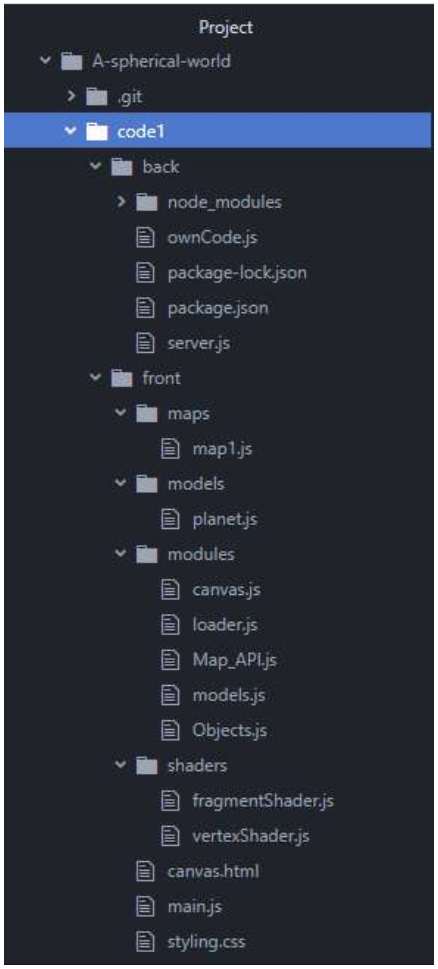
Bit by bit, and line by line

The sphere emerges...

Max Silin

Starting Structure

Folder Structure



Backbone of the application

```html
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>

    <title> My Spherical World </title>
    <meta charset="utf-8">
    <link rel='stylesheet' href='styling.css'/>

    <script src='./modules/canvas.js' type="module"></script>
    <script src='./modules/loader.js' type="module"></script>
    <script src='./modules/Map_API.js' type="module"></script>
    <script src='./modules/models.js' type="module"></script>
    <script src='./modules/Objects.js' type="module"></script>
    <script src='./models/planet.js' type="module"></script>
    <script src='./maps/map1.js' type="module"></script>
    <script src='./shaders/fragmentShader.js' type="module"></script>
    <script src='./shaders/vertexShader.js' type="module"></script>

    <script src='./main.js' type='module'></script>

  </head>
  <body>
    <div id='WebGL_Fallback' hidden='true' class='info_yellow'>
      Web GL is not supported by you or your browser. Please give it some support. A kind word or two will make him feel happier.
    </div>
    <canvas id='canvas'></canvas>
  </body>
</html>
```


Modularising javascript code into different files means you need a backend server to serve these files.

I am using node npm (Node Package Manager) to install the express package and leverage its power.

Express allows you to create a back end server quickly. Here, I am only serving scripts and static assets.

server.js

```js
const express = require('express')
const app = express()
app.use(express.json())

app.use(express.static('../front'))

const PORT = 5000;

app.listen(PORT, ()=>{
  console.log(`Listening on port ${PORT}.`)
})

```

CSS to layout and style canvas element and error box.

styling.css

```
canvas#canvas{
  height: 80%;
  width: 80%;
  margin: 5% 10%;
}

.info_yellow{
  width: 80%;
  margin: auto 8%;
  padding: 2%;
  background: #eeee77;
  border: solid 1px #dddd66;
  border-radius: 5px;
}
```

And of course, the main Javascript file to call, which calls every other.

main.js

```
import { Canvas } from './modules/canvas.js';
import loader from './modules/loader.js';

let myCanvas;

window.addEventListener("load", function onLoad (evt) {
  "use strict"

  // Cleaning after ourselves. The event handler removes
  // itself, because it only needs to run once.
  window.removeEventListener(evt.type, onLoad, false);
  myCanvas = new Canvas('canvas');
  if (!myCanvas.WebGLAvailable){
    // Show fallback message
    let fallbackElement = document.getElementById('WebGL_Fallback');
    fallbackElement.hidden = false;
  }

  const runningGame = loader(myCanvas);
});
```

It sees if WebGL is available and reports an info box if the user's browser does not use WebGL.

Some code that almost works

Loader.js

The file with the game logic.

loader.js

```
1
2  import Map_API from '../modules/Map_API.js';
3  import {Object4D} from './Objects.js';
4
5  import map from '../maps/map1.js';
6
7  function loader(Canvas){
8
9    // Save meshes to renderer Canvas object.
10   let meshIndexes = Object.keys(map.meshes);
11   for (let i=0; i < meshIndexes.length; i++){
12     let index = meshIndexes[i];
13     let mesh = map.meshes[index];
14     Canvas.add_Indexed_Mesh(index, mesh.vertices, mesh.trianglesIndexed);
15   }
16
17   // For first object on map only (0)
18   let meshIndex = Map_API.get_Object_mesh_index(0, map)
19   let size = Map_API.get_Object_size(0, map)
20   let pvuw = Map_API.get_Object_pvuw(0, map)
21
22   // Try to find mesh in Canvas object
23   let meshBufferLocations = Canvas.findMesh(meshIndex);
```

loader.js

```
25    if (meshBufferLocations){
26      // Do this before drawing
27      function UnifromsAttribsAtDraw(gl, {attributeReferences}){
28        gl.bindBuffer(gl.ARRAY_BUFFER, meshBufferLocations.vertexBufferRef);
29        gl.enableVertexAttribArray(attributeReferences['coordinates']);
30        gl.vertexAttribPointer(attributeReferences['coordinates'], 4, gl.FLOAT, false, 0, 0);
31
32        // bind the buffer containing the indices
33        // this.gl.bindBuffer(gl.ARRAY_BUFFER, the_Mesh.vertexBufferRef);
34        gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, meshBufferLocations.indexBufferRef);
35      }
36
37      // Draw the object
38      Canvas.drawObject(UnifromsAttribsAtDraw, meshBufferLocations.trianglesIndexed.length);
39    }
40
41    // let success = Canvas.drawObject(meshIndex, size, pvuw)
42
43  }
44
45  function UnifromsAttribsAtDraw(gl, {meshBuffers}, {attributeReferences}){
46    gl.bindBuffer(gl.ARRAY_BUFFER, meshBuffers[meshIndex]);
47    gl.enableVertexAttribArray(attributeReferences['coordinates']);
48    gl.vertexAttribPointer(attributeReferences['coordinates'], 4, gl.FLOAT, false, 0, 0);
49  }
50
51
52  export default loader;
53
```

Canvas.js

Responsible for communicating with WebGL and setting up the HTML canvas.

canvas.js

```
import {vertexShader as vertexShaderSource} from '../shaders/vertexShader.js';
import {fragmentShader as fragmentShaderSource} from '../shaders/fragmentShader.js';

class Canvas {

  //____________________________________________________________________//

  // Set up canvas and make it black.
  constructor(canvas_HTML_ID){
    this.canvas = document.getElementById(canvas_HTML_ID);
    this.WebGLAvailable = this.getContext();
    if (!this.WebGLAvailable){
      return;
    }
    // Colour screen black
    this.clearColour();

    // Stores references to meshes that were input into the GPU
    // form of list item: {meshID: __, vertexBufferRef: ___ , indexBufferRef: ____ }
    this.meshBuffers = [];

    // Ref to shader program (to be made)
    this.program;
    this.attributeReferences = {};

  }
```

```
getContext(){
  // Sets up this.gl or returns false if unable to.
  // Try to access WEB GL
  this.gl = canvas.getContext("webgl");
  // Report back if you can't get webgl
  if (!this.gl) {
    return false;
  }
  // Set window dimensions
  this.gl.viewport(0, 0, this.gl.drawingBufferWidth, this.gl.drawingBufferHeight)
  return true;
}
clearColour(){
  this.gl.clearColor(0.0,0.0,0.0,1.0);
  this.gl.clear(this.gl.COLOR_BUFFER_BIT)
}

// ================================================================================//
// Thank you webglfundamentals.org for the lovely tutorials and boilerplate.  //
// ================================================================================//

/*
* Creates and compiles a shader.
*
* @param {string} shaderSource The GLSL source code string for the shader.
* @param {number} shaderType The type of shader, VERTEX_SHADER or
*     FRAGMENT_SHADER.
```

```
57    compileShader(shaderSource, shaderType) {
58
59      // Get rendering context
60      let gl = this.gl;
61
62      // Create the shader object
63      var shader = gl.createShader(shaderType);
64
65      // Set the shader source code.
66      gl.shaderSource(shader, shaderSource);
67
68      // Compile the shader
69      gl.compileShader(shader);
70
71      // Check if it compiled
72      var success = gl.getShaderParameter(shader, gl.COMPILE_STATUS);
73      if (!success) {
74        // Something went wrong during compilation; get the error
75        throw "could not compile shader:" + gl.getShaderInfoLog(shader);
76      }
77
78      return shader;
79    }
```

canvas.js

```
57    compileShader(shaderSource, shaderType) {
58
59      // Get rendering context
60      let gl = this.gl;
61
62      // Create the shader object
63      var shader = gl.createShader(shaderType);
64
65      // Set the shader source code.
66      gl.shaderSource(shader, shaderSource);
67
68      // Compile the shader
69      gl.compileShader(shader);
70
71      // Check if it compiled
72      var success = gl.getShaderParameter(shader, gl.COMPILE_STATUS);
73      if (!success) {
74        // Something went wrong during compilation; get the error
75        throw "could not compile shader:" + gl.getShaderInfoLog(shader);
76      }
77
78      return shader;
79    }
```

```
88    createProgram() {
89      let gl = this.gl;
90
91      // create a program.
92      var program = gl.createProgram();
93
94      // compile and attach the shaders.
95      // (SOURCES IMPORTED FROM EXTERNAL FILES)
96      var vertexShader = this.compileShader(vertexShaderSource);
97      gl.attachShader(program, vertexShader);
98      var fragmentShader = this.compileShader(fragmentShaderSource);
99      gl.attachShader(program, fragmentShader);
100
101     // link the program.
102     gl.linkProgram(program);
103
104     // Check if it linked.
105     var success = gl.getProgramParameter(program, gl.LINK_STATUS);
106     if (!success) {
107         // something went wrong with the link
108         throw ("program failed to link:" + gl.getProgramInfoLog (program));
109     }
110
111     this.program = program;
112   };
```

```
116   // TEST with invalid dictionary
117   add_AttributeReference(KeyToVarDict){
118     // in { key : Shader Variable Name String } dictionary
119     // out { key : Attribute Location Reference } dictionary
120     // saves in this.attributeReferences
121
122     // For every attribute
123     let keys = Object.keys(KeyToVarDict);
124     for (let i=0; i < keys.length; i++){
125       let key = keys[i];
126       let ShaderAttributeName = KeyToVarDict[key];
127       // Add the location of the attribute
128       this.attributeReferences[key] = this.gl.getAttribLocation(this.program, ShaderAttributeName);
129     }
130   }
131
132
133
134   add_Indexed_Mesh(ID, vertices, trianglesIndexed){
135     let gl = this.gl;
136
137     // Create and bind buffer, then add vertex coordinates.
138     const vertexBuffer = gl.createBuffer();
139     gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
140     gl.bufferData(
141       gl.ARRAY_BUFFER,
142       new Float32Array(vertices),
143       gl.STATIC_DRAW
144     )
```

```
146     // Create and bind index buffer.
147     const indexBuffer = gl.createBuffer();
148     gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, indexBuffer);
149     // Fill the current element array buffer with data.
150     gl.bufferData(
151         gl.ELEMENT_ARRAY_BUFFER,
152         new Uint16Array(trianglesIndexed),
153         gl.STATIC_DRAW
154     );
155
156     this.meshBuffers.push({meshID: ID, vertexBufferRef: vertexBuffer , indexBufferRef: indexBuffer });
157
158     // At draw time we need to bind whatever buffer holds the indices we want to use.
159     // bind the buffer containing the indices
160     // gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, indexBuffer);
161
162     // When drawing
163     // var primitiveType = gl.TRIANGLES;
164     // var offset = 0;
165     // var count = 6;
166     // gl.drawArrays(primitiveType, offset, count);
167     // var indexType = gl.UNSIGNED_SHORT;
168     // gl.drawElements(primitiveType, count, indexType, offset);
169   }
```

```
171     findMesh(meshID){
172       // Find buffers to use
173       // using linear search
174       let found = false;
175       let meshIndex = 0;
176       let the_Mesh;
177       while ((found !== true) && (meshIndex < this.meshBuffers.length)){
178         the_Mesh = this.meshBuffers[meshIndex];
179         if (the_Mesh.mesh_ID === meshID){
180           found = true
181           return the_Mesh;
182         }
183       }
184       if (found !== true){
185         console.log("Mesh was not found. Hey, programmer! Seems like there's a little mistake!")
186         return undefined;
187       }
188     }
189
```

```
193    drawObject(beforeDraw, numElementsToDraw){
194
195      // Call the requested function to link correct buffers
196      beforeDraw(this.gl, {attributeReferences: this.attributeReferences});
197
198      let primitiveType = gl.TRIANGLES;
199      let offset = 0;
200      // Number of vertices
201      let indexType = gl.UNSIGNED_SHORT;
202      this.gl.drawElements(primitiveType, count, indexType, offset);
203
204      // gl.drawArrays(primitiveType, offset, count);
205
206      // Success
207      return true;
208    }
209  }
210
211
212  export { Canvas }
213
```

Planet.js.

Stores planet mesh. Contains some random data as a prototype right now.

planet.js

```
const planet_vertices = [
  0.0, 0.1, 0.2, 0.5,
  0.1, 0.2, 0.1, 0.6,
  0.3, 0.0, -0.2, 0.7,
  0.2, -0.2, -0.3, 0.8,
  0.1, 0.1, 0.1, 0.9,
]

const planet_trianglesIndexed = [
  1, 2, 3,
  2, 3, 4,
  1, 2, 4,
  4, 3, 1
]

const Model_planet = { vertices: planet_vertices, trianglesIndexed: planet_trianglesIndexed };

export default Model_planet;
```

Map1.js

Stores the objects on the map, and the meshes associated with them.

canvas.html | canvas.js | *map1.js*

```
import Model_planet from '../models/planet.js';
const map = {
  meshes: {
    0: Model_planet,
  },
  objects: [
    {
      meshIndex: 0,
      pvuw: [
        [0.0, 1.0, 0.0, 0.0],
        [1.0, 0.0, 0.0, 0.0],
        [0.0, 0.0, 0.0, 1.0],
        [0.0, 0.0, 1.0, 0.0]
      ],
      size: 0.5,
    }
  ]
}

export default map;

```

Map_API.js

A set of useful functions for handling map objects.

Map_API.js

```
function get_Object_mesh_index(objectIndex, map){
  return map.objects[objectIndex].meshIndex;
}
function get_Object_size(objectIndex, map){
  return map.objects[objectIndex].size;
}
function get_Object_pvuw(objectIndex, map){
  return map.objects[objectIndex].pvuw;
}

function get_Object(objectIndex, map){
  return map.objects[objectIndex];
}

const Map_API = {get_Object, get_Object_mesh_index, get_Object_pvuw, get_Object_size};

export default Map_API;
```

Other files

Some files, like models.js , are currently empty.

Testing.

When testing, the window remains blank, and the loader keeps going. There is a slow down in the sensitivity of the mouse. This seems to imply that an infinite loop is taking over the processing power.

I have found the error to be here:

```
173   findMesh(meshID){
174     // Find buffers to use
175     // using linear search
176     let found = false;
177     let meshIndex = 0;
178     let the_Mesh;
179     while ((found !== true) && (meshIndex < this.meshBuffers.length)){
180       the_Mesh = this.meshBuffers[meshIndex];
181       if (the_Mesh.mesh_ID === meshID){
182         found = true
183         return the_Mesh;
184       }
185     }
186     if (found !== true){
187       console.log("Mesh was not found. Hey, programmer! Seems like there's a little mistake!")
188       return undefined;
189     }
190   }
```

Correction: add meshIndex = meshIndex + 1 at the end of the loop.

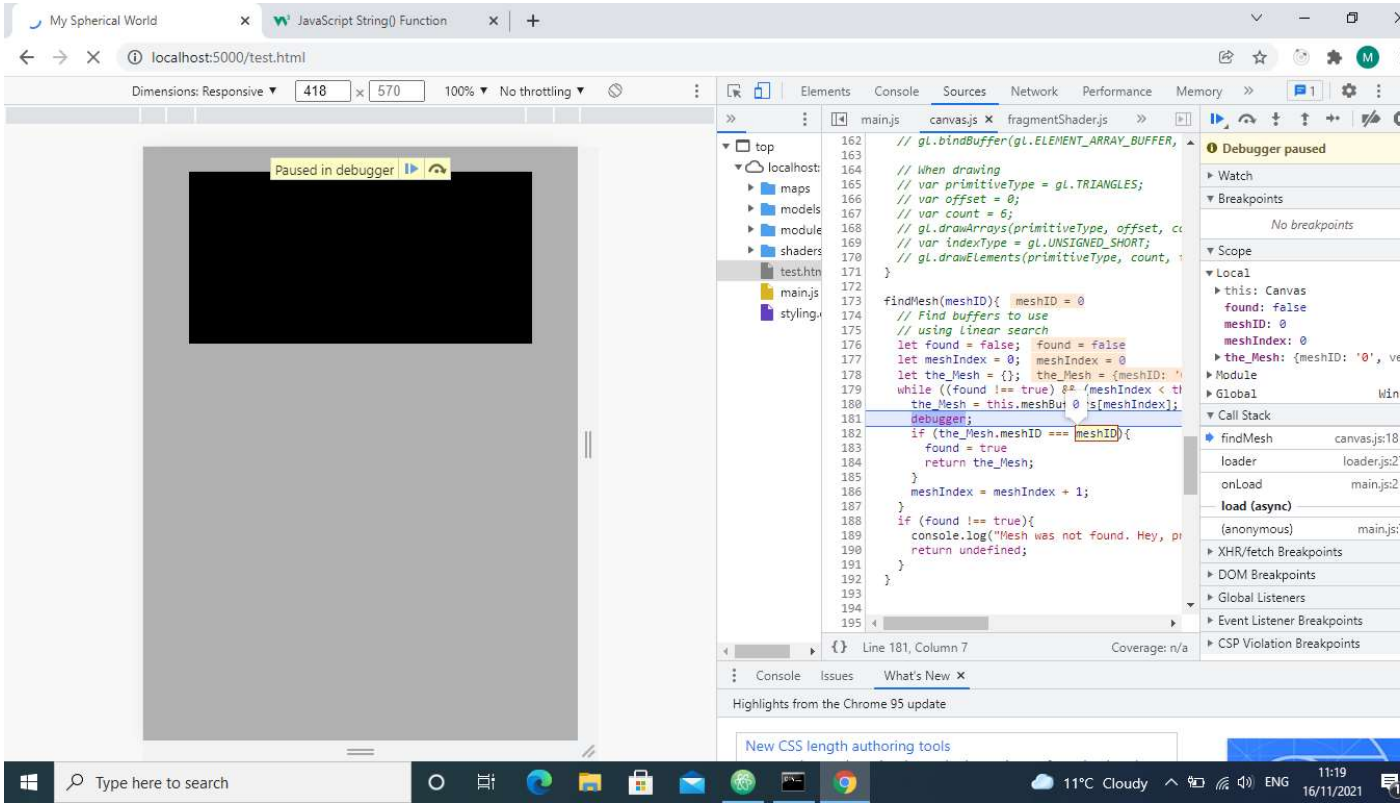
Another error:

```
Created                                                                    canvas.js:28
Mesh was not found. Hey, programmer! Seems like there's a little mistake!  canvas.js:189
Done                                                                       main.js:22
>
```

Correction: meshID instead of mesh_ID.

```
173   findMesh(meshID){
174     // Find buffers to use
175     // using linear search
176     let found = false;
177     let meshIndex = 0;
178     let the_Mesh = {};
179     while ((found !== true) && (meshIndex < this.meshBuffers.length)){
180       the_Mesh = this.meshBuffers[meshIndex];
181       if (the_Mesh.meshID === meshID){
182         found = true
183         return the_Mesh;
184       }
185       meshIndex = meshIndex + 1;
186     }
187     if (found !== true){
188       console.log("Mesh was not found. Hey, programmer! Seems like there's a little mistake!")
189       return undefined;
190     }
191   }
```

Error: Same. Checking debugger:



One is a string but one is an integer. They are not equal.

Correction: toString()

```
findMesh(meshID){
  // Find buffers to use
  // using linear search
  let found = false;
  let meshIndex = 0;
  let the_Mesh = {};
  while ((found !== true) && (meshIndex < this.meshBuffers.length)){
    the_Mesh = this.meshBuffers[meshIndex];
    debugger;
    if (the_Mesh.meshID === meshID.toString()){
      found = true
      return the_Mesh;
    }
    meshIndex = meshIndex + 1;
  }
  if (found !== true){
    console.log("Mesh was not found. Hey, programmer! Seems like there's a little mistake!")
    return undefined;
  }
}
```

Added a fragment shader to colour further objects darker, just to test.

Outcome:

More coherent vertex and fragment shader testing

I have made a tetrahedron of width 1.0 as a test polyhedron for the algorithm. I am going to check whether it renders correctly.

```
const planet_vertices = [
   0.5,  0.0,  0.0,
  -0.5,  0.0,  0.0,
   0.0,  0.5,  0.0,
   0.0, -0.5,  0.0,
   0.0,  0.0,  0.5,
   0.0,  0.0, -0.5
]

const planet_trianglesIndexed = [
  0, 2, 4,
  0, 2, 5,
  0, 3, 4,
  0, 3, 5,
  1, 2, 4,
  1, 2, 5,
  1, 3, 4,
  1, 3, 5,
]
```

I made a new method Canvas.add_UniformReference to allow the program to fetch the GPU locations (addresses) of uniforms to write data to the uniform when needed.

```
  // TEST with invalid dictionary
  add_UniformReference(KeyToVarDict){
    // in { key : Shader Variable Name String } dictionary
    // out { key : Uniform Location Reference } dictionary
    // saves in this.uniformReferences

    // For every uniform
    let keys = Object.keys(KeyToVarDict);
    for (let i=0; i < keys.length; i++){
      let key = keys[i];
      let ShaderUniformName = KeyToVarDict[key];
      // Add the location of the uniform
      this.uniformReferences[key] = this.gl.getUniformLocation(this.program, ShaderUniformName);
    }
  }

```

Loader.js was modified to include the new uniforms - time and a rotation matrix based on time.

It also now has a function loop(), that runs every few milliseconds to update the scene. It only updates after 16ms have passed.

main.js | loader.js | vertexShader.js

```
21
22    Canvas.add_AttributeReference({
23      'coordinates': 'coordinates'
24    })
25    Canvas.add_UniformReference({
26      'time': 'time',
27      'rotationZX': 'rotationZX',
28    })
29
30    // Try to find mesh in Canvas object
31    let meshBufferLocations = Canvas.findMesh(meshIndex);
32    let numPoints = Map_API.get_Mesh_NumPoints(meshIndex, map);
33    const frameRate = 16; // milliseconds per frame
34
35    if (meshBufferLocations){
36      let startTime = Date.now();
37      let previousTime = Date.now();
38
```

```
39      // Attributes do not vary so this function can be defined once.
40      function At_Draw_2__setAttribs(canvasThis){
41        let gl = canvasThis.gl;
42        //canvasThis.gl.bindBuffer(gl.ARRAY_BUFFER, meshBufferLocations.vertexBufferRef);
43        // bind the buffer containing the indices
44        gl.bindBuffer(gl.ARRAY_BUFFER, meshBufferLocations.vertexBufferRef);
45        gl.enableVertexAttribArray(canvasThis.attributeReferences['coordinates']);
46        gl.vertexAttribPointer(canvasThis.attributeReferences['coordinates'], 3, gl.FLOAT, false, 0, 0);
47
48        gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, meshBufferLocations.indexBufferRef);
49
50        gl.enable(gl.DEPTH_TEST);
51
52        gl.useProgram(canvasThis.program);
53      }
54
55
56      function loop(){
57
58        let timeNow = Date.now();
59        let dt = timeNow - previousTime;
60        let elapsedTime = timeNow - startTime;
61
62        if (dt >= frameRate){
63
64          // Uniforms vary so must be recalculated every loop
```

```
65          function At_Draw_1__setUniforms(canvasThis){
66              let gl = canvasThis.gl;
67              gl.uniform1f(canvasThis.uniformReferences['time'], elapsedTime);
68              gl.uniformMatrix3fv(canvasThis.uniformReferences['rotationZX'], false, [
69                Math.cos(elapsedTime/1000), 0, Math.sin(elapsedTime/1000),
70                0, 1, 0,
71                Math.sin(-elapsedTime/1000), 0, Math.cos(elapsedTime/1000)
72              ])
73          }
74
75          function At_Draw(canvasThis){
76            At_Draw_1__setUniforms(canvasThis)
77            At_Draw_2__setAttribs(canvasThis)
78          }
79
80          Canvas.drawObject(At_Draw, numPoints);
81
82
83            previousTime = timeNow;
84        }
85        // loop after some time
86        requestAnimationFrame(loop)
87      }
88
89      requestAnimationFrame(loop);
90    }
91
92    return true;
93  }
```

The vertex shader rotates the starting coordinates by the matrix rotationZX, and sinusoidally displaces the shape from the origin with time.

main.js | loader.js | vertexShader.js

```
const vertexShader = `

  attribute vec3 coordinates;
  uniform float time;
  uniform mat3 rotationZX;
  varying vec3 coords;
  void main() {
    vec3 c_2 = rotationZX * coordinates;
    gl_Position = vec4(c_2.x + rotationZX[2][0]/3.0, c_2.y, c_2.z, 1.0);
    coords = coordinates;

  }
`;

export {vertexShader};
```

The fragment shader uses the coordinates of a point to colour in the shape. Gl_FragColor is in RGBA mode, with floats from 0.0 to 1.0 acting as colours.

main.js | loader.js | vertexShader.js | fragmentShader.js

```
const fragmentShader = `
  precision highp float;

  varying vec3 coords;

  void main(){
      float x = coords.z;
      vec3 color = vec3((coords + 1.0)/2.0);
      gl_FragColor = vec4(color.x, color.y, color.z, 1.0);
  }

`;

export {fragmentShader};
```

The result is almost as expected. It is a rotating diamond. **The thing I didn't expect** is that WebGL stretches the shape to the size of the canvas, so gl_Position is actually the % position from the centre, resulting in a stretched shape.

To solve this, I can specify the screen width and height into the vertex shader. I will do this when I actually have a meaningful vertex shader.

Code Refactoring

I have refactored my code to be able to draw multiple objects easily.

loader.js

```
import Map_API from '../modules/Map_API.js';
import {Object4D} from './Objects.js';

import map from '../maps/map1.js';

function loader(Canvas){

  // Save meshes to renderer Canvas object.
  // For every mesh in the mesh dictionary
  let meshIndexes = Object.keys(map.meshes);
  for (let i=0; i < meshIndexes.length; i++){
    let index = meshIndexes[i];
    let mesh = map.meshes[index];
    Canvas.add_Indexed_Mesh(index, mesh.vertices, mesh.trianglesIndexed);
  }
  // Set up attributes and uniforms references
  Canvas.add_AttributeReference({
    'coordinates': 'coordinates'
  })
  Canvas.add_UniformReference({
    'time': 'time',
    'rotationZX': 'rotationZX',
    'object_pvuw': 'object_pvuw',
    'camera_pvuw': 'camera_pvuw'
  })

// Turn map objects into list of those ready for processing
  let amount_of_objects = Map_API.get_num_Objects(map);
  let processed_objects = []
```

```
// Turn map objects into list of those ready for processing
  let amount_of_objects = Map_API.get_num_Objects(map);
  let processed_objects = []
  for (let i=0; i<amount_of_objects; i++){
    // Find mesh reference in preloaded meshes
    let meshIndex = Map_API.get_Object_mesh_index(i, map)
    let meshBufferLocations = Canvas.findMesh(meshIndex);
    if (meshBufferLocations){
      // Only add if mesh was found
      // map_object_index to access map properties like position
      // meshBufferLocations to access gl vertex rendering function
      processed_objects.push({
        map_object_index: i,
        meshBufferLocations: meshBufferLocations
      })
    }
  }

  // Player with no mesh but acting as a camera
  let Player_pvuw =
  [
    0.0, 0.0, 0.0, 1.0,
    1.0, 0.0, 0.0, 0.0,
    0.0, -1.0, 0.0, 0.0,
    0.0, 0.0, -1.0, 0.0,
  ]

```

```
  let startTime = Date.now();
  let previousTime = Date.now();
  const frameRate = 16; // milliseconds per frame

  // Set up the program here
  function At_Draw_1__generalSetup(canvasThis){
      let gl = canvasThis.gl;
      gl.useProgram(canvasThis.program);
      gl.enable(gl.DEPTH_TEST);
  }
  // Attributes are lists of vertices to render
  function At_Draw_2__setAttribs(canvasThis, {vertexBufferRef, indexBufferRef}){
    let gl = canvasThis.gl;

    // vertex buffer
    gl.bindBuffer(gl.ARRAY_BUFFER, vertexBufferRef);
    gl.enableVertexAttribArray(canvasThis.attributeReferences['coordinates']);
    gl.vertexAttribPointer(canvasThis.attributeReferences['coordinates'], 3, gl.FLOAT, false, 0, 0);

    // Index buffer
    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, indexBufferRef);
  }
```

```
77    // Uniforms vary so must be recalculated every loop
78    // Time based Uniforms
79    function At_Draw_3a__setUniforms(canvasThis, {timeFromStart}){
80        let gl = canvasThis.gl;
81        // Pass in uniforms
82        gl.uniform1f(canvasThis.uniformReferences['time'], timeFromStart);
83        gl.uniformMatrix3fv(canvasThis.uniformReferences['rotationZX'], false, [
84          Math.cos(timeFromStart/1000), 0, Math.sin(timeFromStart/1000),
85          0, 1, 0,
86          Math.sin(-timeFromStart/1000), 0, Math.cos(timeFromStart/1000)
87        ])
88    }
89    // Player position
90    function At_Draw_3b__setUniforms_Player(canvasThis, {Player_pvuw}){
91      let gl = canvasThis.gl;
92      gl.uniformMatrix4fv(canvasThis.uniformReferences['camera_pvuw'], false, Player_pvuw);
93    }
94    // Object position
95    function At_Draw_3c__setUniforms_Object(canvasThis, {Object_pvuw}){
96      let gl = canvasThis.gl;
97      gl.uniformMatrix4fv(canvasThis.uniformReferences['object_pvuw'], false, Object_pvuw);
98    }
```


```
77    // Uniforms vary so must be recalculated every loop
78    // Time based Uniforms
79    function At_Draw_3a__setUniforms(canvasThis, {timeFromStart}){
80        let gl = canvasThis.gl;
81        // Pass in uniforms
82        gl.uniform1f(canvasThis.uniformReferences['time'], timeFromStart);
83        gl.uniformMatrix3fv(canvasThis.uniformReferences['rotationZX'], false, [
84          Math.cos(timeFromStart/1000), 0, Math.sin(timeFromStart/1000),
85          0, 1, 0,
86          Math.sin(-timeFromStart/1000), 0, Math.cos(timeFromStart/1000)
87        ])
88    }
89    // Player position
90    function At_Draw_3b__setUniforms_Player(canvasThis, {Player_pvuw}){
91      let gl = canvasThis.gl;
92      gl.uniformMatrix4fv(canvasThis.uniformReferences['camera_pvuw'], false, Player_pvuw);
93    }
94    // Object position
95    function At_Draw_3c__setUniforms_Object(canvasThis, {Object_pvuw}){
96      let gl = canvasThis.gl;
97      gl.uniformMatrix4fv(canvasThis.uniformReferences['object_pvuw'], false, Object_pvuw);
98    }
```

```
    const Generate_At_Draw = ({vertexBufferRef, indexBufferRef, Player_pvuw, Object_pvuw, timeFromStart}) => (canvasThis) =>
      At_Draw_1__generalSetup(canvasThis)
      At_Draw_2__setAttribs(canvasThis, { vertexBufferRef: vertexBufferRef, indexBufferRef: indexBufferRef })
      At_Draw_3a__setUniforms(canvasThis, {timeFromStart: timeFromStart})
      At_Draw_3b__setUniforms_Player(canvasThis, {Player_pvuw: Player_pvuw})
      At_Draw_3c__setUniforms_Object(canvasThis, {Object_pvuw: Object_pvuw})
    }


    // Define the rendering loop function
    function loop(){

      let timeNow = Date.now();
      let dt = timeNow - previousTime;
      let elapsedTime = 0.2*(timeNow - startTime);

      if (dt >= frameRate){

        let At_Draw;
        // Render every object
        for (let i=0; i< processed_objects.length; i++){
          let object_for_drawing = processed_objects[i];
          At_Draw = Generate_At_Draw({
            vertexBufferRef: object_for_drawing.meshBufferLocations.vertexBufferRef,
            indexBufferRef: object_for_drawing.meshBufferLocations.indexBufferRef,
            Player_pvuw: Player_pvuw,
            Player_pvuw: Player_pvuw,
            Object_pvuw: Map_API.get_Object_pvuw(object_for_drawing.map_object_index, map),
            timeFromStart: elapsedTime
          })
          Canvas.drawObject(At_Draw, object_for_drawing.meshBufferLocations.metadata.numPoints);
        }

        previousTime = timeNow;
      }
      // Loop after some time
      requestAnimationFrame(loop)
    }
    // Call loop to start rendering
    requestAnimationFrame(loop);

    return true;
  }
```

Explanation of commands called just before drawing to screen every frame.

At_Draw_1__generalSetup (canvasThis)

+ Sets up general WebGL settings.

At_Draw_2__setAttribs(canvasThis, {vertexBufferRef, indexBufferRef})

+ Tells WebGL which array of points and which array of triangles to use to draw each object.

At_Draw_3a__setUniforms(canvasThis, {timeFromStart})

+ Sets the things invariant for all points of all objects. For example, time, or orientation of object.

At_Draw_3b__setUniforms_Player (canvasThis, {Player_pvuw})

+ Sets the things invariant to do with the camera.

At_Draw_3c__setUniforms_Object (canvasThis, {Object_pvuw})

+ Sets the things invariant to do with each object.

These are combined using Generate_At_Draw for every object every draw call. All the parameters for the At_Draw_x functions are passed into this. It returns a function At_Draw to which the canvas object can be passed.

This function is passed (as beforeDraw) to Canvas.drawObject(beforeDraw, numPoints)

```
234    drawObject(beforeDraw, numElementsToDraw){
235
236      // Call the requested function to link correct buffers
237      beforeDraw(this);
238
239      let primitiveType = this.gl.TRIANGLES;
240      let offset = 0;
241      // Number of vertices
242      let indexType = this.gl.UNSIGNED_SHORT;
243      this.gl.drawElements(primitiveType, numElementsToDraw, indexType, offset);
244
```

Adding a 4D Sphere Vertex Shader

vertexShader.js

```
const vertexShader = `
  attribute vec3 coordinates;
  uniform float time;
  uniform mat3 rotationZX;
  uniform mat4 camera_pvuw;
  uniform mat4 object_pvuw;
  varying vec3 coords;

  float magnitude(vec4 vector){
    return sqrt(vector.x*vector.x + vector.y*vector.y + vector.z*vector.z + vector.w*vector.w);
  }

  float det(mat2 matrix){
    return ( matrix[0].x*matrix[1].y - matrix[0].y*matrix[1].x ) ;
  }

  mat3 inverse(mat3 matrix) {
      vec3 row0 = matrix[0];
      vec3 row1 = matrix[1];
      vec3 row2 = matrix[2];

      vec3 minors0 = vec3(
          det(mat2(row1.y, row1.z, row2.y, row2.z)),
          det(mat2(row1.z, row1.x, row2.z, row2.x)),
          det(mat2(row1.x, row1.y, row2.x, row2.y))
      );
      vec3 minors1 = vec3(
          det(mat2(row2.y, row2.z, row0.y, row0.z)),
          det(mat2(row2.z, row2.x, row0.z, row0.x)),
```

```
            det(mat2(row2.x, row2.y, row0.x, row0.y))
        );
        vec3 minors2 = vec3(
            det(mat2(row0.y, row0.z, row1.y, row1.z)),
            det(mat2(row0.z, row0.x, row1.z, row1.x)),
            det(mat2(row0.x, row0.y, row1.x, row1.y))
        );

        mat3 adj;
        adj[0] =vec3(minors0.x, minors1.x, minors2.x);
        adj[1] =vec3(minors0.y, minors1.y, minors2.y);
        adj[2] =vec3(minors0.z, minors1.z, minors2.z);

        return (1.0 / dot(row0, minors0)) * adj;
    }

    void main() {

      vec3 c = rotationZX * coordinates;

      vec4 direction = object_pvuw * vec4(0.0, c.xyz);
      direction = direction / magnitude(direction);
      float Distance = magnitude(vec4(0.0,c));
      vec4 center = object_pvuw[0].xyzw;
      float cosD = cos(6.28+Distance*3.14);
      float sinD = sin(6.28+Distance*3.14);
      vec4 Vertex_4D_coordinates = (cosD * center) + (sinD * direction);
```

vertexShader.js

```
58      // Calculating distance and direction player to object
59      vec4 P1 = camera_pvuw[0].xyzw;
60      vec4 P2 = Vertex_4D_coordinates;
61      cosD = dot(P1, P2);
62      sinD = sqrt(1.0-cosD*cosD);
63      vec4 player_to_object = (P2 - (P1 * cosD))/( sinD );
64      if (dot(player_to_object, camera_pvuw[1].xyzw) < 0.0) {
65        // Facing wrong way
66        player_to_object = -player_to_object;
67      }
68
69      // Calculate the 3D direction from player to object
70      mat3 concatenated_axes;
71      concatenated_axes[0] = camera_pvuw[1].xyz;
72      concatenated_axes[1] = camera_pvuw[2].xyz;
73      concatenated_axes[2] = camera_pvuw[3].xyz;
74      mat3 invConcatAxes = inverse(concatenated_axes);
75      vec3 abc = invConcatAxes * player_to_object.xyz;
76
77      // Calculate screen coordinates
78      // ...based on distance to screen
79      float dist = 1.0;
80      float xy_ratio = 2.0;
81      float screen_x_coord = ( abc.y / (abc.x * dist) ) / xy_ratio;
82      float screen_y_coord = ( abc.z / (abc.x * dist) );
83      float screen_z_coord = 0.5;
84
85      // gl_Position = vec4(c, 1.0);
86      gl_Position = vec4(screen_x_coord, screen_y_coord, screen_z_coord, 1.0);
87
88      coords = coordinates;
89    }
90  `;
91
92  export {vertexShader};
```

xy_ratio is a constant and is used to handle the **stretching bug** described earlier.

I will later make it change based on the canvas width and height.

This vertex shader displays the objects passed into the vertex shader, rotating and viewed from the Player_pvuw 4D coordinates, as if on a 4D sphere.

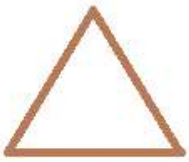
I can uncomment line 85 to view the shape in normal 3D coordinates.

Program to generate spherical objects

I made a program to generate a more spherical looking object of specified radius and smoothness.

It uses this algorithm...

1. Start with a double square based pyramid.



2. For every triangle in the structure...

3. Select the midpoints of the lines making the triangle.

4. Extrude midpoints to sphere.

5. Make four triangles using these new vertices, and the old vertices.

6. Repeat the process for these new triangles to make the shape smoother and smoother.

I have programmed it in python.

```
28  # ========================================================= #
29
30  NUM_ITERATIONS = 1
31  RADIUS = 0.1
32
33  # Use iterative vertex generation method with triangles
34  # Every triangle becomes four triangles
35
36  # Vertex Position (3D) array. Every 3 floats are one coordinate.
37  _v = [
38      0.0,  0.0, RADIUS,
39      0.0,  RADIUS,  0.0,
40      RADIUS, 0.0, 0.0,
41      0.0,  0.0, -RADIUS,
42      0.0,  -RADIUS,  0.0,
43      -RADIUS, 0.0, 0.0,
44  ]
```

```
46  # Triangles made using 3 indexed points. Indexes to _v array.
47  _i = [
48      0, 1, 2,
49      3, 2, 1,
50      2, 4, 0,
51      1, 0, 5,
52      1, 5, 3,
53      4, 5, 0,
54      5, 4, 3,
55      2, 3, 4,
56  ]
57
58  # Above is a starting solid
59  # Below is excitement!
60
61  for hmm in range(NUM_ITERATIONS):
62      # Make a new point for every pair of connected vertices
63      # lines_visited stores [[v1, v2, newVertexIndex]...]
64      lines_visited = []
65      _i_new = []
66      for tri_index in range(len(_i)//3):
67          # For every side of triangle, check and make new vertex.
68          # Store their indexes temporarily here as [[v1, v2, vN]...]
69          midpoints = []
70          for connection in [0, 1, 2]:
71              line = [_i[3*tri_index + (connection)%3], _i[3*tri_index + (connection+1)%3]]
72              result = find_line_found_vN_reversed(lines_visited, line)
```

```
            if result[0] == True: # found
                vN = result[1] # mid vertex index
            else: # not found
                vN_coords = generate_new_vertex_coords_from_other_two(
                    [_v[3*line[0]], _v[3*line[0] + 1], _v[3*line[0] + 2]],
                    [_v[3*line[1]], _v[3*line[1] + 1], _v[3*line[1] + 2]],
                    RADIUS)
                vN = len(_v)//3
                _v.append(vN_coords[0])
                _v.append(vN_coords[1])
                _v.append(vN_coords[2])
                lines_visited.append([line[0],line[1], vN])
                pass
            # Add to list of midpoints of this triangle for next part
            midpoints.append([line[0], line[1], vN])

        # Now make new triangles
        # Middle Triangle
        _i_new.append(midpoints[0][2])
        _i_new.append(midpoints[1][2])
        _i_new.append(midpoints[2][2])
        # Triangle 1
        _i_new.append(midpoints[0][0])
        _i_new.append(midpoints[0][2])
        _i_new.append(midpoints[2][2])
        # Triangle 2
        _i_new.append(midpoints[1][0])
        _i_new.append(midpoints[1][2])
        _i_new.append(midpoints[0][2])
        # Triangle 3
        _i_new.append(midpoints[2][0])
        _i_new.append(midpoints[2][2])
        _i_new.append(midpoints[1][2])
    _i = _i_new

print(_v)
print(_i)

```

```
def find_Line_found_vN_reversed(v1v2vN_list, line):
    # Tries to find the line or the line reversed in a list of lines.
    # Returns found, the vertex associated with the line, whether the line is stored reversed.
    found = False
    reversed = False
    vertex = -1

    for v1v2vN in v1v2vN_list:
        if (v1v2vN[0] == line[0]) and (v1v2vN[1] == line[1]):
            found = True
            reversed = False
            vertex = v1v2vN[2]

        elif (v1v2vN[0] == line[1]) and (v1v2vN[1] == line[0]):
            found = True
            reversed = False
            vertex = v1v2vN[2]

    return [found, vertex, reversed]

def generate_new_vertex_coords_from_other_two(v1, v2, radius):
    mid = [(v1[0]+v2[0]), (v1[1]+v2[1]), (v1[2]+v2[2])]
    magnitude = (mid[0]**2 + mid[1]**2 + mid[2]**2)**(0.5)
    sf = radius/magnitude
    return [sf*mid[0], sf*mid[1], sf*mid[2]]

```

NUM_ITERATIONS is the number of times to run the loop for, and corresponds to the smoothness of the shape.

RADIUS is the maximum radius of the resulting object.

The program turns one array of triangles into a completely new one every iteration.

Here is the output for radius=0.1, and 1 iteration. First is the vertex array, and second is the triangle index array.

```
C:\Users\Maxicl\code\nice\A-spherical-world\code1>cd programs

C:\Users\Maxicl\code\nice\A-spherical-world\code1\programs>python3 makeasphere.py
[0.0, 0.0, 0.1, 0.0, 0.1, 0.0, 0.1, 0.0, 0.0, 0.0, 0.0, -0.1, 0.0, -0.1, 0.0, -0.1,
0.0, 0.0, 0.0, 0.07071067811865475, 0.07071067811865475, 0.07071067811865475, 0.0707
1067811865475, 0.0, 0.07071067811865475, 0.0, 0.07071067811865475, 0.070710678118654
75, 0.0, -0.07071067811865475, 0.0, 0.07071067811865475, -0.07071067811865475, 0.070
71067811865475, -0.07071067811865475, 0.0, 0.0, -0.07071067811865475, 0.070710678118
65475, -0.07071067811865475, 0.0, 0.07071067811865475, -0.07071067811865475, 0.07071
067811865475, 0.0, -0.07071067811865475, 0.0, -0.07071067811865475, -0.0707106781186
5475, -0.07071067811865475, 0.0, 0.0, -0.07071067811865475, -0.07071067811865475]
[6, 7, 8, 0, 6, 8, 1, 7, 6, 2, 8, 7, 9, 7, 10, 3, 9, 10, 2, 7, 9, 1, 10, 7, 11, 12,
8, 2, 11, 8, 4, 12, 11, 0, 8, 12, 6, 13, 14, 1, 6, 14, 0, 13, 6, 5, 14, 13, 14, 15,
10, 1, 14, 10, 5, 15, 14, 3, 10, 15, 16, 13, 12, 4, 16, 12, 5, 13, 16, 0, 12, 13, 16
, 17, 15, 5, 16, 15, 4, 17, 16, 3, 15, 17, 9, 17, 11, 2, 9, 11, 3, 17, 9, 4, 11, 17]
```

The start of the output for 2 iterations.

```
C:\Users\Maxicl\code\nice\A-spherical-world\code1\programs>
C:\Users\Maxicl\code\nice\A-spherical-world\code1\programs>
C:\Users\Maxicl\code\nice\A-spherical-world\code1\programs>python3 makeasphere.py
[0.0, 0.0, 0.1, 0.0, 0.1, 0.0, 0.1, 0.0, 0.0, 0.0, 0.0, -0.1, 0.0, -0.1, 0.0, -0.1, 0.0, 0.0, 0.0, 0.070
71067811865475, 0.07071067811865475, 0.07071067811865475, 0.07071067811865475, 0.0, 0.07071067811865475,
 0.0, 0.07071067811865475, 0.07071067811865475, 0.0, -0.07071067811865475, 0.0, 0.07071067811865475, -0.
07071067811865475, 0.07071067811865475, -0.07071067811865475, 0.0, 0.0, -0.07071067811865475, 0.07071067
811865475, -0.07071067811865475, 0.0, 0.07071067811865475, -0.07071067811865475, 0.07071067811865475, 0.
0, -0.07071067811865475, 0.0, -0.07071067811865475, -0.07071067811865475, -0.07071067811865475, 0.0, 0.0
, -0.07071067811865475, -0.07071067811865475, 0.040824829046386304, 0.08164965809277261, 0.0408248290463
86304, 0.08164965809277261, 0.040824829046386304, 0.040824829046386304, 0.040824829046386304, 0.04082482
9046386304, 0.08164965809277261, 0.0, 0.038268343236508975, 0.09238795325112868, 0.038268343236508975, 0
.0, 0.09238795325112868, 0.038268343236508975, 0.09238795325112868, 0.0, 0.0, 0.09238795325112868, 0.038
268343236508975, 0.09238795325112868, 0.0, 0.038268343236508975, 0.09238795325112868, 0.0382683432365089
75, 0.0, 0.08164965809277261, 0.040824829046386304, -0.040824829046386304, 0.040824829046386304, 0.08164
```

I have put the vertex and index arrays into models/planet.js . I will use the 1-iteration sphere.

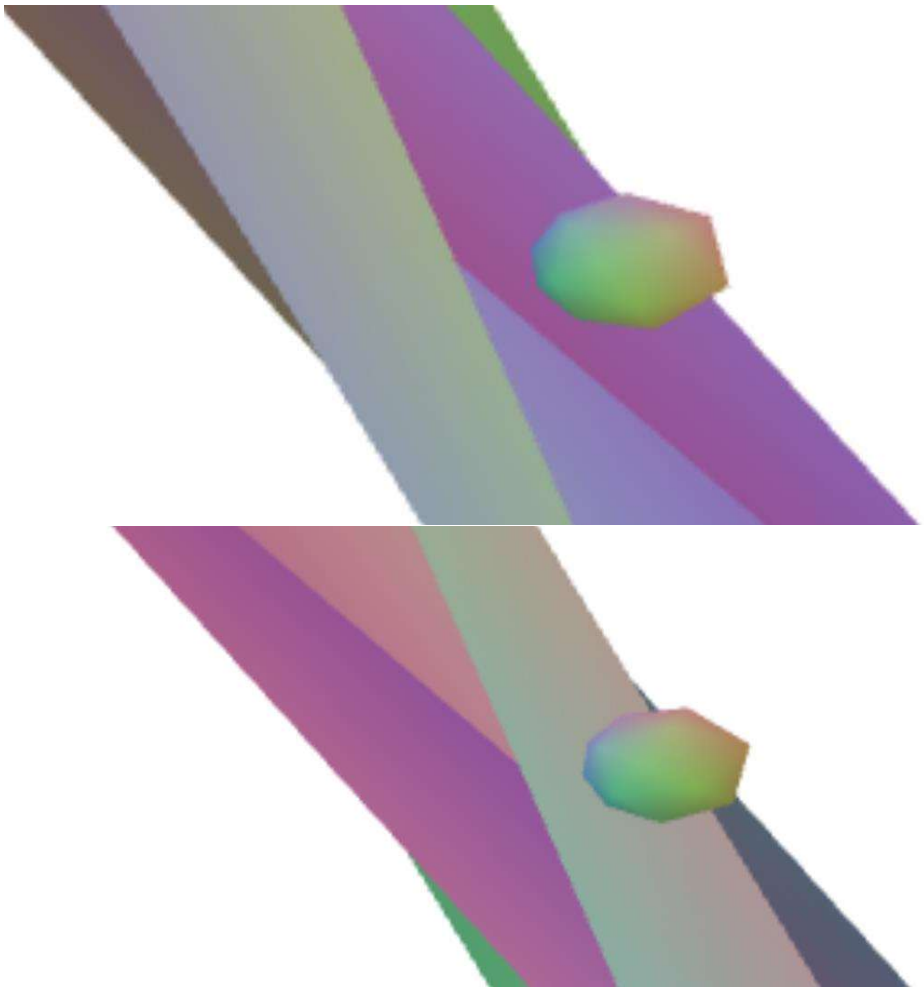
canvas.html | loader.js | makeasphere.py | *planet.js* | Settings

```
    1, 2, 5,
    1, 3, 4,
    1, 3, 5,
  ]

  const pv_1 = [0.0, 0.0, 0.1, 0.0, 0.1, 0.0, 0.1, 0.0, 0.0, 0.0, 0.0, -0.1, 0.0, -0.1, 0.0, -0.1, 0.0, 0.0, 0.0, 0.07071067811865475
  0.07071067811865475, 0.07071067811865475, 0.0, -0.07071067811865475, 0.0, -0.07071067811865475, -0.07071067811865475, -0.0707106781
  const pti_1 = [6, 7, 8, 0, 6, 8, 1, 7, 6, 2, 8, 7, 9, 7, 10, 3, 9, 10, 2, 7, 9, 1, 10, 7, 11, 12, 8, 2, 11, 8, 4, 12, 11, 0, 8, 12,

  const pv_2 = [0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, -1.0, 0.0, -1.0, 0.0, -1.0, 0.0, 0.0, 0.0, 0.7071067811865475,
  0.7071067811865475, 0.0, -0.7071067811865475, 0.0, -0.7071067811865475, -0.7071067811865475, -0.7071067811865475, 0.0, 0.0, -0.7071
  0.3826834323650898, 0.9238795325112867, 0.0, 0.3826834323650898, 0.9238795325112867, 0.3826834323650898, 0.0, 0.816496580927726, 0.
  0.816496580927726, 0.408248290463863, 0.408248290463863, -0.408248290463863, 0.816496580927726, 0.816496580927726, -0.4082482904638
  0.408248290463863, -0.3826834323650898, 0.9238795325112867, 0.0, -0.3826834323650898, 0.0, 0.9238795325112867, -0.9238795325112867,
  0.408248290463863, 0.408248290463863, -0.408248290463863, -0.408248290463863, 0.816496580927726, -0.408248290463863, -0.81649658092
  0.3826834323650898, -0.9238795325112867, 0.408248290463863, -0.408248290463863, -0.816496580927726, 0.408248290463863, -0.816496580
  const pti_2 = [0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, -1.0, 0.0, -1.0, 0.0, -1.0, 0.0, 0.0, 0.0, 0.7071067811865475
  0.7071067811865475, 0.0, -0.7071067811865475, 0.0, -0.7071067811865475, -0.7071067811865475, -0.7071067811865475, 0.0, 0.0, -0.7071
  0.3826834323650898, 0.9238795325112867, 0.0, 0.3826834323650898, 0.9238795325112867, 0.3826834323650898, 0.0, 0.816496580927726, 0.
  0.816496580927726, 0.408248290463863, 0.408248290463863, -0.408248290463863, 0.816496580927726, 0.816496580927726, -0.4082482904638
  0.408248290463863, -0.3826834323650898, 0.9238795325112867, 0.0, -0.3826834323650898, 0.0, 0.9238795325112867, -0.9238795325112867,
  0.408248290463863, 0.408248290463863, -0.408248290463863, -0.408248290463863, 0.816496580927726, -0.408248290463863, -0.81649658092
  0.3826834323650898, -0.9238795325112867, 0.408248290463863, -0.408248290463863, -0.816496580927726, 0.408248290463863, -0.816496580


  const Model_planet = { vertices: pv_1, trianglesIndexed: pti_1 };
```

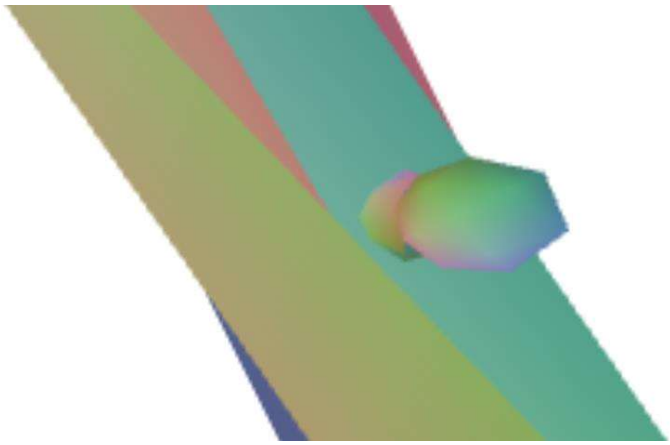
(pv = planet vertices pti = planet triangles indexed)

Testing the vertex shader and sphere generator

Here are the results for 2 spheres at different PVUW positions.



Some for three spheres.



The massive folded structure at the back seems to be a logic error. It may be due to 4D effects.

The other two spheres seem as expected.

Here is the output without the 4D calculation, for one sphere. (The sphere is rotating)

It seems to be squashed, but that is because the xy_ratio is not applied to this debugging mode.

It seems to be a little pixelated - a problem I will need to solve.

Max Silin