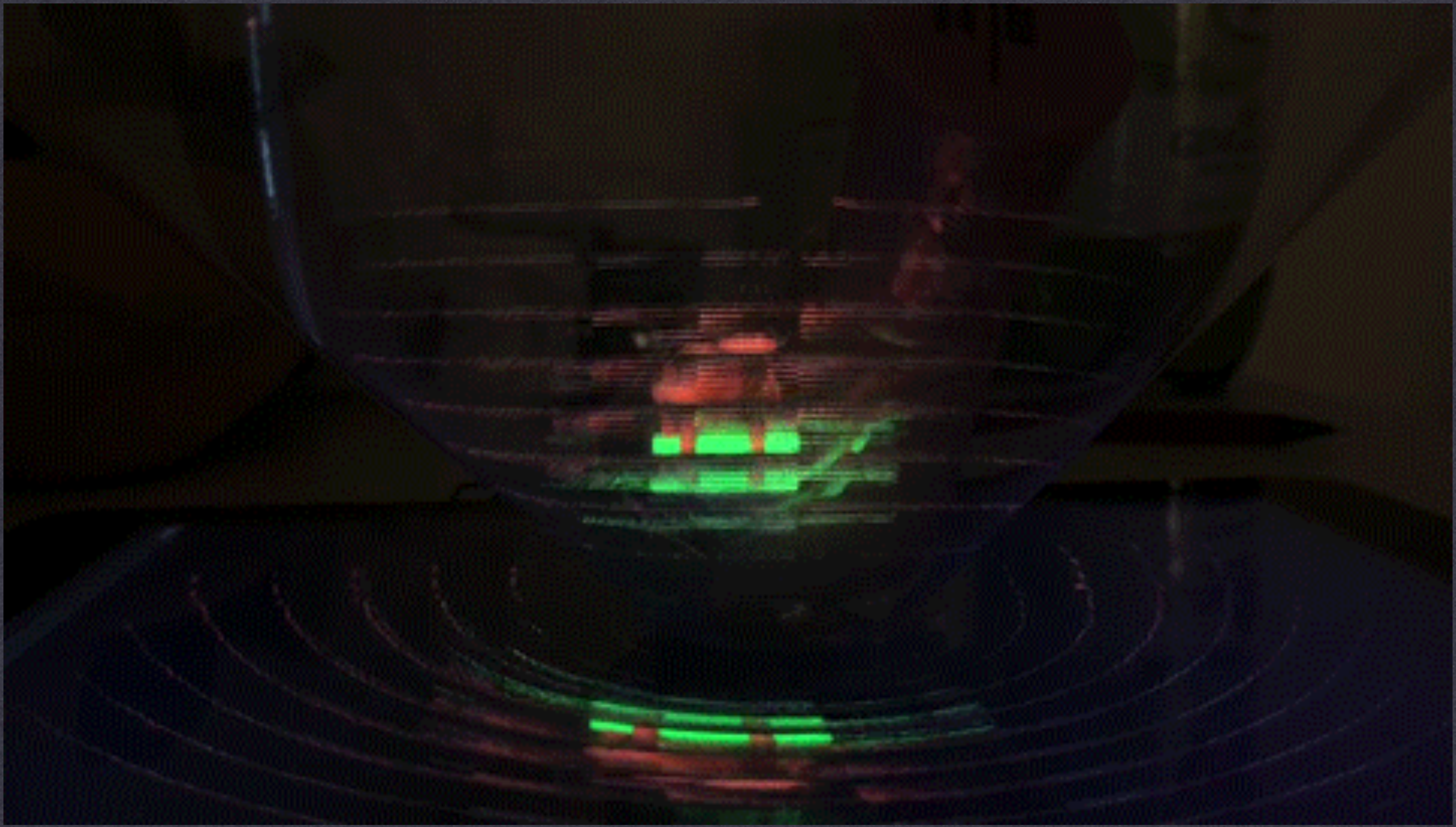


PEPPER’S CONE FOR WEB

CHUNHAN CHEN

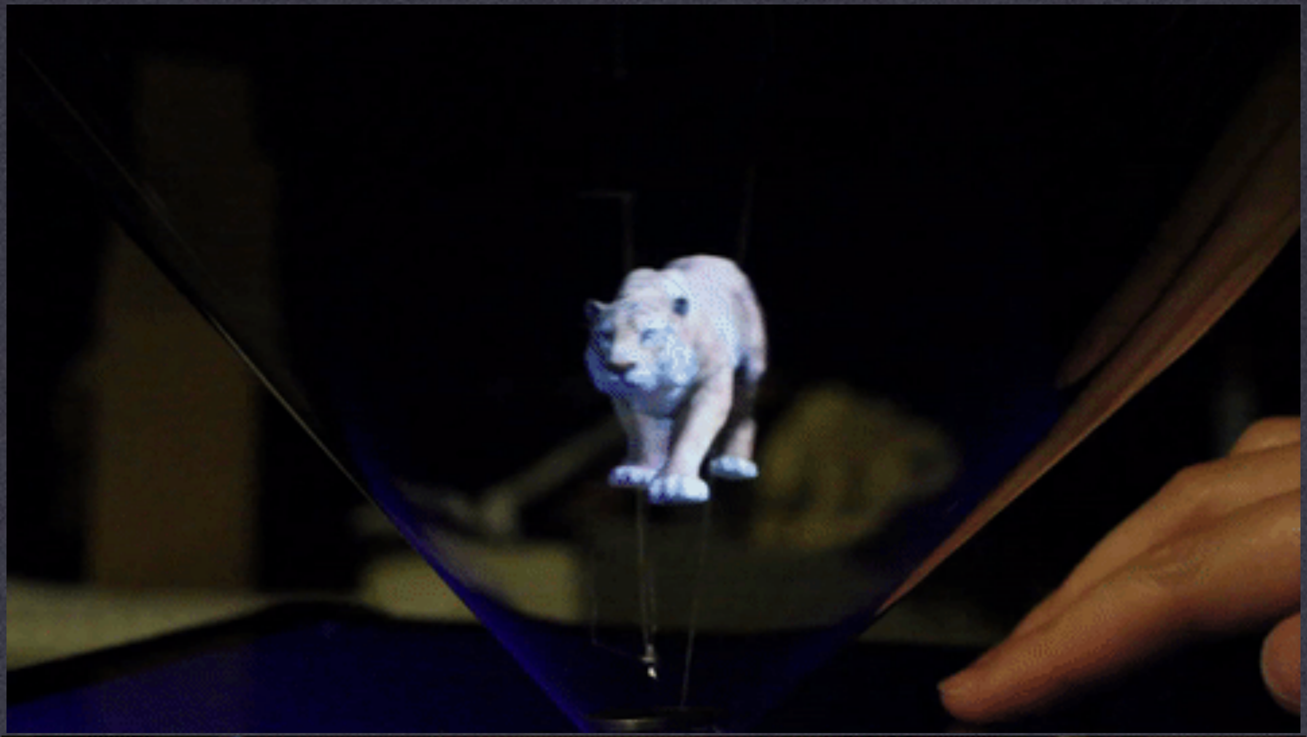
ICM FINAL DEC 2018

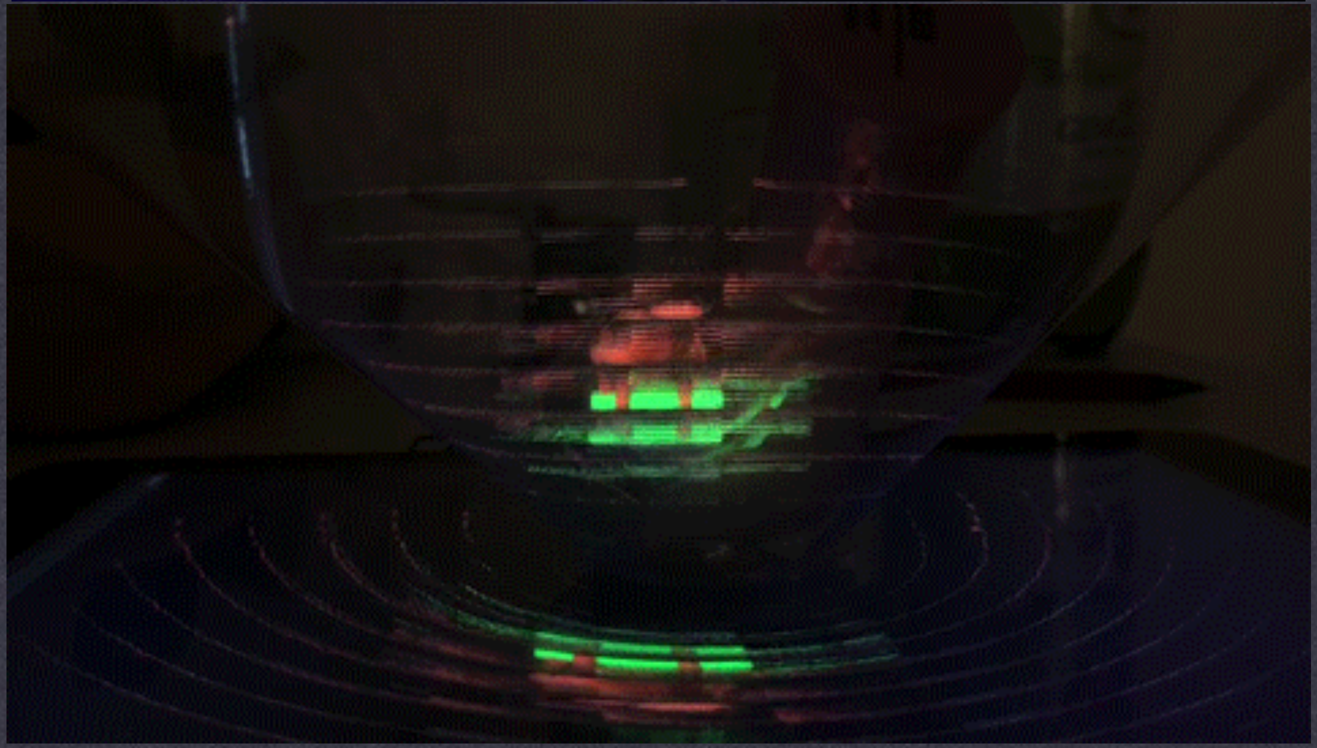
Overview

- In order to augment the visual display of **holograms**.
- A **web version** of Pepper's Cone (*originally created by Luo, Xuan etc in Unity*) is developed to make the 360 degree hologram with **lower cost**.
- Technologically, the Pepper's Cone For Web exploits customized **shaders** in **GLSL**, pre-distortion with **image processing**, 3D scene building with **three.js** and development in **purely Javascript**.
- More **technical** instead of **artistic**.

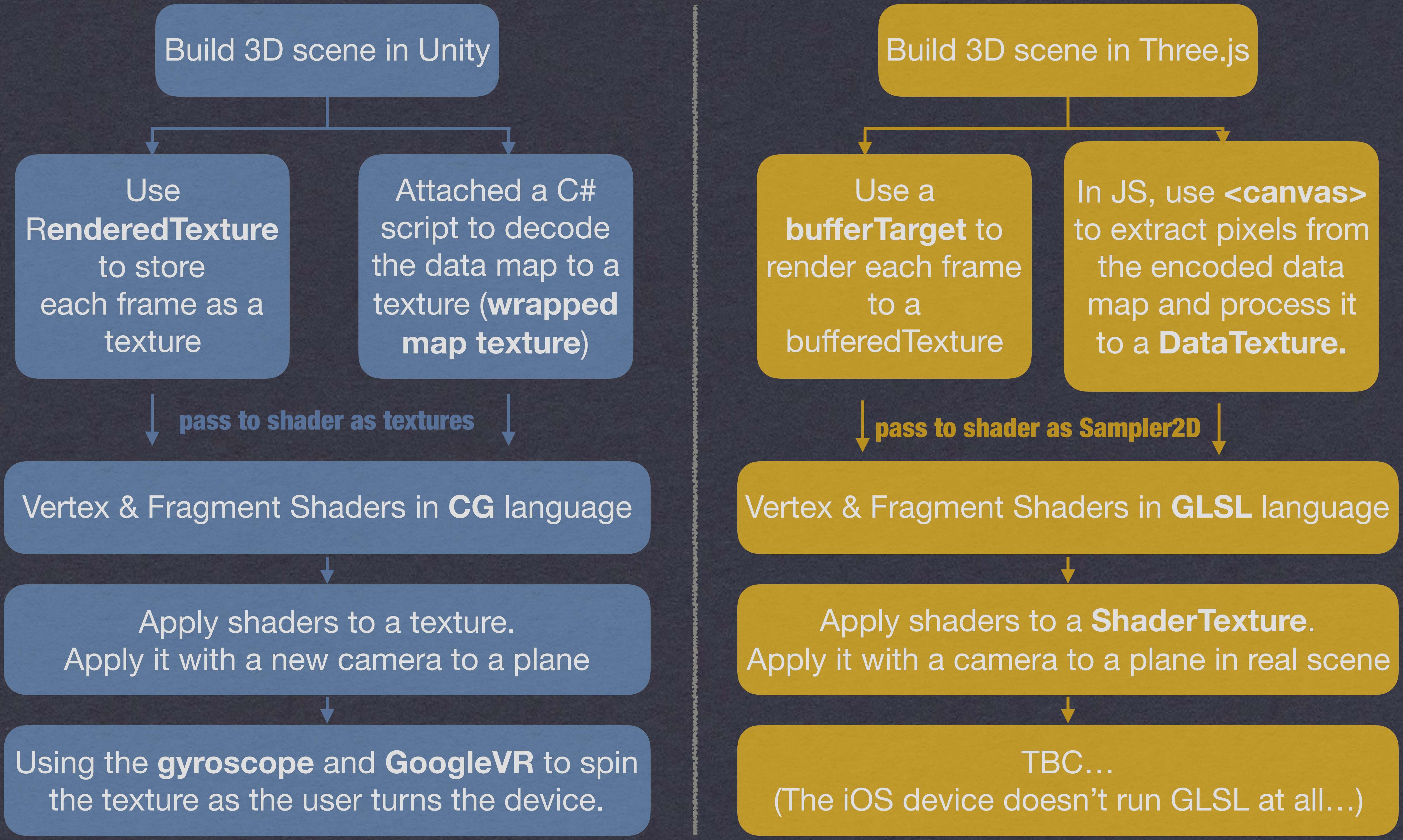
TECHNICAL COMPARISON

WITH LOGIC FLOW





TECHNICAL COMPARISON WITH LOGIC FLOW



Build 3D scene in Unity

Use **RenderedTexture** to store each frame as a texture

Attached a C# script to decode the data map to a texture (**wrapped map texture**)

pass to shader as textures

Vertex & Fragment Shaders in **CG** language

Apply shaders to a texture.
Apply it with a new camera to a plane

Using the **gyroscope** and **GoogleVR** to spin the texture as the user turns the device.

Build 3D scene in Three.js

Use a **bufferTarget** to render each frame to a bufferedTexture

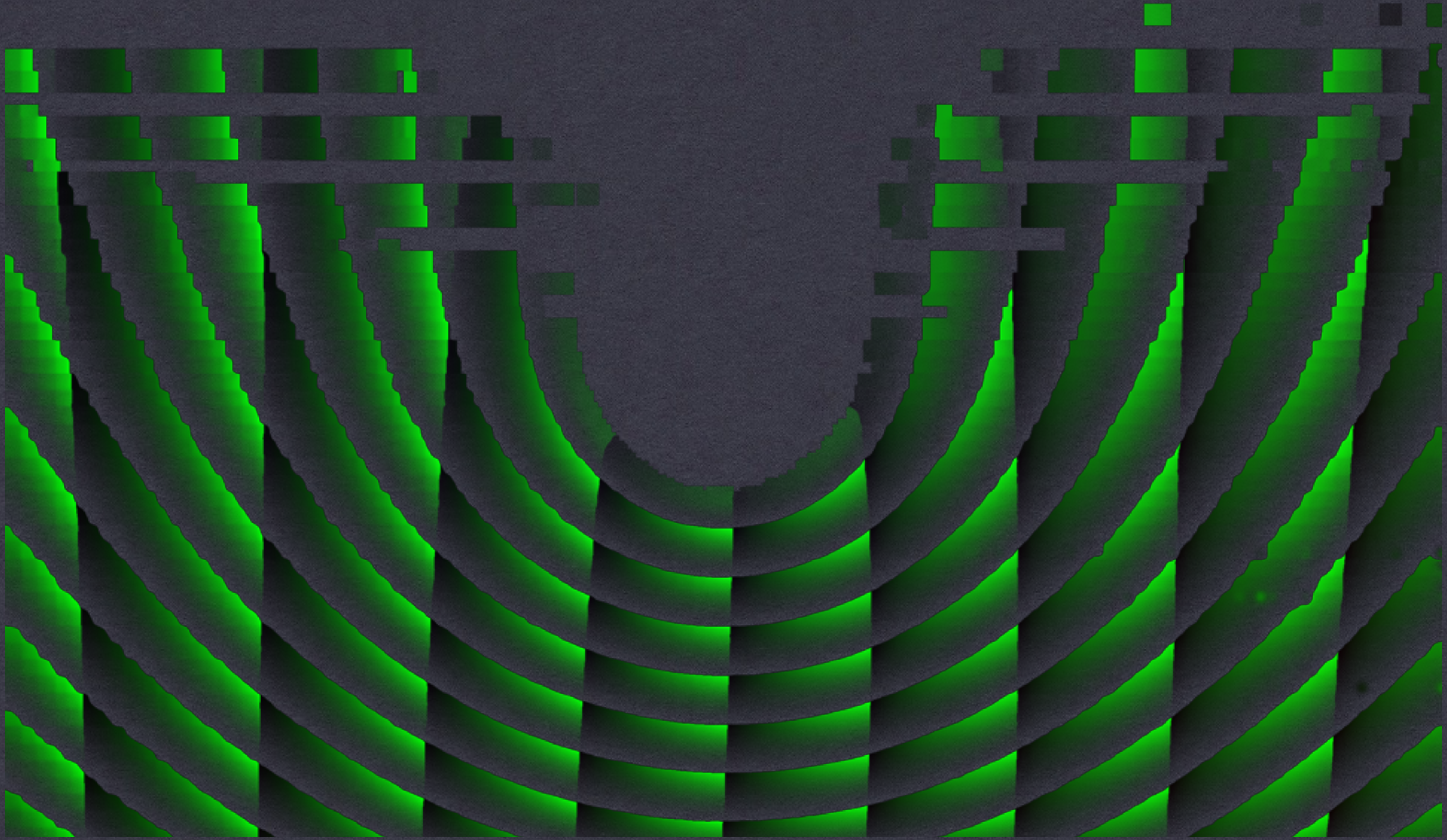
In JS, use **<canvas>** to extract pixels from the encoded data map and process it to a **DataTexture.**

pass to shader as Sampler2D

Vertex & Fragment Shaders in **GLSL** language

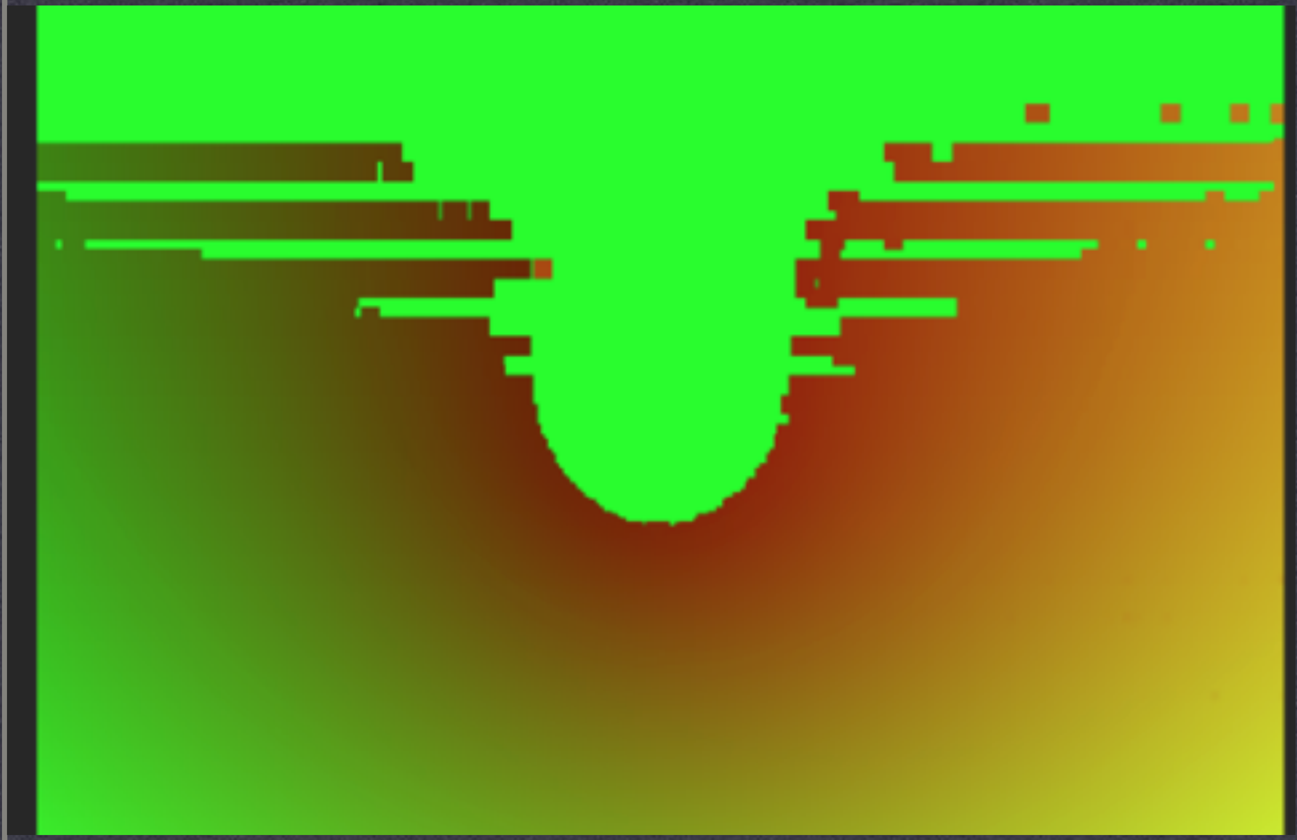
Apply shaders to a **ShaderTexture**.
Apply it with a camera to a plane in real scene

TBC…
(The iOS device doesn't run GLSL at all…)

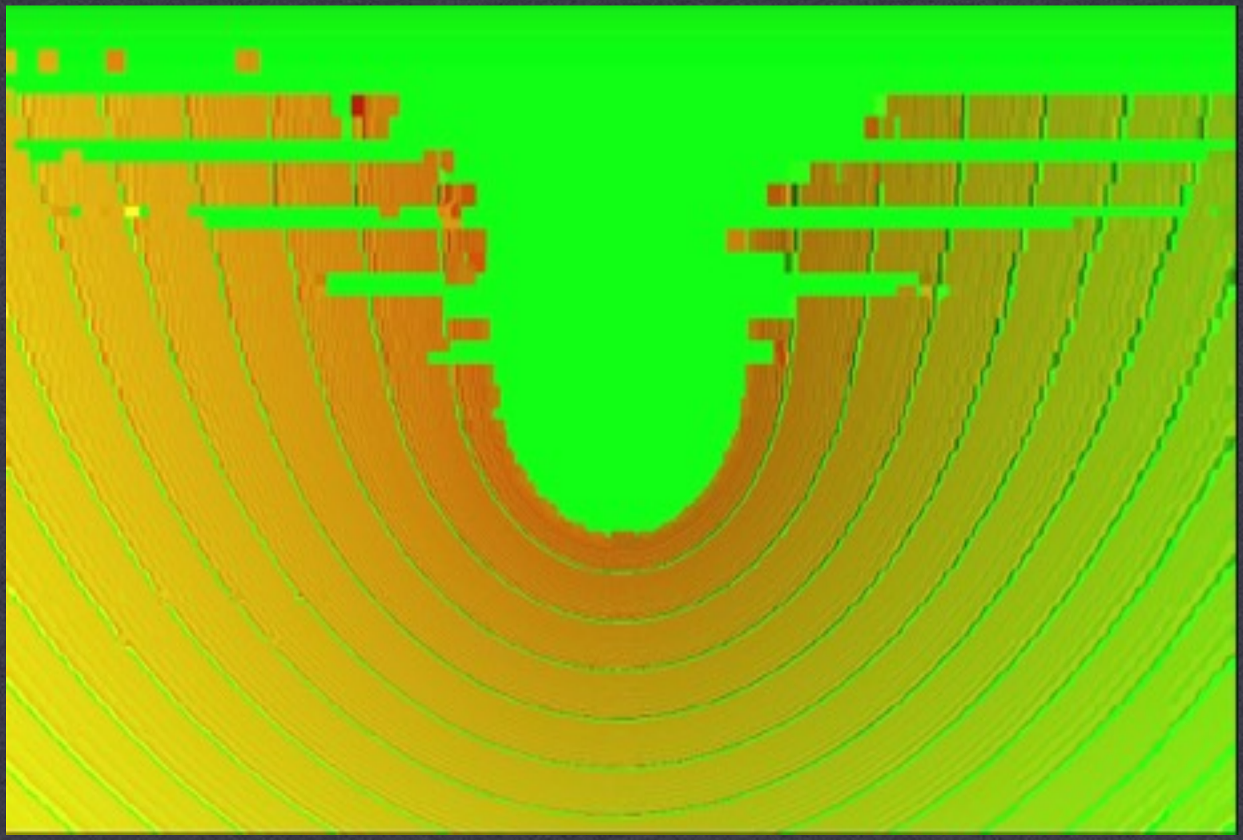


THE ENCODED DATA MAP

BY LUO, XUAN ETC



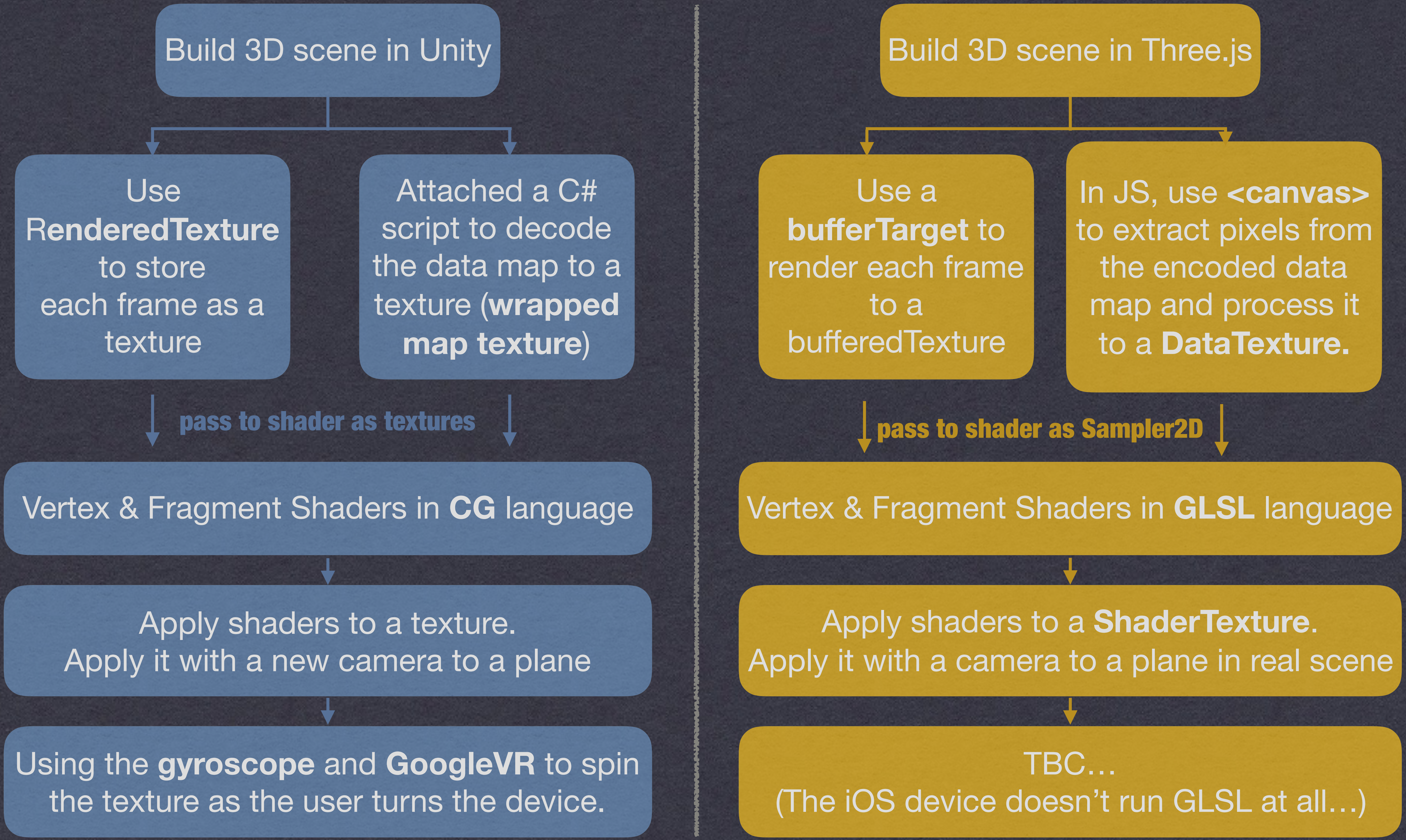
BY LUO, XUAN ETC
IN UNITY
RGFLOAT-FORMAT, TEXTURE2D



BY CHUNHAN
IN JAVASCRIPT & THREE.JS
FLOAT TYPE FORMAT, DATATEXTURE

THE DECODED DATA MAP

TECHNICAL COMPARISON WITH LOGIC FLOW



Build 3D scene in Unity

Use **RenderedTexture** to store each frame as a texture

Attached a C# script to decode the data map to a texture (**wrapped map texture**)

pass to shader as textures

Vertex & Fragment Shaders in **CG** language

Apply shaders to a texture.
Apply it with a new camera to a plane

Using the **gyroscope** and **GoogleVR** to spin the texture as the user turns the device.

Build 3D scene in Three.js

Use a **bufferTarget** to render each frame to a bufferedTexture

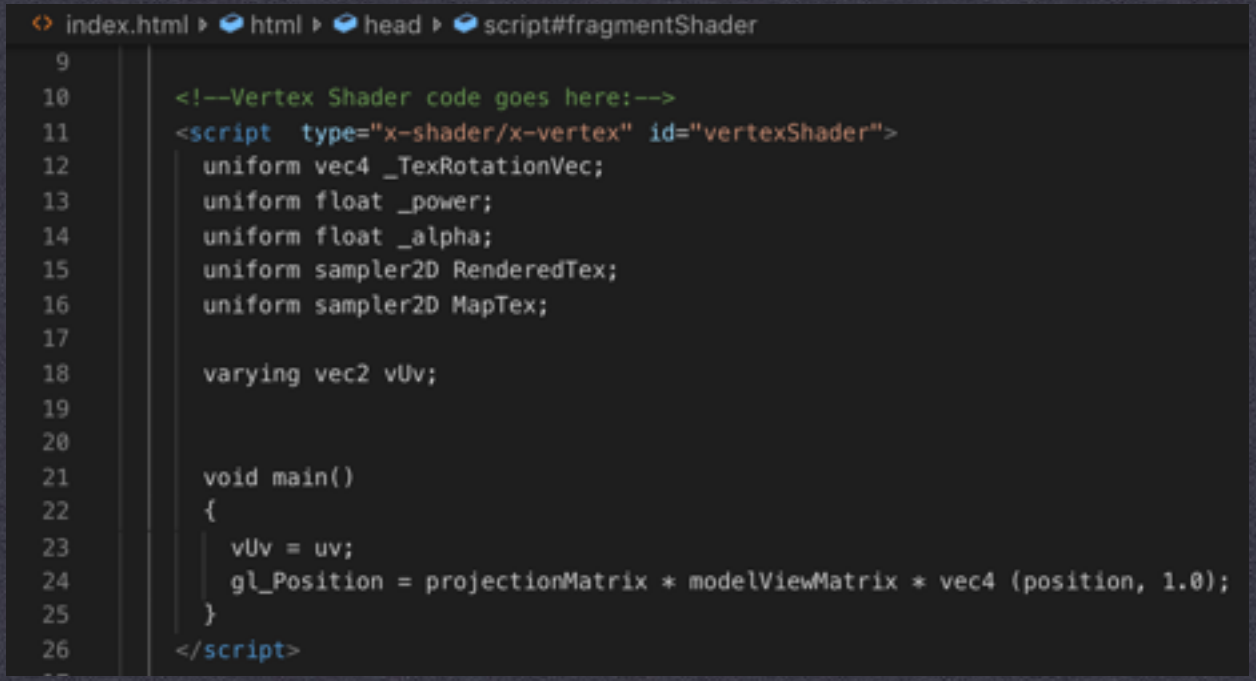
In JS, use **<canvas>** to extract pixels from the encoded data map and process it to a **DataTexture.**

pass to shader as Sampler2D

Vertex & Fragment Shaders in **GLSL** language

Apply shaders to a **ShaderTexture**.
Apply it with a camera to a plane in real scene

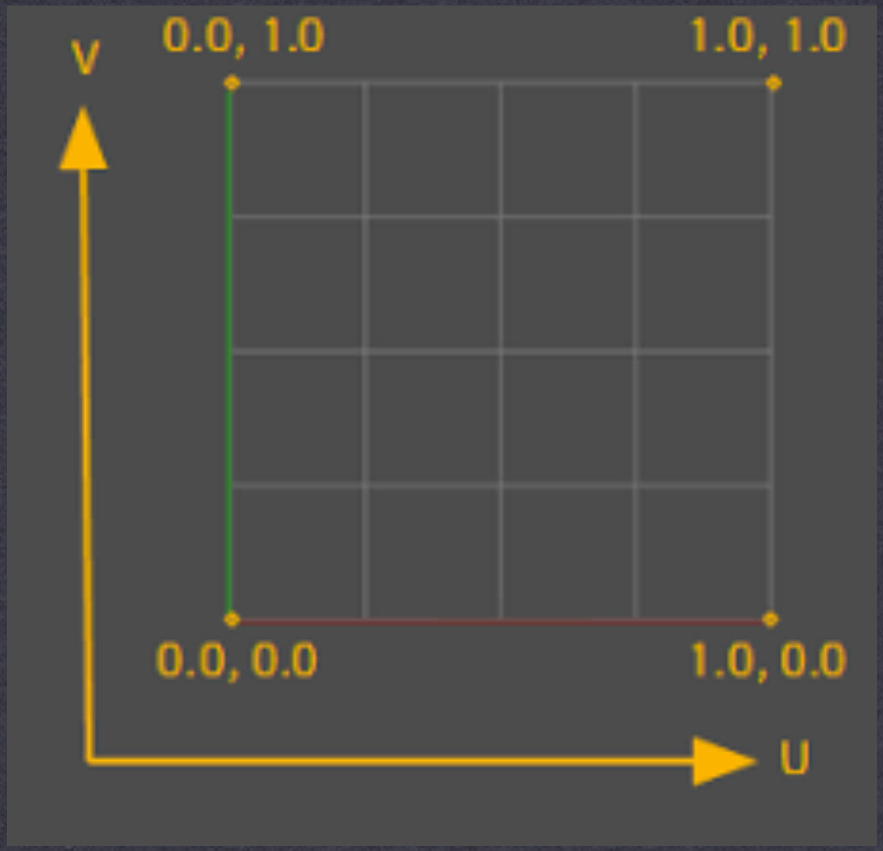
TBC…
(The iOS device doesn't run GLSL at all…)

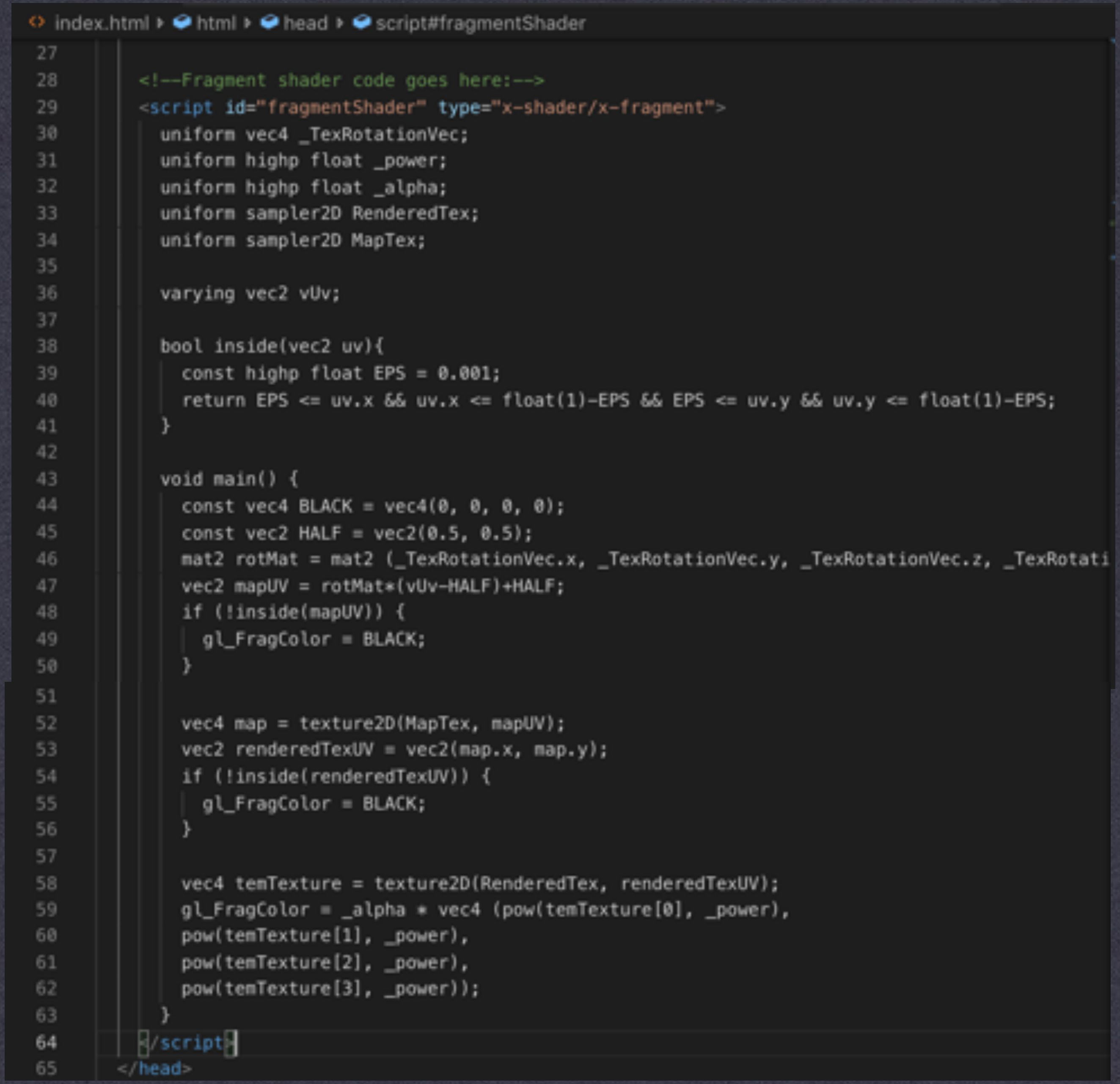


index.html › html › head › script#fragmentShader

```
<!--Vertex Shader code goes here:-->
<script  type="x-shader/x-vertex" id="vertexShader">
  uniform vec4 _TexRotationVec;
  uniform float _power;
  uniform float _alpha;
  uniform sampler2D RenderedTex;
  uniform sampler2D MapTex;

  varying vec2 vUv;


  void main()
  {
    vUv = uv;
    gl_Position = projectionMatrix * modelViewMatrix * vec4 (position, 1.0);
  }
</script>
```

VERTEX SHADER





index.html › html › head › script#fragmentShader

```
  <!--Fragment shader code goes here:-->
  <script id="fragmentShader" type="x-shader/x-fragment">
    uniform vec4 _TexRotationVec;
    uniform highp float _power;
    uniform highp float _alpha;
    uniform sampler2D RenderedTex;
    uniform sampler2D MapTex;

    varying vec2 vUv;

    bool inside(vec2 uv){
      const highp float EPS = 0.001;
      return EPS <= uv.x && uv.x <= float(1)-EPS && EPS <= uv.y && uv.y <= float(1)-EPS;
    }

    void main() {
      const vec4 BLACK = vec4(0, 0, 0, 0);
      const vec2 HALF = vec2(0.5, 0.5);
      mat2 rotMat = mat2 (_TexRotationVec.x, _TexRotationVec.y, _TexRotationVec.z, _TexRotati
      vec2 mapUV = rotMat*(vUv-HALF)+HALF;
      if (!inside(mapUV)) {
        gl_FragColor = BLACK;
      }

      vec4 map = texture2D(MapTex, mapUV);
      vec2 renderedTexUV = vec2(map.x, map.y);
      if (!inside(renderedTexUV)) {
        gl_FragColor = BLACK;
      }

      vec4 temTexture = texture2D(RenderedTex, renderedTexUV);
      gl_FragColor = _alpha * vec4 (pow(temTexture[0], _power),
      pow(temTexture[1], _power),
      pow(temTexture[2], _power),
      pow(temTexture[3], _power));
    }
  </script>
</head>
```

FRAGMENT SHADER

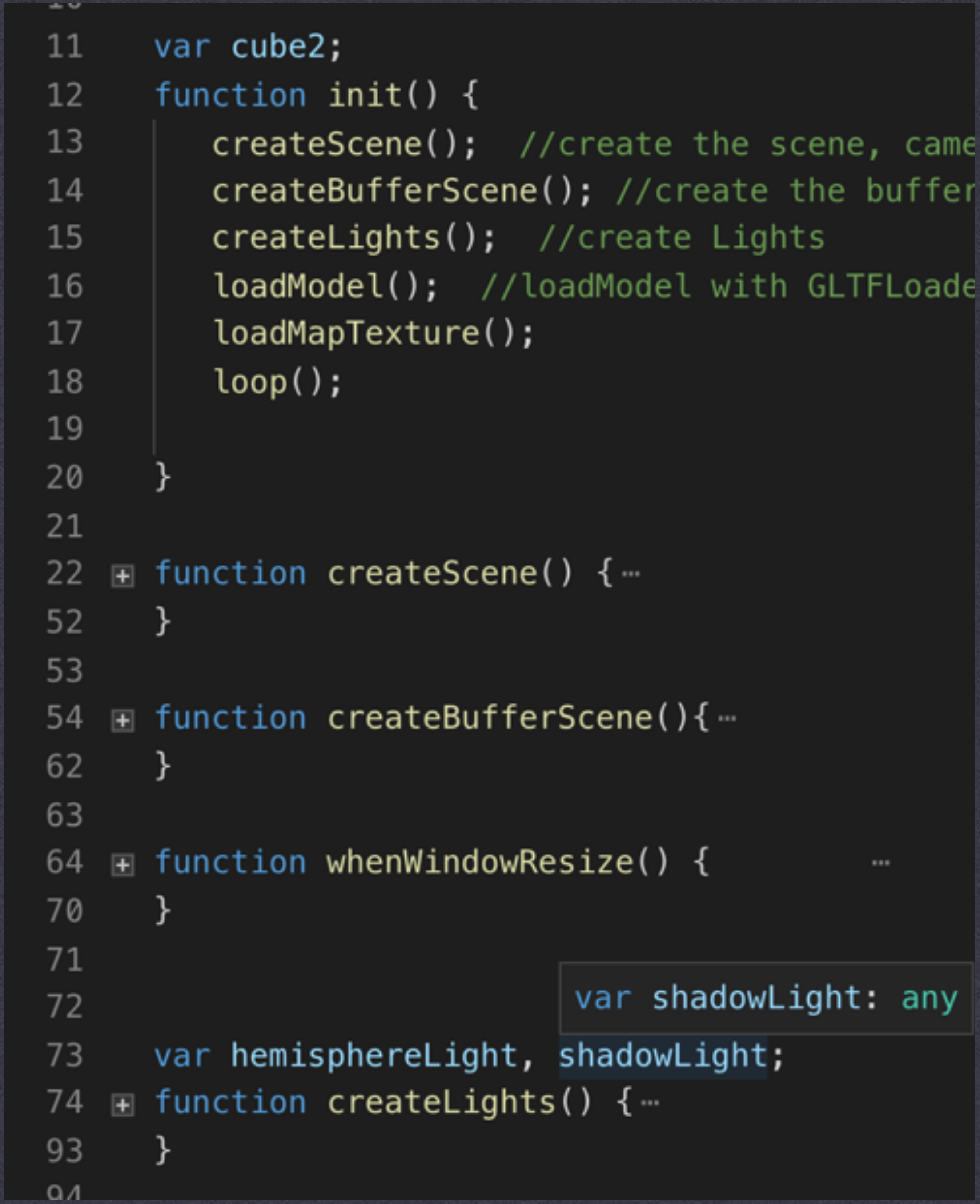
UV AXIS IN VERTEX SHADER & FRAGMENT SHADER

REF: DATA:IMAGE/PNG;BASE64,IVBORW0KGGOAAAANSUHEUGAAAOQAAADDCAMAAACC/C7AAAABELBMVEVMTEXLS0TIYMJFX19XV1DHR0DISK1LTEXBR011Y0VNSU1NZ2DBW1S6AZPPT08+GT5TTLP5TBV/
VBSXHJDOODHSULJOSE1DZD9OTK6YDEVSR01NWODUUUPAVELESKZEPCS6JI6NGZPEKY6MCT/JLYQAAKFTX0BMQCGLCEFGWEH3ZUHURR6VDJV+TXJMOYTKG0GKFZMQB0BGODS4OB84RU3PJ0HMTRTJBDTSVKCZQE+ZMIFVB2/MLT2HLYAGGC9/

What I’ve learned…

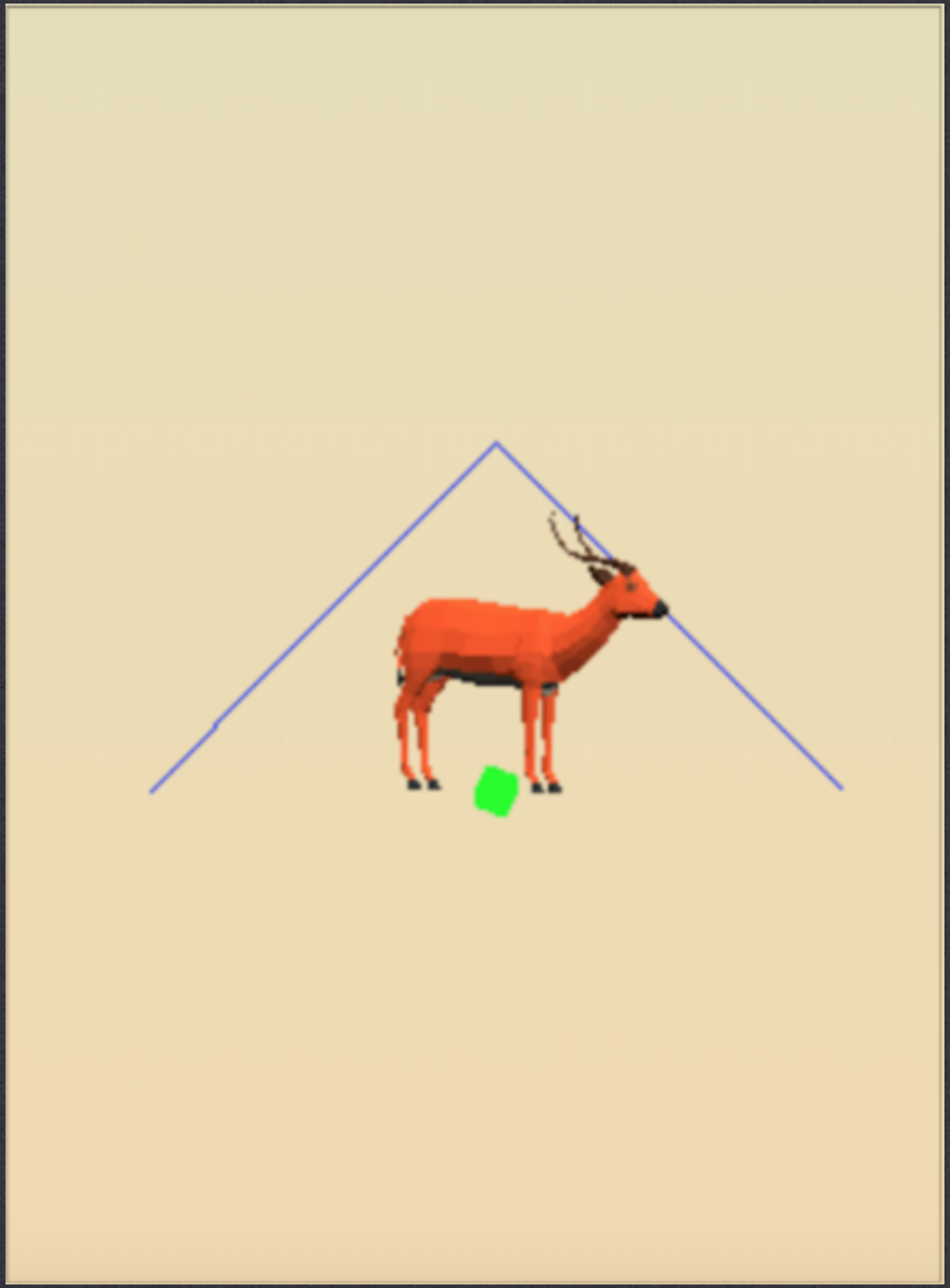
- Computer Graphics basics
- Reviewed my undergrad Linear Algebra
- CG & GLSL shader programming
- C# syntax and Unity development
- Three.js for developers (hope for better docs!)
- How to store wrapping data in a png
- Hardwares issues are HARD.

MORE ILLUSTRATIONS
(IF TIME PERMITTED)

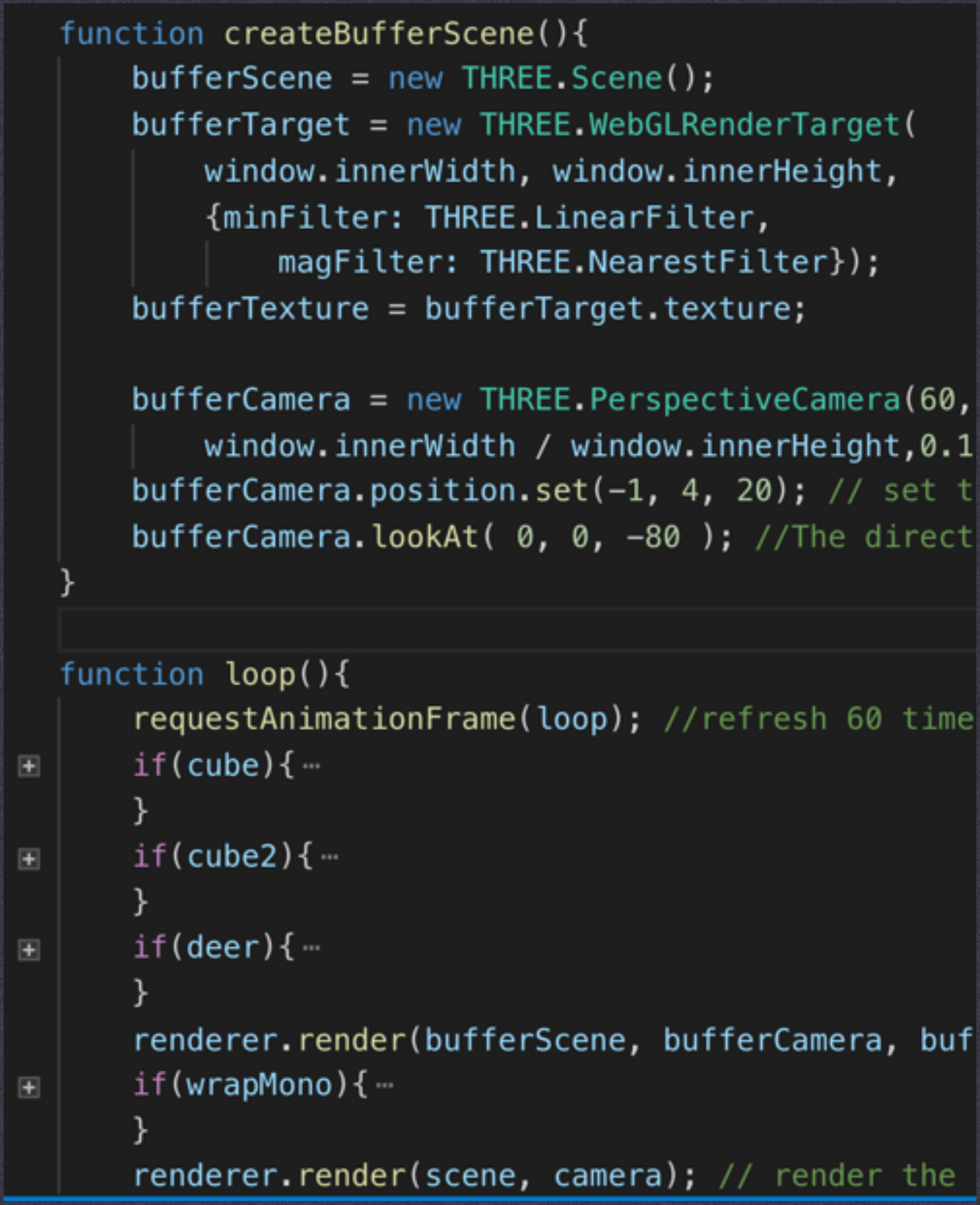


```
var cube2;
function init() {
    createScene();  //create the scene, came
    createBufferScene(); //create the buffer
    createLights();  //create Lights
    loadModel();  //loadModel with GLTFLoade
    loadMapTexture();
    loop();

}

function createScene() {…
}

function createBufferScene(){…
}

function whenWindowResize() {        …
}


var hemisphereLight, shadowLight;
function createLights() {…
}
```

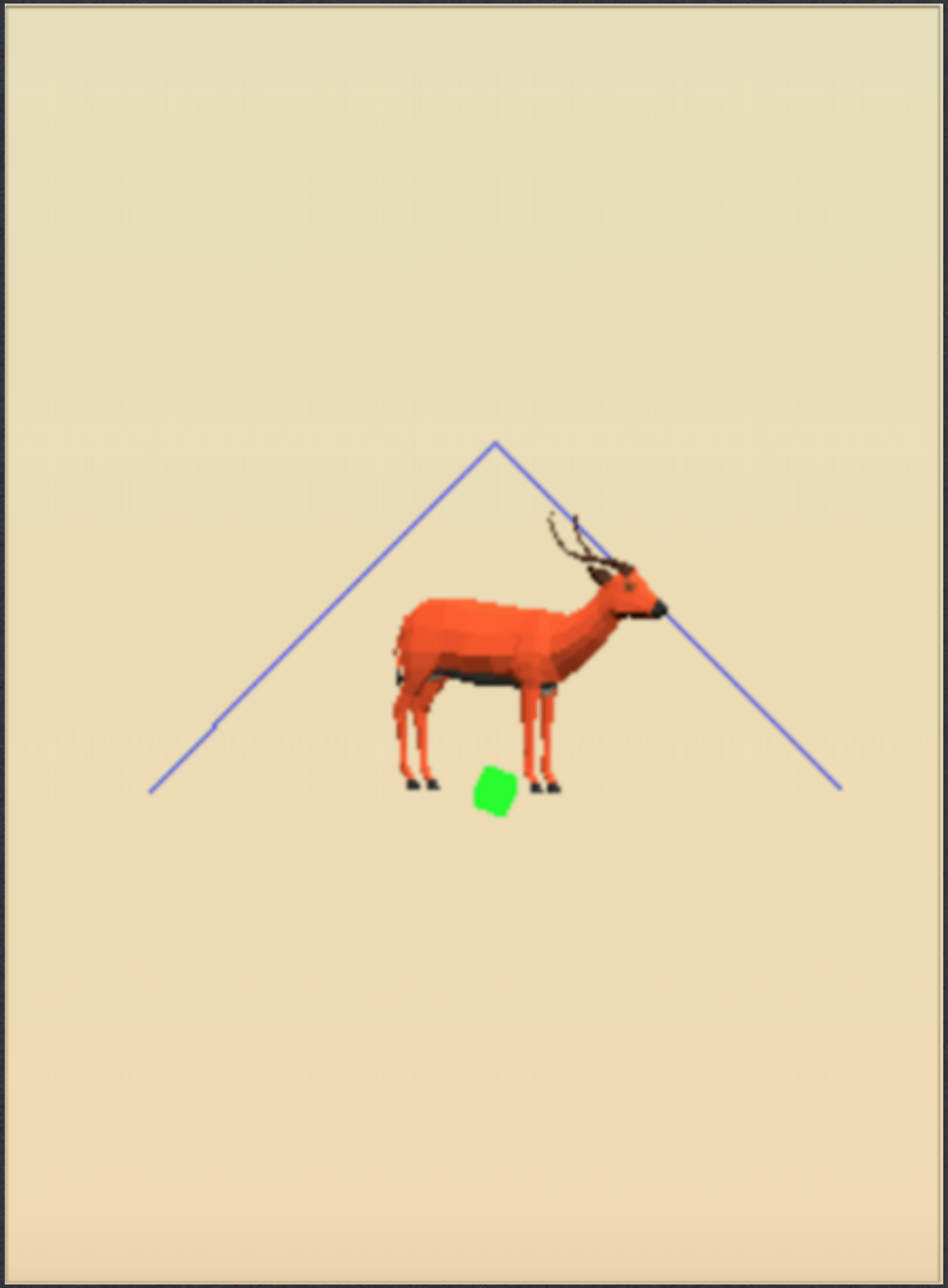
var shadowLight: any



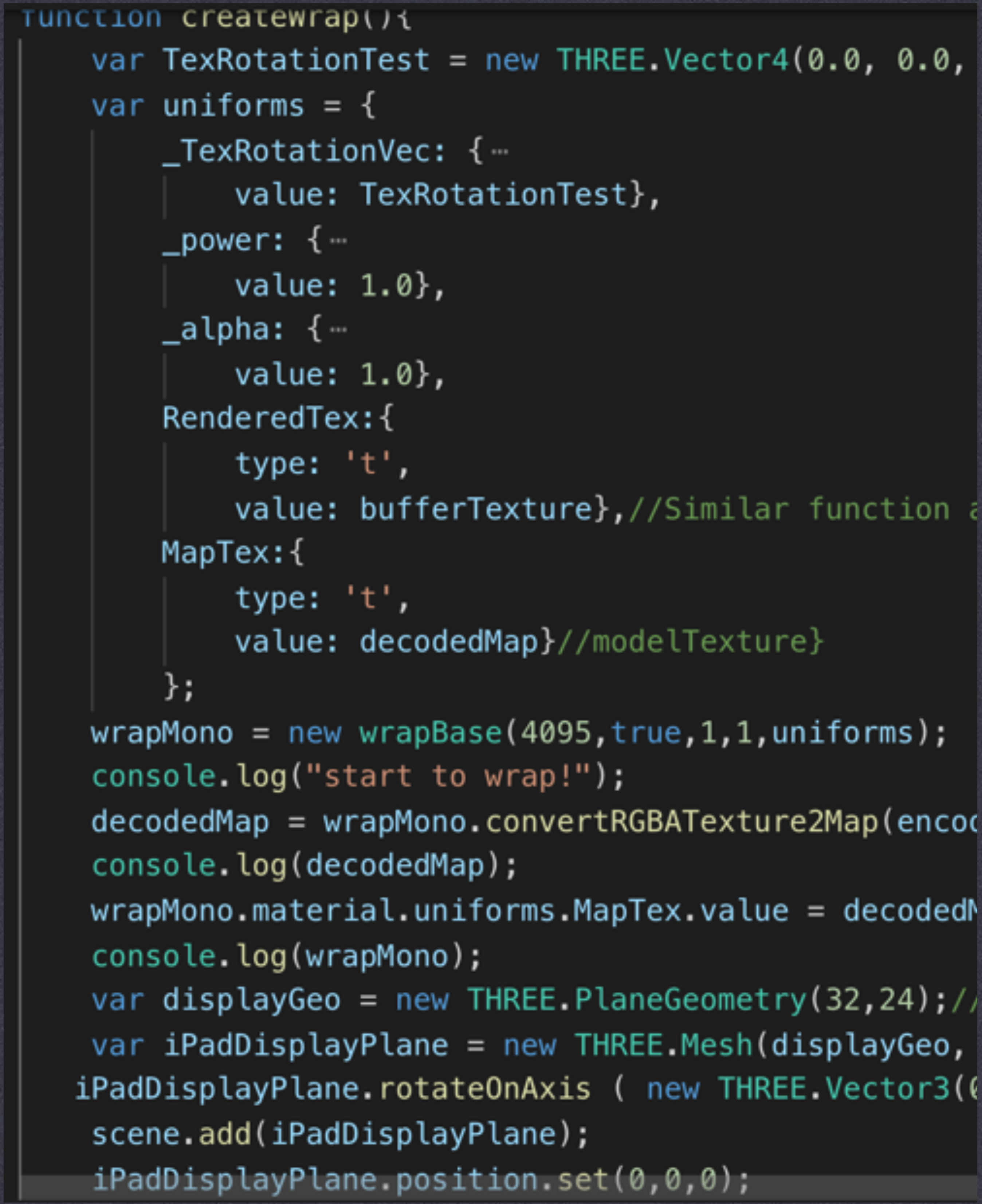
BUILD A 3D SCENE IN THREE.JS AND IMPORT A GLTF MODEL



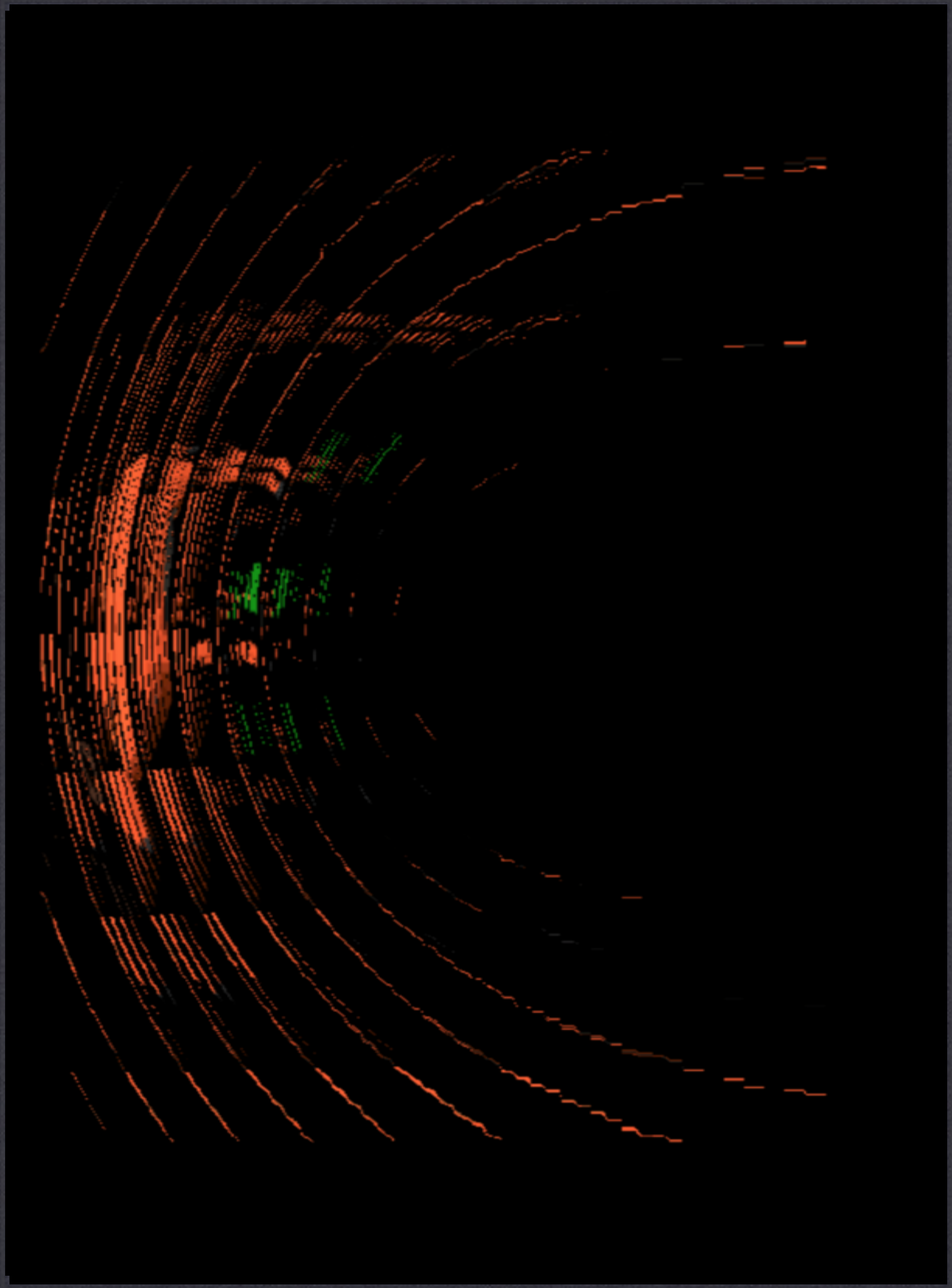
```
function createBufferScene(){
    bufferScene = new THREE.Scene();
    bufferTarget = new THREE.WebGLRenderTarget(
        window.innerWidth, window.innerHeight,
        {minFilter: THREE.LinearFilter,
            magFilter: THREE.NearestFilter});
    bufferTexture = bufferTarget.texture;

    bufferCamera = new THREE.PerspectiveCamera(60,
        window.innerWidth / window.innerHeight,0.1
    bufferCamera.position.set(-1, 4, 20); // set t
    bufferCamera.lookAt( 0, 0, -80 ); //The direct
}

function loop(){
    requestAnimationFrame(loop); //refresh 60 time
    if(cube){⋯
    }
    if(cube2){⋯
    }
    if(deer){⋯
    }
    renderer.render(bufferScene, bufferCamera, buf
    if(wrapMono){⋯
    }
    renderer.render(scene, camera); // render the
```



RENDER THESE TO A BUFFER TEXTURE WITH BUFFER CAMERA



```
function createWrap(){
    var TexRotationTest = new THREE.Vector4(0.0, 0.0,
    var uniforms = {
        _TexRotationVec: {…
            value: TexRotationTest},
        _power: {…
            value: 1.0},
        _alpha: {…
            value: 1.0},
        RenderedTex:{
            type: 't',
            value: bufferTexture},//Similar function a
        MapTex:{
            type: 't',
            value: decodedMap}//modelTexture}
        };
    wrapMono = new wrapBase(4095,true,1,1,uniforms);
    console.log("start to wrap!");
    decodedMap = wrapMono.convertRGBATexture2Map(encod
    console.log(decodedMap);
    wrapMono.material.uniforms.MapTex.value = decodedM
    console.log(wrapMono);
    var displayGeo = new THREE.PlaneGeometry(32,24);//
    var iPadDisplayPlane = new THREE.Mesh(displayGeo,
   iPadDisplayPlane.rotateOnAxis ( new THREE.Vector3(
    scene.add(iPadDisplayPlane);
    iPadDisplayPlane.position.set(0,0,0);
```



THE REAL SCENE WITH ONLY A PLANE AND ITS SHADERTEXTURE

ENJOY :)