Assignment 1: Brush

Comp175: Introduction to Computer Graphics – Fall 2011

Algorithm due: **Monday September 12th** at 11:59pm
Project due: **Monday September 19th** at 11:59pm

1 Introduction

Computer graphics often deals with digital images, which are two-dimensional arrays of color data (the pixels on the screen). These images can be generated in many ways including using a digital camera, taking a screen capture, scanning a photograph, or using a "paint" or "draw" program. In this assignment, you will write a simple painting application sporting several different types of airbrushes, as well as the capability to save the images you create. Check out the demo at xxxxxxxxxxx to get an idea of how your program should look and behave.

This assignment will give you a gentle introduction to programming in Qt/C++ using the Comp175 support code. Memory management and object-oriented design are critical. Youll also get hands-on experience with some graphics programming basics: youll learn about blending colors and drawing images, and learn to cache calculations and tighten loops to improve performance.

2 Requirements

If you have played with the demo, youll see that it includes a GUI with a canvas and controls. Dont stress the GUI setup is taken care of for you in the support code, should you choose to use it. All you have to do is write a program that adds on to the provided GUI and allows the user to draw on the canvas by clicking and dragging the left mouse button. You must implement several different brushes, taking into account different colors of pain, different paint flow rates, and different radii for your airbrush. Your brushes must incorporate the use of a mask (a copy of the airbrush distribution) as a mechanism for reducing computation.

All the brushes you are required to make should be circular in shape. They will differ from each other in the distribution of paint that they place on the canvas. The distribution describes how much paint is placed at a certain point based on its position relative to the point clicked by the user. Below are the details about the distributions for this assignment for this assignment. The first three of these are required (i.e. you MUST code them to receive full credit). For the ambitious, you may implement additional brushes for extra credit. Some suggestions are provided in the demo.

2.1 Mask Distributions

Constant This distribution will place an equal amount of paint at each pixel within the airbrushs radius.

Linear - This distribution will place a linearly decreasing amount of paint at each pixel as you move away from the center point. At the center point, you will have full intensity of paint, whereas at radius pixels away from the center no paint will be put on the canvas.

Quadratic - This distribution will place a quadratically decreasing amount of paint at each pixel as you move away from the center point (i.e., the amount of paint used as you move away from the center decreases with the square of the distance). As with the linear brush, at the center point, you will have full intensity of paint, whereas at radius pixels away from the center no paint will be put on the canvas.

2.2 Smudge Brush

In addition to the airbrushes described in the previous section, you will also implement a smudge brush which will result in a paint smear effect when dragged across your canvas. This brush ignores the paint color selected by the user, and you may use whichever mask distribution you think works best: constant, linear, quadratic, or something else entirely. Below is the pseudocode for the smudge brush; you just need to implement it. Implementing this brush will help you gain proficiency in memory management in C++.

There are two basic operations that must be implemented:

1. *Pick up paint.* This operation copies the part of your 2D image that is currently under the mask centered at the cursor. You should allocate a temporary buffer to be used as storage for this image

fragment. You do not want to copy the entire image only the part under the mask.

2. *Put down paint.* This operation blends the contents of the buffer you filled when you picked up paint with the region of your canvas that is under the brush mask.

Heres how the smudge brush effect works:

1. When the canvas is clicked, *pick up paint.*
2. When the mouse is dragged, *put down paint* and then immediately *pick up paint* again, taking the paint you just put down into account.

2.3 *Optional* Suggestions for Additional Brushes

We have provided two slots for extra bushes you may wish to create for extra credit. You have complete freedom regarding what you may choose to do. The distribution can be whatever you want; it doesnt even have to be an airbrush. In the demo, we have provided a spiderweb brush, which performs a line-drawing effect, as well as a ripple brush which introduces animation. You can rename the radio buttons in the GUI by editing mainwindow.ui. You can earn up to 10 points of extra credit with these brushes, but were going to favor quality over quantity. You will get more credit for one really awesome brush than for two run-of-the-mill brushes.

2.4 *Optional* Artwork

After youve gotten your brushes working, feel free to create some artwork! Hand in one or two original images along with your code (you can use the “Save” option we’ve provided). We’ll post the best submissions on the course web page, and we may also use your artwork as test images for the upcoming Filter assignment.

2.5 Grading

The grading expectations for this assignment are as follows:

- To earn a C - Hard code the masks for each of the required brushes
- To earn a B - Compute resizable masks
- To earn an A Incorporate alpha blending (transparency): final color is a blend if source (color selected by user) and destination (color on the canvas)
- Extra credit:
  - Each additional unique brush (up to a maximum of two) +5pts
  - Load mask from an external file +5pts

3 Paint

Colors on the canvas are represented by three 1-byte unsigned char values, one for each of red, green, and blue (r,g,b for short). This means that each channel can range from 0 to 255, where 0 means that there is no contribution from that color and 255 means that there is full contribution from that color. This translates to 16 million possible color combinations. One important question is “How exactly do I decide what color(s) to put on the canvas?” To decide this, you must consider each of the following components:

1. The distribution of the airbrush
2. The existing color of the pixels you are painting onto
3. The color of the brush

You will want to blend the colors on the canvas with the color of the airbrush. We are not going to give away too many details about how to do this one of the important things about this assignment (as well as the assignments to come) is that there are often many valid ways to solve the problem presented. You should try to figure out your own way of tackling the problem, while obtaining results consistent with the demo.

4 Getting Started

To begin this assignment, you will first want to copy the support code into your working directory. Type ****************************

A Qt project is provided containing support code for all projects you will complete over the course of the semester. You may choose to use Qt Creator to edit and build your project. You will need to make additions to many of the files, including header files. To get started, you may want to look for places marked `[BRUSH]`, where we have left some helpful tips in the comments.

Code carefully! Poor design and coding decisions now will come back to haunt you later. You will be using the same support code for the rest of the semester, as your projects continue to build on one another.

5 Support Code

The Comp175 support code library is adapted from libraries developed for Prof. Andy van Dam’s CS123:

Introduction to Computer Graphics course at Brown University. You may want to reference their online documentation, available at:

`http://www.cs.brown.edu/courses/cs123/docs.html`

There you can browse the sources and see the relationships between the classes graphically. For this assignment, these are the classes you will care most about:

5.1 settings.[cpp,h]

You don't need to edit this file. The static `settings` variable, declared in `settings.cpp` and `settings.h`, allows you to easily retrieve parameters set by the GUI interface. You will need to use the Brush settings in order to correctly implement this assignment. These include:

```
bool brushDockVisible;
int brushType;
int brushRadius;
int brushRed;
int brushGreen;
int brushBlue;
int brushAlpha;
```

The user can interactively change various parameters for the airbrush including the color, radius, flow, and distribution. All the sliders pass integers in the range of 0 to 255. It's not necessary to query the sliders for their values, because the value will be automatically passed to the static *settings* object every time it is changed.

When the user selects one of the airbrush distribution radio buttons (constant, linear, etc.), the `brushType` variable will be set according to an `enum` defined in `settings.h`.

The *Brush Radius* slider, which has been bound to the `brushRadius` setting, controls how large the brush is. The value of the slider should correspond to how many pixels away from the center the mask extends. Note that the mask should always have an odd width and height: if the radius is 1, the mask width and height should be 3, if the radius is 10, the mask width and height should be 21, etc. If the radius is 0, the mask width and height should be 1 (implying that the brush will only color the single pixel under the cursor).

The *Alpha* slider, which has been bound to the `brushAlpha` setting, controls the flow of paint laid down when the brush is applied. This is similar to the *opacity* feature in many image editing tools. If the current brush is a constant distribution, and the flow is set to 255, then the color being laid down should completely replace the existing color on the canvas. If the flow is set to zero, then the brush should have no effect. Intermediate values should somehow blend the existing pixel color and the color of the paint. We are being intentionally vague here; it is up to you to implement a paint combination model that mimics reality.

5.2 canvas2d.[cpp,h]

`Canvas2D` will be your implementation of a two-dimensional image canvas. You will use this canvas for all projects in the course which require pixel-level manipulation of images (namely Brush, Filter, Intersect, and Ray).

The `Canvas2D` class that you are given extends `SupportCanvas2D`, which provides several key features for you:

- You can resize the canvas by calling `setSize()`. The canvas is not automatically resized when the GUI resized; rather, scroll bars allow you to view the entire image.
- Image load and save: `loadImage()` and `saveImage()`
- Marquee selection via right-mouse drag: you can get the two corners of the selection by calling the `marqueeStart()` and `marqueeStop()` functions.
- Display buffer to avoid threading issues
- Mouse events: `mouseUp()`, `mouseDown()`, and `mouseDragged()`.
- Access to the raw image pixel data with `data()`; more on this in the next section.

5.3 Dealing with raw image data

It is one thing to write an airbrush, but another thing to make it *fast.* Make yours fast. You will need to pay close attention to the `data()` method: this method will return a pointer to the beginning block of memory used to store the canvas. In graphics, you will almost never see `get()` and `put()` methods used for pixels; they just can't keep up.

You may notice that `data()` returns type `BGRA*`. This is a pointer to the beginning of an array of `BGRA`s. Each `BGRA` is a 4-byte struct with a blue, green, red, and alpha component (in that order) ranting from 0 to 255. You can either access the members of each `BGRA` using the `.b`, `.g`, `.r` and `.a` fields, or you can *cast* the `BGRA*` to an unsigned `char*`. The pixels are stored in *row-major order.* This means that each consecutive `BGRA` is one pixel to the right of the last one, and they wrap around at the border of the image to the leftmost column of the next row.

5.4 Memory Management

A great deal of time has been spend to ensure that the support code does not leak memory. We expect you'll do the same with your code. Be sure to `delete` any memory you `new` and `free` and memory you `malloc`. Don't mix and match, and don't forget to `delete` or `free`. Don't allocate everything on the stack; it's inefficient and messy to do so. We'll be assessing heavy penalties if your program leaks memory of uses the stack carelessly.

5.5 Other Files

There are many other source files that you will be building along with this project. While you don't need to worry about what they do (or will do), feel free to explore. We've intentionally provided you will *all* the code you're going to get for this course right from the start, ad each of your projects will build on the previous projects. Don't like how we've done something? Feel restricted? Feel free to hack to your heart's content! Just be sure to keep the support code intact for your future assignments. Back up your files often, or better yet, use version control!

You might algo get the sense that something is... missing. That's because things *are* missing. Feel free to extend the support code to provide whatever functionality you need. Since this course will get complex relatively quickly, we recommend that you use the class and inheritance facilities that C++ provides. Good organization from the start will make your life much, much easier in the long run.

5.6 A few words on precision

We've all run into this snafu at some point in our hacking careers: *beware of integer division!* 255 / 256 = 0. To avoid this problem, just cast one of them to a `float`. Alternatively, if one of the numbers is a constant, append ".0" or ".f" to the end of it.

6 How to Submit

In Comp175, there will be two deliverables for each assignment. One is the typical electronic copy of your projects source code. The other is a write-up of the algorithm(s) used in the project, due a week BEFORE you hand in your code.

Complete the algorithm portion of this assignment by yourself without any help from anyone or anything except a current Comp175 TA, the lecture notes, official textbook, and the professor. You may use a calculator or computer algebra system. All your answers should be given in simplest form. When a numerical answer is required, provide a reduced fraction (i.e. 1/3) or at least three decimal places (i.e. 0.333). Show all work.

For the project portion of this assignment, you are encouraged to discuss your strategies with your classmates. However, all students must turn in ORIGINAL WORK, and any collaboration or references must be cited appropriately in the header of your submission.

Hand in the assignment using the following commands:

- Algorithm: `provide comp175 a1-alg`
- Project code: `provide comp175 a1`

Recall from the syllabus that you have two homework passes to use during this semester. Each pass will grant you a penalty-free 72 hour extension, and you may not use more than one pass on a single assignment. Because solutions will be made available for each assignment 72 hours after the due date, no submissions will be accepted more than 72 hours late. Extenuating circumstances may be evaluated on a case-by-case basis by the instructor with documentation by the dean.