

Assignment 5: Filter

Help Session: Thursday, October 22, 7:00 pm, Motorola

Homework Due: Thursday, October 22, 5:00 pm

Algorithm Due: Thursday, October 22, 5:00 pm

Program Due: Wednesday, October 28, 11:59 pm

Introduction

In this assignment you will be exploring the area of image processing. Image processing can be as simple as cropping an image to fit in a certain space, or can be as complicated as removing sinusoidal noise in the image generated as a side effect of the medium through which you obtained it. The realm of image processing covers an infinite space of operations that can be applied to images and the combinations thereof. You will be implementing just a few of the most common image processing algorithms (which, conveniently, happen to be the more useful ones). Play with the demo found at </course/cs123/demo/filter>. If you are looking for pictures to mutilate/enhance, check out the </course/cs123/data/image/> directory.

Demo

Using the demo for this project is as simple as selecting a filter, dragging an outline box on the image with the left mouse button (this is assuming you have loaded an image...), and clicking “Apply” to apply the current filter to the selection. Clicking outside the box, or clicking with the middle or right mouse button will cancel the selection. If there is no selection, the filter will be applied to the entire canvas.

Several of the filters have their own controls which customize filter-specific parameters. You also have the option of “synching” the marquee selection rectangle on all canvases for testing purposes so you can, for example, apply the same filter easily to the same areas on the demo canvas as well as your own. We hope this will make it easier to detect subtle differences between your filter implementations and the demo’s. This does not mean, however, that your filters have to be pixel-perfect replicas of our own.

There are also options for extra filters. The TA demo includes a special filter called seam carving. This is an algorithm that was published in SIGGRAPH a couple summers ago. Feel

free to try your hand at it, although if you decide to take CS224 you may also encounter it there.

Design

The focus of the program is completely on the filters. You won't need to draw the correct selection boxes, or deal with the GUI in any way. Thus, most of the design work is already done, leaving you to focus on the algorithms.

Requirements

There are four filters that you will need to implement. The required filters are invert, blur, edge-detect and scaling. We have already filled in an example grayscale filter for you, so you should check that one out first to get an idea of how you should structure your filters. In detail:

- **Invert:** This filter should be pretty straightforward. All it has to do is invert the color of each pixel in the selected area.
- **Blur:** You must create a blur filter with a triangle kernel. This filter can be a bit tricky, since the blurring is achieved by taking a weighted average over an area of the image. The radius of this area is chosen by the user, so flexibility is a must. However, you are only expected to deal with odd radii – if the user selects an even radius, increment it or decrement it by one and document your choice. One must be extremely careful of boundary conditions when writing this filter.
- **Edge Detect:** You must create a filter that will detect the edges of an image. Run the edge detect on the grayscale images. Grayscale images have the same intensity for red, green and blue. Any resulting edges with an intensity under the user-specified threshold should be set to black. The display of the remaining edges is up to you. We did not discuss edge detection in class, so we suggest using the Power of the Internet. Search for 'Sobel filter.' Talk to a TA if you're unclear on what to do here.
- **Scale:** You must allow the user to scale the image up as well as down. This filter is one of those which has an obvious solution. . . which is wrong. Blindly replicating pixels to fill in gaps when scaling up and removing excess pixels when scaling down doesn't quite cut it. You will need to find a good, efficient way of creating a resized image that looks like it is simply larger or smaller, *not* like it has been put through a cheese grater. **Simply implementing resize as it is shown in the slides will lose you points.** There are faster and cleaner ways to implement scaling. Note that it may be easier to implement horizontal scaling first and add vertical scaling later.

Support Code

The support code for this assignment will consist of the same user interface which you have seen in your other assignments, and templated files for each of the filters which you will be creating. Although the GUI is more complicated, nearly all the work of dealing with the GUI is taken care of for you. You'll have to instantiate a GUI and a canvas, but that's about it. And, of course, you must be aware of what the GUI is expecting you to do - this is clearly documented in the headers of the support code classes.

The important header files are `FilterCanvas.h`, and `Filter.h`. They are very small, but important, so be sure to read them! Aside from the various `Filter` subclasses you will need to fill in, you will also need to edit `MyFilterCanvas` to handle GUI notifications. (Look at the example on how grayscale is being handled) Aside from `MyFilterCanvas`, there is another subclass of `FilterCanvas` that we have provided for you that will only be necessary for seam carving (it let's you paint on the canvas.) If you are interested in implementating seam carving, you will need to have your `MyFilterCanvas` subclass the `FilterDrawingCanvas` instead of `FilterCanvas`. See a TA if you're not sure how this would be set up.

Assertions

Assertions provide a convenient way to check for logically correct operation at various points within a program. The `assert` function takes a boolean (or usually an expression that evaluates to a boolean) as an argument. If the boolean is true, the program proceeds normally. Else, if the boolean is false, `assert` prints out a message stating that the assertion failed and the program is aborted.

Debugging this assignment will be a lot easier if asserts are used to check that the current indices being examined fall within an expected range. The assertions will actually fail and report what was logically incorrect; such information is much more helpful than just having seemingly random skewing and banding in your filter's output. For more information on assertions, type `man assert` in a shell.

Final Notes

In an effort to make some of the expectations more clear, here's some supplementary information. Speed is very important in this program. Less so than a working algorithm (so get them working first), but still important. For this reason, you are forced to use the raw memory directly (see the support code) so no `setPixel`. Try to make the instructions in your inner loops as clean and distilled as possible. There are a lot of boundary conditions, just as in brush, so try to come up with *good/fast* solutions to this issue. Look out for repetitive instructions, and needless multiplications. Using 1D arrays instead of 2D arrays will be faster because the data will be contiguous in memory. Also, when dealing with large amounts of contiguous data in memory (i.e. a row of an image or region of interest), it can

be faster to use `memcpy` and `memset` to copy data from temporarily allocated memory space or fill memory with a specified value, respectively. For more information, see the man pages for each function.

Extra Credit

There are millions of different effects that you can perform on raster images. Play around with Photoshop, the GIMP, or some other image manipulation package to get some ideas, and try your hand at implementing them. The GUI allows for two “wildcard” filters (and of course can be expanded to allow more); take advantage of that. Here are some ideas for the unimaginative:

- Gaussian Blur: Blur the selection using a Gaussian kernel. (medium)
- Sharpen: Enhance the regions of high contrast in the image (medium)
- Emboss: Create embossing/engraving effects to the image (medium)
- Ripple: Make part of the image ripple as if it were a pond and a stone was dropped in it (hard) (remember the demo’s special brush in the Brush assignment?)
- Page curl: Make it look like the corner of the image has been curled up (hard)
- Seam Carving: It’s not too bad to make it work slowly for most cases but can be tricky to get it fast and robust. Even the less robust solution is pretty awesome.

Note: The above are just ideas. *All* the TAs have not implemented *all* of them and are therefore not able to guarantee support for your extra efforts (though they will try send you in the right direction). Also, the demo has some extra credit options that really aren’t “filters” in the sense that they incorporate animation and other temporal effects. But they are extra, and you would get credit for them, so go nuts.

Note: If you would like to add custom controls for your own filters or modify the default controls, see `FilterControls.H` and `FilterControls.C` for clean examples (the built-in controls). Adding GUI controls requires the use of some non-C++ specific Qt features which is not at all the focus of CS123, so if you’re thinking about adding/changing controls, feel free to visit your friendly TA staff first to give you an overview of what this would entail.

There is also a “Rotation” option built into the GUI. Rotation is a *really* cool filter to implement, but for the sake of time we are not requiring it. Nonetheless, we encourage you to give it a try:

- Rotation (by arbitrary angle): This filter is another that can be done “wrong”, but in this case it is painfully obvious that it is (you will get ugly, aliased results; unparalleled even by a “bad” scaling routine). However, being good CS123 students, the wrong way should never even enter your mind. The right way also has a few pitfalls. Depending

on how you tackle the problem, you may come out with excellent rotation, but it will take a long time (and that is no good). There is a really fast way for performing good rotation, and it just takes a little linear algebra (or a little research through graphics texts) to figure it out. Do your best to come up with a good way of rotating an image.

- In the interest of pure, vicious, exhilarating, processor-screaming speed there are a lot of tricks you can do. Also, notice that some filters are linearly separable. What does this mean? How can you use it to make your filter much faster?

Handing In

To hand in your assignment, type `make handin` at a shell prompt.

Handout:	Thursday, October 15	
Homework Due:	Thursday, October 22	5:00 PM
Algo Due:	Thursday, October 22	5:00 PM
Electronic Handin:	Wednesday, October 28	11:59 PM