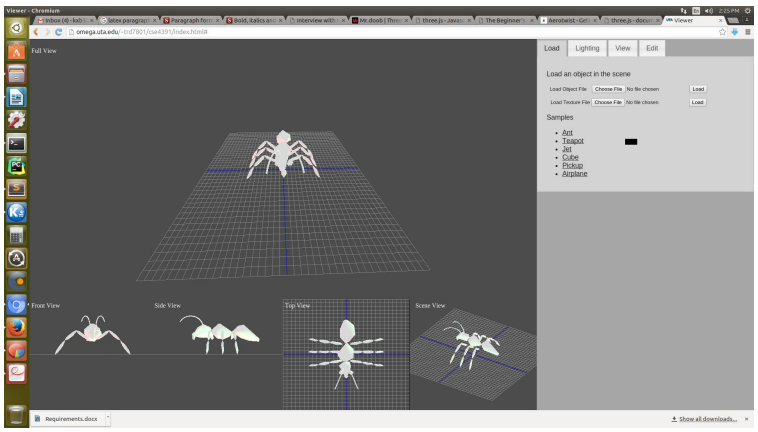
Department of Computer Science & Engineering
The University of Texas at Arlington

Applications of Computer Graphics in the Web
CSE 4314
Spring 2016



Computer Graphics using the Web

Krishna Bhattarai, Tyler D'Spain

Supervising Professor: Dr. Farhad Kamangar

Revision History

Revision	Date	Author(s)	Description
0.1	04.10.2016	KB	Created Document
0.1	04.11.2016	KB	Added Introduction & Background
0.2	04.13.2016	TS	Started Requirements Overview
0.3	04.20.2016	TS, KB	Added Findings
1.0	04.23.2016	KB	Edited Conclusion
1.1	04.26.2015	TS, KB	Updated Final Findings and Figures

CONTENTS

1 INTRODUCTION

The purpose of this project was to learn and apply advanced Computer Graphics techniques in the web. The web medium was chosen because of the popularity of JavaScript and we also wanted to create an application that could simply run on a browser without any cross platform compatibility issues. Technologies that were used are WebGL, JavaScript, and HTML5. These are briefly described here:

1.1 WEBGL

According to Khronos, the organization that manages WebGL, it is a cross-platform web standard for a low-level 3D graphics API based on OpenGL ES 2.0 exposed through the HTML5 Canvas element as Document Object Model(DOM) Interfaces. Its semantics is similar to OpenGL ES 2.0 API and developers familiar with it recognize WebGL as a Shader-based API using GLSL.

1.2 JAVASCRIPT

According to the Mozilla, JavaScript is a cross-platform, object oriented scripting language which is small and lightweight. Inside a host environment like a web browser, JavaScript, can be connected to the objects of its environment to provide programmatic control over them [2].

1.3 HTML5

HTML5 is a World Wide Web Consortium (W3C) specification that defines the fifth major revision of the Hypertext Markup Language (HTML) which was published in October 2014. One of the major changes in HTML5 is in respect to how HTML addresses Web applications [4]. Other new features in HTML5 include specific functions for embedding graphics, audio, video, and interactive documents.

1.4 THREE.JS

Three.js is a cross-browser JavaScript library/API that uses WebGL to render animated 3D computer graphics in a web browser. It has a nice layer of abstraction that allows even new users to quickly apply computer graphics techniques to render objects quickly with little code. Three.js was first started by Ricardo Cabello who is well known in the internet world as Mr. doob. Three.js is as open source project and is continuosly being updated with a dedicated and passionate team of contributors. We initially started our project to investigate just bare WebGL without the help of any APIs such as this. However, after several prototypes we quickly realized that THREE.js provided abstractions that are really worth exploring. So we decided our final product should certainly use it.

2 BACKGROUND

2.1 MISSION

Our mission was to learn various web APIs to render Polygon Meshes on the web platform.

2.2 VISION

Our vision was to develop a web application that could render Polygon Meshes while learning various web APIs at the same time.

2.3 PROJECT METHODOLOGY

Various depth sensing devices such as Google's Tango, or Microsoft's Kinect, produce point cloud data. For this project, our device of choice was the Tango, as we could obtain Polygon Meshes from it rather than just bare point cloud data. This means that we were able to modify its application to produce data containing information like vertices, faces, and colors in the format which is called the .ply format. The ply is called Polygon File Format or Stanford Triangle Format as it was first produced at Stanford University to store 3 dimensional data produced by 3D scanners. A variety of information about a 3D model can be stored using the ply format. This includes vertices, faces, colors, normals, transparency, and texture coordinates. The data produced by the Tango was then analyzed and rendered in a canvas using WebGL.

We also investigated other file formats that were popular in the Computer Graphics world. We learned about the .obj file format, the JSON model, the collada model, the PDBloader etc. However, we choose to work with the .ply file format as it was more widely used and our Tango could easily be programmed to produce data in this format.

We started out our project with simply using plain JavaScript to create a WebGL context and create buffer for vertices and faces and pushing those buffers to the GPU and doing all the hardwork to render 3D models. Our initial intention was to get exposure to shaders and how they work. After we learnt enough about them, we wanted to get a bit more productive and shifted our focus towards using an API to help us stretch our goals further. We explored poltree, and three.js and found the latter to fit our needs quite well. We used PyCharm as our IDE and ocassinally wrote scripts in python until we were comfortable with JavaScript.

Towards the end of the project, we shifted our focus heavily towards developing a simple and attractive Graphical user interface.

3 REQUIREMENTS

3.1 USER INTERFACE

The interface shall have multiple viewports rendering in a single scene. Whenever a user edits any object in the scene it shall update on all viewports respectively. The different viewports will be defined as, the Full View, Front View, Side View, and the Scene View. The interface shall have a tools navigation with a Create, Load, View and Edit option.

3.2 HOSTED

The viewer site shall be hosted on omega.uta.edu/ trd7801/cse4391/index.html It shall load into the viewer page with no objects loaded. The user will then have the option to choose a desired ply file and load it into the application.

3.3 CANVAS VIEWPORTS

The viewer shall have 5 viewports:

- Full View-Perspective. All mouse controls will be handled in the full view. Full view shall allow for orbit controls around the origin of the world coordinates.
- Front View-Orthographic. A view based at eye point z = GRIDMAX
- Side View-Orthographic. A view based at eye point x = GRIDMAX
- Top View-Orthographic. A view based at eye point y = GRIDMAX
- Scene View-Orthographic. A full scene view that rotates around x, z at world coordinates origin

3.4 TOOLS NAVIGATION

The viewer shall have a side tools bar with the following options:

- Load
- Lighting
- View
- Edit

Load shall prompt for a choose file button to request a file to be loaded. It shall also show a list of sample files to load located on the server
Lighting shall be selected to Point light by default. It shall have a selection to select:

- Point
- Directional
- Ambient

Each light may have different options. **Point** shall be able to change the hex value color of the light, the intensity from 0 to 1, and its distance from 0 to GRIDMAX.
Directional shall be able to change the hex value color of the light and intensity from 0 to 1.
Ambient shall be only able to change the hex value color of the light.

Each light shall have a slide bar to adjust its position based on x, y, z. the max values are from -GRIDMAX to +GRIDMAX
View shall list the objects loaded in the scene and be titled Scene Objects. It shall list the objects based upon file name. It shall also have a section for the Lighting and Environment objects. It shall list each object added as Grid being the first, then default lighting, Ambient, Point, Directional
Edit shall only show the objects loaded in the scene to be able to remove.

3.5 Loading an Object

The Viewer shall load into the Load tools tab by default. The will be given the option to load a ply. The load shall be able to generate a wireframe and a bounding box of any supported ply loaded. A list of sample ply files will be given to be loaded.

3.6 Adjusting Light in the Scene

Overview: The Viewer shall on load have three different types of lights in the scene: Point, Directional and Ambient. The Lighting tools tab shall have Point light loaded by default. The effect of changing the color, intensity and distance based on type selected shall effect the lighting object in real time. The Lighting tab shall also have position slide bars for x, y and z and shall go from GRIDMAX to -GRIDMAX. When loaded the values shall be set to 0, 0, 0 respectively

- Point Light Point Light shall be set to Red hex value on load and intensity at 1
- Directional Light Directional Light shall be set to Yellow hex value on load
- Ambient Light Ambient Light shall be set to White hex value on load

3.7 Viewing Objects

The Viewer shall have a list of Objects loaded in the scene. Also it shall have a section for Lighting and Environment. The View tab shall allow for the object loaded to display its visibility, wireframe and bounding box. The Lighting and Environment shall list the objects in the scene and display an option to toggle visibility.

- View Options for Loaded Objects: The loaded object in the View tab shall have three options: Visibility, Wireframe and Bounding Box. When toggling Visible the object will not be displayed in the scene. When toggling Wireframe, only the blue wireframe object becomes visible and the loaded object becomes not visible to only have one object visible in the scene. When toggling Bounding Box, the bounding box that has been computed on load of object will then display a red wireframe cube around the object
- Lighting and Environment: The Grid and default loaded lighting objects shall only have one option: Visibility. When toggling visibility for each, the object will become non visible in the scene. When toggling visible for light, the light object shall not emit light.
- Editing Objects: The Viewer shall display a list of objects loaded in the scene. The object will have the option to remove it from the scene.
- Remove Loaded Objects from Scene: The Edit tab shall have an option to remove the object from the scene. It shall remove the loaded object file, the wireframe and the bounding box associated with that model.

4 System Overview

4.1 How does WebGL work?

WebGL is a bit more complicated than typical web technologies because it is designed to work directly with the graphics card. This low level work is what allows us to rapidly do complex 3D rendering involving lots of calculation. In WebGL, like in most real-time 3D graphics, the triangle is the basic element with which models are drawn. Therefore, the process of drawing in WebGL involves using JavaScript to generate the information that specifies where and how these triangles will be created, and how they will look (colour, shades, textures, etc). This information is then fed to the GPU, which processes it, and returns a view of the scene [1]. Luckily, for us this entire process is hidden under a layer of abstraction because of Three.js.

4.2 How does Three.js work?

To start rendering objects, inside the body of a HTML file we put in our JavaScript file that contains our Three.js code. We also need to link our project with the Three.js or Three.min.js source code. We need three major things. These are: a scene, a camera, and a glcontext termed as renderer so that we can render the scene with the camera [3].

4.3 Creating a Scene

To create a scene in Three.js we simply do the following:

```
var scene = new THREE.Scene();
```

4.4 Creating a Camera

To create a camera in Three.js we simply do the following:

```
var camera = new THREE.PerspectiveCamera( 60, window.innerWidth / window.innerHeight, 5, 2
```

Here, we created a perspective camera object with a field of view of 60 degrees, an aspect ratio of windowWidth/windowHeight and we set up a near and far clipping plane at 5 and 20 respectively.

4.5 Creating a WebGL context or Renderer

To create a renderer in Three.js we simply do the following:

```
var renderer = new THREE.WebGLRenderer();
renderer.setSize( window.innerWidth, window.innerHeight );
document.body.appendChild( renderer.domElement );
```

Here, we created a renderer object, set its size and added it to our HTML's Document Object Model(DOM). This is a canvas element the renderer uses to dsiplay the scene.

4.6 How do we load a model ?

We can either quickly define our own object and specify its properties such as vertices and faces, or in our case, load a pre built object model. Three.js has a collection of methods that load all kinds of file formats that describe 3D models. We also found a loader suitable for our work which was the .ply loader. We then load a ply file and parse its contencts with the ply loader as shown below:

```
var geometry = new THREE.PLYLoader().parse( contents );
```

Here contents is the content of the files we have loaded using FileReader() module. After we have created a geometry based on the contents of the model, we simply add it to our scene by doing:

```
scene.add(object);
```

4.7 Ok, How do we actually do the rendering?

The final step in rendering is creating a rendering loop which is done by the following function.

```
function render() {
requestAnimationFrame( render );
renderer.render( scene, camera );
}
render();
```

This creates the loop that causes the renderer to draw the scene 60 times per second which is more than enought for human eyes.

4.8 Is that all?

The above is a very high level description of the methods that Three.js uses to render objects. Our implementaion is ofcourse a bit more detailed and has lots of other features.

5 Findings and Conclusion

5.1 What did we learn from this project?

5.2 Explored Various WebGL Libraries

We explored various APIs that are out there that provide an abstraction for WebGL. Some of them were **Three.js**, **PhiloGL**, **J3D.js**, **GLGE.js** etc. We decide to go with Three.js as it seems just right for this project.

5.3 HTML5 and CSS

We learnt about various HTML5 elements that we had not used before. We also learnt and used Cascading Style Sheets(CSS) to give our Application a polished look.

5.4 Various 3D Model formats

We learnt about various 3D model formats that were out there that we did not know existed. Some of these were .obj, U3D, .ply, BLEND etc.

5.5 Shaders

We learnt about shaders namely vertex shader and fragment shader. What they are for and how to use them.

5.6 Texture Mapping

We learnt how to do 2D texture mapping on 3D models. We created texture mapping coordinates (UV map) for a cube, a torus, a teapot and various other ply models that we found at Stanford's library. We also investigated how bump mapping works by investigating a work of Mr. Doob which rendered a human head with realistic texture. We even tried to come up with some algorithm that generates UV coordinates utilizing all portions of an image/texture sufficient for each vertex. We quickly realized how naive this approach was. We finally came to the conclusion that it would be very difficult for us to generate a texture for an irregular 3D model with arbitrary number of vertices.
We even watched various videos on youtube that showed us how texture maps are really generated using sophisticated 3D modeling software as Blender, Maya etc.

5.7 Lighting

We learnt how to create and use lights in our scene using Three.js. We explored point light, ambient light, and directional light and applied those into our scene.

REFERENCES

[1] Luz Caballero. An introduction to webgl-part1, 2011.

[2] cvrebert. Introduction-javascript, 2011.

[3] THREE.js TEAM. Creating a scene, 2016.

[4] W3Schools. Html5 introduction, 2015.