3D Pixel Art Renderer

Alexander D. Abel
Alexander.Abel001@umb.edu
University of Massachusetts Boston

Figure 1: Final project for CS460.

ABSTRACT

The purpose of this project was to create a simple website that would take a 2D low-res image (henceforth referred to as a sprite) and render it in Three.js using three-dimensional cubes. The image should be uploaded via a Dat.GUI user interface and rendered onscreen appropriately.

KEYWORDS

WebGL, Visualization, Three.js

ACM Reference Format:
Alexander D. Abel. 2019. 3D Pixel Art Renderer. In *CS460: Computer Graphics at UMass Boston, Fall 2019.* Boston, MA, USA, 3 pages. https://CS460.org

1 INTRODUCTION

Rendering 2D Pixel Art within a 3D Space, using individual cube objects within a scene, while of no probable practical use, makes for a good tech demo showcasing the capabilities of both WebGL and the Three.js framework that lies on top of it. Having a base JavaScript object laying down the foundations for implementing such a demo within a Web Browser opens up the doors for all sorts of different possibilities for tech demos, interactive websites, and the like. This project sought out to create such a base renderer for 2D Pixel Art, and implement it accordingly.

2 RELATED WORK

This project will be using Three.js as the primary framework for which our pixel art-loading object will be built off of. The JavaScript Load Image library was also used here, for the purpose of simplifying the translation of 2D pixel data into a 3D cube array.

3 METHOD

This project's primary focus was to implement a 3D renderer for 2D pixel art, in the form of a class that can be used in any given Three.js. Support for transparency (pixels with an RGB value of (255, 0, 255) are not rendered) was planned from the start. Some stretch goals for implementation were animated sprites and palette editing, however, unfortunately, these goals did not pan out in the end due to time constraints.

3.1 Implementation

Most of the code for actually translating 2D pixel data into a 2D cube group is done in Sprite3D.js, which contains various constructors and attributes for the class.

The HTML file used for the purposes of this demo was modified from Homework 8 for the class. The robot and plane code was removed, and leftover skybox code was kept. Two new document elements were added, an input element and a canvas element, one for reading files uploaded by the user, and another for rendering the image to a canvas so that it could be interpreted by any given Sprite3D object.

```
<input id="myInput" type="file" style="visibility:hidden" />
  <canvas id="imageDump" style="display:none" />
```

Event listeners were also added to ensure that Sprite3D creation did not occur prematurely (i.e. before the image data had been properly loaded and drawn to the canvas).

```
    // add listener so that sprite is
    created once the file has been
    uploaded
document.getElementById("myInput").addEven
tListener("change", function () {
    fileRef =
    document.getElementById("myInput").fil
    es[0];
    loadImage(
        fileRef,
        function(img, data) {
            ctx.drawImage(img, 0, 0);

            st = new Sprite3D(fileRef, 5.,
            1);
            st.loadPixelArray(scene, ctx,
            data.originalWidth,
            data.originalHeight);
        },
        { }
    );


});
```

Once the image had been properly drawn to the canvas, the Sprite3D object loads the pixel array from the canvas and creates an array of cubes in a group, each with an individual material with a color that corresponds to the RGB value on the sprite. Once all pixels are accounted for, the box group is loaded into the scene.

```
Sprite3D.prototype.loadPixelArray =
function(scene, ctx, width, height) {
console.log(width, height);
boxGeo = new
THREE.BoxGeometry(this.pixelSize,
this.pixelSize, this.pixelSize);

for (var x = 0; x < width; x++) {
    for (var y = 0; y < height; y++) {
        color = ctx.getImageData(x, y, 1,
        1).data;
        console.log(color);

        // transparent color
        if (color[0] == 255 && color[2] ==
        255)
            continue;

        colorHex = color[0]
        colorHex = (colorHex << 8) +
        color[1];
        colorHex = (colorHex << 8) +
        color[2];

        var bx = new THREE.Mesh(
            boxGeo, new
            THREE.MeshStandardMaterial({
                "color" : colorHex
            })
        );

        bxx = (x * this.pixelSize) + (x *
        this.pixelDistance);
bxy = (y * this.pixelSize) + (y *
this.pixelDistance);
bx.position.set(bxx, bxy, 0);

this.pixelGroup.add(bx);
        }
    }

    this.pixelGroup.rotateZ(Math.PI);

    scene.add(this.pixelGroup);
}
```

3.2 Milestones

How did you structure the development?

3.2.1 Milestone 1. First, I experimented with object groups in Three.js, and added a dummy function that created an array of blank (white) pixels, just to test whether or not things would work properly with a group.

3.2.2 Milestone 2. File input was implemented. This part was particularly tricky, as much file reading in HTML/JavaScript is done asynchronously. To be sure that the image was loading properly, the canvas was made visible so that its contents could properly be displayed.

3.2.3 Milestone 3. The actual pixel renderer was implemented. With the canvas data containing the image we wanted, all that was left was to read the RGB values from each location on the image and position the cubes accordingly, which was done with two nested for loops.

3.3 Challenges

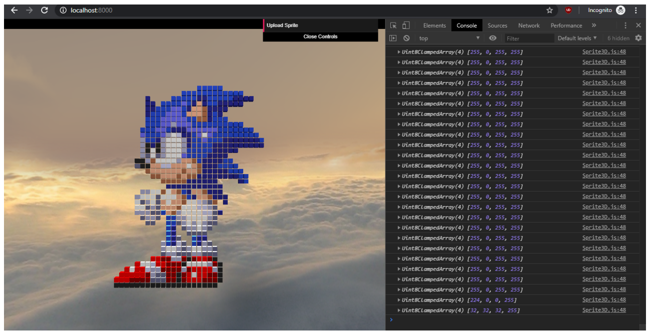
Challenges that were an issue during development:

- As mentioned earlier, file input in HTML/JavaScript was a bit of a problem child, due to its asynchronous nature. At first, writing a BMP parser from scratch was considered, however, in the end, the decision to use an external library to read an image and write to the canvas was made.
- Getting the RGB values correct was a bit of an issue, as the 3 individual values needed to be merged together into one integer for material creation. Refer to Figure 2 for improper color behavior. This, however, was solved after some searching on StackOverflow.

- Having the website read to/write from the canvas turned out to be problematic in Firefox, which seems to have protections against websites doing such things. As a result, the website only properly works in Chrome, at least for the time being.

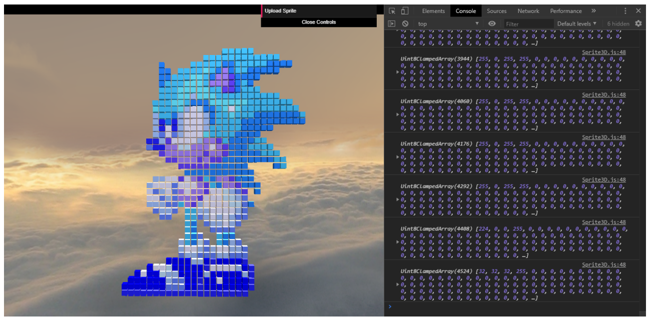
4 RESULTS

While features like increasing/decreasing the distance between individual pixels, palette swapping, and sprite animation could not be implemented due to time constraints, the base renderer was completed, with support for transparency on pixels with an RGB value of (255, 0, 255). Refer to Figure 2.



```
▸ Uint8ClampedArray(4) [255, 0, 255, 255]          Sprite3D.js:48
▸ Uint8ClampedArray(4) [255, 0, 255, 255]          Sprite3D.js:48
▸ Uint8ClampedArray(4) [255, 0, 255, 255]          Sprite3D.js:48
▸ Uint8ClampedArray(4) [255, 0, 255, 255]          Sprite3D.js:48
▸ Uint8ClampedArray(4) [255, 0, 255, 255]          Sprite3D.js:48
▸ Uint8ClampedArray(4) [255, 0, 255, 255]          Sprite3D.js:48
▸ Uint8ClampedArray(4) [255, 0, 255, 255]          Sprite3D.js:48
▸ Uint8ClampedArray(4) [255, 0, 255, 255]          Sprite3D.js:48
▸ Uint8ClampedArray(4) [255, 0, 255, 255]          Sprite3D.js:48
▸ Uint8ClampedArray(4) [255, 0, 255, 255]          Sprite3D.js:48
▸ Uint8ClampedArray(4) [255, 0, 255, 255]          Sprite3D.js:48
▸ Uint8ClampedArray(4) [255, 0, 255, 255]          Sprite3D.js:48
▸ Uint8ClampedArray(4) [255, 0, 255, 255]          Sprite3D.js:48
▸ Uint8ClampedArray(4) [255, 0, 255, 255]          Sprite3D.js:48
▸ Uint8ClampedArray(4) [255, 0, 255, 255]          Sprite3D.js:48
▸ Uint8ClampedArray(4) [255, 0, 255, 255]          Sprite3D.js:48
▸ Uint8ClampedArray(4) [255, 0, 255, 255]          Sprite3D.js:48
▸ Uint8ClampedArray(4) [255, 0, 255, 255]          Sprite3D.js:48
▸ Uint8ClampedArray(4) [255, 0, 255, 255]          Sprite3D.js:48
▸ Uint8ClampedArray(4) [255, 0, 255, 255]          Sprite3D.js:48
▸ Uint8ClampedArray(4) [255, 0, 255, 255]          Sprite3D.js:48
▸ Uint8ClampedArray(4) [255, 0, 255, 255]          Sprite3D.js:48
▸ Uint8ClampedArray(4) [255, 0, 255, 255]          Sprite3D.js:48
▸ Uint8ClampedArray(4) [255, 0, 255, 255]          Sprite3D.js:48
▸ Uint8ClampedArray(4) [224, 0, 0, 255]            Sprite3D.js:48
▸ Uint8ClampedArray(4) [32, 32, 32, 255]           Sprite3D.js:48
>
```

Figure 2: The renderer working with a sprite from Sonic 3.



```
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …]
                                                              Sprite3D.js:48
  Uint8ClampedArray(3944) [255, 0, 255, 255, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
▸ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …]
                                                              Sprite3D.js:48
  Uint8ClampedArray(4060) [255, 0, 255, 255, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
▸ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …]
                                                              Sprite3D.js:48
  Uint8ClampedArray(4176) [255, 0, 255, 255, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
▸ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …]
                                                              Sprite3D.js:48
  Uint8ClampedArray(4292) [255, 0, 255, 255, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
▸ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …]
                                                              Sprite3D.js:48
  Uint8ClampedArray(4408) [224, 0, 0, 255, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
▸ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …]
                                                              Sprite3D.js:48
  Uint8ClampedArray(4524) [32, 32, 32, 255, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
▸ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …]
>
```

Figure 3: An example of incorrect color behavior.

5 CONCLUSIONS

While not all features that ideally would've been implemented could have within the time frame, the base renderer was completed. While the object is a bit reliant on existing code in any given HTML file, as long as a canvas with the image data is found, the sprite can be rendered in 3D. As far as creating a base for other Three.js pixel art projects to build off of, I'd say this project was a success.

REFERENCES