FlashPhoto (Iteration #2) Software Requirements and Assignment Handout

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, *FlashPhoto*!!! (How flashy ☺.)

Time will go quickly. You have just 3 weeks to implement this assignment, and the requirements are more significant than in iteration #1. You will find that you 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 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. FlashPhoto (Iteration #2) will build directly on the functionality that you implemented in Brushwork (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.

Now, on to what you will actually be designing and implementing in Iteration #2. The requirements for FlashPhoto are best explained by breaking them into 4 groups of new features:

1. Image saving/loading,
2. Image filters,
3. New interactive tools, and
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 of 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 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 FlashPhoto project.

Group 1: Image Saving and Loading

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

Both of these image file formats are designed to be efficient and flexible. So, 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 on Moodle.

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 library 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 to 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 Brushwork 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 FlashPhotoApp'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 provided in the FlashPhoto interface code that we distribute:

```
void BaseGfxApp::setWindowDimensions(int width, int height);
```

Group 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 the 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.

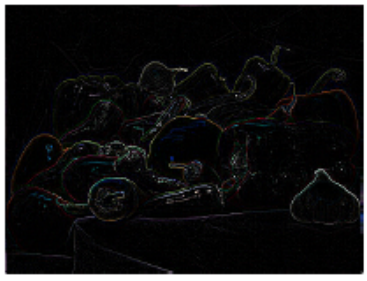
Sharpen(5)

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)

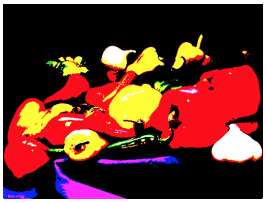
MotionBlur(20,E/W)

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:

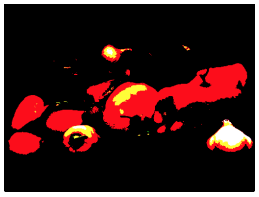
1. Threshold

Each of the color channels will be rounded up to the maximum value of 1.0 or down to the minimum value of 0.0 based on whether the pixel's value is 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.

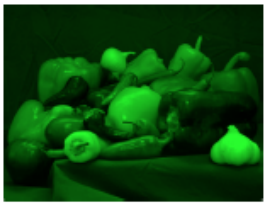
Saturate(-1)

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.

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 that has a brightness value that is larger than this threshold is simply set to white, while every pixel that has a brightness value less than this threshold is set to black. Others, such as Quantize, 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.

Group 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. If you wish, you can make the current tool color modify the stamped image; or you may decide to just ignore the current tool color.

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.

Group 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. 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.

3. Group Handin Requirements

As you plan for working on FlashPhoto as a group. Here are some additional details on what your group will handin to be graded as part of the FlashPhoto project. Note that there is a preliminary due date for just the Team Policies and Expectations document. Then, the FlashPhoto program and associated design documents are due at a later date.

Mon 3/28, 11:55pm	Team Policies and Expectations, handed in on Moodle via the Lab 3 Submission link.
Mon 4/11, 11:55pm	Your FlashPhoto Program, which includes: 1. The source code for your program, 2. Your Group Design Document (3-page PDF),

	3. Your Programmer's Blog Entries (1 per team member). Handed in via github.

(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 handins.

2.1 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 Monday 3/28 at 11:55pm.

2.2 Your FlashPhoto Program and Design Documents

Your group's FlashPhoto program is due on Monday 4/11. There are three things to handin (described in sections 2.2.1 – 2.2.3 below), and all of these can be handed in by simply tagging the final version of your work and pushing it to github.umn.edu using the two simple commands listed in section 2.2.4.

2.2.1 Source Code for Your FlashPhoto Program

When you submit your program for grading, make sure that you only submit your source code (not the .o or .exe files that get built from the source code). The first thing that we will do when grading your submission is to run "make" and see if your program builds without an error. *You must ensure that your code will successfully build on the CSE Labs UNIX computers when we run 'make' in the root directory of your project.* That means that you must have working a Makefile in that directory. We will provide some suggestions in class on how you might want to organize your program, including the image loading libraries that you will use. But, other than making sure your code builds when we type make, there are no specific requirements about how you should organize your source code for the project. That is up to you.

Do not hand your source code in via Moodle, instead use the instructions in section 2.2.4 to tag and push the final versions of your support code, group design document, and programmer's blog entries to github.umn.edu.

2.2.2 Group Design Document (3-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?

To convey this to us, you should handin a Group Design Document that is exactly 3 pages long and adheres to the specific formatting requirements described below.

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 FlashPhoto 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.

Page 2 of the Group Design Document:

Using exactly one page (1-inch margins, 11-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 FlashPhoto. 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!

Save the 3-page Group Design Document in a file named GroupDesign.pdf in the root folder of your repository.

2.2.3 Programmers' Blog Entries

Finally, each member of your team should write his/her own blog entry that describes a section of code that you have personally written and that appears in the final version of the project that is submitted for grading. For each code snippet (this can be a single routine, perhaps 20-25 lines of code), copy and paste the actual code into the blog entry, describe its purpose, how the code accomplishes this purpose, and the coding style, inducing use of documentation within the code. The idea here is that new programmers who join the project should be able to read the collection of blog entries in order to learn not just about important portions of the software but also about appropriate styles for coding and documentation when contributing to the project.

The blog entries should be about a page in length in terms of textual description, not counting lines of code that are pasted into the entry or any imagery that you may wish to include. Other details of the formatting are left for the individual programmers to decide.

Create these blogs in individual .html files (each of the programmers on the team should create his/her own .html file) and place all of the .html files in a web/ subdirectory in your group repository.

2.2.4 Submitting your FlashPhoto Program and Design Documents for Grading

The TAs will download your FlashPhoto code and design documents directly from github.umn.edu. All you have to do is use git to tag the final version that your group wishes to submit and make sure that it is pushed to github.umn.edu. To do this, run the following two commands:

```
git tag -a iter2 -m "final commit for iteration 2"
git push origin --tags
```

3. 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 FlashPhoto 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 one more iteration 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:

30% - Quality of the "most important design decisions" described in the Group Design Document.

- The document explains two design decisions.
- The decisions were significant in some way; for example, they might concern the most important class(es) in your design, or they might have been difficult or controversial.
- The decision is well justified in terms of good design principles, comparisons with alternatives, analysis of advantages and disadvantages.
- The portion of the report is well written and follows the formatting criteria for this section.

30% - Quality of the design decisions made as evaluated by inspecting the source code.

- The overall design is based on good design principles, for example, correct use of "is-a" versus "has-a", consistent levels of abstraction within class interfaces, etc.
- The project design is reasonably easy to understand based on class names, class interfaces, and other prominent parts of your project files.
- The code follows good programming practices (e.g., deleting dynamically allocating memory when done with it, avoiding overly complicated control structures, ...).
- The source code is generally readable.

30% - Implementation Quality – meets the requirements for FlashPhoto.

- The program builds without error by running "make" in the root directory of the project and runs on the CSE Lab machines.
- The implementation contains all the required items described in the four groups of features detailed above (image saving/loading, image filters, new tools, undo/redo).
- The implementation includes an interesting special filter.
- You implementation exhibits the correct behavior (e.g., each filter and new tool works as described above, and there are no undesirable behaviors that occur during the course of our testing).

10% - Quality of the individual programmer blog entries, including evidence that each team member has made a valuable contribution to the final version of the code.

- Each programmer blog entry contains a code snippet that is written by that programmer and appears in the final version of the group submission.
- The blog entry provides evidence that the programmer has made a reasonable overall contribution to the project. (e.g., "This snippet is from the (insert class name) class, which is one of the (insert number) classes I wrote for our team."
- The snippet itself exhibits good coding style, and is easy to read and understand.
- The snippet is explained in terms of its purpose, why it is illustrative, coding style, etc. so that the blog entry is educational for others who might need to know about your team's project code.
- The programmer blog entry is well written, and satisfies all the formatting and related criteria mentioned above.

Total: 100%