WebGL Audio API for 3D Graphics in Three.JS

Igor Fortel
ifortel@gmail.com

Abstract

As a field, web design and development has come a long way in the last 20 or so years. Amongst the biggest leaps that have been made concern 3D graphics. Computer hardware is becoming increasingly powerful and with that increasing computer and graphics processing power. This allows for developers and computer scientists to get very granular about what we are able to see and change from a computer graphics perspective, isolating and manipulating individual pixels. The same advances can be extend to audio processing and usage for web development. Three JS is used in conjunction with the Web Audio API to create a cube of random color, that would scale based on the frequency of the music frame as determined by the analyzer. Future work in this area may use the Audio API to achieve a more real time processing for video as well since audio is obviously a part of video encoding. The possibilities don't stop there, what if we could create a database of real-life audio sounds, and query them to create new and exciting music. This is a very simple stripped down example, however the ideas discussed and the tools used are very powerful.

1. Introduction

Since the creation of computers, technology started with two-dimensional text graphics, very simple renderings of text. The next step in the evolution was two-dimensional graphics, the advent of computer video games propelled computer graphics even further and showed the world that there was even further that we could go. Three dimensional visualization is the latest evolution of Computer Graphics. We live in a 3D world but we have not been able to create a perfectly realistic rendering of the world for with computer graphics technology. To this end, over the last few years, with the increase in computer power and graphics processing power, there has been a growing interest in technologies whose purpose is to present Web 3D scenes in realistic format.

Creating 3D graphics really started to take off in 1995, when HTML gave birth to the first 3D graphics language called VRML (Virtual Reality Modeling Language) using text-based meta-language methods. For almost 15 years, this was all that was available to create web based 3D scenes, however in 2011, the Kronos group developed what is now a standard for web-based computer graphics, WebGL [1].

1.1. WebGL

WebGL is THE standard API for 3D graphics on websites. It gives developers the ability to work with the full power of the computer's rendering hardware in the browser using Javascript. The previous method for developing 3D applications, VRML required developers to use plug-ins or native applications, and ask users to download and install custom software to be able to have a hardware-accelerated 3D-Graphics experience [2]. Essentially, WebGL is a low-level drawng API, that takes arrays of data and a programmed shader, and tell it to draw. It is based on a long-standing graphics API called OpenGL.

Basic Components of a WebGL Application [2]

- Create a canvas element
- Get a drawing context for the canvas
- Set the viewpoint of the camera
- Create buffers that will contain the data that will be rendered (usually some vertices)
- Create matrices to translate vertices from buffer to screen space
- Create shaders to exectute the drawing algorithm
- Set parameters for the shaders
- Draw

WebGL allows developers to take advantage of a few important characteristics of 3D Graphics; the vertex shader, the rasterizer, the fragment shader, frame buffer, and textures. Combining and manipulating these characteristics is how 3D graphics are created.

Now, in a WebGL environment, there also exists a way to essentially read and parse audio data. Modern day HTML5 has two different systems for playing

sounds, the older HTML5 Audio element and the newer Web Audio API. The original HTML5 Audio element was developed specifically for web development and is unfortunately, no broad enough in its functions to be too useful for games. [3].

1.2. WebGL Audio API

Using Javascript to load and play the sounds with this method is exactly the same as the Javascript methods for loading images; an audio object is created, and when it is loaded, there is an event handler to go through a play method and set other parameters like volume. Using this audio element tends to be unreliable and was never fully adopted by browser developers due to its inadequacies. Instead of trying to break through the wall of issues with the HTML audio element, the Web Audio API was developed to grab control of audio on a more granular level mathematically. The API itself is very complex and deep enough that entire books are written on the subject. Consider the API essentially a giant warehouse of audio components that you can use to generate any audio system you can think of. If it can be made in the real world, it can be made using the web audio API. [4] gives a similar example and takes it one step further, mentioning that the API is completely modular. This means that components can be added and removed at any point in the system creation without compatibility issues. Some examples of creations with this API include an analog-style synthesizer or sampler, a 3D holographic music player, a music notation interpreter, a procedure music generator, or a sound visualizer. For the purpose of this paper, a sound visualizer was created and will be explored more in a later section. The point that we are trying to make is that by creating this "tool box" consisting of the building blocks of audio, it is possible to reproduce audio generated by hardware because both are now using the same mathematical principles.

//Loading and playing a sound using this API

Step 1: load the sound file and decode it, this becomes a raw audio file that is called a buffer

Step 2: Append the buffer to audio effects nodes, basically parameters and properties that can be modified and manipulated in the code. For example, one node could be tied to volume, another that would be tied to pitch or bass, etc…

Step 3: Connect the last node in the effects list to the "destination", which is usually the playback device being used.
Step 4: Start the Audio!

These four steps are the basis for loading and playing sound using the Web Audio API in a WebGL application. The way that this API is implemented, for the purposes of this paper is using Three.JS, which is an object-oriented JavaScrpit library for 3D graphics. The basic procedure is to build a scene graph out of three.js objects to represet a 3D world, followed by rendering an image of that world. It's possible to also animate the drawing by modifying properties of the scene graph between frames [3]. Using Three.JS, we can create a 3D Web computer graphics world by creating a few parameters: Scene, Camera, Renderer, the Object3D (Geometry, Material, Objects, scene graph, transformations, lights, as well as image textures and animation if that is needed.

1.3. Three JS Audio API Example

This example comes from [5], and was modified to get a more detailed view of the audio API being used. We start by creating the scene, camera and renderer, and the controls variable. Next we define the geometry, which in this case is will be a cube. We use a single cube in this code, however there is object = new Array() to easily add additional object without needing additional meshes. The cube will be of size 5,5,5 in a Phong material of random color at the point 0,0,0. The random color function will be explained when the function is initialized. Additionally, we add an ending node for the renderer so that there is essentially a stopping point. Finally, for this section, we need to add lights! Without lights, you won't be able to see anything. In this case, a combination of ambient light and point light was used to obtain a more unique metallic color.

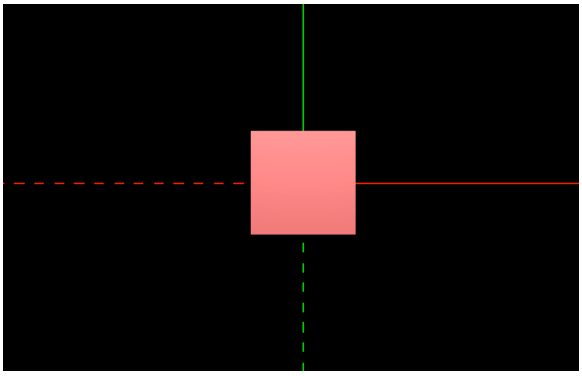

Figure 1. Cube generated in Three.JS

```
var scene = new THREE.Scene();
var camera = new THREE.PerspectiveCamera(50,
$(window).width() / $(window).height(), 1, 1000);
var renderer = new THREE.WebGLRenderer();
var object = new Array();
var controls;

document.body.appendChild(renderer.domElement);
camera.position.z = 50;
controls = new THREE.OrbitControls(camera,

renderer.domElement );
controls.addEventListener('change', render);

var geometry1 = new THREE.CubeGeometry(5,5,5);
var material = new THREE.MeshPhongMaterial({
                        color: randomColor(),
                        ambient: 0x808080,
                        specular: 0xffffff,
                        shininess: 20,
                        reflectivity: 5.5});

object[0] = new Array();
object[0][0] = new THREE.Mesh(geometry1, material);
object[0][0].position = new THREE.Vector3(0, 0, 0);

scene.add(object[0][0]);

var light = new THREE.AmbientLight(0x505050);
scene.add(light);
var pointLight = new THREE.DirectionalLight(0xffffff,
0.7);
            pointLight.position.set(0, 1, 1);
            scene.add(pointLight);
            pointLight = new

THREE.DirectionalLight(0xffffff, 0.7);
            pointLight.position.set(1, 1, 0);
            scene.add(pointLight);
            pointLight = new

THREE.DirectionalLight(0xffffff, 0.7);
            pointLight.position.set(0, -1, -1);
            scene.add(pointLight);
            pointLight = new

THREE.DirectionalLight(0xffffff, 0.7);
            pointLight.position.set(-1, -1, 0);
            scene.add(pointLight);
```

Next we need to set up the render function to bring our variables to life on the screen. In this case the animation formula is also set up such that for all the object, the scale changes based on the frequency of the current frame coming from the music audio file. The render function is also important as it also generates the scene, the camera, and the controls. Also included is the function to generate a random color.

```
var render = function () {

if(typeof array === 'object' && array.length > 0) {
var k = 0;
for(var i = 0; i < object.length; i++) {
            for(var j = 0; j < object[i].length; j++) {
var change = (array[k] + amp) / 30;
            object[i][j].scale.z = (change < 1 ? 1 :
change);            object[i][j].scale.x = (change < 1 ?
1 : change);
            object[i][j].scale.y = (change < 1 ? 1 :
change);
k += (k < array.length ? 1 : 0);}
            requestAnimationFrame(render);
            controls.update();
            renderer.render(scene, camera);};
            render();
            renderer.setSize($(window).width(),
$(window).height());

function randomColor() {
var letters = '0123456789ABCDEF'.split('');
var color = '#';
for (var i = 0; i < 6; i++ ) {
color += letters[Math.floor(Math.random() * 16)];}
return color;}
```

Lastly, we generate the audio data and the usage of it. The API handles audio inside an “audio context”, with modular routing. This gives the developer the ability to create very complex effects using interchangeable layers of audio. There are really four items that are needed to analyze the audio data and render the scene: the audio context, the analyser, the Javascript node, and of course the buffer source. We also need to define a script processor, which is an audio processing node so that we can process the audio in Javascript. Lastly, we define a source buffer which essentially holds the song so that the analyser can decode the source buffer and provide information in real-time so that the scene can be rendered properly. Essentially, everything is ready to go once the src buffer is connect to the analyser, which in turn needs to be connected to the Javascript node and source buffer to the output [audio visualization with web Audio and Three JS]. Once all of that is done, we are ready to go. There are many different parameters that could be added an manipulated further, but these 4 items are the main core of what is needed to add audio to 3D graphics in Three.JS.

In this example, for every frame of audio, we put the frequency of the audio frame into an array to be used by the renderer later. Each frequency is called an “amplitude” value and the visual effects are tied to that amplitude value. Lastly in this example, a few additional features were added such as a Play button, and some fadeout for the music to not come in so abruptly.

```
 // Audio
var context;
var src, srcJs;
var analyser;
var url = 'data/Drive It Like You Stole It -
Glitch_Mob.mp3';
var array = new Array();
var amp = 0;

var interval = window.setInterval(function () {
if ($('#loading_dots').text().length < 3) {
$('#loading_dots').text($('#loading_dots').text() + '.');}
else {$('#loading_dots').text('');}}, 500);
            try {
if (typeof webkitAudioContext === 'function' ||
'webkitAudioContext' in window) {
context = new webkitAudioContext();}
            else {
context = new AudioContext();}}
            catch (e) {$('#info').text('Web Audio API is
not supported in this browser');}

var request = new XMLHttpRequest();
            request.open("GET", url, true);
            request.responseType = "arraybuffer";

request.onload = function () {
context.decodeAudioData(
request.response,

function (buffer) {
if (!buffer) {
$('#info').text('Error decoding file data');
return;

srcJs = context.createScriptProcessor(2048, 1, 1);
srcJs.buffer = buffer;
srcJs.connect(context.destination);

analyser = context.createAnalyser();
analyser.smoothingTimeConstant = 0.6;
analyser.fftSize = 512;

src = context.createBufferSrc();
src.buffer = buffer;
src.loop = true;

src.connect(analyser);
analyser.connect(srcJs);
src.connect(context.destination);
srcJs.onaudioprocess = function (e) {

array = new Uint8Array(analyser.frequencyBinCount);
analyser.getByteFrequencyData(array);
amp = 0;
            for (var i = 0; i < array.length; i++) {
                        amp += array[i];}
                        amp = amp / array.length;
                                                    };
// popup
$('body').append($('<div onclick="play();" id="play"
style="width: ' + $(window).width() + 'px; height: ' +
$(window).height() + 'px;"><div
id="play_link"></div></div>'));

$('#play_link').css('top', ($(window).height() / 2 -
$('#play_link').height() / 2) + 'px');

$('#play_link').css('left', ($(window).width() / 2 -
$('#play_link').width() / 2) + 'px');
$('#play').fadeIn();},
function (error) {

$('#info').text('Decoding error:' + error);};
request.onerror = function () {
$('#info').text('buffer: XHR error');};

request.send();

function displayTime(time) {
if (time < 60) {
return '0:' + (time < 10 ? '0' + time : time);}
else {
var minutes = Math.floor(time / 60);
time -= minutes * 60;
return minutes + ':' + (time < 10 ? '0' + time : time);}

function play() {
$('#play').fadeOut('normal', function () {
$(this).remove();});
src.start(0);}

$(window).resize(function () {
if ($('#play').length === 1)
{$('#play').width($(window).width());$('#play').height($
(window).height());
if ($('#play_link').length === 1) {
$('#play_link').css('top', ($(window).height() / 2 -
$('#play_link').height() / 2) + 'px');
$('#play_link').css('left', ($(window).width() / 2 -
$('#play_link').width() / 2) + 'px');}}});
```

2. Discussion

We can see how simple it is to take an audio mp3 file, decode it, and create 3D graphics based on that audio file. It's easy to see how this can be expanded on to bigger and better ideas. The Audio API is also used in web video games for reat time audio implementation and analysis. Additionally, we've been able to implement a combination of audio file input with a microphone input as well. Adding a listening microphone is a slightly more challenging effect, however it can be done by following many of the same steps mentioned above. The mic becomes the source of the data, however since it is being captured in real time, we need to have a function that not only uses the audio data, but also captures and stores it to be processed. An example of this can be seen below as seen in [6]. On research area that can be explored is to reverse-engineer music using this approach. By creating a database of frequencies and notes, we could potentially create random or analytically generated music using the approach that we have for microphone data collection. We have come extremely far in terms of 3D graphics and audio implementation, but there is still much more room to grow towards an effective and efficient real-time graphics rendering.

```
var $player = $("#player");
$player_source = $player.get(0);
// Creates the analyser

 analyser2=context.createAnalyser(
analyser2.fftSize = 2048;

 frequencyData2 = new
Uint8Array(analyser2.frequencyBi
nCount);

bufferLength =
analyser2.frequencyBinCount;cons
ole.log(bufferLength);

 source_mic=
context.createMediaElementSource
($player_source);

 source_mic.connect(analyser2);


////////////////////////////// Data Array
for capturing//////////////////////////
var $data = [];
var $audioData = []
      function makeArray($data){
if($dataArray == undefined){
var $dataArray = [$data];}
else{
$dataArray.push($data);}
return $dataArray; }

function capture(data) {

source2=
context.createMediaStreamSource(
data);
audio_data = data;
analyser2=context.createAnalyser()
.
// Create the array for the data
values
analyser2.getByteFrequencyData(fr
equencyData2);
var realtimeData = new
Uint8Array(analyser2.frequencyBi
nCount); // Now connect the nodes
together
// Do not connect source node to
destination - to avoid feedback

source2.connect(analyser2);
analyser2.connect(processor);
processor.connect(context.destinati
on);
      }
```

3. References

1. *Stamoulias, Andreas, Eftychia Lakka, and Athanasios G. Malamos. ""Wrapping" X3DOM around Web Audio API."* International Journal of Artificial Intelligence and Interactive Multimedia *3.Regular Issue (2015).*
2. *Parisi, Tony.* Programming 3D Applications with HTML5 and WebGL: 3D Animation and Visualization for Web Pages*. " O'Reilly Media, Inc.", 2014.*
3. *Spuy, Rex van der. "Advanced Game Design with HTML5 and Javascript." (2015).*
4. *GraphicsNotes 2013 -- Section 15: Introduction to Three.js. (n.d.). Retrieved December 14, 2015, from http://math.hws.edu/eck/cs424/notes2013/15_Threejs_Intro.html*
5. *Experimenting with Web Audio API Three.js (WebGL). (n.d.). Retrieved December 14, 2015, from http://srchea.com/experimenting-with-web-audio-api-three-js-webgl*
6. (n.d.). Retrieved December 14, 2015, from https://gist.github.com/webapprentice/8222832.js