CPSC 4040/6040 Computer Graphics Images

Assigned: Aug. 27, 2014
Due: 11:59pm, Tues., Sep. 8, 2014

Programming Assignment 1 — Image Display *(Grading: 0–10 points)*

WARNING: PLEASE READ ALL INSTRUCTIONS CAREFULLY BEFORE YOU BEGIN WRITING ANY CODE.

Problem Description

You are to write a program which reads, displays, performs simple manipulations, and optionally writes an image file. You are required to write this program in C++. For reading/writing, and *only* for reading and writing, you must make use of the OpenImageIO (OIIO) API. Similarly, for displaying, and *only* for displaying, you will use OpenGL and GLUT (or any other utility library that is supported on the SoC Linux environment such as SDL, GLFW, etc.). Your program will run from the command line and utilize command line arguments.

The lectures provided sample code and a brief tutorial on OIIO, including a sample makefiles that we have created. You will need to be well versed in the `ImageSpec`, `TypeDesc`, `ImageInput`, and `ImageOutput` objects for OIIO to complete this assignment. Additionally, you should refer to the OIIO reference manual:

`https://sites.google.com/site/openimageio/documentation`

We have also provided a few samples of OpenGL code, but the "red" book will be your ultimate reference. You will need to understand to how process mouse and keyboard events in a display loop, and how to draw an image on the screen. A older, but usable version of this book is available here:

`http://www.glprogramming.com/red/`
`http://www.glprogramming.com/blue/`

Chapter 8 of "red" is the most relevant, while "blue" is an encyclopedic reference of all commands. You need not do anything more than a set of basic operations in OpenGL, but you're encouraged to be as fancy as you like.

Code Design Requirements

For this program, we specifically expect you to consider design concerns. You will be required to document your code as well as explain your design choices in an included README file.

Before you begin coding, you should explicitly choose the data structure that you will use to store the image. This need not be complicated (e.g., you could use an array of unsigned char), but you may want to consider employing object-oriented structures (e.g., a class for a pixel or an image). We will be using images the entire semester, and you may want to take the opportunity to encapsulate some expected functionality that you believe you will need. The sets of tasks required for this assignment are somewhat representative.

Similarly, we expect each of the major functional units of this program to be coded as either standalone functions or class members. We expect your design process to have considered the inputs, outputs, and expected behaviors for these functions. Each function must be clearly named for its intended purpose, and it must be clearly documented in the source code (using commented) that describe its input, output, and purpose. In your README, we will also ask you to identify the functions that you have written and state as to which of the requirements they will be satisfying.

Basic Requirements

There are six main functional requirements for the code: image display, reading images, writing images, simple image manipulation, command line processing, and user interface.

Image Display For drawing, the recommendation is to use `glDrawPixels()`, but you may also want to consider other options: texturing a square, etc.

As an assistance in displaying the image, you should be aware that the OpenGL routine `glDrawPixels()` will draw an entire array of pixel values to the display in one call. Check the course notes and the OpenGL documentation to get the details of how glDrawPixels() is used. Note, that the documentation talks about an “A” or “alpha” value in addition to red, green, and blue values for each pixel. For now you can simply set the alpha byte for each pixel to 255 (meaning fully opaque). Later we will give alpha more careful attention.

You are responsible for explicitly testing that your program displays the given image properly. Be careful to ensure that the colors of the image are properly displayed and the image is not inverted in the y-axis. Try both the sample images provided as well as images of your own. Experiment with different formats and image sizes. If your code cannot handle certain test cases, be sure to document this in the README.

Reading Images Note, some image formats include a different number of channels. Hence, even though OIIO encapsulates much of image input, you should have a separate function or module that focuses entirely on reading. You will have to handle a couple of different cases, such as if the image has three or four channels. Reading needs to collect the raw pixel data out of the file and store it in a structure which is useful for image display. This step should not happen in your `display()` function, but rather before (so that your program does not continuous read the file over and over again).

Also, you code must be able to handle the fact that the user could type a filename that does not exist, might be incorrectly formatted, or just throw errors from OIIO. If so, these errors should be handled as elegantly as possible.

Writing Images When provided with a filename to write to, your code should use that filename as a target to write out the *displayed image* whenever the user presses the ‘w’ key. Do not directly copy the file. The idea is to test that your conversion to display steps are working properly. You need to grab the pixel buffer from OpenGL and then write its contents to a file. One such function you could use in OpenGL is `glReadPixels()`; however, you could also capture the frame buffer with other calls. The image that you output should be visually identical to the displayed image after its conversion for display, up to channel depth conversion issues and/or image compression artifacts.

Be careful to ensure that your own code has both something to write out before writing (i.e. the read was successful or you exited before hitting the write) AND that the user has specified a valid filename (by catching OIIO’s errors when you open the file for writing). It will be useful to check that writes are correct by opening them in multiple other image editors as well as your own program (e.g. `preview`, `display`, `iv`, etc.)

Image Manipulation You will implement a function that “flips” the image. This function should accept two data structures, one for the input image to be flipped, and one for the output flipped image. The function must copy the pixels from the input to the output image so that they are flipped *horizontally.* This means that you will swap the leftmost column with the rightmost column, the second leftmost column with the second rightmost column, and so on. Your code should behave correctly regardless of the dimensions of the image.

Flipping will be handled through a keyboard command. Upon flipping, you should take the image currently displayed, flip it, and then update the displayed imaged image. Unlike with writing, you need not involve OpenGL for this functionality – you can use your intermediate data structure and write the function so that it updates that data.

Command Line Behavior Your program should use command line parameters to determine the input and output file name(s). We require the following convention: (1) your executable be called `oiioview`, (2) the first parameter should be the name of the input file. If a second parameter is present, it should be the name of the output file. If no second parameter is present, no file writing should be allowed.

For example, the command line

```
$> oiioview cube.ppm
```

will simply read and display the image file named `cube.ppm`, whereas the command line

```
$> oiioview cube.ppm newcube.png
```

will read and display `cube.ppm` and also enable writing to a new image file named `newcube.png`.

Interaction In addition to responding the 'w' and 'f' keys for writing and flipping, you must also respond to the 'q' key. When the user hits 'q', your program should clean up memory and exit.

You also need to augment your code to involve a mouse "click" action. Upon clicking, your code should print out (either to standard output or somewhere visible in the displayed window) the row and column of the pixel that was clicked as well as the color of that pixel.

If you have added any extra features that require interaction, please note them in the README. Hint: you may want to look at `glutMouseFunc()`, `glutMotionFunc()`, `glutSpecialFunc()`, and `glutKeyboardFunc()` in some of the examples from class.

Additional Requirements

Code Structure Your program should be organized, at the very least, into *separate procedures for reading, displaying, manipulating and writing the image*, as well as the separate functions for interacting with keyboard or mouse events. Calls to read should happen before `glutMainLoop()`, whereas calls to write should be as a response to a user event. This overall structure is a building block useful for future assignments, where you will be asked to read an image file, do some processing on it, and save the result. A particular data structure for storing the image is not required: you could use an array of structs, or an array of unsigned ints, or a class of your own design. Again, this same data structure you may want to extend in future assignments so it is worthwhile to think broadly on what you might need.

Code Documentation At a bare minimum, your code must include comments of the following types in each file. (1) Include a header comment in each file of your code that gives a description of the program, your name, the date, email, and any other relevant information. (2) For each function you create, you should include a comment listing what the function takes as input, what it does, and what data it produces. (3) Additional comments inline in the code will help clarify any complicated regions. You should use the code samples from class as guiding examples of how to comment.

In addition, your submission should include a README file that includes: (1) author name and email; (2) brief description of what the code does; (3) how to run the code (parameters, keyboard or mouse controls, etc.); and (4) any known problems, bugs, or discrepancies between the assignment requirements and what has been submitted. Feel free to include anything else you feel relevant for the grader and professor to know.

This README must also answer the following questions:

1. State precisely what data structure you have stored your image in. Do *not* tell me what variable name, I want to know the type.
2. For each of the six functional requirements, state precisely what function/module/file has implemented them.

Advanced Requirements for 6040 students You must implement at least two of the following four features, each are worth 2 of 4 points.

1. Implement some additional display controls. The first will allow the user to press the 'r', 'g', or 'b' keys and display only the red, green, or blue channel in the image as single channel, greyscale image. Pressing the 'o' key should revert the displayed image back to its original form. Note that when the user hits the 'w' key, it should write the active image to the optional second filename that has been specified.

2. Implement additional manipulation controls. By pressing the 'i' key, allow the user to invert the scanline order—flipping the image vertically (as opposed to the horizontal flip. Note that when the user hits the 'w' key, it should write the active image to the optional second filename that has been specified.

3. Provide an OpenGL reshape callback routine for your program, that responds to the user resizing the display window in two ways. (1) If the user increases the size of the display window so that it is bigger than the image, the image should remain centered in the window (note that the OpenGL `glViewport()` command might come in handy for this, but is not required if you can think of a different way). (2) If the user decreases the size of the display window so that it is smaller than the image the image should be uniformly scaled down to the largest size that will still fit the window (note that the OpenGL command `glPixelZoom()` might come in handy for this, but is not required if you can think of a different way). It might also be helpful to display the current window size to the user so that they can use this functionally in conjunction with the 'w' key to write a resized version of the image.

4. Allow your image editor to read multiple images. Using a single display window, allow the user cycle between them using the left and right arrow keys. Do this by specifying multiple filenames at the command line. The last filename should always be considered the target output filename. For example, the command line

   ```
   $> oiioview cube.ppm triangle.png output.jpg
   ```

   will read both the image file named `cube.ppm` as well as `triangle.png`. `cube.ppm` will be displayed first. When the user presses the 'w' key, whichever image is currently displayed will be written to `output.jpg`.

Submission

(Please read all of these instructions carefully.)

Please write the program in C or C++, using OpenGL and GLUT graphics routines for the display. Please take extra care to make sure that your homework compiles on the School of Computing's Ubuntu Linux cluster—testing on your own home machine, even if it runs Ubuntu, may not be sufficient. Instructions are in the syllabus as to how to get an account. Remember:

both a working build script and README must be provided

Consequently, to receive *any credit whatsoever* this code must compile on the cluster, even if it does not accomplish all of the tasks of the assignment. The grader has been instructed to give a zero to any code that does not compile. For those of you familiar with other building tools, such as make and CMake, this is also a valid option provided that the README explains how to build the code.

Submit using the handin procedure outline at `https://handin.cs.clemson.edu/`. You are welcome to use the command line interface, but the web interface is sufficient. The assignment number is `pa01`.

Finally, since we are using multiple files, please only submit a single file which has aggregated everything. This file should be named `[username]_pa01.tgz` where `[username]` is your Clemson id. For example, mine would be `levinej_pa01.tgz`. Please make sure your build script is at the top level directory.

To create such a file, the simplest thing to do is go into the directory containing all of your files and type:

```
$> tar cvzf [username]_pa01.tgz *
```

Which will create the `.tgz` file with everything in that directory.

Rubric for Grading

4040 students will be graded for the following requirements:

I. +2 Displaying

   This can include: displaying the image with correct orientation, displaying the image with correct color channels.

II. +2 Reading

   The can include: properly reading and converting the input image to an internal data structure, can the code handle filenames that do not exist, can the code handle multiple types of files, etc.

III. +2 Writing

   This can include: If the optional parameter is specified does an output file get written (even incorrectly!), can the written image be read and displayed (was the data written *correctly*), are error messages provided if the file name was not a valid image format.

IV. +1 Image Manipulation

   This includes both the 'f' key interaction and the horizontal flipping functionality.

V. +1 Interaction

   This includes both the 'w' key interaction and mouse click interaction.

VI. +2 Clarity

   Does the code have good structure, as outlined above? Is the code commented and is a README provided. Is a working build script provided?

6040 students will receive 6 points for completing the above requirements, by scaling the above by 0.6. To achieve 10 points they must also complete two of the following four tasks:

I. +2 for additional display commands.

   'r', 'g', and 'b', display individual channels of the image. 'o' to return to the original image. 'w' writes whatever version of the image is currently displayed.

II. +2 for additional image manipulation commands.

   'i', allow the viewer flip the image vertically. 'w' writes whatever version of the image is currently displayed.

III. +2 for a reshape callback.

   Allows both scaling up and scaling down the image. 'w' is modified to output the scaled image.

IV. +2 for cycling between multiple images

   Allows for reading two or more images, arrow keys implemented. Memory structure is such that each are stored internally and only one is displayed. 'w' modified to correctly write out the currently displayed image.

Going above and beyond these requirements *may* result in extra credit at the discretion of the instructor and grader. Please note any extra features you implement in the README.