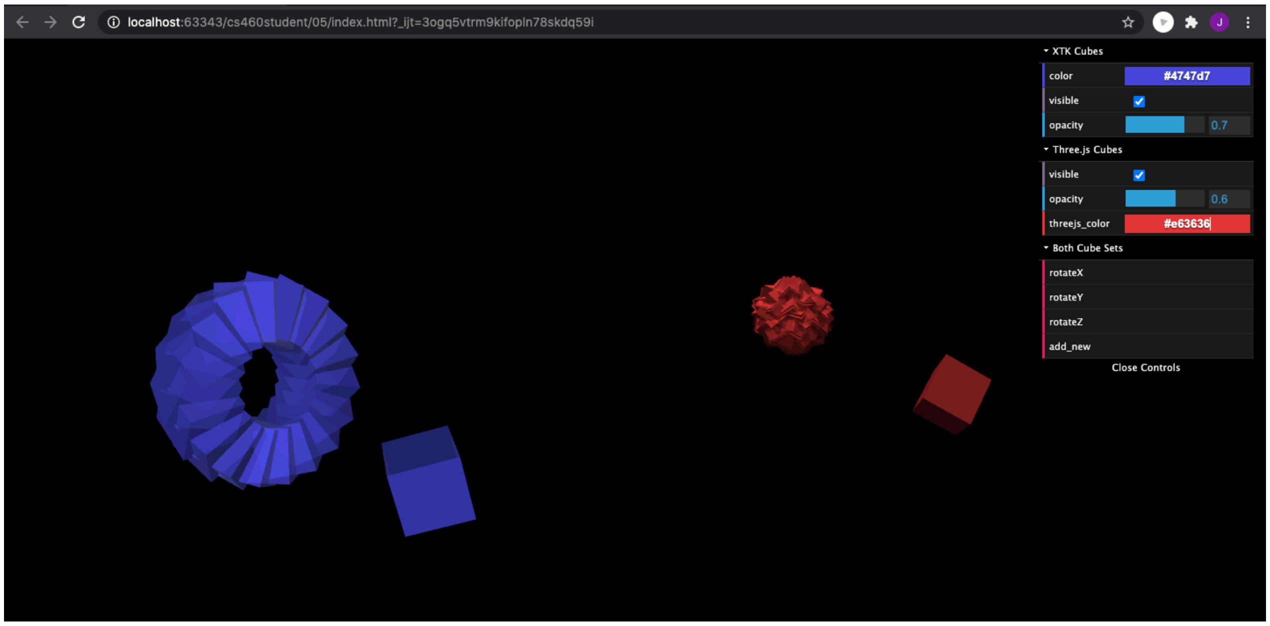
CS460 Fall 2020
Github Username: JamesEdMichaud
Due Date: 10/19/2020

Assignment 5: Scene Control with dat.GUI and Transformations!

Welcome back to framework country! This time we will use XTK and Three.js to study rotations.

In class, we connected the `dat.gui` library with XTK to control properties of a single cube. We also introduced the `transformer` object to rotate the cube along the world `x`- and `y` axis. In this assignment, we will create a website with two 3D scenes. One scene will be based on XTK, and the other will be based on Three.js. Then, we will use `dat.gui` to control objects in the scene. As a final result, each scene will contain two objects. We then can observe two different ways of rotating objects since XTK and Three.js.


localhost:63343/cs460student/05/index.html?_ijt=3ogq5vtrm9kifopln78skdq59i

XTK Cubes
- color: #4747d7
- visible: ✓
- opacity: 0.7

Three.js Cubes
- visible: ✓
- opacity: 0.6
- threejs_color: #e63636

Both Cube Sets
- rotateX
- rotateY
- rotateZ
- add_new

Close Controls

There is no starter code for assignment 5. Please start from scratch and save your code your fork as `05/index.html`.

Part 1 Coding: Configure the `<div>`s. (10 points)

We will create two viewports next to each other. Please add two `<div>` containers in the body of the HTML document. Name these containers `r1` and `r2` using the id property. Then, add styling to the header of the HTML document as follows:

```
<style>
  html, body {
    background-color: #000;
    margin: 0;
    padding: 0;
    height: 100%;
    overflow: hidden !important;
  }
  #r1 {
    width:50%;
    height:100%;
    float: left;
  }
  #r2 {
    width:50%;
    height:100%;
    float:left;
  }
</style>
```

You can verify the placement of the `<div>` containers using the Web Developer Tools. They should be next to each other and together, fill the whole window.

Part 2 Coding: Setup the XTK scene. (10 points)

Add the `xtk_edge.js` and `xtk_xdat.gui.js` libraries using the `<script>` tags as we did in class and in assignment 2. Then, create the `window.onload` function to set up the `X.renderer3D` and add a single `X.cube`. **Since we place the renderer into the `r1` container, we need to set `r.container='r1';` just before calling `r.init();`.** Please check if the XTK cube appears by reloading the website.

Part 3 Coding: Setup the Three.js scene. (15 points)

For Three.js, please add the `three.min.js` and `TrackballControls.js` as we did in assignment 3. Then, follow our old code to setup a `THREE.Scene` with the `THREE.PerspectiveCamera`, the `THREE.WebGLRenderer`, the `THREE.AmbientLight`, the `THREE.DirectionalLight`, and the `THREE.TrackballControls`. **Since we now use a `<div>` container as our viewport, we need to do the following:**

```
var r2 = document.getElementById('r2'); // get the div container!!!
// ...
var ratio = r2.clientWidth / r2.clientHeight; // use the container's clientWidth and clientHeight
                              // rather than window.innerWidth and window.innerHeight
// ...
var camera = new THREE.PerspectiveCamera(fov, ratio, zNear, zFar);

var renderer = new THREE.WebGLRenderer({antialias:true});
renderer.setSize( r2.clientWidth, r2.clientHeight ); // again use the container
r2.appendChild( renderer.domElement ); // and append the domElement to the container

// ...

var controls = new THREE.TrackballControls( camera, r2 ); // pass the container to the camera
```

Please don't forget the `animate` loop! Then, please add the `THREE.BoxBufferGeometry` and the `THREE.MeshStandardMaterial` to create a new `THREE.Mesh` and add it to the scene. **When you reload the page, there should be now two cubes - one with XTK and one with Three.js!**

Part 4 Coding: Connect XTK to `dat.GUI` to control cube properties. (10 points)

Please create the `dat.GUI()` user interface for XTK. For this, we will use `gui.addFolder` and access the `visible`, `opacity`, and `color` properties as we did in class. After reloading, this should work right away.

Part 5 Coding: Introduce the helper object for `dat.GUI`. (5 points)

XTK's properties connect well with `dat.GUI` but for more advanced functionality, and especially to control Three.js, we will need a helper object. Please add the following code just before the `dat.GUI()` setup.

```
var controller = {

  'threejs_color': 0xffffff

};
```

Part 6 Coding: Connect Three.js to `dat.GUI` to control cube properties. (5 points)

To connect `dat.GUI` and Three.js, we will first use `gui.addFolder` to group the controls. Then, we want to access the same properties as in the XTK case. However, connecting Three.js with `dat.GUI` is not as straight forward—even with a helper object :(. It requires the following code:

```
var threejsUI = gui.addFolder('Three.js Cube');
threejsUI.add(cube, 'visible');
threejsUI.add(cube.material, 'opacity', 0, 1).onChange( function() {
  cube.material.transparent = true;
});
threejsUI.addColor(controller, 'threejs_color').onChange( function() {
              cube.material.color.set( controller.threejs_color );
            } );
threejsUI.open();
```

After reloading, this should allow to control the visibility, opacity, and color for both the XTK cube and the THREE.js cube.

Part 7 Coding: Extend the helper object for `dat.GUI` and rotate both cubes. (10 points)

We now want to rotate both cubes with three buttons. For this, we will add a new folder to `dat.GUI` as follows:

```
var both = gui.addFolder('Both Cubes');
both.add(controller, 'rotateX');
both.add(controller, 'rotateY');
both.add(controller, 'rotateZ');
both.open();
```

Then, we will extend the `controller` helper object with three rotate methods that rotate by 20 degrees:

```
var controller = {
  'threejs_color': 0xffffff,

  'rotateX': function() {
    c.transform.rotateX(20);
    cube.rotateX(20);
  },
  'rotateY': function() {
    c.transform.rotateY(20);
    cube.rotateY(20);
  },
  'rotateZ': function() {
    c.transform.rotateZ(20);
    cube.rotateZ(20);
  }
};
```

In the code above, we assume that the XTK cube is accessible as `c` and the THREE.js cube is accessible as `cube`. **After reloading, this should allow to rotate the cubes in X,Y, and Z using the three new buttons.**

Part 8 Coding: Add a second cube. (10 points)

Please extend the `controller` helper object with a new method 'add new' and update the `dat.GUI` controls.

```
var controller = {
    // ...
    'add new': function() {
        // TODO!
    }
};

// ...

both.add(controller, 'add new');
both.open();
```

Now, please replace the `//TODO!` above with code that creates for both, XTK and Three.js, a second cube and adds it the viewport. **The new cube should be positioned at (50, 50, 50).** After reloading, and pressing 'add new', both viewports should show two cubes (maybe hidden by the dat.GUI panel).

Part 9 Explaining: Different rotations? (20 points)

So, if we rotate the cubes before adding the second cube, the rotations in XTK and Three.js are very similar. But, after adding the second cube the rotations are very different. Please try to explain what happens.

The XTK cubes rotate around the global axes, whereas the threejs cubes rotate around their individual axes. The `rotate_()` methods use different frames of reference.

For me, they also rotate in opposite directions.

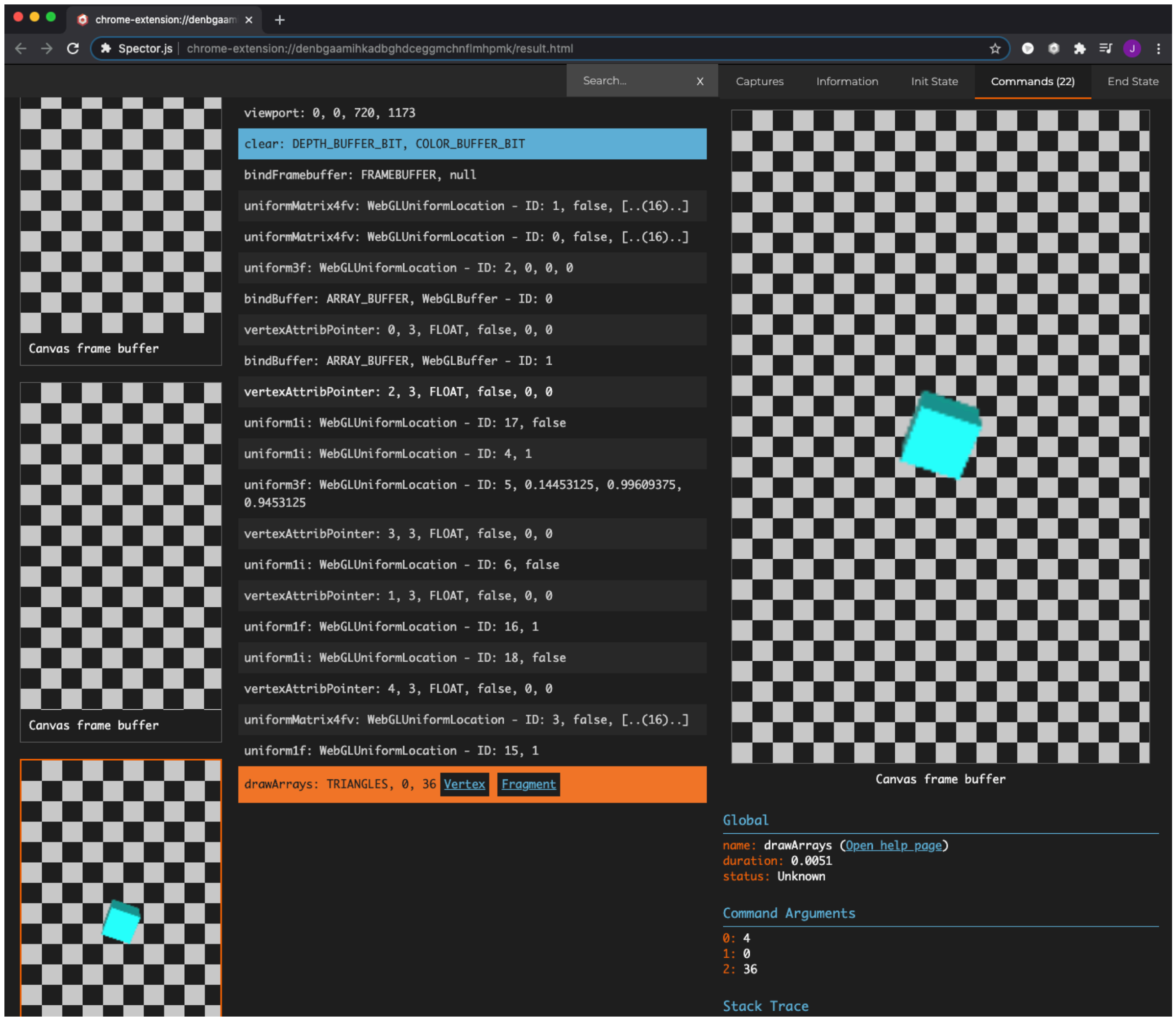
Part 10 Cleanup: Replace the screenshot above, activate Github pages, edit the URL below, and add this PDF to your repo. Then, send a pull request or assignment submission (or do the bonus first). (5 points)

Link to your assignment: `https://jamesedmichaud.github.io`

Bonus (33 points):

We will use `spector.js` to analyse the two viewports. If you did not install this extension yet, please do so by following the instructions at `https://spector.babylonjs.com/`. Then, you can use the extension to capture/record WebGL activity.

Part 1 (5 points): Please use spector.js to capture the viewport that uses XTK and insert a screenshot here.



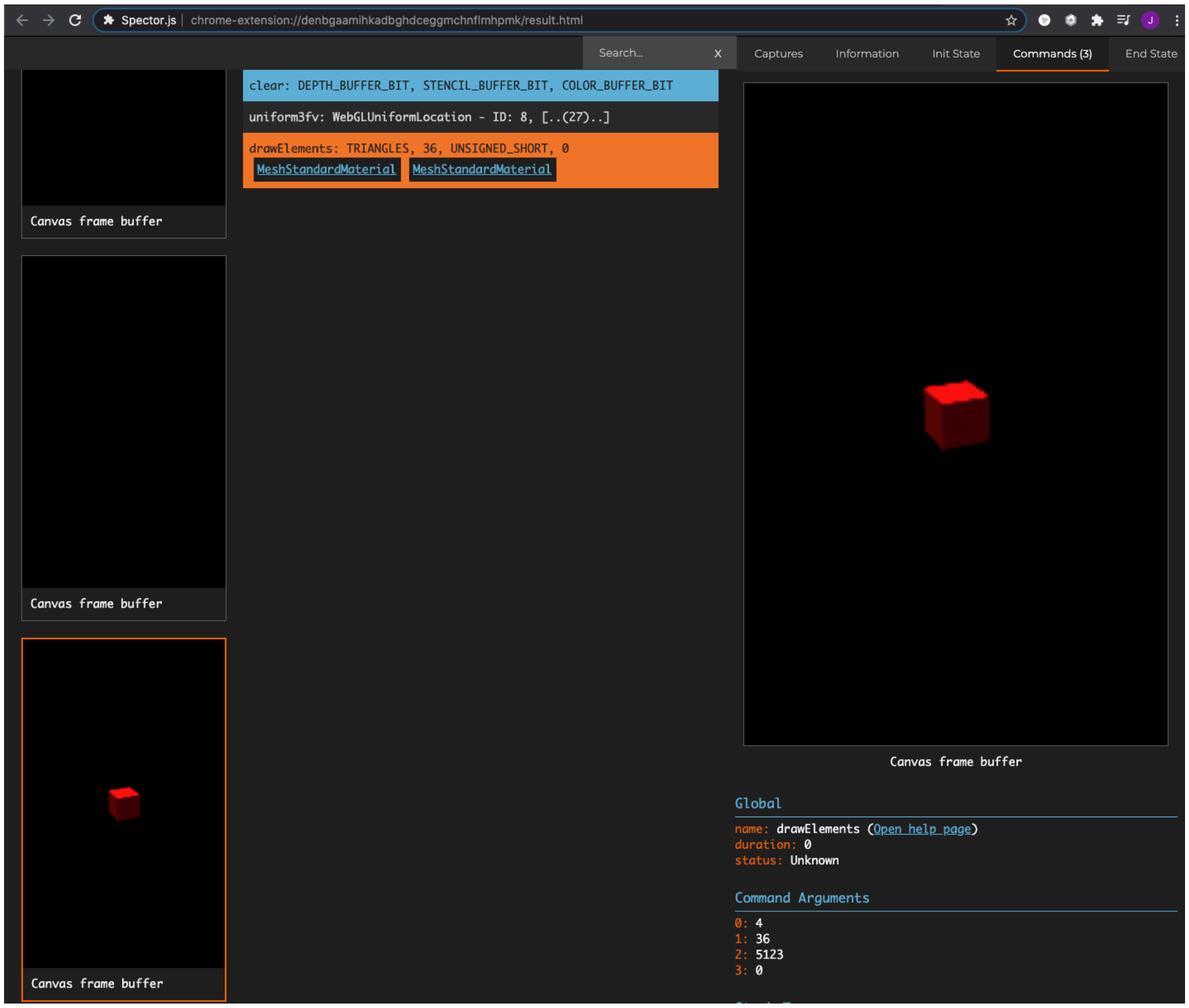
```
viewport: 0, 0, 720, 1173
clear: DEPTH_BUFFER_BIT, COLOR_BUFFER_BIT
bindFramebuffer: FRAMEBUFFER, null
uniformMatrix4fv: WebGLUniformLocation - ID: 1, false, [..(16)..]
uniformMatrix4fv: WebGLUniformLocation - ID: 0, false, [..(16)..]
uniform3f: WebGLUniformLocation - ID: 2, 0, 0, 0
bindBuffer: ARRAY_BUFFER, WebGLBuffer - ID: 0
vertexAttribPointer: 0, 3, FLOAT, false, 0, 0
bindBuffer: ARRAY_BUFFER, WebGLBuffer - ID: 1
vertexAttribPointer: 2, 3, FLOAT, false, 0, 0
uniform1i: WebGLUniformLocation - ID: 17, false
uniform1i: WebGLUniformLocation - ID: 4, 1
uniform3f: WebGLUniformLocation - ID: 5, 0.14453125, 0.99609375,
0.9453125
vertexAttribPointer: 3, 3, FLOAT, false, 0, 0
uniform1i: WebGLUniformLocation - ID: 6, false
vertexAttribPointer: 1, 3, FLOAT, false, 0, 0
uniform1f: WebGLUniformLocation - ID: 16, 1
uniform1i: WebGLUniformLocation - ID: 18, false
vertexAttribPointer: 4, 3, FLOAT, false, 0, 0
uniformMatrix4fv: WebGLUniformLocation - ID: 3, false, [..(16)..]
uniform1f: WebGLUniformLocation - ID: 15, 1
drawArrays: TRIANGLES, 0, 36 Vertex Fragment
```

```
Global
name: drawArrays (Open help page)
duration: 0.0051
status: Unknown

Command Arguments
0: 4
1: 0
2: 36

Stack Trace
```

Part 2 (5 points): Please use spector.js capture the viewport that uses Three.js and insert a screenshot here.



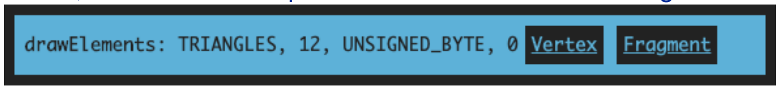
```
clear: DEPTH_BUFFER_BIT, STENCIL_BUFFER_BIT, COLOR_BUFFER_BIT
uniform3fv: WebGLUniformLocation - ID: 8, [..(27)..]
drawElements: TRIANGLES, 36, UNSIGNED_SHORT, 0
 MeshStandardMaterial  MeshStandardMaterial
```

```
Global
name: drawElements (Open help page)
duration: 0
status: Unknown

Command Arguments
0: 4
1: 36
2: 5123
3: 0
```

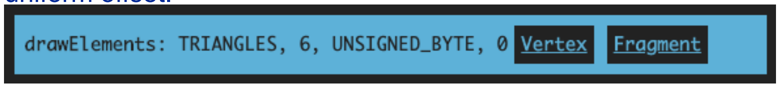
Part 3 (23 points): Compare the `spector.js` recordings. (a) Please report if either XTK or Three.js use an indexed geometry. (b) Also, please explore and compare the length of the GLSL shader codes both libraries use. (c) And, please figure out how the object transformations are passed to the shaders.

(a) I ran the spector.js extension on my assignment 4 solution to see at a lower level what I could find out. I noticed from the output that the drawElements command is used to the draw indexed geometry (ship and wall segments) and that the drawArrays command is used to draw non-indexed geometry (point obstacles). This makes sense, because it lines up with the code we wrote for assignment 4.



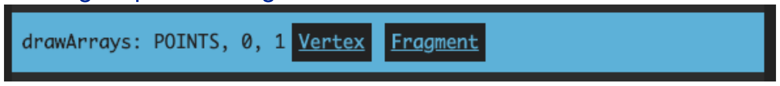
```
drawElements: TRIANGLES, 12, UNSIGNED_BYTE, 0 Vertex Fragment
```

4 triangles make the ship, and 12 vertices means 12 indices in a Uint8Array (UNSIGNED_BYTE array). The 0 corresponds to the 'offset' parameter of the drawElement method. That does something to do with grouping indices (or vertices?) together, like a uniform offset.



```
drawElements: TRIANGLES, 6, UNSIGNED_BYTE, 0 Vertex Fragment
```

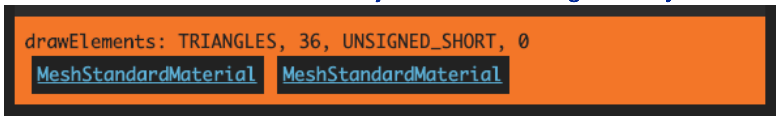
2 triangles per wall segment means 6 indices.



```
drawArrays: POINTS, 0, 1 Vertex Fragment
```

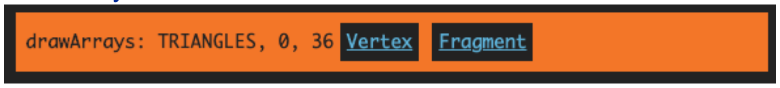
And the single vertex obstacle, drawn using drawArrays. This time the 0 corresponds to the parameter "first", which is the starting index in the array. The 1 is the number of indices to read from that index.

From this we can see that threejs uses indexed geometry.



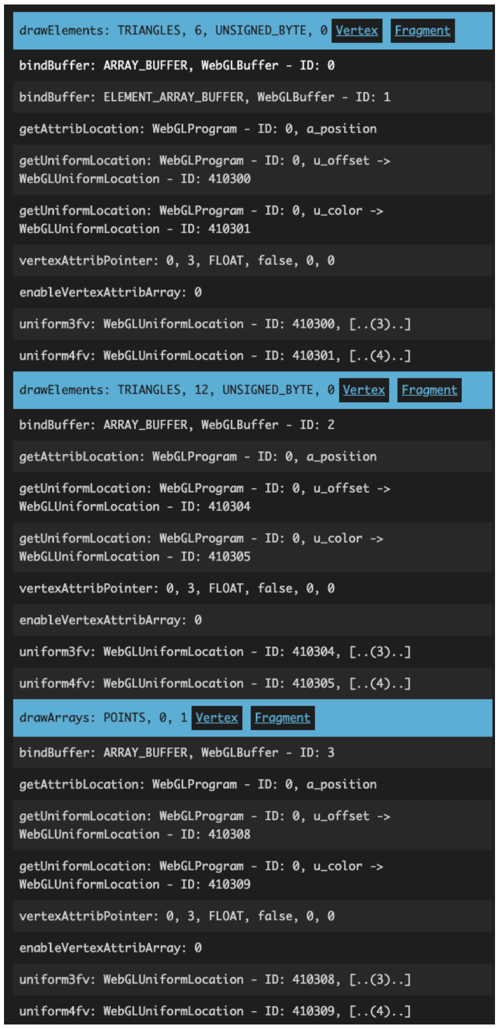
```
drawElements: TRIANGLES, 36, UNSIGNED_SHORT, 0
MeshStandardMaterial MeshStandardMaterial
```

12 triangles per cube results in 36 indices in an UNSIGNED_SHORT array (it looks like threejs uses Uint16Array's). On the XTK side, it looks like indexed geometry is not used. The drawArrays command is used with 36 indices.



```
drawArrays: TRIANGLES, 0, 36 Vertex Fragment
```

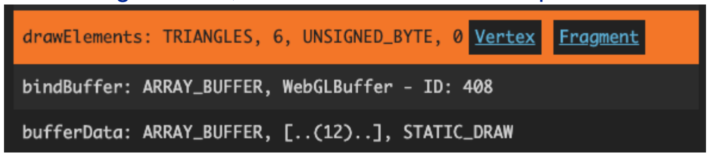
An expanded screenshot of the assignment 4 commands on the left.



```
drawElements: TRIANGLES, 6, UNSIGNED_BYTE, 0 Vertex Fragment
bindBuffer: ARRAY_BUFFER, WebGLBuffer - ID: 0
bindBuffer: ELEMENT_ARRAY_BUFFER, WebGLBuffer - ID: 1
getAttribLocation: WebGLProgram - ID: 0, a_position
getUniformLocation: WebGLProgram - ID: 0, u_offset ->
WebGLUniformLocation - ID: 410300
getUniformLocation: WebGLProgram - ID: 0, u_color ->
WebGLUniformLocation - ID: 410301
vertexAttribPointer: 0, 3, FLOAT, false, 0, 0
enableVertexAttribArray: 0
uniform3fv: WebGLUniformLocation - ID: 410300, [..(3)..]
uniform4fv: WebGLUniformLocation - ID: 410301, [..(4)..]
drawElements: TRIANGLES, 12, UNSIGNED_BYTE, 0 Vertex Fragment
bindBuffer: ARRAY_BUFFER, WebGLBuffer - ID: 2
getAttribLocation: WebGLProgram - ID: 0, a_position
getUniformLocation: WebGLProgram - ID: 0, u_offset ->
WebGLUniformLocation - ID: 410304
getUniformLocation: WebGLProgram - ID: 0, u_color ->
WebGLUniformLocation - ID: 410305
vertexAttribPointer: 0, 3, FLOAT, false, 0, 0
enableVertexAttribArray: 0
uniform3fv: WebGLUniformLocation - ID: 410304, [..(3)..]
uniform4fv: WebGLUniformLocation - ID: 410305, [..(4)..]
drawArrays: POINTS, 0, 1 Vertex Fragment
bindBuffer: ARRAY_BUFFER, WebGLBuffer - ID: 3
getAttribLocation: WebGLProgram - ID: 0, a_position
getUniformLocation: WebGLProgram - ID: 0, u_offset ->
WebGLUniformLocation - ID: 410308
getUniformLocation: WebGLProgram - ID: 0, u_color ->
WebGLUniformLocation - ID: 410309
vertexAttribPointer: 0, 3, FLOAT, false, 0, 0
enableVertexAttribArray: 0
uniform3fv: WebGLUniformLocation - ID: 410308, [..(3)..]
uniform4fv: WebGLUniformLocation - ID: 410309, [..(4)..]
```

(b) The XTK vertex shader is 86 lines long and the fragment shader is 103 lines. The threejs vertex shader is 410 lines and fragment shader is a whopping 1340 lines.

(c) Transformations are passed to the shaders through the buffers?? I'm a bit stumped on where to find this in the spector.js output. In both cases, only a single frame seemed to be captured by spector.js (3 commands each), so I couldn't see any data on the transformations. I know from assignment 4 that I needed to call gl.bufferData() to update the vertex position of wall segments when being transformed. I could find that in the spector.js output from assignment 4, but I'm not able to interpret what it means among the 2293 commands that were logged.



```
drawElements: TRIANGLES, 6, UNSIGNED_BYTE, 0 Vertex Fragment
bindBuffer: ARRAY_BUFFER, WebGLBuffer - ID: 408
bufferData: ARRAY_BUFFER, [..(12)..], STATIC_DRAW
```

That command isn't present anywhere in the assignment 5 spector.js outputs. It may also be the Framebuffers, which I see in the assignment 5 spector.js outputs, but not the assignment 4 output.