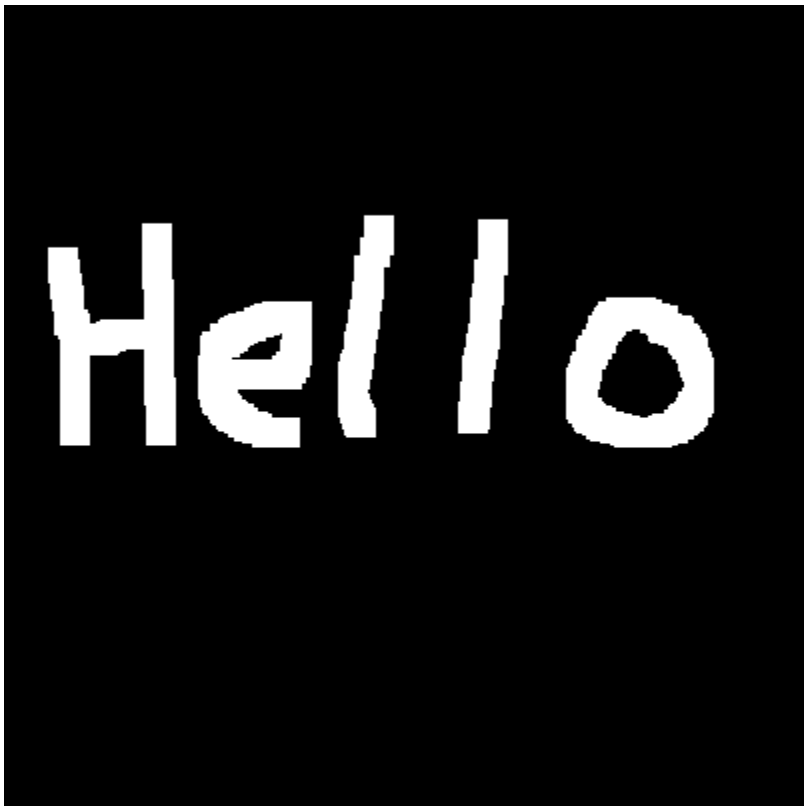
RGB Painting Simulation

Brandon Eng
Computer Engineering
California Polytechnic State University, San Luis Obispo
December 2016

Goal of the Project

The project is making a simulation of painting using an RGB scale. The project focuses on allowing a user to create images by painting on top of a canvas.



User Creating the word "Hello" using the program

This is done using a 400 by 400-pixel canvas. Once the canvas has been modified the canvas cannot be reverted back to its original state. This was in hopes of simulating something similar to realistic painting. This will allow users to practices painting and learning about additive colors. There are six canvases set up to allow a user to test out the various ways he or she could create images.

This was an interesting project to work on and design. The original goal of the project was to make a realistic simulation of painting. This was an interesting idea that I had when I was pitching potential ideas. The number of factors that would apply when attempting to mimic

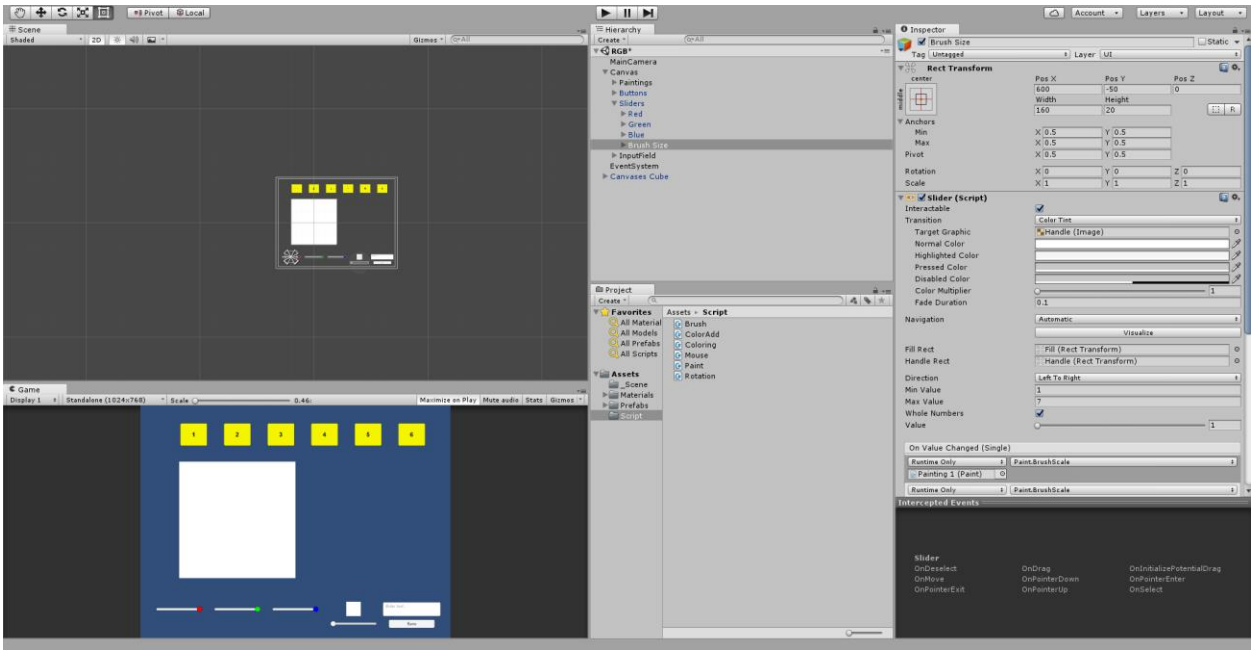
painting with a program was complex. It was a challenging, but fun idea when coming up with the idea. By successfully making a simulation of water color painting it would limit the potential mess when drafting painting. It would allow for a cleaner and faster approach for users to make water paintings and an efficient way to discard paintings that are disliked. There would not be a mess to clean up nor would there be issues with storing the paintings away. There were also several factors that were not taken into account. These factors led to a shrinking in the scope of the project. From the original intent to create a highly complex painting simulation to a smaller version that handles an RGB simulation. This allows users familiarize themselves with the usage of an RGB color scale.

Similar Programs

Some similar programs are the image modification programs. This would include such programs as Microsoft Paint, Adobe Photoshop, and Gimp. These programs also function as image modifying programs. These programs are far more advanced and have more functions. The benefit of my program is the multiple canvases and the cube that displays all of the different canvases. This cube can be rotated around to dynamically show the various changes in the program.

Unity

The design of this program was done in Unity Engine 5. The purpose was the high-level tools that could be used in Unity. The ease of use of Unity allows for an easy implementation of the scripts needed to make a dynamic 2D texture. There were also libraries that could be utilized when implementing the scripts. The use of the way Unity stores away assets, the Hierarchy system, and the inspector allows for fast modifications to the program and adjustments of the various parts.



The Unity Interface

The scene space allows for dynamic changes to what happens in the world that is off screen. This section allows for adjustments in location and size for game objects. The Game section allows for the user to see what would be displayed at the start of simulation. The Hierarchy section allows for a user to organize the system various game objects. This allows for a user to put

similar objects together and to separate out different parts. The section can also be to make a parent and child system. The Inspector portion allows for a user to adjust various values and components of the objects in the scene. When the “play” button is hit, if any values are changed in the Inspector during the simulations then the values are reverted at the end. There is an option to copy the current values of a component to make sure that any changes could be implemented after the simulation has ended. This allows a user to create a working model, but then make adjustments and not have to worry about issues after adjusting. The last section of the user interface is the Project area. This is where all the assets and resources are located. The section can be divided into various files to separate the different tools. The image displayed shows a scene, material, prefab, and scripts folders. The Scene folder shows the current states of the game and how everything is located. The material folders hold the materials for the objects within the scene. This also includes the textures that are located within the scene. The prefab folders store away saved objects that could be reused later. This would include a canvas, buttons, and sliders that would be replicated. These would contain similar fields and function in similar areas. This allows for an easy way to implement multiple objects that have a similar functionality. The last folder would be the script folder. This folder contains all the script components that were written to implement the various functionality of the simulation. This includes the rotation system and the painting system. It stores away scripts that are written and used. It can also house old scripts in case they need to be used again later.

Unity offers a lot of resources including libraries and premade assets. This includes the canvas, image, buttons, and sliders. This made the implementation of the tools easier. There are also premade shapes stored away to as well.

Technical Challenges

The main challenges at the start was to try to find material on realistic painting. This provided issues due to the complexity of replicating realistic painting. There were far more factors to be considered than I anticipated. After doing the research the project was alter scaled down to a more manageable implementation.

The goal was to get an RGB paint simulation implemented. This was to show varying changes in RGB and how it functions. The main goal was trying to figure how to implement a dynamic texture that could change with various mouse clicks. After getting a dynamic 2D texture set up, it had to be respond to clicks and modify the color of the location.

The next issue was to implement multiple canvases and a 3D view of the canvases. The 3D view would be a cube that would hold the current states of all other canvases. Then implementation of variable colors and brush size.

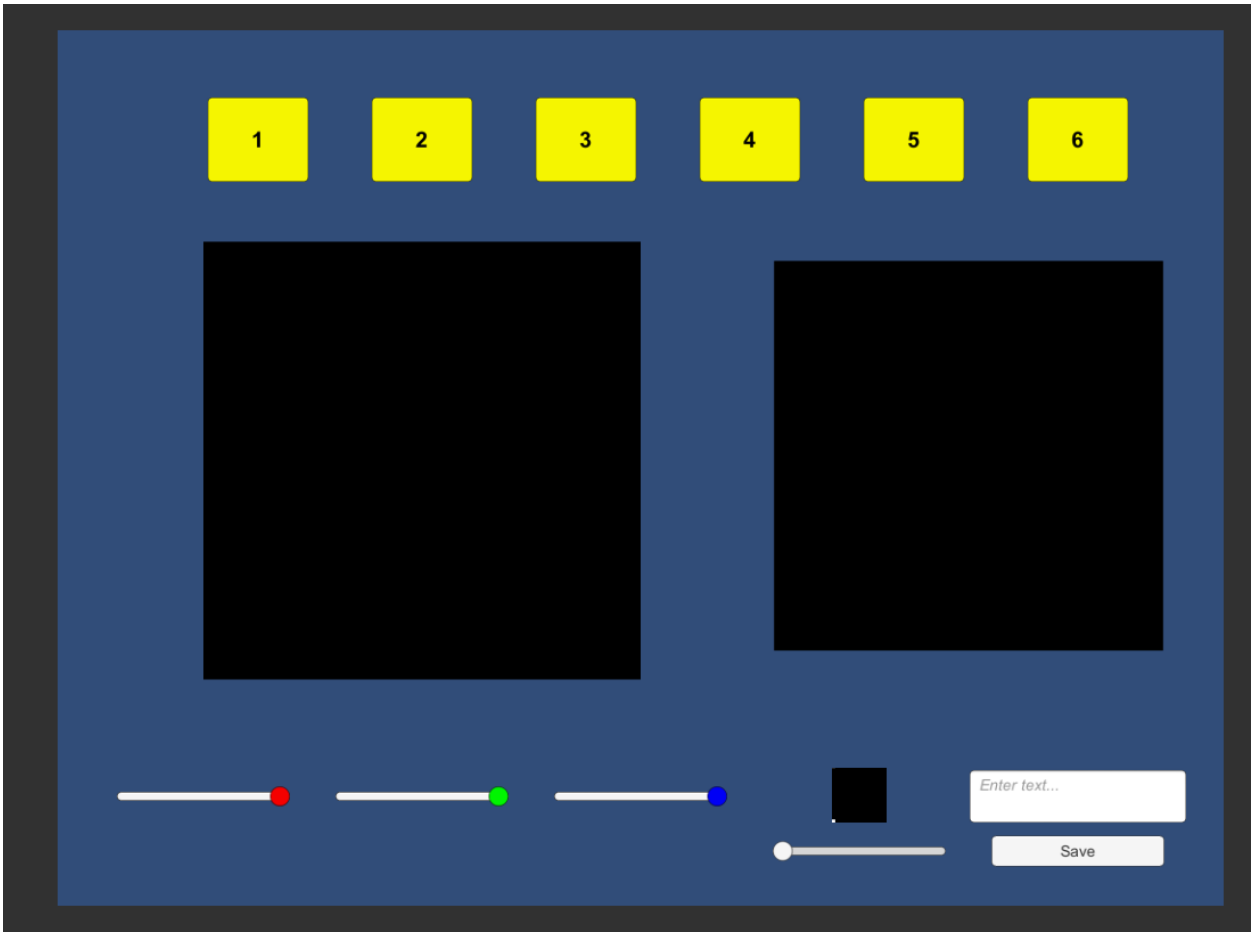
The 2D texture was used to form a dynamic image. The modification was done using the libraries within Unity to modify the specific pixels within the Texture 2D. This was done using the SetPixel method. There were issues with getting the correct mouse location relative to where the image was. To resolve the issue the camera was changed from a perspective view to an orthographic projection. This allowed for the mouse to nearly coincide with the specific locations on the image. There had to be minor adjustments to the mouse due to offset.

The 3D view was done using multiple quads. The Unity Engine did not support modification of the various faces of a cube. To resolve this, I used a set of six quads to imitate a cube. Then made the texture attached to the quad output the source from a camera. Each of the canvases have an attached camera.

The implementation of the buttons and sliders was simple. There were built in buttons and sliders already set up by Unity. These would then be able to execute methods of another object within the simulation. The buttons would move the various canvases away and set them to inactive. The button clicked would be moved into view and set to active. The active allowed for the user to draw. The sliders adjusted the various values for the red, green, blue, and brush size.

Final Results

The final result was a functioning RGB simulator that would output the current image to a PNG file that the user can name.



Buttons

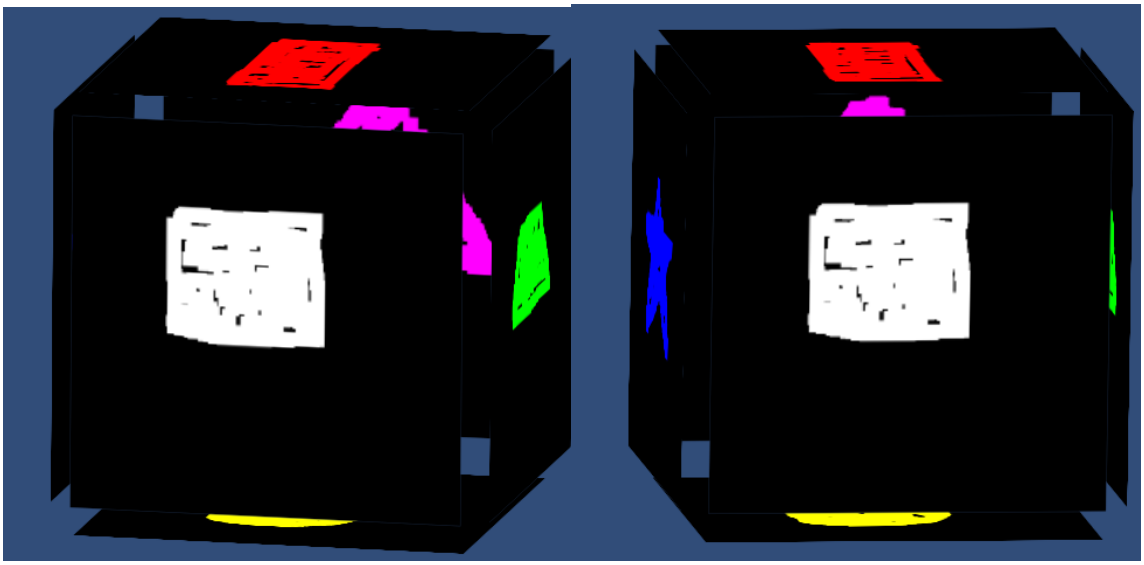
The buttons on the top shows the number of canvases that the user has access to. The way they function is to adjust the position of the canvas to another location and moved the active canvas

into view. The canvas that are moved away are set to inactive so they cannot be modified while off screen by accident.

Canvas

The image on the left is the main painting canvas. This canvas uses the origin of the world (0, 0) as the start point. This allows for easy manipulation on what the user wants to change. The user can adjust the size of the brush with the slider. The modifications would be reflected on the location on the cube. The canvas is set to black as default to the way that RGB works. RGB is additive coloring, which implies that it would slowly add colors (red, green, blue) until it fully mixes and becomes white. The black is when RGB values are all set to zero. This provides the blank canvas instead of the traditional white canvas.

Cube



The cube is the image on the right. The cube can be rotated if the user hits the “Q” key on the keyboard. This is in case the user accidently reaches the cube when painting. The cube can reset its rotation by hitting the button “R” on the keyboard. The cube’s faces correspond to each of the following faces.

These are the corresponding faces to the canvases:

Front – Canvas 1
Bottom – Canvas 2
Back – Canvas 3
Top – Canvas 4
Left – Canvas 5
Right – Canvas 6

The cube faces would change as the user modifies the canvas on the left. All the faces are shown in what the current states they are in.

RGB Sliders

These are the RGB sliders. The adjustment part shows the color that each of the slider modifies. This allows users to easily adjust the color to what he or she might want to use. The scale of the colors go from zero to one. This is to use the Unity Color implementation that stores away RGB values as floats. When the user clicks on the canvas it would apply the current color onto the location of where he or she clicked. If the user continues to overlay multiple colors over one another it would eventually form the color white. After the color becomes white then it would no longer be able to change into another color. This is done so that any changes onto the canvas would be considered permanent and the user can adjust their painting or start over. This follows similar to mistakes made on a normal canvas and reflects the need that any mistakes would stay and the user would have to adjust.

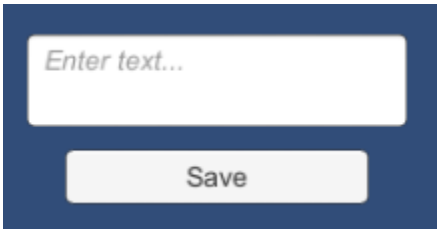
Brush Scale

This slider remains white to represent the current size of the brush. The scale goes from a one by one pixel block up to a seven by seven-pixel block. The user can adjust the size of the brush until he or she finds a suitable size that he or she wants to use.

Brush Example

This is an image that shows what the current state of the brush is. The bottom right shows what the relative size and color of the brush. When you click on the main canvas it will use the same size block and color as the one present in this example. This allow users to know how large the brush they are using is and what the current color of the brush is.

Save Image



This entry box and Save button will save the currently active image into a PNG. The title of the image can be entered in by the current user. After hitting save it will output the image into a PNG file in the assets folder. This allows a user to save a piece of art that they have been working on.

Limitations

The current program still has limitation on the size of the brush. The brush is also limited to only a square and no other shape. The program can also not read in images and allow a user to modify them. There are also limitations on the update rate of the program. Potentially if the update takes too long to execute the program will not be able to modify the color of the canvas. The canvas is also set too only a 400 by 400-pixel image. The program is also limited to using RGB and no other color systems.

There are also issues with mouse drag where the update would instantly push the set colors to their maximum value. To avoid this issue instead of allowing drag the user can make single clicks to adjust the colors.

I would have liked to make a realistic or semi-realistic way to simulate painting. This would have been interesting concept. The factors such as time, material, heat, and humidity would be factors when trying to replicate it. There would also be the changes of color between wet and dry paint. Personally, I wanted to have a nice background with a 3D plane where a user could paint a surround such as walls or a house.

Future Work

In the future, I could potentially expand the different color implantation to include others. This would include CMYK, HSV, and XYZ. Adding features to be able to increase brush size further and change the shape of the brush would be interesting. The ability to load and modify images would allow users to take existing work and modify it to their liking.

Conclusion

This was a very interesting and complex project to work on. The idea of making a simulation was intriguing and could save resources for painters. The current one could be used as a basic learning tools for users who want to draw or learn about the RGB color system. This would be a basic tool for early education as well to let them learn more about colors within a computer and let them play around with art. This could project could be potentially expanded and further worked on. A user could take the time to learn more about how paint functions and implement a proper method to simulate paint. The user should probably look into the various research done to replicate water and oil color painting as a reference point.

References

"Unity Tutorial: Render Textures." *YouTube*. N.p., 08 Mar. 2015. Web. Oct. 2016. <https://youtu.be/pA7ZC8owaeo>.

"Unity3D Tutorial - Simple Texture2D Generation." *YouTube*. N.p., 18 Jan. 2016. Web. Oct. 2016. <https://youtu.be/PR3ndGRkl_Y>.