Using The HTML5 Canvas Element For A Web-Based Multi-User Painting Application

by

Michael J. Snyder with W. Randolf Franklin

April 8, 2011

Rensselaer Polytechnic Institute

Troy, New York, 12180, USA
## Contents

1 Introduction ................................. 8

2 Background & Previous Work ................. 9

  2.1 Photoshop Analysis .......................... 9

    2.1.1 Brush Shape ............................... 9

    2.1.2 Brush Color ............................... 9

    2.1.3 Brush Size ............................... 9

    2.1.4 Brush Rotation ............................ 10

    2.1.5 Brush Roundness ........................... 10

    2.1.6 Brush Flip ............................... 10

    2.1.7 Brush Hardness ............................ 10

    2.1.8 Brush Spacing ............................. 10

    2.1.9 Brush Dynamics ............................ 10

    2.1.10 Brush Smoothing .......................... 10

    2.1.11 Brush Opacity ............................ 11

    2.1.12 Brush Flow .............................. 11

  2.2 HTML5 Canvas Element ...................... 11

    2.2.1 2D Context ............................... 12

    2.2.2 3D Context / WebGL ....................... 12

  2.3 jQuery ................................. 12

  2.4 Current Browser-Based Painting Applications ...... 13

    2.4.1 Sketchpad ............................... 13

    2.4.2 Sketcher ............................... 15

    2.4.3 deviantART muro .......................... 16

    2.4.4 Summary of Current Applications .......... 18

  2.5 Project Motivation .......................... 18

3 Development & Test Platform .................. 18

4 System Overview .............................. 19

  4.1 PHP ........................................ 19

    4.1.1 User Initialization ....................... 19

    4.1.2 User Starting State ....................... 20

    4.1.3 Update Polling (Server-Side) ............... 20

    4.1.4 Update Submission (Server-Side) ............. 21

  4.2 MySQL ...................................... 21

    4.2.1 Users Table ............................. 21

    4.2.2 Updates Table ............................ 21

  4.3 JavaScript ................................ 22

    4.3.1 Whiteboard Object ....................... 22

    4.3.2 Update Packet Objects .................... 23
4.3.3 User Objects
4.3.4 User Input Handling
4.3.5 Layer Objects
4.3.6 Tool Objects
4.4 jQuery
  4.4.1 Animations
  4.4.2 User Interface Elements
  4.4.3 Client-Server Communication
4.5 WebGL
4.6 HTML
4.7 CSS

5 User Interface
  5.1 User Interface Structure
  5.2 Menu Button Objects
    5.2.1 Button State & Behavior
    5.2.2 Settings Panel
  5.3 Layout Motivation

6 Toast Notification Tool
  6.1 Visual Appearance
  6.2 Whiteboard Integration
  6.3 Access & Usability

7 Brush Tool
  7.1 Initial Setup
  7.2 Settings Panel
    7.2.1 Tips
    7.2.2 Shape Parameters
    7.2.3 Application Parameters
    7.2.4 Shape & Stroke Preview
  7.3 Contour Outline
  7.4 Shape Selection
  7.5 Color
  7.6 Resizing
  7.7 Rotation
  7.8 Axis-Aligned Flipping
  7.9 Roundness
  7.10 Spacing
  7.11 Flow
  7.12 Opacity
    7.12.1 2D Context Mode
    7.12.2 WebGL Mode
7.13 Mouse Events .......................................................... 36
7.14 Painting Process ...................................................... 36
   7.14.1 Creating Strokes .............................................. 36
   7.14.2 Mouse Smoothing ............................................ 37
   7.14.3 2D Context Mode ............................................ 38
   7.14.4 WebGL Mode ................................................ 38
   7.14.5 Color Precision .............................................. 39
7.15 Submitting Updates .................................................. 39
7.16 Processing Updates .................................................. 40

8 Chat Tool .................................................................. 40
   8.1 Settings Panel ..................................................... 40
   8.2 Scrollbar Management .......................................... 40
   8.3 Username Setting ............................................... 40
   8.4 Sanitizing Messages ............................................ 41
   8.5 Toast Notifications ............................................. 41
   8.6 Submitting Updates .............................................. 41
   8.7 Processing Updates .............................................. 41

9 Color Selection Tool .................................................. 41
   9.1 Settings Panel ..................................................... 41
   9.2 Selecting Colors ................................................ 42
   9.3 Color Chips ........................................................ 42

10 System Information Tool .............................................. 42
   10.1 Settings Panel ................................................... 42
   10.2 Users List ........................................................ 43
   10.3 Initial Update ID ............................................... 43
   10.4 Latency ............................................................ 43
   10.5 Rendering Mode ............................................... 43

11 Future Work ............................................................ 43
   11.1 Full State Support ............................................. 43
   11.2 Infinite Canvas ................................................ 44
   11.3 Additional Brush Settings .................................... 44
      11.3.1 Hardness .................................................. 44
      11.3.2 Application Mode ........................................ 44
   11.4 Additional Tools ................................................ 44
      11.4.1 Selection Tool ............................................ 45
      11.4.2 Text Tool ................................................ 45
      11.4.3 Eraser Tool ............................................... 45
      11.4.4 Move Tool ................................................ 45
   11.5 Full Layer Support .............................................. 45
11.6 Canvas Zooming ......................................................... 45
11.7 Long Polling ............................................................. 45
11.8 Graphics Tablet Support ............................................. 46
11.9 User Interface Refinements ......................................... 46
  11.9.1 Drag & Drop Interface .......................................... 46
  11.9.2 Slider Refinements ............................................... 46
  11.9.3 Shortcut Keys ...................................................... 46

12 Results & Conclusions ................................................. 46

References ................................................................. 48
Abstract

This project aims to implement a multi-user whiteboard application that runs completely within a browser without any plug-in dependencies. Using WebGL for the rendering of paint on the canvas and traditional HTML DOM objects for the user interface, an efficient and responsive application is offered. Actions performed by a user are recorded in real-time and asynchronously sent to the web server where they are archived in a MySQL database. This database is periodically and automatically queried by all connected users via a PHP interface and new updates are retrieved to present the most up-to-date state of the whiteboard. In addition, care was taken in the design of the application to ensure extensibility and modularization. Performance, design decisions, related works, and possible extensions are all commented on throughout the discussion.

Index Terms

multi-user painting, browser application, HTML5, WebGL, canvas, JavaScript, PHP, MySQL
Acknowledgements

Thank you to all of the professors at Rensselaer Polytechnic Institute that I have had the pleasure of taking a course with over the past four years. In addition, thanks to my advisor, Dr. W. Randolf Franklin, for his help and guidance throughout this process.

Thank you to my parents, friends, and family for their support.
1 Introduction

Throughout the past decade, both computers and technology in general have advanced significantly. With today's mainstream desktops, laptops, smartphones, gaming platforms, and other internet-enabled devices, it is easier than ever before for individuals to connect and stay connected with one another. In order for these individuals to stay connected such devices typically possess some sort of web browser. Technology-centric companies have realized this and have recently begun focusing their efforts on capitalizing on their virtual customers’ desires.

In recent years, companies like Google and Facebook have released browser-based versions of software that traditionally could only be run from a user’s desktop. Google Documents [6] and Facebook Chat [8], word processing and instant messaging programs, respectively, are just two examples of this. Increasingly, internet users are living inside their browsers instead of running a multitude of software applications separately on their desktop. This, consequently, highlights two key offerings of the internet:

1. Location-independent accessibility
2. Interaction and collaboration

First, because the internet is a global network, information can be stored and retrieved regardless of the connection point. In turn, this means that both strangers and well-acquainted individuals can potentially communicate with each other no matter where they are. This ability to quickly and easily send information back and forth fosters a second offering of the internet—interaction and collaboration. Being that individuals across the world are essentially connected with one another at all hours of the day, it is logical to use the global network to work together. Using the internet, individuals today can collaboratively engage in activities ranging in nature from academic to pure entertainment, all without leaving their homes.

It makes sense, then, why software companies are increasingly favoring the browser as their platform of choice. By offering an application that runs inside a web browser, a company is effectively distributing its work to everyone using the internet. Today’s browsers are relatively well-standardized, so compatibility is typically not an issue. Finally, browsers are free, so there is no notion of a platform entrance fee that must be paid for a mainstream consumer operating system. All things considered, marketing browser-based software is rather low-risk. However, this software category is not mature and many classes of applications have not yet made the transition—one of the largest such classes being that of artistic programs.

Multimedia companies Adobe and Autodesk, for example, obtain their revenue primarily by selling software tools to digital artists. Their software applications have elegant user interfaces, perform very well, and are used throughout the professional community. What the software doesn’t have, though, is portability. If an artist has licensed access to a multimedia application from either company, they must use it at the location where it was installed. Furthermore, despite the multimedia industry having a very collaborative environment, it is not possible for two artists to work on the same digital piece using typical commercial software. Thankfully, these two glaring issues disappear completely when working with browser-based software.

With the recent promotion of HTML5 and WebGL, what can be done in a browser has now become virtually limitless. All of the things that

\footnote{Note that although Linux is freely-available in many variations, software support is somewhat limited due to its market share. As a result, most companies develop consumer software for either Windows or OSX, both of which must be purchased.}

\footnote{WebGL is OpenGL ES 2.0 that interfaces with JavaScript to run in a browser.}
users enjoy about traditional desktop multimedia software but were previously not possible within a browser, such as fully fleshed-out interfaces and speed, are perfectly attainable on the web today. Consequently, it is the perfect time to take a look at multimedia software and how it could be implemented and used in a browser.

This project aims to implement a multi-user whiteboard application that runs completely within a browser without any plug-in dependencies\(^3\). Using WebGL for the rendering of paint on the canvas and traditional HTML DOM objects for the user interface, an efficient and responsive application is offered. Actions performed by a user are recorded in real-time and asynchronously sent to the web server where they are archived in a MySQL database. This database is periodically and automatically queried by all connected users via a PHP interface and new updates are retrieved to present the most up-to-date state of the whiteboard. In addition, care was taken in the design of the application to ensure extensibility and modularization. Performance, design decisions, related works, and possible extensions are all commented on throughout the following discussion.

2 Background & Previous Work

Despite the fact that browser-based software has not yet matured, there has been a great deal of work, typically hobbyists’ experiments, in the area. Many of these works are of high quality but lack certain features that would be necessary to emulate a painting program like Adobe Photoshop in a browser. In order to better understand these deficiencies, an analysis of important Photoshop features is provided here. In addition, current web technologies and what has been done with them are observed.

2.1 Photoshop Analysis

Adobe Photoshop is the industry standard for artistic image processing. Offering an extremely wide variety of tools, filters, and features, it is the perfect tool to generate and manipulate digital art pieces. At the core of Photoshop’s functionality is a brush system with which users can paint on a given canvas. This brush system references a multitude of user-adjustable parameters in order to provide artists with complete control over the pixel data in their piece.

2.1.1 Brush Shape

Arguably the most important setting is that of brush shape. Photoshop allows users to select from a base set of brushes ranging from simple hard and soft round tips to more exotic shapes like grass blades and stars. In addition, users can define their own brush tips by simply referencing an image area, allowing for unlimited possibilities.

2.1.2 Brush Color

The color of the brush can be set to any RGB\(^4\) value. The alpha-transparency (alpha) values of each pixel in the source tip remain unchanged.

2.1.3 Brush Size

The size of the brush may be specified in pixels ranging from one to 2,500. If the brush tip is sampled\(^5\), users are not restricted from upscaling or downscaling the reference shape. The size of the

\(^{3}\)A plug-in is considered to be Flash, Java, Silverlight, or any other software a user must install in addition to their browser.

\(^{4}\)RGB is an acronym for Red Green Blue, and denotes the amount of each primary additive color that is present in the final color.

\(^{5}\)A sampled brush tip is one where the user has selected a region of an image and defined it as a brush shape. This process records the pixel data at the time of sampling. This is in contrast to a dynamic brush such as a soft round tip that is generated without quality loss at the desired size.
brush is defined as the maximum of the width and height, so a brush with a size of 150 pixels could be either 150 pixels tall or wide, depending on its aspect ratio.

2.1.4 Brush Rotation

Irrespective of the selected tip shape, users may rotate their brush around the center of the bounding box that encloses the tip.

2.1.5 Brush Roundness

The percent roundness of a brush defines how much the height is scaled prior to rotation. For example, if the roundness parameter is specified as being 50%, the height of the brush will be one-half its original height. A value of 100% denotes no change in height. Figure 1 depicts this parameter.

Figure 1: A soft round brush at 100% roundness, left, is compared to the same brush at 25% roundness, right.

2.1.6 Brush Flip

Users may choose to flip their brush along the X- and/or Y-axis. This flipping simply reflects the selected brush shape. The flip takes place prior to rotation and roundness adjustments.

2.1.7 Brush Hardness

The hardness of a brush is restricted to non-sampled default round brushes. The parameter controls how crisp the boundary of the circle-shaped tip is. A hardness of 0% produces a very smooth, soft circle while a value of 100% offers no softening. Figure 2 depicts this parameter.

Figure 2: A soft round brush at 100% hardness, left, is compared to the same brush at 25% hardness, right.

2.1.8 Brush Spacing

A brush’s spacing determines how far the mouse must move between consecutive applications of the tip to the canvas. The value, specified in percent, is multiplied by the brush size to determine how many pixels must be traversed before a tip application occurs.

2.1.9 Brush Dynamics

A variety of parameters can be specified such that the brush will dynamically update while being used. For instance, a brush can have its size dynamically adjusted based on stroke length so that a long line will taper down to a point. These settings allow artists to incorporate a great deal of organicness into their pieces.

2.1.10 Brush Smoothing

By enabling smoothing, users tell Photoshop to interpolate between mouse readings to provide a less faceted path for their lines. This is an extremely important feature because the operating system cannot offer unrestricted, priority access to the application in order to read every mouse polling sample. As a result, there are occasionally gaps within the screen coordinates reported by
the mouse cursor when the user is painting. By interpolating between the points using a spline, the amount of apparent faceting is significantly reduced.

2.1.11 Brush Opacity

The opacity of the brush is a somewhat complicated and convenient parameter. When set to a value less than 100%, it will scale the alpha component of each pixel in the brush tip by the denoted parameter. In addition, when the user paints on the canvas, the amount of color that can be applied will not exceed the specified opacity value for the duration of one stroke. When the user releases the mouse button to finish a stroke, the opacity limit is lifted and the next stroke may apply color up to the limit once again. Figure 3 shows an example of this parameter.

![Figure 3: A black, soft round brush at 35% opacity, left, is scribbled on the canvas, right. Note how the amount of black applied to the canvas is limited by the opacity value. Even when repeatedly passing over the same area, the white canvas does not change by more than the 35% opaque brush.](image)

2.1.12 Brush Flow

The flow of a brush is very similar to the opacity parameter. When set to a value less than 100%, it scales the alpha component of each pixel in the brush tip the same way the opacity value does. However, the flow parameter does not restrict the amount of color that can be applied to the canvas in a single stroke. This means that if the user repeatedly passes over the same location, additional color will build up. In contrast, the opacity parameter would not allow additional color to be transferred to the canvas. Interestingly, the flow parameter respects the opacity value. Thus, if, for example, a user set the brush to 80% opacity and 30% flow, color would build up until it reached 80% of the source tip alpha value. Figure 4 shows an example of the brush flow setting.

![Figure 4: A black, soft round brush at 35% flow and 100% opacity, left, is scribbled on the canvas, right. Note how the opacity value exceeds the 35% flow limit to reach 100% opacity in regions where the stroke overlaps itself.](image)

In addition to the parameters discussed here, Photoshop offers many other values that can be tweaked by the artist when painting. However, the ones previously analyzed make up the core functionality of the brush tool. It stands to reason, then, that any application attempting to offer a reasonable browser-based alternative to Photoshop must implement a significant subset of these features.

2.2 HTML5 Canvas Element

In order to offer any sort of image manipulation program via a browser, an interface to a mutable array of pixels is necessary. Thankfully, with the recent introduction of HTML5 and the canvas element, such an interface exists.
2.2.1 2D Context

By requesting a 2D context from the instantiated object, the canvas enters a 2-dimensional mode useful for flat image processing techniques. The 2D context allows for a wide variety of transformations to take place very similar to an OpenGL application. Programmers may translate, rotate, draw points and lines, mask regions from changes, and perform many other tasks using the 2D context [14]. This flexible functionality makes the canvas element ideal for a multimedia program, and more specifically an image editing program, that runs in a browser.

Within the last year, browser developers realized the significance of the canvas element and implemented Graphics Processing Unit (GPU) acceleration for 2D contexts [3]. Essentially, this means that most intensive image processing techniques that would typically disrupt a user's experience by running on the CPU can be offloaded to the GPU for a massive performance boost. This is an excellent possibility as it again makes the goal of developing a browser-based painting application very attainable.

2.2.2 3D Context / WebGL

In addition to the 2D context offered by the canvas element, developers can also request a 3D context in the form of a WebGL handle. WebGL is a slight variant of OpenGL ES 2.0, which is well-known for running on many portable devices like smartphones [7]. When using a WebGL-based canvas, developers can re-create many of the graphics that users expect in fully-featured software. Since WebGL by design runs on a GPU, performance is excellent. With respect to an image editing program, however, the most significant benefit of WebGL is the ability to use vertex and fragment shaders.

Vertex and fragment shaders contain small procedures that run once for every vertex and fragment (potential pixel), respectively. The performance implications of these shaders is huge when the possibilities are considered. Using a vertex shader, a texture can be scaled and rotated without any additional computational overhead. Once data is passed to the fragment shader, pixels can be set in the final image based on complex rules, such as only changing values that have not yet been changed. This functionality allows developers to offer rather extravagant effects with little to no performance hit provided the user has a capable GPU.

In addition to running shader programs, WebGL allows one to instantiate and manipulate a wide variety of framebuffers. These buffers can be created at any resolution and can be thought of as canvas layers that are not visible to the user unless specifically presented. Despite being hidden from the user, however, developers have full access to the pixel information stored in their buffers. This means that applications can be written such that multiple passes may be made over pixel information before being presented to the user. As a result, the sheer speed and flexibility of WebGL paired with the straightforward approach to 2D rendering makes the canvas element the perfect means by which to develop a browser-based painting application.

2.3 jQuery

While the canvas interface is an exceptional addition to the traditional web browser, there are some areas of application development where it does not, and sometimes should not, need to be used. One such area where developers can get by without using canvas objects is that of user interface design. Since the browser must necessarily have the ability to monitor mouse and keyboard input, it is very straightforward to implement a graphical user interface (GUI). Browsers already understand that some procedure must occur when a user clicks on a link, so
by re-mapping that functionality it is possible to produce events like animations following user input. In order to help with this, many JavaScript libraries have been developed—one of which is the widely-used jQuery.

The jQuery JavaScript library allows programs to quickly and efficiently traverse the tree of document objects and update display parameters at will [10]. This essentially means that objects on the screen can have their appearance changed based on logic provided by the developer. Element movements, color changes, and image updates as well as client-server communication can all take place when jQuery is used in a web application. As a result, it is ideal for use in front-end web application development and eliminates the need to implement complex user input recognition routines for the canvas element.

2.4 Current Browser-Based Painting Applications

As with most new technologies, since HTML5's mainstream adoption many developers have begun experimenting to push the limits of the canvas application programming interface (API). Many interesting projects have resulted from these experiments, with some of them being drawing and painting applications. In the discussion that follows, a number of these types of applications are reviewed to determine their strengths and weaknesses.

2.4.1 Sketchpad

In 2010, Michael Deal and Charles Pritchard of Orange Honey released Sketchpad, a browser-based drawing application based on the 2D HTML5 canvas element [4]. Of the applications that utilize the canvas technology, this is one of the most robust implementations of a painting program that runs as a browser application. With an intuitive GUI and excellent performance, Sketchpad is a successful paint program. That being said, it does lack some features and could be improved in some areas. Although Sketchpad offers many different tools ranging from cropping to text placement, the analysis here focuses on its brush capabilities. For reference, Figure 5 shows the default Sketchpad interface.

When a user first loads the application, the tool widget windows partially overlap the drawing canvas. This is a small but nice touch as it introduces the user to the idea of moving their workspace around, given that each of the windows within the application can be dragged and dropped. The default, pre-selected tool is the brush, which makes sense because it will be the tool most used within the application.

Upon drawing on the canvas with the brush, it becomes apparent that many parameters can be set for the tool. Users may indicate brush size, hardness, flow, and opacity for the default round brush tip. If the stamp tool, which behaves nearly identically to the brush, is activated, various tip shapes may be selected beyond the traditional round offering. These abnormal tip shapes restrict the brush parameters to size, flow, and opacity by leaving out the option to specify hardness. Interestingly, when using the brush tool users can paint with not only colors but also gradients and textures. This flexibility is great and really helps to demonstrate the power of the canvas element.

Despite offering many brush parameters to its users, Sketchpad could be refined in multiple ways:

Firstly, the opacity parameter of the brush does not properly limit the application of color to the canvas in the same way that Photoshop does. Instead, the opacity setting replicates the traditional flow parameter and allows color application to reach 100%.

Next, the flow parameter does not alter the opacity of the brush before applying it to the canvas. In fact, the flow value doesn’t appear to
Figure 5: Sketchpad, a browser-based painting application using the HTML5 2D canvas element.

have any impact on the actual amount of color leaving the brush. This is a significant oversight as the flow setting allows artists to create very gradual, fine details. Without the option, users cannot properly limit their color application and thus have less freedom than traditional desktop software.

The ability to specify brush spacing is absent from Sketchpad. This is another critical feature that artists require in painting programs. Without any notion of brush spacing, the selected tip is applied to the canvas as many times as possible over the course of a given stroke. Unfortunately, this is not always what a user wants—especially if a true flow setting is available. In conjunction with the flow parameter, brush spacing allows users to decrease the intensity of their strokes, which offers another degree of freedom within the application.

Brush rotation is not possible within the application. This severely limits the user to the exact orientation of the brush tip shape.

The roundness of the brush cannot be set within Sketchpad. As with the other missing features, this reduces the freedom provided to artists as they develop a piece. Not being able to anisotropically resize the selected brush limits the amount of detail that can be produced.

When a user draws on the canvas, their mouse input is not smoothed. Unlike in a traditional desktop application, this is extremely important in a browser-based program. Because all the input that the web application receives is via JavaScript mouse events, it is impossible to track every single user action. As a result of the absence of mouse smoothing from Sketchpad, quickly-drawn loops have a tendency to appear faceted.
When drawing with the various brush tips and sizes, the user’s cursor does not update to reflect their current settings. In traditional desktop painting programs, the brush tip is typically outlined and centered over the operating system’s cursor. This provides the user with precision as they paint into the canvas because they know exactly where their changes will occur.

Finally, the selected brush tip cannot be flipped over either the X- or Y-axis. This is a somewhat less-important feature, but is still welcomed functionality.

Thus, it can be seen that while Sketchpad is a rather functional painting application, it is incomplete. It is important to note, however, that Sketchpad was one of the first painting applications to use the HTML5 canvas element. Consequently, it should be regarded as an excellent first step towards a fully-featured Photoshop-like browser application.

2.4.2 Sketcher

Prior to the development of the canvas element, in 2006 Thor Johansen developed a Java applet named Sketcher [9]. This applet is arguably the best implementation of a multi-user drawing application that can run in a browser. Despite the fact that it does not utilize either the 2D or WebGL canvas contexts, it is worth investigating to understand why it is so successful.

Sketcher is particularly aimed at satisfying the needs of artists who want to take a piece from a rough sketch through to a final digital painting. It excels by offering multiple layer support, color selection, brush parameters rivaling that of Photoshop, and a somewhat clean interface, as seen in Figure 6. Notably, the applet does not allow the user to directly set the opacity or flow of their brush. This is partially mitigated by the fact that users may easily draw on different layers to simulate the functionality of adjusting brush opacity. Furthermore, because the application interfaces with graphics tablets to offer pressure sensitivity, artists are given an excellent amount of freedom.

It is important to take notice of Sketcher’s support for graphics tablets. Professional artists rely on tablets to give them superb control over their input to multimedia programs. When drawing on the tablet surface, the accompanying tablet drivers interpret how hard the user is pressing down, what angle the stylus is at, and how much it has rotated. This sort of information is very useful for a painting program as it allows the application to determine how much color should be applied to the canvas for any given stroke. With a traditional mouse and keyboard interface, users are limited to pre-defined stroke appearances. Once the additional stylus parameters are incorporated, however, the application can offer features like modulating stroke intensities and on-the-fly rotation. This is very important in offering an overall more open environment in which artists can express themselves.

Similarly, it is important to note the other major contribution that Sketcher makes to the library of online painting applications—the fact that it is multi-user. Since the application runs inside a Java environment, it naturally has the ability to perform relatively straight-forward network communication in a client-server environment. Johansen embraced this possibility and ensured that everything a user does locally is propagated to all other users currently painting on the canvas. Furthermore, Sketcher retains user actions in a server-side state so that newly-connected users are able to view the canvas in the same state that every other user does, regardless of differences in connection time. This is a somewhat subtle but very important feature—since all users have the same representation of the canvas, there are no issues with users seeing half of a painting or accidentally drawing over something they cannot see.
Figure 6: Sketcher, a Java-based painting application that supports multi-user interactivity.

All things considered, Sketcher is an immensely capable drawing and painting browser-based application. Its features and performance truly set it apart from other web applications. Unfortunately, since Sketcher was developed prior to the introduction of mainstream HTML5 and the canvas element, it requires the user to have Java installed for their browser. As a result, it would be beneficial to offer an application similar to Sketcher that does not require a third-party plug-in so that any user with a modern browser can access a robust painting program.

2.4.3 deviantART muro

In August of 2010, the developers at the art community site deviantART released an HTML5-based painting application called muro [5]. Following its release it gained a significant following by deviantART’s user base due to its capabilities. Since the community that muro services ranges from highly-skilled professionals to entry-level artists, the application was designed to be very streamlined. This design decision resulted in both positive and negative qualities that affect the application’s overall usefulness.

As Figure 7 shows, muro presents the user with a beautiful, minimalistic interface. Everything that is available to the user is easy to find and the clean lines and subtle gradients offer a pleasing aesthetic. Unlike Sketchpad, muro is geared directly towards artists so it consequently contains more advanced features. These features are implemented such that the user interface provides direct access to them, allowing artists to spend more time painting and less time moving through menus.

When painting on the canvas with the traditional brush tool, users can set their color, opacity, size, and softness. If the artist is using a graphics tablet, additional brush features are enabled to support pressure sensitivity. This support is dependent upon the particular graphics tablet being used, but those offered by Wacom natively offer their stylus data to browser
Figure 7: deviantART muro, a browser-based painting application using the HTML5 2D canvas element.

Applications via a small plug-in [13]. In addition, areas may be selected to restrict painting regions and fills may be applied to the canvas. The application also supports various filters such as blurring and edge extraction.

Importantly, muro provides mouse smoothing for the user’s paint strokes. When a stroke is applied to the canvas, the real-time feedback to the user appears rough and faceted if the action was performed quickly. After the user releases the mouse to finish the stroke, the application interpolates between the readings provided to it to smooth out the resulting line. This is a very nice feature and greatly improves the resulting artwork, albeit at the price of a very minor annoyance with the quality shifting. It would be excellent to apply this technique as the user is drawing to remove the pop-in effect that strokes have after being applied to the canvas.

Although the application offers a variety of brushes, most of them are abstract in nature and there is little ability to customize them. Features such as brush roundness, flow, rotation, and flipping are missing—all of which would greatly diversify the options provided to the artist. The program does, however, present the user with an outline of their round brush over the cursor that updates based on their selected size, which is a very nice touch.

Since muro was developed to meet the needs of a wide range of artistic skill sets, there was a significant emphasis placed on being able to start using the application within seconds of it loading. This resulted in a minimal subset of brush settings that ultimately reduces what artists are capable of within the application.
2.4.4 Summary of Current Applications

In addition to the applications discussed here, there are many other paint programs that have been developed. The selection here simply highlights a subset exhibiting applications that are successful to varying degrees in different categories.

Overall, the current state of browser-based painting applications leaves a lot to be desired. In general, users are limited to what they can do because the current applications don’t offer complete control over their brush strokes. This is somewhat puzzling because the technology available to developers through the canvas element and HTML5 is very robust and performs quite well. At the same time, HTML5 is rather new and the web community is in somewhat of a state of learning and experimentation, so it may not be entirely fair to expect desktop-quality software from the examples discussed here.

2.5 Project Motivation

In reviewing the aforementioned browser-based painting applications, it is clear that many features have been implemented well while others have been glanced over. In particular, there seems to be a significant lack of brush customization beyond simply selecting different tip shapes. This is unfortunate as it does restrict artists quite a bit and makes it more difficult for them to develop their pieces digitally.

Another area that could be refined is that of interface design. With the exception of deviantART’s muro, drawing applications tend to have very disjointed interface styles. Users familiar with desktop multimedia software expect highly-polished GUI’s and using a web application that feels more like a website and less like a piece of software can be a major drawback.

Finally, there have yet to be any successful publicly-known browser-based painting applications that use the canvas element and offer multi-user support. Since Sketcher runs completely within a Java environment, it is not an exception to this. Very little work has been done to investigate the possibility of creating multi-user applications based on the new multimedia technologies available with HTML5.

3 Development & Test Platform

Web developers often have to consider what browser their content will be viewed in so that they can ensure it is presented properly. With the specificity of the canvas API, this concern is somewhat mitigated. Throughout the development of this project, two major web browsers were tested:

1. Mozilla Firefox 3.6 and 4.0
2. Google Chrome 10

These browsers were used primarily because they have consistently been at the forefront of high-performance, quality web browsing. Additionally, at the time of development Internet Explorer (version 8) did not support the canvas element unless a very low-performing mode of emulation was used, so the browser was not considered.

All of the development and testing took place on a Windows 7 desktop and a Windows 7 laptop, though there is nothing preventing the project from running on an Apple operating system. So long as the browser being used conforms to the canvas API, the results displayed should be identical.

Since the application relies heavily on hardware acceleration via the GPU, it should be noted that no significant performance difference was discernible between the desktop and laptop testing. The desktop machine was running Windows 7 Ultimate 64-bit with 4GB of RAM, an Intel Core 2 Quad Q6600 CPU, and a 512MB ATI Radeon HD 4870 GPU. The laptop machine was running
Windows 7 Ultimate 32-bit with 2GB of RAM, an Intel Core 2 Duo T7300 CPU, and a 128MB NVIDIA Quadra NVS 140M GPU.

The web server where the files were hosted for testing was located roughly 30 minutes away from the point of development. Although this offered rather fast response times when querying pages over the internet, it did not affect the quality of the results. Everyone using a broadband connection to the internet should receive the same quality experience within a few fractions of a second depending on their geographic location relative to the web server.

4 System Overview

In order to offer a robust painting application, many programming languages were used. In particular, PHP, MySQL, JavaScript, jQuery, WebGL, HTML, and CSS were relied upon to create the final result. In general, the communication flow follows a regular pattern of polling whereby the client requests any updates that have occurred since the last time the client contacted the server. In between the polling requests, the client may submit an update packet to the server for database archival so that other users can receive the data. The specific role of each language and how it implements key components of the system is explained in the following discussion.

4.1 PHP

PHP is a server-side scripting language and acted as the bridge between the JavaScript running on the local user’s machine and the server storing and distributing the data. Without PHP or an equivalent server-side language, the user accessing the application would be unable to get information from the server. Likewise, the server would be unable to retrieve information from its database. Consequently, PHP was used for many initialization routines and for getting information to the JavaScript application upon request.

4.1.1 User Initialization

When a user accesses the application via a URL, their browser sends a request to the server for the relevant files. After the server receives this request, a user initialization routine is performed. This routine accomplishes two tasks:

1. Providing the user with a globally unique identifier (GUID)
2. Recording the user in the database

The first task is important for a number of reasons including both organization and security.

First, giving each active user a GUID that does not get distributed to other users is essential to keeping track of who submitted an update to the server. Since this application was developed with modularization in mind, it is important to note that a common request by tool objects is to know where the data that it is currently processing came from. When each user has a GUID, this task becomes trivial.

Second, assigning a GUID to each user allows for a significant level of security within the application. One can imagine the possibility of a user communicating with the server using another user’s GUID such that malicious chat messages or drawings are submitted. If undesirable events happen to take place, it is important that the server be able to keep track of who performed the actions so that disciplinary measures and protective actions may be taken.

Note that the GUID assigned to the user upon connecting to the application is an MD5 hashing of the PHP function `uniqid()`\(^6\), the IP address of

\(^6\)The `uniqid()` function returns a random, unique alpha-numeric value based on the server’s current time in microseconds.
the client, and the client’s user agent information. This combination of information, when hashed, produces a virtually un-guessable GUID for the user.

4.1.2 User Starting State

After the user has been both provided with a GUID and initialized in the database, the server must tell the client where it has entered with respect to the current stream of updates. As updates are received by the server, they are archived in the database with an ID number. The server tells the client the ID of the most recent update in the database so that the first polling request will only gather updates since that relative starting point.

Of particular interest here is the fact that users may access the application URL with the parameter \(?start=\#\), with the number sign being the database update ID the user would like to start from. This allows users to rebuild the state of the application so that, for instance, they may synchronize with another currently active user. Furthermore, users can also use this functionality to pick up where they leave off if they lose their internet connection or switch computers.

4.1.3 Update Polling (Server-Side)

Since JavaScript and MySQL cannot communicate directly with one another, PHP must act as a middle layer between the two. When the client wants to know if any new updates have been received by the server, it queries a PHP file that then accesses the database. When the client queries the file, it provides two items of information. The first item is its GUID. The second item is the ID of the last update it received. After receiving this information, the server then responds by performing four tasks:

1. Updating the user’s heartbeat
2. Clearing users from the database that have not updated their heartbeat in the last 60 seconds
3. Returning the updates that have occurred since the provided database ID
4. Returning information about the connected users

After the polling process has been initialized, it occurs five seconds after every response from the server. Thus, assuming the server does not take longer than five seconds to respond to the client, the client will get updates at five second intervals.

The user’s heartbeat is a timestamp that is recorded by the server every time an active user checks in. This information is used so that tools, such as the system information panel, can tell the local user how many people they are interacting with. When the polling process occurs, it updates the requesting user’s heartbeat with the current time and checks the database to see if any users have not checked in within the last 60 seconds. If any recorded users meet this criteria, they are removed from the database on the basis that they are no longer connected to the application. It is important to note here that since there is no dedicated background server process managing the updates from the clients, the server does not push any updates. Instead, all updates to the state of the application result from client requests.

The information returned to the client following the polling request is as would be expected. If no updates have occurred, the response will contain only information about the connected users. If any updates have occurred since the provided database ID, the response will also contain the stored information for those actions.
4.1.4 Update Submission (Server-Side)

A universal action that all tools have the ability to do is perform an update. When the client submits an update to the server, it is encapsulated inside a JSON object. All updates sent to the server contain three items:

1. The tool that submitted the update
2. The GUID of the user that submitted the update
3. The update data

Once the update has been received by the server, the current time is determined and the update packet is stored in the database. While the tool name and the user GUID are built in requirements to the update packet structure, the data that is transmitted can be anything. This offers tools a high degree of flexibility and functionality.

4.2 MySQL

In order to maintain the current state of the application, all updates are archived in a MySQL database as they are received by the server. The database contains two tables—one for users and one for updates.

4.2.1 Users Table

The users table contains four fields:

1. Database ID number
2. GUID
3. Username
4. Heartbeat

First, a database ID is assigned to each user as a means of hiding their GUID during client-server communication. This is important because it is very simple to view the server’s response to a polling request using traditional browser debugging tools. If the user GUID’s were transmitted as a means of keeping track of where an update came from, impersonation would be straight-forward. Instead, the user GUID is only used for communication between the local user and the server as an authenticator for tool and heartbeat updates. All other information that requires a user to be identified uses the database-assigned ID number.

Second, the user GUID is stored in the table so that updates can be verified as they are received by the server.

Next, the username assigned to the active user is stored for access by the application’s tool components. For example, the chat tool allows users to change their username after connecting. This field stores said information so that further updates from that user can be personalized.

Lastly, the heartbeat is stored so that a verified list of active users can be maintained. Note that there will always be at least one user listed in the table regardless of whether there are any active users. This happens because the application only gets notified about a user’s disconnection when a client initiates communication. Thus, if a user connects to the application when no one else is connected, any stale users in the database will be cleared before a response is sent to that client.

4.2.2 Updates Table

The updates table contains five fields:

1. Database ID number
2. User GUID
3. Tool name
4. Timestamp
5. Data
The database ID number simplifies the client process of requesting updates. Since the updates are stored in the database in sequential order, a client only needs to remember the last update it received in order to obtain everything that has occurred since.

The user GUID is stored so that when a polling request comes in, all update data that is returned can be cross-referenced with the users table to hide each user’s GUID value. Every user connected to the application knows only their own GUID—other users are identified based on their database ID.

The tool name is stored so that the proper tool can handle the update packet when it receives it after a user’s polling request. The timestamp field is maintained as a convenience for tool polling under the assumption that it may be useful depending on what the module needs to know. In the current application, this field is not referenced.

Finally, the data field stores all the information originally submitted by the client. Since this is stored in the same format it was received in, parsing of update packets is very straight-forward on the client-side.

4.3 JavaScript

Non-library-based JavaScript makes up the majority of the implementation of this application. Since each of the tools offered by the application is implemented in JavaScript, this section focuses only on the functionality that is utilized by all tools and that which is contained within the primary whiteboard object.

4.3.1 Whiteboard Object

The whiteboard object is the first item to be initialized when the application loads. Upon initialization, it creates the necessary tools and information for the user to interact with the application. Specifically, the whiteboard object creates:

1. An activity layer
2. A local user
3. A remote user
4. The user interface elements
5. Event handlers for mouse input
6. Various convenience functions
7. The tools available to each user

The activity layer is a fully transparent canvas with dimensions equal to the user’s browser window resolution. It is used to collect all the user’s mouse input and sits above all other elements in the application, except for the menu. When initializing the mouse input event handlers, they are attached to the activity layer. If an event handler fires because the user interacted with the activity layer, it is propagated down to the currently active tool.

The local user is instantiated after the PHP user initialization routine has been run, which takes place prior to the JavaScript running. The user’s state, including their active and available tools, currently selected foreground and background colors, database ID, GUID, and username, is maintained within the object. Importantly, a feedback layer is stored within the user object as well, which is used by the brush tool. The feedback layer acts as a temporary canvas onto which the user paints before their changes are committed.

The remote user is instantiated without having to reference server information and maintains a similar state to that of the local user. As a reminder, recall that the only GUID that is known to the user of the application is their own. Consequently, the remote user object serves as a model for all externally-connected users and its state is modified on a per-update basis.
Next, the whiteboard object contains various convenience functions for use by the tools. Since every tool must have a reference to the whiteboard object, these functions are a simple way to aggregate frequently-used procedures. For instance, the whiteboard object contains functions for converting RGB colors to HEX\textsuperscript{9} values. Being that this is a multimedia application, it is likely that many tools will want access to such functionality. Placing it in the whiteboard object helps eliminate redundant code throughout the application.

Finally, the whiteboard object controls which tools are created for which users. This is an important point regarding efficiency, as not all tools submit updates to the server. For instance, the color selection tool does not need to notify the server that the user’s color has changed. Instead, that information is simply passed along by the tools that reference it. When a tool receives an update packet and needs to perform an action using the original user’s color, it obtains the information from the update. Consequently, the remote user does not receive a color selection tool because it would never be used.

4.3.2 Update Packet Objects

Update packets are used to transmit data to and from the tools within the application. Each update packet contains the name of the tool that instantiated it and the GUID of the owning user. Typically, tools will create a single update packet and simply clear the data after they have submitted it to the server. This is in contrast to creating a new update packet after every update is sent out—a practice that could degrade performance when actions are performed in rapid succession. When the whiteboard object receives new updates after a polling request, the data in each update packet is directed to the tool of the same type that created it.

4.3.3 User Objects

As previously mentioned, the user object encapsulates the state of a given user. This information is accessible via the whiteboard object so that all tools can get information about the currently-active users’ preferences and settings.

In addition, the user object contains event handlers for mouse actions that get passed down from the whiteboard object and the activity layer. These event handlers direct the mouse action to the tool that the user is currently working with.

It should be noted that in the current application, there are only two users. The local user and a single remote user that represents all other users. This model works within the system because updates are processed one at a time and each update is self-contained. As a result, the single remote user can update its internal state to match that of the update packet’s original user before making changes to the local user’s state.

4.3.4 User Input Handling

All tools contain generic event handlers for mouse actions. The primary object that the whiteboard assigns an event handler to is the activity layer. When an action is performed, such as a mouse click on the layer, the whiteboard object passes this event to the user. The user then passes the event to its selected tool, where the intended process is carried out.

4.3.5 Layer Objects

The layer objects each store a single canvas element and its associated context. The object provides a convenient method for calling canvas functions without having to repeatedly acquire the context. When initialized, the application specifies whether

\textsuperscript{9}A HEX value is obtained by converting a number to base 16, so the digits can range from 0 to 9 and A to F.
WebGL should be used to create the canvas. If WebGL is requested, a 3D context will be acquired. Otherwise, a 2D context will be obtained.

4.3.6 Tool Objects

While each tool object is unique, they do share common functionality. As mentioned previously, all tools contain event handlers for mouse input. In addition, they all have the ability to set the cursor to a custom-made canvas element. By storing references to both the whiteboard and the owning user, the tools are able to traverse through the application and obtain any necessary information to update and customize the environment.

Each tool also instantiates its own menu button and generates any setting interface elements that may be needed. Essentially, once the whiteboard requests that a tool be created, it is up to the tool object how it will behave and what it will visually offer to the user.

Finally, each tool maintains a variable that denotes whether it is being used or not. This is useful for many reasons, one of which is querying to see if the tool is working before processing a mouse event. For instance, in the brush tool, if the mouse button is not being held down it will not paint on the canvas and thus it is in an “off” state. When a mouse event propagates to the tool from the whiteboard, it is compared against the state variable to determine whether or not action should be taken.

4.4 jQuery

As discussed earlier, jQuery is an excellent JavaScript library for front-end development. In this application, it is used for three primary purposes—animations, user interface elements, and client-server communication.

4.4.1 Animations

All of the animations that take place within the application’s user interface are handled by jQuery. The jQuery library provides a very robust system with which to move and change HTML elements. By simply retaining either a DOM reference to the object or knowing the CSS ID assigned to it, jQuery can create very fluid interface effects using specified elements.

4.4.2 User Interface Elements

The jQuery library offers a companion add-on library called jQuery UI. The jQuery UI add-on contains many pre-configured user interface elements such as calendars, drag-and-drop support, and progress bars. Specifically, this application utilizes the slide bar element.

In addition, jQuery is also used to perform callbacks when text is entered into option boxes. With jQuery it is very simple to specify a function and under what circumstances it should execute following user input into a text box. It is important to realize that the use of jQuery for this purpose does not degrade performance in any way because manually setting up the event handlers would not be any more efficient and would be harder to manage.

4.4.3 Client-Server Communication

Modern browsers have the ability to asynchronously submit HTTP GET and POST requests. Setting up these requests using non-library JavaScript can become cumbersome, so jQuery takes care of the details for developers.

---

10 DOM stands for Document Object Model and is the convention for presenting and interacting with HTML elements.

11 An HTTP GET request consists of information that is passed along at the end of a URL.

12 An HTTP POST request consists of information that is transmitted in the message body, allowing for an arbitrary submission length.
When an asynchronous GET or POST request is submitted to the server, the calling function immediately returns and the program is allowed to continue executing. When the server responds to the request, a success function is triggered and the program is given an opportunity to manipulate the response data. Since it is unknown when the response data will be acquired, or if it will be obtained at all, care must be taken during development to ensure that the program's flow is not detrimentally affected.

In this application, the jQuery library is used to launch POST requests when submitting updates to the server. Similarly, a POST request is launched five seconds after each polling response as a means of maintaining the user's heartbeat and obtaining the latest updates.

All data transmission between the client and the server is done via JSON objects. This restriction makes parsing quite easy and enforces a standard for all tools to expect when updating.

4.5 WebGL

The 3D WebGL context is the preferred rendering method for the canvas element in this application. When the whiteboard object is being initialized it attempts to generate a WebGL context for a canvas object. It retains the success or failure of this attempt so that all tools can query the object and determine whether or not they can use WebGL.

If WebGL cannot be initialized, tools should react appropriately. This may mean that a tool refuses to work, but ideally it will have an alternate code path that allows it to use the 2D canvas context. The latter happens to be the case with the brush tool, being that it is at the core of the application’s functionality. If WebGL cannot be used by the brush tool, painting can still occur on a 2D canvas. There are some undesirable effects that are experienced when falling back to the 2D canvas and a discussion of them is provided in the later section dedicated to the brush tool.

4.6 HTML

HTML elements are the visual building blocks of the application. Various elements ranging from text input boxes to div objects are used to allow the user to interact with the program. These elements are constantly being updated by JavaScript functions to change the appearance of the application and give feedback to the user.

4.7 CSS

The CSS style attributes applied throughout the application help to define the visual appearance. As with the HTML elements, the CSS attributes are very frequently updated by JavaScript procedures.

The application does make use of CSS3 styling attributes via the border-radius property. This property, as its name implies, rounds the corners of an element’s border. This effect is presented on the user interface elements in the menu of the application. Most current browsers support this functionality, and if they do not the style declaration is simply dropped. Support is offered for recently outdated browsers such as FireFox 3.6 via vendor-specific styling\textsuperscript{13} where possible.

5 User Interface

The interface provided by this application is somewhat complicated being that web browsers are not specifically geared towards running desktop-like software. Many tricks and work-arounds were required to present an enjoyable environment to the user. These points of interest are discussed throughout this section. For

\textsuperscript{13}Vendor-specific styling refers to setting CSS properties that are considered experimental or not completed in a given browser. For example, FireFox 3.6 requires the developer to specify their CSS border radius using -moz-border-radius while FireFox 4.0 simply requests border-radius.
reference, Figure 8 presents the default interface for the drawing program.

The user interface for the application requires only a very thin menu bar on the left-hand side of the window. This minimalist design allows the majority of the screen space to be devoted to the painting canvas, which is the focus of the application. The buttons on the menu bar are rather complex objects and maintain various details about their state so as to present the proper appearance to the user.

When the user selects a tool, the tool has the ability to change the current cursor used within the application. The cursor may be set to any image or canvas element, so it has the ability to be dynamically generated. It should be noted here that the application uses a DOM element and repositions it to behave like a cursor because of the resolution limitations placed on actual cursors by the browser. Although the maximum size varies from browser to browser, large cursors with many hundreds of pixels in one or both dimensions (such as the brush cursor) are not allowed. If a cursor of a large size is requested, the browser simply displays the default arrow. In order to circumvent this, the actual cursor is set to a one pixel by one pixel PNG image with 99% transparency. This effectively hides the operating system’s cursor from the user and allows the application to manipulate its own cursor-representing image without interference.

5.1 User Interface Structure

In order to provide sufficient flexibility and functionality within the user interface, great care was taken to order and position the elements. For instance, consider that all the mouse input
distributed to the tools requires interaction with the activity layer. At the same time, the activity layer is not the layer that the tools send updates to. Furthermore, once the custom cursor and menu is added to the mix, it becomes rather complicated to ensure the user's actions are properly executed.

As shown in Figure 9, the user interface relies upon many different layers. At the top of the stack is the menu bar. This element is necessarily placed above all other elements on the page because the user must be able to interact with it without triggering events directed at the canvas. If any other layer was placed above the menu bar, buttons and settings might be visible but would be non-clickable. Similarly, any items associated with the menu bar, such as the flyout settings tabs for each tool, are placed at the same stack level. Again, this is to ensure that the user can interact with the proper elements.

Next, the activity layer is displayed. As discussed previously, the activity layer is responsible for obtaining all of the user's mouse input that is directed towards the canvas. When the user moves their cursor, clicks, or releases the mouse, the activity layer captures the event and passes it on to the appropriate, active tool.

The cursor element is displayed below the activity layer. Since the cursor can be a custom-defined image or canvas element, it has the ability to block mouse events from passing through to the intended canvas layers. In order to avoid complex structures of allowing or disallowing event bubbling\textsuperscript{14}, the cursor was simply placed below the activity layer. When the mouse moves on the activity layer, the application records the cursor's position and updates the location of the custom cursor object.

The local and remote feedback canvas layers are used when the application falls back to the 2D context mode. Their importance will be explained in the section devoted to the brush tool.

Finally, the base canvas is placed at the bottom of the stack. Once events have passed through the activity layer and been processed by the active tool, changes eventually end up here.

5.2 Menu Button Objects

In an effort to provide flexibility within the user interface, the menu button objects are very complex. Nearly all of this complexity is hidden from the developer, however, so any new tools and modules added to the application can be done so in a very straightforward manner.

When a button is created, it requests a reference to the whiteboard object, the name of the tool or module it will represent, and whether or not it has a flyout tab for user settings. The name is used to automatically acquire the proper icon to place on the button. For instance, when “brush” is passed to the button as the name of the tool, the icon located at the path “ui/button_brush.png” is used. This of course only the default behavior and the icon can be easily customized by the tool that owns the button via CSS.

The tab setting determines if the button will offer a settings interface to the user. If the button is created with a tab, an empty div element is created with pre-applied styling consistent with the rest of the user interface. This settings box is then provided to the tool to fill with whatever parameters it wishes. The separation here allows for very robust tools that offer complex, unique settings to the user.

5.2.1 Button State & Behavior

Beyond these settings, the menu buttons keep track of their current state by remembering whether they are currently pressed down or are receiving a mouse event. Since HTML doesn’t provide an element that can simulate different

\textsuperscript{14}Event bubbling is a property of JavaScript whereby a user’s actions can propagate to other elements in the DOM tree.
Figure 9: An exploded diagram presenting the different components of the base application user interface. From top to bottom, the elements are layered with the menu bar, the activity layer, the cursor, the local and remote feedback layers, and the base canvas.

button states including an indefinite depression, the buttons are all implemented using divs and background images. Thus, when a user hovers over the button div, an event is fired that the button object receives. The event is processed based on the current state of both the button and its tab, if it has one. The image representing the button can be either un-modified, hovered, or pressed down. If the button has a tab, the image representing the tab may be in one of nine different states:

1. Un-pressed (button up)
2. Un-pressed (button hovered)
3. Un-pressed (button down)
4. Hovered (button up)
5. Hovered (button hovered)
6. Hovered (button down)
7. Down (button up)
8. Down (button hovered)
9. Down (button down)

A single image is specified for each of the button and tab textures. By storing all of the states in a single image, the browser only needs to make one file request to load that specific component of the user interface. A common mistake seen with many web-based interfaces is the “pop-in effect” that can be seen when a JavaScript event, such as a mouseover, requests that an image be changed to one that has yet to be loaded. When this happens, the image that the user previously saw disappears and the element is left without a texture until the new image has been downloaded. This problem is avoided entirely by storing all the required state images in a single image and simply offsetting the background position to display the desired state. Figure 10 shows the tiled images used for the button and button tabs.

In addition, the menu button object also allows a tool to specify whether or not the button controls the tab. This is a very useful feature for tools that do not take control of the user’s canvas input, such as the chat and system settings tool. These two tools are placed on the menu bar for convenience and clarity, but do not steal the focus of the
Figure 10: The tiled background images used for the buttons, top, and button tabs, bottom. Both the button and the tab can be either up, hovered, or down. Since the button produces a glow effect that bleeds into the tab, the tab image set has nine different states. The images are enlarged to 200%.

application when activated. Consequently, the button object can decide whether a click event on the button will open and close the tab.

5.2.2 Settings Panel

The settings panel operates based on the state of the menu button object. When the menu button is created, whether or not it has a flyout tab determines if a settings panel is provided to the tool. If a tab is requested, the panel is created using default interface styling.

When the user clicks on the tab, the settings panel flies out from under the menu bar, which makes the interface appear to be more than 2-dimensional. This is an important step towards developing robust and effective interfaces in web applications.

The settings panel is oriented such that it sits completely hidden behind the menu bar if it is retracted. When opened, the panel is flush with the menu bar. The top of the settings panel is positioned so that the title bar is in line with the white semi-circle in the button tab. This creates a continuity between the menu bar and the settings panel that keeps the interface uniform and understandable.

Note that the menu button object also has the ability to invert the settings panel. If this feature is requested by a tool, the bottom of the settings panel will contain the title bar. This means that if a tall settings box is required by a tool that is positioned at the bottom of the menu bar, the settings panel can be inverted to ensure the information stays on the screen. If the settings panel were not inverted for a button at the bottom of the menu, the information would extend downwards out of the window. This functionality is utilized by the system information tool.

5.3 Layout Motivation

In many commercial software packages it becomes difficult for the user to locate the settings they need. This is especially true in multimedia programs simply because so many parameters are offered to the user. In this application, the settings were carefully placed in the interface so as to make them intuitive and easy to access.

Since the settings panel provided to each tool is necessarily linked to the location of the button via the flyout tab, the user never has to search for anything. If they need to change a variable for any tool, they simply navigate to the menu bar and click on the tab next to the desired tool. While being very minimalistic, this is also easy and straight-forward for the user.

6 Toast Notification Tool

A toast, with respect to computer software, is a visual notification that pops up on the screen, usually via a vertically-oriented animation. The name is derived from the appearance of toast popping out of a toaster. In this application, toasts are implemented as a privileged tool because they
are very tightly integrated with the whiteboard object. Their functionality is outlined in the sections that follow.

6.1 Visual Appearance

Since the toast notifications are used to provide the user with information quickly and cleanly without significantly interrupting the painting experience, they are displayed in the lower right corner of the application. Figure 11 shows a toast being displayed following a new user’s connection.

![New User](image)

Figure 11: A toast notification letting the local user know that another user has connected to the application.

Both the title bar and the message box are customizable, so toast notifications can be used by all aspects of the application. When a toast is displayed by the application, it slowly rises up from the bottom edge of the user’s browser window. After a short period of time, the toast slowly sinks down below the bottom of the browser window.

6.2 Whiteboard Integration

In order for a toast to be displayed, the whiteboard object must receive a toast request. The request must contain four parameters:

1. Title
2. Message
3. Priority
4. Lifespan

These four parameters allow any part of the application to display information to the user in a relevant manner.

First, the title can be set to any string of characters. If the desired title is longer than the space allowed in the title bar of the toast, it is truncated to the maximum width.

The message for the toast can also be specified as any string of characters. If the message that is requested is longer than roughly two lines of text, it is truncated and an ellipsis is placed at the end.

The priority of the toast can be set to any positive integer. The higher the priority level of a toast, the sooner the user will see it. Since only one toast may be displayed at a time, the priority level allows tools in the application to specify how important a notification is. For instance, the whiteboard object requests a level 10 toast to notify the user if their connection to the server has been lost. If there are no toasts being displayed at that time, the notification appears immediately. If there is a toast being displayed, the newly-submitted one is placed in a priority queue. This enables the user to receive more important information closer to the time that it occurs. By default, the toast title bar changes from green to orange to red depending on the priority level assigned to the notification.

Lastly, the lifespan assigned to the toast request specifies how long the toast will be displayed on the user’s screen. Typically a value of five seconds is sufficient, though the value is not moderated by the application. After a toast has finished its appearance animation by sliding up onto the screen, the lifespan timer begins. When the lifespan timer ends, the toast hides itself and lets the whiteboard object know that the next notification may be displayed, if there are any in the queue.
6.3 Access & Usability

Since the toast tool is heavily integrated with the whiteboard object, all other tool modules have the ability to request toast notifications. This is a very useful feature from a development standpoint as it allows new features to provide information to the user in an unobtrusive manner. By simply supplying the necessary parameters for the toast to the whiteboard, a tool can continue working knowing that the notification will be displayed in a timely manner. Furthermore, because each tool defines its own rules for when to request a toast, the notifications can be highly customized.

7 Brush Tool

Since the core functionality of this application relies upon the brush tool, it is the most robust and functional item in the program. Using both traditional desktop programs like Photoshop and more recent web-based offerings as a reference, key features were replicated for the browser platform. In addition, new features were implemented that have yet to be seen in any browser-based canvas painting application.

7.1 Initial Setup

Since JavaScript cannot query the web server directly for information, some pre-processing must take place before the brush tool is ready to use. Specifically, the tool must know what brush tips are available for selection. This is determined by a small PHP script that scans a pre-defined brush directory for image files. For each image that is found, the path to the file is added to a string of JavaScript code that declares a variable. When the PHP script finishes, it returns the JavaScript string to the browser. Thus, PHP is used to dynamically generate JavaScript during the setup process.

Once the available tips have been reported to the brush tool, the images are loaded into memory. This step ensures that the user does not have to wait after selecting a brush tip in case the image file has not yet been downloaded. As the brush tips are retrieved by the browser, they are added to the settings panel as thumbnails for easy reference.

During this setup process the brush tool also determines whether it will be using the 2D or 3D context for the painting canvas. Since much of the WebGL implementation relies on the 2D context code, the primary difference is whether WebGL objects will be initialized and used.

If WebGL is going to be used, the tool must create multiple objects for use with the 3D rendering mode. Most importantly, the vertex and fragment shaders are loaded and compiled to ensure compatibility with the user’s system. Since WebGL does not have transformation matrices built into its API, these objects are created during the setup phase via Tojiro’s glMatrix library [12]. Finally, various framebuffer objects\footnote{A framebuffer object is simply an array of pixels that can be manipulated without being presented to the user on the screen.} are created to help with the rendering process.

7.2 Settings Panel

The settings panel for the brush tool contains access to many customizable options for the user to create the perfect brush for their task. This interface is shown in Figure 12.

Within the settings panel are a variety of sliders, text boxes, and checkboxes. Each of these interface elements updates the user’s brush in real-time. Sliders update the value of the parameter when the user stops dragging them—this prevents the user’s motion from being interrupted by new brushes being generated. Text boxes update the brush on a per-keystroke basis. The checkboxes update the parameter immediately upon being clicked.
7.2.1 Tips

At the top of the settings area is a section for brush tips. Each of the brush thumbnails is dynamically generated during the setup process after the tip has been loaded into memory. The thumbnails act as buttons which, upon being clicked, set the user’s active tip shape. It should be noted that the tips used within the application are sampled from Photoshop’s brushes, though any image data can be used as a brush.

7.2.2 Shape Parameters

The next section contains parameters that allow the user to customize their selected tip’s shape. Users may adjust the diameter, rotation, roundness, and axis-aligned flipping of the tip.

The diameter interface elements are limited to the range $[1, 500]$. Depending on the browser version being used, very large brushes can incur a performance penalty. Using WebGL and a recently-released browser, however, avoids such issues. For capable machines, this upper limit can be raised much higher to many thousands of pixels (although this would be impractical to paint with).

The rotation slider has a range of $[0, 360]$. The text box will accept any positive integer value and simply set the value to the modulus of that value and 360 to bring it into the range $[0, 360]$.

The roundness percent may be adjusted over the range $[1, 100]$. Although a value of one typically produces a one-pixel thin brush, this is sometimes desirable depending on the texture the artist desires.

Finally, the X and Y flip checkboxes swap the left and right pixels and the top and bottom pixels of the tip, respectively, around the center axes.

7.2.3 Application Parameters

The next section allows the user to specify how the color is applied from their brush to the canvas. The opacity setting behaves as was described earlier. When the value is updated here, no immediate change takes place as the parameter is used when the user is actively painting.

Regardless of which rendering mode the application is using, the alpha value of each pixel in the source tip is scaled by the specified flow parameter. As expected, this results in a more transparent brush tip.

7.2.4 Shape & Stroke Preview

Finally, at the bottom of the settings panel is a section that provides the user with a real-time example of their current brush parameters. This
section displays the brush scaled down to fit within the resolution of the interface.

The box on the left displays a single stamping of the brush as it would appear with the user’s current settings. Black is used instead of any currently-selected color so as to make it easier to see updates to subtle changes in opacity and flow.

The box on the right displays a curved stroke using the current brush settings. As with the stamped brush on the left, black is used to help show updates more clearly. The points used for this stroke remain the same and are actually pre-loaded into the application. This improves the performance of updating the interface as the user adjusts their parameters. Since complex drawing calls are not needed to determine a smooth curved path for every update, the process is rather efficient.

### 7.3 Contour Outline

While various browser-based painting applications provide the user with a cursor that represents their current brush, none do so for non-round tip shapes. A contour tracing algorithm, based on work by Erik Smistad [11], was implemented here to address this exact issue.

The contour tracing algorithm is an extension of Moore Neighbor Contour Tracing. The extended algorithm continues to trace through the image to outline contours inside the identified shape, which allows images with openings and holes to be accurately traced. This is ideal for presenting the shape of a brush as the contour directly represents where the color will be applied to the canvas. Figure 13 shows an example of the contour cursor for a selected maple leaf brush. Note how the cursor is visible on both light and dark backgrounds.

Starting from the top left of the brush tip, the image is scanned from left to right. If a non-transparent pixel is found, the contour line that the pixel belongs to is walked in a clockwise fashion. In order to complete this process, each pixel’s eight neighbors are checked for transparency. If a non-transparent neighbor is found, that pixel is moved to and the tracing continues. When the tracing completes, the algorithm returns to the pixel where it entered the line and continues moving left to right. After a line has been finished, a flag is set to remember whether the process is inside or outside a previously-identified contour. This enables the algorithm to pick up on all contour regions through the traced image.

Once the user’s brush tip has been traced, it is stamped onto a canvas as a black outline. An offset is applied of one pixel horizontally, and the contour is stamped onto the same canvas again as a white outline. Both outlines are stamped with an alpha-transparency component so that they are not completely opaque. This is an important technique to make note of as most multimedia programs display the cursor by performing an XOR logical operation to invert the base canvas. Since an operation like this is not natively built in to the canvas API, a per-pixel traversal would have to take place every time the user moved the
cursor. As this is not feasible due to performance reasons, the approximation used here suffices.

Following these steps, the cursor generation process is complete. After this, the stamped canvas is sent to the whiteboard object where it is set as the application’s active cursor.

7.4 Shape Selection

Unlike most other browser-based painting applications, the shape of the tip used for painting is not limited in any way. By default, the application references pre-loaded brushes stored in a specified directory. If the user has access to the web server where the application is being hosted, they can simply upload an image file to the brush directory and the application will automatically offer it as a tip the next time it is loaded. Since the brushes are based on sampled images, there is no restriction on what can or cannot be a tip shape.

7.5 Color

The color of the tip is set differently depending upon the rendering mode being used by the application.

If the 2D context is being used, color is set individually for each pixel in the brush tip. The pixels in the tip are traversed and each one has its RGB value set to the user’s current foreground color. The alpha component of each pixel remains unchanged.

If the WebGL context is being used, color is not referenced until the actual painting process occurs. The fragment shader that applies color to the canvas queries the user’s foreground color and simply adjusts its output. This certainly results in a performance boost over the color-setting procedure of the 2D context mode.

7.6 Resizing

The size of the brush tip is equal to the maximum dimension, whether it be width or height. This definition helps to maintain consistency between brushes of different aspect ratios.

When the brush is resized, the scale factor to transform the maximum dimension to the desired size is calculated. Both the width and height are then multiplied by this factor to produce an isotropically-scaled image. Note that the brush is not permitted to resize itself to the point where one or both dimensions are zero. This ensures that the user’s tip will always be capable of dispensing some amount of color onto the canvas.

7.7 Rotation

In order to rotate the selected tip by an arbitrary amount, a temporary canvas must be used. The dimensions of the temporary canvas use the length of the diagonal of the source tip as the width and height. This ensures that the rotated brush tip will not be clipped by the bounding region of the canvas.

After rotating the brush about its center, there is typically a massive amount of excess padding that remains. Although this excess padding is all transparent and would not affect the application of color to the canvas, its size contribution can degrade the performance of other steps in the brush generation process. Consequently, the minimum bounding box for the rotated tip is determined and the brush is cropped to that region.

7.8 Axis-Aligned Flipping

The flipping process is very straight-forward. If the brush requests an axis flip, the pixels are traversed along the desired dimension until the halfway point is reached. At each pixel, the color value is swapped with the pixel that is at the same location but mirrored over the desired axis. The
process for X and Y flipping is nearly identical with the difference being the traversal pattern.

Note that the flipping of the brush takes place prior to rotation. This sequence ensures that a flipped brush does not produce unexpected results as the user rotates it.

7.9 Roundness

The roundness of the brush is simply a measure of its height. At 100% roundness, the brush exhibits its original aspect ratio. Any value lower than 100% results in a scaling of the height by that value.

Note that the roundness of the brush is set prior to rotation. If this sequence was not enforced, the brush would change its shape as it was rotated.

7.10 Spacing

The spacing of the brush determines how many pixels the user must draw over between subsequent applications of the tip to the canvas. When calculating the value, the size of the brush is used. This allows for different results depending on the particular brush and its dimensions.

Figure 14: A curved line drawn with a spacing value of 177%. Despite the fact that the pixels in between the dots were passed over, the brush distributes its color sparsely.

Figure 14 displays an example of brush spacing within the application. The curved line was drawn as a single stroke. Despite the fact that the pixels between the dots were passed over, the brush did not paint into the canvas continuously because the spacing was set to 177%.

7.11 Flow

The flow of the brush can be thought of as controlling the percent transparency of each pixel in the tip. At 100% flow, the alpha component of each tip pixel is unchanged. For lower values, the alpha components are scaled to produce a more transparent brush. This parameter ultimately allows the user to gradually build up color on the canvas.

7.12 Opacity

The opacity parameter requires some rather complicated actions to take place in order to produce the expected results. These actions are different depending on the rendering mode being used.

7.12.1 2D Context Mode

When using the 2D context, changing the opacity value results in two events:

1. Setting the visual opacity of the feedback layer
2. Setting the global alpha value for the context of the base canvas

The first event takes place in order to provide the user with accurate visual feedback while painting. By setting the opacity value of the feedback layer, which is the temporary layer that all strokes paint into before being transferred to the base canvas, the range of color values is limited. Specifically, 100% opacity gets mapped to the opacity value specified by the user. This change in scale presents the user’s brush strokes accurately with respect to their opacity parameter.

The second event properly scales the data contained on the feedback layer for application to the base canvas layer. Recall that since the
opacity of the feedback layer was changed and not the information being deposited into it, the pixels actually contain more opaque data. When the global alpha value for the base canvas is set, the alpha value of all data drawn to it is properly scaled. This results in the exact appearance of the feedback layer being displayed on top of the base canvas layer but in a single layer.

7.12.2 WebGL Mode

When using the WebGL context, changing the opacity value does not result in any immediate events. Instead, the value is referenced when the painting process occurs. The specifics will be discussed in the section devoted to the brush painting process, but the general idea is similar to that of the 2D context. In this case, however, the separate layers are implemented as framebuffer objects and a temporary canvas is produced on the screen.

Note that both of these methods respect the flow parameter, as the flow simply controls how fast color builds up to the given limit. This is the first time the dynamic between flow and opacity has ever been displayed in a canvas-based painting application.

7.13 Mouse Events

Without mouse event handlers, the brush tool would not be able to convert the user’s actions into paint strokes. Thus, they are a critical component of the tool.

When the user clicks on the activity layer, a flag is set in the internal state of the tool noting that the brush is now working. Any subsequent events can reference that flag in order to take appropriate action.

As the user moves the cursor around the canvas, a mouse movement handler is executed. When the event fires, the flag that denotes if the tool is working is checked. If the tool is indeed working, the brush applies the tip to the canvas. If the tool is not working, the brush is not applied. Recall that the mouse events are always propagated from the activity layer to the active tool. This makes the previous check necessary because an event will be received by the tool denoting that the cursor has moved with no knowledge of the mouse button state.

Finally, when the mouse is released, the flag is set to note that the tool is no longer working. When this occurs, a procedure executes that handles the user’s recently-made stroke.

7.14 Painting Process

The painting process requires multiple steps in order to translate the user’s mouse input into brush strokes that appear on the final canvas. Various different layers are used to provide both real-time visual feedback and proper compositing depending on the rendering mode. The details of these steps and components are outlined in the following sections.

7.14.1 Creating Strokes

Since JavaScript does not have access to a mouse buffer in the same way a desktop application might, gaps necessarily appear in the sampled readings. Ideally, the brush tool would like to receive the location of every pixel that the user’s cursor passes over. This is impossible within the environment of a browser application, especially when the user makes fast mouse movements. In order to work around this issue, Bresenham’s line algorithm is used.

The Bresenham line algorithm is perfect for filling in the gaps in user input data because it provides clean, one-pixel thick lines connecting two end points [1]. Although one-pixel thick lines are not necessarily the intended stroke result, the location of each pixel between the two end points serves as a reference for where the mouse would
have passed over had the polling rate been high enough. Furthermore, the Bresenham algorithm avoids floating point operations, so it is very fast and efficient. Given that the generation of brush strokes occurs extremely frequently, this benefit is immense.

7.14.2 Mouse Smoothing

Unfortunately, when using the Bresenham line algorithm to fill in the gaps between mouse location readings, the connecting lines are perfectly straight. In practice, users will very rarely want, or even be able to, draw an exactly straight line. Instead, their physical brush strokes will tend to have curvature to them and will flow smoothly. In order to mitigate this discrepancy, mouse smoothing using Catmull-Rom splines [2] is used.

Catmull-Rom splines create an interpolation path given four input points. The two end points are used to determine the incoming and outgoing tangent directions for the curve that passes through the middle points. By default, the interpolation curve does not pass through the start or end points, but this can be forced by supplying the start and end point twice and generating multiple segments. Furthermore, the computations needed to determine intermediate points on the curve require simple addition, subtraction, and multiplication, which makes the algorithm very fast. As a result, this type of interpolation spline is perfect for smoothing out the rather intermittent mouse input supplied by the browser to the application.

As the user draws on the canvas, the mouse readings that the application is able to receive are stored in an array. The distance between every two sample points is calculated and if it exceeds a threshold of 20 pixels, the gap is smoothed using points on the calculated spline. This produces substantially better results than if the input were not smoothed, as seen in Figure 15, though it is by no means ideal. Given the current possibilities of applications running with a browser, however, it is somewhat unrealistic to expect better results.

A note should be made about the selection process for \( t \) values when solving for spline-based interpolation points. When producing Catmull-Rom splines, the equation requires an input value, \( t \), in the range \([0, 1]\). Unfortunately, the points returned based on the value of \( t \) do not correspond to their position on the curve. For instance, a \( t \) value of 0.5 will not necessarily produce a point halfway on the spline between the two points. In order to select values like this, the spline equation would have to be re-parametrized. This process is not overly complicated, but the calculations required are too costly to run while
the user is drawing in real-time. Instead, the value of $t$ is stepped from zero to one at an interval of $t_{\text{stepping}} = 0.75 \times \max(|x_0 - x_1|, |y_0 - y_1|)$. This stepping value uses the larger differential between X and Y coordinates to determine approximately how many pixels should be filled in. The 0.75 scaling factor helps disperse the samples. Furthermore, if the point sample returned by the spline equation is within one pixel of the previously-accepted point, it is rejected. This helps to reduce jitter in the smoothed line resulting from spline samples that were taken very close to one another.

### 7.14.3 2D Context Mode

The 2D context painting mode is reserved as a fall-back for users with browsers that do not support the higher quality WebGL mode. When a user paints in this mode, their brush is stamped onto the local feedback layer. The local feedback layer serves as a barrier between the base canvas and the user’s input so that opacity values can be scaled properly. As the user paints into the feedback layer, the opacity values of the pixels increase until they are completely opaque. However, since the feedback layer had previously been styled with a CSS attribute that made it somewhat transparent, the user views their strokes as though they are limited to the correct opacity. When the user finishes their stroke, the data on the feedback layer is transferred to the base canvas.

In transferring the data to the base canvas, the opacity value of the feedback layer must be respected. Since the pixel values actually have much higher opacity information than what is visible, the global alpha parameter of the 2D canvas context is set. This scales the opacity value of all incoming pixel data in a write operation, effectively preserving the level of transparency of the user’s stroke.

After the stroke has been transferred to the base canvas layer, the feedback layer is cleared of all data so that the user’s next stroke may repeat the painting process.

Of particular importance here is the fact that the quality of the results when painting using the 2D context is worse than when WebGL is used. The reason for this is the presence of alpha pre-multiplication within the canvas-based system. Since the brush tips are saved in the PNG file format they necessarily have their RGB values pre-multiplied by the alpha component. For example, if a pixel has an RGB value of $[100, 50, 0]$ and an alpha component of 175, the color will be stored in the PNG file as $[69, 34, 0]$ after being scaled by the 0.686 alpha value. Unfortunately, this ultimately results in blending artifacts when the user’s strokes are composited onto the canvas, as seen in Figure 16.

### 7.14.4 WebGL Mode

In order to circumvent the blending artifacts produced when painting in the 2D context mode, a WebGL mode is offered within the application. Although the painting process varies, the only difference in the results is in the blending.

When the user makes a brush stroke while in the WebGL mode, the brush tip is stamped into a framebuffer object. This framebuffer is then drawn, with scaled opacity values, on top of another framebuffer representing the canvas as it appeared before the user began their stroke. This composited image is then displayed for the user. As the user continues their stroke, the framebuffer is updated with new stamps from the brush tip and the layering process keeps occurring. When the user completes their stroke, the temporary framebuffer is merged with the original state of the base canvas while respecting the opacity limit. This effectively emulates the process for the 2D context, but has the benefit of accessing fragment

---

16 The alpha value of 0.686 is calculated by dividing 175 by the total possible value of 255.
Figure 16: The poor results seen when blending PNG brushes into the base canvas with the 2D context, left, is compared to the desired result, right. The source brush was set to an opacity value of 1% and stamped on the canvas 100 times. Since the PNG file format stores the color values pre-multiplied by their alpha component, the blending is incorrect. The example on the right depicts the type of results achievable with the WebGL mode. Specifically, note how the left image contains regions of gray despite the brush color being specified as red.

shaders for manipulating pixel values in their pre-multiplied state.

Since all of the pixel colors that get stored on the base canvas layer pass through a fragment shader in the WebGL mode, it is very simple to revert the premultiplication, perform blending, and store the blended color as a pre-multiplied value. The major reason why this is possible in the WebGL mode and not with the 2D context is performance. Technically the color values could be individually calculated and blended when using the 2D context, but the amount of computations necessary would present too much of an interruption to the user’s mouse input. Since the fragment shader runs in parallel on the GPU, the computations can be done extremely quickly so that high quality, accurate blending is possible.

7.14.5 Color Precision

It should be noted that some minor artifacting can be seen in both modes due to the fact that the canvas element consists of 8-bit color buffers. This means that each color channel can have one of 255 different values. When very low opacity or flow values are specified, fractional color values get rounded to integers and some detail is lost. The only way around this is to maintain a separate pixel buffer with a higher precision and do all blending calculations in that space. Unfortunately, the performance of such an approach is likely not suited to running on the web in a browser.

7.15 Submitting Updates

Since the goal of this application is to offer a quality, multi-user painting application that can be run in a browser, the data sent over the internet is minimized. It would be possible to have each user upload an image representing their brush stroke after every action, but this would consume massive amounts of bandwidth and would be very slow. Instead, the information needed to re-create each user’s actions is uploaded to the server after every stroke.

When a user paints a line onto the canvas, the pixel coordinates that were used to stamp the brush tip are recorded. Upon release of the mouse, and thereby termination of the current stroke, the coordinates are uploaded asynchronously to the application’s web server. Accompanying this data is information noting all of the user’s brush parameters. This allows the remote users to
re-create the stroke exactly as it was made once they receive the update.

7.16 Processing Updates

When an update packet is received by the brush tool it typically contains multiple brush strokes from various users connected to the application. The tool processes the updates sequentially by performing the same painting process that occurs when the local user is editing the canvas. Essentially, the update packets contain everything needed to re-create the state of the remote users so that their actions can be exactly re-created.

Of particular importance here is that both the 2D context and WebGL modes supply a separate remote feedback layer. This layer is painted into so that the state of the local user is not modified when processing updates. For instance, if the local user is making a brush stroke with an opacity of 50% and an update comes in for a stroke with 100% opacity, the actions must be processed separately. If the actions are interwoven, the remote user's stroke would modify the local user's state and change the result of both strokes. By separating the two, the local user's state is completely encapsulated.

8 Chat Tool

Being that it is often easier and faster to type a message than it is to write one, the application offers a chat tool to all connected users. The tool does not take control of the user's mouse input, so they are free to paint while the chat window is open and active.

8.1 Settings Panel

The settings panel for the chat tool is very minimalistic. As seen in Figure 17, there is a small window for chat messages and a text area at the bottom where the user can submit text. As new messages come in they are displayed at the bottom of the window so that old messages scroll upwards.

Figure 17: The settings interface for the chat tool.

8.2 Scrollbar Management

Since the window where chat messages are displayed necessarily has a dynamic height, it can be difficult for the user to scroll up to read a past message and remain in the desired location. In order to alleviate this issue, an intelligent scrollbar is implemented.

Before a new message is added to the chat window, the position of the scrollbar is checked. If the scrollbar is at the bottom of the window, it maintains its position at the bottom so that the most recent messages can be read. However, if the user has scrolled up to read a message, the scrollbar does not automatically move to display the most recent message. Instead, it remains at its current location, allowing the user to continue reading. When the user scrolls back to the bottom of the chat window, the scrollbar again auto-scrolls to show the latest messages.

8.3 Username Setting

When users first connect to the application they are provided with a generic name consisting of the
word “User” and their database ID number. This can be changed by typing `/name <desired_name>` into the chat window. Upon receiving this as input, the tool checks with the server to see if the name is being used by any other active users. If it is available, the user’s name is updated and all connected clients are notified of the change. If the name is not available, the local user is notified and no change takes place.

### 8.4 Sanitizing Messages

It is important to recognize that since the users can submit any text input they wish into the chat window there is a possible security hole. When chat messages are displayed on the screen, they are placed inside HTML elements and parsed by the browser. If the content of a message happened to be malicious JavaScript or some other browser-based exploit, all connected users could be affected. This is easily avoided, however, by sanitizing the submitted messages.

When a message is submitted to the tool, all characters are converted to their HTML encoding. This conversion process ensures that things like script tags and DOM objects simply appear as source code instead of actually being interpreted by the browser. As a result, all connected users are protected no matter what messages are transferred.

### 8.5 Toast Notifications

Since the chat window takes up valuable working space in front of the canvas, it is likely that users will keep it closed while working unless they want to submit a message. This practice would prevent them from seeing what other users are saying, though, which could result in failed cooperation while painting. Using the whiteboard object’s toast notifications, this is a non-issue.

When an update packet is processed by the chat tool, the state of the settings panel is queried. If the chat window is currently closed, a toast notification is requested containing an abbreviated version of the received message. This results in a small, unobtrusive notification in the corner of the user’s browser. Based on this notification, the user can then decide to join in on the conversation. If the user has the chat window open when the update is processed, the toast is not displayed.

### 8.6 Submitting Updates

The chat tool has a very simple process for submitting updates. When the local user types a message and hits enter on their keyboard, the tool sanitizes the text and places it, along with their username, in an update packet. The update packet is then forwarded to the whiteboard object where it is sent to the server.

### 8.7 Processing Updates

When an update is received by the chat tool it is processed in an equally-simple manner. The chat message and its author are extracted from the packet and the two pieces of information are placed in a styled container div. This div is then appended to the chat window so the user may view the message.

### 9 Color Selection Tool

The color selection tool is a very straight-forward component of the application. It provides the user with a color gradient where they can choose both a foreground and background color.

#### 9.1 Settings Panel

The color selection tool’s settings panel contains a selection interface based on HSB\(^{17}\) values, as seen in Figure 18. The hue strip on the right allows the user to select the particular color they want.

\(^{17}\)HSB stands for Hue Saturation Brightness.
The saturation and brightness gradient on the left allows them to select the intensity. By clicking and dragging up or down the strip, the hue can be confirmed. By clicking on a pixel in the gradient box, the color at that pixel will be selected.

If the user wishes to select a color from the settings panel, they simply click on a pixel in the gradient box. With the tool activated, the user may also sample the pixel colors in the canvas. When sampling in this mode the cursor changes to an eyedropper to visually support the functionality of the tool.

9.3 Color Chips

The color chips represent an instance of a tool adding both a button and an item to the menu bar. Menu items, unlike buttons, do not have any functionality built in and thus all management is left up to the tool.

The foreground and background color chips act as a convenient method for the user to store the colors they are currently working with. When the foreground chip is clicked on, the selection tool will update the foreground color with any samples it observes. Similarly, clicking on the background chip uses the tool’s samples to update that color.

The color chips also have two small buttons that allow the user to swap their foreground and background colors as well as reset them. When resetting, the foreground color is set to black and the background color to white.

10 System Information Tool

The system information tool provides the user with information about the current state of the application. It is provided with updates by the application after every polling response.

10.1 Settings Panel

The settings panel for the system information tool consists of various text entries depicting the current state of the application. As seen in Figure 19, the panel uses the inverted option provided by the menu button object.
10.2 Users List

The list of currently-connected users is updated after every polling response from the server. If a new user connects or a user leaves the application, the list is updated and a toast notification is provided to the local user. Note that since the list of users changes over time, the height of the settings panel can change. In order to keep the panel aligned with the menu button, the position is reset as needed.

10.3 Initial Update ID

As discussed earlier, users may access the application with a request to start at a given update ID. The update ID that the user began working from is displayed in the information panel should the user need to reference it later.

10.4 Latency

After every polling request, the amount of time spent waiting is calculated. This span of time is displayed here so that the user can gauge the responsiveness of the application’s communication with the web server.

10.5 Rendering Mode

The current rendering mode is displayed in the information panel so the user can pinpoint any performance or quality problems between the two code paths.

11 Future Work

Although this application is fairly successful in providing a multi-user painting application, there are certainly areas that could be improved and extended. Since the code for the program is structured in such a way that new tools can easily be incorporated, many of these features would integrate nicely with the current system. Other ideas would require more substantial refinements to the overall application structure.

11.1 Full State Support

As mentioned throughout the discussion, the user has the ability to load the application with a given state corresponding to a database update ID. This allows for continuity between sessions, but becomes impractical as the session length grows. A better approach would be to maintain a version of the canvas on the server in much the same way that the connected clients do.

Throughout the development process a feature was implemented and tested that processed every update using the PHP GD Image Library. As the updates were received by the server, an image would be updated using the PHP equivalent of the JavaScript that runs in the client browsers. As a result, when a new user connected to the application, they could simply download a single image to view the current collective state. Unfortunately, in practice this feature did not perform very well.

The major issue with the feature was the processing time. Being that the web server does not apply any GPU acceleration when
manipulating images through the GD Image Library, everything is computed on the CPU. With thousands of pixels needing to be blended together many times per update, the response times were unacceptable. Following the processing of a typical update packet the server would occasionally time out and no state update would take place because the image processing code took too long to execute. Consequently, this feature was abandoned.

As an alternative to this approach, it would be interesting to experiment with alternative server-side image libraries such as ImageMagick. Furthermore, it would be possible to set the server up to queue the updates and use as much time as needed to process them. This would allow any new users to get a relatively-recent copy of the application’s state and then update locally from there.

11.2 Infinite Canvas

In the current implementation of the application the canvas is limited to the resolution of the browser window. It would be nice to allow users the ability to pan around the canvas and select a new area to begin painting. Since the application supports multiple users, it stands to reason that eventually the limited space of the browser window will force users to begin painting over each other’s work. Offering users space in any direction would mitigate this issue.

This feature could be implemented by allowing the user to drag and drop the base canvas. When the canvas is dropped, the amount of translation in both the X and Y directions could be recorded and the canvas could be stamped onto a new canvas. All painting data sent and received from the web server would have to be offset by this translation to ensure a consistent state between the users. Such an implementation would be accompanied by the challenge of how to provide updates based on the region of the global canvas currently being viewed. This could likely be taken care of by filtering paint updates based on their coordinates in the database.

11.3 Additional Brush Settings

While the majority of the important brush settings are offered in this application, the feature set could be extended somewhat.

11.3.1 Hardness

The hardness of the brush would be a welcome parameter for the user to adjust. Currently, the user has a great deal of control over the shape of their brush but not necessarily its intensity.

Hardness could be implemented by simply blurring the source brush tip. The intensity of the blur could be controlled by the user to provide different hardness levels. Since the process would only need to occur once per brush generation, any performance penalty incurred would be warranted.

11.3.2 Application Mode

In the current application the user can only paint on the canvas with the brush tool. Most desktop painting programs also offer airbrush and pencil painting modes. In the airbrush mode, paint is continually applied to the canvas as long as the user has the mouse held down. The pencil mode filters the brush tip to eliminate semi-transparent pixels via a pre-defined threshold. Implementing each of these application modes would be straight-forward within the current system.

11.4 Additional Tools

In addition to a brush tool, multimedia programs typically offer other ways to interact with the canvas. Given the robust canvas API and the use of WebGL, many of these tools could be implemented.
11.4.1 Selection Tool

Marquee and lasso selection tools are useful for masking areas of the canvas to either accept or reject paint. Both the 2D and WebGL contexts offer methods for performing masking, so implementing a selection tool would involve primarily interface work. Animating marching ants around the defined selection would likely be the challenging component of this feature.

11.4.2 Text Tool

The 2D context offers a method for drawing text to the canvas. A small issue might be encountered given that each user could have different font libraries installed. This could likely be circumvented, however, by only referencing fonts available to the application on the web server.

11.4.3 Eraser Tool

An eraser tool could likely re-use much of the code implemented for the brush in the application. Notably, since the eraser would have to respect an opacity value and limit the amount of color that can be removed per stroke, the tool would be rather complicated to implement for the 2D context mode. The WebGL rendering mode could use a similar approach to painting opacity and just subtract the opacity framebuffer from the base canvas layer.

11.4.4 Move Tool

The ability to move painted elements around would be very useful. In the current application, once paint is applied to the canvas there is no way to change its location. Since many users will want to paint at the same time, it would be valuable to have the option to move regions of the canvas around so as to preserve content.

11.5 Full Layer Support

Although the current system could technically offer users more than one layer to work with, they are presently limited to one. This restriction is primarily due to the lack of an eraser tool. Since users must currently draw with white paint in order to erase, there is no way to reveal layers that might be lower in the stack. A true eraser would allow for this feature.

It should be noted that additional logic would likely be needed to ensure that all users are presented with the same layer ordering. Alternatively, it could be left up to each user to view the layers in whatever order they desire.

11.6 Canvas Zooming

In order to develop very fine details it is often useful to zoom in on the canvas. The current application does not support this. The WebGL rendering mode would lend itself well to this extension, however, as a simple translation along the Z-axis could be performed to zoom in and out. Zooming in the 2D context mode would likely be more involved.

11.7 Long Polling

The polling system used currently fires a request at regular intervals regardless of whether it is obtaining new data. For improved efficiency, a technique known as long polling could be used. Essentially, with long polling a request is received by the server and the server does not respond until it has information to provide or a maximum waiting time is reached. This procedure typically requires a specially-configured web server, though, and may limit the deployment platforms for the application.
11.8 Graphics Tablet Support

Although one of the key points of this application was to avoid third-party plug-ins, the goal was set because current web technologies perform many of the features offered by plug-ins. In the case of graphics tablets, web browsers offer no support. Thus, it would be worthwhile to allow users the opportunity to use their tablets even though it would require a small additional download.

11.9 User Interface Refinements

Following the development of the application, a few areas of the interface that could be improved were noted. While not critical to the program's performance, attention to these items would result in a more pleasant user experience.

11.9.1 Drag & Drop Interface

While the user interface is structured with minimalism in mind, it is still easy for the various items to obscure the work area. Being able to drag and drop the items, such as the settings panels, would greatly reduce this issue. Given the opportunity to move interface items around, users would be able to place things in areas they are not currently working in order to efficiently use their allotted space.

11.9.2 Slider Refinements

Even though the sliders in the settings panels work as intended, their functionality could be improved slightly. For sliders that represent a very large range of values, such as the brush spacing parameter, the values at the lower end are typically used more. Since the slider uses a linear scale to determine the selected value based on the handle position, it is often difficult to precisely select a small value. Instead, a varying scale that, for instance, allocates half of the slider to the first quarter of the possible values would improve its functionality drastically.

11.9.3 Shortcut Keys

Transitioning from desktop software to web-based applications highlights how often shortcut keys are used in typical work flows. Since so much user input is received by browsers, web applications tend to avoid shortcut keys. In the case of full-screen applications that are attempting to emulate traditional software, this is not a concern because any input received is definitely intended for the program. Thus, a key event handler would be useful for the application.

12 Results & Conclusions

In reviewing the final version of the application, it is clear that a successful multi-user, browser-based painting program was developed. In addition, a number of contributions were made towards advancing the quality and functionality of browser-based applications.

Through the development of the brush tool, the dynamic between paint opacity and flow was explored. Prior to this application, such a feature was only implemented in desktop multimedia software. Properly translating this feature to a web-based application is a true testament to the possibilities of the web browser and helps to reinforce the concept that the internet is no longer limited to just websites.

Similarly, the use of highly-customized cursors, while not a completely new feature, have had little exposure inside web applications. Specifically, all previous painting applications either did not provide a visual representation of the selected brush or limited it to the default round offerings. The implementation of contour tracing here to generate an accurate visualization of the affected canvas areas while painting is a basic feature,
but goes a long way towards improving the user experience.

All previous canvas-based painting applications allowed the user to set a very limited subset of the traditional brush options. Notably, settings such as rotation, roundness, axis-aligned flipping, and spacing have not been publically implemented before. The inclusion of these parameters provides the user with a significantly higher degree of freedom when producing their artwork.

In general, the work done to ensure the application behaves properly in a networked, multi-user environment should be regarded as a step forward in a previously-unexplored area for applications of this nature. Without exception, as reviewed earlier, the canvas element has not been used as a means for providing a multi-user painting experience. Its use here proves that the web has very few technological limitations, even with regard to newer features.

Having pointed out those successes, it should be recalled that this application is not without its flaws. Importantly, the program offers no true eraser tool to the user. This is a key tool used by artists quite frequently when using painting programs, so its implementation should definitely be considered for future revisions. Furthermore, being that the program is geared towards artists familiar with commercial multimedia programs, support for multiple layers and graphics tablets would be ideal.

In summary, then, it can be seen that the HTML5 canvas element serves as a fantastic foundation upon which to develop a painting program. Due to complications with blending when using the 2D context, the WebGL mode is certainly preferred. Furthermore, one can imagine that many additional possibilities arise when working in 3D with access to vertex and fragment shaders. Different blending modes, filters, and other special effects can all be implemented without fear of performance problems due to the sheer power of the GPU. As a result, not only is the canvas element a great basis for a painting program, but also for any type of multimedia application.

Looking ahead to the future of the internet, it is interesting to consider the possibility of traditional software ceasing to exist and everything being run from within a browser. Such a shift is not entirely impossible, as recent trends are starting to show. With increasing numbers of applications being developed to work on smart phones and other mobile devices, the web browser is quickly becoming a mandatory, instead of simply optional, platform to support.

Even with the attention devoted to web applications, little focus has been put on developing multi-user, collaborative programs. Given that these new applications are running in a networked browser environment, it makes perfect sense to link users up so they can interact with one another. As seen with the painting application developed here, the infrastructure to provide a reliable, efficient, and enjoyable multi-user environment exists today. Hopefully as the web evolves and developers continue experimenting to push its boundaries, this somewhat niche area of collaboration tools will expand to the point where users can seamlessly work in tandem with one another to accomplish their common goals.
References


