计算机图形学Project2项目报告

16307130167 赵广源

一、项目概述

编程画一个真实感静态实物。

二、代码实现

本次课程项目采用WebGL来完成真实感静态实物的绘制。一方面，通过WebGL绘制的图形可以很方便地在网页上展示出来，另一方面WebGL依赖的包少，对开发和运行环境没有特别的要求，可移植性强，适合轻量级的课程项目的实现。

- WebGL

  WebGL（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，这样以来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。

- HTML Canvas

  HTML5<canvas>元素提供了一个通过JavaScript来绘制图形的方式。它可以用于动画、游戏画面、数据可视化、图片编辑以及实时视频处理等方面。

使用WebGL原生的API来写3D程序是一件非常痛苦的事情，好在现在有很多WebGL的开源框架可供使用，其中GitHub上的开源项目`three.js`就是非常优秀的一个，它去除了很多麻烦的细节，使用方法也非常简单。

```
//Download the minified library and include it in your HTML, or install and
import it as a module, Alternatively, see how to build the library yourself.

<script src="js/three.min.js"></script>
```

接下来的内容将对代码实现过程中的关键步骤做分析和说明：

在`three.js`中，要渲染物体到网页中去，需要3个组件：渲染器（renderer），相机（camera），和场景（scene）。

首先设置WebGL渲染器，其中`renderer.shadowMap`属性的设置指定了在场景中使用Percentage-Closer Soft Shadows (PCSS) 算法来过滤阴影映射，并且启用了阴影自动更新。渲染器`renderer`的`domElement`元素表示渲染器中的画布，所有的渲染都是画在`domElement`上的，这里的`appendChild()`表示将这个`domElement`挂接在body下面，这样渲染的结果就能够在页面中显示了。

```
    renderer = new THREE.WebGLRenderer({antialias: true, precision: 'highp'});
    renderer.setSize(window.innerWidth, window.innerHeight);
    renderer.shadowMap.enabled = true;
    renderer.shadowMap.autoUpdate = true;
    renderer.shadowMap.type = THREE.PCFSoftShadowMap;
    document.body.appendChild(renderer.domElement);
```

其次设置相机，相机决定了场景中那个角度的景色会显示出来。在`three.js`中相机的表示是`THREE.camera`，它包括两种相机类型，正投影相机和透视投影相机，这里选用的是透视投影相机。`PerspectiveCamera()`中的四个参数分别表示相机的视角大小，近平面距离和远平面距离和纵横比。

```
    camera = new THREE.PerspectiveCamera(45, window.innerWidth /
window.innerHeight, 0.1, 1000);
    camera.position.x = 0;
    camera.position.y = 70;
    camera.position.z = 70;
    camera.lookAt(new THREE.Vector3(0, 0, 0));
```

最后设置场景，场景的设置比较简单，在这里不做赘述。

完成了基本的设置之后，`initLight()`函数设置了环境光（`THREE.AmbientLight`）和聚光灯（`THREE.SpotLight`），`CreateEarth()`制作了地球的3D模型。

除此之外，为了绘制一个真实的地球出来，还需要在制作的球体上面添加纹理，也就是贴图，把地球表面的图像贴上去。在`CreateEarth()`函数调用的`CreateSphere()`函数中，`THREE.MeshPhongMaterial`指定了球体表面所用的纹理图片，`THREE.Mesh`把模型的形状和材质相连接。

```
    var sphereGeometry = new THREE.SphereGeometry(r, w, h);
    var sphereMaterial = new THREE.MeshPhongMaterial({map:new
THREE.TextureLoader().load(pic)});
    var sphere = new THREE.Mesh(sphereGeometry, sphereMaterial);
```

为了实现与场景的交互，引入了`three.js`的轨道控制器`OrbitControls.js`，控制场景的旋转，平移，缩放等，同时引入了`three.js`的辅助库`stats.js`来测试WebGL代码的渲染性能，检测运行时的帧数。

```
<script type="text/javascript" src="./OrbitControls.js"></script>
<script type="text/javascript" src="./stats.js"></script>

...

var stats;
function initStats() {}

var controls;
function initControls() {}
```

最后为项目编写一个简单的展示界面。

```
<!DOCTYPE html>
<html>
<head>
    <style type="text/css">
        html, body { margin: 0; height: 100%; }
        canvas { display: block; }
    </style>
</head>
<body onload="main();">
</body>
<script type="text/javascript" src="./three.min.js"></script>
<script type="text/javascript" src="./OrbitControls.js"></script>
<script type="text/javascript" src="./stats.js"></script>
<script type="text/javascript">
...
</script>
```

```
</html>
```

三、项目运行

由于浏览器不允许资源的跨域访问，因此直接打开`HTML`文件无法显示模型的纹理（`"./earth.jpg"`）。对这一问题有以下两个解决方法：

1. 把项目部署在本地的Web服务器上。
2. 修改浏览器的运行参数（以chrome为例）：
    - 在命令行中用以下命令打开浏览器：`Chrome installation path\chrome.exe --allow-file-access-from-files`；
    - 或者临时复制一个chrome的快捷方式，右键在属性窗口的目标一栏添加运行参数：`"Chrome installation path\chrome.exe" --allow-file-access-from-files`，之后重新启动chrome。

四、结果展示

真实感静态实物效果如下图所示。