PhotoShop (Iteration #2) Requirements and Group Hand-in Procedures

1. Introduction

Congratulations on completing Iteration #1 of the semester project for CSci-3081W; now on to Iteration #2! In this iteration, we are going to turn this digital painting application into a super powerful tool. In fact, we will be adding so many features to the tool, that we'll no longer think of it as just a digital painting program, it will now also be powerful enough to accomplish important photo-editing tasks. So, let's change the application's name to be more appropriate. From now on, our project is called, *Photoshop*. (We won't be quite as powerful as Adobe Photoshop, but let's borrow their name for extra inspiration anyway ☺.)

You have more time to complete this iteration of the project (4 weeks, not including spring break). So, the requirements are more significant, and you will find that you also have more of a need to organize your group effectively in order to complete the work at a high level, avoid conflicts when editing files, etc. Practice your communication and team programming skills, and don't forget to turn in a revised version of your Group Policies and Expectations with an accompanying plan/schedule for Iteration #2 during the first week of work this iteration.

2. Technical Programming Requirements

First, a note of the starting point for Iteration #2. Photoshop (Iteration #2) will build directly on the functionality that you implemented in PaintShop (Iteration #1). We are excited for you to continue to build upon your own work, i.e., the code your group already wrote in Iteration #1. However, if Iteration #1 did not go as well as you had hoped, we don't want you to fall behind in the course. Don't worry about fixing your Iteration #1, instead, you can simply switch to using the instructors' implementation of Iteration #1 as your starting point for Iteration #2. All of the code for our solution of Iteration #1 is available on Moodle. There will be an additional code base with the new User Interface for PhotoShop.

Now, on to what you will actually be designing and implementing in Iteration #2. The requirements for PhotoShop are best explained by breaking them into four categories of new features:

1. Image saving/loading
2. Image filters
3. New interactive tools
4. Undo/redo

We'll describe each of these in detail in the sections that follow, but first a note on the user interface. Together, these features dramatically update the functionality of the program. This has a major impact on the user interface, which we are implementing in this project using GLUI. To help with consistency of grading and expectations, the TAs have implemented a revised GLUI user interface (see Figure 1) that we will distribute to you. The relevant files are available via Moodle, and you should simply merge these with your own project code so that we are all working from the same starting point in terms of the user interface. Aside from this user interface code (which we distribute just to help with consistency and grading), there will be no more "support code" from the TAs. You are on your own to design and implement this project!

Figure 1: GLUI-based user interface for the PhotoShop project.

Category 1: Image Saving and Loading

PhotoShop must be able to both load and save image files. Most image manipulation programs support tens of different file types. In PhotoShop, we'll support two of the most popular: PNG and JPG.

Both of these image file formats are designed to be efficient and flexible. Therefore, the way the image data are stored on disk is complex – it is not a smart idea to write a PNG and JPG image loading routine from scratch. Instead, this is a great opportunity to link your code with an external library.

You should use version 1.6.16 of the libpng library and version 9a of the libjpeg library. Links and additional information can be found in the supplementary material wiki.

Each of these libraries comes with documentation and examples. You'll need to read these to learn how to interface with the libraries. In addition, you'll need to modify your Makefile appropriately so that the C++ compiler can find the directory where the .h header files for the PNG and JPG libraries are located and so that the C++ linker can find libpng.a and libjpg.a. For C++ libraries, when you see a .a file you should think of it as containing all of the object code (what we usually compile into .o files) for the library. The .a files are essentially just an archive of all of the .o's that were generated when the library was compiled. So, when you want to include a library in your C++ program, the .a file for that library needs to be used in the linking step just the same way that the .o files from your own code are used. Since the .a files will typically not be located in the same directory as your .o files, you not only need to tell the linker to include the .a files in the linking step, but you also need to tell the linker the path to the directory that contains them. Updating your Makefile to do this will be one of the challenges of this portion of the assignment.

Note that if you design your code appropriately, once you have saving and loading working for one image format, it should be simple to add the other (or even more formats in the future). Plan for this; this is the main thing that is important in creating a good software design or loading and saving images.

There is one more issue that will arise as you extend your program to support loading and saving images. In PaintShop, remember that we simply assumed that the canvas would always be a fixed size (800x800). This doesn't work if we want to load in images of different sizes. So, now we need our canvas to be resizable. The way to handle this in your program is as follows. When loading in a new image from a file:

1. Use the appropriate library (JPG or PNG) to read the image file.
2. Determine the size (width x height) of the image.

3. Create a new PixelBuffer of this size and copy the colors from the image file to the PixelBuffer.
4. Switch PhotoShop's current PixelBuffer to the new one you have created so that this new PixelBuffer will be the one drawn on the screen.
5. Update the size of the graphics window that is drawn on the screen using this function:

   ```
   void BaseGfxApp::setWindowDimensions(int width, int height);
   ```
   This will be found in the PhotoShop interface code

Category 2: Image Filters

The features in the second group all deal with image filtering. This is a very exciting addition to the program. With image filters, we'll now be able to blur, sharpen, or adjust saturation levels in photographs or even automatically detect all the edge features in an arbitrary image.

Image filtering is a complex topic, but it turns out that many cool filters can be created quite simply using the concept of convolving a small kernel with the pixels in an image. We will discuss this approach in class, so please refer to the slides from class for more technical details. In your program, you are responsible for implementing 4 different convolution-based filters:

Original Image

1. **Blur**
   Blur the image proportional to the amount specified in the GLUI input.

Blur(5)

Blur(10)

Blur(20)

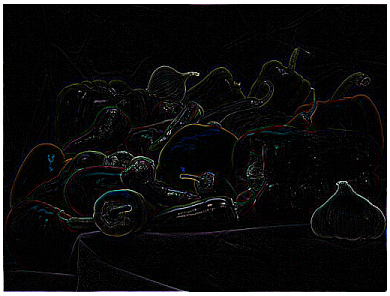
2. **Sharpen**
   Sharpen the image proportional to the amount specified in the GLUI input.

Sharpen(5)

Sharpen(10)

Sharpen(20)

3. **Edge Detection**
   Create a representation of the rate-of-change in the image. Pixels on the border of differently colored regions will be bright, while pixels in areas of low change will be dark.



Edge Detect

4. **Motion Blur**
   Blur the image in a directional manner. Use the amount specified in the GLUI input as the distance of effect a pixel has in each direction.

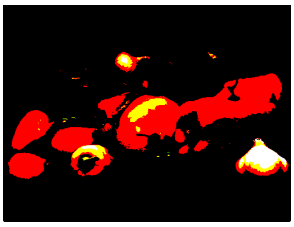
MotionBlur(5,N/S)

MotionBlur(10,NE/SW)

MotionBlur(20,E/W)

You also need to develop a series of filters that are not based on convolution but apply some other algorithm to the entire image. There are five more filters that fit in this category:

1. **Threshold**
   Each of the color channels will be rounded up or down if they are greater or less than the GLUI input value, respectively.



Thresh(0.1)

Thresh(0.5)

Thresh(0.8)

2. **Adjust saturation**
   Either increase, decrease, or invert the colorfulness of your image.


Saturate(-1)


Saturate(0)


Saturate(2)

3. **Adjust R,G,B levels**
   Increase/decrease the intensity of each color channel. This should be done by multiplying each channel by its corresponding GLUI input value.


Channels(0,1,0)


Channels(0.5,0.5,1.5)

4. **Quantize**
   Reduce the number of possible colors by binning each color value into the the number of bins specified. If using 4 bins, there will only be 4 possible intensity values for each color channel, spaced evenly: 0%, 33%, 66%, and 100%.


Quantize(2)


Quantize(4)


Quantize(8)

5. **Special Filter**
   Here is your chance to extend your design and create a filter of your own! You can implement an existing filter type (ex. dithering, corner detection, emboss) or create your own. Although you are not required to use a parameter as input, your filter should have some thought behind it. Describe your reasoning and thoughts on your filter on the fourth page of your design document.

The algorithms behind these filters vary. Some are extremely simple. For example, the Threshold filter is based on one parameter, a threshold, and every pixel channel that has an intensity value that is larger than this threshold is simply set to 100%, while every pixel channel that has an intensity value less than this threshold is set to 0%. Others, such as Sharpen, are a bit more complex. Again, all of these are most easily described in person with examples. So, we will cover the technical details of how to implement these during class using a slideshow.

Category 3: New Interactive Tools

Once you have the new capabilities to load and filter images in place, we expect that you will simply not be able to pass up the opportunity to create some new interactive tools that will add to the collection of tools we created in Iteration #1. Your specific assignment is to create two more tools:

Rubber Stamp: This tool takes an image loaded from a file and uses it as a stamp that can be placed anywhere on the canvas. The stamp is always centered around the location of the mouse cursor, and when the mouse is clicked, the stamp is applied once to the canvas. This copies the pixel data from the stamp image to the canvas. The image used for the stamp will typically be very small, much smaller than the size of the canvas, but users can load images of whatever size they want for the stamp.
Also, the tool's color should work as just as the "Channels" filter. For example, having white selected as your stamp's color will apply the stamp as it is loaded in. Having red selected as your stamp's color will apply the stamp with the red values values. Yellow will result in a stamp with the red and green values applied to the canvas.

Blur Tool: This tool uses the blur image filter functionality discussed above in the Image Filters section, but instead of applying this filter to the image as a whole, this tool acts locally, applying the blur just to the pixels within a certain radius of the brush. To make the blur feel natural, the “amount of blur” should appear to “fade out”, where the pixels near the center of the tool are the blurriest and the pixels farther away are less blurry. You can accomplish this by using a mask, such as a linear falloff, together with the blur kernel you developed for image filtering. Adjust the strength of the blur and the radius of the tool to find default settings for your blur tool that make the blur look obvious.

Category 4: Undo/Redo

Last but not least, we can't expect artists to throw out their physical pencils, paint, and paper and switch to using our digital tools unless we can do something that is physically impossible in real life – how about undo?

Your program should undo the last operation applied to the canvas when the Undo button on the user interface is clicked. (An "operation" is from mouse-down to mouse-up, when performing either a painting stroke or the application of a filter.) To make this work, you'll need to save the state of the program whenever a change is made to the canvas. (One way to do this is to save a copy of the PixelBuffer.) Then, if Undo is pressed, you need to restore the program to this previous state. Undo is very useful, so save every operation you can in your undo stack and make that stack grow for as long as you have the memory available to hold it.

Redo should "undo the last undo", reapplying the state that was most recently removed from the canvas by an undo operation. One nuance with Redo is that you can only perform a Redo if the last operation applied to the canvas was an Undo. If you Undo to revert to a previous state and then start painting on the canvas, then you can no longer Redo, this would be an invalid operation because it would overwrite the painting that just occurred. Use the new UI methods `PhotoShop::undoEnabled(bool enabled)` and `PhotoShop::redoEnabled(bool enabled)` to set whether the buttons are clickable.

3. Group Hand-in Requirements

As you plan for working on PhotoShop as a group. Here are some additional details on what your group will hand-in to be graded as part of the PhotoShop project. There are 2 hand-in submissions in total:

Sun 3/8, 11:55pm	Revision of Team Policies and Expectations, most importantly a schedule and plan for Iteration #2.
Sun 4/5, 11:55pm	Your PhotoShop Program, which includes: 1. The source code for your program, 2. Your Group Design Document (4-page PDF).

(Reminder: No late work will be accepted. We will post our solution to the assignment online the day after it is due.)

Read the sections below carefully for instructions on each of the hand-ins.

3.1 Team Policies and Expectations

This is the same document that you prepared with your team and turned in for Iteration #1. If everything worked well for you during Iteration #1, then you might not even need to revise the policies and expectations. On the other hand, if you would like to make a change to help your group work more effectively, then this is the right time to do that.

Either way, one thing you **will need to update** is your group's plan/schedule for Iteration #2. Create this in the same style as before, including intermediate milestones and treat it as an attachment to your Policies and Expectations document. Hand-in the Policies and Expectations along with the attached plan/schedule on Moodle by the end of the first week of work on Iteration #2, that's by Sunday 3/8 at 11:55pm.

3.2 Your PhotoShop Program

Your group's PhotoShop program is due on Sunday 4/5. There are two hand-ins for the PhotoShop program: 1. The actual source code, 2. A 4-page document that describes key design decisions that you made in your code. Expectations for each of these are described in detail here.

3.2.1 Source Code for Your PhotoShop Program

To turn in your source code, upload a compressed zip file of your PhotoShop source code to Moodle, containing a Makefile which builds your project on the CSE Labs UNIX computers. This compressed zip file should include all the files in your PhotoShop project that are necessary to build your code, including the glui source code and prebuilt jpg/png libraries located in a directory where the Makefile can find them. It will also contain your Group Design Document (see section 2.2.2). Upload the single .zip file to the link provided on Moodle by Sunday 3/1 at 11:55pm.

Note: You must ensure that your code will successfully build and run on the CSE Labs UNIX computers when you submit your source code. This will be vital for grading.

3.2.2 Group Design Document (4-page PDF)

Since this course is as much about good program design as it is about implementing a working program, we want you to tell us about your design. What aspects of the design are you most proud of? What is the most interesting aspect of your design? What is the most controversial, i.e. what decisions did your group debate the most before settling on a final design? We also want you to report a quick indication of how well your group functioned as a team during this iteration of the project.

To convey this to us, you should hand-in a Group Design Document that is exactly 4 pages long and adheres to the specific formatting requirements described below.

To submit your Group Design Document, place a file called iteration2-designdoc.pdf in the PhotoShop/documentation directory before you compress the PhotoShop directory. (You can also have this document committed to your git repo.)

Page 1 of the Group Design Document:

This is a cover page that should have the following 4 bits of information:

1. Names of group members.

2. Name of Github group repository.

3. Statement of completeness of the solution. Ideally, this will just be a single sentence saying that you believe that you have completely and correctly implemented PhotoShop according to the specifications provided. If your implementation does not work entirely as intended, then please list what works and what does not work to help us understand how far you got in the project and give you as much credit as possible for the work that you completed.

4. A self-assessment of contributions to the group by each member. Here's how this works: Your group gets 3 stars (*) for each member of your team. If you have a 3-person team, this means you have 9 stars in total. If you have a 4-person team, you have 12 stars in total. The stars belong collectively to the whole team. At the end of the iteration, you get to give these stars to individual team members as a reward. You have to give out all the stars, but you don't have to give an equal number to each group member. Give as many as you think that group member earned given his/her effort on this iteration of the project.

Here's an example. Peter, Paul, and Mary are a 3-member team. In general, they worked well together and followed the expectations and policies that they agreed upon at the beginning of the project. They each took on slightly different roles, but they each contributed roughly equally to the success of the team. In their report, they can simply write their self-assessment as:

Peter	***
Paul	***
Mary	***

Another example: John, Paul, George, and Ringo are a 4-member team. They started off strong and had a great first meeting during Friday's lab. Ringo offered to take the lead on preparing the PDF hand-in; John, Paul, and George thought this was a great idea because they were eager to spend most of their time learning hands-on C++ programming. They all agreed to meet as a team in person on Tuesday and Thursday nights. The team made some progress in week 1. In week 2, Ringo got busy with other classes and did not show up for the scheduled meetings. He also responded so slowly to emails that he was basically out of the loop for any important programming decisions. At the end, he showed up to try to do his part by creating the PDF hand-in, but by that point he was so behind that he didn't really have the knowledge to create this document on his own, so John and Paul did it for him. The team discusses the situation, and they divide the stars up as shown below. Ringo is not too happy about this. He was honestly overloaded with responsibilities for other classes, but the other team members point out that he agreed to a plan at the beginning and he let them down. They don't want this to happen again! And, they are obligated to report this via the star system. Ultimately, Ringo can't be too upset because this is essentially a just a warning; however, he knows that he needs to do a better job contributing to the team in the next iteration of the project.

John	****
Paul	****
George	***
Ringo	*

The star system (and the discussion that it forces you to have about how well your team has functioned) is intended to be most useful to you (your group) not us (the instructors). Following the policy in our syllabus, the instructors will use this information simply to get a broad sense of how well each group is functioning as a team. Be honest, this self-assessment is a concrete way for you to critique your group. It will not affect your grade, but if someone in your group is not contributing equally, then this is a very clear way to tell him/her and us.

Page 2 of the Group Design Document:

Using exactly one page (1-inch margins, 12-point Times New Roman font, single spaced, can include a small, simplified UML or other diagram(s) as a visual aid) describe your justification for the most important design decision that you made as a group in designing PhotoShop. Note, this cannot be the decision to use masks – we imposed that design decision as a requirement of the project – this needs to be a design decision that you made yourself. This description will be most compelling if you can describe alternative designs that you considered within your group and then discarded. For example:

Tools are implemented in our design as follows: [describe in a few paragraphs, reference a UML diagram or other figure if this makes the explanation clearer].

This design is the result of several hours of discussion and diagramming. During that process our team also considered two main alternatives designs.

Alternative #1 was [describe in a paragraph].

Alternative #2 was [describe in a paragraph].

The main advantage of our final design as compared to Alternative #1 is [a few sentences]. Another advantage is [a few sentences]. Compared to Alternative #2, we believe our final design is better because [a few more sentences].

Page 3 of the Group Design Document:

The same thing as Page 2, but for the second most important design decision. Keep in mind that the "most important" design decision might actually be the "most controversial" design decision. Think about what your group spent the most time discussing and describe the technical details of that discussion. We want to see that you put some serious effort into this design and that you learned something from it!

Page 4 of the Group Design Document:

Give a description and an example image of your special filter. Explain what your intentions were with the special filter, and what approach you chose in achieving your desired effect.

4. High-Level Grading Rubric

As you plan how to allocate tasks within your group, please keep in mind the following grading rubric that we will use. We include it here specifically to emphasize the importance of the design aspects of this project. It is possible to implement PhotoShop through a brute force approach to programming that doesn't make good use of abstract data types or plan for future change, which you should know is coming since we have more iterations of the project to complete in this course ☺. For the CSci-3081W course, we're not interested in the brute force approach, even if the implementation works perfectly. We want to see a thoughtful design that works not just now but also for the future. This is reflected in the way we will assess the work:

33% - Quality of the "most important design decisions" described in the Group Design Document.

33% - Quality of the design decisions made as evaluated by inspecting the actual source code.

33% - Quality of the implementation – meets the requirements for PhotoShop. Note that part of the quality grade will be based on whether your code compiles on the CSE Labs machines on the first try after we unzip your project source code and run make.

Total: 100%