Brian Coughlin
...@gmu.edu

CS551 Summer 2013 Project
WebGL on iOS devices - Insights and Experiments

WebGL Background

WebGL is effectively a Javascript API based on OpenGL ES 2.0, where the target viewing plane is an HTML5 canvas. For developers already familiar with OpenGL, they typically think of WebGL as "OpenGL ES in a web browser". For users, WebGL means that more advanced 3D graphics are possible from a website or web application, and without requiring any additional plugins to extend the web browser.

WebGL is managed and maintained by the Khronos Group, which is an industry consortium that develops and maintains standards for the authoring and acceleration of parallel computing, graphics and dynamic media on a wide variety of platforms and devices. For application developers, WebGL is a cross-platform, royalty-free standard API platform for 3D computer graphics.

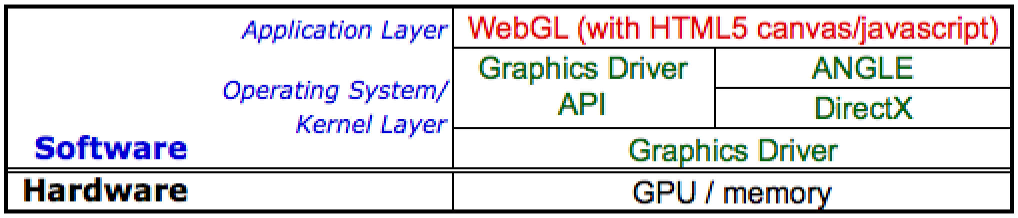
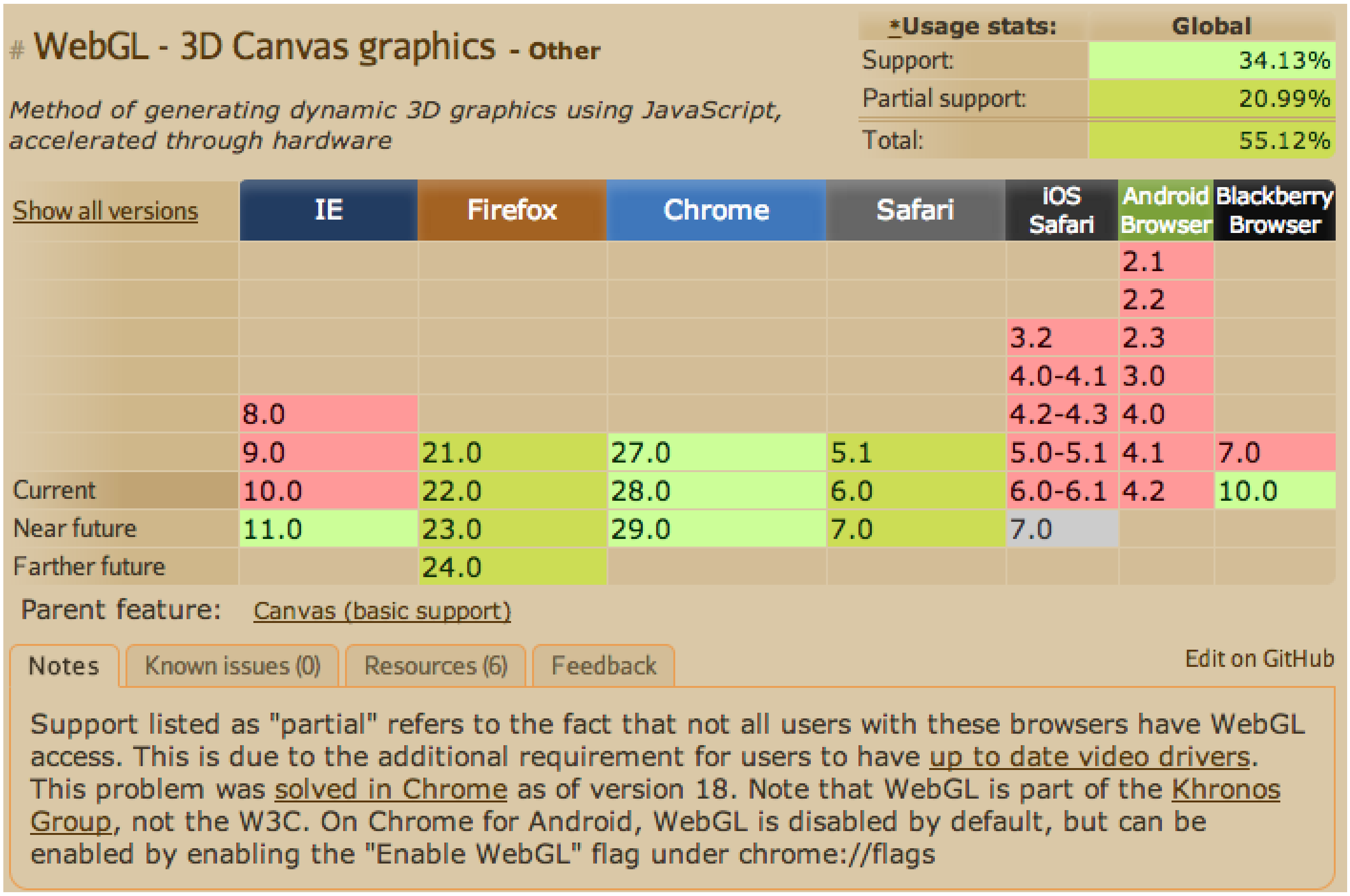


Figure 1: WebGL software stack

With its 1.0 Specification only recently released as of March 2011 [ref5], WebGL is still quite a young as a standard. As such, it is not yet implemented on all major web browsers running on the major operating system platforms. Most recently in this summer of 2013, Microsoft announced it was adding WebGL support to Internet Explorer version 11 which is an important 'tipping point' in getting WebGL implemented at least partially across all major vendors of PC web browsers.



WebGL - 3D Canvas graphics - Other

Method of generating dynamic 3D graphics using JavaScript, accelerated through hardware

*Usage stats:	Global
Support:	34.13%
Partial support:	20.99%
Total:	55.12%

Show all versions	IE	Firefox	Chrome	Safari	iOS Safari	Android Browser	Blackberry Browser
						2.1	
						2.2	
					3.2	2.3	
					4.0-4.1	3.0	
	8.0				4.2-4.3	4.0	
	9.0	21.0	27.0	5.1	5.0-5.1	4.1	7.0
Current	10.0	22.0	28.0	6.0	6.0-6.1	4.2	10.0
Near future	11.0	23.0	29.0	7.0	7.0		
Farther future		24.0					

Parent feature: Canvas (basic support)

Notes | Known issues (0) | Resources (6) | Feedback | Edit on GitHub

Support listed as "partial" refers to the fact that not all users with these browsers have WebGL access. This is due to the additional requirement for users to have up to date video drivers. This problem was solved in Chrome as of version 18. Note that WebGL is part of the Khronos Group, not the W3C. On Chrome for Android, WebGL is disabled by default, but can be enabled by enabling the "Enable WebGL" flag under chrome://flags

Figure 2: WebGL Implementations [ref1]

However what is still lacking is support for WebGL on the majority of default browsers on mobile device platforms, the lone exceptions being the browser on Blackberry 10, and the beta version of Chrome on Android which must be installed by the user because it is not the stock browser included with Android. This is ironic despite given that OpenGL ES 2.0 itself is supported on iOS and Android mobile devices otherwise. So even having support on PC browsers is not optimal for truly widespread adoption of WebGL because new iOS and Android devices are now outselling new PCs on a per-unit basis. Furthermore future projections show mobile device sales increasing while PC sales are expected to remain flat at best.

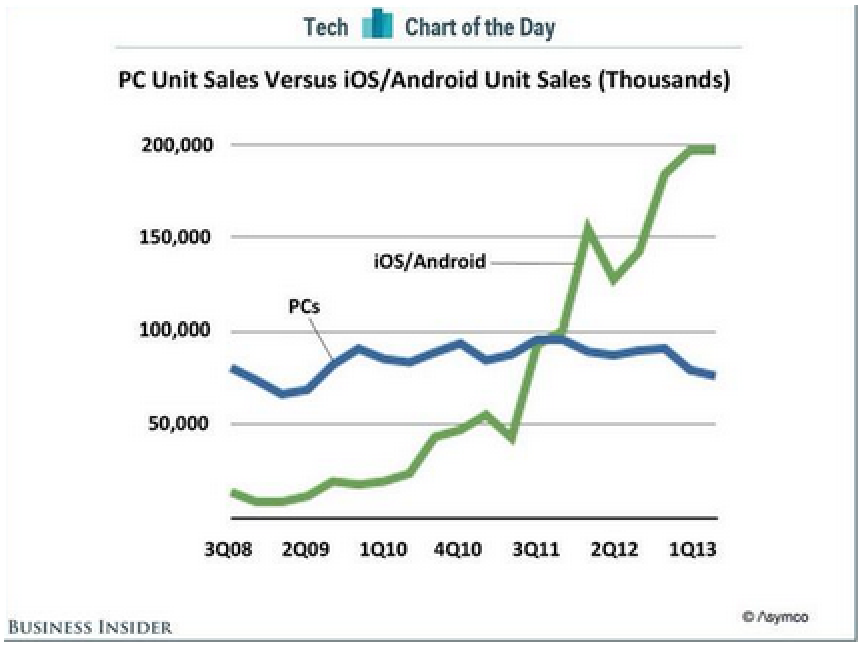
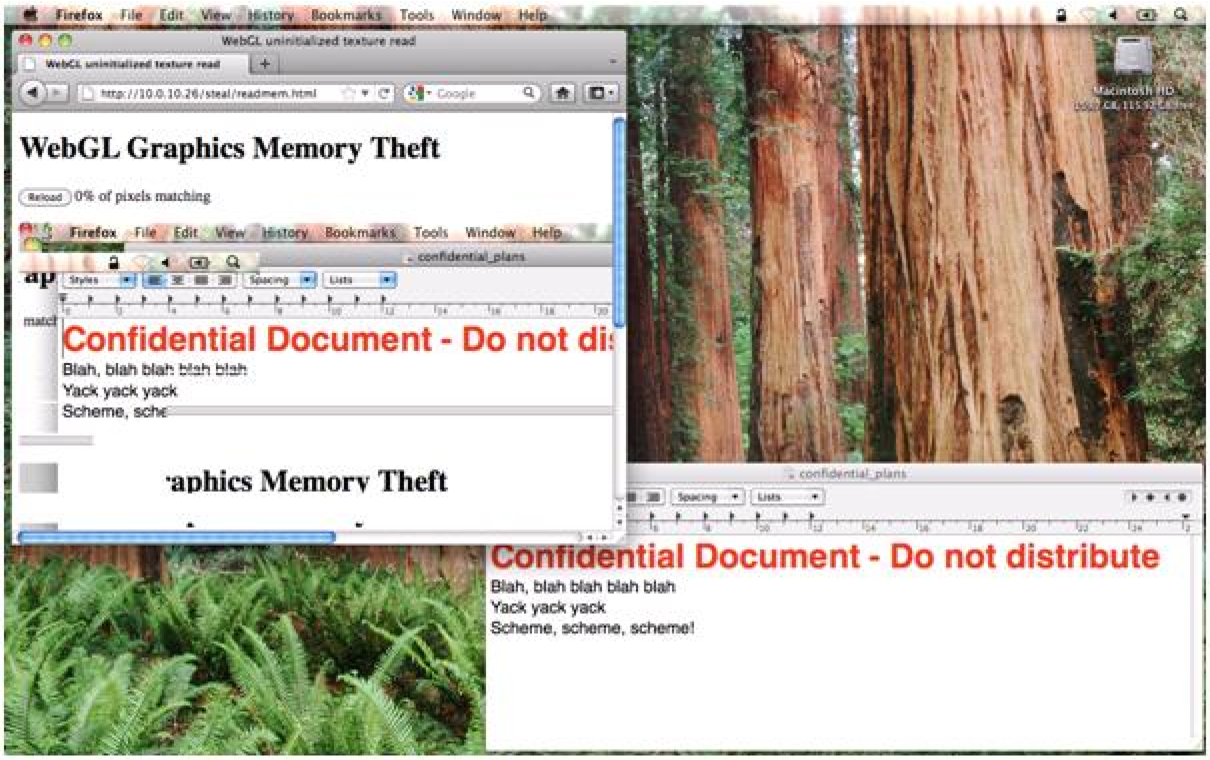


Figure 3: Unit Sales: PCs via iOS/Android devices

iOS and WebGL

For iOS specifically, Apple has thus far expressed no intentions of enabling WebGL in Mobile Safari or its iOS WebKit framework used by any web browser on iOS. This position remains unchanged even in pre-release versions of iOS 7 available for developers today but not otherwise generally released to the public. This appears to be mostly for non-technical reasons, because in fact WebGL is enabled in Apple's iAd platform for iOS.

It is also possible that Apple's lack of WebGL support in mobile safari may also be for security reasons, in that security flaws had been found in some implementations of WebGL where application memory in the GPU from other software could be hijacked by WebGL code running in a browser.


Firefox File Edit View History Bookmarks Tools Window Help
WebGL uninitialized texture read
http://10.0.10.26/steal/readmem.html
Reload
confidential_plans
Styles Spacing Lists
Macintosh HD

WebGL Graphics Memory Theft

0% of pixels matching

Confidential Document - Do not distribute

Blah, blah blah blah blah

Yack yack yack

Scheme, scheme, scheme!

Figure 4: WebGL memory theft

This initial security flaw was addressed by the Khronos group by the version 1.0.1 and 1.0.2 updates to the WebGL Specification.[ref2]

Introducing Ejecta

So despite these limitations on iOS this paper introduces Ejecta [ref3], a application container technology that provides a separate HTML5 canvas implementation with WebGL support. What is novel about Ejecta is that it is not a web browser but rather a WebGL container written in Objective-C. Ejecta implements an HTML5 canvas and other minimal functions to allow WebGL applications written purely in javascript, where Ejecta along with a developer's own javascript WebGL code is packaged and deployed as a standard iOS application. For detailed documentation of all the functions supported on Ejecta, see the Ejecta website for details on the current release, currently version 1.3.

Per section 2.17 of Apple's App Store Review Guidelines, "iOS applications that browse the web must use the iOS WebKit framework and WebKit Javascript". So Ejecta specifically works around Apple's lack of WebGL support in its iOS WebKit because Ejecta is not designed or intended for use cases of general web browsing, and as such does not use WebKit on iOS.

Ejecta runs on all iPad models and most iPhone models starting with the iPhone 3GS or any subsequent iPhone model. Applications on Ejecta do not require any special logic to run or behave differently on an iPad vs and iPhone.

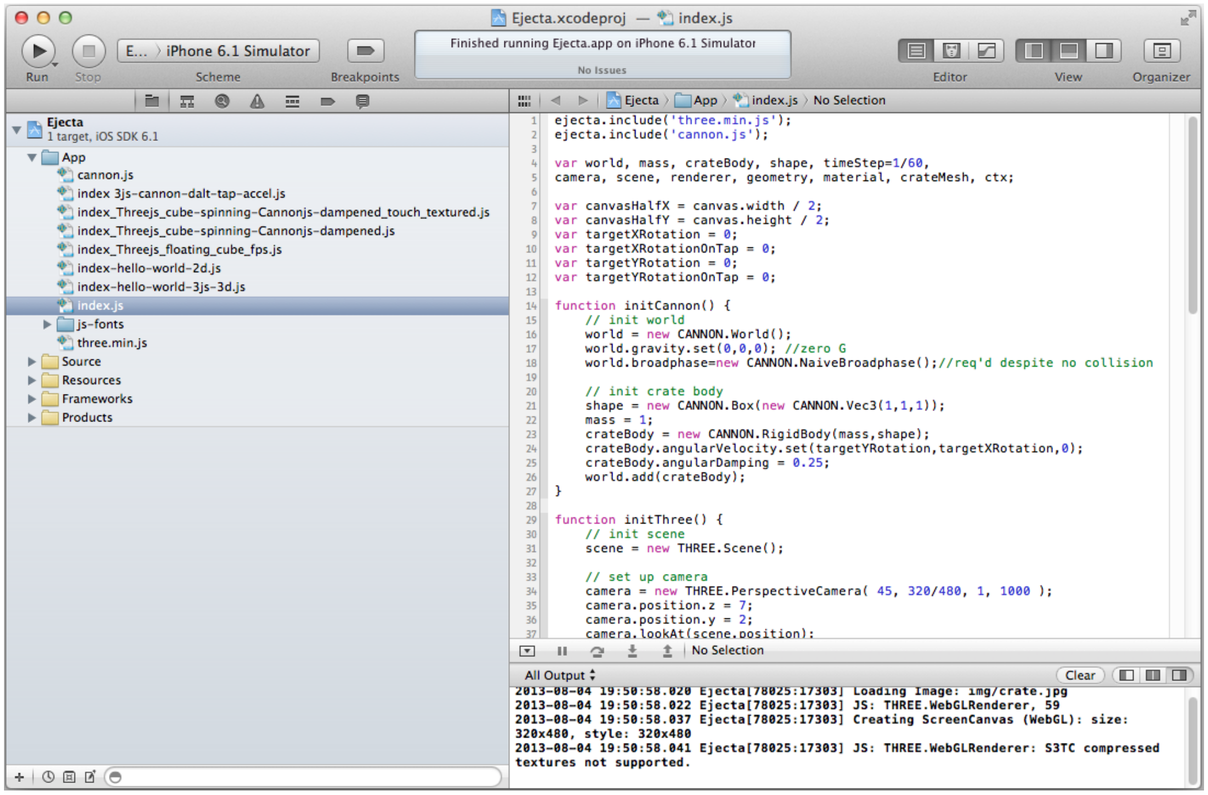
Required Developer Tools

Ejecta is an iOS application container, and requires Apple's Xcode SDK. Because Xcode includes iOS simulator, one can run an Ejecta project there without having an actual iOS device. However there are some limitations [ref4] to the iOS simulator so having an actual iOS device to test on is always preferable.

Structure of an Ejecta Project

Ejecta's structure is very simple by providing an "App" directory which must contain the WebGL javascript code and any local supporting resources, similar to the Document Root directory on a web server like Apache. However unlike a web server, Ejecta only wants to execute javascript code so at minimum this App directory must contain an index.js file. So while the Ejecta framework itself is written mostly in Objective-C, developing apps on Ejecta requires the developer to write code in Javascript.

Furthermore just like a web server, any additional resource files may be added to this App directory as needed. If additional javascript library (.js) files are needed, they can be placed in this App directory alongside the index.js file. If graphics or other static resource files are needed, they can also be included and put inside sub-directories as well.


Ejecta.xcodeproj — index.js
Run
Stop
E… 〉 iPhone 6.1 Simulator
Scheme
Breakpoints
Finished running Ejecta.app on iPhone 6.1 Simulator
No Issues
Editor
View
Organizer
Ejecta 〉 App 〉 index.js 〉 No Selection
Ejecta
1 target, iOS SDK 6.1
App
cannon.js
index 3js-cannon-dalt-tap-accel.js
index_Threejs_cube-spinning-Cannonjs-dampened_touch_textured.js
index_Threejs_cube-spinning-Cannonjs-dampened.js
index_Threejs_floating_cube_fps.js
index-hello-world-2d.js
index-hello-world-3js-3d.js
index.js
js-fonts
three.min.js
Source
Resources
Frameworks
Products
No Selection
All Output
Clear

```
ejecta.include('three.min.js');
ejecta.include('cannon.js');

var world, mass, crateBody, shape, timeStep=1/60,
camera, scene, renderer, geometry, material, crateMesh, ctx;

var canvasHalfX = canvas.width / 2;
var canvasHalfY = canvas.height / 2;
var targetXRotation = 0;
var targetXRotationOnTap = 0;
var targetYRotation = 0;
var targetYRotationOnTap = 0;

function initCannon() {
    // init world
    world = new CANNON.World();
    world.gravity.set(0,0,0); //zero G
    world.broadphase=new CANNON.NaiveBroadphase();//req'd despite no collision

    // init crate body
    shape = new CANNON.Box(new CANNON.Vec3(1,1,1));
    mass = 1;
    crateBody = new CANNON.RigidBody(mass,shape);
    crateBody.angularVelocity.set(targetYRotation,targetXRotation,0);
    crateBody.angularDamping = 0.25;
    world.add(crateBody);
}

function initThree() {
    // init scene
    scene = new THREE.Scene();

    // set up camera
    camera = new THREE.PerspectiveCamera( 45, 320/480, 1, 1000 );
    camera.position.z = 7;
    camera.position.y = 2;
    camera.lookAt(scene.position);
```

```
2013-08-04 19:50:58.020 Ejecta[78025:17303] Loading Image: img/crate.jpg
2013-08-04 19:50:58.022 Ejecta[78025:17303] JS: THREE.WebGLRenderer, 59
2013-08-04 19:50:58.037 Ejecta[78025:17303] Creating ScreenCanvas (WebGL): size:
320x480, style: 320x480
2013-08-04 19:50:58.041 Ejecta[78025:17303] JS: THREE.WebGLRenderer: S3TC compressed
textures not supported.
```

Figure 5: Ejecta in Xcode

Hello, World without WebGL

First to validate that Xcode and Ejecta were working correctly, I tested a minimal "Hello, World" application with Ejecta using a 2D canvas so no WebGL was used.

index.js :

```
ctx = canvas.getContext('2d');
ctx.textAlign = 'center';
ctx.fillStyle = '#ffffff';
ctx.font = "22pt Avenir-Black";
ctx.fillText("Hello, World", canvas.width/2, canvas.height/2);
```

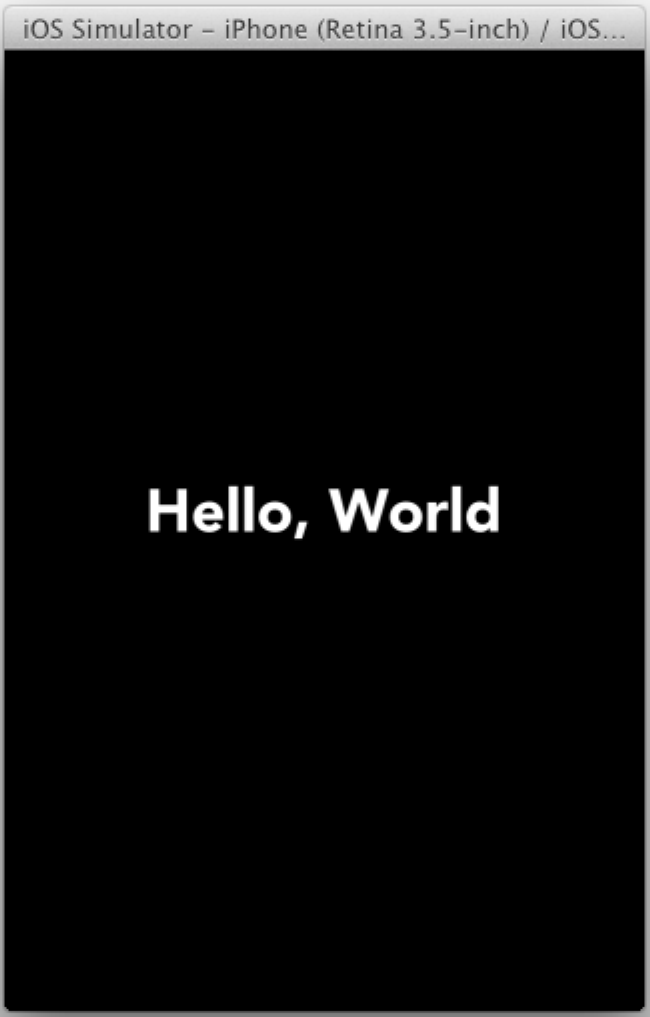


Figure 6: Hellow World running in 2D without WebGL

Hello, World with WebGL

Next I wanted to create a Hello World screen using WebGL, which would require rendering a font in a WebGL 3D context.

Introducing the Three.js javascript library

Basic WebGL has no built-in font rendering capabilities, so I began to consider using a javascript library to support my intentions. While I worried that using javascript libraries would prevent my learning "pure" webGL, I noticed that all the vendor demos and tests at https://www.khronos.org/registry/webgl/sdk/ also use different javascript libraries!

So I went ahead and selected the three.js [ref6] javascript library to facilitate my goals. Three.js is one of the most widely used WebGL javascript libraries and provides objects and methods that are conceptually similar to those used in CS-551 assignments with JOGL. Three.js is well documented, and there are a wealth of three.js examples on the web already [ref20].

index.js:

```
//include min-ified three.js
ejecta.include('three.min.js');
//TrueType font converted to javascript using
//http://typeface.neocracy.org/fonts.html
ejecta.include('js-fonts/optimer_regular.typeface.js');

// init and clear WebGL renderer
var renderer = new THREE.WebGLRenderer({antialias: true});
renderer.clear();

// set up Perspective Camera
var camera = new THREE.PerspectiveCamera(60, canvas.width/canvas.height, 1, 1000);
camera.position.set(0,0,120);

//init scene
var scene = new THREE.Scene();

// define 3D text, add to scene
var textMaterial = new THREE.MeshBasicMaterial( { color: 0x8822ff } );
var textGeom = new THREE.TextGeometry( "Hello, World",
                                       {
                                       size: 12, height: 0, curveSegments: 5,
                                       font: "optimer", weight: "normal", style: "normal",
                                       });
var textMesh = new THREE.Mesh(textGeom, textMaterial );

textGeom.computeBoundingBox();
var textWidth = textGeom.boundingBox.max.x - textGeom.boundingBox.min.x;

textMesh.position.set( -0.5 * textWidth, 0, 0 );
scene.add(textMesh);
```

```

// display code that will iterate continually
function run()
{
   camera.lookAt(scene.position);
   renderer.render( scene, camera );
   requestAnimationFrame(run);
}

run();
```

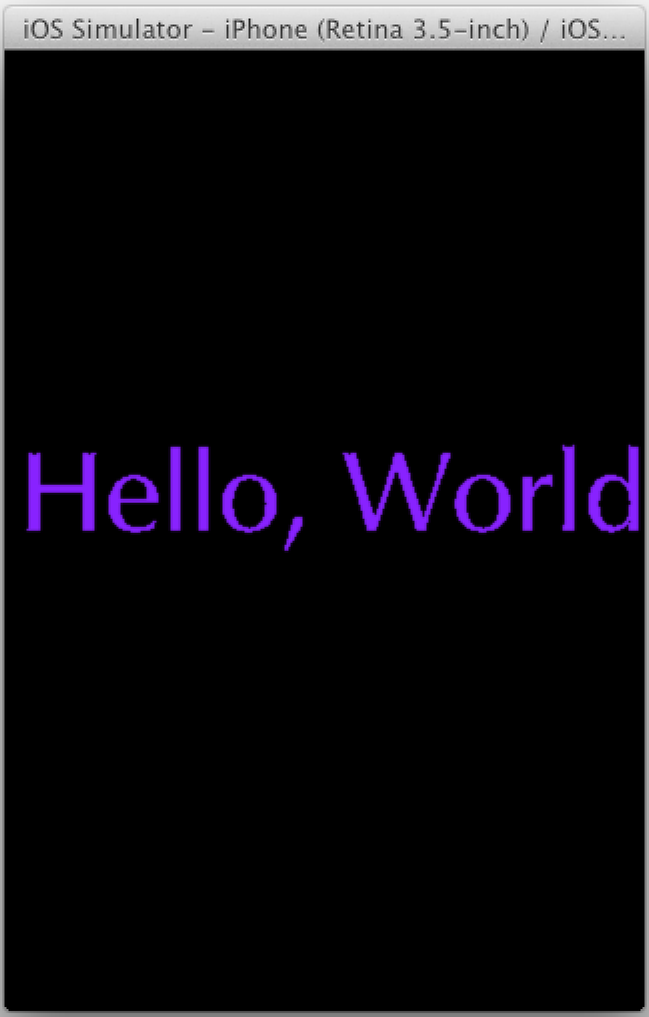


Figure 7: Hello World in WebGL using Three.js

Notable Aspects

Even though this is a static view, note lines 34-41with the run() function and the requestAnimationFrame(run) call on line 38. This is the block of code that gets called continually to support animation updates to the scene [ref7].

Animating 3D Shapes with WebGL

In this example, I was able to add multiple objects to a scene, as well as a light source, shadows, etc. and successfully animate one particular object (litCube) using the requestAnimationFrame(callback) technique.

index.js:

```
//include min-ified version of three.js library
ejecta.include('three.min.js');

// init WebGL Renderer
var renderer = new THREE.WebGLRenderer({antialias: true});
renderer.setClearColor(new THREE.Color(0xEEEEEE));
renderer.clear();

// set up camera
var camera = new THREE.PerspectiveCamera(45, 320/480, 1, 10000);
camera.position.z = 200;
camera.position.y = 280;
camera.position.x = 15;

//init scene
var scene = new THREE.Scene();

// define cube, add to scene
var cube = new THREE.Mesh(new THREE.CubeGeometry(45,45,45),
                         new THREE.MeshBasicMaterial({color: 0x005500}));
scene.add(cube);

// set up light, add to scene
var light = new THREE.SpotLight();
light.position.set( 170, 330, -160 );
scene.add(light);

// set up a lit cube, add to scene
var litCube = new THREE.Mesh(
                            new THREE.CubeGeometry(45,45,45),
                            new THREE.MeshLambertMaterial({color: 0xFFFFFF}));

scene.add(litCube);

// set up plane under cubes, add to scene
var planeGeo = new THREE.PlaneGeometry(400, 200, 10, 10);
var planeMat = new THREE.MeshLambertMaterial({color: 0xFFFFFF});
var plane = new THREE.Mesh(planeGeo, planeMat);
```

```javascript
plane.rotation.x = -Math.PI/2;
//plane.position.y = -25;
plane.receiveShadow = true;
scene.add(plane);

// camera and scene are set now, OK to lookAt
camera.lookAt(scene.position);

var frameNum=0,startTime,runningTime,fps;  // tracking vars for frames-per-sec

startTime=new Date().getTime();

// iteration block
function run()
{
    // modify litCube position and rotation as a function of time
    var t = new Date().getTime()
    litCube.position.x = Math.cos(t/600)*85;
    litCube.position.y = 60-Math.sin(t/900)*25;
    litCube.position.z = Math.sin(t/600)*85;
    litCube.rotation.x = t/500;
    litCube.rotation.y = t/800;

    // Render the scene
    frameNum++;
    runningTime=(new Date().getTime()-startTime)/1000;
    fps=frameNum/runningTime;
    renderer.render( scene, camera );
    console.log("mean fps: "+fps);

  //  renderer.render( scene, camera );

    // Ask for another frame to keep iterating
    requestAnimationFrame(run);
}

run();
```

Here are 2 different screen captures taken at different times in the animation.

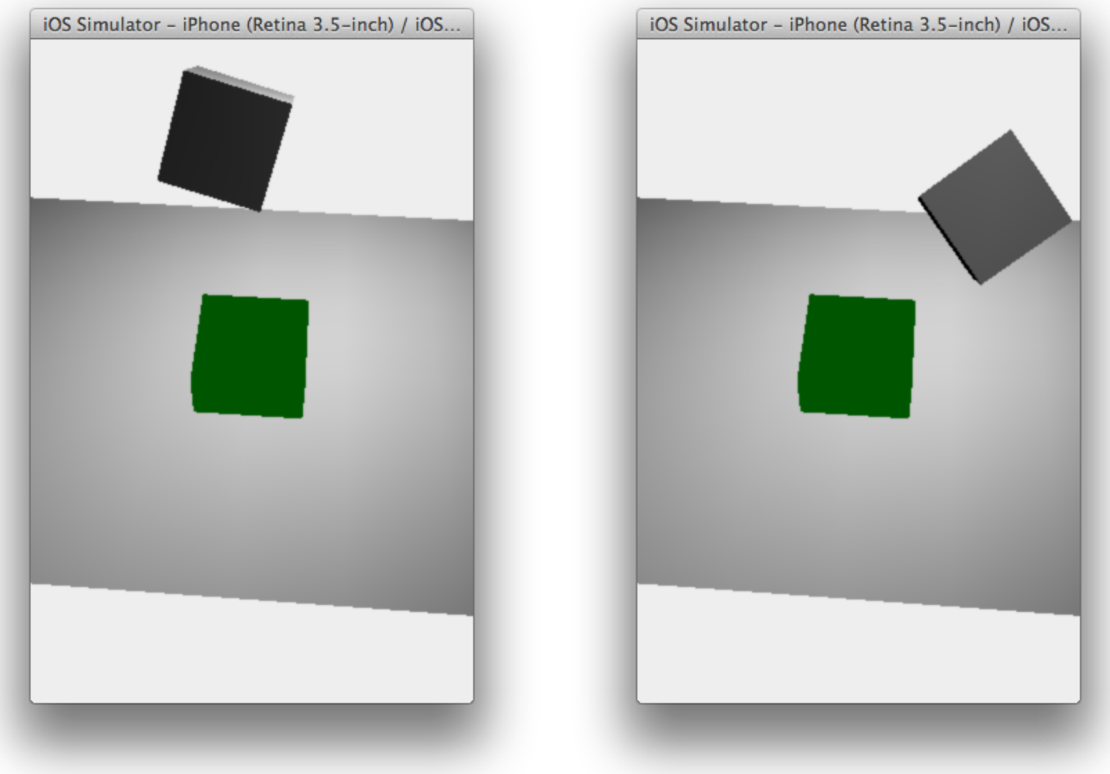
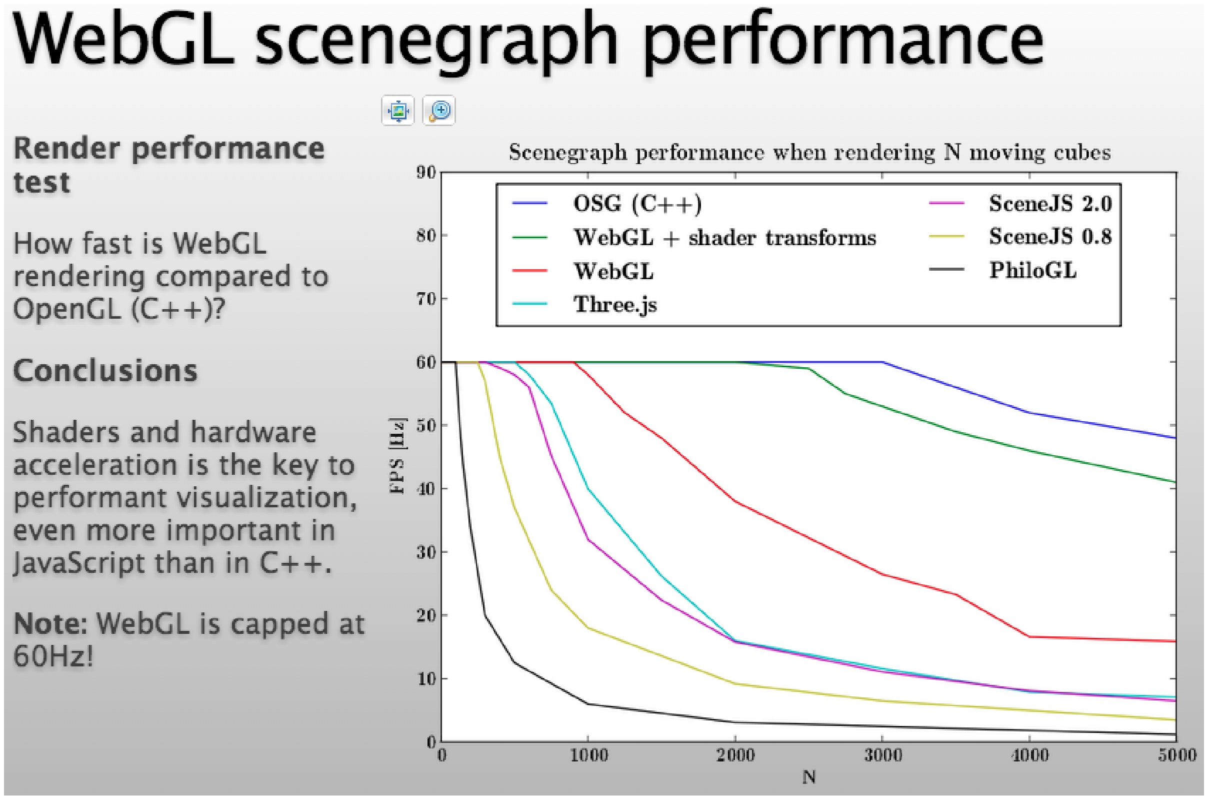

iOS Simulator – iPhone (Retina 3.5-inch) / iOS...
iOS Simulator – iPhone (Retina 3.5-inch) / iOS...

Figure 8: Snapshots during animation of 1 cube moving over another one with a plane and lighting

Notable Findings - WebGL Performance

In this example, I also measured the frames-per-second based on incrementing the frameNum variable on line 63 inside the rendering block of code and dividing that in line 65 by the runningTime variable updated in line 64. In both the iOS simulator on my Mac PowerBook and on my personal iPhone 4 device, I averaged between 59 and 60 frames per second.

Since writing this code but with no time remaining to add to this project, I have learned that most implementations of WebGL cap the frame rates to a maximum of 60Hz. In general the performance of rendering larger numbers of objects reduces the frames-per-second rate, and the rate of degradation varies depending on which methods one uses for the WebGL scene and objects. Certainly this is an area that could be studied and analyzed in more depth in the future.


WebGL scenegraph performance

Render performance test

How fast is WebGL rendering compared to OpenGL (C++)?

Conclusions

Shaders and hardware acceleration is the key to performant visualization, even more important in JavaScript than in C++.

Note: WebGL is capped at 60Hz!

Figure 9: WebGL implementations vs OpenGL [ref13]

Adding a Physics Library

Now that I had the ability to animate a scene with different objects, I wanted to also include some physics properties in an animation. For this I choose cannon.js [ref10] because it blended very logically with Three.js. Physics libraries have become more prominent in recent years, perhaps in part due to the success of the Angry Birds game which uses the Box2D physics engine. [ref12]

In this example, a cube rotates about its vertical (Y) axis, only its rotation is dampened such that it rotates more slowly over time to the point of stopping the rotation.

index.js:

```
ejecta.include('three.min.js');
// cannon.js physics - see http://cannonjs.org/
ejecta.include('cannon.js');

var world, mass, cubeBody, shape, timeStep=1/60,  // vars for cannon.js
camera, scene, renderer, geometry, material, cubeMesh; // vars for three.js

function initCannon() {
    // init world physics
    world = new CANNON.World();
    world.gravity.set(0,0,0); // zero gravity world
    world.broadphase=new CANNON.NaiveBroadphase();//req'd despite no collisions

    // define body physics for Cube
    shape = new CANNON.Box(new CANNON.Vec3(1,1,1));
    mass = 1;
    cubeBody = new CANNON.RigidBody(mass,shape);
    cubeBody.angularVelocity.set(0,10,0);  //rotate around y axis
    cubeBody.angularDamping = 0.25; // with this damping factor

    // add this cubeBody to world
    world.add(cubeBody);
}

function initThree() {
    // init scene
    scene = new THREE.Scene();

    // set up camera
    camera = new THREE.PerspectiveCamera( 75, 320/480, 1, 1000 );
    camera.position.z = 5;
    scene.add( camera );

    // set up wireframe cube in red
    geometry = new THREE.CubeGeometry( 2, 2, 2 );
    material = new THREE.MeshBasicMaterial( {color: 0xff0000,wireframe: true});
    cubeMesh = new THREE.Mesh( geometry, material );
    scene.add( cubeMesh );

    // set up WebGL renderer
    renderer = new THREE.WebGLRenderer({antialias: true});
}

function updatePhysics() {
    // Step forward in time
    world.step(timeStep);

    // update Three.js cubeMesh quat coordinates with Cannon.js physics calc's
    cubeBody.quaternion.copy(cubeMesh.quaternion); //rotation update
}

function animate() {
    // update physics model
    updatePhysics();
    // render the scene
    renderer.render( scene, camera );
    // get next frame for animation!
    requestAnimationFrame( animate );
}

initThree();
```

```
initCannon();
animate();
```

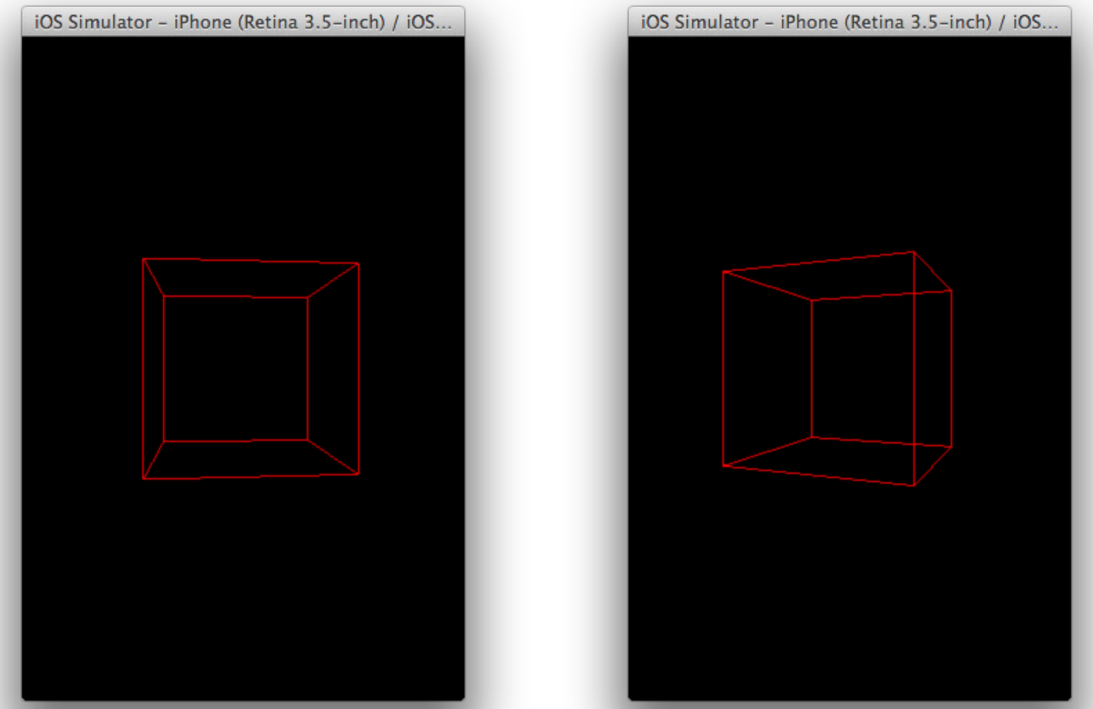

iOS Simulator – iPhone (Retina 3.5-inch) / iOS...
iOS Simulator – iPhone (Retina 3.5-inch) / iOS...

Figure 10: Rotating cube with dampening using Three.js and cannon.js together

Notable Findings

cannon.js blended very cleanly with three.js, in part because it was designed originally in javascript rather than being a port of a physics engine written in another language into javascript [ref14]. The key to three.js and cannon.js working together is that they both use different key variable names (lines 5 & 6) and elements in cannon.js mapped neatly to those in three.js (line 49).

Adding swipe controls, texture map to spinning cube

Since this Project is specific to iOS where one of its key features is its touch-base user interface, my final experiment was to add swipe (touch) controls to the spinning cube, along with a texture map [ref9] to make the cube appear to be a crate box just to make a more natural looking object. Not unlike our JOGL homework assignments requiring mouse inputs, swipe controls entailed defining a Listener method and then once it is invoked applying the x/y values of the swipe to the rotation parameters of the cube's body variable.

index.js:

```
ejecta.include('three.min.js');
ejecta.include('cannon.js');

var world, mass, crateBody, shape, timeStep=1/60,
camera, scene, renderer, geometry, material, crateMesh, ctx;

var canvasHalfX = canvas.width / 2;
var canvasHalfY = canvas.height / 2;
var targetXRotation = 0;
var targetXRotationOnTap = 0;
var targetYRotation = 0;
var targetYRotationOnTap = 0;

function initCannon() {
    // init world
    world = new CANNON.World();
    world.gravity.set(0,0,0); //zero G
    world.broadphase=new CANNON.NaiveBroadphase();//req'd despite no collision

    // init crate body
    shape = new CANNON.Box(new CANNON.Vec3(1,1,1));
    mass = 1;
    crateBody = new CANNON.RigidBody(mass,shape);
    crateBody.angularVelocity.set(targetYRotation,targetXRotation,0);
    crateBody.angularDamping = 0.25;
    world.add(crateBody);
}

function initThree() {
    // init scene
    scene = new THREE.Scene();

    // set up camera
    camera = new THREE.PerspectiveCamera( 45, 320/480, 1, 1000 );
    camera.position.z = 7;
    camera.position.y = 2;
    camera.lookAt(scene.position);
    scene.add( camera );

    // add subtle blue ambient lighting
    var ambientLight = new THREE.AmbientLight(0x000044);
    scene.add(ambientLight);

    // directional lighting
    var directionalLight = new THREE.DirectionalLight(0xffffff);
    directionalLight.position.set(1, 1, 1).normalize();
    scene.add(directionalLight);

    // set up crate Mesh
    geometry = new THREE.CubeGeometry( 2, 2, 2 );
    material = new THREE.MeshLambertMaterial(
                        {map:THREE.ImageUtils.loadTexture('img/crate.jpg')});
    crateMesh = new THREE.Mesh( geometry, material );
    crateMesh.overdraw = true;
    scene.add( crateMesh );

    // init renderer
    renderer = new THREE.WebGLRenderer({antialias: true});

    //add Event Listeners for touch events
    document.addEventListener( 'touchstart', onDocumentTouchStart, false );
        document.addEventListener( 'touchmove', onDocumentTouchMove, false );
}

function onDocumentTouchStart( event ) {
        if ( event.touches.length === 1 ) { //only respond to single touch
        event.preventDefault(); //prevent the default behavior where touch
                                // shifts the whole screen around in iOS
        crateBody.angularDamping = 1;//"catch" crate by stoping any rotation

        // calculate tap start x/y vars
        tapXstart = event.touches[ 0 ].pageX - canvasHalfX;
        tapYstart = event.touches[ 0 ].pageY - canvasHalfY;

        // capture current X/Y Rotation values
        targetXRotationOnTap = targetXRotation;
        targetYRotationOnTap = targetYRotation;
        }
}

function onDocumentTouchMove( event ) {
        if ( event.touches.length === 1 ) {  //only respond to single touch
        event.preventDefault(); //prevent the default behavior where touch
                                // shifts the whole screen around in iOS
        crateBody.angularDamping = 0.25;//set back to orig Damping factor

        // calculate touch move x/y vars
        tapX = event.touches[ 0 ].pageX - canvasHalfX;
        tapY = event.touches[ 0 ].pageY - canvasHalfY;

        //update X/Y Rotation values
        targetXRotation = targetXRotationOnTap + ( tapX - tapXstart ) * 0.05;
        targetYRotation = targetYRotationOnTap + ( tapY - tapYstart ) * 0.05;

        // update crate's Angular Velocity values X/Y
        crateBody.angularVelocity.set(targetYRotation,targetXRotation,0);
        }
}

function updatePhysics() {
    // Step the physics world
    world.step(timeStep);
    // Copy quat's from Cannon.js body to Three.js mesh
    crateBody.quaternion.copy(crateMesh.quaternion); // update rotation
}

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

initThree();
initCannon();
animate();
```

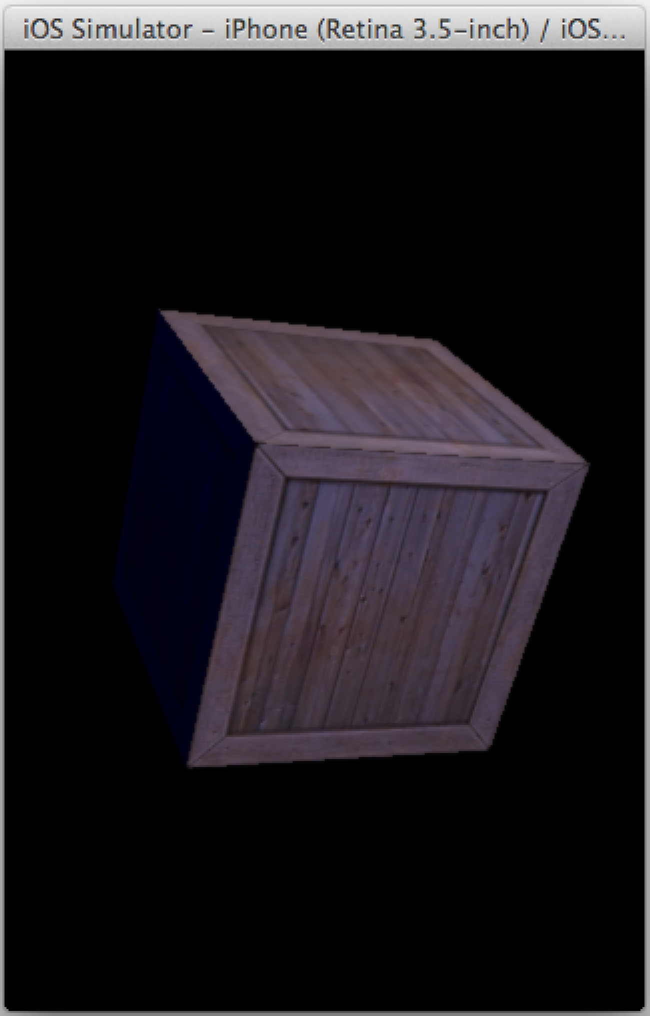


Figure 12: Screen capture of touch-controlled spinning crate

Notable Findings - Ejecta Support of iOS specific UI Events

In this final example, I successfully mapped an image file texture onto the cube making in now a crate, added some lighting sources. Furthermore, leveraging iOS touch events [ref19,ref8], I am able to control its rotation in any direction via listeners responding to those touch events by assigning the X/Y direction and magnitude of each touch event to the X and Y components of the crate's angular velocity established via cannon.js.

Ejecta supports not only iOS touch events but also devicemotion events [ref18]. I was able to validate this technically using javascript console logging, but did not have time to write another Ejecta project that utilized devicemotion events in a more meaningful way.

Conclusions and Suggested Areas for Future Work

Ejecta has proven to be a young but viable platform for WebGL development on iOS. However WebGL by itself I find rather primitive and only offers low-level capabilities. So being new to WebGL, libraries like Three.js are essential for providing key capabilities like management of a scene, placing the camera, geometric object methods, etc. There are many more capabilities I didn't have time to explore like the ability to load and animate external object files from software like Blender and Wavefront, and more. [ref15,ref16]

It must be noted that not all javascript libraries that offer WebGL methods work with Ejecta, because those libraries are written assuming that they will be used in Web browsers. Ejecta only supports a minimal subset of Web browser HTML5 capabilities, and many javascript libraries supporting WebGL have been developed before Ejecta was even available. One in particular that I tried to use with Ejecta but couldn't due to javascript errors was processing.js which supports a WebGL rendering mode [ref17] for example.

While in many cases WebGL is used for development of games I am personally curious about applications for data visualization in 3D using WebGL, especially where the application is able to pull the source data to be visualized and presented from external sources using RESTful methods [ref11].

Finally, all source code referenced in this paper can be found at http://mason.gmu.edu/~bcoughl2/cs551/Ejecta-1.3-bcoughl2-gmu-CS511.zip .

References and Citations

1. http://caniuse.com/#feat=webgl
2. http://www.khronos.org/webgl/security/#Cross-Origin_Media
3. http://impactjs.com/ejecta
4. http://developer.apple.com/library/ios/#documentation/IDEs/Conceptual/iOS_Simulator_Guide/TestingontheiOSSimulator/TestingontheiOSSimulator.html
5. http://www.khronos.org/registry/webgl/specs/1.0/
6. http://threejs.org/
7. http://www.w3.org/TR/animation-timing/#requestAnimationFrame
8. http://developer.apple.com/library/ios/#documentation/AppleApplications/Reference/SafariWebContent/HandlingEvents/HandlingEvents.html
9. Spinning Crate inspiration: http://www.html5canvastutorials.com/three/html5-canvas-webgl-texture-with-three-js/
10. Cannon.js home - http://cannonjs.org/ ; github home - http://github.com/schteppe/cannon.js
11. http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm
12. http://www.geek.com/games/box2d-creator-asks-rovio-for-angry-birds-credit-at-gdc-1321779/
13. http://granular.cs.umu.se/pres2/#slide16
14. http://granular.cs.umu.se/pres2/#slide18
15. http://impactjs.com/ejecta/supported-apis-methods
16. https://github.com/mrdoob/three.js/wiki/Features
17. http://processingjs.org/articles/RenderingModes.html
18. http://developer.apple.com/library/safari/#documentation/SafariDOMAdditions/Reference/DeviceMotionEventClassRef/DeviceMotionEvent/DeviceMotionEvent.html
19. http://developer.apple.com/library/safari/#documentation/UserExperience/Reference/TouchEventClassReference/TouchEvent/TouchEvent.html
20. http://stemkoski.github.io/Three.js/