WebGL Whiteboard Eraser

Michael J. Snyder

Advanced Computer Graphics with Professor Barb Cutler
Rensselaer Polytechnic Institute
May 5, 2011

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 and Facebook Chat, 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 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 build upon a previously-developed real-time, multi-user WebGL painting application by offering a fully-functioning eraser tool. Being that the original version of the application presented a robust painting interface, much of the functionality offered by the brush tool is mimicked by the eraser. Tip customization such as shape, rotation, roundness, and size is possible as well as the management of opacity, flow, and spacing parameters. In combination with the brush tool, the eraser developed here provides users with complete control over the information stored on their canvas.

2 Related 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. Specifically, most current browser-based painting applications do not offer layer support and consequently do not provide a true eraser tool. This lack of an eraser is justifiable in such situations because information can be removed from a single-layer canvas by simply painting with the designated background color. Applications such as Sketchpad [2] and CanvasPaint [1] are two examples of painting programs that take this approach to eraser functionality. Very recently, however, art community site deviantART released an HTML5 canvas painting application called muro [3] that supports both layers and an eraser tool.

The deviantART muro application is a notable exception to other current browser painting programs as it allows users to utilize layers and a true eraser. The layering functionality in muro behaves as would be expected, with layers higher in the stack taking display precedence over layers lower in the stack. Furthermore, blending is performed based on the alpha component of each pixel in each visible layer. The erasing functionality is somewhat limited and only allows the user to specify brush opacity, size, and softness. Important settings such as brush shape, rotation, and roundness as well as more complicated parameters such as flow are omitted. As a result, the user does not have the same flexibility offered when adding data to the canvas as they do when removing it—which is a significant oversight.

Similarly, various open source desktop painting applications have been developed with eraser functionality. In general, these applications, such as GIMP [4] and MyPaint [5], offer relatively robust feature sets and eraser parameters but have the luxury of running outside a web browser. This ability to run natively at the user's desktop drastically improves performance and allows per-pixel computations to be performed on the CPU. Given that the application being extended here necessarily runs in a WebGL environment, per-pixel operations are not possible on the CPU for performance reasons and an alternative approach must be used.

In reviewing currently-available browser painting applications, it can be seen that a fully robust eraser tool does not yet exist. Although various procedures have been implemented to provide basic eraser functionality, these implementations either do not fully utilize WebGL or are based on per-pixel operations on the CPU. Consequently, the eraser tool for this application must approach the issue in a new and performance-friendly manner.

3 Methodology

In order to provide a fully-functional eraser tool, all of the options available when painting must be offered to the user when removing information from the canvas. Specifically, brush size, shape, and orientation as well as the dynamic between flow and opacity had to be implemented and respected. With regard to erasing, these parameters behave identically to their painting counterparts but deal with removing, instead of adding, information from the canvas.

3.1 Eraser Shape

Unlike most other browser-based painting applications, the shape of the tip used for erasing 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.

3.2 Eraser Size

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 eraser 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 removing some amount of information from the canvas.

3.3 Eraser Rotation

In order to rotate the selected tip by an arbitrary amount, a temporary 2D 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 eraser 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 removal of color from the canvas when used as a mask, 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.

3.4 Axis-Aligned Eraser Flipping

The flipping process is very straight-forward. If the user requests an axis flip, the brush pixels are traversed along the desired dimension until the halfway point is reached. At each pixel, the alpha 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.

3.5 Eraser Roundness

The roundness of the eraser 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.

3.6 Eraser Flow

The flow of the eraser 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 removed from the canvas in a single stroke. This means that if the user repeatedly passes over the same location, additional color will be eliminated. In contrast, the opacity parameter would not allow additional color to be removed from the canvas. Interestingly, the flow parameter respects the opacity value. Thus, if, for example, a user set the eraser to 80% opacity and 30% flow, color would be removed until it reached 80% of the source tip alpha value. Figure 1 shows an example of the eraser flow setting.

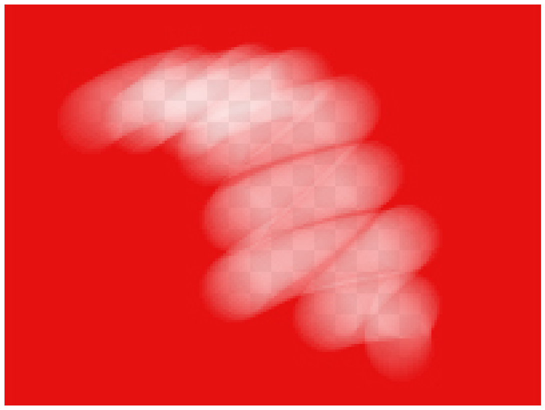


Figure 1: The eraser being applied to the canvas with a 20% flow setting. Note how color is continuously removed from the canvas as the stroke passes over the same pixels multiple times.

In the WebGL mode of the application, the flow parameter for the eraser tool is referenced in a GLSL shader for alpha-scaling purposes. The 2D context mode of the application scales the transformed source tip's alpha values by the selected flow parameter. These procedures allow for the gradual removal of color from the canvas.

3.7 Eraser Opacity

The opacity of the eraser 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 tip by the denoted parameter. In addition, when the user erases from the canvas, the amount of color that can be removed 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 eliminate color up to the limit once again. Figure 2 shows an example of this parameter.

The opacity limit for color removal is enforced similarly in both the WebGL and 2D context modes. For the WebGL mode, the application utilizes a temporary framebuffer that is scaled to the specified opacity limit as the user paints into it. After each application of the tip to the canvas, the framebuffer is subtracted from the base canvas to provide real-time feedback.

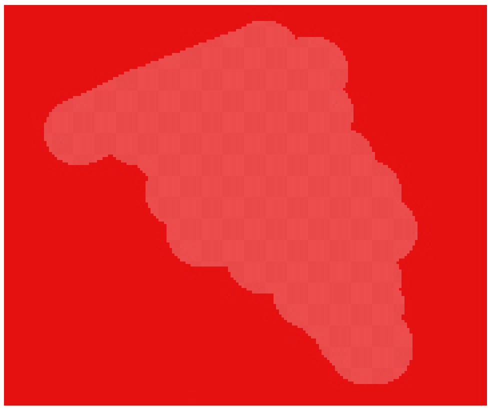


Figure 2: The eraser being applied to the canvas with a 20% opacity setting. Note how the amount of color removed from the canvas never exceeds the limit set by the user.

The 2D context mode is less robust for performance reasons. In the 2D mode, the application treats the eraser as a paintbrush and essentially allows the user to draw over information on the canvas using the background color.

3.8 Process Overview

Unfortunately, the goal of offering a true eraser tool within the 2D context mode of the application was not realistically attainable for performance reasons. When erasing, especially with opacity and flow parameters, the alpha values of the source tip and the base canvas must be observed in order to determine how to modify the base canvas. This process is rather expensive computationally when working at the low level of the 2D context. As a result, the application's 2D mode only offers an eraser that behaves as a background-painting brush.

In contrast, the application's WebGL mode provides access to framebuffers and highly-efficient GLSL shaders. These features make it possible to provide a fully-functioning eraser tool that performs in real-time. The process implemented for the WebGL eraser tool is shown in Figure 3.

The erasing process begins by allowing the user to select and configure their source tip. Using the procedures outlined in the previous sections, the user is able to customize their tip rather extensively. After these settings are applied, the application references the flow parameter when the user begins erasing.

When the user clicks in the application window, the initial state of the base canvas is remembered in an off-screen framebuffer. As the user performs a brush stroke with the eraser across the canvas, the selected brush tip is stamped into a separate off-screen framebuffer where every application is scaled by the flow parameter. Then, for each application of the tip to the off-screen framebuffer, the opacity is scaled to adhere to the user-defined limit and the entire stroke is subtracted from the previously-remembered base canvas. This subtraction process is performed by a custom *glBlendFunc()* where the source value is scaled by *GL_ZERO* and the destination is scaled by *GL_ONE_MINUS_SRC_ALPHA*.

Over the course of a single brush stroke, the user eventually builds up information in the off-screen opacity framebuffer as in the example shown in the last row of the diagram. As with each intermediate step, the opacity of the framebuffer is scaled and then the set of pixels is subtracted from the base canvas. When the user arbitrarily releases the mouse button, the erase stroke is committed to the base canvas and the opacity framebuffer is cleared. This allows for a cyclical cycle whereby the user can remove information from the canvas gradually due to the flow parameter and in a limited fashion due to the opacity value.

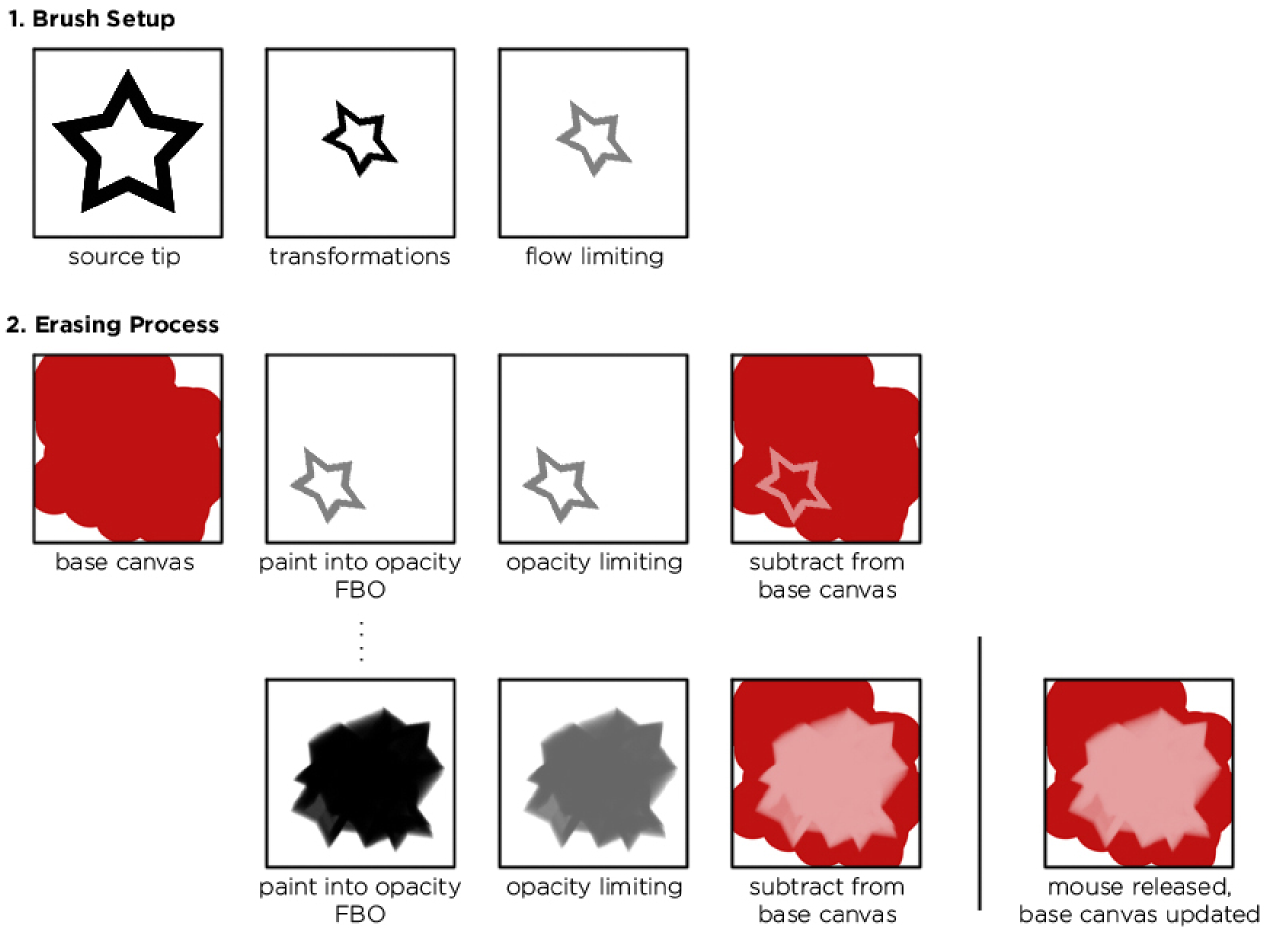
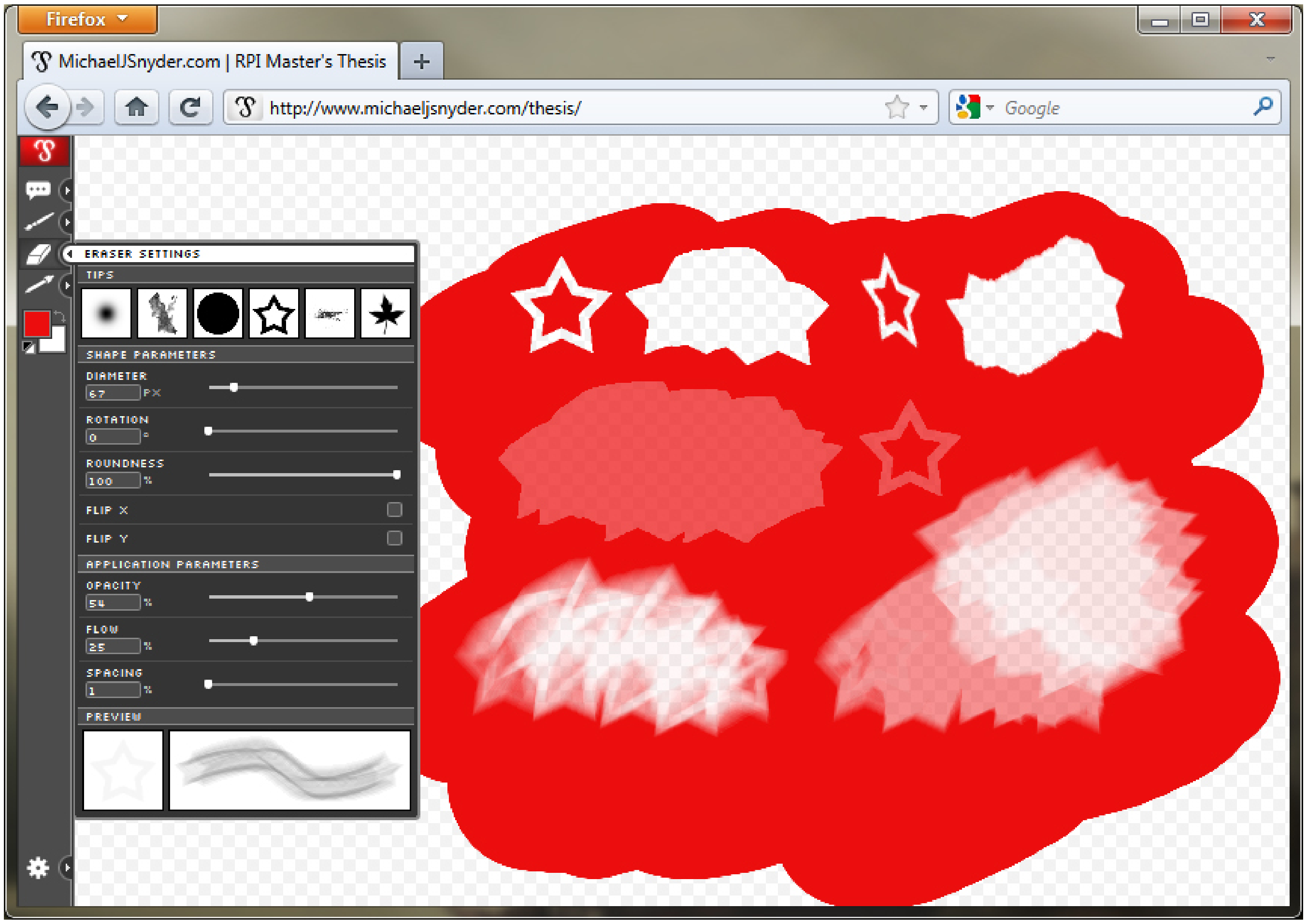


Figure 3: The eraser process implemented for the application's WebGL mode. After the brush has been configured and its alpha levels have been scaled by the flow value, the user's brush stroke is painted into an off-screen opacity buffer. This buffer is then subtracted from the state of the canvas prior to the brush stroke beginning. When the user completes their brush stroke, the base canvas is permanently updated.

4 Results

Over the course of four weeks and roughly 30-40 hours, the eraser tool described here was implemented. Included in that time was a significant effort to provide an eraser that relied solely on GLSL shaders and did not require multiple framebuffers to be maintained. This approach depended upon a single framebuffer that was sampled as the user erased to determine how much additional information, if any, could be removed from the base canvas. Unfortunately, this approach proved unsuccessful due to the inability to extrapolate the amount of information to remove in a given tip application from the amount of information previously removed. It was trivial to determine the total amount of information that could be removed as the value was equal to the difference between the sampled value in the off-screen framebuffer and total coverage of 1.0. The fact that the base canvas was not remembered, however, produced a blending equation that had to be solved for an unknown amount. As a result, the approach discussed here was developed with little to no extra performance penalty.

Similarly, an issue was encountered with the tool during development whereby only the first application of the tip to the canvas would remove any information. All subsequent applications for both the given and future strokes would either not remove information or would


Firefox
MichaelJSnyder.com | RPI Master's Thesis
http://www.michaeljsnyder.com/thesis/
Google

ERASER SETTINGS
TIPS
SHAPE PARAMETERS
DIAMETER 67 PX
ROTATION 0 °
ROUNDNESS 100 %
FLIP X
FLIP Y
APPLICATION PARAMETERS
OPACITY 54 %
FLOW 25 %
SPACING 1 %
PREVIEW

Figure 4: Various eraser settings being used to remove color from the canvas. The shape of the brush is customized and both the opacity and flow parameters are adjusted.

actually paint into the canvas. After careful analysis, it was determined that the blend mode applied to the process was reset to the typical alpha-blending equation when the real-time feedback was provided to the user. By enforcing the custom eraser blend mode when subtracting the off-screen framebuffer from the base canvas, this issue was fixed and the eraser began functioning properly.

As seen in Figure 4, the final eraser tool developed for this application fully performs a wide variety of features. Not only are traditional features such as hard- and soft-tip erasing supported, but more advanced options such as brush rotation and roundness are also available. Furthermore, and most importantly, the advanced customization offered by flow and opacity values is fully-functional.

Of course, these successes only pertain to the application's WebGL mode. As mentioned earlier, the 2D mode is restricted, for performance reasons, to a lesser eraser tool that does not actually remove information from the base canvas. Although this is acceptable for the current state of the application, moving forward it would pose an issue with the implementation of features such as layers and selection tools.

5 Conclusions

In reviewing the extended version of the painting application, it is clear that a successful eraser tool was implemented. Specifically, many features such as eraser shape and roundness as well as respect for the complex parameters of opacity and flow were developed. It is important to take note of these features being that no other currently-available HTML5 canvas painting application offers such robustness.

In looking ahead to future possibilities within the application, the eraser opens the door to many additional features. Most importantly, with a true eraser the application can be designed to offer layer support. Furthermore, rich selection tools can be implemented given that the eraser tool allows for clear definitions between what data is and is not present on the canvas.

In summary, it can be seen that the HTML5 canvas element is an excellent candidate for a multi-user, browser-based painting application. Through the addition of the eraser tool, the base application was enhanced to the point where many core, critical features to a program of this type can now be implemented. This is a significant step forward, as it reinforces the strength of the HTML5 technology and lessens the gap between browser-based software and traditional desktop applications. Hopefully as the web evolves, advancements like this will help the somewhat niche area of collaboration tools to expand to the point where users can seamlessly work in tandem with one another to accomplish their common goals.

References

[1] CLAY, C. CanvasPaint. http://canvaspaint.org/, 2006.

[2] DEAL, M., AND PRITCHARD, C. Sketchpad. http://www.orangehoney.com/studio.html, 2010.

[3] DEVIANTART. Muro - dive in, leave your mark. http://news.deviantart.com/article/125373, August 2010.

[4] GIMP. The GNU Image Manipulation Program. http://www.gimp.org/, 2011.

[5] MYPAINT. Open Source Painting. http://mypaint.intilinux.com/, 2011.