实验课程大作业

地球仪的绘制

课程	计算机图形学
班级	软 件 8 4
学号	2186113939
姓名	朱 乾 坤
日期	2020 年 11 月 18 日

目录

1 实验要求

1. 绘制圆球，采用纹理映射的方式将世界地图贴到圆球上，纹理贴图要实现无缝；
2. 实现缩放球体功能和拖动旋转球体功能；
3. 增加支架，生成真实的地球仪；
4. 加入光照，阴影，增加逼真度。

2 实验环境

2.1 OpenGL ES 介绍

OpenGL 是一个跨平台的高性能 3D 渲染 API，而 OpenGL ES 是它的嵌入式平台版本，该 API 由 Khronos 集团定义推广，Khronos 是一个图形软硬件行业协会，该协会主要关注图形和多媒体方面的开放标准。

(a) OpenGL ES 图标　(b) WebGL 图标　(c) Three.js 图标

图 1: 各种图标

2.2 WebGL 介绍

WebGL（全称 Web Graphics Library）是一种 3D 绘图协议，这种绘图技术标准允许把 JavaScript 和 OpenGL ES 2.0 结合在一起，通过增加 OpenGL ES 2.0 的一个 JavaScript 绑定，WebGL 可以为 HTML5 Canvas 提供硬件 3D 加速渲染，这样 Web 开发人员就可以借助系统显卡来在浏览器里更流畅地展示 3D 场景和模型了，还能创建复杂的导航和数据视觉化。显然，WebGL 技术标准免去了开发网页专用渲染插件的麻烦，可被用于创建具有复杂 3D 结构的网站页面，甚至可以用来设计 3D 网页游戏。

2.3 Three.js 介绍

Three.js 是一款运行在浏览器中的 3D 引擎，为 WebGL 的一个工具库，隐藏了很多底层的细节。使用者可以用它创建各种三维场景，包括了摄影机、光

影、材质等各种对象。

2.4 建立编程环境

1. 编辑器选择 HBuilder。创建最基本的 html 工程，包含 css，js，img 三个文件夹以及 index.html，创建工程页面如图2(a)。创建的工程结构如图2(b)。

2. 本实验基本不需要写 css 文件，因为地球仪被渲染在 html5 的 canvas（画布）上，不需要对格式进行修改，所以仅增添针对”Earth” 标题的特效进行点缀即可。修改”Earth” 样式后的效果如图2(c)：

(a) 选择合适的工程并初始化　(b) 创建的工程结构　(c) 标题

图 2: 初始化工程

3. 在 GitHub 搜索 three.js，本地克隆后将 three.js(或者 three.min.js) 以及 OrbitControls.js 文件复制到目录的 js 文件夹中，作为库文件引用。

4. 本实验的 HTML 文件的代码如下：

```
<!DOCTYPE html>
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
      user-scalable=no, minimum-scale=1.0, maximum-scale=1.0">
  <link rel="stylesheet" type="text/css" href="./css/style.css
      " />
  <link rel="icon" type="image/png" href="./img/favicon.png">
  <title>Earth</title>
</head>

<body>
  <a id="title">Earth</a>
  <script src="./js/three.min.js"></script>
  <script src="./js/OrbitControls.js"></script>
  <script src="./js/main.js"></script>
</body>
</html>
```

3 实验步骤

3.1 绘制圆球

1. Three.js 中有对球体的构造函数：
   THREE.SphereGeometry(radius, segmentsWidth, segmentsHeight, phiStart,
   phiLength, thetaStart, thetaLength) 各个参数的意义是：
   - radius：半径；
   - segmentsWidth：经度上的切片数，相当于经度被切成了几瓣；
   - segmentsHeight：纬度上的切片数，相当于纬度被切成了几层；
   - phiStart：经度开始的弧度；
   - phiLength：经度跨过的弧度；
   - thetaStart：纬度开始的弧度；
   - thetaLength：纬度跨过的弧度。

2. Three.js 也提供对球体构造函数的重载，只包含半径，经纬线条数。创建半径为 2，经纬线条数均为 25 的圆球线框的如图3：

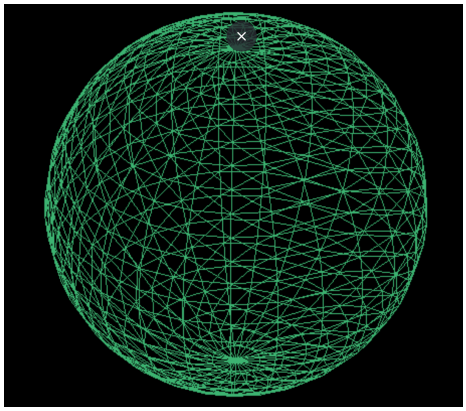


图 3: 显示带颜色线框的球体

3. 创建线框球体示例的代码如下：

```
var cube = new THREE.Mesh(new THREE.SphereGeometry(2, 25, 25),
new THREE.MeshBasicMaterial({
  color: 0x3CB371,
  wireframe: true
})
```

3.2 材质与纹理

1. 3D 世界的纹理由图片组成。将纹理以一定的规则映射到几何体上，一般是三角形上，那么这个几何体就有纹理皮肤了。在 threejs 中，实现纹理首先应该有一个纹理类，其次是有一个加载图片的方法，将这张图片和这个纹理类捆绑起来。在 threejs 中，纹理类由 THREE.Texture 表示，其构造函数如下所示：
   THREE.Texture(image, mapping, wrapS, wrapT, magFilter, minFilter, format, type, anisotropy) 各个参数的意义是：
   • Image：这是一个图片类型，基本上它有 ImageUtils 来加载，如代码：var image = THREE.ImageUtils.loadTexture(url);
   • Mapping：是一个 THREE.UVMapping() 类型，它表示的是纹理坐标；
   • wrapS：表示 x 轴的纹理的回环方式，就是当纹理的宽度小于需要贴图的平面的宽度的时候，平面剩下的部分应该以何种方式贴图的问题；
   • wrapT：表示 y 轴的纹理回环方式。magFilter 和 minFilter 表示过滤的方式，这是 OpenGL 的基本概念；
   • format：表示加载的图片的格式，这个参数可以取值 THREE.RGBAFormat，RGBFormat 等。THREE.RGBAFormat 表示每个像素点要使用四个分量表示，分别是红、绿、蓝、透明来表示。RGBFormat 则不使用透明，也就是说纹理不会有透明的效果；
   • type：表示存储纹理的内存的每一个字节的格式，是有符号，还是没有符号，是整形，还是浮点型。不过这里默认是无符号型(THREE.UnsignedByteType)；
   • anisotropy：各向异性过滤。使用各向异性过滤能够使纹理的效果更好，但是会消耗更多的内存、CPU、GPU 时间。

2. 选择的世界地图如图4；

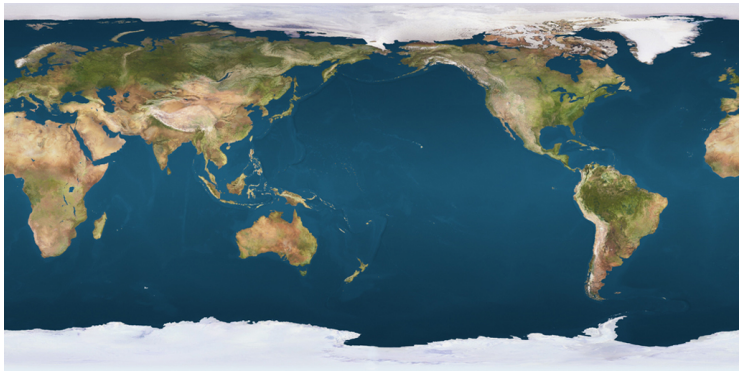


图 4: 世界地图

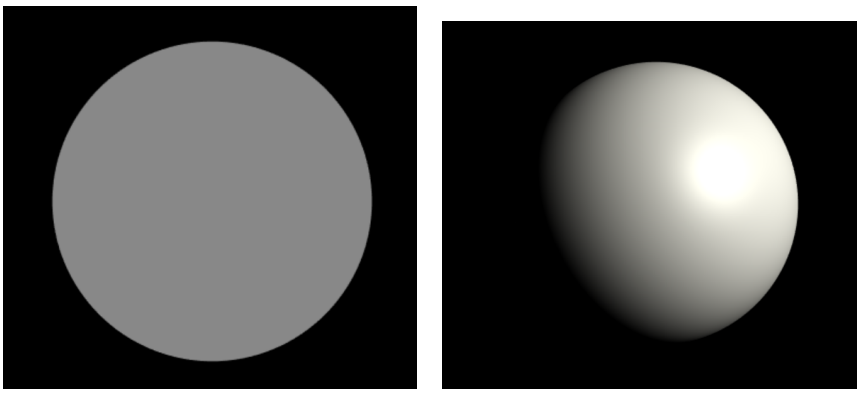
3. 设置地球的材质并贴上纹理的代码如下：

```
var earthMaterial = new THREE.MeshPhongMaterial({
  //绑定世界地图纹理
  map: new THREE.ImageUtils.loadTexture("img/earth.jpg"),
  color: 0xaaaaaa,
  specular: 0x333333,
  shininess: 25,
  side: THREE.DoubleSide,
});
```

3.3 光照效果

1. three.js 库提供了一些列光源，而且没种光源都有特定的行为和用途。这些光源包括：
   • AmbientLight(环境光)：这是一种基础光源，它的颜色会添加到整个场景和所有对象的当前颜色上；
   • PointLight(点光源)：空间中的一点，朝所有的方向发射光线；
   • SpotLight(聚光灯光源)：这种光源有聚光的效果，类似台灯、天花板上的吊灯，或者手电筒；
   • DirectionLight(方向光)：也称为无限光。从这种光源发出的光线可以看着平行的。例如，太阳光；
   • HemishpereLight(半球光)：这是一种特殊光源，可以用来创建更加自然的室外光线，模拟放光面和光线微弱的天空；
   • AreaLight(面光源)：使用这种光源可以指定散发光线的平面，而不是空间中的一个点；
   • LensFlare(镜头眩光)：这不是一种光源，但是通过 LensFlare 可以为场景中的光源添加眩光效果。

2. 只设置 DirectionalLight 时，光源发出的光线是平行的，无法显示出立体效果，显示效果如图5(a)：

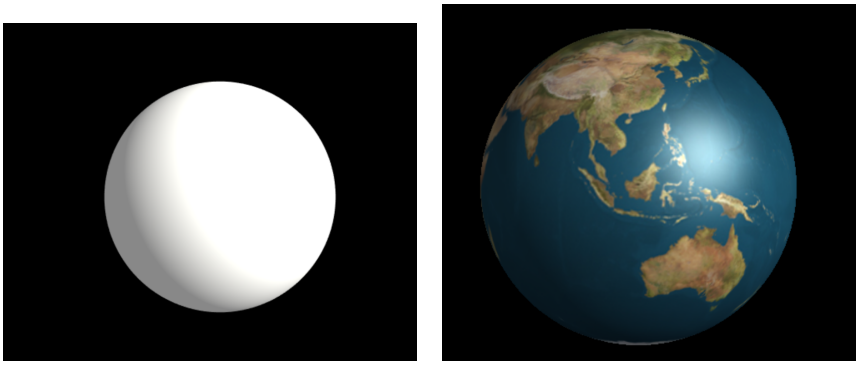


(a) 只设置 DirectionalLight 的球体显示效果　(b) 只设置 AmbientLight 的球体显示效果

图 5: 平行光和环境光效果展示

3. 只设置 AmbientLight 时，环境光的显示效果如图5(b)。

4. 同时设置 DirectionalLight 和 AmbientLight 时，光源共同作用的的显示效果如图6(a)。



(a) 同时设置 DirectionalLight 和 AmbientLight 的球体显示效果

(b) 贴纹理后的光照显示效果

图 6: 平行光和环境光效果展示

5. 在设置了 DirectionalLight 和 AmbientLight 时的基础上贴图后，地球在光源下的显示效果如图6(b)。

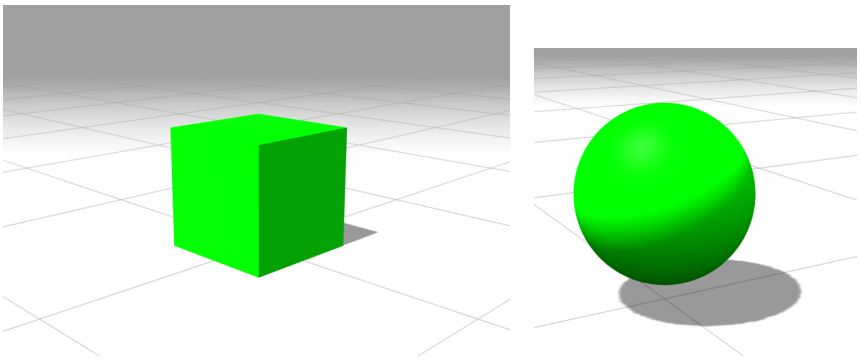
3.4 阴影效果

1. 渲染器开启阴影渲染：
   renderer.shadowMap.enabled = true;

2. 灯光需要开启投射阴影：light.castShadow = true;

3. 告诉模型哪些物体需要投射阴影，哪些物体需要接收阴影：
   sphere.castShadow = true;
   cube.castShadow = true;

4. 阴影的显示效果实例如图7(a)和图7(b)，为简化制造阴影效果在实例中未进行贴图：

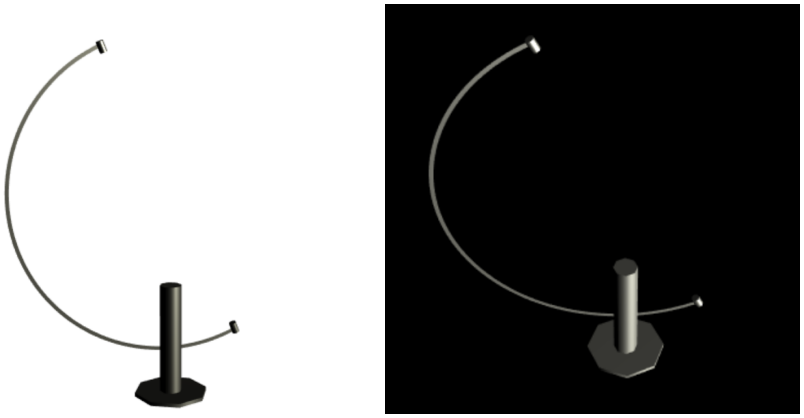


(a) 立方体阴影效果　　(b) 圆球阴影效果

图 7: 阴影效果展示

3.5 金属支架

先创造几何体并将它们绑定到一起，设置个部件的相对位置后再设置物体的材质，金属支架的显示效果如图8(a)和图8(b)：



(a) 金属支架正面效果　　(b) 金属支架侧面效果

图 8: 阴影效果展示

3.6 创建背景

背景可设置单一颜色或背景图片。创建平台，近处和远处的平台颜色作出区别，初始背景如图9(a), 可增加显示网格效果如图9(b)。

(a) 纯色背景　　(b) 网格背景

图 9: 阴影效果展示

3.7 缩放与旋转

通过 Three.js 提供的引入控制旋转和缩放的库函数 OrbitControls.js，加入缩放与旋转的功能的代码如下：

```
var orbit = new THREE.OrbitControls(camera, renderer.domElement);
```

实现缩放和拖动的源代码 (OrbitControls.js) 详见实验代码部分。

4 实验代码

4.1 main.js

创建摄像机、场景和渲染器，并将场景内的地球仪、桌子、网格背景等组件引入：

```
import('../js/light.js')
import('../js/earth.js')
import('../js/earthSupport.js')
// import('../js/table.js')
import('../js/table.min.js')
import('../js/plane.js')
import('../js/helper.js')

var scene = new THREE.Scene();

var camera = new THREE.PerspectiveCamera(45, window.innerWidth /
    window.innerHeight, 1, 1000);
var renderer = new THREE.WebGLRenderer({
  antialias: true,
  alpha: false
});
renderer.setSize(window.innerWidth, window.innerHeight);
renderer.shadowMap.enabled = true;
renderer.shadowMap.type = THREE.PCFSoftShadowMap;
renderer.setPixelRatio(window.devicePixelRatio);
```

```
renderer.setClearColor(new THREE.Color(0xE0FFFF));
document.body.appendChild(renderer.domElement);
camera.position.set(20, 0, 40);

var group = new THREE.Group();
group.position.set(0,-5.2,0);
scene.add(group);

var orbit = new THREE.OrbitControls(camera, renderer.domElement);
orbit.target.set(0,2,0);
orbit.update();
```

4.2 earth.js

创建地球：

```
var earthGeometry = new THREE.SphereGeometry(4, 100, 100);
var earthMaterial = new THREE.MeshPhongMaterial({
  map: new THREE.ImageUtils.loadTexture("img/earth.jpg"),
  color: 0xaaaaaa,
  specular: 0x333333,
  shininess: 25,
  side: THREE.DoubleSide,
});

var earth = new THREE.Mesh(earthGeometry, earthMaterial);
earth.position.set(0,5.6, 0);
earth.castShadow = true;
group.add(earth);

var render = function () {
  earth.rotation.y += 0.008;
  renderer.render(scene, camera);
  requestAnimationFrame(render);  //更新动画
};
render();
```

4.3 earthSupport.js

创建地球仪支架：

```
var metalMaterial = new THREE.MeshPhysicalMaterial({
  color:0xE9E9D8,
  // 材质像金属的程度. 非金属材料, 如木材或石材, 使用0.0, 金属使用
      1.0
  metalness: 1.0,
  // 材料的粗糙程度. 0.0表示平滑的镜面反射, 1.0表示完全漫反射. 默认
      0.5
  roughness: 0.6,
});

```

```
var pillarGeometry = new THREE.CylinderGeometry(0.3,0.3,3); //创建圆柱体
var pillar = new THREE.Mesh(pillarGeometry, metalMaterial);
pillar.position.set(0,1.5,0);
pillar.castShadow = true;

var standGeometry = new THREE.CylinderGeometry(2,2,0.2,100);
var stand = new THREE.Mesh(standGeometry, metalMaterial);
stand.position.set(0,0.1,0);
stand.castShadow = true;

var supportGeometry = new THREE.CylinderGeometry(0.1,0.1,0.4,100);
var support_up = new THREE.Mesh(supportGeometry, metalMaterial);
support_up.position.set(1.8,1.8,0);
support_up.rotation.set(0.1,0.1,0.4);
var support_down = new THREE.Mesh(supportGeometry, metalMaterial);
support_down.position.set(-1.8,9.2,0);
support_down.rotation.set(0.1,0.1,0.4);
support_up.castShadow = true;
support_down.castShadow = true;

var ringGeometry = new THREE.RingGeometry(4.2,4.3,100,100,90, Math.PI);  //创建圆环
// var ringMaterial = new THREE.MeshBasicMaterial( {
//  color: 0xff0 ,
//  });
var ring = new THREE.Mesh(ringGeometry, metalMaterial);
ring.position.set(0,5.5,0);
ring.castShadow = true;

group.add(pillar);
group.add(stand);
group.add(support_up);
group.add(support_down);
group.add(ring);
```

4.4 light.js

创建光源并开启阴影效果：

```
const light = new THREE.DirectionalLight( 0xffffff);
light.position.set( - 35, 40, 35 );
light.castShadow = true;

const d = 30;
light.shadow.camera.left = - d;
light.shadow.camera.right = d;
light.shadow.camera.top = d;
light.shadow.camera.bottom = - d;

light.shadow.camera.near = 0;
```

```
light.shadow.camera.far = 100;

light.shadow.mapSize.x = 1024;
light.shadow.mapSize.y = 1024;

scene.add( light );

var light0 = new THREE.AmbientLight(0x888888,0.8);
scene.add(light0);
```

4.5 OrbitControls.js

实现缩放和拖动的代码的核心部分如下：

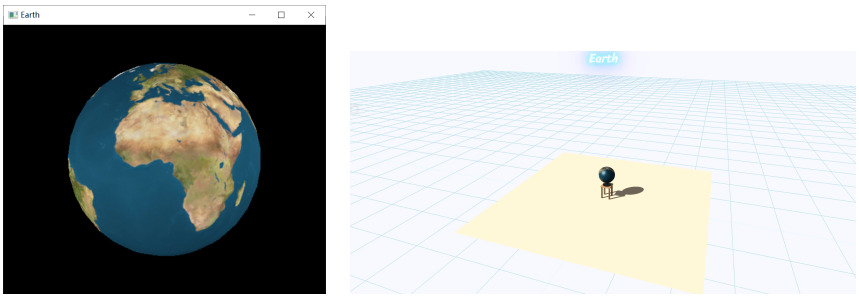
```
  function onMouseDown( event ) {
    if ( scope.enabled === false ) return;
    event.preventDefault();
    scope.domElement.focus ? scope.domElement.focus() :
        window.focus();
    switch ( event.button ) {
      case 0:
      switch ( scope.mouseButtons.LEFT ) {
        case THREE.MOUSE.ROTATE:
        if ( event.ctrlKey || event.metaKey || event.shiftKey ) {
          if ( scope.enablePan === false ) return;
          handleMouseDownPan( event );
          state = STATE.PAN;
        } else {
          if ( scope.enableRotate === false ) return;
          handleMouseDownRotate( event );
          state = STATE.ROTATE;
        }
        break;
        case THREE.MOUSE.PAN:
        if ( event.ctrlKey || event.metaKey || event.shiftKey ) {
          if ( scope.enableRotate === false ) return;
          handleMouseDownRotate( event );
          state = STATE.ROTATE;
        } else {
          if ( scope.enablePan === false ) return;
          handleMouseDownPan( event );
          state = STATE.PAN;
        }
        break;
        default:
        state = STATE.NONE;
      }
      break;

      case 1:
      switch ( scope.mouseButtons.MIDDLE ) {
```

```
37          case THREE.MOUSE.DOLLY:
38          if ( scope.enableZoom === false ) return;
39          handleMouseDownDolly( event );
40          state = STATE.DOLLY;
41          break;
42          default:
43          state = STATE.NONE;
44        }
45        break;
46
47        case 2:
48        switch ( scope.mouseButtons.RIGHT ) {
49          case THREE.MOUSE.ROTATE:
50          if ( scope.enableRotate === false ) return;
51          handleMouseDownRotate( event );
52          state = STATE.ROTATE;
53          break;
54          case THREE.MOUSE.PAN:
55          if ( scope.enablePan === false ) return;
56          handleMouseDownPan( event );
57          state = STATE.PAN;
58          break;
59          default:
60          state = STATE.NONE;
61        }
62        break;
63      }
64      if ( state !== STATE.NONE ) {
65        document.addEventListener( 'mousemove', onMouseMove, false );
66        document.addEventListener( 'mouseup', onMouseUp, false );
67        scope.dispatchEvent( startEvent );
68      }
69    }
70
71    function onMouseMove( event ) {
72      if ( scope.enabled === false ) return;
73      event.preventDefault();
74      switch ( state ) {
75        case STATE.ROTATE:
76        if ( scope.enableRotate === false ) return;
77        handleMouseMoveRotate( event );
78        break;
79        case STATE.DOLLY:
80        if ( scope.enableZoom === false ) return;
81        handleMouseMoveDolly( event );
82        break;
83        case STATE.PAN:
84        if ( scope.enablePan === false ) return;
85        handleMouseMovePan( event );
86        break;
87      }
88    }
```

```
89
90    function onMouseUp( event ) {
91      if ( scope.enabled === false ) return;
92      handleMouseUp( event );
93      document.removeEventListener( 'mousemove', onMouseMove, false )
            ;
94      document.removeEventListener( 'mouseup', onMouseUp, false );
95      scope.dispatchEvent( endEvent );
96      state = STATE.NONE;
97    }
98
99    function onMouseWheel( event ) {
100     if ( scope.enabled === false || scope.enableZoom === false || (
            state !== STATE.NONE && state !== STATE.ROTATE ) ) return;
101     event.preventDefault();
102     event.stopPropagation();
103     scope.dispatchEvent( startEvent );
104     handleMouseWheel( event );
105     scope.dispatchEvent( endEvent );
106   }
```

5 实验结果

5.1 OpenGL 实现（仅完成基本效果）



(a) OpenGL 地球　　(b) WebGL 地球

图 10: 两种实现

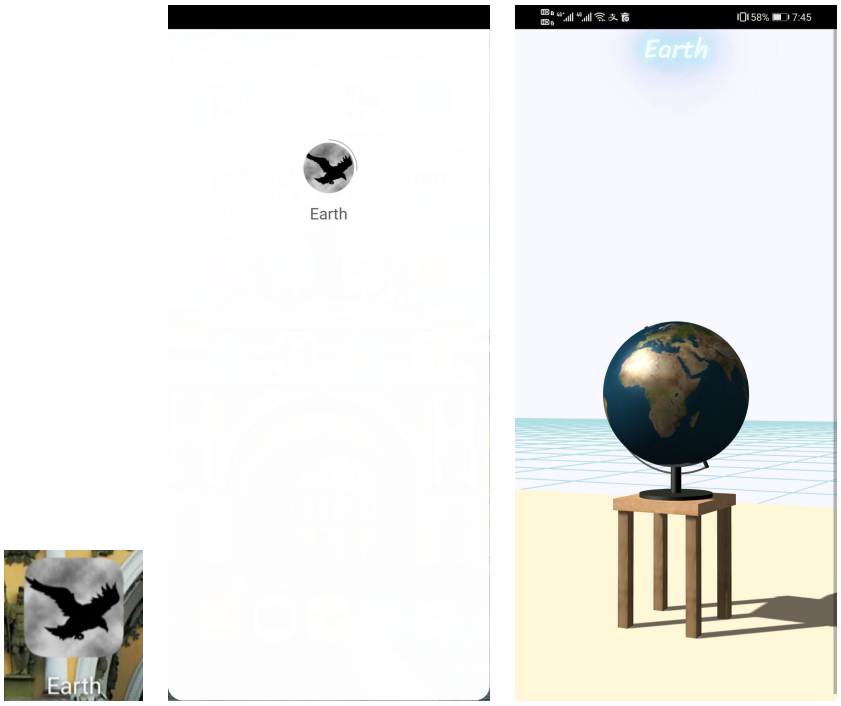
5.2 Javascript 语言实现：浏览器端

(a) 地球仪 无背景 普通支架　(b) 地球仪 无背景 金属支架　(c) 地球仪 有背景

图 11: 地球仪效果

5.3 Javascript 语言实现：APP 端

利用 HBuilder 自带的代码打包工具，生成 Android 端 apk 应用。



(a) APP 图标　(b) APP 载入界面　(c) APP 界面

图 12: 打包为安卓应用

6 实验总结

1. 通过这次作业，我对 3D 建模有了更深刻的理解，建模是要反复修改物体大小、位置、材质和旋转角度等属性来提高显示效果。

2. 在设计时要注重组件化编程，将不同的组件写在不同的文件中，方便后面组合使用和对个别属性的更改，也能使工程的结构更加清晰明了。

3. 在设置光照时，需要注意光源的选择和位置、亮度设置，以达到最好的显示效果。

4. 使用 Javascript 语言编程，可以将自己的工程迅速地部署到网页端，也可以生成移动端的应用文件。

5. C/C++ 涉及很多底层的知识，在编程中出现的有些错误可能不是很容易解决，而 Javascript 的 OpenGL 衍生库封装实现了很多功能，可以更快速地实现很多图形学的场景。

6. 这次作业让我在实践中体会了最基本的图形学知识的运用，也感谢老师在教学过程中的认真指导。