<http://www.pigzz.com/201109/092YI42011.html>

var vertexShader = getShader(gl, "shader-vs");

shaderProgram = gl.createProgram();

gl.attachShader(shaderProgram, vertexShader);

gl.attachShader(shaderProgram, fragmentShader);

gl.linkProgram(shaderProgram);

if (!gl.getProgramParameter(shaderProgram, gl.LINK\_STATUS)) {

alert("Could not initialise shaders");

}

gl.useProgram(shaderProgram);

正如你所见，它使用getShader函数来获得两个渲染器：一个片段渲染器和一个顶点渲染器，接着将两者绑定在一个WebGL“程序”上。一个程序是一段放置在系统WebGL上的代码；你可以把它视作一种运行在图形卡上的特定方式。正如你所期望的，你可以将它和一些渲染器联系在一起，每个渲染器都可以视为程序中的一个代码片段；确切地说，每个程序可以拥有一个片段渲染器和一个顶点渲染器。让我们简单地看一下：

JavaScript Code复制内容到剪贴板

shaderProgram.vertexPositionAttribute =

gl.getAttribLocation(shaderProgram, "aVertexPosition");

gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);

一旦设置好程序并绑定了渲染器，函数将得到一个“属性”的引用，该属性存储在vertexPositionAttribute对象中。我们再次利用JavaScript把任一字段设置在任一对象上；默认情况下对象没有vertexPositionAttribute字段，但是对于我们来说将两个值保留在一起是很方便的，因此我们仅设置程序中新字段的属性。

那么，vertexPositionAttribute是做什么的呢？也许你还记得，我们在drawScene函数中使用过它；如果你回过去看一看从适当的缓冲区设置三角形顶点位置的那段代码，你将看到我们所做的就是将缓冲区与该属性关联在一起。稍后你将明白这是什么意思。现在，只需注意到我们也使用gl.enableVertexAttribArray函数来指示WebGL使用一个数组来为该属性提供数值。

JavaScript Code复制内容到剪贴板

shaderProgram.pMatrixUniform = gl.getUniformLocation(shaderProgram, "uPMatrix");

shaderProgram.mvMatrixUniform = gl.getUniformLocation(shaderProgram, "uMVMatrix");

initShaders函数所做的最后一件事就是从程序中获取两个多的值，这两个变量称作uniform变量，我们很快会再遇见它们，但现在你只需注意到，正如属性一样，我们为了方便而将其存储在对象中。

现在，我们来看看getShader函数：

JavaScript Code复制内容到剪贴板

function getShader(gl, id) {

var shaderScript = document.getElementById(id);

if (!shaderScript) {

return null;

}

var str = "";

var k = shaderScript.firstChild;

while (k) {

if (k.nodeType == 3)

str += k.textContent;

k = k.nextSibling;

}

var shader;

if (shaderScript.type == "x-shader/x-fragment") {

shader = gl.createShader(gl.FRAGMENT\_SHADER);

}

else if (shaderScript.type == "x-shader/x-vertex") {

shader = gl.createShader(gl.VERTEX\_SHADER);

}

else {

return null;

}

gl.shaderSource(shader, str);

gl.compileShader(shader);

if (!gl.getShaderParameter(shader, gl.COMPILE\_STATUS)) {

alert(gl.getShaderInfoLog(shader));

return null;

}

return shader;

}

这是另一个比看起来要简单的函数。我们要做的是在HTML网页中寻找一个元素，其具有与传入参数匹配的ID，取出其内容并基于其类型创建一个片段渲染器或者一个顶点渲染器（以后我们将更多地解释它们的区别），接着将其传入到WebGL中编译成可以在图形卡上运行的形式。接下来，代码进行出错处理，最后完成整个处理。当然，我们只能在JavaScript中将渲染器定义为字符串而不能从HTML中提取——通过这样做，我们使其更易读，因为它们被定义为网页中的脚本，就像它们本身就是JavaScript一样。

看完这个以后，我们应该来看看渲染器的代码：

关于这些你要记住的第一件事就是：这些代码不是用JavaScript所写，即使这两种脚本语言的祖先十分相似。事实上，它们使用一种特殊的与C语言有很大关系的渲染器语言（当然，JavaScript也是如此）所写。第一个渲染器——即片段渲染器——什么也不做；它简单地规定了被绘制的物体将被绘制成白色（怎么给物体着色是下一节课程的话题）。第二个渲染器有点意思，它是一个顶点渲染器——还记得吧，它是一段图形卡上的代码，能用一个顶点完成它想做的任何事。与之相关联的是，它有两个uniform变量：uMVMatrix和uPMatrix。uniform变量十分有用，因为它们能在渲染器之外获得——实际上是在包含它们的程序之外，你可能还记得，当时我们从initShaders函数获得了它们，它们也可以从将其设置为模型视图矩阵和投影矩阵的代码（我敢肯定你已经实现了它）中获得。你可能认为渲染器程序是一个对象（在面向对象的场景中），而统一变量是对象中的字段。现在，渲染器在每个顶点上调用，顶点作为aVertexPosition参数传入到渲染器的代码中，由于在drawScene函数中使用vertexPositionAttribute，此时，我们将其属性与缓冲区关联在一起。渲染器主程序中的这部分代码只是将顶点与模型视图矩阵和投影矩阵相乘，然后作为顶点最终位置的结果传出。webGLStart函数调用initShaders函数，在网页的脚本中使用getShader函数装载了片段渲染器和顶点渲染器，以便它们能被编译后传入到WebGL，最终使用WebGL渲染出三维场景。

剩下还没有说明的代码是setMatrixUniforms函数，一旦你理解了上面所讲的，就很容易理解它。

JavaScript Code复制内容到剪贴板

function setMatrixUniforms() {

gl.uniformMatrix4fv(shaderProgram.pMatrixUniform, false,

new WebGLFloatArray(pMatrix.flatten()));

gl.uniformMatrix4fv(shaderProgram.mvMatrixUniform, false,

new WebGLFloatArray(mvMatrix.flatten()));

}

通过引用uniform来表示initShaders中的投影矩阵和模型视图矩阵，我们将值从JavaScript类型矩阵传递给了WebGL。