Fundamental Framework of GAMES 202

Mengzhu Wang

2024 年 5 月 15 日

1 Content

GAMES 202 代码框架基于 WebGL 实现。

2 index.html

通过访问相对地址引入 js 文件。body 部分<canvas>标记创建了绘制图形，js 文件可以用定义的 id "glcanvas"访问这个 canvas。

3 Engine

用关键字function定义主函数GAMES202Main。调用函数前，index.html定义的页面对应的 DOM 树已经生成。在GAMES202Main直接获取页面中定义的canvas元素。然后将长宽更新为屏幕视口的长宽。创建一个WebGLRenderingContext对象作为 3D 渲染的上下文。

接下来准备渲染所需的参数。首先用THREE.js中的PerspectiveCamera创建相机实例，该函数需要的参数为 FOV、长宽比、近平面和远平面，以及定义相机位置。为实现鼠标和场景的交互，增加OrbitControls，允许缩放、旋转、移动，并设置速度。监听resize事件，在窗口缩放时更新相机长宽比和投影变换矩阵。

创建场景所需的光源PointLight，并初始化亮度和位置。

创建一个WebGLRenderer渲染器，将 WebGL 渲染上下文和相机都绑定到其上。然后向其添加光源以及调用loadOBJ载入模型。

用THREE.js中的dat.gui创建 GUI 实例，增加数据面板和可调整的参数。

最后是一个循环，每一次循环都先通过OrbitControls定义的cameraControls更新相机参数，再使用WebGLRenderer定义的renderer更新canvas元素中的渲染内容。requestAnimationFrame会定义一个回调函数，不需要设置间隔时间，屏幕刷新一次回调函数跟着执行一次，保证了动画的流畅。

4 Texture

Texture类通过传入WebGLRenderingContext和HTMLImageElement来构造WebGLTexture、设置一系列参数、拷贝纹理数据。

5 Shader

从Material端调用Shader的构造函数，Shader.js中通过传入顶点着色器 vertex Shader 和片元着色器 fragment Shader 的源码来编译并链接形成完整的着色器程序。

vertex shader 主要处理顶点数据和顶点变换。attribute只能在 vertex shader 使用，一般表示一些顶点数据，如顶点坐标、法线、纹理坐标等。uniform可以在 vertex 和 fragment 间共享，一般表示变换矩阵、光照参数等。varying用于 vertex 和 fragment 之间的数据传递，一般 vertex

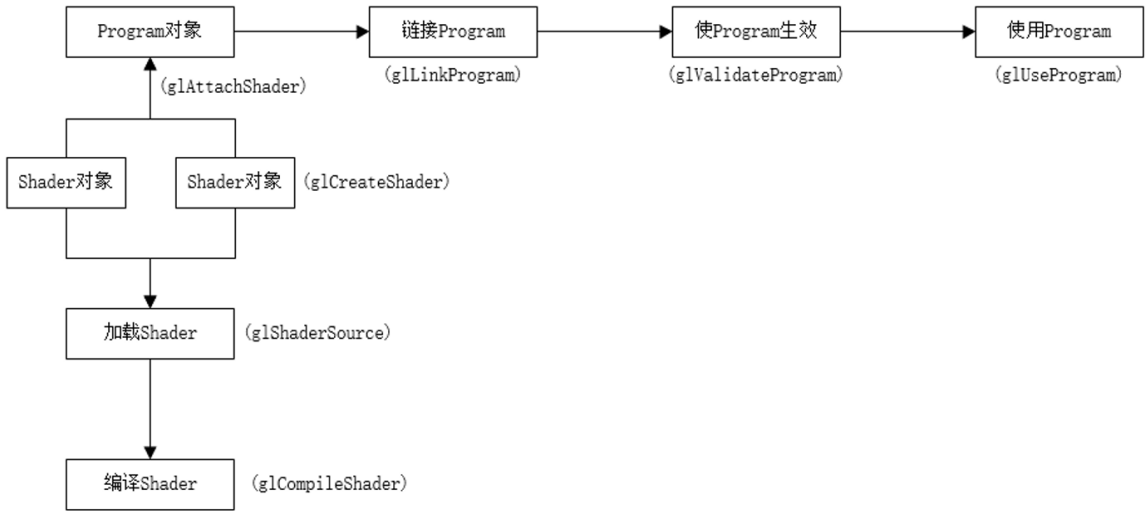
shader 修改varying变量的值，然后 fragment shader 使用该varying变量的值。vertex shader 的输出一般有经过 MVP 变换后的顶点坐标gl_Position和赋值后的varying变量。

fragment shader 主要处理颜色、贴图、光照阴影等计算。它声明全局uniform变量和继承自 vertex shader 的varying变量。glsl 的精度限定符有lowp、mediump和highp，分别对应低、中、高精度，可以用于修饰变量以规定精度。fragment shader 的输出一般是gl_FragColor。

得到着色器源码后，对其进行编译。具体包括构造、编译以及判断是否编译成功，最后输出生成的 shader。

生成着色器后，还需要生成程序对象。程序对象是管理着色器对象的容器，可以实现 shader 与 shader 之间数据的传递，它是着色器之间数据沟通的连接。WebGL 中一个程序对象必须包含一个顶点着色器和一个片元着色器。过程包括程序对象生成，分配着色器attachShader，连接着色器使得变量同名同类型且一一对应。

得到与着色器链接的程序glShaderProgram后,会与shaderLocations对象一同被载入。shaderLocations包括 attributes 和 uniforms, 经过addShaderLocations后, 当需要给这些uniform和attribute赋值时，就可以直接通过已经获取到的位置进行操作，而不需要每次都去查询位置。



6 Renderer

在engine.js中调用WebGLRenderer类渲染场景。首先传入 WebGL 渲染上下文和相应的透视投影相机进行构造。添加一组点光源，除了保存点光源外，使用光源对应的 mesh 和材质创建一个光源对应的MeshRender，并添加一个MeshRender。清空画布，让场景中的光源以一种特定的周期性空间曲线路径移动。遍历场景中每一个光源，用当前的新的lightPos更新并渲染其 mesh。如果没有任何光源，则直接渲染 mesh。

MeshRender.js提供MeshRender类保存渲染 mesh 所需的 mesh 对象（顶点、法线、纹理坐标）、材质、着色器实例。创建WebGLBuffer通过 attributes 将 JavaScript 端的数据传递给 WebGL。

MeshRender还提供draw函数，先向 WebGL 通过设置 attributes 来传递各类顶点坐标、法线、纹理坐标等数据，接着通过gl.uniformXXX系列方法从材质实例和draw函数参数中读取并设置 shader 的各类uniforms渲染参数。所有参数设置完成后，使用顶点索引渲染模型。

7 Object

Mesh.js存储网格体，需要传入四个参数verticesAttrib、normalsAttrib、texcoordsAttrib和indices。前三个参数的数据类型为 AttribType，最后一个是顶点索引。

8 Material

光栅化渲染中，材质定义了为该表面某一个像素进行着色的过程中所需要的各类参数，以及如何使用这些参数和场景中的各类信息来计算该像素颜色的值。

Material类中flatten_uniform存储渲染所需的uniform变量,flatten_attribs存储渲染所需的attribute 变量。从源码编译 shader，返回着色器程序和各类attribute，uniform变量名与地址的对应关系。

9 Load

loadOBJ.js首先创建THREE.LoadingManager观察加载进度。接着创建一个MTLLoader用来加载mtl格式的材质库文件，并调用load函数、传入加载完成后的回调函数，在加载完成后调用回调函数、传入加载好的材质（纹理）相关数据。传入MTLLoader的回调函数里创建了一个OBJLoader用于加载obj模型的顶点、法线、纹理坐标等信息，该加载器同样具有加载完成的回调函数，该回调函数中传入了加载完成的模型实例，包含顶点、纹理等数据，用这些数据创建 Mesh，Texture，Material等数据实例，最终创建出一个MeshRender并添加到WebGLRenderer中。

loadShader.js使用Promise，加载完成后调用resolve表示加载完成。外部使用时可以在async函数中使用await等待resolve执行完成，也可以直接对Promise调用then方法获取该Promise中调用resolve所传入的参数，实现异步加载 shader 源码文件。

10 Light

光源有自己的 mesh 和材质。Light.js定义了自发光材质EmissiveMaterial，包括光强和颜色两个属性。PointLight.js定义了光源的材质和自身的 mesh。