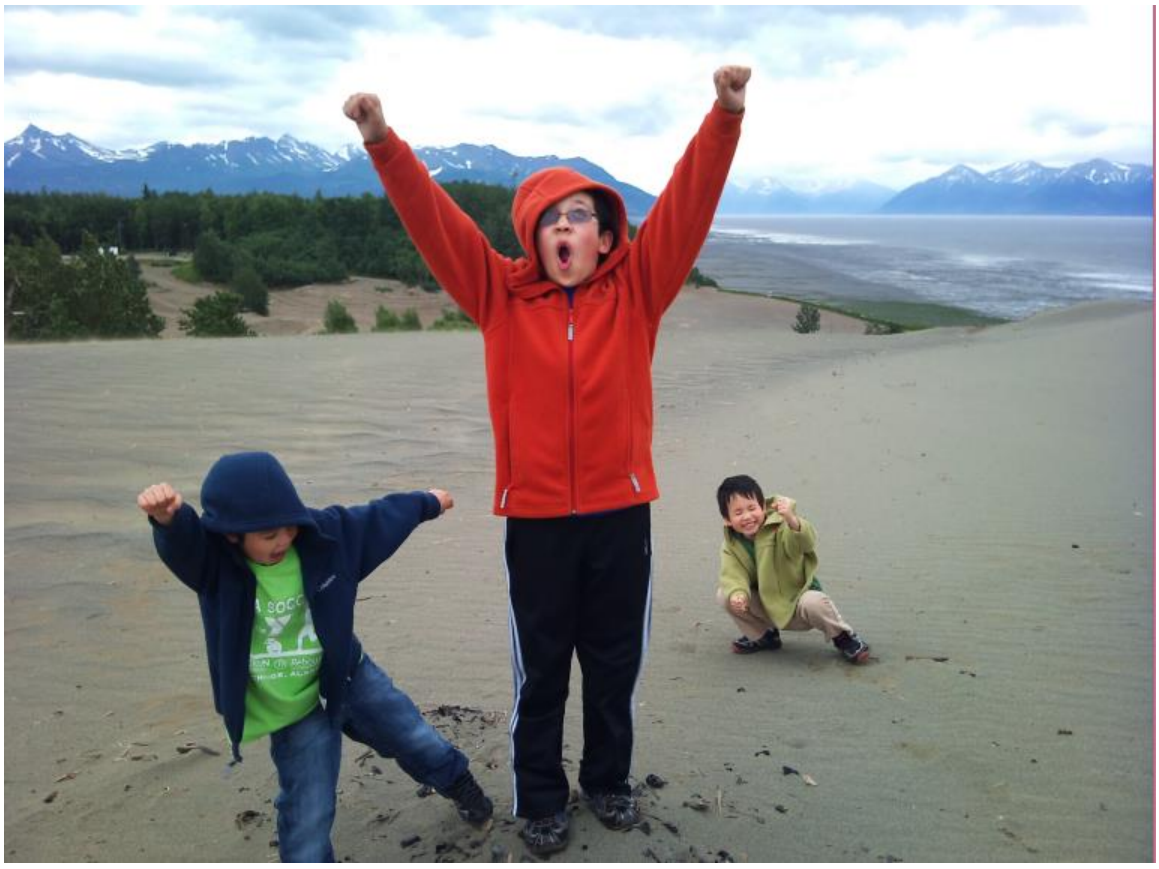
```
void display()
{
	// Clear the screen
	glClear(GL_COLOR_BUFFER_BIT);
	glBegin(GL_POINTS);

	for (int j = 0; j < image.getHeight(); j++)
		for (int i = 0; i < image.getWidth(); i++)
		{
			int r, g, b;
			image.getRGB(i, j, r, g, b);
			glColor3f((float)r / 255, (float)g / 255, (float)b / 255);
			glVertex2f(i, j);
		}
	glEnd();

	glutSwapBuffers();
	glFlush();

	return;
}
```

The program loops through the height and width of the image, reads in the RGB value, maps each RGB value to the range 0-1, then plots it using glVertex2f. Here is the image that should display using the sand dune picture of my kids:



There is a lot of material here that is mysterious for now, but don't worry about that – we'll just focus on how to manipulate the picture image.

Image Brightness

Here is a slight modification to the loop that displays the image. Can you guess what it will do?

```
glClear(GL_COLOR_BUFFER_BIT);
glBegin(GL_POINTS);
for (int j = 0; j < image.getHeight(); j++)
	for (int i = 0; i < image.getWidth(); i++)
	{
		int r, g, b;
		image.getRGB(i, j, r, g, b);
		r = static_cast<int>(r / 2.5);
		g = static_cast<int>(g / 2.5);
		b = static_cast<int>(b / 2.5);
		glColor3f((float)r / 255, (float)g / 255, (float)b / 255);

		glVertex2f(i, j);
	}
glEnd();
```

In this case we decrease the red, green, and blue components by 2.5. This darkens the entire image. If we used a large enough value each pixel would become black.

If we wanted to brighten the image, we might try changing the code so we multiply by 2.5 instead of dividing by 2.5:

```
		r = static_cast<int>(r * 2.5);
		g = static_cast<int>(g * 2.5);
		b = static_cast<int>(b * 2.5);
```

The individual color components max out at 255 so the picture does look washed out if a component is multiplied by too much.

Changing Color Values - Grayscale

We can also use our basic nested loop to easily convert an image to grayscale. A color of gray is one in which the red = green = blue. Large values are white and small values are black. An easy way to make an grayscale image out of color is to set each color value to the average of all three:

Gray = (Red + Green + Blue) / 3
Red = Gray
Green = Gray
Blue = Gray

Here is an example:

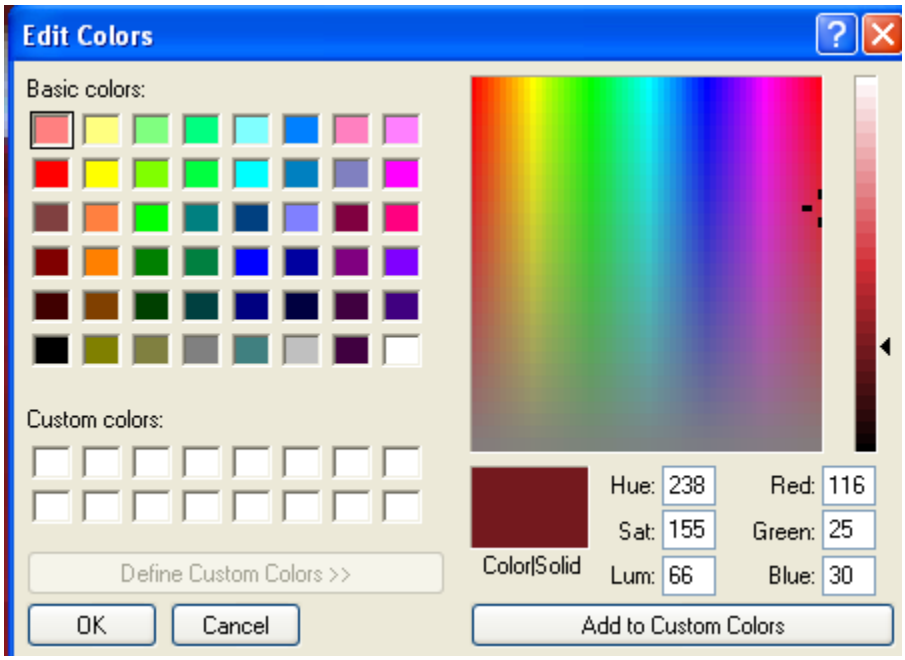
```
for (int j = 0; j < image.getHeight(); j++)
       for (int i = 0; i < image.getWidth(); i++)
       {
        int r, g, b;
        image.getRGB(i, j, r, g, b);
        int gray = (r + g + b) / 3;
        glColor3f((float)gray / 255, (float)gray / 255, (float)gray / 255);
        glVertex2f(i, j);
       }
```

The image with the kids becomes this:

Selective Color Changes

Let's say that red is no longer our favorite color and we would like to turn the color of the red on the shirt to green.

Using a paint program (Paint in Windows will do) we can examine the range of pixel values and coordinate locations for the red shirt. In Paint we can use the eye dropper tool to find coordinates, select a color, and find its RGB from the Edit Colors menu.



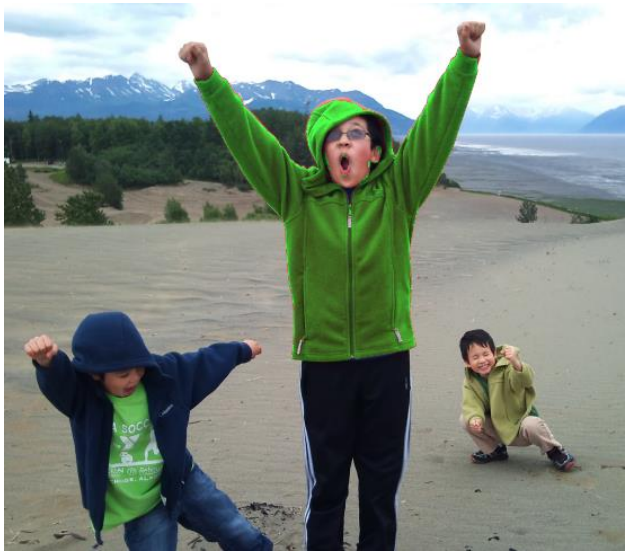
Using this we can see that the red appears to be over twice as big as the green and twice as big as the blue.

If we loop over the image and find all pixels in this range, we can make them much greener by decreasing the amount of red and increasing the amount of green:

```
for (int j = 0; j < image.getHeight(); j++)
	for (int i = 0; i < image.getWidth(); i++)
	{
		int r, g, b;
		image.getRGB(i, j, r, g, b);
		if ((r > 2.2 * g) && (r > 2.2 * b))
		{
			r = r / 3;
			g = g * 3;
		}
		glColor3f((float)r / 255, (float)g / 255, (float)b / 255);

		glVertex2f(i, j);
	}
```

The result is pretty close! We only missed a few colors on the top of the hood and inadvertently changed a few pixels around the mouth.



One way out of this problem would be to make separate loops that apply only to a small area instead of the entire image. For example we could make a loop that only changes pixels in the general area of the shirt. Use a paint program to find the coordinates of the pixels to change.

To make this a little easier we'll switch to the green shirt in the lower left and turn it red. If we limit our processing to only the area around the green shirt then we can be more aggressive in what colors we change. In our case, if the green component is 30 more than the red and blue then we change it to red.

```
for (int j = 0; j < image.getHeight(); j++)
	for (int i = 0; i < image.getWidth(); i++)
	{
		int r, g, b;
		image.getRGB(i, j, r, g, b);
		if ((i>136) && (i < 215) && (j > 366) && (j<492))
		{
			if ((g - 30 > r) && (g - 30 > b))
			{
				g = g / 3;
				r = r * 3;
			}
		}
		glColor3f((float)r / 255, (float)g / 255, (float)b / 255);

		glVertex2f(i, j);
	}
```

We still get a little bit of green right around the collar. If we used separate loops or additional if statements for those pixels then we could perfect the color change.

A very similar process is done when performing red-eye reduction on an image in a photo editing program. The user typically selects the eye region (so the program knows what area to look for) and changes any reddish pixels in that area to dark pixels.