Digital PaintChat:
Examining and Expanding
Collaborative Tools for Digital Artists

by Helen Jiang

Boston College Honors Thesis
Advisor: Professor William Ames
May 8, 2012

Contents

Abstract

The digital world has revolutionized virtually every aspect of peoples' lives. Many professional illustrators have begun to use digital tools in order to simplify their drawing process and make it more efficient. There are many different software programs that artists use, each fitted to meet different needs, such as photo manipulation, painting, or animation. Although digital art is constantly evolving and expanding, and there is little research on how artists interact with digital media.

Communication is one of the areas in which technology has had the most profound change. People from anywhere in the world have the ability to contact each other at a moment's notice. This reality has lead to new, fruitful collaborations in a variety of fields. Thus far, there are no fully-functional artist tools that enable direct communication between artists. My thesis involves the planning and implementation of such a program.

I first conducted a digital arts survey to gather data on how current digital artists interact with the programs they are using, the way they use tools that are common among all digital art software programs, as well as the shortcomings of these tools and digital art in general. The survey was answered by both amateur and professional artists from online art communities, the majority of whom have been using art programs for over four years. Afterwards, I began programming a basic drawing program based on the results of the survey, and added networking capabilities.

1. Introduction

1.1 What is Digital Art?

Digital Art is defined as any art that was created with the aid of a computer. There are a multitude of types of digital art, such as Graphic Design and 3D Modeling. The focus of this thesis is Digital Illustration, which is defined as the use of digital tools under direct manipulation of the artist to produce 2D images, usually through a pointing device such as a mouse or a drawing tablet. In simpler terms, an artist uses a mouse or tablet to draw.

There are many applications that have been developed to aid artists in digital illustration. They contain a virtual canvas with a large amount of tools and instruments, some meant to mimic the purpose of tools that traditional artists use, and others that do not exist outside of the computer. The digital-only tools are what give digital art a distinct look and feel from traditional art. Unfortunately, there is very little academic research on digital illustration.

1.2 Drawing Tablet

A drawing tablet is an pointing input device that allows users to hand-draw images onto a computer. It usually consists of a surface to draw upon and a special drawing pen. Coordinates on the drawing surface directly correspond to coordinates on the computer screen. Thus, unlike a mouse, the pen location is not calculated relatively to the original location of the cursor. Aside from being able to transpose brush strokes easily onto a digital canvas, the drawing tablet's other most important feature is that of pen pressure detection. With pen pressure enabled, artists can simulate the effects of light versus heavy strokes on a canvas.

1.3 Digital Art Applications

Figure 1: Commonly used digital illustration applications for drawing and painting. Clockwise from the top left: Adobe Photoshop, Adobe Illustrator, GIMP, OpenCanvas, Clip Paint Lab, SAI PaintTool, Corel Painter

There are generally two types of digital illustration applications, raster (or bitmap) and vector. In raster programs, information is stored in individual pixels, and as the artist points at certain pixel locations with their mouse or tablet, the pixel information at those locations is changed and replaced. These rows and columns of pixels can be on different layers for added complexity. Raster programs are thus good at creating effects with less rigid lines for painting, as well as photo-retouching. Because the images created by these programs are pixel-based, their resolution is pre-determined. A well-known example of a raster program is Adobe Photoshop, which is mainly used for manipulation of photos but is also widely used as a painting program by digital illustrators.

On the other hand, in vector programs, content is stored as mathematical formulas that describe curves (open paths) and shapes (closed paths). Artists construct paths in a much more precise manner, as they are able to control the locations of all the points along a path. Art generated by vector programs thus have more well-defined edges, and are more so used for

graphics meant to convey information, such as maps and icons. Vector images are also not fixed in resolution, as their mathematical nature allows the artist to zoom without constraints. A well-known example of a vector program is Adobe Illustrator. The vector-based nature of Adobe Illustrator makes is difficult for it to be used as a painting program.

1.4 Collaboration

The idea of a online art collaboration for digital artists is not a new one. In all of its implementations, however, many crucial features are lacking. There are many collaborative projects where different artists edit the same canvas. Most of these projects do not support real-time editing where artists can work at the same time, and those that do only support the most simple tools.

The most commonly used "PaintChat" applications where artists draw together over the internet are through web browsers. Some examples of these are iScribble and Japanese application ShiPainter. Because they are browser-based tools are rather primitive and there are limitations based on browser type. None of the web-browser PaintChats by themselves support pen pressure, and the add-ons that need to be installed in order for pressure to be enabled on certain drawing boards do not universally work on all platforms and browsers.

1.5 Tools

Though each digital illustration application features different tools, there are many tools that are shared between all of them. Applications may implement the tools using different algorithms, which result in a different look and feel.

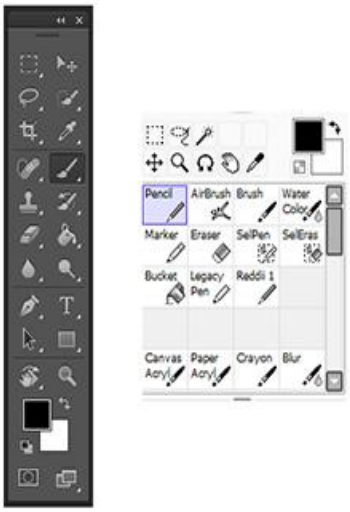

Figure 2: Comparison of Adobe Photoshop toolbar (left) with SAI PaintTool toolbar (right)

In the Adobe Photoshop toolbar, core tools are shown, sectioned off by the type of utility they offered. If a tool is clicked and held, a few options for variations of those tools are offered. For example, if the erase button is clicked and held, the user will be able to access different types of erase, such as the Background Eraser tool and the Magic Eraser Tool. In the SAI Paint Tool toolbar, tools are separated between select tools, manipulation of canvas view, and editing tools. The editing tools section has empty slots which allows users to create their own custom tools. These custom tools are basically the default tools with certain changed settings that are saved and renamed to a new tool.

Tools were important in the survey that I conducted as the first part of my thesis. The first section of the survey was aimed at evaluating artists' relationship with the tools that were common among all digital art programs (brush, eraser, paint bucket, zoom, etc).

1.6 Tool Customization

All digital art programs, to some degree, allow artists to customize tools. In the most simplistic programs with tool customization, such as Microsoft Paint, the user is able to

manipulate the size of the brush tool. In the more complex programs that I am investigating, users are able to change a variety of tool properties. For example, the brush tool has size, opacity, flow, persistence, color, texture, and shape properties. Tool customization gives artists flexibility as well as an opportunity to grow their own methods and style while painting. If customizability does not have enough range, the program is in danger of limiting artists to a certain look and feel. In my survey, I examined the utility of tool customization and whether or not certain tools required more customization than others.

2. The Survey

The survey was divided into four sections; Tools, Customizability, Improvement, and Input. I distributed the survey to the massive online digital art community, and used social networking sites such as Tumblr to gather answers from people of a variety of skill and interest levels. There were a total of 197 responses.

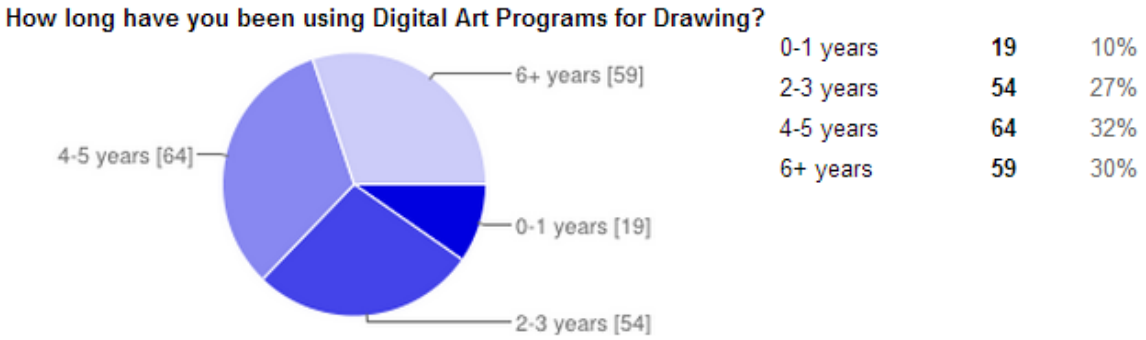

Figure 3: Pie Chart of years experience in Digital Art in my Digital Art Tools Survey

2.1 Tools

The tools section first asked artists to choose which tools are necessary for a basic art program. The purpose of asking this question was to determine what a bare-bones program would need in order to be effective. Artists were to choose from a list of tools that are commonly shared among all digital art programs, and were also allowed to suggest other tools. Artists were then asked to rank the usefulness of the same tools in the list on a scale of 1 (least useful) to 5 (most useful). This additional step was so that the importance of each tool could be closer analyzed.

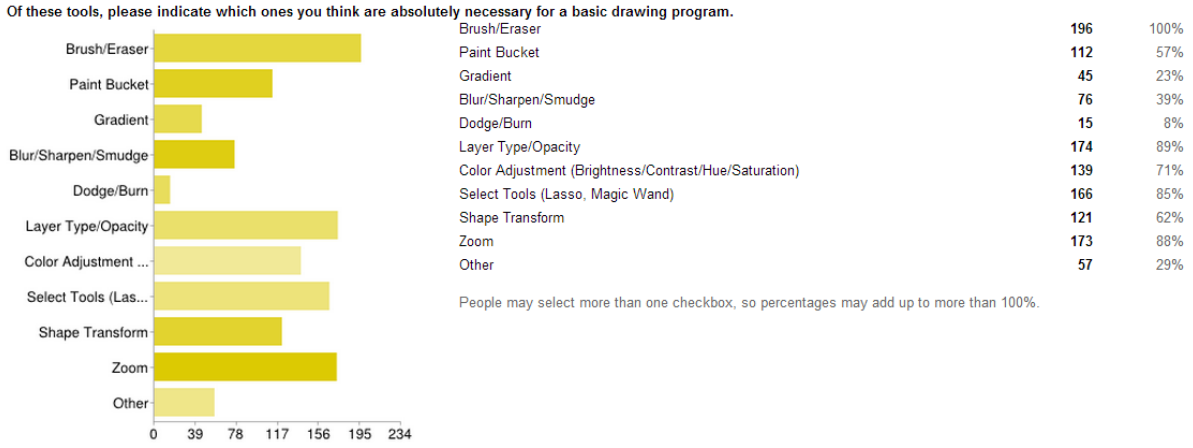

Figure 4: Survey answer summary for necessary tools in a basic drawing program

In the results, the tools that were thought to be absolutely necessary in the first question also had higher average usefulness ratings. Of the ten tools that artists had to consider, the top seven all had over fifty percent of survey takers who thought they were absolutely necessary for a basic drawing program as per the first question. The same top seven tools also all scored an average usefulness rating of 3. These seven tools were, in order from highest to lowest affirmative ratings: Brush/Eraser, Layer Type/Opacity, Zoom, Select Tools, Color Adjustment, Shape Transform, and Paint/Fill Bucket. Most importantly, three tools were not considered useful by digital artists. These three are, from lowest to highest affirmative ratings: Dodge/Burn, Gradient, and Blur/Sharpen/Smudge.

In the first question, there was also an open-ended option where artist could input their tools other than the ones listed that they thought were absolutely necessary in a drawing program. Several tools were mentioned by artists multiple times, which emphasizes their importance because the artists taking the survey placed them there on their own accord. These suggestions have been condensed as such:

- A popular category of suggestions involved color, and tools to help artists choose color. These included a simple color picker, as well as swatch panels where colors could be saved, and mixing panels.
- Artists also expressed interest in user friendly rotate and flip tools that would enable them to quickly view the canvas from virtually any angle without much effort.
- Although the Blur/Sharpen/Smudge tool was unpopular, artist did commonly suggest some sort of blending or "watercolor" tool. This tool owes its popular to the number one most used digital art program by artists taking the survey, called SAI Paint Tool.
- Tools specific to the creation of pixel art were also requested. Essentially, this means artists want to be able to manipulate images pixel by pixel, using a binary brush.
- Other tools such as Undo and Eyedropper were suggested, and their usefulness was immediately apparent.

A third, open-ended question asked artists to list tools that they frequently use from the top of their head. This was to find commonly used tools that were otherwise missed in the previous questions. The main difference was that these tools, although commonly used and helpful to artists, were not strictly necessary. Artists rely on these kinds of tools to make their process simpler and more efficient, but could live without these.

In the response to this question, the interface came up quite a bit. Artist want to be able to access certain features with the least amount of effort possible, but in a way that is still intuitive. Hotkeys were a big concern--they should be easy to use as well as easily customizable. Windows were also preferably customizable in their position and size, so that they could arrange their canvas according to their comfort. Artists also heavily customize their brush settings, and wanted there to be a way to save custom brushes and distribute them as well.

Many times artists listed specific features of tools such as brushes, selection tools, and layers that they used the most often. This presented a good perspective on how wide the scope of each of these tools should be.

Other notable tools were listed as common used by multiple artists:

- The color wheel was once again frequently mentioned, as artists work with color often and want to be able to switch between and save colors palettes easily
- Move and Zoom tools, in addition to the afore-mentioned flip and rotate, were required for flexibility in the way the artist viewed their canvas.

2.2 Customizability

In this section, there was just one question to rank the importance of customizability from seven commonly customized tools: Brush, Eraser, Color Palette, History, Fill Bucket, Layer and Zoom.

The results of this section were very simple; of the seven tools, five of them (Brush, Eraser, Color Palette, Layer, and Zoom) scored above three on a scale of one to five. This is especially interesting in the case of the Zoom tool, which usually is not highly customizable in digital art programs.

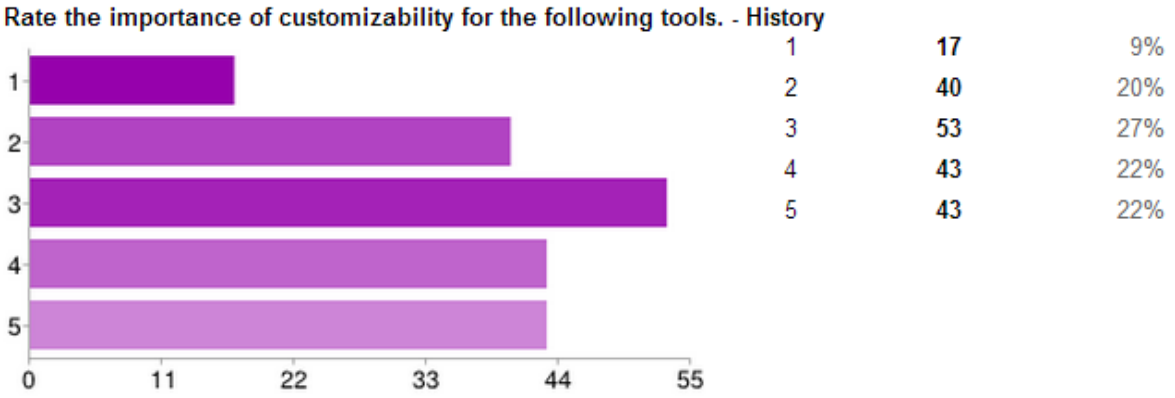

Figure 5: With an average score of 3, there was no consensus on the importance of customizability for the History tool.

The two that fell under three on the scale were History and Fill Bucket. I had considered making history more customizable as a new tool to introduce to artist, but this result left me with doubts as to whether artists really needed it. This would be further investigated on in the last question of the Input section of the survey, where artists were asked for their input on such an option.

2.3 Improvement

In this section, artists were asked open-ended questions about their frustrations with currently available digital tools as well as their suggestions as to how these tools could be improved. They were also asked if they had ideas for new tools.

In the question about frustrations, artists were given a chance to point out what they believed were the shortcomings of the programs they currently used. There were quite a few commonalities in artists' frustrations as well. These "complaints" ranged from problems with current tools to small pet peeves about the program that hindered but did not pose a large obstacle to the drawing process.

There were quite a few commonalities in artists' frustrations, especially with the selection tool, and anti-aliasing getting in the way of Selection. Artists also noted that they preferred certain implementations of tools in one program over another's, and expressed wishes to combine those tools into one program for easy access.

There were some notable suggestions—for example, a vector-based tool that would increase accuracy and efficiency.

2.4 Input

Artists were asked for input on two specific innovations to digital art programs. The first is the main aspect to my program--a collaboration feature that would allow multiple artists to work on the same canvas. Artists listed similar tools already in existence, as well as their general opinion of the idea and its utility. The second new feature in question was "Branched History", which would give artists the ability to return to stages of their art even if they have overwritten it by undo.

For the collaborative art tool, there was generally a positive response. Artists listed pre-existing programs that allowed them to collaborate, but also noted that the tools in these programs were generally limited because the programs were web-based. An early version of OpenCanvas, a non web-based program, had network capabilities. However, networking was removed in later versions and artists expressed frustration with the limited toolset and unfriendly user interface, as well as connection issues.

For the idea of branched history, there were mixed responses. The general consensus was that the tool may be an innovative idea, but was not practical or completely necessary to improve the creation process. Artists expressed worries it would only be rarely used in certain situations and might be confusing for artists when keeping track of the steps they have taken thus far.

3. Programming SimplePaint

3.1 Java

I chose to program the application in Java for its ability to run on any operating system. Many programs were limited to one operating system, which many artists expressed frustration with in the Digital Art Tools survey. An example of this would be SAI PaintTool, which was confirmed by developers to only be available on the Windows operating system. In response to SAI PaintTool, another company developed Clip PaintLab, which was made to work only on Mac OS. In addition to this consideration, Java was also the programming language that I was most familiar with. It has extensive graphics and networking capabilities in its libraries already.

3.2 Program Organization

The canvas of SimplePaint appears white upon startup, but actually consists of three BufferedImages that stack in a manner similar to that of layers. The first layer is the backgroundLayer and is uneditable. It is what makes the background of the canvas appear white rather than grey. Eventually the view of the background layer should be toggle-able to show transparency in the form of a grey and white checkered pattern as is custom. The second layer is the baseLayer, which will hold the images drawn, and acts like any editable layer on most digital art applications. The third layer is the strokeLayer, which only temporarily holds strokes. The strokeLayer was introduced in order to differentiate between compositing methods within a single stroke itself versus how the stroke blends with the rest of the image (on the baseLayer). These three layers are initialized at the beginning when a new canvas is created and are stored as private variables in the canvas class. The canvas class, called NetworkShapeCanvas, extends JComponent. In the implemented paintComponent class, which runs every time repaint() is

called, the three layers are all drawn, in the order of backgroundLayer, baseLayer, and strokeLayer on the very top.

In my Program I added a PenListener from the JPen library. Although at first I also added a MouseListener, I eventually used the penButtonEvent method from the PenListener Interface to detect and differentiate mouse clicks from pen clicks. Because input from the mouse only included coordinates while input from the pen also included pressure and tilt, I had to program them separately. The penLevelEvent method from the PenListener Interface detects and runs when the pen is in close contact with the surface of the drawing tablet. This includes when the pen is hovering slightly above the surface as well as when the pen is touching the surface. To only make changes when the pen is down, I encapsulated everything in an 'if statement' of (PressureDetected > 0).

The program also contains Booleans to keep track of which modes and tools are currently active. These Booleans can be changed through the GUI by the user, by calling a method in the canvas class to update values of the Boolean variables.

3.3 Vector & Raster

Although most digital illustration applications are either raster-based or vector-based, I wanted to combine the two. SimplePaint appears as a raster-based program but the way the information is stored in the underpinnings of the program is vector-based. For example, the brushStroke information is not stored on a pixel-to-pixel basis, but rather as points along the stroke with pressure, tilt, color, and shape information. This setup will give users more flexibility, allowing them to have the liberty of changing image resolution in ways that raster-based programs cannot offer, but also having the look and feel of a painting program in a way that vector-based programs do not have.

3.4 Pen Pressure

Pen pressure enables artists to create more realistic strokes with their drawing tablets. One of the biggest complaints of current "paintchat" networking applications is the incompatibility with pen pressure. I searched for a java library that would help detect pen pressure from common tablets such as the Wacom Intuos 5 Tablet that I used. On the website called SourceForge.com, I found a library called JPen that could access drawing tablets and pointing devices using Java 5. It includes event and listener architecture. Device access is implemented through providers and conains providers for Linux, Windows, Mac OS X, and the java system mouse.

For those operations that would be affected by the pen's additional inputs of pressure and tilt, I made sure to make an extra constructor that would store those initial values. The instance of the operation is initialized when the pen first touches the tablet surface (within penButtonEvent method from the PenListener Interface). When the pen is dragged, the x-coordinate, y-coordinate, pen pressure, and tilt are continuously passed into the instance of the operation (within the penLevelEvent method from the PenListener Interface). A method within the operation is then called to continuously draw and repaint, which achieves the effect of varying pen pressure. For a brush stroke, pressure was multiplied against a base size and opacity. Since pressure was a double from 0 to 1, the base size and opacity were in fact the maximum size and opacity available and could only be achieved if the user pressed down with maximum pressure. Though this was not coded into the user interface, it is possible to make either or both size and opacity static if the user wishes to do so.

3.5 Operations

Features were divided into two categories—those that would affect the image and therefore be "undoable", and those that would not. Examples of the former are brush strokes, erase strokes, and select. Examples of the latter are zoom, flip canvas view, rotate canvas view, and the undo feature itself.

For the "undoable" features, I created the abstract class Operation which is extended by all classes of "undoable" features. This is beneficial in the long run for the undo and networking features. For the History, instead of having to deal with lists of mixed class types, a simple ArrayList of type Operation holds all the information. For Networking, Operations are passed through an ObjectOutputStream and then casted to type Operation when they are being read to the client.

The abstract class Operation contains four important methods. The first two are startOperation and endOperation, which control what happens at the tailends of the Operation. The third method is the draw method. I will use a brushStroke to illustrate how these three methods work. When the pen is first detected to have hit the surface of the drawing tablet, or when the mouse is pushed, startOperation is called. When the pen or mouse is dragged, the draw method is repeatedly called. When the pen leaves the surface or the mouse click is released, the endOperation is called. The fourth method is the redraw method. All instances of Operations should contain enough data to redraw themselves on a canvas. Thus, the entire process of a drawing could be recreated if the History list of Operations was run through and the redo method was called each time. This also helps with Networking—once the instance is passed to the client, the redraw method is called onto the client's canvas.

3.6 Compositing

In order to achieve some of the effects I desired in this program, I had to sometimes manipulate the compositing of the baseLayer and the strokeLayer, especially in manipulating the alpha values to change opacity. I will first give a brief introduction into how compositing works.

Composites define how two inputs are blended together mathematically. In Java, the AlphaComposite class supports standard Porter-Duff compositing rules, which were fully developed by Porter and Duff in a 1984 paper. For a BufferedImage, the two inputs are the SRC (source) and DST_IN (destination in). The SRC input is simply what is being drawn onto the BufferedImage. The DST_IN input is what the BufferedImage already holds. The composite blends this using a formula, obtaining the output which is named DST_OUT (destination out). What formerly was the DST_IN input is then replaced by the DST_OUT. The default compositing mode for BufferedImages is called SRC_OVER, which is the same as a normal painting operation. In SRC_OVER, if the object being rendered (SRC) overlaps the previously rendered pixels (DST_IN), the SRC is rendered over the DST_IN. If the Alpha values overlap, and the pixels are not opaque, you will see the overlap.

Aside from SRC_OVER, many different possible operations exist. IN operations such as SRC_IN work as clipping, where only the SRC input in the area overlapping the DST_IN is outputted. On the other hand, the ATOP operations allow the SRC input to draw only in areas where there is overlap, but also preserves the DST_IN in the output as well (unlike IN).

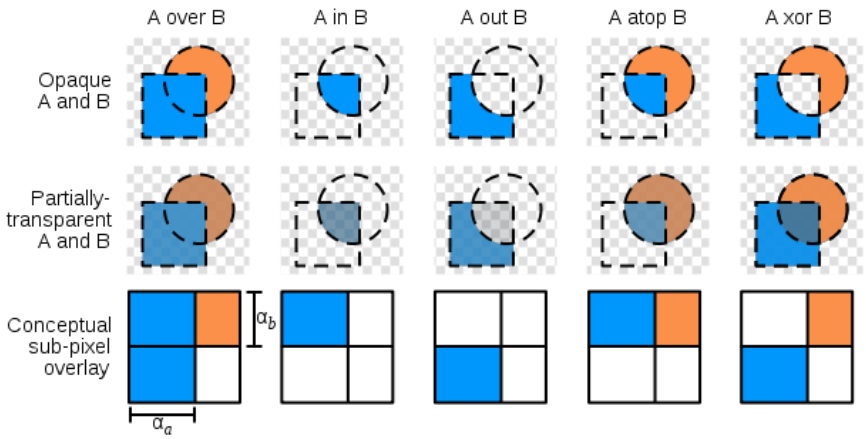

Figure 6: Illustrations the over, in, out, atop, and xor operations outlined by Porter & Duff in the 1984 paper

Because I had many specific blending properties in mind, there were times when the basic compositing methods outlined by Porter and Duff and built into Java were not sufficient and I had to create custom composites. This will be explained in greater detail within the individual Operations where this was necessary.

3.7 BrushStroke

Each unique BrushStroke contains a set shape and color. The size, opacity, and tilt variables, however, can change along different points of one BrushStroke. Size and opacity all change according to pen pressure, and tilt changes according to pen tilt. Every instance of a BrushStroke contains a 2D array where each row contains an array of four doubles. The four doubles are in the order of x-coordinate, y-coordinate, pen pressure, and pen tilt. Every time the pen passes in a coordinate, a new row is added onto the 2D array.

Bresenham's Algorithm is used to draw the stroke. The algorithm determines which order to form a straight line between two given points.

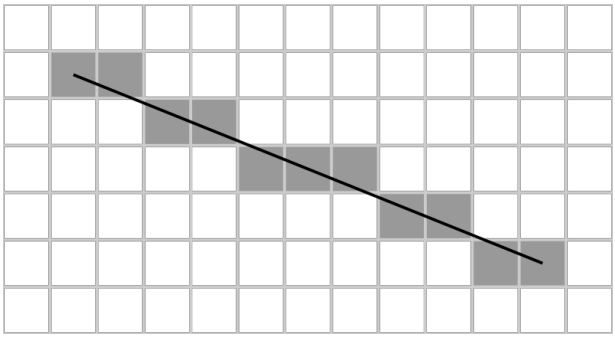

Figure 7: A diagram illustrating Bresenham's line algorithm

Rather than drawing individual pixels at the locations given by Bresenham's, the shape of the brush is drawn there and the locations are stored to support the vector-based backing of the program.

At first, the stroke was drawn directly onto the BufferedImage that contained the rest of the image. The effect that this created was undesirable, however—because of the default SRC_OVER compositing mode, if the stroke was not at full opaqueness (alpha = 1) areas of intersection would visibly show the overlap. This poses a problem for artists because the effect gives artists less control over the opacity of the image.

Figure 8: Self-Intersecting BrushStroke when drawn on baseLayer BufferedImage of SRC_OVER composite

For example, if an artist attempted to cover a larger amount of area with a single stroke with a color of low opacity, they would have to be extremely careful to minimize overlapping areas so that the low opacity would be preserved. Opacity should only be affected when the artist consciously places more pressure onto the pen. On the other hand, the alpha overlap is desirable between the new stroke and the rest of the image being drawn.

To get both desired effects, the strokeLayer was created. By having the stroke on a separate visible layer, the strokeLayer composite could be changed without changing how the stroke appears on top of the baseLayer when the image is repainted.

I first attempted to make use of the Porter and Duff compositing rules supported by the Java AlphaComposite class. The SRC compositing mode matched best what I wanted to achieve. Instead of blending the SRC alpha over the DST_IN alphas, it completely replaced the DST_IN with the SRC. Through this, artists would have complete control of opacity through pen pressure. The achieved result is shown in the figure below.

Figure 9: A single stroke on the strokeLayer with SRC composite, drawn with decreasing pen pressure applied, first from left to right and then back across from right to left to show intersection points.

The new portions of the stroke are drawn completely over, even if the SRC has lower alpha than the DST_IN. This is not desirable either. I want the SRC to replace the DST_IN only if it has a higher alpha value. Otherwise, I want the DST_IN to remain the same. In order to

achieve this effect, I had to create a custom composite called MaxAlphaComposite. MaxAlphaComposite is identical to the SRC Composite in all ways except for how the alpha value is blended. Instead of automatically taking the alpha of the SRC, it compares alpha of the SRC to the alpha of the DST_IN and takes the larger one. MaxAlphaComposite achieved the desired effect, as shown in the figure below.

```
DST_OUT.alpha = max(SRC.alpha, DST_IN.alpha);
```

Figure 10: A similar stroke to that of Figure 9, drawn on the strokeLayer with MaxAlphaCompsite

At the end of the stroke, when the pen or mouse is released, the strokeLayer is drawn onto the baseLayer. Because the baseLayer has composite SRC_OVER, this preserves the overlap quality between the new stroke and the rest of the image. After the two BufferedImages are merged, the strokeLayer is cleared. The brushStroke Operation is then appended to the History array and written to the ObjectOutputStream if networking is on.

3.8 EraseStroke

The EraseStroke Operation is much simpler than the BrushStroke, in that it does not require use of the strokeLayer or any custom composites. The EraseStroke also uses Bresenham's algorithm to determine where to draw the brush shape. Though it has no color, it still contains shape as well as size, opacity, and tilt.

When the user chooses the erase tool, the composite of the baseLayer is changed from SRC_OVER to DST_IN. When pixels in the source and destination overlap in "Destination-In" compositing mode, the alpha form the source is applied to the destination pixels in the overlapping area. Because this is a little bit backwards from what EraseAlpha, the opacity value is flipped before being applied so that low pressure from the pen would generate a higher alpha and thus only erase a little while high pressure from the pen would generate a lower alpha and erase more. Once another tool is chosen by the user, the baseLayer composite will be changed back to the default SRC_OVER or to whatever composite is appropriate for the next Operation.

A slight problem exists with this implementation of EraseStroke. Since the erase is happening directly to the baseLayer, and since many times the parts of the EraseStroke being drawn overlap, opacity is hard to control. Usually, even at low pen pressure, because of the amount of overlaps, the EraseStroke appears to erase to a much lower opacity then expected. An easy fix has been implemented by simply reducing the effect pen pressure has on the alpha value. A formal fix—AlphaEraseStroke—was attempted, but still has some bugs. As of writing this thesis, only EraseStroke is implemented.

3.9 AlphaEraseStroke

AlphaEraseStroke was an attempt to use the strokeLayer to draw out eraseStrokes rather than directly erasing onto the baseLayer, similar to the way brushStroke works. In order to do this, lengthy steps had to be taken. AlphaEraseStroke lets you choose opacity and then remains the same opacity throughout the stroke, without varying by pen pressure. Size, however, still changes with pen pressure.

Within startOperation method, which is only called when the pen first hits the surface of the tablet, several preparatory steps have to taken. First, a copy of the baseLayer is created and

the opacity of the entire copy is reduced by the opacity value of the entire stroke. The purpose of this copy with reduced alpha value is to serve as the 'preview' image as the AlphaEraseStroke is drawing out. While it is possible to show the stroke all at once after it is finished, there is no direct way to show the stroke as the pen is dragged across the canvas because the information would have to come from the baseLayer and draw upon the strokeLayer. In an attempt to work around this, a custom composite called EraseAlphaComposite was made to read in an image as a reference raster. The baseLayerCopy is set as the reference raster, and instead of outputting a blending of SRC and DST_IN, it is a combination of SRC and the reference raster. It should simply output the pixels of the reference raster at the location of the SRC drawn.

After the AlphaEraseComposite is set, the draw method operates on the strokeLayer in a similar fashion to eraseStroke and brushStroke. At the end of the stroke, the strokeLayer is merged onto the baseLayer. In order for the merge to work properly, a custom composite once again was created. The custom composite named EraseMergeComposite is assigned to the baseLayer just for the merge operation. It does not change anything about the DST_IN except for the alpha value. If the alpha of the source is 0, the alpha is unchanged. Otherwise, the alpha of the source replaces the alpha of the destination.

```
DST_OUT.alpha = SRC.alpha==0 ? DST_IN.alpha : SRC.alpha;
```

This was necessary because the DST_IN composite that was previously used for eraseStroke would have otherwise cleared the entire campus other than the stroke. When strokeLayer is being drawn into baseLayer, most of the area will in fact have an alpha of 0, and with the rules of DST_IN composite, that alpha of 0 will be transferred onto the baseLayer. EraseMergeComposite is simply a slightly modified version of DST_IN, where the alpha of 0 preserves the image rather than clears it.

3.10 History & Undo

The History class contains an ArrayList of type Operation, which keeps track of all changes made to the canvas. In order to undo, the following steps are taken:

```
historyList.removeLastStep();
clearCanvas();
for (historyList) {
      temp = historyList.getStep(i);
      temp.redo(strokeLayer, baseLayer);
      mergeStrokeAndBase();
      clearStrokeLayer();
}
repaint();
```

The undo function essentially removes the last Operation, clears the canvas, and redraws everything by going through all the Operations in the history list. This can become slow if many steps have been taking. A typical digital painting can contain thousands of strokes. In order to speed up the process, every so often an Operation in the history list will be a keyframe, which will have a saved copy of the baseLayer(s) so that the redraws can start from that point rather than the very beginning. The user would be able to set the frequency of keyframes so that they could better customize the program to fit the needs of their computer.

3.11 Networking

The program currently requires one computer to run the server separately. The IP address of the computer must then be entered into any clients that wish to connect to the server. The server controls one socket and blocks for incoming input. The server keeps a synchronized list of all clients. Clients connected to the server write Operations to the objectOutputStream, which the server then sends to all clients other than itself to avoid redrawing over itself. When clients receive Operations from the server, they call the redraw method in the Operation and call repaint.

A separate History needs to be maintained to keep track of incoming Operations versus Operations that the user has run from their client canvas. Two different layers type should also be created—one where both users can edit upon one layer, and another that is only editable by the owner of the layer (but visible on other clients).

In the future it may be possible to make the server run continuously on web and let people connect more easily through that. Currently, only computers within the same network can connect because of firewall and router issues.

4. Further Steps

SimplePaint is far from finished. There are a multitude of features that I believe the ideal digital illustration Paintchat program should include, such as additional customizability, layer types, resolution-independent zoom, fill bucket, select lasso, eyedropper, and many more. After the survey that I conducted in part one of this thesis, I have a well-formulated idea on the kinds of things artists are looking for. As a digital artist myself, I am extremely interested in continuing this work. Eventually, it may become a real possibility to end up with a product that could eventually be distributed and used by artists everywhere across the world. That would be the ultimate long-term goal of this project.

5. References

[1] Sotiris P. Christodoulou, Georgios D. Styliaras: Digital art 2.0: art meets web 2.0 trend. DIMEA 2008: 158-165

[2] Porter, Thomas, and Tom Duff. "Compositing Digital Images." *ACM SIGGRAPH Computer Graphics* 18.3 (1984): 253-59. Print.

[3] Du, Weiming, Zhongyang Li, and Qian Gao. "Analysis of the Interaction between Digital Art and Traditional Art." *2010 International Conference on Networking and Digital Society* (2010): 441-43. Print

[4] Bresenham, J. E. (1 January 1965). "Algorithm for computer control of a digital plotter". *IBM Systems Journal* **4** (1): 25–30. doi:10.1147/sj.41.0025