Image Effects

Assignment #2

CPSC 453 • Winter 2016 • University of Calgary

Overview & Objectives

There are two main objectives of this second assignment in CPSC 453. The first is to learn how basic transformations work by applying them to a simple two-dimensional shape, a quadrilateral, that frames a digital image. The second is to explore a variety manipulations that can be applied to the samples (pixels) of an image to achieve different effects.

You will also gain an understanding of the OpenGL rendering pipeline and familiarize yourself with shader programming at both the vertex and the fragment stages in this assignment. The image translation, rotation, and scaling transformations are best achieved through programming the vertex shader. Image effects including colour manipulation, edge detection, edge enhancement, and blurring can all be implemented very efficiently through programming the fragment shader.

This assignment consists of a written component and a programming component. The programming part is described first, but **note that written answers to the assignment questions are due ahead of the program submission.**

Due Dates

Written component	**Thursday, October 6 at 11:59 PM**
Programming component	**Friday, October 14 at 11:59 PM**

Programming Component

There are a total of four parts to this programming assignment with an additional requirement to integrate the three scenes into a single application. This component is worth a total of 20 points, with the point distribution as indicated in each part, and an addition possibility of earning a "bonus" designation. Since the bonus is a binary state, it will only be awarded to submissions that do an exemplary job of meeting the requirements.

Part I: Displaying an Image (8 points)

Create a program that reads a digital image from file and displays the picture on the screen within a window, with the correct aspect ratio. Then provide a means to interactively change the position, orientation, and magnification of the image. Map mouse (and optionally keyboard) input to these actions in a way that makes it as intuitive as possible to manipulate the image on the screen. For example, you may want to click and drag to pan the image, or scroll to zoom in and out.

Use keyboard input to toggle between showing each of the five images provided for this assignment, as well as at least one image of your own choosing. To receive full credit for this part of the assignment, the image panning feature in your program should function such that the image always moves in the direction of your mouse drag, and only as much much as your mouse does, no matter what magnification and orientation your image is in. (Play with Google Maps a bit if this part is not clear to you from the description.) In addition, implement rotation and scaling so that the image always rotates and zooms about the centre of your window, rather than the centre of the image itself.

Notes & Hints:

- We will be using the ImageMagick library for reading image data from files. Details on how to do this will be covered in tutorial and code snippets will be provided. Of course, you are free to use any library you'd like for this purpose provided your program compiles and runs in your TA's grading environment. *Please check with them first!*
- Implementing the image transforms purely in C++ may be the easiest way to start. Trying to move some of the transform operations to the vertex shader may give you valuable experience for future assignments. However, composing your transforms correctly to achieve the desired effect may prove a bit more challenging.

Part II: Colour Effects (4 points)

Write, or modify, your fragment shader to perform a colour to greyscale conversion on your images. Do this by setting the red, green, and blue channels of the output fragment colour to

the same luminance value, L. Program your shader to employ each of the following three possibilities for RGB to luminance conversion:

1. L = 0.333 R + 0.333 G + 0.333 B
2. L = 0.299 R + 0.587 G + 0.114 B
3. L = 0.213 R + 0.715 G + 0.072 B

Provide a means to select which of the three conversion is used. Which one looks the most "correct" to you? Why do you think different formulas exist?

Implement at least one additional "creative" colour effect of your choosing. Possibilities include sepia tone, cel shading, hue shift, exposure adjustment, or similar effects. Use keyboard input to switch between the different greyscale conversions, your creative colour effect(s), and displaying the original, unaltered image. Ensure that your image panning, zooming, and rotation from Part I still works.

Part III: Edge Effects (4 points)

Interesting effects can be achieved by applying filters, or kernels, via the process of 2D convolution to an image. For example, the vertical and horizontal Sobel edge operators look like this:

+1	0	-1
+2	0	-2
+1	0	-1

Vertical Sobel

-1	-2	-1
0	0	0
+1	+2	+1

Horizontal Sobel

0	-1	0
-1	5	-1
0	-1	0

Unsharp Mask

This means that if you were to compute the intensity of an output image pixel using the vertical Sobel kernel, you would add the intensities of the adjacent image pixels from the top-right and bottom-right, twice the intensity of the pixel on the right, then subtract the pixels from the top-left, bottom-left, and twice the intensity of the left. (Recall that convolution mirrors the kernel.) When working with colour images, you can either apply the filter independently to each colour channel, or convert the image to grey and apply the filter to the luminance. Since the Sobel operator can produce negative values, it may be best to set the output to the absolute value of the result.

Among edge filters, the Sobel operators approximate the first spatial derivatives of the image in each direction, while the discrete Laplace operator (not shown) approximates the second derivative. Summing the original image intensity with its Laplacian results in a kernel, often called "unsharp mask" and shown above, that has an edge-enhancing effect on an image.

Modify or create a new fragment shader that applies the horizontal and vertical Sobel edge filters, and the unsharp mask operator, to your images. Provide a means to select which of these three filters are applied to your image. Again, ensure that your image selection and manipulation functionality from Part I still works. In particular, make sure that your filters still do the right thing even when your image is rotated or zoomed.

Part IV: Gaussian Blur (4 points)

Sometimes we want to do just the opposite of edge enhancement. The Gaussian kernel has the effect of smoothing out or "blurring" a signal or an image. This can have the result of suppressing image noise, or can simply be used as an aesthetic effect, like how the present versions of iOS or Mac OS X blurs your background image on your login screen.

A two-dimensional Gaussian is function is the product of two one-dimensional Gaussians in orthogonal directions:

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\frac{x^2}{2\sigma^2}}$$

Normalized 1D Gaussian

$$G(x,y) = \frac{1}{2\pi\sigma^2}\, e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Normalized 2D Gaussian

A discrete Gaussian kernel approximates the continuous Gaussian function. Below are approximations of normalized (sum to unity) Gaussian kernels of different sizes.

0.2	0.6	0.2

1D Gaussian

0.04	0.12	0.04
0.12	0.36	0.12
0.04	0.12	0.04

3×3 Gaussian

0.06	0.24	0.4	0.24	0.06

1D, 5-point Gaussian

An approximation of a 7-point discrete Gaussian (not shown) has values of 0.03, 0.11, 0.22, 0.28, 0.22, 0.11, and 0.03. A two-dimensional Gaussian kernel can be formed by multiplying each sample in the 1D kernel by a vertical version of the kernel. For example, multiplying 0.2 by (0.2, 0.6, 0.2) gives you (0.04, 0.12, 0.04), the first column of the 3×3 Gaussian kernel. Discrete Gaussian kernels can also be calculated directly by evaluating the continuous functions at the (x,y) pixel or sample positions with an appropriate choice for σ.

Modify or create a new fragment shader to blur your images by applying 3×3, 5×5, and 7×7 Gaussian kernels. You may implement larger kernels if you desire. Provide a means to control the amount of blur by selecting the size of the kernel. Before you submit your final program for evaluation, ensure that all elements required in Parts I through III still work as expected.

Bonus Part: Separating the Gaussian Kernel

If you experimented with using very large kernels in Part IV, you may have noticed that your program started to run very slow (if it didn't run out of memory first). Think about how many operations your poor fragment program had to do when it was trying to apply a 101×101 Gaussian kernel. As you may have expected, this performance can be improved.

The 2D Gaussian filter is a *linearly separable* filter. Recall that we were able to construct the 2D Gaussian kernel as a convolution of 1D Gaussian kernels in orthogonal directions. This means that if you were to apply a 1D Gaussian filter to your image in the horizontal direction, followed by the same 1D Gaussian filter in the vertical direction, you would achieve the same effect as if you had applied the full 2D kernel. Write your program to take advantage of this separability in applying a large and efficient Gaussian blur. Provide a means to adjust the kernel size up to several hundred pixels. Your program should still be able to run at very interactive rates (e.g. 60 frames/second) even as you're moving or rotating the image on the screen.

Submission

We encourage you to learn the course material by discussing concepts with your peers or studying other sources of information. However, all work you submit for this assignment must be your own, or explicitly provided to you for this assignment. *Submitting source code you did not author yourself is plagiarism!* If you wish to use other template or support code for this assignment, please obtain permission from the instructors first. Cite any sources of code you used to a large extent for inspiration, but did not copy, in completing this assignment.

Please upload your source file(s) to the appropriate drop box on the course Desire2Learn site, and indicate any late days used here. Include a "readme" file that briefly explains the mouse and keyboard controls for operating your program, the platform and compiler (OS and version) you built your submission on, and specific instructions for compiling your program if needed. If your program does not compile and run on the graphics lab computers in MS 239, *the onus is on you to ensure that your submission runs on your TA's grading environment for your platform!* Broken submissions will be returned for repair at a minimum penalty of one late day, and may not be accepted if the problem is severe. Ensure that you upload any supporting files (*e.g.* makefiles, project files, shaders, data files) needed to compile and run your program, but please do not upload compiled binaries or object files.

Your program must also conform to the OpenGL 3.2+ Core Profile, meaning that you should not be using any functions deprecated in the OpenGL API, to receive credit for this part of the assignment. We highly recommend using the official OpenGL 4 reference pages as your definitive guide, located at: https://www.opengl.org/sdk/docs/man/.

Assignment Questions

The written component consists of three questions, each with several parts, corresponding to the parts of the programming assignment, and worth 10 points in total. When answering the questions, show enough of your work, or provide sufficient explanation, to convince the instructors that you arrived at your answer by means of your own.

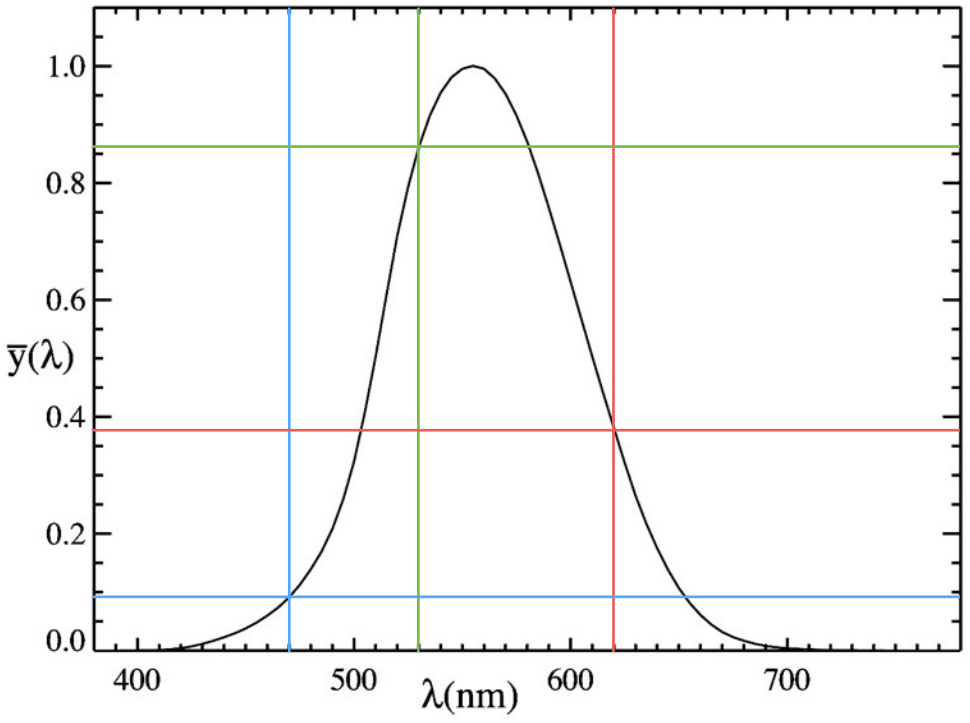
Question 1: Displaying an Image (2 points)

A. Suppose you took a photo with your digital camera that measured 3648 pixels horizontally and 2432 pixels vertically. If you were to display this image undistorted and to just fully fit within a square OpenGL window by mapping it onto a rectangle, what would the (x,y) normalized device coordinates of the four corners be?

B. If you took the rectangle from 1A and rotated it 30° counter-clockwise, what would the coordinates of its corners be?

Question 2: Colour Effects (2 points)

The graph below shows the relative sensitivity of the human visual system to light stimulus of different wavelengths. Suppose you have a display that emits red light at 620 nm, green light at 530 nm, and blue light at 470 nm (conveniently marked on the graph for you).



A. What approximate ratio of "pure red" to "pure green" light would result in the same perceived intensity by your average human observer? Likewise, what approximate ratio of "pure green" to "pure blue" light would be perceived at the same intensity?

B. If an image were encoded in RGB with these three primary colours, what might be a reasonable function to use for converting that image to greyscale (luminance)?

Question 3: Edge Effects (3 points)

A. Consider again the digital photograph described in 1A. If you displayed that photo to fit in a 1000×1000 pixel window (again as in 1A), how many image pixels are there for every display pixel the image occupies on the screen?

B. Does the above discrepancy affect the output of the Sobel edge filter in your program? If you resampled that same photo to a size of 1000×667 pixels, then applied the Sobel edge filter to it in your program, would the result look any different? Briefly explain why or why not.

C. What is the effect of applying the unsharp mask filter twice on an image? Write a 2D convolution kernel that would create the same effect as applying the 3×3 unsharp mask kernel, shown in Part III, twice to an image.

Question 4: Gaussian Blur (3 points)

A. Evaluate the Gaussian function, $G(x)$, given in Part IV at points $x = 0$, 1, 2, and 3 using $\sigma = 1.4$.

B. What value of σ was used to generate the discrete 5-point Gaussian kernel in Part IV?

C. Briefly describe a general procedure for creating a one-dimensional, discrete n-point Gaussian kernel, then use it to write the values for a 9-point Gaussian kernel.

You may submit your answers in digital form (typed or scanned) in the appropriate drop box on the course Desire2Learn site, or as hard copy in the assignment boxes on the second floor of Math Sciences. Remember that late days may not be used for the written component!