CSGrandeur的WebGL学习笔记

CSGrandeur

Published
with GitBook

目錄

WebGL学习

简介

WebGL是什么

HTML5是当前HTML的新标准，其中一个特性就可以用JavaScript写调用显卡的程序，在新的网页元素Canvas上显示华丽的3D效果，开发WebGL主要使用 JavaScript语言。

我学WebGL目前有两点目的：

1. Web应该是跨平台最舒服的手段了，没有哪个现代操作系统（哪怕手机系统）不配浏览器吧，HTML5的到来，感觉Web前途一片光明。
2. JavaScript开发3D好像舒服很多啊，用C++写OpenGL，编译老半天，调试一步一步扯得蛋疼。

当然这两点目的可能很傻很天真，不过学点有用的东西总是好的，也不必太较真儿了。

学习WebGL需要什么基础

开始学习WebGL时我的基础：

- 具备一点PHP、HTML、JS的基础知识，做过几个Web的小项目。
- 做过几年程序设计竞赛，对C/C++语言还算比较熟悉。
- 跟着《OpenGL编程指南》第八版学过一点入门知识，懵懂了解可编程渲染管线的概念。

建议：

- 至少具备一些基本的编程技能，理解变量、函数等基础知识。
- 不必担心对Web相关领域的陌生，毕竟那不是重点，也不会成为学习WebGL过程中的难点。
- WebGL的学习过程和OpenGL是相似相通的，所以也不必担心是否需要OpenGL的基础知识

这本书干什么

这本书是我学习WebGL过程的记录，会根据学习的经验写成教程的形式。

第一章是“Learning WebGL”的内容，部分是翻译，部分是自己不懂的地方查阅资料补充的，也省略了一些觉得不必解释过细的内容，可以理解为“Learning WebGL”的一个中文版。之后学习的内容会开辟新的章节。

这本书的GitBook主页：https://www.gitbook.com/book/csgrandeur/webgl-learn

对应的Github主页：https://github.com/CSGrandeur/csgrandeur-webgl-learn

代码演示：http://csgrandeur.github.io/WebGL_Learn_Code

初学者，难免有错误与疏漏，欢迎指正，我会及时修改更新。

参考

Learning WebGL 他的网站是：http://learningwebgl.com 。

```
By CSGrandeur, 2015年4月28日
```

第一章 起步

本章从一个能够运行的WebGL基础代码开始

我们不考虑HTML的结构，比如header了body了什么的，只关注WebGL相关代码，而且这样的代码是可以用的~~

当然为了更好的体验，可以网上找个HTML页面代码，再加上些css文件弄美观些，然后把WebGL代码写在<body></body>里。

准备工作

浏览器

比较新的浏览器基本没什么问题，一般决定学习WebGL的同学多少有点极客精神，不至于用太老旧的浏览器吧。

目前来说Chrome一般是开发者的首选，如果你用的不是Chrome浏览器，随便上网搜一些WebGL的例子打开看自己的浏览器能不能正常显示应该也差不多。

如果我什么时候弄了个服务器，就把每一节的练习源码放上去，到时候会把链接放在这里，到时也可以用这些页面测试浏览器是否支持。

所以，先准备一个最新的浏览器吧。

开发环境

Web一般是用浏览器来看，所以用不到什么编译器，那么像UltraEdit、Sublime，或者不怎么极客的我没用过的Vim和Emacs口碑这么好应该也可以。

之前做项目已经习惯了Eclipse，可以装各种扩展，语法高亮、代码提示什么的都不错，就是配置到很顺手还是要花点功夫的。

建议做个本地服务器，毕竟WebGL是在Web上的，有个服务器调试代码能有更好的Web体验。我就用XAMPP了，装上之后根目录的htdocs里放自己的Web页面，就可以在浏览器用127.0.0.1访问了。当然也可以修改它的apache配置文件满足自己的其他需要，不再赘述。

如何调试

前端程序员应该很熟悉，大多浏览器用F12可打开调试器，在Console里查看页面错误。

可以利用调试器点选网页元素，看元素的代码位置、内容和相关的css，帮助解决一些基本的前端问题。 太高深的我也不懂，就不多说了。了解一些前端知识总是有好处的。

使用的外部库

jQuery

jQuery是一个家喻户晓的JS库，大家一般不太喜欢直面JS原生代码，使用jQuery会方便很多。大家都在用，不用担心冷门什么的，而且也不会影响对WebGL知识的学习。

jQuery官网：http://jquery.com/

jQuery开源主页：https://github.com/jquery/jquery

可以在自己WebGL代码页面的最顶端加上一行

```
<script src="jquery.min.js" type="text/javascript"></script>
```

当然src=要加上能找到jquery.min.js文件的相对路径。.js前面有个min的 意思是被压缩过的js，功能和不加min的一样，但文件更小加快网页打开速度。如果需要调试时候查看源码，就用不带min的版本。

也可以用CDN：

```
<script src="http://code.jquery.com/jquery-2.1.4.min.js"
type="text/javascript"></script>
```

用CDN的前提是查看网页的这台电脑要联网。

gl-matrix

做GL会涉及很多图形变换，在计算机中对坐标的变换是通过矩阵来实现的，比如可以去了解一下仿射变换。WebGL并没有集成矩阵的运算，我们专注WebGL的学习，没有必要一开始花费太多精力去写矩阵操作，轮子造好了，请享用。

gl-matrix开源主页：https://github.com/toji/gl-matrix

仓库的dist文件夹下有gl-matrix-min.js和gl-matrix.js两个文件。同样在页面顶端加入

```
<script src="gl-matrix-min.js" type="text/javascript"></script>
```

好，我们开始吧 

三角与矩形

首先记得在自己的代码最顶部加上本章开头说的**jQuery**与**gl-matrix**的引入代码，后面我们的示例代码中不再说明。

本节我们的目标是在Canvas里画一个如图1的三角和方块。

图1

每节的内容都可以去本书简介中给的代码演示地址查看具体效果，可以从页面查看源码。附录1也给出了每节的WebGL代码。

放控件

首先把canvas放到网页上

```
<canvas id = "test01-canvas" width = "800" height = "600"></canvas>
```

canvas是画布的意思，它不止可以做WebGL，还可以做画图等其他事情。

画布的大小需要在这里设置或用JS修改，但如果用css文件去设置尺寸的话，会发现最后绘制出来的东西被扭曲了，因为css改变了控件的大小却没有改变画布大小。

id是控件的唯一标识，原则上你不能写两个canvas用同一个id。我们会在JS代码中通过这个id找到这个canvas。

程序的入口

接下来就开始我们的JS代码了。再提一次我们前面说过引入jQuery，下面的代码虽然用到jQuery地方很少，但是不引入jQuery的话代码是跑不动的。

```
<script type="text/javascript">
```

HTML文件中插入JS代码，需要先加入标签，JS代码写在标签下面。结束的时候当然对应也要加上：

```
</script>
```

好，JS代码开始。

```
$(document).ready(function ()
{
    webGLStart();
});

function webGLStart()
{
    var canvas = $("#test01-canvas")[0];
    initGL(canvas);
    initShaders();
    initBuffers();

    gl.clearColor(0.0, 0.0, 0.0, 1.0);
    gl.enable(gl.DEPTH_TEST);

    drawScene();
}
```

“$(document).ready(function (){});”是jQuery语法中，表示函数内的内容在Web页面加载完成时运行。如果不这样，可能在Web尚未加载完的时候，JS就开始尝试找某个控件，结果扑个空，代码逻辑就不能正常执行了。

webGLStart函数是一切的开始，首先我们用canvas的id“test01-canvas”找到它，$("#xxx")是jQuery用id找控件的方法，用class找的话就是$(".xxx")。找到之后得到的是jQuery对象，这里用一个数组下标[0]获取canvas的HTML对象，然后赋值给我们定义的canvas变量。

接下来三步是对不同部分的初始化，我们后面会说。

gl是我们自己定义的变量名，对应绘制的内容，后面会说它的初始化。

好比canvas是画布，gl是画笔，我们gl.clearColor四个参数，前三个RGB，第四个不透明度，这句就是让gl把整个canvas涂黑。gl.enable用来开启一些功能，这里DEPTH_TEST是深度测试，现在我们先不管它有什么用，写到这就是。

drawScene()就是具体绘制操作的函数，也是我们自己定义的，后面会说。

一套代码流程的整体思路就是：

1. JS找到画布(canvas)->
2. 从画布拿到画它的工具(gl)->
3. 设置要画的内容（点的坐标，点之间的拓扑关系，存放到buffer里）->
4. 告诉显卡要怎么画（从哪个方向看，点线面的颜色之类，用shader来交流）->
5. 画！

初始化“画笔”

```
var gl;
function initGL(canvas)
{
    try
    {
        gl = canvas.getContext("webgl") ||
            canvas.getContext("experimental-webgl");
        gl.viewportWidth = canvas.width;
        gl.viewportHeight = canvas.height;
    }catch(e){}
    if(!gl)
    {
        alert("无法初始化“WebGL”。");
    }
}
```

这里我们定义了全局变量 gl，一般全局变量是不提倡的，第一节我们尽量少关注其他问题，先这样做，解释起来方便一些。

canvas在入口里已经获取到了，canvas.getContext来拿到它的“内容”给gl画，之后gl无论画什么，都是在这个canvas的画布上。参数用了“experimental-webgl”和“webgl”并通过或运算连起来，因为webgl标准存在一个“实验阶段”，要通过“experimental-webgl”来获取内容，我现在直接只用“webgl”就可以，不过为了兼容性和其他不确定问题，两个都写上总保险些。

获取canvas的宽高，让gl“随身携带”，后面会用到这两个数据来设置视口。

设置要画的内容

如果一个点一个点告诉显卡的话，累死我们，急死显卡。 把要画的内容所有的点存到一个缓冲区，对应显卡内存一块区域，让显卡一起处理，对数据规模较大又要即时演算做动画的情况，提高效率。

```
var triangleVertexPositionBuffer;
var squareVertexPositionBuffer;
```

分别定义三角的buffer和矩形的buffer，刚定义的时候它们当然只是个什么都没有的变量。

```
function initBuffers()
{
    triangleVertexPositionBuffer = gl.createBuffer
```

用gl给他们申请buffer。

```
    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
```

告诉WebGL，下面的代码在有新的变化之前，对gl.ARRAY_BUFFER操作就是对绑定的缓冲区（triangleVertexPositionBuffer）操作。

```
    var vertices = [
                    0.0,  1.0,  0.0,
                   -1.0, -1.0,  0.0,
                    1.0, -1.0,  0.0
                    ];
```

定义三角形的三个顶点，我们现在画平面的形状，z坐标就设为0。

```
    gl.bufferData(gl.ARRAY_BUFFER,
        new Float32Array(vertices), gl.STATIC_DRAW);
```

前面已经绑定了buffer，这里把定义的三角形顶点坐标数组告诉gl.ARRAY_BUFFER，也就是告诉了triangleVertexPositionBuffer。第三个参数STATIC_DRAW理解为“这数据的用途”，Float32Array和STATIC_DRAW这里先直接用，以后再解释。

```
    triangleVertexPositionBuffer.itemSize = 3;
    triangleVertexPositionBuffer.numItems = 3;
```

itemSize和numItems并不是WebGL的内置变量，不过JavaScript这方面比较自由，变量不用定义就能用，让相关对象自己携带相关信息方便许多。这里增加的信息就是明确“buffer里由vertices给出的九个数，表达的是三个顶点”。

```
    squareVertexPositionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexPositionBuffer);
    vertices = [
                 1.0,  1.0,  0.0,
                -1.0,  1.0,  0.0,
                 1.0, -1.0,  0.0,
                -1.0, -1.0,  0.0
                ];
    gl.bufferData(gl.ARRAY_BUFFER,
        new Float32Array(vertices), gl.STATIC_DRAW);
    squareVertexPositionBuffer.itemSize = 3;
    squareVertexPositionBuffer.numItems = 4;

}
```

同理我们设置了矩形的数据。

告诉显卡怎么画

看《OpenGL编程指南》的时候，到这个地方就蒙了。主要还是以前不知道显卡的工作流程。

我们先搞懂三个名词：渲染、管线、着色器。定义在网上都找得到，我说说我的理解，虽然可能比较狭义（甚至可能有一些误解），但能帮助我们理解程序。

- 渲染：

是一个或一组动作。

一般我们看电影动画片，都是一帧一帧的“图片”，是二维数据，把他们适当换算后打到屏幕的每个像素上。

而图形，要把表述三维形状的数据，转化成可以在二维屏幕上显示的画面，多了一个维度就多了非常多的运算量。

我们在现实世界从某个角度看一些物体，其实也是经过“计算”的——引用知乎一个有趣的民间提法：上帝算法——就是物理规律决定哪个物体遮挡了哪个物体，哪个物体是什么颜色，然后进入到我们的眼睛里，映射到视网膜这个“屏幕”上。在计算机中，这些三维数据表示的虚拟物体和我们设定的、虚拟的、看这些物体的“眼睛”或者说“相机”，对应的这个“看”的“物理过程”则是需要GPU来算的。

GPU把三维数据按照我们需要的观察角度、光照环境、物体颜色等属性算出映射到“眼睛”里的平面图像，这个过程理解为渲染。

渲染一般对应英文中“render”这个词。

- 管线：

一般连起来称为渲染管线。渲染不是一下子完成的，根据任务目标（坐标变换，光照计算什么的）和硬件执行特点，标准制定者制定了渲染过程的多个步骤，它们就像制造工厂的流水线一样，一个步骤对数据处理后的结果，交给下一个步骤处理，多个步骤连起来，就成了一个流水线，这个流水线就是渲染管线。

过去渲染管线是固定的，我们把准备好的特定格式的数据送给GPU，在渲染管线里走一趟，得到二维图像显示在屏幕上。

现在允许对渲染管线的某些步骤进行个性化修改，灵活自由些总是更好的，至少我们可以让有些GPU比CPU擅长的工作，从CPU上搬到GPU上来运行了。

- 着色器

管线的一些步骤可以修改了，按照面向对象的思想，一个特定的管线上的特定步骤的任务调整，就有一个特定的“小东西”去负责，这个“小东西”就是着色器。

调整一个步骤的具体任务，需要用GPU看得懂的方式来表达，这就是着色器语言（GLSL）。别害怕，这并不是说我们又要马上学一门陌生的语言。幸运的是GLSL长的跟C语言真像，写起来和C语言一个手感哦~~在学习的过程中我们会慢慢来了解它具体的特性

着色器一般对应英文中的“shader”这个词。

WebGL主要是片段着色器（fragment）和顶点着色器（vertex）这俩，用着用着，结合它们的名字我们就会熟悉它们分别做什么的了。

打个简单的比方：

三个点的坐标->全部沿x轴左平移5->全部涂成红色->输出

这就是一条渲染管线了，中间的两个操作对应两个不同功能的shader。

好，回到这一节的代码，我们现在要告诉显卡怎么“画”。三步走：写自己的shader，把写好的shader按流水线连起来，告诉显卡。

```
<script id = "shader-fs" type = "x-shader/x-fragment">
    precision mediump float;
    void main(void)
    {
        gl_FragColor = vec4(1.0, 1.0, 1.0, 1.0);
    }
</script>
<script id = "shader-vs" type = "x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;
    void main(void)
    {
        gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
    }
</script>
```

为了方便，我们也把GLSL的代码写到页面里了，但是用<script>圈了起来，不然的话HTML会把这些代码直接显示到页面的。我们这一节的WebGL代码是写在类似的一个<script>里，这几行<script>包围的GLSL代码要和WebGL代码并列，即写在主体WebGL代码<script></script>的外面，具体参考本节末尾的完整代码。

上面就是基础功能的片段着色器（fragment shader）和顶点着色器（vertex shader）。shader不一定要以这种形式保存，其实我们的GLSL代码可以写在某个变量里，别的控件里，单独的文件里，甚至别人的网站里。。。只要WebGL在需要编译链接shader的时候，能用某种方法拿到整个GLSL代码的字符串就可以了。

我们这里把shader直接写在页面里，就要写一个对应的方法在WebGL运行时拿到这些代码的字符串。

```
function getShader(gl, id)
{
    var shaderScript = $("#" + id);
    if(!shaderScript.length)
    {
        return null;
    }
    var str = shaderScript.text();

    var shader;
    if(shaderScript[0].type == "x-shader/x-fragment")
    {
        shader = gl.createShader(gl.FRAGMENT_SHADER);
    }
    else if(shaderScript[0].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;
}
```

两个参数，一个是我们之前说过的gl，另一个是页面里那两个保存着shader代码（GLSL）的控件的id，“shader-fs”或“shader-vs”。

- 用jQuery取得控件->
- 判断有没有“扑个空”->
- 获取内部字符串（GLSL代码）->
- 判断控件的type标签是fragment还是vertex（id和标签都是自己定义的，对应上就好）->
- 用gl新建相应类型的shader对象，把代码字符串交给对象（shaderSource()）->
- 编译shader->
- 查看编译是否出错（就像编译C语言一样，也会研究语法错误什么的）->
- 都顺利的话这个函数就返回了编译好的shader

```
var shaderProgram;
function initShaders()
{
    var fragmentShader = getShader(gl, "shader-fs");
    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("无法初始化“Shader”。");
    }
    gl.useProgram(shaderProgram);
```

拿到编译好的shader，就该把shader告诉显卡了。我们得把两个独立的shader连城流水线。

用gl申请一个Program给shaderProgram，把两个shader交给shaderProgram，用shaderProgram链接它们，然后交给“画笔”gl.useProgram(shaderProgram)。

```
    shaderProgram.vertexPositionAttribute =
        gl.getAttribLocation(shaderProgram, "aVertexPosition");
    gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);
```

后面画的时候需要设置GLSL里面的某些变量与输入数据的关系，所以需要gl.getAttribLocation(shaderProgram, "aVertexPosition")得到GLSL中aVertexPosition这个变量的“位置”。 拿

到这个位置，可以就让shaderProgram随身带着，shaderProgram.vertexPositionAttribute是我们自己加上去的（还是JS的特性，变量不用定义直接用）。

gl.enableVertexAttribArray告诉WebGL，这个属性（attribute）的值是用array给出的。

```
    shaderProgram.pMatrixUniform =
        gl.getUniformLocation(shaderProgram, "uPMatrix");
    shaderProgram.mvMatrixUniform =
        gl.getUniformLocation(shaderProgram, "uMVMatrix");
}
```

后面还会用到uniform变量，用getUniformLocation()来取得它的位置，也让shaderProgram随身带着，加上去。

```
var mvMatrix = mat4.create();
var pMatrix = mat4.create();
```

新建一个模型视图矩阵mvMatrix，模型矩阵用来控制物体平移旋转缩放，视图矩阵控制观察者位置的相应变化，前面提过仿射变换通过矩阵相乘实现，模型和视图矩阵通常一起用，所以这里定义为模型-视图矩阵。

pMatrix是投影矩阵，控制视野。视野有一定张角，并限定视野最远看到哪里，最近看到哪里，超出范围的数据不显示，只将张角、远近平面围起来的区域的数据进行计算映射。

```
function setMatrixUniforms()
{
    gl.uniformMatrix4fv(shaderProgram.pMatrixUniform, false, pMatrix);
    gl.uniformMatrix4fv(shaderProgram.mvMatrixUniform, false, mvMatrix);
}
```

前面让shaderProgram随身携带shader中uniform变量的位置，这里通过gl.uniformMatrix4fv把JS中我们定义的两个矩阵与GLSL中的对应变量绑定了，这样GPU就能得到我们在CPU中（执行JS代码）为两个矩阵准备的值。

```
function drawScene()
{
    gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    mat4.perspective(pMatrix, 45, gl.viewportWidth / gl.viewportHeight, 0.1, 100.0);
```

万事俱备，可以“画”了。首先设置视口大小，我们就设置成画布大小。

清理buffer。

mat4是外部库gl-matrix提供的矩阵运算类，gl-matrix也在维护更新，写这篇的时候是2.x版本，与1.x版本不

同的是把一些函数的输出矩阵传参的位置，从最后一个提到第一个了，所以可能会与LearningWebGL有所不同。

这里设置视场垂直张角45°，宽高比为画布的宽高比，视野最近限制距离和最远限制距离。把实现这套视野条件的矩阵存放到前面JS代码中定义的pMatrix里。

```
    mat4.identity(mvMatrix);

    mat4.translate(mvMatrix, mvMatrix, [-1.5, 0.0, -7.0]);
```

然后要把三角和矩形移动一下。我们前面两个数组设置每个点的坐标时候，是以原点为重心的，直接画就会让三角和矩形重叠在一起。

我们分别让它们一个左移一点，一个右移一点。

在计算机中坐标的平移、旋转、缩放是通过仿射变换实现的（可以上网查一下仿射变换）。把三维坐标放到一个4*4的矩阵里，当然会多一维出来，在计算中是有用的。根据坐标变换的需要，设计一个或一系列4*4的变换矩阵，然后把他们乘起来，就把坐标变换到了需要的位置，矩阵乘法是满足结合律的，所以一系列变换可以乘好为一个矩阵，再去乘原坐标的矩阵。至于为什么4*4，第四维填什么数字，为什么矩阵乘就能坐标变换，那都是数学上的事了。我们用gl-matrix这个优秀的库，就省去了考虑这些繁琐的东西。

学过线性代数知道，任何矩阵乘以单位矩阵结果还是它自己。我们先用mat4.identity()把mvMatrix初始化为单位矩阵，然后再去变换，才能得到预期的结果（如果第一步已经不是单位矩阵了，那么它相当于已经进行过若干变换）。

mat4.translate就完成了平移操作（当然这里只是生成了进行平移操作的矩阵）。

```
    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
        triangleVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);
```

又见bindBuffer，前面解释过，这时又进入了“对gl.ARRAY_BUFFER操作就是对triangleVertexPositionBuffer操作”的状态。

shaderProgram随身携带的，shader代码中“vertexPositionAttribute”变量的位置这里用上了，triangleVertexPositionBuffer的itemSize也是前面我们让它随身带上的。

这样就告诉GPU：shader的GLSL代码中vertexPositionAttribute变量使用的是gl目前绑定的buffer（也就是triangleVertexPositionBuffer），每itemSize(这里是3）个数组成一个item。剩下的参数以后再说。

```
    setMatrixUniforms();
```

我们设置好了模型-视图矩阵mvMatrix和投影矩阵pMatrix，用前面定义的setMatrixUniforms()把他们映射到shader里的对应两个uniform变量，这样GPU中执行shader时，就可以从GLSL的代码中uniform变量“那个pMatrix”得到这里的“这个pMatrix”，mvMatrix同样。

```
    gl.drawArrays(gl.TRIANGLES, 0, triangleVertexPositionBuffer.numItems);
```

终于是真正的画了，把给出的顶点数组当做三角形来画，从第0个item开始，画numItems（“随身携带”，我保证后面不再重复这个JS特性了。。）个。

```
    mat4.translate(mvMatrix, mvMatrix, [ 3.0, 0.0,  0.0]);
    gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexPositionBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
        squareVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);
    setMatrixUniforms();
    gl.drawArrays(gl.TRIANGLE_STRIP, 0,
        squareVertexPositionBuffer.numItems);
}
```

矩形同理，注意两点，一是这里没对mvMatrix使用identity，而直接变换，那将接着刚才向左平移的-1.5进行变换，这次变换之后实际相对原点是向右平移了1.5。二是这里用的TRIANGLE_STRIP——三角条带，就是画了第一个三角形后，每个新点都和之前两个点组成三角形，这里相当于画两个三角形组成了矩形。

呼~~第一节完成了，我们可以在网页上看到开头的那个效果图——黑色背景中左边三角右边矩形。

本以为能比LearningWebGL写得精简些，没想到也写了这么多。

完整代码方便测试，见附录1。

为减少第一次接触Web的同学的工作量，第一节的完整代码顶部加了前面说的jQuery和gl-matrix的引入，用的bootstrap中文网的CDN，如果这个CDN不失效的话，应该是不加任何额外代码就可以达到本节预期效果的。后面的内容不再在示例代码中添加这些外部JS，大家要自行添加。

添加颜色

本节在上节基础上添加颜色，效果如图2。

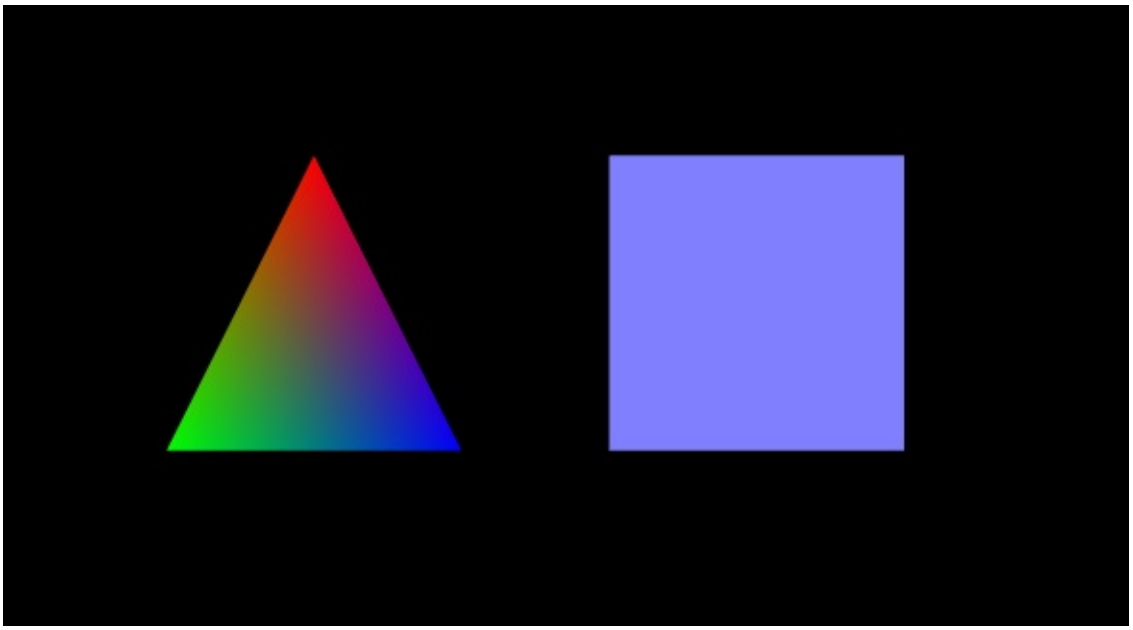


图2

在着色器中添加与颜色有关的代码，并对应修改相关函数与着色器变量的关系。

首先是顶点着色器。

```
<script id = "shader-vs" type = "x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    attribute vec4 aVertexColor;
```

attribute变量对应三维模型里每个顶点的属性，shader是一个顶点一个顶点操作的。这两个属性的值我们在initBuffer里给出。

```
    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;
```

uniform在挨个处理顶点的过程中是不变的，这里就是模型-视图矩阵与投影矩阵在一次流水线显然要中保持恒定。不然这个点左移，那个点右移，就改变形状了，这不是顶点着色器该有的含义。

```
    varying vec4 vColor;
    void main(void)
    {
        gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
        vColor = aVertexColor;
    }
</script>
```

显卡绘制需要考虑每个显示的点的颜色，而模型的顶点与顶点之间还有许多会在屏幕上显示的却并没有属性的点，通过vColor把顶点属性的颜色传入，将获得线性插值的颜色映射。

```
<script id = "shader-fs" type = "x-shader/x-fragment">
    precision mediump float;

    varying vec4 vColor;

    void main(void)
    {
        gl_FragColor = vColor;
    }
</script>
```

片元着色器直接把插值后的颜色给片元，这里就是一个显示的像素。

然后我们在initShaders()中增加aVertexColor的对应内容。

```
    shaderProgram.vertexColorAttribute =
        gl.getAttribLocation(shaderProgram, "aVertexColor");
    gl.enableVertexAttribArray(shaderProgram.vertexColorAttribute);
```

在initBuffers()中添加颜色数据。

```
var triangleVertexColorBuffer;
var squareVertexColorBuffer;
```

先增加全局变量。

```
function initBuffers()
{
    //...
    triangleVertexColorBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexColorBuffer);
    var colors = [
                  1.0, 0.0, 0.0, 1.0,
                  0.0, 1.0, 0.0, 1.0,
                  0.0, 0.0, 1.0, 1.0
                  ];
      gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(colors), gl.STATIC_DRAW);
      triangleVertexColorBuffer.itemSize = 4;
      triangleVertexColorBuffer.numItems = 3;
    //...
    squareVertexColorBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexColorBuffer);
    colors = []
    for (var i=0; i < 4; i++) {
      colors = colors.concat([0.5, 0.5, 1.0, 1.0]);
    }
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(colors), gl.STATIC_DRAW);
    squareVertexColorBuffer.itemSize = 4;
```

```
    squareVertexColorBuffer.numItems = 4;
}
```

写法和triangleVertexPositionBuffer、squareVertexPositionBuffer是一样的，它们都是顶点的属性。

最后在drawScene()中增加对应颜色的代码。

```
function drawScene()
{
    //...
    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexColorBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexColorAttribute,
        triangleVertexColorBuffer.itemSize, gl.FLOAT, false, 0, 0);
    //...
    gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexColorBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexColorAttribute,
        squareVertexColorBuffer.itemSize, gl.FLOAT, false, 0, 0);
    //...
}
```

也和坐标buffer类似。

完整代码见附录1。

一点运动

这一节我们要让前面的三角和矩形动一动。想想我们看电影，无论是胶片的还是数字的，都不是绝对连续的，而是一帧一帧有微小区别的画面连起来，快速变换（大约一秒钟30帧），让人感觉是连续的。

WebGL也是这样，我们每隔很小的一段时间，往canvas上画一遍和前一个画面有微小差别的内容。

效果如图3。

图3

首先修改webGLStart()。

```
function webGLStart()
{
    //...
    //drawScene();
    tick();
}
```

drawScene()不再在这个地方执行，而是调用了一个我们后面要定义的tick()函数，， 下面我们看看tick()做些什么。

```
<script type="text/javascript">
    requestAnimFrame = window.requestAnimationFrame ||
        window.mozRequestAnimationFrame ||
        window.webkitRequestAnimationFrame ||
        window.msRequestAnimationFrame ||
        window.oRequestAnimationFrame ||
        function(callback) { setTimeout(callback, 1000 / 60); };
</script>
```

我们先在主程序外面额外写一段代码如上。由于标准不统一的问题，requestAnimationFrame这个功能在不同浏览器上有不同的名字，为了兼容，我们还用了或运算，用requestAnimFrame来统一代替它们，在下面的tick()中使用它。

```
function tick()
{
    requestAnimFrame(tick);
```

requestAnimFrame(tick)表示定时反复执行tick()。大多我们做这类事的时候会用定时器setInterval，不过现在人们使用浏览器，一般会打开多个标签页，不同的标签页可能还有不同的WebGL内容，使用定时器的话，无论我们在浏览哪里，这个定时器都会一直跑，电脑压力蛮大的。requestAnimationFrame帮我们解决了这个问题，它会以一个合适的频率执行传入的函数，并且只在标签可见的时候运行。

```
    drawScene();
    animate();
}
```

规律地执行tick()，我们便反复执行“画”和“动”了，drawScene()我们已了解，animate()就是计算下一时刻我们希望呈现的帧的样子。

```
var rTri = 0;
var rSquare = 0;
```

定义两个全局变量表示三角和矩形旋转的状态，它们随时间变化。使用全局变量并不好，后面的章节中会介绍如何更优雅地组织这些变量。

```
function drawScene()
{
    //...
    ////mat4.translate(mvMatrix, mvMatrix, [-1.5, 0.0, -7.0]);
    mvPushMatrix();
    mat4.rotate(mvMatrix, mvMatrix, degToRad(rTri), [0, 1, 0]);
    ////gl.drawArrays(gl.TRIANGLES, 0,
    ////    triangleVertexPositionBuffer.numItems);
    mvPopMatrix();
    ////mat4.translate(mvMatrix, mvMatrix, [ 3.0, 0.0,  0.0]);
    mvPushMatrix();
    mat4.rotate(mvMatrix, mvMatrix, degToRad(rSquare), [1, 0, 0]);
    //...
    ////gl.drawArrays(gl.TRIANGLE_STRIP, 0,
    ////    squareVertexPositionBuffer.numItems);
    mvPopMatrix();
}
```

我们在drawScene()中增加了一些代码。

这里消除一个常见误解（我当初就迷惑过），我们计算运动的时候，不是从上一帧状态计算这一帧状态，而是从初始状态通过时间差计算这一帧状态。所以我们在计算这一帧的运动状态前，先把初始状态保存起来，当画好这一帧之后，再把初始状态拿回来。

rotate()的参数中传入了一个向量，是旋转轴的方向。

mvPushMatrix()是我们自己实现的，将当前的模型-视图矩阵（初始状态）压栈暂存。degToRad()也是我们自己写的，简单地把“度数”转化为“弧度”。通过gl-matrix.js提供的方法，就方便地完成了旋转，计算出了图形在这个时间点对应的旋转位置。文中“////”表示插入新代码位置的上下文。

上面使用了rTri和rSquare，它们是需要根据“当前时间”来计算的。

```
var lastTime = 0;
function animate()
{
    var timeNow = new Date().getTime();
    if(lastTime != 0)
    {
        var elapsed = timeNow - lastTime;

        rTri += (90 * elapsed) / 1000.0;
        rSquare += (75 * elapsed) / 1000.0;
    }
    lastTime = timeNow;
}
```

我们不断保存上一帧的时间，“这一帧”调用JS函数取得当前时间，然后计算三角和矩形应该转过的角度，转多快可以自己修改这部分代码来控制。

```
var mvMatrixStack = [];

function mvPushMatrix()
{
    var copy = mat4.clone(mvMatrix);
```

数组的直接赋值是引用，操作新数组会改变原数组（可以用直接赋值看看效果的区别，俩形状转得更欢）。

所以这里用了gl-matrix.js提供的clone 。LearningWebGL中使用的是“mat4.set(mvMatrix, copy)”，可能也是gl-matrix的1.x和2.x版本的区别，我看新版本中没有这个方法了。

```
    mvMatrixStack.push(copy);
}
function mvPopMatrix()
{
    if(mvMatrixStack.length == 0)
    {
        throw "不合法的矩阵出栈操作!";
    }
    mvMatrix = mvMatrixStack.pop();
}
```

JS本身有在数组上的push和pop方法，所以mvMatrixStack数组当栈直接使用了。

```
function degToRad(degrees)
{
```

```
    return degrees * Math.PI / 180;
}
```

0~180对应0~π的转换就很简单了。JS的Math对象有常用的数学常量与方法。

来点真正的3D

这节我们把三角改成四棱椎，矩形改成正方体。

效果如图4。

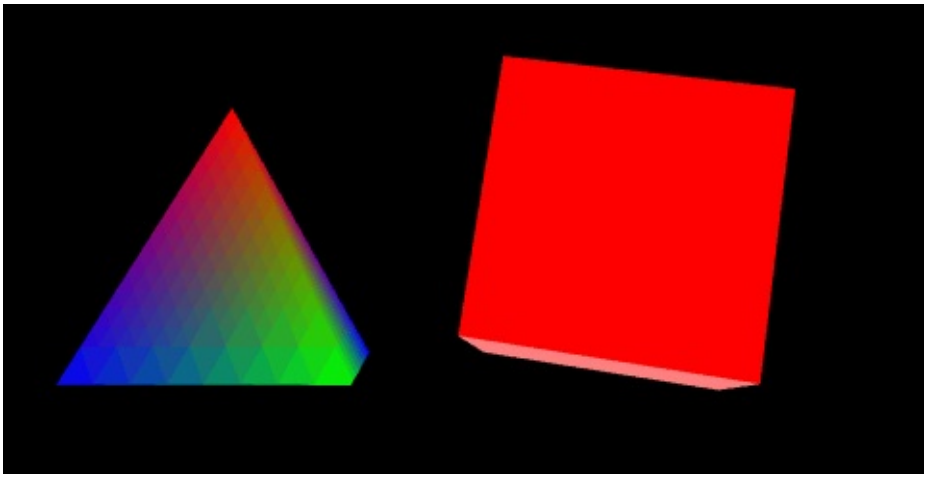


图4

之前几节的图形是一个面，而立体的图形是由多个面组成的，我们一个面一个面地给出顶点坐标，画上去就得到了三维图形。

在绘制正方体的时候，我们增加一个索引。原因是：

1. 相邻的面是有公共点的，如果所有面都各自用坐标表示，就多了许多重复工作，这时我们增加一个索引（index），只需给出所有点的坐标，就可以用索引来重复使用点。
2. 如果我们用“POLYGON”画面，程序不清楚我们每个面有几条边，就要用for循环去一个面一个面画。如果用“TRIANGLE_STRIP”来画，则这个面画完，可能会和另一个面的顶点组成一个我们并不想画的三角形。如果用“TRIANGLES”来画，一个面四个点，等于两个三角形，两个公共点的坐标我们就需要多提供一次。使用索引+“TRIANGLES”，就方便多了。

为了让变量名更有意义一些，我们先把三角和矩形对应的变量名改成四面体和立方体。比如rTri改为rPyramid，triangleVertexPositionBuffer改成pyramidVertexPositionBuffer等等。

四面体用老方法来画，正方体用点索引的方法（具体方法是随意的，看怎么方便了）。

```
var cubeVertexIndexBuffer;
```

增加全局变量，正方体顶点索引的buffer。

```
function initBuffers()
{
    pyramidVertexPositionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, pyramidVertexPositionBuffer);
    var vertices = [
                    // 正面
```

```
                     0.0, 1.0, 0.0,
                    -1.0, -1.0, 1.0,
                     1.0, -1.0, 1.0,
                    // 右侧面
                     0.0, 1.0, 0.0,
                     1.0, -1.0, 1.0,
                     1.0, -1.0, -1.0,
                    // 背面
                     0.0, 1.0, 0.0,
                     1.0, -1.0, -1.0,
                    -1.0, -1.0, -1.0,
                    // 左侧面
                     0.0, 1.0, 0.0,
                    -1.0, -1.0, -1.0,
                    -1.0, -1.0, 1.0
                    ];
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices),
        gl.STATIC_DRAW);
    pyramidVertexPositionBuffer.itemSize = 3;
    pyramidVertexPositionBuffer.numItems = 12;

    pyramidVertexColorBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, pyramidVertexColorBuffer);
    var colors = [
                 // 正面
                 1.0, 0.0, 0.0, 1.0,
                 0.0, 1.0, 0.0, 1.0,
                 0.0, 0.0, 1.0, 1.0,
                 // 右侧面
                 1.0, 0.0, 0.0, 1.0,
                 0.0, 0.0, 1.0, 1.0,
                 0.0, 1.0, 0.0, 1.0,
                 // 背面
                 1.0, 0.0, 0.0, 1.0,
                 0.0, 1.0, 0.0, 1.0,
                 0.0, 0.0, 1.0, 1.0,
                 // 左侧面
                 1.0, 0.0, 0.0, 1.0,
                 0.0, 0.0, 1.0, 1.0,
                 0.0, 1.0, 0.0, 1.0
                ];
     gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(colors),
         gl.STATIC_DRAW);
     pyramidVertexColorBuffer.itemSize = 4;
     pyramidVertexColorBuffer.numItems = 12;
```

对四面体，给出了三维世界中每个面的顶点坐标，相应的numItems也自然和三角的不一样了。

```
    cubeVertexPositionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
    vertices = [
                // 正面
                -1.0, -1.0,  1.0,
                 1.0, -1.0,  1.0,
                 1.0,  1.0,  1.0,
                -1.0,  1.0,  1.0,
```

```
                // 背面
                -1.0, -1.0, -1.0,
                -1.0,  1.0, -1.0,
                 1.0,  1.0, -1.0,
                 1.0, -1.0, -1.0,

                // 顶部
                -1.0,  1.0, -1.0,
                -1.0,  1.0,  1.0,
                 1.0,  1.0,  1.0,
                 1.0,  1.0, -1.0,

                // 底部
                -1.0, -1.0, -1.0,
                 1.0, -1.0, -1.0,
                 1.0, -1.0,  1.0,
                -1.0, -1.0,  1.0,

                // 右侧面
                 1.0, -1.0, -1.0,
                 1.0,  1.0, -1.0,
                 1.0,  1.0,  1.0,
                 1.0, -1.0,  1.0,

                // 左侧面
                -1.0, -1.0, -1.0,
                -1.0, -1.0,  1.0,
                -1.0,  1.0,  1.0,
                -1.0,  1.0, -1.0,
                ];
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices),
        gl.STATIC_DRAW);
    cubeVertexPositionBuffer.itemSize = 3;
    cubeVertexPositionBuffer.numItems = 24;

    cubeVertexColorBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexColorBuffer);
    colors = [
              [1.0, 0.0, 0.0, 1.0],     // 正面
              [1.0, 1.0, 0.0, 1.0],     // 背面
              [0.0, 1.0, 0.0, 1.0],     // 顶部
              [1.0, 0.5, 0.5, 1.0],     // 底部
              [1.0, 0.0, 1.0, 1.0],     // 右侧面
              [0.0, 0.0, 1.0, 1.0]     // 左侧面
            ];
    var unpackedColors = [];
    for (var i in colors)
    {
        var color = colors[i];
        for (var j=0; j < 4; j++)
        {
            unpackedColors = unpackedColors.concat(color);
        }
    }
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(unpackedColors),
        gl.STATIC_DRAW);
    cubeVertexColorBuffer.itemSize = 4;
    cubeVertexColorBuffer.numItems = 24;
```

代码还是把中正方体每个面的顶点分别给出了，以便给不同的面设置不同颜色，因为一个点有三个相邻面，我们无法给一个点设置三种颜色，但是可以在同一个位置放三个不同颜色的点。

颜色按顶点的顺序对应给出，这个for循环代码的逻辑，思考一下应该会明白。

```
    cubeVertexIndexBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);
    var cubeVertexIndices =
        [
          0, 1, 2,      0, 2, 3,    // 正面
          4, 5, 6,      4, 6, 7,    // 背面
          8, 9, 10,     8, 10, 11,  // 顶部
          12, 13, 14,   12, 14, 15, // 底部
          16, 17, 18,   16, 18, 19, // 右侧面
          20, 21, 22,   20, 22, 23  // 左侧面
        ];
    gl.bufferData(gl.ELEMENT_ARRAY_BUFFER,
        new Uint16Array(cubeVertexIndices), gl.STATIC_DRAW);
    cubeVertexIndexBuffer.itemSize = 1;
    cubeVertexIndexBuffer.numItems = 36;
}
```

索引的设置方法与前面的顶点、颜色类似，只是相应改变了一些参数，如gl.ELEMENT_ARRAY_BUFFER、Uint16Array等。

```
function drawScene()
{
    //...
    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);
    setMatrixUniforms();
    gl.drawElements(gl.TRIANGLES,
        cubeVertexIndexBuffer.numItems, gl.UNSIGNED_SHORT, 0);
    ////mvPopMatrix();
}
```

绘制的时候，四面体依然使用gl.drawArrays()直接顶点数组画。对正方体使用索引来绘制，函数相应改为gl.drawElements()，当然执行绘制函数之前绑定的buffer也要对应IndexBuffer。

这样我们就得到了两个立体的旋转着的图形。

引入纹理

三维模型表面有了自己的纹理，桌子才成为桌子，房子才成为房子。

纹理是用体现表面特征的图片贴上去的，就像装修贴墙纸一样。图片可以拉伸扭曲，假设我们用一张图贴一面墙，给出图左上贴墙左上，图右上贴墙右上……四个角都对应上，图就贴上去了。而位置的关系，用坐标来表示。

这一节我们学习如何贴纹理，为了减少些工作量，先把上一节代码所有四面体有关的部分删除，来给正方体贴纹理。

效果如图5。

图5

```
function webGLStart()
{
    //...
    ////initBuffers();
    initTexture();
    ////gl.clearColor(0.0, 0.0, 0.0, 1.0);
    ////gl.enable(gl.DEPTH_TEST);
    //setTimeout("tick()", 100);
    //tick();
}
```

一开始加入纹理的初始化函数。tick()我们知道是执行动画的部分，直接用tick()的话，我看到了一个WARNING：

> [.WebGLRenderingContext]RENDER WARNING: texture bound to texture unit 0 is not renderable. It maybe non-power-of-2 and have incompatible texture filtering or is not 'texture complete'

我猜应该是JS异步执行这个特点造成的，纹理加载需要时间，在纹理还没加载完成的时候就开始绘制了，于是有了这个提示。用JS的setTimeout简单处理了一下，大意就是等“一小会儿”再绘制。不过这个方法还是不稳定，网页响应慢的话还是会WARNING。最后把tick()改到下面读取纹理时加载图片的响应函数里了。

```
var myTexture;
function initTexture()
```

```
{
    myTexture = gl.createTexture();
    myTexture.image = new Image();
    myTexture.image.onload = function()
    {
        handleLoadedTexture(myTexture);
        tick();
    }
    myTexture.image.src = "/Public/image/mytexture.jpg";
}
```

先用createTexture()建立一个纹理对象给myTexture，再让myTexture随身携带它的纹理的图片，是一个JS的image对象。为这个image对象的onload事件指定触发我们处理纹理的函数handleLoadedTexture()。tick()是从webGLStart()挪过来的，保证加载图片后再tick()，避免那个WARNING。

之后设置image的地址，可以是自己服务器的相对地址也可以是网络上图片的地址。指定地址后浏览器就开始在后台异步获取这个图片，获取成功后就触发了onload事件。

```
function handleLoadedTexture(texture)
{
    gl.bindTexture(gl.TEXTURE_2D, texture);
```

gl.bindTexture和gl.bindBuffer相似，绑定texture为“当前纹理”，接下来没有新的绑定的话，对纹理的处理都是对texture的处理。

```
    gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, 1);
```

gl.pixelstorei用于设置像素存储模式，第一个参数表示读入纹理的图片垂直翻转，这个是因为图片和纹理的坐标系统不同，翻过来就对应了，后面写纹理坐标会好写点。后一个参数表示存储器中每个像素行有1个字节对齐，我们先不用管。

```
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA,
        gl.UNSIGNED_BYTE, texture.image);
```

gl.texImage2D，将刚读入的图片传入显卡纹理缓存，参数从前往后为：使用图片类型、细节层次（以后说）、显卡中储存格式（重复两次，以后说）、每个通道的规格（存储R、G、B等的数据类型）、图像对象本身。

```
    gl.texParameteri(gl.TEXTURE_2D,
        gl.TEXTURE_MAG_FILTER, gl.NEAREST);
    gl.texParameteri(gl.TEXTURE_2D,
        gl.TEXTURE_MIN_FILTER, gl.NEAREST);
```

这两句分别表示目标区域比纹理图片大了或小了的时候如何处理，TEXTURE_MAG_FILTER和TEXTURE_MIN_FILTER分别是放大和缩小。NEAREST是使用纹理坐标中最接近的像素颜色作为需要绘制的颜色，速度快，效果看起来可能会有好多“块块”，不平滑。找个尺寸小点的图片当纹理，对比下

gl.LINEAR试试看。

```
    gl.bindTexture(gl.TEXTURE_2D, null);
}
```

最后把“当前纹理”绑定为空。这是不必要的，但是一种好习惯。

接着我们要在initBuffers()中设置纹理坐标。既然有纹理贴图，就不需要之前的颜色了，把设置颜色有关的变量和函数都替换为纹理相关的。

```
var cubeVertexTextureCoordBuffer;
function initBuffers()
{
    //...
    cubeVertexTextureCoordBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexTextureCoordBuffer);
    textureCoords = [
                    // 正面
                    0.0, 0.0,
                    1.0, 0.0,
                    1.0, 1.0,
                    0.0, 1.0,

                    // 背面
                    1.0, 0.0,
                    1.0, 1.0,
                    0.0, 1.0,
                    0.0, 0.0,

                    // 顶部
                    0.0, 1.0,
                    0.0, 0.0,
                    1.0, 0.0,
                    1.0, 1.0,

                    // 底部
                    1.0, 1.0,
                    0.0, 1.0,
                    0.0, 0.0,
                    1.0, 0.0,

                    // 右侧面
                    1.0, 0.0,
                    1.0, 1.0,
                    0.0, 1.0,
                    0.0, 0.0,

                    // 左侧面
                    0.0, 0.0,
                    1.0, 0.0,
                    1.0, 1.0,
                    0.0, 1.0,
                   ];
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(textureCoords), gl.STATIC_DRAW);
    cubeVertexTextureCoordBuffer.itemSize = 2;
```

```
    cubeVertexTextureCoordBuffer.numItems = 24;
    //...
}
```

和之前设置点坐标、点颜色类似，纹理坐标也是点的一个属性（attribute），纹理坐标看做纹理图片上的一个比例，左下角是(0, 0)，右上角是(1, 1)，每个三维点对应纹理图片上按比例换算的一个位置，点与点之间也在这个比例上自动插值，显卡就知道了纹理图片与三维形状每个位置的对应关系。

```
var xRot = 0;
var yRot = 0;
var zRot = 0;
function drawScene()
{
    ////mat4.translate(mvMatrix, mvMatrix, [0.0, 0.0, -5.0]);
    mat4.rotate(mvMatrix, mvMatrix, degToRad(xRot), [1, 0, 0]);
    mat4.rotate(mvMatrix, mvMatrix, degToRad(yRot), [0, 1, 0]);
    mat4.rotate(mvMatrix, mvMatrix, degToRad(zRot), [0, 0, 1]);
    ////gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
    ////gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
    ////    cubeVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexTextureCoordBuffer);
    gl.vertexAttribPointer(shaderProgram.textureCoordAttribute,
        cubeVertexTextureCoordBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.activeTexture(gl.TEXTURE0);
    gl.bindTexture(gl.TEXTURE_2D, myTexture);
    gl.uniform1i(shaderProgram.samplerUniform, 0);
    //...
}
```

改了一下旋转的方式，这个不做重点，应该可以理解。

对于textureCoordAttribute，和点坐标一样把纹理坐标派给shader中的对应变量。

还记得setMatrixUniforms()吗，那里我们用的uniformMatrix4fv()，uniform1i()也类似的，“uniform”后面的“1i”表示参数的类型，告诉显卡我们用的0号纹理，下面修改GLSL代码来看在shader里是怎样的。

```
<script id = "shader-vs" type = "x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    attribute vec2 aTextureCoord;//new

    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;

    varying vec2 vTextureCoord;//new
    void main(void)
    {
        gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
        vTextureCoord = aTextureCoord;//new
    }
</script>
```

顶点着色器中，我们用纹理代替了之前的颜色，和颜色属性（attribute）类似的，传给下一个着色器的过程中纹理坐标会自动插值。

```
<script id = "shader-fs" type = "x-shader/x-fragment">
    precision mediump float;
    varying vec2 vTextureCoord;
    uniform sampler2D uSampler;
    void main(void)
    {
        gl_FragColor =
            texture2D(uSampler, vec2(vTextureCoord.s, vTextureCoord.t));
    }
</script>
```

片元着色器中，得到了插值后的纹理坐标，通过texture2D()来通过纹理获取这个片元的颜色。uSampler是我们前面用uniform1i()给出的纹理编号，这里的s和t是别名，用x和y也是可以的。

于是我们完成了对模型的贴图。

键盘交互与纹理过滤

这一节做一些简单的交互，并认识一下不同的纹理过滤类型。

效果如图6。

图6

用四个方向键控制绕x轴和绕y轴的旋转，Page Up和Page Down键控制z轴方向上的放大与缩小，用F键在三种纹理过滤类型间切换。

```
var crateTextures = Array();
function initTexture()
{
    var crateImage = new Image();
    for(var i = 0; i < 3; i ++)
    {
        var texture = gl.createTexture();
        texture.image = crateImage;
        crateTextures.push(texture);
    }
    crateImage.onload = function()
    {
        handleLoadedTexture(crateTextures);
    }
    crateImage.src = "/Public/image/crate.gif";
}
```

与上一节不同的是，我们创建了三个纹理对象，存到了一个数组里，纹理换成了一个板条箱的图片。这次传给handleLoadedTexture()的是一个数组。

```
function handleLoadedTexture(textures)
{
    gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, 1);

    gl.bindTexture(gl.TEXTURE_2D, textures[0]);
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA,
        gl.UNSIGNED_BYTE, textures[0].image);
    gl.texParameteri(gl.TEXTURE_2D,
        gl.TEXTURE_MAG_FILTER, gl.NEAREST);
```

```
    gl.texParameteri(gl.TEXTURE_2D,
        gl.TEXTURE_MIN_FILTER, gl.NEAREST);

    gl.bindTexture(gl.TEXTURE_2D, textures[1]);
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA,
        gl.UNSIGNED_BYTE, textures[1].image);
    gl.texParameteri(gl.TEXTURE_2D,
        gl.TEXTURE_MAG_FILTER, gl.LINEAR);
    gl.texParameteri(gl.TEXTURE_2D,
        gl.TEXTURE_MIN_FILTER, gl.LINEAR);

    gl.bindTexture(gl.TEXTURE_2D, textures[2]);
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA,
        gl.UNSIGNED_BYTE, textures[2].image);
    gl.texParameteri(gl.TEXTURE_2D,
        gl.TEXTURE_MAG_FILTER, gl.LINEAR);
    gl.texParameteri(gl.TEXTURE_2D,
        gl.TEXTURE_MIN_FILTER, gl.LINEAR_MIPMAP_NEAREST);

    gl.generateMipmap(gl.TEXTURE_2D);

    gl.bindTexture(gl.TEXTURE_2D, null);
}
```

对传进来的纹理对象数组中三个对象使用了三种参数组合。 textures[0]和上一节一样，用的gl.NEAREST。

NEAREST

使用纹理坐标中最接近的一个像素，作为需要绘制的像素的颜色。边界分明，纹理图片太小的话会有马赛克的感觉。

LINEAR

使用纹理坐标中最接近的若干个颜色，通过加权平均得到需要绘制的像素的颜色。

MIPMAP

LINEAR方式在纹理放大的时候比NEAREST效果好一些，但是在缩小的时候并没有明显差别，都会产生锯齿。

可以在我们这一节做出的交互页面上试试看，木板与木板之间有细缝，当缩小到大约1/10的时候，NEAREST和LINEAR模式都会在某些角度看上去一些细缝消失了。这个很好理解，当纹理需要缩小到一个尺寸的时候，需要绘制的模型上的两个点，换算到纹理上的位置，可能会刚好跨过细缝。

在纹理需要缩小比较多的情况下，如果想让LINEAR解决这个问题，就需要取更多的像素算均值，运算量会变大很多。MIPMAP方法是对原纹理生成了一个MIPMAP图，这是个“图像金字塔”，即图像的1/2、1/4、1/8...等多个图，算一下等比数列求和会知道，至多只多占了一倍的空间，这在算法上是常数级的，可以接受。当然，在生成MIPMAP的时候图片已经做抗锯齿处理了，这是一次性的，总比在即时演算的时候反复做处理好得多。于是在纹理需要缩小时，选择最接近的纹理尺寸再做LINEAR处理。需要生成MIPMAP图，于是有了这句gl.generateMipmap(gl.TEXTURE_2D)。

```
var xRot = 0;
```

```
var xSpeed = 0;
var yRot = 0;
var ySpeed = 0;
var z = -5.0;
var filter = 0;
```

定义与旋转、缩放、纹理过滤类型标记有关的全局变量。

```
var currentlyPressedKeys = {};
function handleKeyDown(event)
{
    currentlyPressedKeys[event.keyCode] = true;
    if(String.fromCharCode(event.keyCode) == "F")
    {
        filter += 1;
        if(filter == 3)
        {
            filter = 0;
        }
    }
}
function handleKeyUp(event)
{
    currentlyPressedKeys[event.keyCode] = false;
}
function handleKeys()
{
    if(currentlyPressedKeys[190])
    {
        //"."/">"句号键
        z -= 0.05;
    }
    if(currentlyPressedKeys[188])
    {
        //","/"<"逗号键
        z += 0.05;
    }
    if(currentlyPressedKeys[65])
    {
        //A
        ySpeed -= 1;
    }
    if(currentlyPressedKeys[68])
    {
        //D
        ySpeed += 1;
    }
    if(currentlyPressedKeys[87])
    {
        //W
        xSpeed -= 1;
    }
    if(currentlyPressedKeys[83])
    {
        //S
        xSpeed += 1;
    }
```

```
}
function tick()
{
    //...
    drawScene();
}
```

上面是对键盘事件的处理逻辑。键盘上每个按键有自己的编码，网上可以查到keycode表。用一个数组currentlyPressedKeys标记每个按键的按下状态，它就像个字典一样。按下事件标记，松开事件取消标记。handleKeys()放在tick()中就会在每个周期运行一次去查currentlyPressedKeys这个“字典”完成需要的操作。

“F”键控制纹理过滤类型的选择，我们想让它按一下变一次，就在keydown里执行。改变速度和缩放我们想按着不松手就持续变，就放到tick周期里执行。并且处理方向键的时候都用了if而没有else或return，这样就允许了它们并行执行。想想在打赛车游戏的时候，转弯就必须松油门还不能同时刹车甩尾的话，感觉一定很糟糕。

```
function webGLStart()
{
    //...
    $(document).keydown(handleKeyDown);
    $(document).keyup(handleKeyUp);
}
```

handleKeys()是tick()控制的，但handleKeyDown()和handleKeyUp()却是我们“一厢情愿”想让事件触发的，这就需要把他们绑定到JS的事件去，jQuery可以像上面这样写，keydown和keyup是jQuery的真实事件，我们让发生这些事件的时候执行想要的函数。

```
function drawScene()
{
    //...
    mat4.translate(mvMatrix, mvMatrix, [0.0, 0.0, z]);
    mat4.rotate(mvMatrix, mvMatrix, degToRad(xRot), [1, 0, 0]);
    mat4.rotate(mvMatrix, mvMatrix, degToRad(yRot), [0, 1, 0]);
    //...
    gl.bindTexture(gl.TEXTURE_2D, crateTextures[filter]);
    //...
}
```

drawScene这一步我们已经轻车熟路了，按照预想的方式放大缩小、运动，三个纹理对象设置为选中的那一个。

方向光与环境光

没有大自然的“上帝算法”，计算机中光照的效果也是要计算的。对于目前的着色器，光照计算也分“逐顶点”和“逐片元”（或者说“逐像素”）。逐顶点计算光照和前面计算颜色、纹理相似，相邻顶点之间的像素通过插值计算得到，相当于把顶点之间理解成“平”的，这对于不平的面效果就会差些，于是会有逐片元的计算方法。

这一节我们对那个板条箱做逐顶点的光照，板条箱本来就是平的，逐顶点效果就挺好。

我们先了解计算机图形中的两种光：

1. 方向光（directional light）：从一个方向照过来的绝对平行的光。
2. 环境光（ambient light）：类似于真实世界中周围环境比如墙壁、空气灰尘散射之后，均匀地从各个方向照过来的光。

物体表面对光的两种反射：

1. 漫反射（Diffuse）：无论什么角度照射，都会朝所有方向反射；无论什么角度观看，亮度只取决于入射角（光线与入射表面法线的夹角）——入射角越大看上去越暗。
2. 镜面反射（Specular）：看到的物体亮度取决于视线是否沿着反射光线。通过调整镜面反射的程度模拟诸如木头、玻璃、金属等不同类型的表面。

Phong光照模型把这些类型综合在一起，所有的光具有两个特性：

1. 漫反射得到的RGB增益
2. 镜面反射得到的RGB增益

所有的材料具有四个特性：

1. 环境光反射的RGB增益
2. 漫反射得到的RGB增益
3. 镜面反射得到的RGB增益
4. 决定镜面反射细节的物体的光泽

场景中每个点的颜色，取决于自身颜色、灯光颜色、灯光效果。

根据Phong光照模型，要确完全确定场景中光照性质，需要每个光的两个属性和物体表面每个点的四个属性。

如图7是维基百科上一个颜色相加得到整体颜色效果的例子。

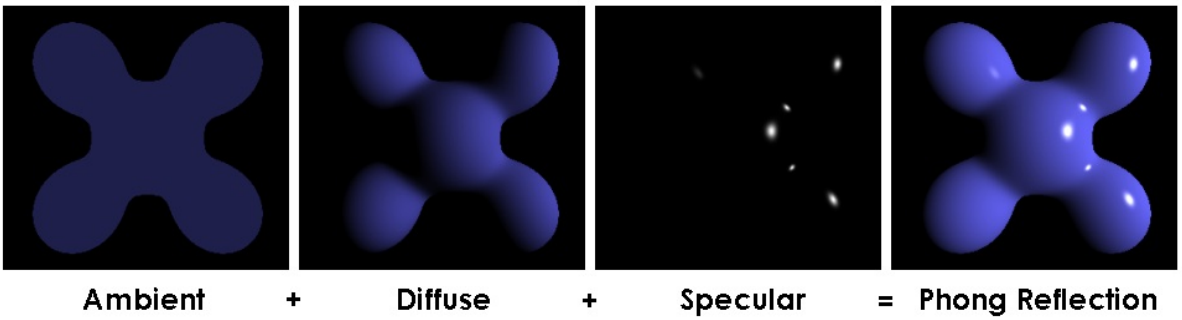


图7

我们将在shader里完成这些颜色相加的工作——计算每个点对环境光、漫反射、镜面反射的RGB的贡献，把这些RGB增益加起来。

要计算反射的效果，需要知道光线与模型表面的夹角，模型表面的方向是由法线向量来表达的。我们逐顶点计算光照，就要给出每个顶点的法线向量。

于是，我们要做的是：

- 为每个顶点保存一个法向量。
- 设定一个方向光的方向向量。
- 在顶点着色器中，对每个顶点计算表面方向与光的夹角并算出一个合适的RGB增益，再加上环境光的RGB增益。

本节效果如图8。

图8

```
<input type="checkbox" id="lighting" checked /> 使用光照
<br/>
(逗号/句号 控制 靠近/远离，WSAD键控制四个方向旋转速度)
<br/>
<h4>方向光:</h4>
<table>
    <tr>
        <td><b>方向:</b></td>
        <td>X: <input type="text" id="lightDirectionX" value="-0.25" /></td>
        <td>Y: <input type="text" id="lightDirectionY" value="-0.25" /></td>
        <td>Z: <input type="text" id="lightDirectionZ" value="-1.0" /></td>
    </tr>
    <tr>
```

```
        <td><b>颜色:</b></td>
        <td>R: <input type="text" id="directionalR" value="0.8" /></td>
        <td>G: <input type="text" id="directionalG" value="0.8" /></td>
        <td>B: <input type="text" id="directionalB" value="0.8" /></td>
    </tr>
</table>
<h4>环境光:</h4>
<table style="border: 0; padding: 10px;">
    <tr>
        <td><b>颜色:</b></td>
        <td>R: <input type="text" id="ambientR" value="0.2" /></td>
        <td>G: <input type="text" id="ambientG" value="0.2" /></td>
        <td>B: <input type="text" id="ambientB" value="0.2" /></td>
    </tr>
</table>
```

在网页上加入光照开关、设置方向光方向与RGB增益、设置环境光RGB增益的控件。

```
<script id = "shader-vs" type = "x-shader/x-vertex">
    //...
    attribute vec3 aVertexNormal;
    uniform mat3 uNMatrix;

    uniform bool uUseLighting;
    varying vec3 vLightWeighting;

    uniform vec3 uAmbientColor;
    uniform vec3 uLightingDirection;
    uniform vec3 uDirectionalColor;

    void main(void)
    {
        //...
        if(!uUseLighting)
        {
            vLightWeighting = vec3(1.0, 1.0, 1.0);
        }
        else
        {
            vec3 transformedNormal = uNMatrix * aVertexNormal;
            float directionalLightWeighting =
                max(dot(transformedNormal, uLightingDirection), 0.0);
            vLightWeighting =
                uAmbientColor + uDirectionalColor * directionalLightWeighting;
        }
    }
</script>
```

新的顶点着色器。

aVertexNormal是顶点的法线向量。

uNMatrix（Mat3）是向量的变换矩阵，向量不能直接使用顶点变换矩阵uMVMatrix（Mat4）。想一下如果向量是(1,0,0)，而顶点平移(-1,0,0)，那向量就变成了(0,0,0)，这显然是不合理的。向量方向要随顶点旋转，但不可因平移而改变，于是需要一个独立的，又跟随uMVMatrix改变而改变的矩阵用来对法线向量做变

化，取模型-视图矩阵的左上3*3部分的逆矩阵的转置作为uNMatrix。至于为什么这样，这又是数学问题了。

uUseLighting标记是否使用光照，vLightWeighting是光照效果的和，uAmbientColor是环境光RGB增益，uLightingDirection是方向光方向，uDirectionalColor是方向光RGB增益。

dot()是点积，两向量对应位置的元素的乘积的和，或者两向量模的积乘以它们夹角的余弦值。根据光照模型，方向光照射在一点的反射强度由光的方向向量与该位置法线向量夹角的余弦决定，而当两向量都为单位向量时，余弦与点积相等。

总的光照RGB增益就是两种光增益的和。

```
<script id = "shader-fs" type = "x-shader/x-fragment">
    //...
    varying vec3 vLightWeighting;

    void main(void)
    {
        //...
        gl_FragColor =
            vec4(textureColor.rgb * vLightWeighting, textureColor.a);
    }
</script>
```

新的片元着色器，使用顶点着色器计算好的、插值后的RGB增益调整纹理颜色。这里增益是个三维量，所以把textureColor拆开计算，再还原为四维量。

```
var cubeVertexNormalBuffer;
function initBuffers()
{
    //...
    cubeVertexNormalBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexNormalBuffer);
    var vertexNormals = [
                        // 正面
                          0.0,  0.0,  1.0,
                          0.0,  0.0,  1.0,
                          0.0,  0.0,  1.0,
                          0.0,  0.0,  1.0,

                        // 背面
                          0.0,  0.0, -1.0,
                          0.0,  0.0, -1.0,
                          0.0,  0.0, -1.0,
                          0.0,  0.0, -1.0,

                        // 顶部
                          0.0,  1.0,  0.0,
                          0.0,  1.0,  0.0,
                          0.0,  1.0,  0.0,
                          0.0,  1.0,  0.0,

                        // 底部
                          0.0, -1.0,  0.0,
```

```
                          0.0, -1.0,  0.0,
                          0.0, -1.0,  0.0,
                          0.0, -1.0,  0.0,

                        // 右侧面
                          1.0,  0.0,  0.0,
                          1.0,  0.0,  0.0,
                          1.0,  0.0,  0.0,
                          1.0,  0.0,  0.0,

                        // 左侧面
                         -1.0,  0.0,  0.0,
                         -1.0,  0.0,  0.0,
                         -1.0,  0.0,  0.0,
                         -1.0,  0.0,  0.0,
                       ];
    gl.bufferData(gl.ARRAY_BUFFER,
        new Float32Array(vertexNormals), gl.STATIC_DRAW);
    cubeVertexNormalBuffer.itemSize = 3;
    cubeVertexNormalBuffer.numItems = 24;
    //...
}
```

很熟悉了，每个顶点再加一个属性——用来计算光的反射效果的法线向量。

```
function drawScene()
{
    //...
    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexNormalBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexNormalAttribute,
        cubeVertexNormalBuffer.itemSize, gl.FLOAT, false, 0, 0);
```

把顶点法线向量发给shader里对应的变量。

```
    //..
    gl.activeTexture(gl.TEXTURE0);
    gl.bindTexture(gl.TEXTURE_2D, crateTexture);
    gl.uniform1i(shaderProgram.samplerUniform, 0);
```

纹理直接用MIPMAP那个方案，不再给三个切换。

```


    var lighting = $("#lighting").is(":checked");
    gl.uniform1i(shaderProgram.useLightingUniform, lighting);
```

从开关控件获取用户的设置——是否使用光照，并把这个bool变量传给shader里的useLightingUniform。

```
    if(lighting)
    {
        gl.uniform3f(
            shaderProgram.ambientColorUniform,
```

```
            parseFloat($("#ambientR").val()),
            parseFloat($("#ambientG").val()),
            parseFloat($("#ambientB").val())
            );
        var lightingDirection = [
            parseFloat($("#lightDirectionX").val()),
            parseFloat($("#lightDirectionY").val()),
            parseFloat($("#lightDirectionZ").val())
            ];
        var adjustedLD = vec3.create();
        vec3.normalize(adjustedLD, lightingDirection);
        vec3.scale(adjustedLD, adjustedLD, -1);
        gl.uniform3fv(shaderProgram.lightingDirectionUniform, adjustedLD);
        gl.uniform3f(
            shaderProgram.directionalColorUniform,
            parseFloat($("#directionalR").val()),
            parseFloat($("#directionalG").val()),
            parseFloat($("#directionalB").val())
            );
    }
    //..
}
```

把两种光的相关参数从网页上用户在控件的输入传给shader。特别的对于光的方向向量，由于要用点积计算与法线向量的余弦，需要归一化处理（normalize）为单位向量，另外光是“向里射”，而法线是“向外”的，要让点积得正数（算锐角），就再把光的方向反向（scale里乘-1）。

```
function setMatrixUniforms()
{
    //...
    var normalMatrix = mat3.create();

    mat3.fromMat4(normalMatrix, mvMatrix);
    mat3.invert(normalMatrix, normalMatrix);
    mat3.transpose(normalMatrix, normalMatrix);

    gl.uniformMatrix3fv(shaderProgram.nMatrixUniform, false, normalMatrix);
}
```

如前面所说的，对法线向量的变换矩阵uNMatrix，要取模型-视图矩阵的左上3*3部分的逆矩阵的转置。

于是我们实现了基本的方向光与环境光。

深度缓冲，透明度与混合

深度缓冲

回顾一下渲染管线：

1. 每个顶点被顶点着色器处理一次，进行相应的变换；
2. 顶点之间进行线性插值，得到许多片元递给片元着色器；
3. 每个片元被片元着色器处理一次，算出它的颜色；
4. 把结果送给帧缓存。

如果模型中有前后多个物体，同一个像素就会传入不同的颜色让片元着色器处理多次，我们肯定只绘制最前面物体的颜色，遮挡后面的物体，着色器如何知道哪个前哪个后，这就需要深度缓冲。深度缓冲记录着每个像素点的深度，在绘制每个像素之前，如果启用了深度缓冲，系统会把它的深度值和已经存储在缓冲里的这个像素的深度值进行比较，而具体保留哪个值，由设置决定。

混合

混合常用来绘制透明或半透明物体。混合是把对应同一个片元的不同物体的颜色，通过需要的方式结合在一起计算，得到一个新的颜色送入帧缓存。具体用什么方式计算，也由设置决定。

这一节，我们分别看使用深度缓冲的效果，与使用混合来实现半透明的效果，如图9。

图9

```
<input type="checkbox" id="blending" checked /> 使用混合
<br/>
不透明度（Alpha）级别 <input type="text" id="alpha" value="0.5" /><br/>
```

首先加上这一节用的混合开关与不透明度设置的控件。

```
<script id = "shader-fs" type = "x-shader/x-fragment">
    //...
    uniform float uAlpha;
    void main(void)
    {
```

```
        //...
        gl_FragColor =
            vec4(textureColor.rgb * vLightWeighting, textureColor.a * uAlpha);
    }
</script>
```

然后修改片元着色器，加入不透明度的设置。

在webGLStart()中，我们一直有这样一行代码：

```
    gl.enable(gl.DEPTH_TEST);
```

相当于告诉WebGL，注意深度缓冲。深度处理已经有一个默认的策略，即常规理解的近处挡住远处的物体。我们也可以明确指定这个策略：

```
    gl.depthFunc(gl.LESS);
```

当然也可以进行其它的设定，这里就不涉及了。

```
function drawScene()
{
    //...
    var blending = $("#blending").is(":checked");
    if(blending)
    {
        gl.blendFunc(gl.SRC_ALPHA, gl.ONE);
```

gl.blendFunc(gl.SRC_ALPHA, gl.ONE) 第一个参数指定红绿蓝和alpha源混合因子如何计算，第二个参数指定红绿蓝和alpha目标混合因子如何计算。 源混合因子就是片元着色器正在画的这个片元，目标混合因子是已经送入帧缓存的片元。

gl.SRC_ALPHA表示使用源颜色的alpha值作为因子，gl.ONE表示用1.0作为因子。其他参数可上网查阅。

设源颜色为$Color_s = (R_s, G_s, B_s, A_s)$，目标颜色为$Color_d = (R_d, G_d, B_d, A_d)$，则按照我们这里使用的参数，计算结果为： $Color_{result} = Color_s * A_s + Color_d$

这种方式不会是产生透明效果最好的方法，不过在这个例子中效果还不错。

```
        gl.enable(gl.BLEND);
        gl.disable(gl.DEPTH_TEST);
        gl.uniform1f(shaderProgram.alphaUniform, parseFloat($("#alpha").val()));
    }
    else
    {
        gl.disable(gl.BLEND);
        gl.enable(gl.DEPTH_TEST);
        gl.uniform1f(shaderProgram.alphaUniform, 1);
    }
}
```

根据网页控件上的开关，控制BLEND和DEPTH_TEST的开关，读取网页上设置的alpha值。

理清一个概念：透明的视觉效果不能和颜色参数中“不透明度”混淆，并不是说把“前面物体”的Alpha降低，“后面物体”就能被看到，那是“上帝算法”。计算机图形中的透明是通过两个物体颜色的合理混合，得到一个颜色作为结果，产生的半透明效果，如果想理解为“视觉欺骗”也无不可。

好了，这节该有的效果有了，不过不透明度设置并不完美。比如把方向光X、Y设为0，Z设为1，按理说透明玻璃，光会照过来的嘛，但是并没有。 所以大家知道了，任何效果都是要计算的，而不是像我们在真实世界中那么简单的因果关系，更好的透明效果，需要更细致的对光、绘制方式的设计。

优化代码结构，多物体运动

这次画多个相同的运动物体，把这个物体封装成一个类，用一个像闪耀的星星的纹理，来给每个星星加特技，Duang~

效果如图10。

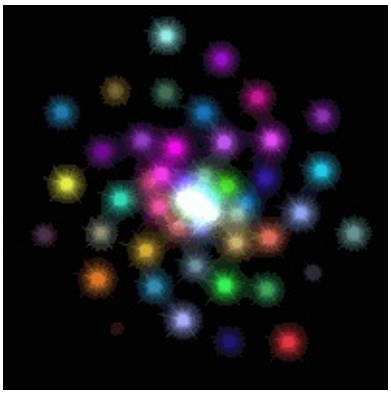


图10

简单来说，JS定义对象和常见的C++、PHP不大一样，不是一个class{}或是struct{}的样子。JS的每个函数就是一个对象，函数对象有一个子对象prototype，用new创建一个类的实例对象的时候，prototype对象的成员都成为实例化对象的成员，具体来看代码。

```
function Star(startingDistance, rotationSpeed)
{
    this.angle = 0;
    this.dist = startingDistance;
    this.rotationSpeed = rotationSpeed;
    //设置一个起始颜色
    this.randomiseColors();
}
```

我们定义Star这么个类，同时这就是它的构造函数，有三个成员变量，还执行了一个自己的方法。

```
Star.prototype.randomiseColors = function()
{
    //给星星一个随机的颜色
    this.r = Math.random();
    this.g = Math.random();
    this.b = Math.random();
    //如果打开了发亮开关，则多绘制一次，用另一个随机的颜色作为亮光
    this.twinkleR = Math.random();
    this.twinkleG = Math.random();
    this.twinkleB = Math.random();
};
```

构造函数中调用的方法，给自己设置一下颜色。

```
var effectiveFPMS = 60 / 1000;
Star.prototype.animate = function(elapsedTime)
{
    this.angle += this.rotationSpeed * effectiveFPMS * elapsedTime;
    //逐步减小与中心的距离
    this.dist -= 0.01 * effectiveFPMS * elapsedTime;
    if(this.dist < 0.0)
    {
        this.dist += 5.0;
        this.randomiseColors();
    }
};
```

不同物体相互独立，它们的运动也应该是相互独立的，这个函数让它们“自己动”。这个代码表示了一个螺旋运动，当“星星”到了中心，就把它挪回最外面继续往里螺旋运动。

把速度理解为相邻两帧画面的运动距离，effectiveFPMS就理解为60帧每秒的情况下，每毫秒多少帧，进而计算每个tick()周期一次跑了多远。

旋转的速度、平移的速度合理设计，才能看到想要的效果，比如如果把旋转速度都设成一样的，那估计就看到的是一根闪亮的棒子一圈圈地转了。

```
Star.prototype.draw = function(tilt, spin, twinkle)
{
    mvPushMatrix();
    //移动到位
    mat4.rotate(mvMatrix, mvMatrix, degToRad(this.angle), [0.0, 1.0, 0.0]);
    mat4.translate(mvMatrix, mvMatrix, [this.dist, 0.0, 0.0]);
    //旋转
    mat4.rotate(mvMatrix, mvMatrix, degToRad(-this.angle), [0.0, 1.0, 0.0]);
    mat4.rotate(mvMatrix, mvMatrix, degToRad(-tilt), [1.0, 0.0, 0.0]);
    if(twinkle)
    {
        //画一个不旋转的star
        gl.uniform3f(shaderProgram.colorUniform,
            this.twinkleR, this.twinkleG, this.twinkleB);
        drawStar();
    }
    //所有Star围绕Z旋转
    mat4.rotate(mvMatrix, mvMatrix, degToRad(spin), [0.0, 0.0, 1.0]);
    //画颜色
    gl.uniform3f(shaderProgram.colorUniform, this.r, this.g, this.b);
    drawStar();
    mvPopMatrix();
};
```

接着设置一个“自己控制如何画自己”的函数，先把变换矩阵压栈，这是前面章节用过的，以保证算完这个星星之后，不影响别的星星。

注意别忘了每个星星不是一个点，而是多个点组成的图形贴上纹理，这里的旋转就是对顶点的操作。我们这次画的是一个个贴上纹理的正方形，是个平面图形，那么星星既要围绕Y轴旋转，又要在用户的控制下围绕X轴旋转，还得始终“面对”观察者而不至于成为一个“扁片”，于是就有了这些旋转。

这里要说一下矩阵变换的问题。

- 矩阵相乘，是左边矩阵的行去乘右边矩阵的列，我们习惯在矩阵中一行一行的思考问题，所以变换矩阵习惯上是乘在坐标矩阵左边的，可参考顶点shader的代码。

而变换矩阵在左边，施加变换在顶点上的顺序，则与我们对变换矩阵做的操作顺序是相反的。

于是我们倒着看上面的几个操作。首先我们的视角是在Z轴上“俯视”XY平面，先按spin让每个星星“自转”，再让星星沿X轴以tilt相反方向转（这样后面我们整体旋转视角的时候，才能让星星纹理“正对”我们），接着让星星沿Y轴反向转（也是为了后面旋转保证纹理“正对”我们），再沿X轴平移（这时候星星中心才离开原点），然后沿着Y轴转到星星最终的位置。在这一系列变换之前（mvPushMatrix()之前），会在drawScene()中设置一个沿X轴按tilt的正向旋转，那么按照逆序，这将是最后对所有星星的统一操作。于是得到了全部“面对”我们的，沿着X轴有一个我们设定的倾角的一群旋转的星星。

```
function drawStar()
{
    gl.activeTexture(gl.TEXTURE0);
    gl.bindTexture(gl.TEXTURE_2D, starTexture);
    gl.uniform1i(shaderProgram.samplerUniform, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, starVertexTextureCoordBuffer);
    gl.vertexAttribPointer(shaderProgram.textureCoordAttribute,
        starVertexTextureCoordBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, starVertexPositionBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
        starVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);

    setMatrixUniforms();
    gl.drawArrays(gl.TRIANGLE_STRIP, 0, starVertexPositionBuffer.numItems);
}
```

和之前一样的画法，每次画一个星星。

```
function webGLStart()
{
    //...
    initWorldObjects();
}
var stars = [];
function initWorldObjects()
{
    var numStars = 50;
    for(var i = 0; i < numStars; i ++)
    {
        stars.push(new Star((i / numStars) * 5.0, i / numStars));
    }
}
```

前面定义了星星的类，这里实例化50个对象。

```
function drawScene()
{
    gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    mat4.perspective(pMatrix, 45,
        gl.viewportWidth / gl.viewportHeight, 0.1, 100.0);

    gl.blendFunc(gl.SRC_ALPHA, gl.ONE);
    gl.enable(gl.BLEND);

    mat4.identity(mvMatrix);

    mat4.translate(mvMatrix, mvMatrix, [0.0, 0.0, zoom]);

    mat4.rotate(mvMatrix, mvMatrix, degToRad(tilt), [1, 0, 0]);

    var twinkle = $("#twinkle").is(":checked");
    for(var i in stars)
    {
        stars[i].draw(tilt, spin, twinkle);
        spin += 0.1;
    }
}
```

把分别画每个星星的代码，放到了一个循环里。

除了把图形封装起来之外，没有什么太多不一样的，相信学到这一节，对大多代码的功能已经很熟悉了，其他地方的修改就不再作说明，可以参考附录的完整代码。

加载场景，基本相机操作

从文件加载模型数据，并控制相机实现在模型中行走。

效果如图11。

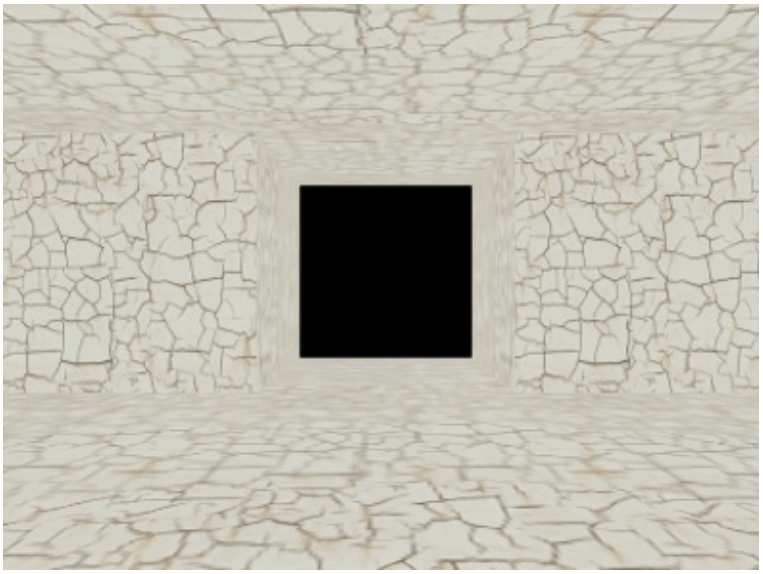


图11

```
<div id="loadingtext">正在加载世界……</div>
```

加载文件可能需要些时间，放一句提示能适当提高用户体验，在这个基础上也可以根据自己的喜好加一些css，再在加载完成之后用JS代码去掉这个提示。

```
function webGLStart()
{
    //...
    loadWorld();
    gl.enable(gl.DEPTH_TEST);
}
function loadWorld()
{
    $.getJSON(
        "/Public/json/world.json",
        function(data)
        {
            handleLoadedWorld(data);
        }
    );
}
```

DEPTH_TEST这次是需要的。用jQuery的ajax来拿存好的数据world.json，由于json是JS常用数据传输格式，这里的getJSON是$.ajax方法的特例。关于ajax、get、post这里就不多解释了。

```
var worldVertexPositionBuffer = null;
var worldVertexTextureCoordBuffer = null;
function handleLoadedWorld(data)
{
    var vertexCount = 0;
    var vertexPositions = [];
    var vertexTextureCoords = [];

    for(var i = 0; typeof(data[i]) != "undefined"; i ++)
    {
        //环境中一个部分的顶点坐标
        vertexPositions =
            vertexPositions.concat(data[i].vertexPositions);
        //然后是纹理坐标
        vertexTextureCoords =
            vertexTextureCoords.concat(data[i].vertexTextureCoords);
        vertexCount ++;
    }
    worldVertexPositionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, worldVertexPositionBuffer);
    gl.bufferData(gl.ARRAY_BUFFER,
        new Float32Array(vertexPositions), gl.STATIC_DRAW);
    worldVertexPositionBuffer.itemSize = 3;
    worldVertexPositionBuffer.numItems = data.vertexCount;

    worldVertexTextureCoordBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, worldVertexTextureCoordBuffer);
    gl.bufferData(gl.ARRAY_BUFFER,
        new Float32Array(vertexTextureCoords), gl.STATIC_DRAW);
    worldVertexTextureCoordBuffer.itemSize = 2;
    worldVertexTextureCoordBuffer.numItems = data.vertexCount;

    $("#loadingtext").text("");
}
```

从事先准备的json文件拿到顶点与纹理的坐标数据，把数据放到对应数组里之后，就是我们熟悉的事情了。这里的world.json只是自己编写的数据，是标准的json格式，不过并不是什么规范的3D数据。

```
function drawScene()
{
    //...
    mat4.rotate(mvMatrix, mvMatrix, degToRad(-pitch), [1, 0, 0]);
    mat4.rotate(mvMatrix, mvMatrix, degToRad(-yaw), [0, 1, 0]);
    mat4.translate(mvMatrix, mvMatrix, [-xPos, -yPos, -zPos]);
}
```

设置相机，让视角在场景中移动。WebGL并不支持直接的相机操作，不过模拟一个不难。相机需要怎么动，让整个场景相反地动。

```
var pitch = 0;
var pitchRate = 0;

var yaw = 0;
var yawRate = 0;
```

```
var xPos = 0;
var yPos = 0.4;
var zPos = 0;

var speed = 0;
function handleKeys()
{
    if(currentlyPressedKeys[188])
    {
        //","/"<"逗号键
        pitchRate = -0.1;
    }
    else if(currentlyPressedKeys[190])
    {
        //"."/">"句号键
        pitchRate = 0.1;
    }
    else
    {
        pitchRate = 0;
    }
    if (currentlyPressedKeys[65])
    {
        //A
        yawRate = 0.1;
    }
    else if (currentlyPressedKeys[68])
    {
        //D
        yawRate = -0.1;
    }
    else
    {
        yawRate = 0;
    }
    if(currentlyPressedKeys[87])
    {
        //W
        speed = 0.003;
    }
    else if(currentlyPressedKeys[83])
    {
        //S
        speed = -0.003;
    }
    else
    {
        speed = 0;
    }
}
var lastTime = 0;
var joggingAngle = 0;
function animate()
{
    var timeNow = new Date().getTime();
    if(lastTime != 0)
    {
        var elapsed = timeNow - lastTime;
        if(speed != 0)
```

```
        {
            xPos -= Math.sin(degToRad(yaw)) * speed * elapsed;
            zPos -= Math.cos(degToRad(yaw)) * speed * elapsed;
            joggingAngle += elapsed * 0.6;
            yPos = Math.sin(degToRad(joggingAngle)) / 20 + 0.4;
        }
        yaw += yawRate * elapsed;
        pitch += pitchRate * elapsed;
    }
    lastTime = timeNow;
}
```

和之前处理移动类似，把XZ理解为运动的平面，Y理解为高度，joggingAngle是个有意思的小把戏，模仿走得过程的“颠簸”，玩FPS游戏应该都能感觉到。

本节主要介绍了一种数据获取的方式和模拟相机效果的实现技巧，具体的代码都是已经熟悉的内容了。

球、旋转矩阵、鼠标事件

这一节我们做一个球体，贴上月球的纹理，加入鼠标的交互。

效果如图12。

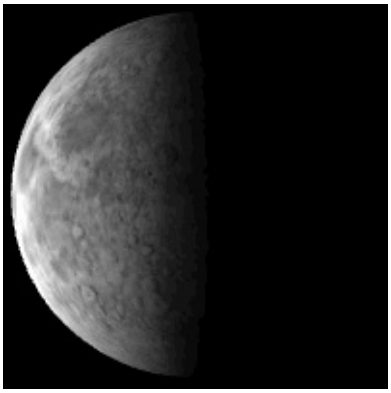


图12

光照方面使用第七节的代码，不再多说。

```
function webGLStart()
{
    //...
    canvas.mousedown(handleMouseDown);
    $(document).mouseup(handleMouseUp);
    $(document).mousemove(handleMouseMove);
}
`
```

类似键盘事件，让鼠标事件触发我们的函数。这里鼠标按下事件绑定的canvas，表示在canvas外面按下鼠标是没有效果的，而鼠标移动事件依然是$(document)，在canvas内按下鼠标之后移出canvas依然可以保持交互效果。可以把canvas和$(document)相互修改一下试试效果。

```
var moonVertexPositionBuffer;
var moonVertexNormalBuffer;
var moonVertexTextureCoordBuffer;
var moonVertexIndexBuffer;
function initBuffers()
{
    var latitudeBands = 30;
    var longitudeBands = 30;
    var radius = 2;
    var vertexPositionData = [];
    var normalData = [];
    var textureCoordData = [];
    for(var latNumber = 0; latNumber <= latitudeBands; latNumber ++)
    {
        var theta = latNumber * Math.PI / latitudeBands;
        var sinTheta = Math.sin(theta);
        var cosTheta = Math.cos(theta);
```

```
        for(var longNumber = 0; longNumber <= longitudeBands; longNumber ++)
        {
            var phi = longNumber * 2 * Math.PI / longitudeBands;
            var sinPhi = Math.sin(phi);
            var cosPhi = Math.cos(phi);
            var x = cosPhi * sinTheta;
            var y = cosTheta;
            var z = sinPhi * sinTheta;
            var u = 1 - (longNumber / longitudeBands);
            var v = 1 - (latNumber / latitudeBands);

            normalData.push(x);
            normalData.push(y);
            normalData.push(z);
            textureCoordData.push(u);
            textureCoordData.push(v);
            vertexPositionData.push(radius * x);
            vertexPositionData.push(radius * y);
            vertexPositionData.push(radius * z);
        }
    }
```

画一个球体，表面是由多个四边形来近似的，有一些基础几何知识就比较容易理解了。想一想或查一查地球仪，看看经纬度的划分，画出的等间距经线与纬线把地球表面划分成一个个曲面四边形（南北极周围是三角形，为了方便，极点也直接算成多个重合的点，这样在代码处理中就都当四边形了），我们用平面四边形来近似它们。

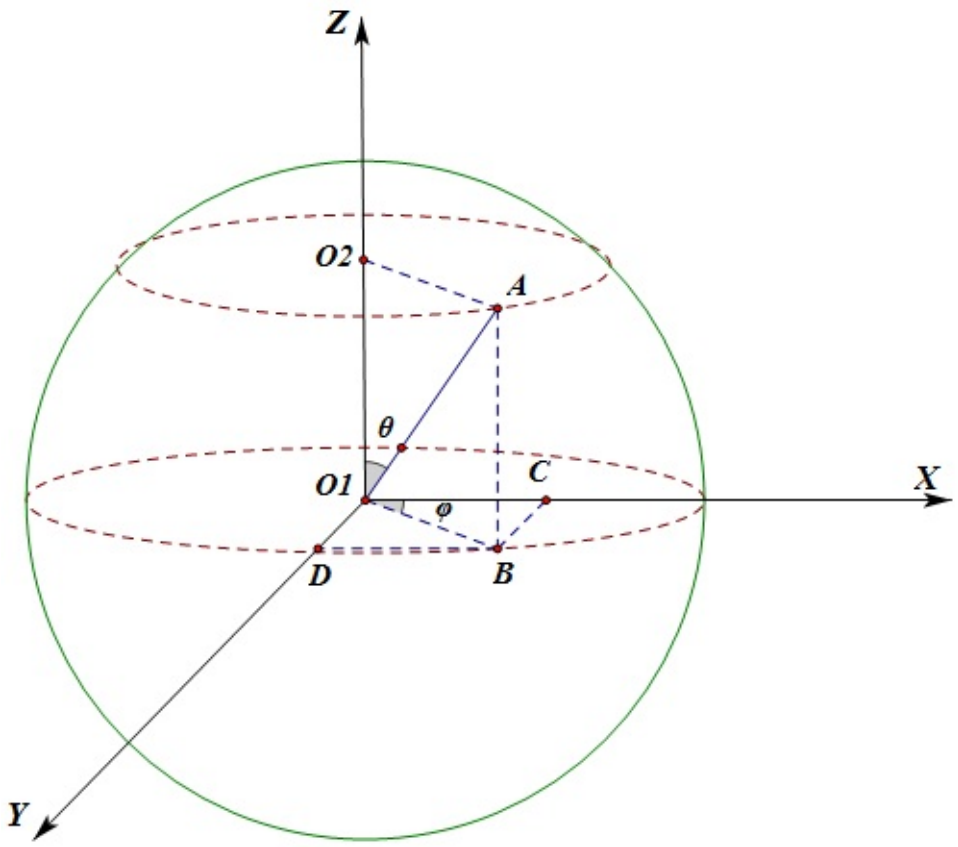


图13，球面坐标

如图13所示，球坐标系统中，球表面任意一点的坐标，由一个与Z轴正方向的夹角θ（0~180°），和原点与

该点在XY平面投影点连线与X轴正方向夹角φ（0~360°）决定。

在图中，|O1A|就是球的半径r，在XY平面的投影|O1B|=|O1A|*sin(θ)。于是：

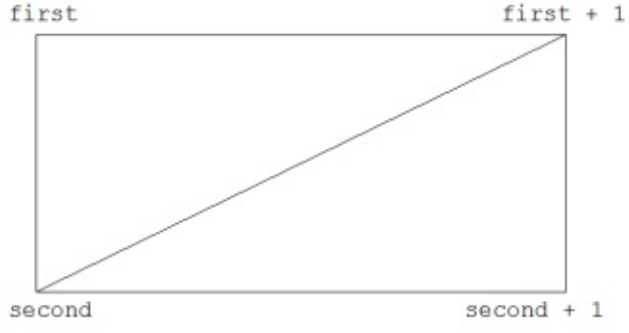
```
X坐标值x为：|O1C|=|O1B|*sin(φ)  --> x=r*sin(θ)*cos(φ);
Y坐标值y为：|O1D|=|O1B|*sin(φ)  --> y=r*sin(θ)*sin(φ);
Z坐标值z为：|O1O2|=|O1A|*cos(θ) --> z=r*cos(θ);
```

均匀枚举θ和φ，然后算出每个点的坐标。我们的视角一直相当于从Z轴俯视，“上”方向相当于是Y轴正方向，所以代码中把Y和Z的计算互换了一下。

球体表面每个点的法向量都是球心与点连线方向，所以为计算光照而需要的法向量刚好在计算坐标的时候就得到了。

```
    var indexData = [];
    for(var latNumber = 0; latNumber < latitudeBands; latNumber ++)
    {
        for(var longNumber = 0; longNumber < longitudeBands; longNumber ++)
        {
            var first = (latNumber * (longitudeBands + 1)) + longNumber;
            var second = first + longitudeBands + 1;
            indexData.push(first);
            indexData.push(second);
            indexData.push(first + 1);
            indexData.push(second);
            indexData.push(second + 1);
            indexData.push(first + 1);
        }
    }
    //...createBuffer(), bindBuffer(), bufferData(), itemSize, numItems等
}
```

得到每个点坐标之后，还得合理组织这些四边形，在计算坐标时候我们是按顺序“一圈一圈”像削平果一样算下来的，于是再一圈一圈把四边形像过去的方法一样用两个三角形表示，把顶点序号组织成一个个三角形，如图14：



> 图14，球面分割的小四边形的序号组织

```
var mouseDown = false;
var lastMouseX = null;
var lastMouseY = null;
var moonRotationMatrix = mat4.create();
mat4.identity(moonRotationMatrix);
function handleMouseDown(event)
{
    mouseDown = true;
    lastMouseX = event.clientX;
    lastMouseY = event.clientY;
}
function handleMouseUp(event)
{
    mouseDown = false;
}
function handleMouseMove(event)
{
    if(!mouseDown)
    {
        return;
    }
    var newX = event.clientX;
    var newY = event.clientY;
    var newRotationMatrix = mat4.create();
    mat4.identity(newRotationMatrix);

    var deltaX = newX - lastMouseX;
    mat4.rotate(newRotationMatrix, newRotationMatrix,
        degToRad(deltaX / 10), [0, 1, 0]);
    var deltaY = newY - lastMouseY;
    mat4.rotate(newRotationMatrix, newRotationMatrix,
        degToRad(deltaY / 10), [1, 0, 0]);

    mat4.multiply(moonRotationMatrix, newRotationMatrix, moonRotationMatrix);

    lastMouseX = newX;
    lastMouseY = newY;
}
```

用一个moonRotationMatrix记录累积旋转的状态，每次新的旋转乘在moonRotationMatrix上，在让mvMatrix去乘以它。这里旋转矩阵用了乘，则要注意左乘右乘了，可以试试调换相乘两个矩阵的顺序看会发生什么。再说明一下gl-matrix的2.x版本，第一个参数都是输出。

这里我有过一个疑惑，在鼠标斜着移动时候，先绕X和先绕Y真的都没关系么？于是用两个矩阵分别计算两个旋转顺序的结果并比较，是不同的。但是为什么实际体验中似乎并没有什么问题（想象一下，如果鼠标从中心出发往左下斜着走很远，那先绕X走很远和先绕Y走，应该最后一个落在Y轴上，一个落在X轴上，而不是预期的落在左下方，可是实际操作却并不如此）。导致这个效果是因为这个计算并不是“一下子”完成的，handleMouseMove()在tick周期中执行，大约每秒60次，鼠标移动的这个计算过程被细分了，产生了“积分”的效果，于是我们往左下移动鼠标的过程，近似了绕着[左上/右下]方向轴旋转的操作。

点光源

效果如图15。

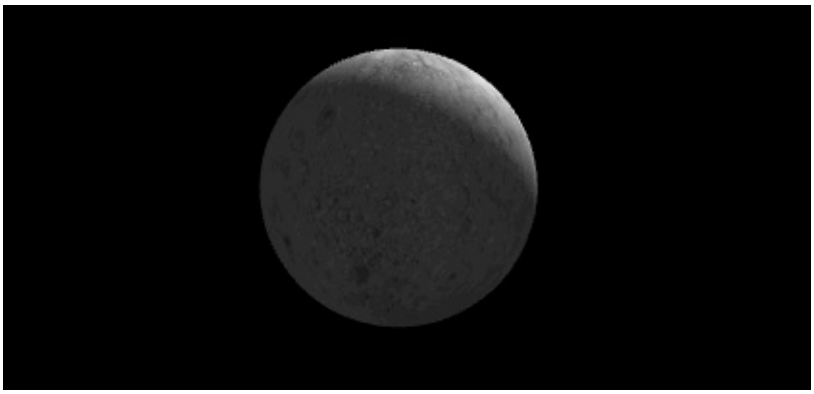


图15

```
var textureFlag = [0, 0];
var startTick = false;
var waitTick;
function webGLStart()
{
    //...
    waitTexture = setInterval("tick()", 100);
}
function tick()
{
    if(textureFlag[0] == 1 && textureFlag[1] == 1)
    {
        startTick = true;
        textureFlag[0] = 2;
        textureFlag[1] = 2;
        clearInterval(waitTexture);
    }
    if(startTick)
    {
        requestAnimFrame(tick);
        drawScene();
        animate();
    }
}
```

这次让板条箱和月亮围着一个中心点旋转，需要加载两个纹理，加载速度慢的话那个关于texture的WARNING就又回来了。为了确保纹理加载完之后再tick()，增加了三个标记和一个定时器。代码逻辑还好理解，用setInterval轮询纹理是否加载完毕而不用while，是因为while执行太快了，会卡的，100ms询问一次就可以了。textureFlag在分别在两个纹理加载完成的时候变为true。

```
var lastTime = 0;
function animate()
{
    var timeNow = new Date().getTime();
```

```
    if(lastTime != 0)
    {
        var elapsed = timeNow - lastTime;

         moonAngle += 0.05 * elapsed;
         cubeAngle += 0.05 * elapsed;
    }
    lastTime = timeNow;
}
```

要让它们旋转，animate自然要放回来的。

```
function drawScene()
{
    //...
    var lighting = $("#lighting").is(":checked");
    gl.uniform1i(shaderProgram.useLightingUniform, lighting);
    if(lighting)
    {
        gl.uniform3f(
            shaderProgram.ambientColorUniform,
            parseFloat($("#ambientR").val()),
            parseFloat($("#ambientG").val()),
            parseFloat($("#ambientB").val())
            );
        gl.uniform3f(
            shaderProgram.pointLightingLocationUniform,
            parseFloat($("#lightPositionX").val()),
            parseFloat($("#lightPositionY").val()),
            parseFloat($("#lightPositionZ").val())
        );

        gl.uniform3f(
            shaderProgram.pointLightingColorUniform,
            parseFloat($("#pointR").val()),
            parseFloat($("#pointG").val()),
            parseFloat($("#pointB").val())
        );

    }
```

点光源的参数还是和之前类似方法用uniform给出。

```


    mat4.identity(mvMatrix);
    mat4.translate(mvMatrix, mvMatrix, [0, 0, -20]);

    mvPushMatrix();

    mat4.rotate(mvMatrix, mvMatrix, degToRad(moonAngle), [0, 1, 0]);
    mat4.translate(mvMatrix, mvMatrix, [5, 0, 0]);

    //...绘制月亮，和之前绘制月亮一样。

    mvPopMatrix();
```

```
    mvPushMatrix();
    mat4.rotate(mvMatrix, mvMatrix, degToRad(cubeAngle), [0, 1, 0]);
    mat4.translate(mvMatrix, mvMatrix, [5, 0, 0]);

    //...绘制板条箱，和前面绘制板条箱一样。
}
```

利用mvPopMatrix()和mvPushMatrix()，就把月亮的绘制和板条箱的绘制独立开来，都从原点平移相同距离，再旋转不同角度。

```
<script id = "shader-vs" type = "x-shader/x-vertex">
    //...
    uniform vec3 uPointLightingLocation;
    uniform vec3 uPointLightingColor;

    void main(void)
    {
        vec4 mvPosition = uMVMatrix * vec4(aVertexPosition, 1.0);
        gl_Position = uPMatrix * mvPosition;
        vTextureCoord = aTextureCoord;
        if (!uUseLighting)
        {
            vLightWeighting = vec3(1.0, 1.0, 1.0);
        }
        else
        {
            vec3 lightDirection =
                normalize(uPointLightingLocation - mvPosition.xyz);
            vec3 transformedNormal = uNMatrix * aVertexNormal;
            float directionalLightWeighting =
                max(dot(transformedNormal, lightDirection), 0.0);
            vLightWeighting =
                uAmbientColor + uPointLightingColor * directionalLightWeighting;
        }
    }
</script>
```

点光源照来的光，与相同方向照来的平行光是没有两样的，利用当前计算的点移动到的位置mvPosition，和点光源的位置uPointLightingLocation，就算出了点光源照到该点的方向，而这个方向向量就可以和前面章节计算方向光效果一样方法使用了。于是在计算gl_Position的时候，与之前不同地分了两步，先计算了mvPosition。

好了，我们有了点光源。

逐片元光照，多着色方案切换

效果如图16。

图16

经过前面的章节，应该已经理解了顶点着色器、片元着色器这两部分别针对的是什么样的数据。之前我们计算光照，是在顶点着色器中，计算出每个顶点的光照增益，这个增益值在传给片元着色器的过程中，会在顶点与顶点之间差值。想一下图17的情况。

效果如图17。

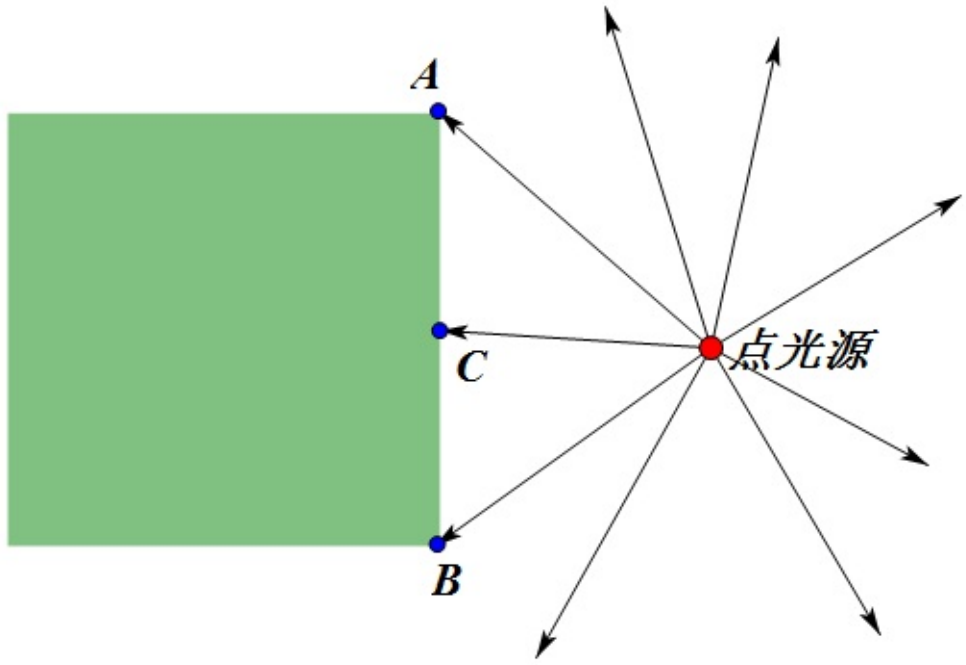


图17

光源照在A点和B点，计算出的增益效果是很接近的，且都比较暗。按照常理，光源照在C位置，由于几乎是垂直照射，应该比A和B的位置亮很多。而在逐顶点计算光照的过程中，由于C不是顶点，它的光照增益是由A到B差值得到的，那么将和A、B的光照增益接近。

在逐片元计算光照过程中，我们在顶点着色器中只计算A和B在矩阵变换后的法向量，这个法向量也将在传

给片元着色器的过程中差值，图17的情况C和A、B的法向量相同。在片元着色器再计算光照增益，C的法向量方向与点光源照到C的方向几乎相同，就能计算出更为真实的光照效果。对于弧面，通过法向量的差值，就可以得到平滑的法向量变化，进而得到平滑的光照效果。维基百科这张图形象说明了逐顶点与逐片元在处理弧面光照效果的差别。

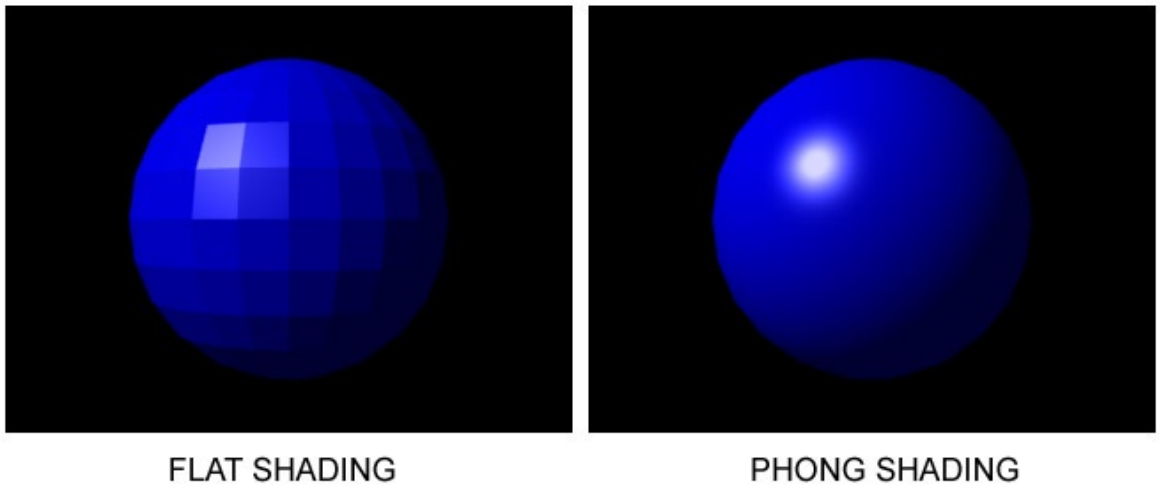


图18

本例的演示页，打开/关闭逐片元计算光照，可以看到板条箱中间的光照效果不同。关闭纹理，就更容易看出月球表面光照效果在使用逐片元计算光照时更为平滑。

```
<script id = "per-vertex-lighting-vs" type = "x-shader/x-vertex">
//...
</script>
<script id = "per-vertex-lighting-fs" type = "x-shader/x-fragment">
//...
</script>
```

把之前逐顶点计算光照的shader控件的id改一下，作为区分。

```
<script id = "per-fragment-lighting-vs" type = "x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    attribute vec3 aVertexNormal;
    attribute vec2 aTextureCoord;

    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;
    uniform mat3 uNMatrix;

    varying vec2 vTextureCoord;
    varying vec3 vTransformedNormal;
    varying vec4 vPosition;


    void main(void)
    {
        vPosition = uMVMatrix * vec4(aVertexPosition, 1.0);
        gl_Position = uPMatrix * vPosition;
        vTextureCoord = aTextureCoord;
```

```
        vTransformedNormal = uNMatrix * aVertexNormal;
    }
</script>
<script id = "per-fragment-lighting-fs" type = "x-shader/x-fragment">
    precision mediump float;

    varying vec2 vTextureCoord;
    varying vec3 vTransformedNormal;
    varying vec4 vPosition;

    uniform bool uUseLighting;
    uniform bool uUseTextures;

    uniform vec3 uAmbientColor;

    uniform vec3 uPointLightingLocation;
    uniform vec3 uPointLightingColor;

    uniform sampler2D uSampler;


    void main(void)
    {
        vec3 lightWeighting;
        if (!uUseLighting)
        {
            lightWeighting = vec3(1.0, 1.0, 1.0);
        }
        else
        {
            vec3 lightDirection =
                normalize(uPointLightingLocation - vPosition.xyz);

            float directionalLightWeighting =
                max(dot(normalize(vTransformedNormal), lightDirection), 0.0);
            lightWeighting =
                uAmbientColor + uPointLightingColor * directionalLightWeighting;
        }

        vec4 fragmentColor;
        if (uUseTextures)
        {
            fragmentColor =
                texture2D(uSampler, vec2(vTextureCoord.s, vTextureCoord.t));
        }
        else
        {
            fragmentColor = vec4(1.0, 1.0, 1.0, 1.0);
        }
        gl_FragColor = vec4(fragmentColor.rgb * lightWeighting, fragmentColor.a);
    }
</script>
```

加入新的逐片元计算光照的shader。可以看到，顶点着色器的任务变少了，只需要计算好法向量。片元着色器得到差值后的法向量后，用类似逐顶点计算光照的顶点着色器的方法来计算光照增益。要注意的是差值后的法向量不一定是单位向量，我们需要把它normalize()一下。

```
var currentProgram;
var perVertexProgram;
var perFragmentProgram;
function initShaders()
{
    perVertexProgram =
        createProgram("per-vertex-lighting-vs", "per-vertex-lighting-fs");
    perFragmentProgram =
        createProgram("per-fragment-lighting-vs", "per-fragment-lighting-fs");
}
```

把之前的initShaders写成了createProgram这个用来调用的函数，分别加载两种着色器方案。

```
function drawScene()
{
    //...
    var perFragmentLighting = $("#per-fragment").is(":checked");
    if(perFragmentLighting)
    {
        currentProgram = perFragmentProgram;
    }
    else
    {
        currentProgram = perVertexProgram;
    }
    gl.useProgram(currentProgram);


}
```

不同的shader都加载编译链接好之后，就可以这样在绘制的时候切换了。

镜面高光

效果如图19。

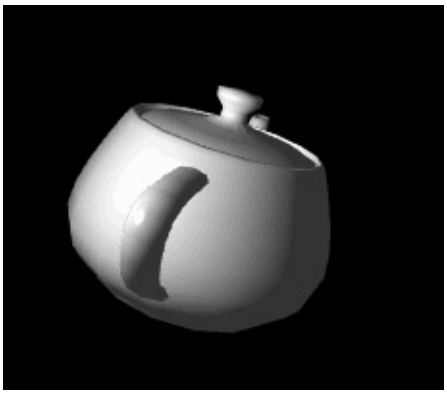


图19

计算高光我们用逐片元计算，顶点shader不用修改，在片元shader中计算镜面高光。

```
<script id = "per-fragment-lighting-fs" type = "x-shader/x-fragment">
    //...
    uniform float uMaterialShininess;
    uniform bool uShowSpecularHighlights;

    uniform vec3 uPointLightingLocation;
    uniform vec3 uPointLightingSpecularColor;
    uniform vec3 uPointLightingDiffuseColor;

    uniform sampler2D uSampler;

    void main(void)
    {
        vec3 lightWeighting;
        if (!uUseLighting)
        {
            lightWeighting = vec3(1.0, 1.0, 1.0);
        }
        else
        {
            vec3 lightDirection =
                normalize(uPointLightingLocation - vPosition.xyz);
            vec3 normal = normalize(vTransformedNormal);

            float specularLightWeighting = 0.0;

            if(uShowSpecularHighlights)
            {
```

这里我们保留了在片元shader前已插值的法向量并归一化，作为中间变量。

高光开关如果是关闭的话增益为0。

计算镜面反射亮度的方程为：

> $(R_m * V)^{\alpha}$

R_m是光线在该点镜面反射后的归一化方向向量，V是眼（相机）到该点的归一化方向向量，α描述光洁度，α越大则越光亮，相当于材质的反光特征。

```
                vec3 eyeDirection = normalize(-vPosition.xyz);
```

我们的相机一直在原点，所以这个向量很好算。

```
                vec3 reflectionDirection = reflect(-lightDirection, normal);
```

GLSL提供了计算反射向量的函数reflect，lightDirection加负号是因为为了方便计算漫反射颜色，我们前面计算的lightDirection变量方向和光线是相反的。

```
                specularLightWeighting =
                    pow(max(dot(reflectionDirection, eyeDirection), 0.0),
                    uMaterialShininess);
            }
```

用公式计算镜面反射增益。

```
            float diffuseLightWeighting = max(dot(normal, lightDirection), 0.0);
            lightWeighting = uAmbientColor +
                uPointLightingSpecularColor * specularLightWeighting +
                uPointLightingDiffuseColor * diffuseLightWeighting;
        }

        ////vec4 fragmentColor;
        //...
        gl_FragColor = vec4(fragmentColor.rgb * lightWeighting, fragmentColor.a);
    }
</script>
```

最后计算总颜色增益还是把它们加起来。

前面我们加载过json数据，这一节一样方式加载茶壶的数据，具体参考完整代码。

镜面地图

效果如图20。

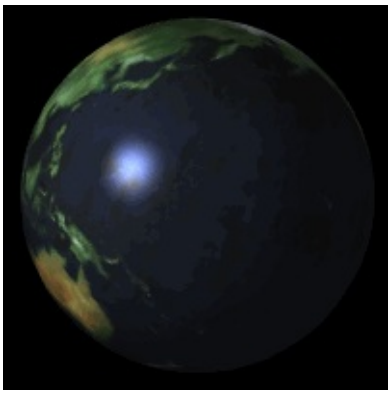


图20

我们要地球仪上镜面反射光在陆地和海洋部分不相同，甚至不同位置都有独特的镜面反射效果设置，于是需要在片元着色时候有海洋还是陆地这样区分位置的标记信息。

海洋和陆地是纹理决定的，我们用另一个和地球纹理对应的低色彩分辨率纹理来做标记就很不错。

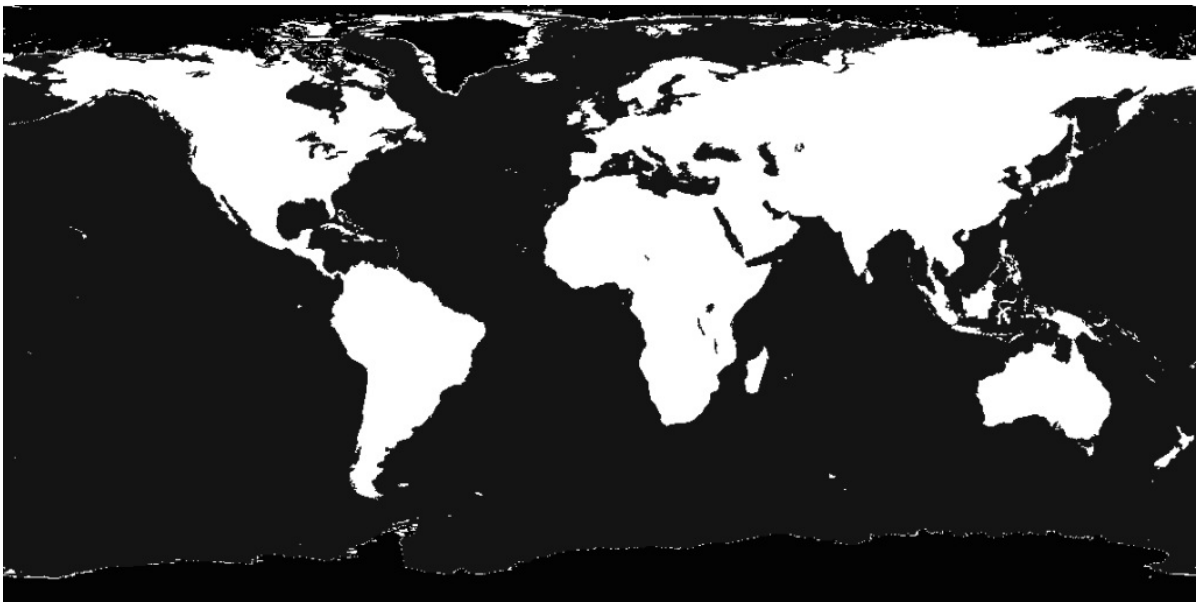


图21

代码是人写的，把它当纹理加载进去传给shader，然后在shader中不把它当纹理用，而是计算高光用，这个很自由。加载两个纹理，原来的纹理做纹理贴图，低色彩分辨率纹理只当高光效果标记。

```
function drawScene()
{
    //...
    gl.activeTexture(gl.TEXTURE0);
    gl.bindTexture(gl.TEXTURE_2D, earthColorMapTexture);
    gl.uniform1i(shaderProgram.specularMapSamplerUniform, 0);
```

```
    gl.activeTexture(gl.TEXTURE1);
    gl.bindTexture(gl.TEXTURE_2D, earthSpecularMapTexture);
    gl.uniform1i(shaderProgram.specularMapSamplerUniform, 1);
}
```

之前我们加载板条箱与月球两个纹理的时候，由于它们是分别绘制的，互不相干，所以只用一个纹理缓存就可以。而这里我们需要在shader中同时使用两个纹理，就用到了两个纹理缓存。

```
<script id = "per-fragment-lighting-fs" type = "x-shader/x-fragment">
    //...
    uniform sampler2D uColorMapSampler;
    uniform sampler2D uSpecularMapSampler;
    void main(void)
    {
        //...
            float shininess = 32.0;
            if (uUseSpecularMap)
            {
                shininess =
                    texture2D(uSpecularMapSampler,
                    vec2(vTextureCoord.s, vTextureCoord.t)).r * 255.0;
            }
        //...
    }
</script>
```

用另一个纹理计算了新的高光参数，就得到了不同位置高光不同的效果。

渲染到纹理

这一节实现在纹理中渲染3D场景，这是个很有用的方法，可以用来实现鼠标选取物体、倒影等许多3D效果。

如图22

图22

为实现这个效果，我们可以动态地把板条箱与月球的运动场景“画”到一个纹理缓存中，再把这个纹理缓存作为纹理“贴”到笔记本电脑屏幕位置。

```
function webGLStart()
{
    //...
    initTextureFramebuffer();
    loadLaptop();
}
```

先读取一个笔记本电脑模型的json文件，这个和之前的方法一样。

这一节的重头戏是帧缓存（FrameBuffer），它是显示画面的一个直接映象，帧缓存的每一存储单元对应一个要绘制的像素，整个帧缓存对应一帧图像。帧缓存可以包含颜色缓存、深度缓存等的组合。WebGL有一个默认的帧缓存，也就是我们之前一直在用的，绘制到整个canvas的帧缓存。

我们也可以自己额外建立帧缓存，用于在显示区域的特定位置另外渲染3D场景。

```
var rttFramebuffer;
var rttTexture;
function initTextureFramebuffer()
{
    rttFramebuffer = gl.createFramebuffer();
    gl.bindFramebuffer(gl.FRAMEBUFFER, rttFramebuffer);
    rttFramebuffer.width = 512;
    rttFramebuffer.height = 512;
```

先定义全局变量，建立帧缓存，设置将要使用的帧缓存对应的场景大小，这个帧缓存输出的图像将作为纹

理，而纹理尺寸需要是 2 的幂，512在这里是个比较合适的尺寸，256太小，1024没有明显改善。

```
    rttTexture = gl.createTexture();
    gl.bindTexture(gl.TEXTURE_2D, rttTexture);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
```

建立一个纹理对象，这里和之前相似。

```
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA,
        rttFramebuffer.width, rttFramebuffer.height,
        0, gl.RGBA, gl.UNSIGNED_BYTE, null);
```

texImage2D和之前不大一样，之前我们使用的纹理都来自图片，而这次是来自自己定义的帧缓存渲染的结果。这里使用了texImage2D的另一个版本，告诉显卡准备好特定大小的缓存。最后一个参数用来提供一个数组指针，把数组内容复制到显卡开辟的缓存区，这里给一个null，表示没有要复制的数组。

```
    var renderbuffer = gl.createRenderbuffer();
    gl.bindRenderbuffer(gl.RENDERBUFFER, renderbuffer);
    gl.renderbufferStorage(gl.RENDERBUFFER, gl.DEPTH_COMPONENT16,
        rttFramebuffer.width, rttFramebuffer.height);
```

建立一个渲染缓存，用来为帧缓存的绘制提供处理数据的存储空间，申请了给定长宽的16位数值空间。

```
    gl.framebufferTexture2D(
        gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0, gl.TEXTURE_2D, rttTexture, 0);
    gl.framebufferRenderbuffer(
        gl.FRAMEBUFFER, gl.DEPTH_ATTACHMENT, gl.RENDERBUFFER, renderbuffer);
```

上面建立帧缓存的时候已经bind它了，还是老意思，它作为“当前”帧缓存，这里就给当前帧缓存分配渲染目标（把颜色涂到哪）——刚刚建立的纹理；再给当前帧缓存分配处理深度信息的存储空间——刚刚建立的渲染缓存。

```
    gl.bindTexture(gl.TEXTURE_2D, null);
    gl.bindRenderbuffer(gl.RENDERBUFFER, null);
    gl.bindFramebuffer(gl.FRAMEBUFFER, null);
}
```

然后清理一下三个“当前”。

```
var laptopAngle = 0;

function drawScene()
{
    gl.bindFramebuffer(gl.FRAMEBUFFER, rttFramebuffer);
    drawSceneOnLaptopScreen();
    gl.bindFramebuffer(gl.FRAMEBUFFER, null);
```

上面说过，有一个默认的帧缓存，我们绘制都直接绘制到canvas上。这里把“当前”帧缓存设置为rttFramebuffer，drawSceneOnLaptopScreen()是个和之前章节drawScene()一样的绘制函数，把板条箱和月球绘制到“当前”帧缓存“，只不过为了方便，代码中把之前提供表单由用户输入的数据设置为固定值了。

绘制完后，板条箱和月球的场景已经在前面与rttFramebuffer绑定的纹理缓存中了。再把“当前”帧缓存设置为null，那么接下来的绘制又会在默认的帧缓存——我们的canvas中。

```
    //...

    gl.uniform1i(shaderProgram.showSpecularHighlightsUniform, true);
    gl.uniform3f(shaderProgram.pointLightingLocationUniform, -1, 2, -1);
    gl.uniform3f(shaderProgram.ambientLightingColorUniform, -1, 2, -1);

    gl.uniform3f(shaderProgram.ambientLightingColorUniform, 0.2, 0.2, 0.2);
    gl.uniform3f(shaderProgram.pointLightingDiffuseColorUniform, 0.8, 0.8, 0.8);
    gl.uniform3f(shaderProgram.pointLightingSpecularColorUniform, 0.8, 0.8, 0.8);

    gl.uniform3f(shaderProgram.materialAmbientColorUniform, 1.0, 1.0, 1.0);
    gl.uniform3f(shaderProgram.materialDiffuseColorUniform, 1.0, 1.0, 1.0);
    gl.uniform3f(shaderProgram.materialSpecularColorUniform, 1.5, 1.5, 1.5);
    gl.uniform1f(shaderProgram.materialShininessUniform, 5);
    gl.uniform3f(shaderProgram.materialEmissiveColorUniform, 0.0, 0.0, 0.0);
    gl.uniform1i(shaderProgram.useTexturesUniform, false);

    if (laptopVertexPositionBuffer)
    {
        //绘制笔记本电脑模型
    }
```

在绘制笔记本模型之前对光线进行了一系列设置，这次考虑了不同材质对光线的反射性质有不同，所以不但设置了环境光、漫反射光、高光，还增加设置了材质本身对这三种光反射类型的特定增益/削弱，在片元shader中材质特征会与光反射特征结合计算光照效果。

```
    gl.uniform3f(shaderProgram.materialAmbientColorUniform, 0.0, 0.0, 0.0);
    gl.uniform3f(shaderProgram.materialDiffuseColorUniform, 0.0, 0.0, 0.0);
    gl.uniform3f(shaderProgram.materialSpecularColorUniform, 0.5, 0.5, 0.5);
    gl.uniform1f(shaderProgram.materialShininessUniform, 20);
    gl.uniform3f(shaderProgram.materialEmissiveColorUniform, 1.5, 1.5, 1.5);
    gl.uniform1i(shaderProgram.useTexturesUniform, true);

    gl.bindBuffer(gl.ARRAY_BUFFER, laptopScreenVertexPositionBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
        laptopScreenVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, laptopScreenVertexNormalBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexNormalAttribute,
        laptopScreenVertexNormalBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, laptopScreenVertexTextureCoordBuffer);
    gl.vertexAttribPointer(shaderProgram.textureCoordAttribute,
        laptopScreenVertexTextureCoordBuffer.itemSize, gl.FLOAT, false, 0, 0);
```

```
    gl.activeTexture(gl.TEXTURE0);
    gl.bindTexture(gl.TEXTURE_2D, rttTexture);
    gl.uniform1i(shaderProgram.samplerUniform, 0);

    setMatrixUniforms();
    gl.drawArrays(gl.TRIANGLE_STRIP, 0,
        laptopScreenVertexPositionBuffer.numItems);

    mvPopMatrix();
}
```

画笔记本电脑模型的屏幕，为了真实感，显示屏更多的光应该是自己发出的，所以引入了一个放射光（Emissive Color）的概念，其实它就相当于环境光，只是不跟环境光混淆，这样如果环境光改变（比如变成红色），也不影响模型中屏幕应该“发出”的颜色。

模型的屏幕就是一个贴了纹理的矩形，纹理是前面帧缓存绘制的板条箱和月球，矩形在initBuffer()中设置了顶点、法向量和纹理。

```
<script id = "per-fragment-lighting-fs" type = "x-shader/x-fragment">
    //...
        if (uUseTextures)
        {
            vec4 textureColor =
                texture2D(uSampler, vec2(vTextureCoord.s, vTextureCoord.t));
            materialAmbientColor = materialAmbientColor * textureColor.rgb;
            materialDiffuseColor = materialDiffuseColor * textureColor.rgb;
            materialEmissiveColor = materialEmissiveColor * textureColor.rgb;
            alpha = textureColor.a;
        }
        gl_FragColor = vec4(
                            materialAmbientColor * ambientLightWeighting +
                            materialDiffuseColor * diffuseLightWeighting +
                            materialSpecularColor * specularLightWeighting +
                            materialEmissiveColor,
                            alpha
                        );
    }
</script>
```

结合材质与光照的片元shader，与之前的原理相同，只是增加了一些计算，注意最后加上的materialEmissiveColor。具体参考完整代码。

附录1：完整代码

虽然第一节的代码拼起来就是完整的，后面在此基础上修改的代码也是完整的，但是零零碎碎还是有一些不便。

以附录的形式给出每一节的完整代码，方便和我一样的初学者对照调试自己的代码。虽然有很多重复，但是方便啊~

三角与矩形

```
<script src="http://cdn.bootcss.com/jquery/2.1.4/jquery.js"></script>
<script src="http://cdn.bootcss.com/gl-matrix/2.2.1/gl-matrix.js"></script>

<div class="page-header"><h3>1、三角与矩形</h3></div>

<canvas id = "test01-canvas" width = "800" height = "600"></canvas>

<script id = "shader-fs" type = "x-shader/x-fragment">
    precision mediump float;
    void main(void)
    {
        gl_FragColor = vec4(1.0, 1.0, 1.0, 1.0);
    }
</script>
<script id = "shader-vs" type = "x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;
    void main(void)
    {
        gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
    }
</script>

<script type="text/javascript">

$(document).ready(function ()
{
    webGLStart();
});

function webGLStart()
{
    var canvas = $("#test01-canvas")[0];
    initGL(canvas);
    initShaders();
    initBuffers();

    gl.clearColor(0.0, 0.0, 0.0, 1.0);
    gl.enable(gl.DEPTH_TEST);

    drawScene();
}

var gl;
function initGL(canvas)
{
    try
    {
        gl = canvas.getContext("webgl") || canvas.getContext("experimental-webgl");
        gl.viewportWidth = canvas.width;
        gl.viewportHeight = canvas.height;
    }catch(e){}
    if(!gl)
```

```
    {
        alert("无法初始化“WebGL”。");
    }
}

function getShader(gl, id)
{
    var shaderScript = $("#" + id);
    if(!shaderScript.length)
    {
        return null;
    }
    var str = shaderScript.text();

    var shader;
    if(shaderScript[0].type == "x-shader/x-fragment")
    {
        shader = gl.createShader(gl.FRAGMENT_SHADER);
    }
    else if(shaderScript[0].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;
}

var shaderProgram;
function initShaders()
{
    var fragmentShader = getShader(gl, "shader-fs");
    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("无法初始化“Shader”。");
    }
    gl.useProgram(shaderProgram);

    shaderProgram.vertexPositionAttribute =
        gl.getAttribLocation(shaderProgram, "aVertexPosition");
    gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);

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

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


var mvMatrix = mat4.create();
var pMatrix = mat4.create();

function setMatrixUniforms()
{
    gl.uniformMatrix4fv(shaderProgram.pMatrixUniform, false, pMatrix);
    gl.uniformMatrix4fv(shaderProgram.mvMatrixUniform, false, mvMatrix);
}


var triangleVertexPositionBuffer;
var squareVertexPositionBuffer;

function initBuffers()
{
    triangleVertexPositionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
    var vertices = [
                     0.0,  1.0,  0.0,
                    -1.0, -1.0,  0.0,
                     1.0, -1.0,  0.0
                     ];
    gl.bufferData(gl.ARRAY_BUFFER,
        new Float32Array(vertices), gl.STATIC_DRAW);
    triangleVertexPositionBuffer.itemSize = 3;
    triangleVertexPositionBuffer.numItems = 3;

    squareVertexPositionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexPositionBuffer);
    vertices = [
                 1.0,  1.0,  0.0,
                -1.0,  1.0,  0.0,
                 1.0, -1.0,  0.0,
                -1.0, -1.0,  0.0
                ];
    gl.bufferData(gl.ARRAY_BUFFER,
        new Float32Array(vertices), gl.STATIC_DRAW);
    squareVertexPositionBuffer.itemSize = 3;
    squareVertexPositionBuffer.numItems = 4;

}
function drawScene()
{
    gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    mat4.perspective(pMatrix, 45,
        gl.viewportWidth / gl.viewportHeight, 0.1, 100.0);

    mat4.identity(mvMatrix);

    mat4.translate(mvMatrix, mvMatrix, [-1.5, 0.0, -7.0]);
    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
```

```
            triangleVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);
        setMatrixUniforms();
        gl.drawArrays(gl.TRIANGLES, 0, triangleVertexPositionBuffer.numItems);

        mat4.translate(mvMatrix, mvMatrix, [ 3.0, 0.0,  0.0]);
        gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexPositionBuffer);
        gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
            squareVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);
        setMatrixUniforms();
        gl.drawArrays(gl.TRIANGLE_STRIP, 0, squareVertexPositionBuffer.numItems);
    }
    </script>
```

添加颜色

```
<div class="page-header"><h3>2、添加颜色</h3></div>

<canvas id = "test02-canvas" width = "800" height = "600"></canvas>

<script id = "shader-vs" type = "x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    attribute vec4 aVertexColor;
    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;
    varying vec4 vColor;
    void main(void)
    {
        gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
        vColor = aVertexColor;
    }
</script>

<script id = "shader-fs" type = "x-shader/x-fragment">
    precision mediump float;

    varying vec4 vColor;

    void main(void)
    {
        gl_FragColor = vColor;
    }
</script>

<script type="text/javascript">

$(document).ready(function ()
{
    webGLStart();
});

function webGLStart()
{
    var canvas = $("#test02-canvas")[0];
    initGL(canvas);
    initShaders();
    initBuffers();

    gl.clearColor(0.0, 0.0, 0.0, 1.0);
    gl.enable(gl.DEPTH_TEST);

    drawScene();
}

var gl;
function initGL(canvas)
{
    try
    {
        gl = canvas.getContext("webgl") || canvas.getContext("experimental-webgl");
```

```
        gl.viewportWidth = canvas.width;
        gl.viewportHeight = canvas.height;
    }catch(e){}
    if(!gl)
    {
        alert("无法初始化“WebGL”。");
    }
}

function getShader(gl, id)
{
    var shaderScript = $("#" + id);
    if(!shaderScript.length)
    {
        return null;
    }
    var str = shaderScript.text();

    var shader;
    if(shaderScript[0].type == "x-shader/x-fragment")
    {
        shader = gl.createShader(gl.FRAGMENT_SHADER);
    }
    else if(shaderScript[0].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;
}

var shaderProgram;
function initShaders()
{
    var fragmentShader = getShader(gl, "shader-fs");
    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("无法初始化“Shader”。");
    }
    gl.useProgram(shaderProgram);

    shaderProgram.vertexPositionAttribute =
        gl.getAttribLocation(shaderProgram, "aVertexPosition");
```

```
    gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);

    shaderProgram.vertexColorAttribute =
        gl.getAttribLocation(shaderProgram, "aVertexColor");
    gl.enableVertexAttribArray(shaderProgram.vertexColorAttribute);

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


var mvMatrix = mat4.create();
var pMatrix = mat4.create();

function setMatrixUniforms()
{
    gl.uniformMatrix4fv(shaderProgram.pMatrixUniform, false, pMatrix);
    gl.uniformMatrix4fv(shaderProgram.mvMatrixUniform, false, mvMatrix);
}


var triangleVertexPositionBuffer;
var triangleVertexColorBuffer;
var squareVertexPositionBuffer;
var squareVertexColorBuffer;

function initBuffers()
{
    triangleVertexPositionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
    var vertices = [
                     0.0,  1.0,  0.0,
                    -1.0, -1.0,  0.0,
                     1.0, -1.0,  0.0
                     ];
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices),
        gl.STATIC_DRAW);
    triangleVertexPositionBuffer.itemSize = 3;
    triangleVertexPositionBuffer.numItems = 3;

    triangleVertexColorBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexColorBuffer);
    var colors = [
                  1.0, 0.0, 0.0, 1.0,
                  0.0, 1.0, 0.0, 1.0,
                  0.0, 0.0, 1.0, 1.0
                  ];

      gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(colors),
          gl.STATIC_DRAW);
      triangleVertexColorBuffer.itemSize = 4;
      triangleVertexColorBuffer.numItems = 3;


    squareVertexPositionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexPositionBuffer);
    vertices = [
```

```
                 1.0,  1.0,  0.0,
                -1.0,  1.0,  0.0,
                 1.0, -1.0,  0.0,
                -1.0, -1.0,  0.0
                ];

    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices),
        gl.STATIC_DRAW);
    squareVertexPositionBuffer.itemSize = 3;
    squareVertexPositionBuffer.numItems = 4;

    squareVertexColorBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexColorBuffer);
    colors = []
    for (var i=0; i < 4; i++) {
      colors = colors.concat([0.5, 0.5, 1.0, 1.0]);
    }

    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(colors),
        gl.STATIC_DRAW);
    squareVertexColorBuffer.itemSize = 4;
    squareVertexColorBuffer.numItems = 4;

}
function drawScene()
{
    gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    mat4.perspective(pMatrix, 45, gl.viewportWidth /
        gl.viewportHeight, 0.1, 100.0);

    mat4.identity(mvMatrix);

    mat4.translate(mvMatrix, mvMatrix, [-1.5, 0.0, -7.0]);
    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
        triangleVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexColorBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexColorAttribute,
        triangleVertexColorBuffer.itemSize, gl.FLOAT, false, 0, 0);

    setMatrixUniforms();
    gl.drawArrays(gl.TRIANGLES, 0,
        triangleVertexPositionBuffer.numItems);

    mat4.translate(mvMatrix, mvMatrix, [ 3.0, 0.0,  0.0]);
    gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexPositionBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
        squareVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexColorBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexColorAttribute,
        squareVertexColorBuffer.itemSize, gl.FLOAT, false, 0, 0);

    setMatrixUniforms();
    gl.drawArrays(gl.TRIANGLE_STRIP, 0,
        squareVertexPositionBuffer.numItems);
```

```
}
</script>
```

一点运动

```
<div class="page-header"><h3>3、一点运动</h3></div>

<canvas id = "test03-canvas" width = "800" height = "600"></canvas>

<script id = "shader-vs" type = "x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    attribute vec4 aVertexColor;
    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;
    varying vec4 vColor;
    void main(void)
    {
        gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
        vColor = aVertexColor;
    }
</script>

<script id = "shader-fs" type = "x-shader/x-fragment">
    precision mediump float;

    varying vec4 vColor;

    void main(void)
    {
        gl_FragColor = vColor;
    }
</script>
<script type="text/javascript">
    requestAnimFrame = window.requestAnimationFrame ||
        window.mozRequestAnimationFrame ||
        window.webkitRequestAnimationFrame ||
        window.msRequestAnimationFrame ||
        window.oRequestAnimationFrame ||
        function(callback) { setTimeout(callback, 1000 / 60); };
</script>
<script type="text/javascript">

$(document).ready(function ()
{
    webGLStart();
});

function webGLStart()
{
    var canvas = $("#test03-canvas")[0];
    initGL(canvas);
    initShaders();
    initBuffers();

    gl.clearColor(0.0, 0.0, 0.0, 1.0);
    gl.enable(gl.DEPTH_TEST);

//     drawScene();
```

```
    tick();
}

function tick()
{
    requestAnimFrame(tick);
    drawScene();
    animate();
}

var gl;
function initGL(canvas)
{
    try
    {
        gl = canvas.getContext("webgl") || canvas.getContext("experimental-webgl");
        gl.viewportWidth = canvas.width;
        gl.viewportHeight = canvas.height;
    }catch(e){}
    if(!gl)
    {
        alert("无法初始化“WebGL”。");
    }
}

function getShader(gl, id)
{
    var shaderScript = $("#" + id);
    if(!shaderScript.length)
    {
        return null;
    }
    var str = shaderScript.text();

    var shader;
    if(shaderScript[0].type == "x-shader/x-fragment")
    {
        shader = gl.createShader(gl.FRAGMENT_SHADER);
    }
    else if(shaderScript[0].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;
}

var shaderProgram;
function initShaders()
```

```
{
    var fragmentShader = getShader(gl, "shader-fs");
    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("无法初始化“Shader”。");
    }
    gl.useProgram(shaderProgram);

    shaderProgram.vertexPositionAttribute =
        gl.getAttribLocation(shaderProgram, "aVertexPosition");
    gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);

    shaderProgram.vertexColorAttribute =
        gl.getAttribLocation(shaderProgram, "aVertexColor");
    gl.enableVertexAttribArray(shaderProgram.vertexColorAttribute);

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


var mvMatrix = mat4.create();

var mvMatrixStack = [];

var pMatrix = mat4.create();

function mvPushMatrix()
{
    var copy = mat4.clone(mvMatrix);
    mvMatrixStack.push(copy);
}
function mvPopMatrix()
{
    if(mvMatrixStack.length == 0)
    {
        throw "不合法的矩阵出栈操作!";
    }
    mvMatrix = mvMatrixStack.pop();
}

function setMatrixUniforms()
{
    gl.uniformMatrix4fv(shaderProgram.pMatrixUniform, false, pMatrix);
    gl.uniformMatrix4fv(shaderProgram.mvMatrixUniform, false, mvMatrix);
}


var triangleVertexPositionBuffer;
var triangleVertexColorBuffer;
var squareVertexPositionBuffer;
```

```
var squareVertexColorBuffer;

function initBuffers()
{
    triangleVertexPositionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
    var vertices = [
                     0.0,  1.0,  0.0,
                    -1.0, -1.0,  0.0,
                     1.0, -1.0,  0.0
                     ];
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices),
        gl.STATIC_DRAW);
    triangleVertexPositionBuffer.itemSize = 3;
    triangleVertexPositionBuffer.numItems = 3;

    triangleVertexColorBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexColorBuffer);
    var colors = [
                  1.0, 0.0, 0.0, 1.0,
                  0.0, 1.0, 0.0, 1.0,
                  0.0, 0.0, 1.0, 1.0
                  ];
      gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(colors),
          gl.STATIC_DRAW);
      triangleVertexColorBuffer.itemSize = 4;
      triangleVertexColorBuffer.numItems = 3;


    squareVertexPositionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexPositionBuffer);
    vertices = [
                 1.0,  1.0,  0.0,
                -1.0,  1.0,  0.0,
                 1.0, -1.0,  0.0,
                -1.0, -1.0,  0.0
                ];

    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices),
        gl.STATIC_DRAW);
    squareVertexPositionBuffer.itemSize = 3;
    squareVertexPositionBuffer.numItems = 4;

    squareVertexColorBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexColorBuffer);
    colors = []
    for (var i=0; i < 4; i++) {
      colors = colors.concat([0.5, 0.5, 1.0, 1.0]);
    }

    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(colors),
        gl.STATIC_DRAW);
    squareVertexColorBuffer.itemSize = 4;
    squareVertexColorBuffer.numItems = 4;

}

var rTri = 0;
var rSquare = 0;
```

```
function drawScene()
{
    gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    mat4.perspective(pMatrix, 45, gl.viewportWidth /
        gl.viewportHeight, 0.1, 100.0);

    mat4.identity(mvMatrix);

    mat4.translate(mvMatrix, mvMatrix, [-1.5, 0.0, -7.0]);

    mvPushMatrix();
    mat4.rotate(mvMatrix, mvMatrix, degToRad(rTri), [0, 1, 0]);

    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
        triangleVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexColorBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexColorAttribute,
        triangleVertexColorBuffer.itemSize, gl.FLOAT, false, 0, 0);

    setMatrixUniforms();
    gl.drawArrays(gl.TRIANGLES, 0,
        triangleVertexPositionBuffer.numItems);

    mvPopMatrix();

    mat4.translate(mvMatrix, mvMatrix, [ 3.0, 0.0,  0.0]);

    mvPushMatrix();
    mat4.rotate(mvMatrix, mvMatrix, degToRad(rSquare), [1, 0, 0]);

    gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexPositionBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
        squareVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexColorBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexColorAttribute,
        squareVertexColorBuffer.itemSize, gl.FLOAT, false, 0, 0);

    setMatrixUniforms();
    gl.drawArrays(gl.TRIANGLE_STRIP, 0,
        squareVertexPositionBuffer.numItems);

    mvPopMatrix();
}
function degToRad(degrees)
{
    return degrees * Math.PI / 180;
}
var lastTime = 0;
function animate()
{
    var timeNow = new Date().getTime();
    if(lastTime != 0)
    {
        var elapsed = timeNow - lastTime;
```

```
        rTri += (90 * elapsed) / 1000.0;
        rSquare += (75 * elapsed) / 1000.0;
    }
    lastTime = timeNow;
}
</script>
```

来点真正的3D

```
<div class="page-header"><h3>4、来点真正的3D</h3></div>

<canvas id = "test04-canvas" width = "800" height = "600"></canvas>

<script id = "shader-vs" type = "x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    attribute vec4 aVertexColor;
    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;
    varying vec4 vColor;
    void main(void)
    {
        gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
        vColor = aVertexColor;
    }
</script>

<script id = "shader-fs" type = "x-shader/x-fragment">
    precision mediump float;

    varying vec4 vColor;

    void main(void)
    {
        gl_FragColor = vColor;
    }
</script>
<script type="text/javascript">
    requestAnimFrame = window.requestAnimationFrame ||
        window.mozRequestAnimationFrame ||
        window.webkitRequestAnimationFrame ||
        window.msRequestAnimationFrame ||
        window.oRequestAnimationFrame ||
        function(callback) { setTimeout(callback, 1000 / 60); };
</script>
<script type="text/javascript">

$(document).ready(function ()
{
    webGLStart();
});

function webGLStart()
{
    var canvas = $("#test04-canvas")[0];
    initGL(canvas);
    initShaders();
    initBuffers();

    gl.clearColor(0.0, 0.0, 0.0, 1.0);
    gl.enable(gl.DEPTH_TEST);

//     drawScene();
```

```
    tick();
}

function tick()
{
    requestAnimFrame(tick);
    drawScene();
    animate();
}

var gl;
function initGL(canvas)
{
    try
    {
        gl = canvas.getContext("webgl") || canvas.getContext("experimental-webgl");
        gl.viewportWidth = canvas.width;
        gl.viewportHeight = canvas.height;
    }catch(e){}
    if(!gl)
    {
        alert("无法初始化“WebGL”。");
    }
}

function getShader(gl, id)
{
    var shaderScript = $("#" + id);
    if(!shaderScript.length)
    {
        return null;
    }
    var str = shaderScript.text();

    var shader;
    if(shaderScript[0].type == "x-shader/x-fragment")
    {
        shader = gl.createShader(gl.FRAGMENT_SHADER);
    }
    else if(shaderScript[0].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;
}

var shaderProgram;
function initShaders()
```

```
{
    var fragmentShader = getShader(gl, "shader-fs");
    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("无法初始化“Shader”。");
    }
    gl.useProgram(shaderProgram);

    shaderProgram.vertexPositionAttribute =
        gl.getAttribLocation(shaderProgram, "aVertexPosition");
    gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);

    shaderProgram.vertexColorAttribute =
        gl.getAttribLocation(shaderProgram, "aVertexColor");
    gl.enableVertexAttribArray(shaderProgram.vertexColorAttribute);

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


var mvMatrix = mat4.create();

var mvMatrixStack = [];

var pMatrix = mat4.create();

function mvPushMatrix()
{
    var copy = mat4.clone(mvMatrix);
    mvMatrixStack.push(copy);
}
function mvPopMatrix()
{
    if(mvMatrixStack.length == 0)
    {
        throw "不合法的矩阵出栈操作!";
    }
    mvMatrix = mvMatrixStack.pop();
}

function setMatrixUniforms()
{
    gl.uniformMatrix4fv(shaderProgram.pMatrixUniform, false, pMatrix);
    gl.uniformMatrix4fv(shaderProgram.mvMatrixUniform, false, mvMatrix);
}


var pyramidVertexPositionBuffer;
var pyramidVertexColorBuffer;
var cubeVertexPositionBuffer;
```

```
var cubeVertexColorBuffer;
var cubeVertexIndexBuffer;

function initBuffers()
{
    pyramidVertexPositionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, pyramidVertexPositionBuffer);
    var vertices = [
                    // 正面
                     0.0, 1.0, 0.0,
                    -1.0, -1.0, 1.0,
                     1.0, -1.0, 1.0,
                    // 右侧面
                     0.0, 1.0, 0.0,
                     1.0, -1.0, 1.0,
                     1.0, -1.0, -1.0,
                    // 背面
                     0.0, 1.0, 0.0,
                     1.0, -1.0, -1.0,
                    -1.0, -1.0, -1.0,
                    // 左侧面
                     0.0, 1.0, 0.0,
                    -1.0, -1.0, -1.0,
                    -1.0, -1.0, 1.0
                    ];
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices),
        gl.STATIC_DRAW);
    pyramidVertexPositionBuffer.itemSize = 3;
    pyramidVertexPositionBuffer.numItems = 12;

    pyramidVertexColorBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, pyramidVertexColorBuffer);
    var colors = [
                 // 正面
                 1.0, 0.0, 0.0, 1.0,
                 0.0, 1.0, 0.0, 1.0,
                 0.0, 0.0, 1.0, 1.0,
                 // 右侧面
                 1.0, 0.0, 0.0, 1.0,
                 0.0, 0.0, 1.0, 1.0,
                 0.0, 1.0, 0.0, 1.0,
                 // 背面
                 1.0, 0.0, 0.0, 1.0,
                 0.0, 1.0, 0.0, 1.0,
                 0.0, 0.0, 1.0, 1.0,
                 // 左侧面
                 1.0, 0.0, 0.0, 1.0,
                 0.0, 0.0, 1.0, 1.0,
                 0.0, 1.0, 0.0, 1.0
                ];
     gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(colors),
         gl.STATIC_DRAW);
     pyramidVertexColorBuffer.itemSize = 4;
     pyramidVertexColorBuffer.numItems = 12;


    cubeVertexPositionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
    vertices = [
```

```
                // 正面
                -1.0, -1.0,  1.0,
                 1.0, -1.0,  1.0,
                 1.0,  1.0,  1.0,
                -1.0,  1.0,  1.0,

                // 背面
                -1.0, -1.0, -1.0,
                -1.0,  1.0, -1.0,
                 1.0,  1.0, -1.0,
                 1.0, -1.0, -1.0,

                // 顶部
                -1.0,  1.0, -1.0,
                -1.0,  1.0,  1.0,
                 1.0,  1.0,  1.0,
                 1.0,  1.0, -1.0,

                // 底部
                -1.0, -1.0, -1.0,
                 1.0, -1.0, -1.0,
                 1.0, -1.0,  1.0,
                -1.0, -1.0,  1.0,

                // 右侧面
                 1.0, -1.0, -1.0,
                 1.0,  1.0, -1.0,
                 1.0,  1.0,  1.0,
                 1.0, -1.0,  1.0,

                // 左侧面
                -1.0, -1.0, -1.0,
                -1.0, -1.0,  1.0,
                -1.0,  1.0,  1.0,
                -1.0,  1.0, -1.0,
                ];
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices),
        gl.STATIC_DRAW);
    cubeVertexPositionBuffer.itemSize = 3;
    cubeVertexPositionBuffer.numItems = 24;

    cubeVertexColorBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexColorBuffer);
    colors = [
              [1.0, 0.0, 0.0, 1.0],     // 正面
              [1.0, 1.0, 0.0, 1.0],     // 背面
              [0.0, 1.0, 0.0, 1.0],     // 顶部
              [1.0, 0.5, 0.5, 1.0],     // 底部
              [1.0, 0.0, 1.0, 1.0],     // 右侧面
              [0.0, 0.0, 1.0, 1.0]     // 左侧面
            ];
    var unpackedColors = [];
    for (var i in colors)
    {
        var color = colors[i];
        for (var j=0; j < 4; j++)
        {
            unpackedColors = unpackedColors.concat(color);
        }
```

```
    }
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(unpackedColors),
        gl.STATIC_DRAW);
    cubeVertexColorBuffer.itemSize = 4;
    cubeVertexColorBuffer.numItems = 24;

    cubeVertexIndexBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);
    var cubeVertexIndices =
        [
          0, 1, 2,      0, 2, 3,    // 正面
          4, 5, 6,      4, 6, 7,    // 背面
          8, 9, 10,     8, 10, 11,  // 顶部
          12, 13, 14,   12, 14, 15, // 底部
          16, 17, 18,   16, 18, 19, // 右侧面
          20, 21, 22,   20, 22, 23  // 左侧面
        ];
    gl.bufferData(gl.ELEMENT_ARRAY_BUFFER,
        new Uint16Array(cubeVertexIndices), gl.STATIC_DRAW);
    cubeVertexIndexBuffer.itemSize = 1;
    cubeVertexIndexBuffer.numItems = 36;
}

var rPyramid = 0;
var rCube = 0;
function drawScene()
{
    gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    mat4.perspective(pMatrix, 45, gl.viewportWidth /
        gl.viewportHeight, 0.1, 100.0);

    mat4.identity(mvMatrix);

    mat4.translate(mvMatrix, mvMatrix, [-1.5, 0.0, -8.0]);

    mvPushMatrix();
    mat4.rotate(mvMatrix, mvMatrix, degToRad(rPyramid), [0, 1, 0]);

    gl.bindBuffer(gl.ARRAY_BUFFER, pyramidVertexPositionBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
        pyramidVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, pyramidVertexColorBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexColorAttribute,
        pyramidVertexColorBuffer.itemSize, gl.FLOAT, false, 0, 0);

    setMatrixUniforms();
    gl.drawArrays(gl.TRIANGLES, 0,
        pyramidVertexPositionBuffer.numItems);

    mvPopMatrix();

    mat4.translate(mvMatrix, mvMatrix, [ 3.0, 0.0, 0.0]);

    mvPushMatrix();
    mat4.rotate(mvMatrix, mvMatrix, degToRad(rCube), [1, 1, 1]);
```

```
    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
        cubeVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexColorBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexColorAttribute,
        cubeVertexColorBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);
    setMatrixUniforms();
    gl.drawElements(gl.TRIANGLES,
        cubeVertexIndexBuffer.numItems, gl.UNSIGNED_SHORT, 0);

    mvPopMatrix();
}
function degToRad(degrees)
{
    return degrees * Math.PI / 180;
}
var lastTime = 0;
function animate()
{
    var timeNow = new Date().getTime();
    if(lastTime != 0)
    {
        var elapsed = timeNow - lastTime;

        rPyramid += (90 * elapsed) / 1000.0;
        rCube += (75 * elapsed) / 1000.0;
    }
    lastTime = timeNow;
}
</script>
```

引入纹理

```
<div class="page-header"><h3>5、引入纹理</h3></div>

<canvas id = "test05-canvas" width = "800" height = "600"></canvas>

<script id = "shader-vs" type = "x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    attribute vec2 aTextureCoord;

    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;

    varying vec2 vTextureCoord;
    void main(void)
    {
        gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
        vTextureCoord = aTextureCoord;
    }
</script>

<script id = "shader-fs" type = "x-shader/x-fragment">
    precision mediump float;

    varying vec2 vTextureCoord;
    uniform sampler2D uSampler;

    void main(void)
    {
        gl_FragColor =
            texture2D(uSampler, vec2(vTextureCoord.s, vTextureCoord.t));
    }
</script>
<script type="text/javascript">
    requestAnimFrame = window.requestAnimationFrame ||
        window.mozRequestAnimationFrame ||
        window.webkitRequestAnimationFrame ||
        window.msRequestAnimationFrame ||
        window.oRequestAnimationFrame ||
        function(callback) { setTimeout(callback, 1000 / 60); };
</script>
<script type="text/javascript">

$(document).ready(function ()
{
    webGLStart();
});

function webGLStart()
{
    var canvas = $("#test05-canvas")[0];
    initGL(canvas);
    initShaders();
    initBuffers();
    initTexture();
```

```
    gl.clearColor(0.0, 0.0, 0.0, 1.0);
    gl.enable(gl.DEPTH_TEST);

//    setTimeout("tick()", 100);
}
var myTexture;
function initTexture()
{
    myTexture = gl.createTexture();
    myTexture.image = new Image();
    myTexture.image.onload = function()
    {
        handleLoadedTexture(myTexture);
        tick();
    }
    myTexture.image.src = "/Public/image/mytexture.jpg";
}
function handleLoadedTexture(texture)
{
    gl.bindTexture(gl.TEXTURE_2D, texture);
    gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, 1);
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA,
        gl.UNSIGNED_BYTE, texture.image);
    gl.texParameteri(gl.TEXTURE_2D,
        gl.TEXTURE_MAG_FILTER, gl.NEAREST);
    gl.texParameteri(gl.TEXTURE_2D,
        gl.TEXTURE_MIN_FILTER, gl.NEAREST);
    gl.bindTexture(gl.TEXTURE_2D, null);
}
function tick()
{
    requestAnimFrame(tick);
    drawScene();
    animate();
}

var gl;
function initGL(canvas)
{
    try
    {
        gl = canvas.getContext("webgl") || canvas.getContext("experimental-webgl");
        gl.viewportWidth = canvas.width;
        gl.viewportHeight = canvas.height;
    }catch(e){}
    if(!gl)
    {
        alert("无法初始化“WebGL”。");
    }
}

function getShader(gl, id)
{
    var shaderScript = $("#" + id);
    if(!shaderScript.length)
    {
        return null;
    }
    var str = shaderScript.text();
```

```
    var shader;
    if(shaderScript[0].type == "x-shader/x-fragment")
    {
        shader = gl.createShader(gl.FRAGMENT_SHADER);
    }
    else if(shaderScript[0].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;
}

var shaderProgram;
function initShaders()
{
    var fragmentShader = getShader(gl, "shader-fs");
    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("无法初始化“Shader”。");
    }
    gl.useProgram(shaderProgram);

    shaderProgram.vertexPositionAttribute =
        gl.getAttribLocation(shaderProgram, "aVertexPosition");
    gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);

    shaderProgram.textureCoordAttribute =
        gl.getAttribLocation(shaderProgram, "aTextureCoord");
    gl.enableVertexAttribArray(shaderProgram.textureCoordAttribute);

    shaderProgram.pMatrixUniform =
        gl.getUniformLocation(shaderProgram, "uPMatrix");
    shaderProgram.mvMatrixUniform =
        gl.getUniformLocation(shaderProgram, "uMVMatrix");
    shaderProgram.samplerUniform =
        gl.getUniformLocation(shaderProgram, "uSampler");
}


var mvMatrix = mat4.create();
```

```
var mvMatrixStack = [];

var pMatrix = mat4.create();

function mvPushMatrix()
{
    var copy = mat4.clone(mvMatrix);
    mvMatrixStack.push(copy);
}
function mvPopMatrix()
{
    if(mvMatrixStack.length == 0)
    {
        throw "不合法的矩阵出栈操作!";
    }
    mvMatrix = mvMatrixStack.pop();
}

function setMatrixUniforms()
{
    gl.uniformMatrix4fv(shaderProgram.pMatrixUniform, false, pMatrix);
    gl.uniformMatrix4fv(shaderProgram.mvMatrixUniform, false, mvMatrix);
}


var cubeVertexPositionBUffer;
var cubeVertexTextureCoordBuffer;
var cubeVertexIndexBuffer;

function initBuffers()
{
    cubeVertexPositionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
    vertices = [
                // 正面
                -1.0, -1.0,  1.0,
                 1.0, -1.0,  1.0,
                 1.0,  1.0,  1.0,
                -1.0,  1.0,  1.0,

                // 背面
                -1.0, -1.0, -1.0,
                -1.0,  1.0, -1.0,
                 1.0,  1.0, -1.0,
                 1.0, -1.0, -1.0,

                // 顶部
                -1.0,  1.0, -1.0,
                -1.0,  1.0,  1.0,
                 1.0,  1.0,  1.0,
                 1.0,  1.0, -1.0,

                // 底部
                -1.0, -1.0, -1.0,
                 1.0, -1.0, -1.0,
                 1.0, -1.0,  1.0,
                -1.0, -1.0,  1.0,
```

```
                // 右侧面
                 1.0, -1.0, -1.0,
                 1.0,  1.0, -1.0,
                 1.0,  1.0,  1.0,
                 1.0, -1.0,  1.0,

                // 左侧面
                -1.0, -1.0, -1.0,
                -1.0, -1.0,  1.0,
                -1.0,  1.0,  1.0,
                -1.0,  1.0, -1.0,
                ];
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices),
        gl.STATIC_DRAW);
    cubeVertexPositionBuffer.itemSize = 3;
    cubeVertexPositionBuffer.numItems = 24;

    cubeVertexTextureCoordBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexTextureCoordBuffer);
    textureCoords = [
                    // 正面
                    0.0, 0.0,
                    1.0, 0.0,
                    1.0, 1.0,
                    0.0, 1.0,

                    // 背面
                    1.0, 0.0,
                    1.0, 1.0,
                    0.0, 1.0,
                    0.0, 0.0,

                    // 顶部
                    0.0, 1.0,
                    0.0, 0.0,
                    1.0, 0.0,
                    1.0, 1.0,

                    // 底部
                    1.0, 1.0,
                    0.0, 1.0,
                    0.0, 0.0,
                    1.0, 0.0,

                    // 右侧面
                    1.0, 0.0,
                    1.0, 1.0,
                    0.0, 1.0,
                    0.0, 0.0,

                    // 左侧面
                    0.0, 0.0,
                    1.0, 0.0,
                    1.0, 1.0,
                    0.0, 1.0,
                   ];
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(textureCoords),
        gl.STATIC_DRAW);
    cubeVertexTextureCoordBuffer.itemSize = 2;
```

```
    cubeVertexTextureCoordBuffer.numItems = 24;

    cubeVertexIndexBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);
    var cubeVertexIndices =
        [
          0, 1, 2,      0, 2, 3,    // 正面
          4, 5, 6,      4, 6, 7,    // 背面
          8, 9, 10,     8, 10, 11,  // 顶部
          12, 13, 14,   12, 14, 15, // 底部
          16, 17, 18,   16, 18, 19, // 右侧面
          20, 21, 22,   20, 22, 23  // 左侧面
        ];
    gl.bufferData(gl.ELEMENT_ARRAY_BUFFER,
        new Uint16Array(cubeVertexIndices), gl.STATIC_DRAW);
    cubeVertexIndexBuffer.itemSize = 1;
    cubeVertexIndexBuffer.numItems = 36;
}

var xRot = 0;
var yRot = 0;
var zRot = 0;
function drawScene()
{
    gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    mat4.perspective(pMatrix, 45, gl.viewportWidth /
        gl.viewportHeight, 0.1, 100.0);

    mat4.identity(mvMatrix);

    mat4.translate(mvMatrix, mvMatrix, [0.0, 0.0, -5.0]);

    mat4.rotate(mvMatrix, mvMatrix, degToRad(xRot), [1, 0, 0]);
    mat4.rotate(mvMatrix, mvMatrix, degToRad(yRot), [0, 1, 0]);
    mat4.rotate(mvMatrix, mvMatrix, degToRad(zRot), [0, 0, 1]);

    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
        cubeVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexTextureCoordBuffer);
    gl.vertexAttribPointer(shaderProgram.textureCoordAttribute,
        cubeVertexTextureCoordBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.activeTexture(gl.TEXTURE0);
    gl.bindTexture(gl.TEXTURE_2D, myTexture);
    gl.uniform1i(shaderProgram.samplerUniform, 0);

    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);
    setMatrixUniforms();
    gl.drawElements(gl.TRIANGLES,
        cubeVertexIndexBuffer.numItems, gl.UNSIGNED_SHORT, 0);
}
function degToRad(degrees)
{
    return degrees * Math.PI / 180;
}
```

```
var lastTime = 0;
function animate()
{
    var timeNow = new Date().getTime();
    if(lastTime != 0)
    {
        var elapsed = timeNow - lastTime;

        xRot += (90 * elapsed) / 1000.0;
        yRot += (90 * elapsed) / 1000.0;
        zRot += (90 * elapsed) / 1000.0;
    }
    lastTime = timeNow;
}
</script>
```

键盘交互与纹理过滤

```
<div class="page-header"><h3>6、键盘输入与纹理过滤</h3></div>

<canvas id = "test06-canvas" width = "800" height = "600"></canvas>
<li>逗号/句号 控制 靠近/远离</li>
<li>WSAD键控制四个方向旋转速度</li>
<li>F键控制纹理过滤类型切换</li>
<script id = "shader-vs" type = "x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    attribute vec2 aTextureCoord;

    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;

    varying vec2 vTextureCoord;
    void main(void)
    {
        gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
        vTextureCoord = aTextureCoord;
    }
</script>

<script id = "shader-fs" type = "x-shader/x-fragment">
    precision mediump float;
    varying vec2 vTextureCoord;
    uniform sampler2D uSampler;
    void main(void)
    {
        gl_FragColor =
            texture2D(uSampler, vec2(vTextureCoord.s, vTextureCoord.t));
    }
</script>
<script type="text/javascript">
    requestAnimFrame = window.requestAnimationFrame ||
        window.mozRequestAnimationFrame ||
        window.webkitRequestAnimationFrame ||
        window.msRequestAnimationFrame ||
        window.oRequestAnimationFrame ||
        function(callback) { setTimeout(callback, 1000 / 60); };
</script>
<script type="text/javascript">

$(document).ready(function ()
{
    webGLStart();
});

function webGLStart()
{
    var canvas = $("#test06-canvas")[0];
    initGL(canvas);
    initShaders();
    initBuffers();
    initTexture();
```

```
    gl.clearColor(0.0, 0.0, 0.0, 1.0);
    gl.enable(gl.DEPTH_TEST);

//    setTimeout("tick()", 100);

    $(document).keydown(handleKeyDown);
    $(document).keyup(handleKeyUp);
}
var crateTextures = Array();
function initTexture()
{
    var crateImage = new Image();
    for(var i = 0; i < 3; i ++)
    {
        var texture = gl.createTexture();
        texture.image = crateImage;
        crateTextures.push(texture);
    }
    crateImage.onload = function()
    {
        handleLoadedTexture(crateTextures);
        tick();
    }
    crateImage.src = "/Public/image/crate.gif";
}
function handleLoadedTexture(textures)
{
    gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, 1);

    gl.bindTexture(gl.TEXTURE_2D, textures[0]);
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA,
        gl.UNSIGNED_BYTE, textures[0].image);
    gl.texParameteri(gl.TEXTURE_2D,
        gl.TEXTURE_MAG_FILTER, gl.NEAREST);
    gl.texParameteri(gl.TEXTURE_2D,
        gl.TEXTURE_MIN_FILTER, gl.NEAREST);

    gl.bindTexture(gl.TEXTURE_2D, textures[1]);
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA,
        gl.UNSIGNED_BYTE, textures[1].image);
    gl.texParameteri(gl.TEXTURE_2D,
        gl.TEXTURE_MAG_FILTER, gl.LINEAR);
    gl.texParameteri(gl.TEXTURE_2D,
        gl.TEXTURE_MIN_FILTER, gl.LINEAR);

    gl.bindTexture(gl.TEXTURE_2D, textures[2]);
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA,
        gl.UNSIGNED_BYTE, textures[2].image);
    gl.texParameteri(gl.TEXTURE_2D,
        gl.TEXTURE_MAG_FILTER, gl.LINEAR);
    gl.texParameteri(gl.TEXTURE_2D,
        gl.TEXTURE_MIN_FILTER, gl.LINEAR_MIPMAP_NEAREST);

    gl.generateMipmap(gl.TEXTURE_2D);

    gl.bindTexture(gl.TEXTURE_2D, null);
}
function tick()
{
```

```
    requestAnimFrame(tick);
    handleKeys();
    drawScene();
    animate();
}

var gl;
function initGL(canvas)
{
    try
    {
        gl = canvas.getContext("webgl") || canvas.getContext("experimental-webgl");
        gl.viewportWidth = canvas.width;
        gl.viewportHeight = canvas.height;
    }catch(e){}
    if(!gl)
    {
        alert("无法初始化“WebGL”。");
    }
}

function getShader(gl, id)
{
    var shaderScript = $("#" + id);
    if(!shaderScript.length)
    {
        return null;
    }
    var str = shaderScript.text();

    var shader;
    if(shaderScript[0].type == "x-shader/x-fragment")
    {
        shader = gl.createShader(gl.FRAGMENT_SHADER);
    }
    else if(shaderScript[0].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;
}

var shaderProgram;
function initShaders()
{
    var fragmentShader = getShader(gl, "shader-fs");
    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("无法初始化“Shader”。");
    }
    gl.useProgram(shaderProgram);

    shaderProgram.vertexPositionAttribute =
        gl.getAttribLocation(shaderProgram, "aVertexPosition");
    gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);

    shaderProgram.textureCoordAttribute =
        gl.getAttribLocation(shaderProgram, "aTextureCoord");
    gl.enableVertexAttribArray(shaderProgram.textureCoordAttribute);

    shaderProgram.pMatrixUniform =
        gl.getUniformLocation(shaderProgram, "uPMatrix");
    shaderProgram.mvMatrixUniform =
        gl.getUniformLocation(shaderProgram, "uMVMatrix");
    shaderProgram.samplerUniform =
        gl.getUniformLocation(shaderProgram, "uSampler");
}


var mvMatrix = mat4.create();

var mvMatrixStack = [];

var pMatrix = mat4.create();

function mvPushMatrix()
{
    var copy = mat4.clone(mvMatrix);
    mvMatrixStack.push(copy);
}
function mvPopMatrix()
{
    if(mvMatrixStack.length == 0)
    {
        throw "不合法的矩阵出栈操作!";
    }
    mvMatrix = mvMatrixStack.pop();
}

function setMatrixUniforms()
{
    gl.uniformMatrix4fv(shaderProgram.pMatrixUniform, false, pMatrix);
    gl.uniformMatrix4fv(shaderProgram.mvMatrixUniform, false, mvMatrix);
}


var cubeVertexPositionBUffer;
var cubeVertexTextureCoordBuffer;
var cubeVertexIndexBuffer;
```

```
function initBuffers()
{
    cubeVertexPositionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
    vertices = [
                // 正面
                -1.0, -1.0,  1.0,
                 1.0, -1.0,  1.0,
                 1.0,  1.0,  1.0,
                -1.0,  1.0,  1.0,

                // 背面
                -1.0, -1.0, -1.0,
                -1.0,  1.0, -1.0,
                 1.0,  1.0, -1.0,
                 1.0, -1.0, -1.0,

                // 顶部
                -1.0,  1.0, -1.0,
                -1.0,  1.0,  1.0,
                 1.0,  1.0,  1.0,
                 1.0,  1.0, -1.0,

                // 底部
                -1.0, -1.0, -1.0,
                 1.0, -1.0, -1.0,
                 1.0, -1.0,  1.0,
                -1.0, -1.0,  1.0,

                // 右侧面
                 1.0, -1.0, -1.0,
                 1.0,  1.0, -1.0,
                 1.0,  1.0,  1.0,
                 1.0, -1.0,  1.0,

                // 左侧面
                -1.0, -1.0, -1.0,
                -1.0, -1.0,  1.0,
                -1.0,  1.0,  1.0,
                -1.0,  1.0, -1.0,
                ];
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices),
        gl.STATIC_DRAW);
    cubeVertexPositionBuffer.itemSize = 3;
    cubeVertexPositionBuffer.numItems = 24;

    cubeVertexTextureCoordBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexTextureCoordBuffer);
    textureCoords = [
                    // 正面
                    0.0, 0.0,
                    1.0, 0.0,
                    1.0, 1.0,
                    0.0, 1.0,

                    // 背面
                    1.0, 0.0,
                    1.0, 1.0,
                    0.0, 1.0,
```

```
                    0.0, 0.0,

                    // 顶部
                    0.0, 1.0,
                    0.0, 0.0,
                    1.0, 0.0,
                    1.0, 1.0,

                    // 底部
                    1.0, 1.0,
                    0.0, 1.0,
                    0.0, 0.0,
                    1.0, 0.0,

                    // 右侧面
                    1.0, 0.0,
                    1.0, 1.0,
                    0.0, 1.0,
                    0.0, 0.0,

                    // 左侧面
                    0.0, 0.0,
                    1.0, 0.0,
                    1.0, 1.0,
                    0.0, 1.0,
                   ];
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(textureCoords),
        gl.STATIC_DRAW);
    cubeVertexTextureCoordBuffer.itemSize = 2;
    cubeVertexTextureCoordBuffer.numItems = 24;

    cubeVertexIndexBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);
    var cubeVertexIndices =
        [
          0, 1, 2,      0, 2, 3,    // 正面
          4, 5, 6,      4, 6, 7,    // 背面
          8, 9, 10,     8, 10, 11,  // 顶部
          12, 13, 14,   12, 14, 15, // 底部
          16, 17, 18,   16, 18, 19, // 右侧面
          20, 21, 22,   20, 22, 23  // 左侧面
        ];
    gl.bufferData(gl.ELEMENT_ARRAY_BUFFER,
        new Uint16Array(cubeVertexIndices), gl.STATIC_DRAW);
    cubeVertexIndexBuffer.itemSize = 1;
    cubeVertexIndexBuffer.numItems = 36;
}

var xRot = 0;
var xSpeed = 0;

var yRot = 0;
var ySpeed = 0;

var z = -5.0;

var filter = 0;

var currentlyPressedKeys = {};
```

```
function handleKeyDown(event)
{
    currentlyPressedKeys[event.keyCode] = true;
    if(String.fromCharCode(event.keyCode) == "F")
    {
        filter += 1;
        if(filter == 3)
        {
            filter = 0;
        }
    }
}
function handleKeyUp(event)
{
    currentlyPressedKeys[event.keyCode] = false;
}
function handleKeys()
{
    if(currentlyPressedKeys[190])
    {
        //"."/">"句号键
        z -= 0.05;
    }
    if(currentlyPressedKeys[188])
    {
        //","/"<"逗号键
        z += 0.05;
    }
    if(currentlyPressedKeys[65])
    {
        //A
        ySpeed -= 1;
    }
    if(currentlyPressedKeys[68])
    {
        //D
        ySpeed += 1;
    }
    if(currentlyPressedKeys[87])
    {
        //W
        xSpeed -= 1;
    }
    if(currentlyPressedKeys[83])
    {
        //S
        xSpeed += 1;
    }
}
function drawScene()
{
    gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    mat4.perspective(pMatrix, 45, gl.viewportWidth /
        gl.viewportHeight, 0.1, 100.0);

    mat4.identity(mvMatrix);
```

```
    mat4.translate(mvMatrix, mvMatrix, [0.0, 0.0, z]);

    mat4.rotate(mvMatrix, mvMatrix, degToRad(xRot), [1, 0, 0]);
    mat4.rotate(mvMatrix, mvMatrix, degToRad(yRot), [0, 1, 0]);

    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
        cubeVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexTextureCoordBuffer);
    gl.vertexAttribPointer(shaderProgram.textureCoordAttribute,
        cubeVertexTextureCoordBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.activeTexture(gl.TEXTURE0);
    gl.bindTexture(gl.TEXTURE_2D, crateTextures[filter]);
    gl.uniform1i(shaderProgram.samplerUniform, 0);

    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);
    setMatrixUniforms();
    gl.drawElements(gl.TRIANGLES,
        cubeVertexIndexBuffer.numItems, gl.UNSIGNED_SHORT, 0);
}
function degToRad(degrees)
{
    return degrees * Math.PI / 180;
}
var lastTime = 0;
function animate()
{
    var timeNow = new Date().getTime();
    if(lastTime != 0)
    {
        var elapsed = timeNow - lastTime;

        xRot += (xSpeed * elapsed) / 1000.0;
        yRot += (ySpeed * elapsed) / 1000.0;
    }
    lastTime = timeNow;
}
</script>
```

方向光与环境光

```
<div class="page-header"><h3>7、方向光与环境光</h3></div>

<canvas id = "test07-canvas" width = "800" height = "600"></canvas>

<br/>
<input type="checkbox" id="lighting" checked /> 使用光照
<br/>
(逗号/句号 控制 靠近/远离, WSAD键控制四个方向旋转速度)
<br/>
<h4>方向光:</h4>
<table>
    <tr>
        <td><b>方向:</b></td>
        <td>X: <input type="text" id="lightDirectionX" value="-0.25" /></td>
        <td>Y: <input type="text" id="lightDirectionY" value="-0.25" /></td>
        <td>Z: <input type="text" id="lightDirectionZ" value="-1.0" /></td>
    </tr>
    <tr>
        <td><b>颜色:</b></td>
        <td>R: <input type="text" id="directionalR" value="0.8" /></td>
        <td>G: <input type="text" id="directionalG" value="0.8" /></td>
        <td>B: <input type="text" id="directionalB" value="0.8" /></td>
    </tr>
</table>
<h4>环境光:</h4>
<table style="border: 0; padding: 10px;">
    <tr>
        <td><b>颜色:</b></td>
        <td>R: <input type="text" id="ambientR" value="0.2" /></td>
        <td>G: <input type="text" id="ambientG" value="0.2" /></td>
        <td>B: <input type="text" id="ambientB" value="0.2" /></td>
    </tr>
</table>

<script id = "shader-vs" type = "x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    attribute vec3 aVertexNormal;
    attribute vec2 aTextureCoord;

    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;
    uniform mat3 uNMatrix;

    uniform vec3 uAmbientColor;

    uniform vec3 uLightingDirection;
    uniform vec3 uDirectionalColor;

    uniform bool uUseLighting;

    varying vec2 vTextureCoord;
    varying vec3 vLightWeighting;

    void main(void)
```

```
    {
        gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
        vTextureCoord = aTextureCoord;

        if(!uUseLighting)
        {
            vLightWeighting = vec3(1.0, 1.0, 1.0);
        }
        else
        {
            vec3 transformedNormal = uNMatrix * aVertexNormal;
            float directionalLightWeighting =
                max(dot(transformedNormal, uLightingDirection), 0.0);
            vLightWeighting =
                uAmbientColor + uDirectionalColor * directionalLightWeighting;
        }
    }
</script>

<script id = "shader-fs" type = "x-shader/x-fragment">
    precision mediump float;
    varying vec2 vTextureCoord;
    varying vec3 vLightWeighting;

    uniform sampler2D uSampler;
    void main(void)
    {
        vec4 textureColor =
            texture2D(uSampler, vec2(vTextureCoord.s, vTextureCoord.t));
        gl_FragColor =
            vec4(textureColor.rgb * vLightWeighting, textureColor.a);
    }
</script>
<script type="text/javascript">
    requestAnimFrame = window.requestAnimationFrame ||
        window.mozRequestAnimationFrame ||
        window.webkitRequestAnimationFrame ||
        window.msRequestAnimationFrame ||
        window.oRequestAnimationFrame ||
        function(callback) { setTimeout(callback, 1000 / 60); };
</script>
<script type="text/javascript">

$(document).ready(function ()
{
    webGLStart();
});

function webGLStart()
{
    var canvas = $("#test07-canvas")[0];
    initGL(canvas);
    initShaders();
    initBuffers();
    initTexture();

    gl.clearColor(0.0, 0.0, 0.0, 1.0);
    gl.enable(gl.DEPTH_TEST);
```

```
//    setTimeout("tick()", 100);

    $(document).keydown(handleKeyDown);
    $(document).keyup(handleKeyUp);
}
function tick()
{
    requestAnimFrame(tick);
    handleKeys();
    drawScene();
    animate();
}

var gl;
function initGL(canvas)
{
    try
    {
        gl = canvas.getContext("webgl") || canvas.getContext("experimental-webgl");
        gl.viewportWidth = canvas.width;
        gl.viewportHeight = canvas.height;
    }catch(e){}
    if(!gl)
    {
        alert("无法初始化“WebGL”。");
    }
}

var crateTexture;
function initTexture()
{
    crateTexture = gl.createTexture();
    crateTexture.image = new Image();
    crateTexture.image.onload = function()
    {
        handleLoadedTexture(crateTexture);
        tick();
    }
    crateTexture.image.src = "/Public/image/crate.gif";
}
function handleLoadedTexture(texture)
{
    gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, 1);


    gl.bindTexture(gl.TEXTURE_2D, texture);
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA,
        gl.UNSIGNED_BYTE, texture.image);
    gl.texParameteri(gl.TEXTURE_2D,
        gl.TEXTURE_MAG_FILTER, gl.LINEAR);
    gl.texParameteri(gl.TEXTURE_2D,
        gl.TEXTURE_MIN_FILTER, gl.LINEAR_MIPMAP_NEAREST);

    gl.generateMipmap(gl.TEXTURE_2D);

    gl.bindTexture(gl.TEXTURE_2D, null);
}
function getShader(gl, id)
{
```

```
    var shaderScript = $("#" + id);
    if(!shaderScript.length)
    {
        return null;
    }
    var str = shaderScript.text();

    var shader;
    if(shaderScript[0].type == "x-shader/x-fragment")
    {
        shader = gl.createShader(gl.FRAGMENT_SHADER);
    }
    else if(shaderScript[0].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;
}

var shaderProgram;
function initShaders()
{
    var fragmentShader = getShader(gl, "shader-fs");
    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("无法初始化“Shader”。");
    }
    gl.useProgram(shaderProgram);

    shaderProgram.vertexPositionAttribute =
        gl.getAttribLocation(shaderProgram, "aVertexPosition");
    gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);

    shaderProgram.vertexNormalAttribute =
        gl.getAttribLocation(shaderProgram, "aVertexNormal");
    gl.enableVertexAttribArray(shaderProgram.vertexNormalAttribute);

    shaderProgram.textureCoordAttribute =
        gl.getAttribLocation(shaderProgram, "aTextureCoord");
    gl.enableVertexAttribArray(shaderProgram.textureCoordAttribute);
```

```
    shaderProgram.pMatrixUniform =
        gl.getUniformLocation(shaderProgram, "uPMatrix");
    shaderProgram.mvMatrixUniform =
        gl.getUniformLocation(shaderProgram, "uMVMatrix");
    shaderProgram.nMatrixUniform  =
        gl.getUniformLocation(shaderProgram, "uNMatrix");
    shaderProgram.samplerUniform =
        gl.getUniformLocation(shaderProgram, "uSampler");
    shaderProgram.useLightingUniform =
        gl.getUniformLocation(shaderProgram, "uUseLighting");
    shaderProgram.ambientColorUniform =
        gl.getUniformLocation(shaderProgram, "uAmbientColor");
    shaderProgram.lightingDirectionUniform =
        gl.getUniformLocation(shaderProgram, "uLightingDirection");
    shaderProgram.directionalColorUniform =
        gl.getUniformLocation(shaderProgram, "uDirectionalColor");
}


var mvMatrix = mat4.create();

var mvMatrixStack = [];

var pMatrix = mat4.create();

function mvPushMatrix()
{
    var copy = mat4.clone(mvMatrix);
    mvMatrixStack.push(copy);
}
function mvPopMatrix()
{
    if(mvMatrixStack.length == 0)
    {
        throw "不合法的矩阵出栈操作!";
    }
    mvMatrix = mvMatrixStack.pop();
}

function setMatrixUniforms()
{
    gl.uniformMatrix4fv(shaderProgram.pMatrixUniform, false, pMatrix);
    gl.uniformMatrix4fv(shaderProgram.mvMatrixUniform, false, mvMatrix);


    var normalMatrix = mat3.create();

    mat3.fromMat4(normalMatrix, mvMatrix);
    mat3.invert(normalMatrix, normalMatrix);
    mat3.transpose(normalMatrix, normalMatrix);

    gl.uniformMatrix3fv(shaderProgram.nMatrixUniform, false, normalMatrix);
}


var cubeVertexPositionBUffer;
var cubeVertexTextureCoordBuffer;
var cubeVertexIndexBuffer;
var cubeVertexNormalBuffer;
```

```
function initBuffers()
{
    cubeVertexPositionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
    vertices = [
                // 正面
                -1.0, -1.0,  1.0,
                 1.0, -1.0,  1.0,
                 1.0,  1.0,  1.0,
                -1.0,  1.0,  1.0,

                // 背面
                -1.0, -1.0, -1.0,
                -1.0,  1.0, -1.0,
                 1.0,  1.0, -1.0,
                 1.0, -1.0, -1.0,

                // 顶部
                -1.0,  1.0, -1.0,
                -1.0,  1.0,  1.0,
                 1.0,  1.0,  1.0,
                 1.0,  1.0, -1.0,

                // 底部
                -1.0, -1.0, -1.0,
                 1.0, -1.0, -1.0,
                 1.0, -1.0,  1.0,
                -1.0, -1.0,  1.0,

                // 右侧面
                 1.0, -1.0, -1.0,
                 1.0,  1.0, -1.0,
                 1.0,  1.0,  1.0,
                 1.0, -1.0,  1.0,

                // 左侧面
                -1.0, -1.0, -1.0,
                -1.0, -1.0,  1.0,
                -1.0,  1.0,  1.0,
                -1.0,  1.0, -1.0,
                ];
    gl.bufferData(gl.ARRAY_BUFFER,
        new Float32Array(vertices), gl.STATIC_DRAW);
    cubeVertexPositionBuffer.itemSize = 3;
    cubeVertexPositionBuffer.numItems = 24;

    cubeVertexNormalBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexNormalBuffer);
    var vertexNormals = [
                      // 正面
                        0.0,  0.0,  1.0,
                        0.0,  0.0,  1.0,
                        0.0,  0.0,  1.0,
                        0.0,  0.0,  1.0,

                      // 背面
                        0.0,  0.0, -1.0,
                        0.0,  0.0, -1.0,
```

```
                    0.0,  0.0, -1.0,
                    0.0,  0.0, -1.0,

                  // 顶部
                    0.0,  1.0,  0.0,
                    0.0,  1.0,  0.0,
                    0.0,  1.0,  0.0,
                    0.0,  1.0,  0.0,

                  // 底部
                    0.0, -1.0,  0.0,
                    0.0, -1.0,  0.0,
                    0.0, -1.0,  0.0,
                    0.0, -1.0,  0.0,

                  // 右侧面
                    1.0,  0.0,  0.0,
                    1.0,  0.0,  0.0,
                    1.0,  0.0,  0.0,
                    1.0,  0.0,  0.0,

                  // 左侧面
                   -1.0,  0.0,  0.0,
                   -1.0,  0.0,  0.0,
                   -1.0,  0.0,  0.0,
                   -1.0,  0.0,  0.0,
                 ];
    gl.bufferData(gl.ARRAY_BUFFER,
        new Float32Array(vertexNormals), gl.STATIC_DRAW);
    cubeVertexNormalBuffer.itemSize = 3;
    cubeVertexNormalBuffer.numItems = 24;

    cubeVertexTextureCoordBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexTextureCoordBuffer);
    textureCoords = [
                  // 正面
                  0.0, 0.0,
                  1.0, 0.0,
                  1.0, 1.0,
                  0.0, 1.0,

                  // 背面
                  1.0, 0.0,
                  1.0, 1.0,
                  0.0, 1.0,
                  0.0, 0.0,

                  // 顶部
                  0.0, 1.0,
                  0.0, 0.0,
                  1.0, 0.0,
                  1.0, 1.0,

                  // 底部
                  1.0, 1.0,
                  0.0, 1.0,
                  0.0, 0.0,
                  1.0, 0.0,
```

```
                    // 右侧面
                    1.0, 0.0,
                    1.0, 1.0,
                    0.0, 1.0,
                    0.0, 0.0,

                    // 左侧面
                    0.0, 0.0,
                    1.0, 0.0,
                    1.0, 1.0,
                    0.0, 1.0,
                   ];
    gl.bufferData(gl.ARRAY_BUFFER,
        new Float32Array(textureCoords), gl.STATIC_DRAW);
    cubeVertexTextureCoordBuffer.itemSize = 2;
    cubeVertexTextureCoordBuffer.numItems = 24;

    cubeVertexIndexBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);
    var cubeVertexIndices =
        [
          0, 1, 2,        0, 2, 3,    // 正面
          4, 5, 6,        4, 6, 7,    // 背面
          8, 9, 10,        8, 10, 11,  // 顶部
          12, 13, 14,    12, 14, 15, // 底部
          16, 17, 18,    16, 18, 19, // 右侧面
          20, 21, 22,    20, 22, 23  // 左侧面
        ];
    gl.bufferData(gl.ELEMENT_ARRAY_BUFFER,
        new Uint16Array(cubeVertexIndices), gl.STATIC_DRAW);
    cubeVertexIndexBuffer.itemSize = 1;
    cubeVertexIndexBuffer.numItems = 36;


}

var xRot = 0;
var xSpeed = 3;

var yRot = 0;
var ySpeed = -3;

var z = -5.0;

var filter = 0;

var currentlyPressedKeys = {};
function handleKeyDown(event)
{
    currentlyPressedKeys[event.keyCode] = true;
    if(String.fromCharCode(event.keyCode) == "F")
    {
        filter += 1;
        if(filter == 3)
        {
            filter = 0;
        }
    }
}
```

```
function handleKeyUp(event)
{
    currentlyPressedKeys[event.keyCode] = false;
}
function handleKeys()
{
    if(currentlyPressedKeys[190])
    {
        //"."/">"句号键
        z -= 0.05;
    }
    if(currentlyPressedKeys[188])
    {
        //","/"<"逗号键
        z += 0.05;
    }
    if(currentlyPressedKeys[65])
    {
        //A
        ySpeed -= 1;
    }
    if(currentlyPressedKeys[68])
    {
        //D
        ySpeed += 1;
    }
    if(currentlyPressedKeys[87])
    {
        //W
        xSpeed -= 1;
    }
    if(currentlyPressedKeys[83])
    {
        //S
        xSpeed += 1;
    }
}
function drawScene()
{
    gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    mat4.perspective(pMatrix, 45,
        gl.viewportWidth / gl.viewportHeight, 0.1, 100.0);

    mat4.identity(mvMatrix);

    mat4.translate(mvMatrix, mvMatrix, [0.0, 0.0, z]);

    mat4.rotate(mvMatrix, mvMatrix, degToRad(xRot), [1, 0, 0]);
    mat4.rotate(mvMatrix, mvMatrix, degToRad(yRot), [0, 1, 0]);

    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
        cubeVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexNormalBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexNormalAttribute,
        cubeVertexNormalBuffer.itemSize, gl.FLOAT, false, 0, 0);
```

```
    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexTextureCoordBuffer);
    gl.vertexAttribPointer(shaderProgram.textureCoordAttribute,
        cubeVertexTextureCoordBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.activeTexture(gl.TEXTURE0);
    gl.bindTexture(gl.TEXTURE_2D, crateTexture);
    gl.uniform1i(shaderProgram.samplerUniform, 0);

    var lighting = $("#lighting").is(":checked");
    gl.uniform1i(shaderProgram.useLightingUniform, lighting);
    if(lighting)
    {
        gl.uniform3f(
            shaderProgram.ambientColorUniform,
            parseFloat($("#ambientR").val()),
            parseFloat($("#ambientG").val()),
            parseFloat($("#ambientB").val())
            );
        var lightingDirection = [
            parseFloat($("#lightDirectionX").val()),
            parseFloat($("#lightDirectionY").val()),
            parseFloat($("#lightDirectionZ").val())
            ];
        var adjustedLD = vec3.create();
        vec3.normalize(adjustedLD, lightingDirection);
        vec3.scale(adjustedLD, adjustedLD, -1);
        gl.uniform3fv(shaderProgram.lightingDirectionUniform, adjustedLD);
        gl.uniform3f(
            shaderProgram.directionalColorUniform,
            parseFloat($("#directionalR").val()),
            parseFloat($("#directionalG").val()),
            parseFloat($("#directionalB").val())
            );
    }
    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);
    setMatrixUniforms();
    gl.drawElements(gl.TRIANGLES,
        cubeVertexIndexBuffer.numItems, gl.UNSIGNED_SHORT, 0);
}
function degToRad(degrees)
{
    return degrees * Math.PI / 180;
}
var lastTime = 0;
function animate()
{
    var timeNow = new Date().getTime();
    if(lastTime != 0)
    {
        var elapsed = timeNow - lastTime;

        xRot += (xSpeed * elapsed) / 1000.0;
        yRot += (ySpeed * elapsed) / 1000.0;
    }
    lastTime = timeNow;
}
</script>
```

深度缓冲，透明度与混合

```
<div class="page-header"><h3>8、深度缓冲，透明度与混合</h3></div>

<canvas id = "test08-canvas" width = "800" height = "600"></canvas>
<br/>
<input type="checkbox" id="blending" checked /> 使用混合
<br/>
不透明度（Alpha）级别 <input type="text" id="alpha" value="0.5" /><br/>
<br/>
<input type="checkbox" id="lighting" checked /> 使用光照
<br/>
(逗号/句号 控制 靠近/远离，WSAD键控制四个方向旋转速度)
<br/>
<h4>方向光:</h4>
<table>
    <tr>
        <td><b>方向:</b></td>
        <td>X: <input type="text" id="lightDirectionX" value="-0.25" /></td>
        <td>Y: <input type="text" id="lightDirectionY" value="-0.25" /></td>
        <td>Z: <input type="text" id="lightDirectionZ" value="-1.0" /></td>
    </tr>
    <tr>
        <td><b>颜色:</b></td>
        <td>R: <input type="text" id="directionalR" value="0.8" /></td>
        <td>G: <input type="text" id="directionalG" value="0.8" /></td>
        <td>B: <input type="text" id="directionalB" value="0.8" /></td>
    </tr>
</table>
<h4>环境光:</h4>
<table style="border: 0; padding: 10px;">
    <tr>
        <td><b>颜色:</b></td>
        <td>R: <input type="text" id="ambientR" value="0.2" /></td>
        <td>G: <input type="text" id="ambientG" value="0.2" /></td>
        <td>B: <input type="text" id="ambientB" value="0.2" /></td>
    </tr>
</table>

<script id = "shader-vs" type = "x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    attribute vec3 aVertexNormal;
    attribute vec2 aTextureCoord;

    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;
    uniform mat3 uNMatrix;

    uniform vec3 uAmbientColor;

    uniform vec3 uLightingDirection;
    uniform vec3 uDirectionalColor;

    uniform bool uUseLighting;

    varying vec2 vTextureCoord;
```

```
    varying vec3 vLightWeighting;

    void main(void)
    {
        gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
        vTextureCoord = aTextureCoord;

        if(!uUseLighting)
        {
            vLightWeighting = vec3(1.0, 1.0, 1.0);
        }
        else
        {
            vec3 transformedNormal = uNMatrix * aVertexNormal;
            float directionalLightWeighting =
                max(dot(transformedNormal, uLightingDirection), 0.0);
            vLightWeighting =
                uAmbientColor + uDirectionalColor * directionalLightWeighting;
        }
    }
</script>

<script id = "shader-fs" type = "x-shader/x-fragment">
    precision mediump float;
    varying vec2 vTextureCoord;
    varying vec3 vLightWeighting;

    uniform float uAlpha;

    uniform sampler2D uSampler;
    void main(void)
    {
        vec4 textureColor =
            texture2D(uSampler, vec2(vTextureCoord.s, vTextureCoord.t));
        gl_FragColor =
            vec4(textureColor.rgb * vLightWeighting, textureColor.a * uAlpha);
    }
</script>
<script type="text/javascript">
    requestAnimFrame = window.requestAnimationFrame ||
        window.mozRequestAnimationFrame ||
        window.webkitRequestAnimationFrame ||
        window.msRequestAnimationFrame ||
        window.oRequestAnimationFrame ||
        function(callback) { setTimeout(callback, 1000 / 60); };
</script>
<script type="text/javascript">

$(document).ready(function ()
{
    webGLStart();
});

function webGLStart()
{
    var canvas = $("#test08-canvas")[0];
    initGL(canvas);
    initShaders();
    initBuffers();
```

```
    initTexture();

    gl.clearColor(0.0, 0.0, 0.0, 1.0);
    gl.enable(gl.DEPTH_TEST);
//  gl.depthFunc(gl.LESS);
//    setTimeout("tick()", 100);

    $(document).keydown(handleKeyDown);
    $(document).keyup(handleKeyUp);
}
function tick()
{
    requestAnimFrame(tick);
    handleKeys();
    drawScene();
    animate();
}

var gl;
function initGL(canvas)
{
    try
    {
        gl = canvas.getContext("webgl") || canvas.getContext("experimental-webgl");
        gl.viewportWidth = canvas.width;
        gl.viewportHeight = canvas.height;
    }catch(e){}
    if(!gl)
    {
        alert("无法初始化“WebGL”。");
    }
}

var crateTexture;
function initTexture()
{
    crateTexture = gl.createTexture();
    crateTexture.image = new Image();
    crateTexture.image.onload = function()
    {
        handleLoadedTexture(crateTexture);
        tick();
    }
    crateTexture.image.src = "/Public/image/glass.gif";
}
function handleLoadedTexture(texture)
{
    gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, 1);


    gl.bindTexture(gl.TEXTURE_2D, texture);
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA,
        gl.UNSIGNED_BYTE, texture.image);
    gl.texParameteri(gl.TEXTURE_2D,
        gl.TEXTURE_MAG_FILTER, gl.LINEAR);
    gl.texParameteri(gl.TEXTURE_2D,
        gl.TEXTURE_MIN_FILTER, gl.LINEAR_MIPMAP_NEAREST);

    gl.generateMipmap(gl.TEXTURE_2D);
```

```
    gl.bindTexture(gl.TEXTURE_2D, null);
}
function getShader(gl, id)
{
    var shaderScript = $("#" + id);
    if(!shaderScript.length)
    {
        return null;
    }
    var str = shaderScript.text();

    var shader;
    if(shaderScript[0].type == "x-shader/x-fragment")
    {
        shader = gl.createShader(gl.FRAGMENT_SHADER);
    }
    else if(shaderScript[0].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;
}

var shaderProgram;
function initShaders()
{
    var fragmentShader = getShader(gl, "shader-fs");
    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("无法初始化“Shader”。");
    }
    gl.useProgram(shaderProgram);

    shaderProgram.vertexPositionAttribute =
        gl.getAttribLocation(shaderProgram, "aVertexPosition");
    gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);

    shaderProgram.vertexNormalAttribute =
        gl.getAttribLocation(shaderProgram, "aVertexNormal");
    gl.enableVertexAttribArray(shaderProgram.vertexNormalAttribute);
```

```
    shaderProgram.textureCoordAttribute =
        gl.getAttribLocation(shaderProgram, "aTextureCoord");
    gl.enableVertexAttribArray(shaderProgram.textureCoordAttribute);

    shaderProgram.pMatrixUniform =
        gl.getUniformLocation(shaderProgram, "uPMatrix");
    shaderProgram.mvMatrixUniform =
        gl.getUniformLocation(shaderProgram, "uMVMatrix");
    shaderProgram.nMatrixUniform  =
        gl.getUniformLocation(shaderProgram, "uNMatrix");
    shaderProgram.samplerUniform =
        gl.getUniformLocation(shaderProgram, "uSampler");
    shaderProgram.useLightingUniform =
        gl.getUniformLocation(shaderProgram, "uUseLighting");
    shaderProgram.ambientColorUniform =
        gl.getUniformLocation(shaderProgram, "uAmbientColor");
    shaderProgram.lightingDirectionUniform =
        gl.getUniformLocation(shaderProgram, "uLightingDirection");
    shaderProgram.directionalColorUniform =
        gl.getUniformLocation(shaderProgram, "uDirectionalColor");
    shaderProgram.alphaUniform =
        gl.getUniformLocation(shaderProgram, "uAlpha");
}


var mvMatrix = mat4.create();

var mvMatrixStack = [];

var pMatrix = mat4.create();

function mvPushMatrix()
{
    var copy = mat4.clone(mvMatrix);
    mvMatrixStack.push(copy);
}
function mvPopMatrix()
{
    if(mvMatrixStack.length == 0)
    {
        throw "不合法的矩阵出栈操作!";
    }
    mvMatrix = mvMatrixStack.pop();
}

function setMatrixUniforms()
{
    gl.uniformMatrix4fv(shaderProgram.pMatrixUniform, false, pMatrix);
    gl.uniformMatrix4fv(shaderProgram.mvMatrixUniform, false, mvMatrix);


    var normalMatrix = mat3.create();

    mat3.fromMat4(normalMatrix, mvMatrix);
    mat3.invert(normalMatrix, normalMatrix);
    mat3.transpose(normalMatrix, normalMatrix);

    gl.uniformMatrix3fv(shaderProgram.nMatrixUniform, false, normalMatrix);
```

```
}


var cubeVertexPositionBUffer;
var cubeVertexTextureCoordBuffer;
var cubeVertexIndexBuffer;
var cubeVertexNormalBuffer;

function initBuffers()
{
    cubeVertexPositionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
    vertices = [
                // 正面
                -1.0, -1.0,  1.0,
                 1.0, -1.0,  1.0,
                 1.0,  1.0,  1.0,
                -1.0,  1.0,  1.0,

                // 背面
                -1.0, -1.0, -1.0,
                -1.0,  1.0, -1.0,
                 1.0,  1.0, -1.0,
                 1.0, -1.0, -1.0,

                // 顶部
                -1.0,  1.0, -1.0,
                -1.0,  1.0,  1.0,
                 1.0,  1.0,  1.0,
                 1.0,  1.0, -1.0,

                // 底部
                -1.0, -1.0, -1.0,
                 1.0, -1.0, -1.0,
                 1.0, -1.0,  1.0,
                -1.0, -1.0,  1.0,

                // 右侧面
                 1.0, -1.0, -1.0,
                 1.0,  1.0, -1.0,
                 1.0,  1.0,  1.0,
                 1.0, -1.0,  1.0,

                // 左侧面
                -1.0, -1.0, -1.0,
                -1.0, -1.0,  1.0,
                -1.0,  1.0,  1.0,
                -1.0,  1.0, -1.0,
                ];
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices),
        gl.STATIC_DRAW);
    cubeVertexPositionBuffer.itemSize = 3;
    cubeVertexPositionBuffer.numItems = 24;

    cubeVertexNormalBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexNormalBuffer);
    var vertexNormals = [
                      // 正面
                        0.0,  0.0,  1.0,
```

```
                    0.0,  0.0,  1.0,
                    0.0,  0.0,  1.0,
                    0.0,  0.0,  1.0,

                  // 背面
                    0.0,  0.0, -1.0,
                    0.0,  0.0, -1.0,
                    0.0,  0.0, -1.0,
                    0.0,  0.0, -1.0,

                  // 顶部
                    0.0,  1.0,  0.0,
                    0.0,  1.0,  0.0,
                    0.0,  1.0,  0.0,
                    0.0,  1.0,  0.0,

                  // 底部
                    0.0, -1.0,  0.0,
                    0.0, -1.0,  0.0,
                    0.0, -1.0,  0.0,
                    0.0, -1.0,  0.0,

                  // 右侧面
                    1.0,  0.0,  0.0,
                    1.0,  0.0,  0.0,
                    1.0,  0.0,  0.0,
                    1.0,  0.0,  0.0,

                  // 左侧面
                   -1.0,  0.0,  0.0,
                   -1.0,  0.0,  0.0,
                   -1.0,  0.0,  0.0,
                   -1.0,  0.0,  0.0,
                 ];
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertexNormals), gl.STATIC_DRAW);
    cubeVertexNormalBuffer.itemSize = 3;
    cubeVertexNormalBuffer.numItems = 24;

    cubeVertexTextureCoordBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexTextureCoordBuffer);
    textureCoords = [
                  // 正面
                  0.0, 0.0,
                  1.0, 0.0,
                  1.0, 1.0,
                  0.0, 1.0,

                  // 背面
                  1.0, 0.0,
                  1.0, 1.0,
                  0.0, 1.0,
                  0.0, 0.0,

                  // 顶部
                  0.0, 1.0,
                  0.0, 0.0,
                  1.0, 0.0,
                  1.0, 1.0,
```

```
                    // 底部
                    1.0, 1.0,
                    0.0, 1.0,
                    0.0, 0.0,
                    1.0, 0.0,

                    // 右侧面
                    1.0, 0.0,
                    1.0, 1.0,
                    0.0, 1.0,
                    0.0, 0.0,

                    // 左侧面
                    0.0, 0.0,
                    1.0, 0.0,
                    1.0, 1.0,
                    0.0, 1.0,
                   ];
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(textureCoords),
        gl.STATIC_DRAW);
    cubeVertexTextureCoordBuffer.itemSize = 2;
    cubeVertexTextureCoordBuffer.numItems = 24;

    cubeVertexIndexBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);
    var cubeVertexIndices =
        [
          0, 1, 2,         0, 2, 3,    // 正面
          4, 5, 6,         4, 6, 7,    // 背面
          8, 9, 10,         8, 10, 11,  // 顶部
          12, 13, 14,    12, 14, 15, // 底部
          16, 17, 18,    16, 18, 19, // 右侧面
          20, 21, 22,    20, 22, 23  // 左侧面
        ];
    gl.bufferData(gl.ELEMENT_ARRAY_BUFFER,
        new Uint16Array(cubeVertexIndices), gl.STATIC_DRAW);
    cubeVertexIndexBuffer.itemSize = 1;
    cubeVertexIndexBuffer.numItems = 36;


}

var xRot = 0;
var xSpeed = 3;

var yRot = 0;
var ySpeed = -3;

var z = -5.0;

var filter = 0;

var currentlyPressedKeys = {};
function handleKeyDown(event)
{
    currentlyPressedKeys[event.keyCode] = true;
    if(String.fromCharCode(event.keyCode) == "F")
    {
        filter += 1;
```

```
        if(filter == 3)
        {
            filter = 0;
        }
    }
}
function handleKeyUp(event)
{
    currentlyPressedKeys[event.keyCode] = false;
}
function handleKeys()
{
    if(currentlyPressedKeys[190])
    {
        //"."/">"句号键
        z -= 0.05;
    }
    if(currentlyPressedKeys[188])
    {
        //","/"<"逗号键
        z += 0.05;
    }
    if(currentlyPressedKeys[65])
    {
        //A
        ySpeed -= 1;
    }
    if(currentlyPressedKeys[68])
    {
        //D
        ySpeed += 1;
    }
    if(currentlyPressedKeys[87])
    {
        //W
        xSpeed -= 1;
    }
    if(currentlyPressedKeys[83])
    {
        //S
        xSpeed += 1;
    }
}
function drawScene()
{
    gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    mat4.perspective(pMatrix, 45,
        gl.viewportWidth / gl.viewportHeight, 0.1, 100.0);

    mat4.identity(mvMatrix);

    mat4.translate(mvMatrix, mvMatrix, [0.0, 0.0, z]);

    mat4.rotate(mvMatrix, mvMatrix, degToRad(xRot), [1, 0, 0]);
    mat4.rotate(mvMatrix, mvMatrix, degToRad(yRot), [0, 1, 0]);

    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
```

```
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
        cubeVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexNormalBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexNormalAttribute,
        cubeVertexNormalBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexTextureCoordBuffer);
    gl.vertexAttribPointer(shaderProgram.textureCoordAttribute,
        cubeVertexTextureCoordBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.activeTexture(gl.TEXTURE0);
    gl.bindTexture(gl.TEXTURE_2D, crateTexture);
    gl.uniform1i(shaderProgram.samplerUniform, 0);

    var blending = $("#blending").is(":checked");
    if(blending)
    {
        gl.blendFunc(gl.SRC_ALPHA, gl.ONE);
        gl.enable(gl.BLEND);
        gl.disable(gl.DEPTH_TEST);
        gl.uniform1f(shaderProgram.alphaUniform, parseFloat($("#alpha").val()));
    }
    else
    {
        gl.disable(gl.BLEND);
        gl.enable(gl.DEPTH_TEST);
        gl.uniform1f(shaderProgram.alphaUniform, 1);
    }

    var lighting = $("#lighting").is(":checked");
    gl.uniform1i(shaderProgram.useLightingUniform, lighting);
    if(lighting)
    {
        gl.uniform3f(
            shaderProgram.ambientColorUniform,
            parseFloat($("#ambientR").val()),
            parseFloat($("#ambientG").val()),
            parseFloat($("#ambientB").val())
            );
        var lightingDirection = [
            parseFloat($("#lightDirectionX").val()),
            parseFloat($("#lightDirectionY").val()),
            parseFloat($("#lightDirectionZ").val())
            ];
        var adjustedLD = vec3.create();
        vec3.normalize(adjustedLD, lightingDirection);
        vec3.scale(adjustedLD, adjustedLD, -1);
        gl.uniform3fv(shaderProgram.lightingDirectionUniform, adjustedLD);
        gl.uniform3f(
            shaderProgram.directionalColorUniform,
            parseFloat($("#directionalR").val()),
            parseFloat($("#directionalG").val()),
            parseFloat($("#directionalB").val())
            );
    }
    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);
    setMatrixUniforms();
    gl.drawElements(gl.TRIANGLES,
```

```
        cubeVertexIndexBuffer.numItems, gl.UNSIGNED_SHORT, 0);
}
function degToRad(degrees)
{
    return degrees * Math.PI / 180;
}
var lastTime = 0;
function animate()
{
    var timeNow = new Date().getTime();
    if(lastTime != 0)
    {
        var elapsed = timeNow - lastTime;

        xRot += (xSpeed * elapsed) / 1000.0;
        yRot += (ySpeed * elapsed) / 1000.0;
    }
    lastTime = timeNow;
}
</script>
```

优化代码结构，多物体运动

```
<div class="page-header"><h3>9、优化代码结构，多物体运动</h3></div>

<canvas id = "test09-canvas" width = "800" height = "600"></canvas>

<br/>
<input type="checkbox" id="twinkle" /> 发亮
<br/>
(逗号/句号 控制 靠近/远离，W/S 控制旋转)

<script id = "shader-vs" type = "x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    attribute vec2 aTextureCoord;

    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;
    varying vec2 vTextureCoord;

    void main(void)
    {
        gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
        vTextureCoord = aTextureCoord;
    }
</script>

<script id = "shader-fs" type = "x-shader/x-fragment">
    precision mediump float;
    varying vec2 vTextureCoord;

    uniform sampler2D uSampler;

    uniform vec3 uColor;

    void main(void)
    {
        vec4 textureColor =
            texture2D(uSampler, vec2(vTextureCoord.s, vTextureCoord.t));
        gl_FragColor =
            textureColor * vec4(uColor, 1.0);
    }
</script>
<script type="text/javascript">
    requestAnimFrame = window.requestAnimationFrame ||
        window.mozRequestAnimationFrame ||
        window.webkitRequestAnimationFrame ||
        window.msRequestAnimationFrame ||
        window.oRequestAnimationFrame ||
        function(callback) { setTimeout(callback, 1000 / 60); };
</script>
<script type="text/javascript">

$(document).ready(function ()
{
    webGLStart();
});
```

```
function webGLStart()
{
    var canvas = $("#test09-canvas")[0];
    initGL(canvas);
    initShaders();
    initBuffers();
    initTexture();

    initWorldObjects();

    gl.clearColor(0.0, 0.0, 0.0, 1.0);

    $(document).keydown(handleKeyDown);
    $(document).keyup(handleKeyUp);
}
function tick()
{
    requestAnimFrame(tick);
    handleKeys();
    drawScene();
    animate();
}

var gl;
function initGL(canvas)
{
    try
    {
        gl = canvas.getContext("webgl") || canvas.getContext("experimental-webgl");
        gl.viewportWidth = canvas.width;
        gl.viewportHeight = canvas.height;
    }catch(e){}
    if(!gl)
    {
        alert("无法初始化“WebGL”。");
    }
}

var starTexture;
function initTexture()
{
    starTexture = gl.createTexture();
    starTexture.image = new Image();
    starTexture.image.onload = function()
    {
        handleLoadedTexture(starTexture);
        tick();
    }
    starTexture.image.src = "/Public/image/star.gif";
}
function handleLoadedTexture(texture)
{
    gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, 1);


    gl.bindTexture(gl.TEXTURE_2D, texture);
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA,
        gl.UNSIGNED_BYTE, texture.image);
```

```
    gl.texParameteri(gl.TEXTURE_2D,
        gl.TEXTURE_MAG_FILTER, gl.LINEAR);
    gl.texParameteri(gl.TEXTURE_2D,
        gl.TEXTURE_MIN_FILTER, gl.LINEAR);

    gl.bindTexture(gl.TEXTURE_2D, null);
}
function getShader(gl, id)
{
    var shaderScript = $("#" + id);
    if(!shaderScript.length)
    {
        return null;
    }
    var str = shaderScript.text();

    var shader;
    if(shaderScript[0].type == "x-shader/x-fragment")
    {
        shader = gl.createShader(gl.FRAGMENT_SHADER);
    }
    else if(shaderScript[0].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;
}

var shaderProgram;
function initShaders()
{
    var fragmentShader = getShader(gl, "shader-fs");
    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("无法初始化“Shader”。");
    }
    gl.useProgram(shaderProgram);

    shaderProgram.vertexPositionAttribute =
        gl.getAttribLocation(shaderProgram, "aVertexPosition");
    gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);
```

```
    shaderProgram.textureCoordAttribute =
        gl.getAttribLocation(shaderProgram, "aTextureCoord");
    gl.enableVertexAttribArray(shaderProgram.textureCoordAttribute);

    shaderProgram.pMatrixUniform =
        gl.getUniformLocation(shaderProgram, "uPMatrix");
    shaderProgram.mvMatrixUniform =
        gl.getUniformLocation(shaderProgram, "uMVMatrix");
    shaderProgram.samplerUniform =
        gl.getUniformLocation(shaderProgram, "uSampler");
    shaderProgram.colorUniform =
        gl.getUniformLocation(shaderProgram, "uColor");

}


var mvMatrix = mat4.create();

var mvMatrixStack = [];

var pMatrix = mat4.create();

function mvPushMatrix()
{
    var copy = mat4.clone(mvMatrix);
    mvMatrixStack.push(copy);
}
function mvPopMatrix()
{
    if(mvMatrixStack.length == 0)
    {
        throw "不合法的矩阵出栈操作!";
    }
    mvMatrix = mvMatrixStack.pop();
}

function setMatrixUniforms()
{
    gl.uniformMatrix4fv(shaderProgram.pMatrixUniform, false, pMatrix);
    gl.uniformMatrix4fv(shaderProgram.mvMatrixUniform, false, mvMatrix);
}


var starVertexPositionBuffer;
var starVertexTextureCoordBuffer;

function initBuffers()
{
    starVertexPositionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, starVertexPositionBuffer);
    vertices = [
        -1.0, -1.0,  0.0,
         1.0, -1.0,  0.0,
        -1.0,  1.0,  0.0,
         1.0,  1.0,  0.0
        ];
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices),
        gl.STATIC_DRAW);
```

```
    starVertexPositionBuffer.itemSize = 3;
    starVertexPositionBuffer.numItems = 4;

    starVertexTextureCoordBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, starVertexTextureCoordBuffer);
    var textureCoords = [
        0.0, 0.0,
        1.0, 0.0,
        0.0, 1.0,
        1.0, 1.0
        ];
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(textureCoords),
        gl.STATIC_DRAW);
    starVertexTextureCoordBuffer.itemSize = 2;
    starVertexTextureCoordBuffer.numItems = 4;
}


var currentlyPressedKeys = {};
function handleKeyDown(event)
{
    currentlyPressedKeys[event.keyCode] = true;
}
function handleKeyUp(event)
{
    currentlyPressedKeys[event.keyCode] = false;
}

var zoom  = -15;
var tilt = 90;
var spin = 90;
function handleKeys()
{
    if(currentlyPressedKeys[190])
    {
        //"."/">"句号键
        zoom -= 0.1;
    }
    if(currentlyPressedKeys[188])
    {
        //","/"<"逗号键
        zoom += 0.1;
    }
    if(currentlyPressedKeys[87])
    {
        //W
        tilt += 2;
    }
    if(currentlyPressedKeys[83])
    {
        //S
        tilt -= 2;
    }
}
function drawScene()
{
    gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
```

```
    mat4.perspective(pMatrix, 45,
        gl.viewportWidth / gl.viewportHeight, 0.1, 100.0);

    gl.blendFunc(gl.SRC_ALPHA, gl.ONE);
    gl.enable(gl.BLEND);

    mat4.identity(mvMatrix);

    mat4.translate(mvMatrix, mvMatrix, [0.0, 0.0, zoom]);

    mat4.rotate(mvMatrix, mvMatrix, degToRad(tilt), [1, 0, 0]);

    var twinkle = $("#twinkle").is(":checked");
    for(var i in stars)
    {
        stars[i].draw(tilt, spin, twinkle);
        spin += 0.1;
    }
}
var stars = [];
function initWorldObjects()
{
    var numStars = 50;
    for(var i = 0; i < numStars; i ++)
    {
        stars.push(new Star((i / numStars) * 5.0, i / numStars));
    }
}
//==========
//Star类
function Star(startingDistance, rotationSpeed)
{
    this.angle = 0;
    this.dist = startingDistance;
    this.rotationSpeed = rotationSpeed;
    //设置一个起始颜色
    this.randomiseColors();
}
Star.prototype.draw = function(tilt, spin, twinkle)
{
    mvPushMatrix();
    //移动到位
    mat4.rotate(mvMatrix, mvMatrix, degToRad(this.angle), [0.0, 1.0, 0.0]);
    mat4.translate(mvMatrix, mvMatrix, [this.dist, 0.0, 0.0]);
    //旋转
    mat4.rotate(mvMatrix, mvMatrix, degToRad(-this.angle), [0.0, 1.0, 0.0]);
    mat4.rotate(mvMatrix, mvMatrix, degToRad(-tilt), [1.0, 0.0, 0.0]);
    if(twinkle)
    {
        //画一个不旋转的star
        gl.uniform3f(shaderProgram.colorUniform,
            this.twinkleR, this.twinkleG, this.twinkleB);
        drawStar();
    }
    //所有Star围绕Z旋转
    mat4.rotate(mvMatrix, mvMatrix, degToRad(spin), [0.0, 0.0, 1.0]);
    //画颜色
    gl.uniform3f(shaderProgram.colorUniform, this.r, this.g, this.b);
    drawStar();
```

```
    mvPopMatrix();
};

var effectiveFPMS = 60 / 1000;
Star.prototype.animate = function(elapsedTime)
{
    this.angle += this.rotationSpeed * effectiveFPMS * elapsedTime;
    //逐步减小与中心的距离
    this.dist -= 0.01 * effectiveFPMS * elapsedTime;
    if(this.dist < 0.0)
    {
        this.dist += 5.0;
        this.randomiseColors();
    }
};

Star.prototype.randomiseColors = function()
{
    //给星星一个随机的颜色
    this.r = Math.random();
    this.g = Math.random();
    this.b = Math.random();
    //如果打开了发亮开关，则多绘制一次，用另一个随机的颜色作为亮光
    this.twinkleR = Math.random();
    this.twinkleG = Math.random();
    this.twinkleB = Math.random();
};
//==========

function drawStar()
{
    gl.activeTexture(gl.TEXTURE0);
    gl.bindTexture(gl.TEXTURE_2D, starTexture);
    gl.uniform1i(shaderProgram.samplerUniform, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, starVertexTextureCoordBuffer);
    gl.vertexAttribPointer(shaderProgram.textureCoordAttribute,
        starVertexTextureCoordBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, starVertexPositionBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
        starVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);

    setMatrixUniforms();
    gl.drawArrays(gl.TRIANGLE_STRIP, 0, starVertexPositionBuffer.numItems);
}

function degToRad(degrees)
{
    return degrees * Math.PI / 180;
}
var lastTime = 0;
function animate()
{
    var timeNow = new Date().getTime();
    if(lastTime != 0)
    {
        var elapsed = timeNow - lastTime;
```

```
            for(var i in stars)
            {
                stars[i].animate(elapsed);
            }
        }
        lastTime = timeNow;
    }
    </script>
```

加载场景，基本相机操作

```
<div class="page-header"><h3>10、加载场景，基本相机操作</h3></div>

<canvas id = "test10-canvas" width = "800" height = "600"></canvas>
<div id="loadingtext">正在加载世界……</div>
<br/>
(WSAD控制移动，逗号/句号 控制 上下看)

<script id = "shader-vs" type = "x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    attribute vec2 aTextureCoord;

    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;
    varying vec2 vTextureCoord;

    void main(void)
    {
        gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
        vTextureCoord = aTextureCoord;
    }
</script>

<script id = "shader-fs" type = "x-shader/x-fragment">
    precision mediump float;
    varying vec2 vTextureCoord;
    uniform sampler2D uSampler;
    void main(void)
    {
        gl_FragColor = texture2D(uSampler, vec2(vTextureCoord.s, vTextureCoord.t));
    }
</script>
<script type="text/javascript">
    requestAnimFrame = window.requestAnimationFrame ||
        window.mozRequestAnimationFrame ||
        window.webkitRequestAnimationFrame ||
        window.msRequestAnimationFrame ||
        window.oRequestAnimationFrame ||
        function(callback) { setTimeout(callback, 1000 / 60); };
</script>
<script type="text/javascript">

$(document).ready(function ()
{
    webGLStart();
});

function webGLStart()
{
    var canvas = $("#test10-canvas")[0];
    initGL(canvas);
    initShaders();
    initBuffers();
    initTexture();
```

```
    loadWorld();

    gl.clearColor(0.0, 0.0, 0.0, 1.0);
    gl.enable(gl.DEPTH_TEST);

    $(document).keydown(handleKeyDown);
    $(document).keyup(handleKeyUp);
}
function tick()
{
    requestAnimFrame(tick);
    handleKeys();
    drawScene();
    animate();
}
function loadWorld()
{
    $.getJSON(
        "/Public/json/world.json",
        function(data)
        {
            handleLoadedWorld(data);
        }
    );
}
var worldVertexPositionBuffer = null;
var worldVertexTextureCoordBuffer = null;
function handleLoadedWorld(data)
{
    var vertexCount = 0;
    var vertexPositions = [];
    var vertexTextureCoords = [];

    for(var i = 0; typeof(data[i]) != "undefined"; i ++)
    {
        //环境中一个部分的顶点坐标
        vertexPositions =
            vertexPositions.concat(data[i].vertexPositions);
        //然后是纹理坐标
        vertexTextureCoords =
            vertexTextureCoords.concat(data[i].vertexTextureCoords);
        vertexCount ++;
    }
    worldVertexPositionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, worldVertexPositionBuffer);
    gl.bufferData(gl.ARRAY_BUFFER,
        new Float32Array(vertexPositions), gl.STATIC_DRAW);
    worldVertexPositionBuffer.itemSize = 3;
    worldVertexPositionBuffer.numItems = data.vertexCount;

    worldVertexTextureCoordBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, worldVertexTextureCoordBuffer);
    gl.bufferData(gl.ARRAY_BUFFER,
        new Float32Array(vertexTextureCoords), gl.STATIC_DRAW);
    worldVertexTextureCoordBuffer.itemSize = 2;
    worldVertexTextureCoordBuffer.numItems = data.vertexCount;

    $("#loadingtext").text("");
}
```

```
var gl;
function initGL(canvas)
{
    try
    {
        gl = canvas.getContext("webgl") || canvas.getContext("experimental-webgl");
        gl.viewportWidth = canvas.width;
        gl.viewportHeight = canvas.height;
    }catch(e){}
    if(!gl)
    {
        alert("无法初始化“WebGL”。");
    }
}

var crackTexture;
function initTexture()
{
    crackTexture = gl.createTexture();
    crackTexture.image = new Image();
    crackTexture.image.onload = function()
    {
        handleLoadedTexture(crackTexture);
        tick();
    }
    crackTexture.image.src = "/Public/image/cracktexture.jpg";
}
function handleLoadedTexture(texture)
{
    gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, 1);


    gl.bindTexture(gl.TEXTURE_2D, texture);
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA,
        gl.UNSIGNED_BYTE, texture.image);
    gl.texParameteri(gl.TEXTURE_2D,
        gl.TEXTURE_MAG_FILTER, gl.LINEAR);
    gl.texParameteri(gl.TEXTURE_2D,
        gl.TEXTURE_MIN_FILTER, gl.LINEAR_MIPMAP_NEAREST);

    gl.generateMipmap(gl.TEXTURE_2D);

    gl.bindTexture(gl.TEXTURE_2D, null);
}
function getShader(gl, id)
{
    var shaderScript = $("#" + id);
    if(!shaderScript.length)
    {
        return null;
    }
    var str = shaderScript.text();

    var shader;
    if(shaderScript[0].type == "x-shader/x-fragment")
    {
        shader = gl.createShader(gl.FRAGMENT_SHADER);
    }
    else if(shaderScript[0].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;
}

var shaderProgram;
function initShaders()
{
    var fragmentShader = getShader(gl, "shader-fs");
    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("无法初始化“Shader”。");
    }
    gl.useProgram(shaderProgram);

    shaderProgram.vertexPositionAttribute =
        gl.getAttribLocation(shaderProgram, "aVertexPosition");
    gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);

    shaderProgram.textureCoordAttribute =
        gl.getAttribLocation(shaderProgram, "aTextureCoord");
    gl.enableVertexAttribArray(shaderProgram.textureCoordAttribute);

    shaderProgram.pMatrixUniform =
        gl.getUniformLocation(shaderProgram, "uPMatrix");
    shaderProgram.mvMatrixUniform =
        gl.getUniformLocation(shaderProgram, "uMVMatrix");
    shaderProgram.samplerUniform =
        gl.getUniformLocation(shaderProgram, "uSampler");
}


var mvMatrix = mat4.create();

var mvMatrixStack = [];

var pMatrix = mat4.create();

function mvPushMatrix()
{
```

```
    var copy = mat4.clone(mvMatrix);
    mvMatrixStack.push(copy);
}
function mvPopMatrix()
{
    if(mvMatrixStack.length == 0)
    {
        throw "不合法的矩阵出栈操作!";
    }
    mvMatrix = mvMatrixStack.pop();
}

function setMatrixUniforms()
{
    gl.uniformMatrix4fv(shaderProgram.pMatrixUniform, false, pMatrix);
    gl.uniformMatrix4fv(shaderProgram.mvMatrixUniform, false, mvMatrix);
}


var starVertexPositionBuffer;
var starVertexTextureCoordBuffer;

function initBuffers()
{
    starVertexPositionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, starVertexPositionBuffer);
    vertices = [
        -1.0, -1.0,    0.0,
         1.0, -1.0,    0.0,
        -1.0,    1.0,    0.0,
         1.0,    1.0,    0.0
        ];
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices),
        gl.STATIC_DRAW);
    starVertexPositionBuffer.itemSize = 3;
    starVertexPositionBuffer.numItems = 4;

    starVertexTextureCoordBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, starVertexTextureCoordBuffer);
    var textureCoords = [
        0.0, 0.0,
        1.0, 0.0,
        0.0, 1.0,
        1.0, 1.0
        ];
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(textureCoords),
        gl.STATIC_DRAW);
    starVertexTextureCoordBuffer.itemSize = 2;
    starVertexTextureCoordBuffer.numItems = 4;
}


var currentlyPressedKeys = {};
function handleKeyDown(event)
{
    currentlyPressedKeys[event.keyCode] = true;
}
function handleKeyUp(event)
{
```

```
    currentlyPressedKeys[event.keyCode] = false;
}

var pitch = 0;
var pitchRate = 0;

var yaw = 0;
var yawRate = 0;
var xPos = 0;
var yPos = 0.4;
var zPos = 0;

var speed = 0;
function handleKeys()
{
    if(currentlyPressedKeys[188])
    {
        //","/"<"逗号键
        pitchRate = -0.1;
    }
    else if(currentlyPressedKeys[190])
    {
        //"."/">"句号键
        pitchRate = 0.1;
    }
    else
    {
        pitchRate = 0;
    }
    if (currentlyPressedKeys[65])
    {
        //A
        yawRate = 0.1;
    }
    else if (currentlyPressedKeys[68])
    {
        //D
        yawRate = -0.1;
    }
    else
    {
        yawRate = 0;
    }
    if(currentlyPressedKeys[87])
    {
        //W
        speed = 0.003;
    }
    else if(currentlyPressedKeys[83])
    {
        //S
        speed = -0.003;
    }
    else
    {
        speed = 0;
    }
}
function drawScene()
```

```
{
    gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    if(worldVertexTextureCoordBuffer == null || worldVertexPositionBuffer == null)
    {
        return;
    }
    mat4.perspective(pMatrix, 45,
        gl.viewportWidth / gl.viewportHeight, 0.1, 100.0);

    mat4.identity(mvMatrix);

    mat4.rotate(mvMatrix, mvMatrix, degToRad(-pitch), [1, 0, 0]);
    mat4.rotate(mvMatrix, mvMatrix, degToRad(-yaw), [0, 1, 0]);
    mat4.translate(mvMatrix, mvMatrix, [-xPos, -yPos, -zPos]);

    gl.activeTexture(gl.TEXTURE0);
    gl.bindTexture(gl.TEXTURE_2D, crackTexture);
    gl.uniform1i(shaderProgram.samplerUniform, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, worldVertexTextureCoordBuffer);
    gl.vertexAttribPointer(shaderProgram.textureCoordAttribute,
        worldVertexTextureCoordBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, worldVertexPositionBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
        worldVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);

    setMatrixUniforms();
    gl.drawArrays(gl.TRIANGLES, 0, worldVertexPositionBuffer.numItems);
}

function degToRad(degrees)
{
    return degrees * Math.PI / 180;
}
var lastTime = 0;
var joggingAngle = 0;
function animate()
{
    var timeNow = new Date().getTime();
    if(lastTime != 0)
    {
        var elapsed = timeNow - lastTime;
        if(speed != 0)
        {
            xPos -= Math.sin(degToRad(yaw)) * speed * elapsed;
            zPos -= Math.cos(degToRad(yaw)) * speed * elapsed;
            joggingAngle += elapsed * 0.6;
            yPos = Math.sin(degToRad(joggingAngle)) / 20 + 0.4;
        }
        yaw += yawRate * elapsed;
        pitch += pitchRate * elapsed;
    }
    lastTime = timeNow;
}
</script>
```

球、旋转矩阵、鼠标事件

```
<div class="page-header"><h3>11、球、旋转矩阵、鼠标事件</h3></div>

<canvas id = "test11-canvas" width = "800" height = "600"></canvas>

<br/>
<input type="checkbox" id="lighting" checked /> 使用光照
<br/>
(鼠标拖拽对月球进行转动)
<br/>
<h4>方向光:</h4>
<table>
    <tr>
        <td><b>方向:</b></td>
        <td>X: <input type="text" id="lightDirectionX" value="-0.25" /></td>
        <td>Y: <input type="text" id="lightDirectionY" value="-0.25" /></td>
        <td>Z: <input type="text" id="lightDirectionZ" value="-1.0" /></td>
    </tr>
    <tr>
        <td><b>颜色:</b></td>
        <td>R: <input type="text" id="directionalR" value="0.8" /></td>
        <td>G: <input type="text" id="directionalG" value="0.8" /></td>
        <td>B: <input type="text" id="directionalB" value="0.8" /></td>
    </tr>
</table>
<h4>环境光:</h4>
<table style="border: 0; padding: 10px;">
    <tr>
        <td><b>颜色:</b></td>
        <td>R: <input type="text" id="ambientR" value="0.2" /></td>
        <td>G: <input type="text" id="ambientG" value="0.2" /></td>
        <td>B: <input type="text" id="ambientB" value="0.2" /></td>
    </tr>
</table>
<br/>
月球表面纹理图片来自:
<a href="http://maps.jpl.nasa.gov/">the Jet Propulsion Laboratory</a>.
<script id = "shader-vs" type = "x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    attribute vec3 aVertexNormal;
    attribute vec2 aTextureCoord;

    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;
    uniform mat3 uNMatrix;

    uniform vec3 uAmbientColor;

    uniform vec3 uLightingDirection;
    uniform vec3 uDirectionalColor;

    uniform bool uUseLighting;

    varying vec2 vTextureCoord;
    varying vec3 vLightWeighting;
```

```
    void main(void)
    {
        gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
        vTextureCoord = aTextureCoord;

        if(!uUseLighting)
        {
            vLightWeighting = vec3(1.0, 1.0, 1.0);
        }
        else
        {
            vec3 transformedNormal = uNMatrix * aVertexNormal;
            float directionalLightWeighting =
                max(dot(transformedNormal, uLightingDirection), 0.0);
            vLightWeighting =
                uAmbientColor + uDirectionalColor * directionalLightWeighting;
        }
    }
</script>

<script id = "shader-fs" type = "x-shader/x-fragment">
    precision mediump float;
    varying vec2 vTextureCoord;
    varying vec3 vLightWeighting;

    uniform sampler2D uSampler;
    void main(void)
    {
        vec4 textureColor =
            texture2D(uSampler, vec2(vTextureCoord.s, vTextureCoord.t));
        gl_FragColor =
            vec4(textureColor.rgb * vLightWeighting, textureColor.a);
    }
</script>

<script type="text/javascript">
    requestAnimFrame = window.requestAnimationFrame ||
        window.mozRequestAnimationFrame ||
        window.webkitRequestAnimationFrame ||
        window.msRequestAnimationFrame ||
        window.oRequestAnimationFrame ||
        function(callback) { setTimeout(callback, 1000 / 60); };
</script>
<script type="text/javascript">

$(document).ready(function ()
{
    webGLStart();
});

function webGLStart()
{
    var canvas = $("#test11-canvas");
    initGL(canvas[0]);
    initShaders();
    initBuffers();
    initTexture();
```

```
    gl.clearColor(0.0, 0.0, 0.0, 1.0);
    gl.enable(gl.DEPTH_TEST);

    canvas.mousedown(handleMouseDown);
    $(document).mouseup(handleMouseUp);
    $(document).mousemove(handleMouseMove);
}
function tick()
{
    requestAnimFrame(tick);
    drawScene();
}
var gl;
function initGL(canvas)
{
    try
    {
        gl = canvas.getContext("webgl") || canvas.getContext("experimental-webgl");
        gl.viewportWidth = canvas.width;
        gl.viewportHeight = canvas.height;
    }catch(e){}
    if(!gl)
    {
        alert("无法初始化“WebGL”。");
    }
}

var moonTexture;
function initTexture()
{
    moonTexture = gl.createTexture();
    moonTexture.image = new Image();
    moonTexture.image.onload = function()
    {
        handleLoadedTexture(moonTexture);
        tick();
    }
    moonTexture.image.src = "/Public/image/moon.gif";
}
function handleLoadedTexture(texture)
{
    gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, 1);


    gl.bindTexture(gl.TEXTURE_2D, texture);
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA,
        gl.UNSIGNED_BYTE, texture.image);
    gl.texParameteri(gl.TEXTURE_2D,
        gl.TEXTURE_MAG_FILTER, gl.LINEAR);
    gl.texParameteri(gl.TEXTURE_2D,
        gl.TEXTURE_MIN_FILTER, gl.LINEAR_MIPMAP_NEAREST);

    gl.generateMipmap(gl.TEXTURE_2D);

    gl.bindTexture(gl.TEXTURE_2D, null);
}
function getShader(gl, id)
{
    var shaderScript = $("#" + id);
```

```
    if(!shaderScript.length)
    {
        return null;
    }
    var str = shaderScript.text();

    var shader;
    if(shaderScript[0].type == "x-shader/x-fragment")
    {
        shader = gl.createShader(gl.FRAGMENT_SHADER);
    }
    else if(shaderScript[0].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;
}

var shaderProgram;
function initShaders()
{
    var fragmentShader = getShader(gl, "shader-fs");
    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("无法初始化“Shader”。");
    }
    gl.useProgram(shaderProgram);

    shaderProgram.vertexPositionAttribute =
        gl.getAttribLocation(shaderProgram, "aVertexPosition");
    gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);

    shaderProgram.vertexNormalAttribute =
        gl.getAttribLocation(shaderProgram, "aVertexNormal");
    gl.enableVertexAttribArray(shaderProgram.vertexNormalAttribute);

    shaderProgram.textureCoordAttribute =
        gl.getAttribLocation(shaderProgram, "aTextureCoord");
    gl.enableVertexAttribArray(shaderProgram.textureCoordAttribute);

    shaderProgram.pMatrixUniform =
```

```
        gl.getUniformLocation(shaderProgram, "uPMatrix");
    shaderProgram.mvMatrixUniform =
        gl.getUniformLocation(shaderProgram, "uMVMatrix");
    shaderProgram.nMatrixUniform  =
        gl.getUniformLocation(shaderProgram, "uNMatrix");
    shaderProgram.samplerUniform =
        gl.getUniformLocation(shaderProgram, "uSampler");
    shaderProgram.useLightingUniform =
        gl.getUniformLocation(shaderProgram, "uUseLighting");
    shaderProgram.ambientColorUniform =
        gl.getUniformLocation(shaderProgram, "uAmbientColor");
    shaderProgram.lightingDirectionUniform =
        gl.getUniformLocation(shaderProgram, "uLightingDirection");
    shaderProgram.directionalColorUniform =
        gl.getUniformLocation(shaderProgram, "uDirectionalColor");
}


var mvMatrix = mat4.create();

var mvMatrixStack = [];

var pMatrix = mat4.create();

function mvPushMatrix()
{
    var copy = mat4.clone(mvMatrix);
    mvMatrixStack.push(copy);
}
function mvPopMatrix()
{
    if(mvMatrixStack.length == 0)
    {
        throw "不合法的矩阵出栈操作!";
    }
    mvMatrix = mvMatrixStack.pop();
}

function setMatrixUniforms()
{
    gl.uniformMatrix4fv(shaderProgram.pMatrixUniform, false, pMatrix);
    gl.uniformMatrix4fv(shaderProgram.mvMatrixUniform, false, mvMatrix);

    var normalMatrix = mat3.create();

    mat3.fromMat4(normalMatrix, mvMatrix);
    mat3.invert(normalMatrix, normalMatrix);
    mat3.transpose(normalMatrix, normalMatrix);

    gl.uniformMatrix3fv(shaderProgram.nMatrixUniform, false, normalMatrix);
}

var mouseDown = false;
var lastMouseX = null;
var lastMouseY = null;
var moonRotationMatrix = mat4.create();
mat4.identity(moonRotationMatrix);
function handleMouseDown(event)
{
```

```
    mouseDown = true;
    lastMouseX = event.clientX;
    lastMouseY = event.clientY;
}
function handleMouseUp(event)
{
    mouseDown = false;
}
function handleMouseMove(event)
{
    if(!mouseDown)
    {
        return;
    }
    var newX = event.clientX;
    var newY = event.clientY;
    var newRotationMatrix = mat4.create();
    mat4.identity(newRotationMatrix);

    var deltaX = newX - lastMouseX;
    mat4.rotate(newRotationMatrix, newRotationMatrix,
        degToRad(deltaX / 10), [0, 1, 0]);
    var deltaY = newY - lastMouseY;
    mat4.rotate(newRotationMatrix, newRotationMatrix,
        degToRad(deltaY / 10), [1, 0, 0]);

    mat4.multiply(moonRotationMatrix, newRotationMatrix, moonRotationMatrix);

    lastMouseX = newX;
    lastMouseY = newY;
}
function equal(a, b)
{
    for(var i = 0; i < a.length; i ++)
        if(a[i] != b[i]) return false;
    return true;
}
var moonVertexPositionBuffer;
var moonVertexNormalBuffer;
var moonVertexTextureCoordBuffer;
var moonVertexIndexBuffer;

function initBuffers()
{
    var latitudeBands = 30;
    var longitudeBands = 30;
    var radius = 2;
    var vertexPositionData = [];
    var normalData = [];
    var textureCoordData = [];
    for(var latNumber = 0; latNumber <= latitudeBands; latNumber ++)
    {
        var theta = latNumber * Math.PI / latitudeBands;
        var sinTheta = Math.sin(theta);
        var cosTheta = Math.cos(theta);
        for(var longNumber = 0; longNumber <= longitudeBands; longNumber ++)
        {
            var phi = longNumber * 2 * Math.PI / longitudeBands;
            var sinPhi = Math.sin(phi);
```

```
            var cosPhi = Math.cos(phi);
            var x = cosPhi * sinTheta;
            var y = cosTheta;
            var z = sinPhi * sinTheta;
            var u = 1 - (longNumber / longitudeBands);
            var v = 1 - (latNumber / latitudeBands);

            normalData.push(x);
            normalData.push(y);
            normalData.push(z);
            textureCoordData.push(u);
            textureCoordData.push(v);
            vertexPositionData.push(radius * x);
            vertexPositionData.push(radius * y);
            vertexPositionData.push(radius * z);
        }
    }
    var indexData = [];
    for(var latNumber = 0; latNumber < latitudeBands; latNumber ++)
    {
        for(var longNumber = 0; longNumber < longitudeBands; longNumber ++)
        {
            var first = (latNumber * (longitudeBands + 1)) + longNumber;
            var second = first + longitudeBands + 1;
            indexData.push(first);
            indexData.push(second);
            indexData.push(first + 1);
            indexData.push(second);
            indexData.push(second + 1);
            indexData.push(first + 1);
        }
    }
    moonVertexNormalBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, moonVertexNormalBuffer);
    gl.bufferData(gl.ARRAY_BUFFER,
        new Float32Array(normalData), gl.STATIC_DRAW);
    moonVertexNormalBuffer.itemSize = 3;
    moonVertexNormalBuffer.numItems = normalData.length / 3;

    moonVertexTextureCoordBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, moonVertexTextureCoordBuffer);
    gl.bufferData(gl.ARRAY_BUFFER,
        new Float32Array(textureCoordData), gl.STATIC_DRAW);
    moonVertexTextureCoordBuffer.itemSize = 2;
    moonVertexTextureCoordBuffer.numItems = textureCoordData.length / 2;

    moonVertexPositionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, moonVertexPositionBuffer);
    gl.bufferData(gl.ARRAY_BUFFER,
        new Float32Array(vertexPositionData), gl.STATIC_DRAW);
    moonVertexPositionBuffer.itemSize = 3;
    moonVertexPositionBuffer.numItems = vertexPositionData.length / 3;

    moonVertexIndexBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, moonVertexIndexBuffer);
    gl.bufferData(gl.ELEMENT_ARRAY_BUFFER,
        new Uint16Array(indexData), gl.STATIC_DRAW);
    moonVertexIndexBuffer.itemSize = 1;
    moonVertexIndexBuffer.numItems = indexData.length;
```

```
}
function drawScene()
{
    gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    mat4.perspective(pMatrix, 45,
        gl.viewportWidth / gl.viewportHeight, 0.1, 100.0);

    var lighting = $("#lighting").is(":checked");
    gl.uniform1i(shaderProgram.useLightingUniform, lighting);
    if(lighting)
    {
        gl.uniform3f(
            shaderProgram.ambientColorUniform,
            parseFloat($("#ambientR").val()),
            parseFloat($("#ambientG").val()),
            parseFloat($("#ambientB").val())
            );
        var lightingDirection = [
            parseFloat($("#lightDirectionX").val()),
            parseFloat($("#lightDirectionY").val()),
            parseFloat($("#lightDirectionZ").val())
            ];
        var adjustedLD = vec3.create();
        vec3.normalize(adjustedLD, lightingDirection);
        vec3.scale(adjustedLD, adjustedLD, -1);
        gl.uniform3fv(shaderProgram.lightingDirectionUniform, adjustedLD);
        gl.uniform3f(
            shaderProgram.directionalColorUniform,
            parseFloat($("#directionalR").val()),
            parseFloat($("#directionalG").val()),
            parseFloat($("#directionalB").val())
            );
    }

    mat4.identity(mvMatrix);
    mat4.translate(mvMatrix, mvMatrix, [0, 0, -6]);

    mat4.multiply(mvMatrix, mvMatrix, moonRotationMatrix);

    gl.activeTexture(gl.TEXTURE0);
    gl.bindTexture(gl.TEXTURE_2D, moonTexture);
    gl.uniform1i(shaderProgram.samplerUniform, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, moonVertexTextureCoordBuffer);
    gl.vertexAttribPointer(shaderProgram.textureCoordAttribute,
        moonVertexTextureCoordBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, moonVertexPositionBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
        moonVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, moonVertexNormalBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexNormalAttribute,
        moonVertexNormalBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, moonVertexIndexBuffer);
    setMatrixUniforms();
```

```
    gl.drawElements(gl.TRIANGLES,
        moonVertexIndexBuffer.numItems, gl.UNSIGNED_SHORT, 0);
}

function degToRad(degrees)
{
    return degrees * Math.PI / 180;
}
</script>
```

点光源

```
<div class="page-header"><h3>12、点光源</h3></div>

<canvas id = "test12-canvas" width = "800" height = "600"></canvas>

<br/>
<input type="checkbox" id="lighting" checked /> 使用光照
<br/>
<h4>点光:</h4>
<table>
    <tr>
        <td><b>位置:</b></td>
        <td>X: <input type="text" id="lightPositionX" value="0.0" /></td>
        <td>Y: <input type="text" id="lightPositionY" value="0.0" /></td>
        <td>Z: <input type="text" id="lightPositionZ" value="-20.0" /></td>
    </tr>
    <tr>
        <td><b>颜色:</b></td>
        <td>R: <input type="text" id="pointR" value="0.8" /></td>
        <td>G: <input type="text" id="pointG" value="0.8" /></td>
        <td>B: <input type="text" id="pointB" value="0.8" /></td>
    </tr>
</table>
<h4>环境光:</h4>
<table style="border: 0; padding: 10px;">
    <tr>
        <td><b>颜色:</b></td>
        <td>R: <input type="text" id="ambientR" value="0.2" /></td>
        <td>G: <input type="text" id="ambientG" value="0.2" /></td>
        <td>B: <input type="text" id="ambientB" value="0.2" /></td>
    </tr>
</table>
<br/>
月球表面纹理图片来自:
<a href="http://maps.jpl.nasa.gov/">the Jet Propulsion Laboratory</a>.
<script id = "shader-vs" type = "x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    attribute vec3 aVertexNormal;
    attribute vec2 aTextureCoord;

    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;
    uniform mat3 uNMatrix;

    uniform vec3 uAmbientColor;

    uniform vec3 uPointLightingLocation;
    uniform vec3 uPointLightingColor;

    uniform bool uUseLighting;

    varying vec2 vTextureCoord;
    varying vec3 vLightWeighting;

    void main(void)
```

```
    {
        vec4 mvPosition = uMVMatrix * vec4(aVertexPosition, 1.0);
        gl_Position = uPMatrix * mvPosition;
        vTextureCoord = aTextureCoord;

        if (!uUseLighting)
        {
            vLightWeighting = vec3(1.0, 1.0, 1.0);
        }
        else
        {
            vec3 lightDirection =
                normalize(uPointLightingLocation - mvPosition.xyz);
            vec3 transformedNormal = uNMatrix * aVertexNormal;
            float directionalLightWeighting =
                max(dot(transformedNormal, lightDirection), 0.0);
            vLightWeighting =
                uAmbientColor + uPointLightingColor * directionalLightWeighting;
        }
    }
</script>

<script id = "shader-fs" type = "x-shader/x-fragment">
    precision mediump float;

    varying vec2 vTextureCoord;
    varying vec3 vLightWeighting;

    uniform sampler2D uSampler;

    void main(void)
    {
        vec4 textureColor =
            texture2D(uSampler, vec2(vTextureCoord.s, vTextureCoord.t));
        gl_FragColor =
            vec4(textureColor.rgb * vLightWeighting, textureColor.a);
    }
</script>

<script type="text/javascript">
    requestAnimFrame = window.requestAnimationFrame ||
        window.mozRequestAnimationFrame ||
        window.webkitRequestAnimationFrame ||
        window.msRequestAnimationFrame ||
        window.oRequestAnimationFrame ||
        function(callback) { setTimeout(callback, 1000 / 60); };
</script>
<script type="text/javascript">
$(document).ready(function ()
{
    webGLStart();
});

function webGLStart()
{
    var canvas = $("#test12-canvas");
    initGL(canvas[0]);
    initShaders();
    initBuffers();
```

```
    initTexture();

    gl.clearColor(0.0, 0.0, 0.0, 1.0);
    gl.enable(gl.DEPTH_TEST);

    waitTexture = setInterval("tick()", 100);
}

var textureFlag = [0, 0];
var startTick = false;
var waitTick;
function tick()
{
    if(textureFlag[0] == 1 && textureFlag[1] == 1)
    {
        startTick = true;
        textureFlag[0] = 2;
        textureFlag[1] = 2;
        clearInterval(waitTexture);
    }
    if(startTick)
    {
        requestAnimFrame(tick);
        drawScene();
        animate();
    }
}
var gl;
function initGL(canvas)
{
    try
    {
        gl = canvas.getContext("webgl") || canvas.getContext("experimental-webgl");
        gl.viewportWidth = canvas.width;
        gl.viewportHeight = canvas.height;
    }catch(e){}
    if(!gl)
    {
        alert("无法初始化“WebGL”。");
    }
}

var moonTexture;
var crateTexture;
function initTexture()
{
    moonTexture = gl.createTexture();
    moonTexture.image = new Image();
    moonTexture.image.onload = function()
    {
        handleLoadedTexture(moonTexture);
        textureFlag[0] = true;
    }
    moonTexture.image.src = "/Public/image/moon.gif";


    crateTexture = gl.createTexture();
    crateTexture.image = new Image();
    crateTexture.image.onload = function()
```

```
    {
        handleLoadedTexture(crateTexture);
        textureFlag[1] = true;
    }
    crateTexture.image.src = "/Public/image/crate.gif";
}
function handleLoadedTexture(texture)
{
    gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, 1);


    gl.bindTexture(gl.TEXTURE_2D, texture);
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA,
        gl.UNSIGNED_BYTE, texture.image);
    gl.texParameteri(gl.TEXTURE_2D,
        gl.TEXTURE_MAG_FILTER, gl.LINEAR);
    gl.texParameteri(gl.TEXTURE_2D,
        gl.TEXTURE_MIN_FILTER, gl.LINEAR_MIPMAP_NEAREST);

    gl.generateMipmap(gl.TEXTURE_2D);

    gl.bindTexture(gl.TEXTURE_2D, null);
}
function getShader(gl, id)
{
    var shaderScript = $("#" + id);
    if(!shaderScript.length)
    {
        return null;
    }
    var str = shaderScript.text();

    var shader;
    if(shaderScript[0].type == "x-shader/x-fragment")
    {
        shader = gl.createShader(gl.FRAGMENT_SHADER);
    }
    else if(shaderScript[0].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;
}

var shaderProgram;
function initShaders()
{
    var fragmentShader = getShader(gl, "shader-fs");
```

```
    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("无法初始化“Shader”。");
    }
    gl.useProgram(shaderProgram);

    shaderProgram.vertexPositionAttribute =
        gl.getAttribLocation(shaderProgram, "aVertexPosition");
    gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);

    shaderProgram.vertexNormalAttribute =
        gl.getAttribLocation(shaderProgram, "aVertexNormal");
    gl.enableVertexAttribArray(shaderProgram.vertexNormalAttribute);

    shaderProgram.textureCoordAttribute =
        gl.getAttribLocation(shaderProgram, "aTextureCoord");
    gl.enableVertexAttribArray(shaderProgram.textureCoordAttribute);

    shaderProgram.pMatrixUniform =
        gl.getUniformLocation(shaderProgram, "uPMatrix");
    shaderProgram.mvMatrixUniform =
        gl.getUniformLocation(shaderProgram, "uMVMatrix");
    shaderProgram.nMatrixUniform  =
        gl.getUniformLocation(shaderProgram, "uNMatrix");
    shaderProgram.samplerUniform =
        gl.getUniformLocation(shaderProgram, "uSampler");
    shaderProgram.useLightingUniform =
        gl.getUniformLocation(shaderProgram, "uUseLighting");
    shaderProgram.ambientColorUniform =
        gl.getUniformLocation(shaderProgram, "uAmbientColor");
    shaderProgram.pointLightingLocationUniform  =
        gl.getUniformLocation(shaderProgram, "uPointLightingLocation");
    shaderProgram.pointLightingColorUniform  =
        gl.getUniformLocation(shaderProgram, "uPointLightingColor");
}


var mvMatrix = mat4.create();

var mvMatrixStack = [];

var pMatrix = mat4.create();

function mvPushMatrix()
{
    var copy = mat4.clone(mvMatrix);
    mvMatrixStack.push(copy);
}
function mvPopMatrix()
{
    if(mvMatrixStack.length == 0)
    {
```

```
            throw "不合法的矩阵出栈操作!";
        }
        mvMatrix = mvMatrixStack.pop();
    }

    function setMatrixUniforms()
    {
        gl.uniformMatrix4fv(shaderProgram.pMatrixUniform, false, pMatrix);
        gl.uniformMatrix4fv(shaderProgram.mvMatrixUniform, false, mvMatrix);

        var normalMatrix = mat3.create();

        mat3.fromMat4(normalMatrix, mvMatrix);
        mat3.invert(normalMatrix, normalMatrix);
        mat3.transpose(normalMatrix, normalMatrix);

        gl.uniformMatrix3fv(shaderProgram.nMatrixUniform, false, normalMatrix);
    }

    var cubeVertexPositionBUffer;
    var cubeVertexTextureCoordBuffer;
    var cubeVertexIndexBuffer;
    var cubeVertexNormalBuffer;

    var moonVertexPositionBuffer;
    var moonVertexNormalBuffer;
    var moonVertexTextureCoordBuffer;
    var moonVertexIndexBuffer;

    function initBuffers()
    {
        //箱子
        cubeVertexPositionBuffer = gl.createBuffer();
        gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
        vertices = [
                    // 正面
                    -1.0, -1.0,  1.0,
                     1.0, -1.0,  1.0,
                     1.0,  1.0,  1.0,
                    -1.0,  1.0,  1.0,

                    // 背面
                    -1.0, -1.0, -1.0,
                    -1.0,  1.0, -1.0,
                     1.0,  1.0, -1.0,
                     1.0, -1.0, -1.0,

                    // 顶部
                    -1.0,  1.0, -1.0,
                    -1.0,  1.0,  1.0,
                     1.0,  1.0,  1.0,
                     1.0,  1.0, -1.0,

                    // 底部
                    -1.0, -1.0, -1.0,
                     1.0, -1.0, -1.0,
                     1.0, -1.0,  1.0,
                    -1.0, -1.0,  1.0,
```

```
                // 右侧面
                 1.0, -1.0, -1.0,
                 1.0,  1.0, -1.0,
                 1.0,  1.0,  1.0,
                 1.0, -1.0,  1.0,

                // 左侧面
                -1.0, -1.0, -1.0,
                -1.0, -1.0,  1.0,
                -1.0,  1.0,  1.0,
                -1.0,  1.0, -1.0,
                ];
    gl.bufferData(gl.ARRAY_BUFFER,
        new Float32Array(vertices), gl.STATIC_DRAW);
    cubeVertexPositionBuffer.itemSize = 3;
    cubeVertexPositionBuffer.numItems = 24;

    cubeVertexNormalBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexNormalBuffer);
    var vertexNormals = [
                    // 正面
                      0.0,  0.0,  1.0,
                      0.0,  0.0,  1.0,
                      0.0,  0.0,  1.0,
                      0.0,  0.0,  1.0,

                    // 背面
                      0.0,  0.0, -1.0,
                      0.0,  0.0, -1.0,
                      0.0,  0.0, -1.0,
                      0.0,  0.0, -1.0,

                    // 顶部
                      0.0,  1.0,  0.0,
                      0.0,  1.0,  0.0,
                      0.0,  1.0,  0.0,
                      0.0,  1.0,  0.0,

                    // 底部
                      0.0, -1.0,  0.0,
                      0.0, -1.0,  0.0,
                      0.0, -1.0,  0.0,
                      0.0, -1.0,  0.0,

                    // 右侧面
                      1.0,  0.0,  0.0,
                      1.0,  0.0,  0.0,
                      1.0,  0.0,  0.0,
                      1.0,  0.0,  0.0,

                    // 左侧面
                    -1.0,  0.0,  0.0,
                    -1.0,  0.0,  0.0,
                    -1.0,  0.0,  0.0,
                    -1.0,  0.0,  0.0,
                  ];
    gl.bufferData(gl.ARRAY_BUFFER,
        new Float32Array(vertexNormals), gl.STATIC_DRAW);
    cubeVertexNormalBuffer.itemSize = 3;
```

```
    cubeVertexNormalBuffer.numItems = 24;

    cubeVertexTextureCoordBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexTextureCoordBuffer);
    textureCoords = [
                    // 正面
                    0.0, 0.0,
                    1.0, 0.0,
                    1.0, 1.0,
                    0.0, 1.0,

                    // 背面
                    1.0, 0.0,
                    1.0, 1.0,
                    0.0, 1.0,
                    0.0, 0.0,

                    // 顶部
                    0.0, 1.0,
                    0.0, 0.0,
                    1.0, 0.0,
                    1.0, 1.0,

                    // 底部
                    1.0, 1.0,
                    0.0, 1.0,
                    0.0, 0.0,
                    1.0, 0.0,

                    // 右侧面
                    1.0, 0.0,
                    1.0, 1.0,
                    0.0, 1.0,
                    0.0, 0.0,

                    // 左侧面
                    0.0, 0.0,
                    1.0, 0.0,
                    1.0, 1.0,
                    0.0, 1.0,
                   ];
    gl.bufferData(gl.ARRAY_BUFFER,
        new Float32Array(textureCoords), gl.STATIC_DRAW);
    cubeVertexTextureCoordBuffer.itemSize = 2;
    cubeVertexTextureCoordBuffer.numItems = 24;

    cubeVertexIndexBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);
    var cubeVertexIndices =
        [
          0, 1, 2,        0, 2, 3,    // 正面
          4, 5, 6,        4, 6, 7,    // 背面
          8, 9, 10,        8, 10, 11,  // 顶部
          12, 13, 14,    12, 14, 15, // 底部
          16, 17, 18,    16, 18, 19, // 右侧面
          20, 21, 22,    20, 22, 23  // 左侧面
        ];
    gl.bufferData(gl.ELEMENT_ARRAY_BUFFER,
        new Uint16Array(cubeVertexIndices), gl.STATIC_DRAW);
```

```
    cubeVertexIndexBuffer.itemSize = 1;
    cubeVertexIndexBuffer.numItems = 36;

    //月亮
    var latitudeBands = 30;
    var longitudeBands = 30;
    var radius = 2;
    var vertexPositionData = [];
    var normalData = [];
    var textureCoordData = [];
    for(var latNumber = 0; latNumber <= latitudeBands; latNumber ++)
    {
        var theta = latNumber * Math.PI / latitudeBands;
        var sinTheta = Math.sin(theta);
        var cosTheta = Math.cos(theta);
        for(var longNumber = 0; longNumber <= longitudeBands; longNumber ++)
        {
            var phi = longNumber * 2 * Math.PI / longitudeBands;
            var sinPhi = Math.sin(phi);
            var cosPhi = Math.cos(phi);
            var x = cosPhi * sinTheta;
            var y = cosTheta;
            var z = sinPhi * sinTheta;
            var u = 1 - (longNumber / longitudeBands);
            var v = 1 - (latNumber / latitudeBands);

            normalData.push(x);
            normalData.push(y);
            normalData.push(z);
            textureCoordData.push(u);
            textureCoordData.push(v);
            vertexPositionData.push(radius * x);
            vertexPositionData.push(radius * y);
            vertexPositionData.push(radius * z);
        }
    }
    var indexData = [];
    for(var latNumber = 0; latNumber < latitudeBands; latNumber ++)
    {
        for(var longNumber = 0; longNumber < longitudeBands; longNumber ++)
        {
            var first = (latNumber * (longitudeBands + 1)) + longNumber;
            var second = first + longitudeBands + 1;
            indexData.push(first);
            indexData.push(second);
            indexData.push(first + 1);
            indexData.push(second);
            indexData.push(second + 1);
            indexData.push(first + 1);
        }
    }
    moonVertexNormalBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, moonVertexNormalBuffer);
    gl.bufferData(gl.ARRAY_BUFFER,
        new Float32Array(normalData), gl.STATIC_DRAW);
    moonVertexNormalBuffer.itemSize = 3;
    moonVertexNormalBuffer.numItems = normalData.length / 3;

    moonVertexTextureCoordBuffer = gl.createBuffer();
```

```
    gl.bindBuffer(gl.ARRAY_BUFFER, moonVertexTextureCoordBuffer);
    gl.bufferData(gl.ARRAY_BUFFER,
        new Float32Array(textureCoordData), gl.STATIC_DRAW);
    moonVertexTextureCoordBuffer.itemSize = 2;
    moonVertexTextureCoordBuffer.numItems = textureCoordData.length / 2;

    moonVertexPositionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, moonVertexPositionBuffer);
    gl.bufferData(gl.ARRAY_BUFFER,
        new Float32Array(vertexPositionData), gl.STATIC_DRAW);
    moonVertexPositionBuffer.itemSize = 3;
    moonVertexPositionBuffer.numItems = vertexPositionData.length / 3;

    moonVertexIndexBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, moonVertexIndexBuffer);
    gl.bufferData(gl.ELEMENT_ARRAY_BUFFER,
        new Uint16Array(indexData), gl.STATIC_DRAW);
    moonVertexIndexBuffer.itemSize = 1;
    moonVertexIndexBuffer.numItems = indexData.length;
}

var moonAngle = 180;
var cubeAngle = 0;

function drawScene()
{
    gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    mat4.perspective(pMatrix, 45,
        gl.viewportWidth / gl.viewportHeight, 0.1, 100.0);

    var lighting = $("#lighting").is(":checked");
    gl.uniform1i(shaderProgram.useLightingUniform, lighting);
    if(lighting)
    {
        gl.uniform3f(
            shaderProgram.ambientColorUniform,
            parseFloat($("#ambientR").val()),
            parseFloat($("#ambientG").val()),
            parseFloat($("#ambientB").val())
            );
        gl.uniform3f(
            shaderProgram.pointLightingLocationUniform,
            parseFloat($("#lightPositionX").val()),
            parseFloat($("#lightPositionY").val()),
            parseFloat($("#lightPositionZ").val())
        );

        gl.uniform3f(
            shaderProgram.pointLightingColorUniform,
            parseFloat($("#pointR").val()),
            parseFloat($("#pointG").val()),
            parseFloat($("#pointB").val())
        );

    }

    mat4.identity(mvMatrix);
```

```
    mat4.translate(mvMatrix, mvMatrix, [0, 0, -20]);

    mvPushMatrix();

    mat4.rotate(mvMatrix, mvMatrix, degToRad(moonAngle), [0, 1, 0]);
    mat4.translate(mvMatrix, mvMatrix, [5, 0, 0]);

    gl.activeTexture(gl.TEXTURE0);
    gl.bindTexture(gl.TEXTURE_2D, moonTexture);
    gl.uniform1i(shaderProgram.samplerUniform, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, moonVertexTextureCoordBuffer);
    gl.vertexAttribPointer(shaderProgram.textureCoordAttribute,
        moonVertexTextureCoordBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, moonVertexPositionBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
        moonVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, moonVertexNormalBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexNormalAttribute,
        moonVertexNormalBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, moonVertexIndexBuffer);
    setMatrixUniforms();
    gl.drawElements(gl.TRIANGLES,
        moonVertexIndexBuffer.numItems, gl.UNSIGNED_SHORT, 0);

    mvPopMatrix();

    mvPushMatrix();
    mat4.rotate(mvMatrix, mvMatrix, degToRad(cubeAngle), [0, 1, 0]);
    mat4.translate(mvMatrix, mvMatrix, [5, 0, 0]);

    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
        cubeVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexNormalBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexNormalAttribute,
        cubeVertexNormalBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexTextureCoordBuffer);
    gl.vertexAttribPointer(shaderProgram.textureCoordAttribute,
        cubeVertexTextureCoordBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.activeTexture(gl.TEXTURE0);
    gl.bindTexture(gl.TEXTURE_2D, crateTexture);
    gl.uniform1i(shaderProgram.samplerUniform, 0);

    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);
    setMatrixUniforms();
    gl.drawElements(gl.TRIANGLES,
        cubeVertexIndexBuffer.numItems, gl.UNSIGNED_SHORT, 0);
    mvPopMatrix();
}

function degToRad(degrees)
{
```

```
    return degrees * Math.PI / 180;
}
var lastTime = 0;
function animate()
{
    var timeNow = new Date().getTime();
    if(lastTime != 0)
    {
        var elapsed = timeNow - lastTime;

         moonAngle += 0.05 * elapsed;
         cubeAngle += 0.05 * elapsed;
    }
    lastTime = timeNow;
}
</script>
```

逐片元光照，多着色方案切换

```
<div class="page-header"><h3>13、逐片元光照，多着色方案切换</h3></div>

<canvas id = "test13-canvas" width = "800" height = "600"></canvas>
<br/>

<input type="checkbox" id="lighting" checked /> 使用光照
<br/>
<input type="checkbox" id="per-fragment" checked /> 逐片元计算光照
<br/>
<input type="checkbox" id="textures" checked /> 使用纹理
<br/>
<h4>点光:</h4>
<table>
    <tr>
        <td><b>位置:</b></td>
        <td>X: <input type="text" id="lightPositionX" value="0.0" /></td>
        <td>Y: <input type="text" id="lightPositionY" value="0.0" /></td>
        <td>Z: <input type="text" id="lightPositionZ" value="-5.0" /></td>
    </tr>
    <tr>
        <td><b>颜色:</b></td>
        <td>R: <input type="text" id="pointR" value="0.8" /></td>
        <td>G: <input type="text" id="pointG" value="0.8" /></td>
        <td>B: <input type="text" id="pointB" value="0.8" /></td>
    </tr>
</table>
<h4>环境光:</h4>
<table style="border: 0; padding: 10px;">
    <tr>
        <td><b>颜色:</b></td>
        <td>R: <input type="text" id="ambientR" value="0.2" /></td>
        <td>G: <input type="text" id="ambientG" value="0.2" /></td>
        <td>B: <input type="text" id="ambientB" value="0.2" /></td>
    </tr>
</table>
<br/>
月球表面纹理图片来自
<a href="http://maps.jpl.nasa.gov/">the Jet Propulsion Laboratory</a>.
<script id = "per-vertex-lighting-vs" type = "x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    attribute vec3 aVertexNormal;
    attribute vec2 aTextureCoord;

    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;
    uniform mat3 uNMatrix;

    uniform vec3 uAmbientColor;

    uniform vec3 uPointLightingLocation;
    uniform vec3 uPointLightingColor;

    uniform bool uUseLighting;
```

```
    varying vec2 vTextureCoord;
    varying vec3 vLightWeighting;

    void main(void)
    {
        vec4 mvPosition = uMVMatrix * vec4(aVertexPosition, 1.0);
        gl_Position = uPMatrix * mvPosition;
        vTextureCoord = aTextureCoord;

        if (!uUseLighting)
        {
            vLightWeighting = vec3(1.0, 1.0, 1.0);
        }
        else
        {
            vec3 lightDirection =
                normalize(uPointLightingLocation - mvPosition.xyz);

            vec3 transformedNormal = uNMatrix * aVertexNormal;
            float directionalLightWeighting =
                max(dot(transformedNormal, lightDirection), 0.0);
            vLightWeighting =
                uAmbientColor + uPointLightingColor * directionalLightWeighting;
        }
    }
</script>

<script id = "per-vertex-lighting-fs" type = "x-shader/x-fragment">
    precision mediump float;

    varying vec2 vTextureCoord;
    varying vec3 vLightWeighting;

    uniform bool uUseTextures;

    uniform sampler2D uSampler;

    void main(void)
    {
        vec4 fragmentColor;
        if (uUseTextures)
        {
            fragmentColor = texture2D(uSampler, vec2(vTextureCoord.s, vTextureCoord.t));
        }
        else
        {
            fragmentColor = vec4(1.0, 1.0, 1.0, 1.0);
        }
        gl_FragColor = vec4(fragmentColor.rgb * vLightWeighting, fragmentColor.a);
    }
</script>



<script id = "per-fragment-lighting-vs" type = "x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    attribute vec3 aVertexNormal;
    attribute vec2 aTextureCoord;
```

```
    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;
    uniform mat3 uNMatrix;

    varying vec2 vTextureCoord;
    varying vec3 vTransformedNormal;
    varying vec4 vPosition;


    void main(void)
    {
        vPosition = uMVMatrix * vec4(aVertexPosition, 1.0);
        gl_Position = uPMatrix * vPosition;
        vTextureCoord = aTextureCoord;
        vTransformedNormal = uNMatrix * aVertexNormal;
    }
</script>
<script id = "per-fragment-lighting-fs" type = "x-shader/x-fragment">
    precision mediump float;

    varying vec2 vTextureCoord;
    varying vec3 vTransformedNormal;
    varying vec4 vPosition;

    uniform bool uUseLighting;
    uniform bool uUseTextures;

    uniform vec3 uAmbientColor;

    uniform vec3 uPointLightingLocation;
    uniform vec3 uPointLightingColor;

    uniform sampler2D uSampler;


    void main(void)
    {
        vec3 lightWeighting;
        if (!uUseLighting)
        {
            lightWeighting = vec3(1.0, 1.0, 1.0);
        }
        else
        {
            vec3 lightDirection =
                normalize(uPointLightingLocation - vPosition.xyz);

            float directionalLightWeighting =
                max(dot(normalize(vTransformedNormal), lightDirection), 0.0);
            lightWeighting =
                uAmbientColor + uPointLightingColor * directionalLightWeighting;
        }

        vec4 fragmentColor;
        if (uUseTextures)
        {
            fragmentColor =
                texture2D(uSampler, vec2(vTextureCoord.s, vTextureCoord.t));
```

```
        }
        else
        {
            fragmentColor = vec4(1.0, 1.0, 1.0, 1.0);
        }
        gl_FragColor = vec4(fragmentColor.rgb * lightWeighting, fragmentColor.a);
    }
</script>
<script type="text/javascript">
    requestAnimFrame = window.requestAnimationFrame ||
        window.mozRequestAnimationFrame ||
        window.webkitRequestAnimationFrame ||
        window.msRequestAnimationFrame ||
        window.oRequestAnimationFrame ||
        function(callback) { setTimeout(callback, 1000 / 60); };
</script>
<script type="text/javascript">
$(document).ready(function ()
{
    webGLStart();
});

function webGLStart()
{
    var canvas = $("#test13-canvas");
    initGL(canvas[0]);
    initShaders();
    initBuffers();
    initTexture();

    gl.clearColor(0.0, 0.0, 0.0, 1.0);
    gl.enable(gl.DEPTH_TEST);

    waitTexture = setInterval("tick()", 100);
}

var textureFlag = [0, 0];
var startTick = false;
var waitTick;
function tick()
{
    if(textureFlag[0] == 1 && textureFlag[1] == 1)
    {
        startTick = true;
        textureFlag[0] = 2;
        textureFlag[1] = 2;
        clearInterval(waitTexture);
    }
    if(startTick)
    {
        requestAnimFrame(tick);
        drawScene();
        animate();
    }
}
var gl;
function initGL(canvas)
{
    try
```

```
    {
        gl = canvas.getContext("webgl") || canvas.getContext("experimental-webgl");
        gl.viewportWidth = canvas.width;
        gl.viewportHeight = canvas.height;
    }catch(e){}
    if(!gl)
    {
        alert("无法初始化“WebGL”。");
    }
}

var moonTexture;
var crateTexture;
function initTexture()
{
    moonTexture = gl.createTexture();
    moonTexture.image = new Image();
    moonTexture.image.onload = function()
    {
        handleLoadedTexture(moonTexture);
        textureFlag[0] = true;
    }
    moonTexture.image.src = "/Public/image/moon.gif";


    crateTexture = gl.createTexture();
    crateTexture.image = new Image();
    crateTexture.image.onload = function()
    {
        handleLoadedTexture(crateTexture);
        textureFlag[1] = true;
    }
    crateTexture.image.src = "/Public/image/crate.gif";
}
function handleLoadedTexture(texture)
{
    gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, 1);


    gl.bindTexture(gl.TEXTURE_2D, texture);
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA,
        gl.UNSIGNED_BYTE, texture.image);
    gl.texParameteri(gl.TEXTURE_2D,
        gl.TEXTURE_MAG_FILTER, gl.LINEAR);
    gl.texParameteri(gl.TEXTURE_2D,
        gl.TEXTURE_MIN_FILTER, gl.LINEAR_MIPMAP_NEAREST);

    gl.generateMipmap(gl.TEXTURE_2D);

    gl.bindTexture(gl.TEXTURE_2D, null);
}
function getShader(gl, id)
{
    var shaderScript = $("#" + id);
    if(!shaderScript.length)
    {
        return null;
    }
    var str = shaderScript.text();
```

```
    var shader;
    if(shaderScript[0].type == "x-shader/x-fragment")
    {
        shader = gl.createShader(gl.FRAGMENT_SHADER);
    }
    else if(shaderScript[0].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;
}
function createProgram(vertexShaderID, fragmentShaderID)
{
    var vertexShader = getShader(gl, vertexShaderID);
    var fragmentShader = getShader(gl, fragmentShaderID);

    var program = gl.createProgram();
    gl.attachShader(program, vertexShader);
    gl.attachShader(program, fragmentShader);
    gl.linkProgram(program);
    if(!gl.getProgramParameter(program, gl.LINK_STATUS))
    {
        alert("无法初始化“Shader”。");
    }

    program.vertexPositionAttribute =
        gl.getAttribLocation(program, "aVertexPosition");
    gl.enableVertexAttribArray(program.vertexPositionAttribute);

    program.vertexNormalAttribute =
        gl.getAttribLocation(program, "aVertexNormal");
    gl.enableVertexAttribArray(program.vertexNormalAttribute);

    program.textureCoordAttribute =
        gl.getAttribLocation(program, "aTextureCoord");
    gl.enableVertexAttribArray(program.textureCoordAttribute);

    program.pMatrixUniform =
        gl.getUniformLocation(program, "uPMatrix");
    program.mvMatrixUniform =
        gl.getUniformLocation(program, "uMVMatrix");
    program.nMatrixUniform  =
        gl.getUniformLocation(program, "uNMatrix");
    program.samplerUniform =
        gl.getUniformLocation(program, "uSampler");
    program.useTexturesUniform =
        gl.getUniformLocation(program, "uUseTextures");
```

```
    program.useLightingUniform =
        gl.getUniformLocation(program, "uUseLighting");
    program.ambientColorUniform =
        gl.getUniformLocation(program, "uAmbientColor");
    program.pointLightingLocationUniform  =
        gl.getUniformLocation(program, "uPointLightingLocation");
    program.pointLightingColorUniform  =
        gl.getUniformLocation(program, "uPointLightingColor");
    return program;
}
var currentProgram;
var perVertexProgram;
var perFragmentProgram;
function initShaders()
{
    perVertexProgram =
        createProgram("per-vertex-lighting-vs", "per-vertex-lighting-fs");
    perFragmentProgram =
        createProgram("per-fragment-lighting-vs", "per-fragment-lighting-fs");
}

var mvMatrix = mat4.create();

var mvMatrixStack = [];

var pMatrix = mat4.create();

function mvPushMatrix()
{
    var copy = mat4.clone(mvMatrix);
    mvMatrixStack.push(copy);
}
function mvPopMatrix()
{
    if(mvMatrixStack.length == 0)
    {
        throw "不合法的矩阵出栈操作!";
    }
    mvMatrix = mvMatrixStack.pop();
}

function setMatrixUniforms()
{
    gl.uniformMatrix4fv(currentProgram.pMatrixUniform, false, pMatrix);
    gl.uniformMatrix4fv(currentProgram.mvMatrixUniform, false, mvMatrix);

    var normalMatrix = mat3.create();

    mat3.fromMat4(normalMatrix, mvMatrix);
    mat3.invert(normalMatrix, normalMatrix);
    mat3.transpose(normalMatrix, normalMatrix);

    gl.uniformMatrix3fv(currentProgram.nMatrixUniform, false, normalMatrix);
}

var cubeVertexPositionBUffer;
var cubeVertexTextureCoordBuffer;
var cubeVertexIndexBuffer;
var cubeVertexNormalBuffer;
```

```
var moonVertexPositionBuffer;
var moonVertexNormalBuffer;
var moonVertexTextureCoordBuffer;
var moonVertexIndexBuffer;

function initBuffers()
{
    //箱子
    cubeVertexPositionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
    vertices = [
                // 正面
                -1.0, -1.0,  1.0,
                 1.0, -1.0,  1.0,
                 1.0,  1.0,  1.0,
                -1.0,  1.0,  1.0,

                // 背面
                -1.0, -1.0, -1.0,
                -1.0,  1.0, -1.0,
                 1.0,  1.0, -1.0,
                 1.0, -1.0, -1.0,

                // 顶部
                -1.0,  1.0, -1.0,
                -1.0,  1.0,  1.0,
                 1.0,  1.0,  1.0,
                 1.0,  1.0, -1.0,

                // 底部
                -1.0, -1.0, -1.0,
                 1.0, -1.0, -1.0,
                 1.0, -1.0,  1.0,
                -1.0, -1.0,  1.0,

                // 右侧面
                 1.0, -1.0, -1.0,
                 1.0,  1.0, -1.0,
                 1.0,  1.0,  1.0,
                 1.0, -1.0,  1.0,

                // 左侧面
                -1.0, -1.0, -1.0,
                -1.0, -1.0,  1.0,
                -1.0,  1.0,  1.0,
                -1.0,  1.0, -1.0,
                ];
    gl.bufferData(gl.ARRAY_BUFFER,
        new Float32Array(vertices), gl.STATIC_DRAW);
    cubeVertexPositionBuffer.itemSize = 3;
    cubeVertexPositionBuffer.numItems = 24;

    cubeVertexNormalBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexNormalBuffer);
    var vertexNormals = [
                    // 正面
                      0.0,  0.0,  1.0,
                      0.0,  0.0,  1.0,
```

```
                        0.0,  0.0,  1.0,
                        0.0,  0.0,  1.0,

                      // 背面
                        0.0,  0.0, -1.0,
                        0.0,  0.0, -1.0,
                        0.0,  0.0, -1.0,
                        0.0,  0.0, -1.0,

                      // 顶部
                        0.0,  1.0,  0.0,
                        0.0,  1.0,  0.0,
                        0.0,  1.0,  0.0,
                        0.0,  1.0,  0.0,

                      // 底部
                        0.0, -1.0,  0.0,
                        0.0, -1.0,  0.0,
                        0.0, -1.0,  0.0,
                        0.0, -1.0,  0.0,

                      // 右侧面
                        1.0,  0.0,  0.0,
                        1.0,  0.0,  0.0,
                        1.0,  0.0,  0.0,
                        1.0,  0.0,  0.0,

                      // 左侧面
                       -1.0,  0.0,  0.0,
                       -1.0,  0.0,  0.0,
                       -1.0,  0.0,  0.0,
                       -1.0,  0.0,  0.0,
                    ];
    gl.bufferData(gl.ARRAY_BUFFER,
        new Float32Array(vertexNormals), gl.STATIC_DRAW);
    cubeVertexNormalBuffer.itemSize = 3;
    cubeVertexNormalBuffer.numItems = 24;

    cubeVertexTextureCoordBuffer = gl.createBuffer();

    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexTextureCoordBuffer);
    textureCoords = [
                      // 正面
                      0.0, 0.0,
                      1.0, 0.0,
                      1.0, 1.0,
                      0.0, 1.0,

                      // 背面
                      1.0, 0.0,
                      1.0, 1.0,
                      0.0, 1.0,
                      0.0, 0.0,

                      // 顶部
                      0.0, 1.0,
                      0.0, 0.0,
                      1.0, 0.0,
                      1.0, 1.0,
```

```
                    // 底部
                    1.0, 1.0,
                    0.0, 1.0,
                    0.0, 0.0,
                    1.0, 0.0,

                    // 右侧面
                    1.0, 0.0,
                    1.0, 1.0,
                    0.0, 1.0,
                    0.0, 0.0,

                    // 左侧面
                    0.0, 0.0,
                    1.0, 0.0,
                    1.0, 1.0,
                    0.0, 1.0,
                   ];
    gl.bufferData(gl.ARRAY_BUFFER,
        new Float32Array(textureCoords), gl.STATIC_DRAW);
    cubeVertexTextureCoordBuffer.itemSize = 2;
    cubeVertexTextureCoordBuffer.numItems = 24;

    cubeVertexIndexBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);
    var cubeVertexIndices =
        [
          0, 1, 2,        0, 2, 3,    // 正面
          4, 5, 6,        4, 6, 7,    // 背面
          8, 9, 10,        8, 10, 11,  // 顶部
          12, 13, 14,    12, 14, 15, // 底部
          16, 17, 18,    16, 18, 19, // 右侧面
          20, 21, 22,    20, 22, 23  // 左侧面
        ];
    gl.bufferData(gl.ELEMENT_ARRAY_BUFFER,
        new Uint16Array(cubeVertexIndices), gl.STATIC_DRAW);
    cubeVertexIndexBuffer.itemSize = 1;
    cubeVertexIndexBuffer.numItems = 36;

    //月亮
    var latitudeBands = 30;
    var longitudeBands = 30;
    var radius = 1;
    var vertexPositionData = [];
    var normalData = [];
    var textureCoordData = [];
    for(var latNumber = 0; latNumber <= latitudeBands; latNumber ++)
    {
        var theta = latNumber * Math.PI / latitudeBands;
        var sinTheta = Math.sin(theta);
        var cosTheta = Math.cos(theta);
        for(var longNumber = 0; longNumber <= longitudeBands; longNumber ++)
        {
            var phi = longNumber * 2 * Math.PI / longitudeBands;
            var sinPhi = Math.sin(phi);
            var cosPhi = Math.cos(phi);
            var x = cosPhi * sinTheta;
            var y = cosTheta;
```

```
            var z = sinPhi * sinTheta;
            var u = 1 - (longNumber / longitudeBands);
            var v = 1 - (latNumber / latitudeBands);

            normalData.push(x);
            normalData.push(y);
            normalData.push(z);
            textureCoordData.push(u);
            textureCoordData.push(v);
            vertexPositionData.push(radius * x);
            vertexPositionData.push(radius * y);
            vertexPositionData.push(radius * z);
        }
    }
    var indexData = [];
    for(var latNumber = 0; latNumber < latitudeBands; latNumber ++)
    {
        for(var longNumber = 0; longNumber < longitudeBands; longNumber ++)
        {
            var first = (latNumber * (longitudeBands + 1)) + longNumber;
            var second = first + longitudeBands + 1;
            indexData.push(first);
            indexData.push(second);
            indexData.push(first + 1);
            indexData.push(second);
            indexData.push(second + 1);
            indexData.push(first + 1);
        }
    }
    moonVertexNormalBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, moonVertexNormalBuffer);
    gl.bufferData(gl.ARRAY_BUFFER,
        new Float32Array(normalData), gl.STATIC_DRAW);
    moonVertexNormalBuffer.itemSize = 3;
    moonVertexNormalBuffer.numItems = normalData.length / 3;

    moonVertexTextureCoordBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, moonVertexTextureCoordBuffer);
    gl.bufferData(gl.ARRAY_BUFFER,
        new Float32Array(textureCoordData), gl.STATIC_DRAW);
    moonVertexTextureCoordBuffer.itemSize = 2;
    moonVertexTextureCoordBuffer.numItems = textureCoordData.length / 2;

    moonVertexPositionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, moonVertexPositionBuffer);
    gl.bufferData(gl.ARRAY_BUFFER,
        new Float32Array(vertexPositionData), gl.STATIC_DRAW);
    moonVertexPositionBuffer.itemSize = 3;
    moonVertexPositionBuffer.numItems = vertexPositionData.length / 3;

    moonVertexIndexBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, moonVertexIndexBuffer);
    gl.bufferData(gl.ELEMENT_ARRAY_BUFFER,
        new Uint16Array(indexData), gl.STATIC_DRAW);
    moonVertexIndexBuffer.itemSize = 1;
    moonVertexIndexBuffer.numItems = indexData.length;
}

var moonAngle = 180;
```

```
var cubeAngle = 0;

function drawScene()
{
    gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    mat4.perspective(pMatrix, 45,
        gl.viewportWidth / gl.viewportHeight, 0.1, 100.0);

    var perFragmentLighting = $("#per-fragment").is(":checked");
    if(perFragmentLighting)
    {
        currentProgram = perFragmentProgram;
    }
    else
    {
        currentProgram = perVertexProgram;
    }
    gl.useProgram(currentProgram);

    var lighting = $("#lighting").is(":checked");
    gl.uniform1i(currentProgram.useLightingUniform, lighting);
    if(lighting)
    {
        gl.uniform3f(
            currentProgram.ambientColorUniform,
            parseFloat($("#ambientR").val()),
            parseFloat($("#ambientG").val()),
            parseFloat($("#ambientB").val())
            );
        gl.uniform3f(
            currentProgram.pointLightingLocationUniform,
            parseFloat($("#lightPositionX").val()),
            parseFloat($("#lightPositionY").val()),
            parseFloat($("#lightPositionZ").val())
        );

        gl.uniform3f(
            currentProgram.pointLightingColorUniform,
            parseFloat($("#pointR").val()),
            parseFloat($("#pointG").val()),
            parseFloat($("#pointB").val())
        );

    }

    var textures = $("#textures").is(":checked");
    gl.uniform1i(currentProgram.useTexturesUniform, textures);

    mat4.identity(mvMatrix);
    mat4.translate(mvMatrix, mvMatrix, [0, 0, -5]);
    mat4.rotate(mvMatrix, mvMatrix, degToRad(30), [1, 0, 0]);

    mvPushMatrix();

    mat4.rotate(mvMatrix, mvMatrix, degToRad(moonAngle), [0, 1, 0]);
    mat4.translate(mvMatrix, mvMatrix, [2, 0, 0]);
```

```
    gl.activeTexture(gl.TEXTURE0);
    gl.bindTexture(gl.TEXTURE_2D, moonTexture);
    gl.uniform1i(currentProgram.samplerUniform, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, moonVertexTextureCoordBuffer);
    gl.vertexAttribPointer(currentProgram.textureCoordAttribute,
        moonVertexTextureCoordBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, moonVertexPositionBuffer);
    gl.vertexAttribPointer(currentProgram.vertexPositionAttribute,
        moonVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, moonVertexNormalBuffer);
    gl.vertexAttribPointer(currentProgram.vertexNormalAttribute,
        moonVertexNormalBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, moonVertexIndexBuffer);
    setMatrixUniforms();
    gl.drawElements(gl.TRIANGLES,
        moonVertexIndexBuffer.numItems, gl.UNSIGNED_SHORT, 0);

    mvPopMatrix();

    mvPushMatrix();
    mat4.rotate(mvMatrix, mvMatrix, degToRad(cubeAngle), [0, 1, 0]);
    mat4.translate(mvMatrix, mvMatrix, [1.25, 0, 0]);

    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
    gl.vertexAttribPointer(currentProgram.vertexPositionAttribute,
        cubeVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexNormalBuffer);
    gl.vertexAttribPointer(currentProgram.vertexNormalAttribute,
        cubeVertexNormalBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexTextureCoordBuffer);
    gl.vertexAttribPointer(currentProgram.textureCoordAttribute,
        cubeVertexTextureCoordBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.activeTexture(gl.TEXTURE0);
    gl.bindTexture(gl.TEXTURE_2D, crateTexture);
    gl.uniform1i(currentProgram.samplerUniform, 0);

    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);
    setMatrixUniforms();
    gl.drawElements(gl.TRIANGLES,
        cubeVertexIndexBuffer.numItems, gl.UNSIGNED_SHORT, 0);
    mvPopMatrix();
}

function degToRad(degrees)
{
    return degrees * Math.PI / 180;
}
var lastTime = 0;
function animate()
{
    var timeNow = new Date().getTime();
    if(lastTime != 0)
```

```
    {
        var elapsed = timeNow - lastTime;

         moonAngle += 0.05 * elapsed;
         cubeAngle += 0.05 * elapsed;
    }
    lastTime = timeNow;
}
</script>
```

镜面高光

```
<div class="page-header"><h3>14、镜面高光</h3></div>

<canvas id = "test14-canvas" width = "800" height = "600"></canvas>
<br/>
<div id="loadingtext">正在加载数据...</div>
<br/>
<input type="checkbox" id="specular" checked /> 显示镜面高光
<br/>
<input type="checkbox" id="lighting" checked /> 使用光照
<br/>
<h4>纹理：</h4>
<select id="texture">
    <option value="none">无</option>
    <option selected value="galvanized">镀锌</option>
    <option value="earth">地球</option>
</select>
<br/>
<h4>材质：</h4>
<table style="border: 0; padding: 10px;">
    <tr>
        <td><b>Shininess:</b></td>
        <td><input type="text" id="shininess" value="32.0" /></td>
    </tr>
</table>
<br/>
<h4>点光:</h4>
<table>
    <tr>
        <td><b>位置:</b></td>
        <td>X: <input type="text" id="lightPositionX" value="-10.0" />
        </td>
        <td>Y: <input type="text" id="lightPositionY" value="4.0" /></td>
        <td>Z: <input type="text" id="lightPositionZ" value="-20.0" /></td>
    </tr>
    <tr>
        <td><b>镜面反射颜色:</b></td>
        <td>R: <input type="text" id="specularR" value="0.8" /></td>
        <td>G: <input type="text" id="specularG" value="0.8" /></td>
        <td>B: <input type="text" id="specularB" value="0.8" /></td>
    </tr>
    <tr>
        <td><b>漫反射颜色:</b>
        <td>R: <input type="text" id="diffuseR" value="0.8" />
        <td>G: <input type="text" id="diffuseG" value="0.8" />
        <td>B: <input type="text" id="diffuseB" value="0.8" />
    </tr>
</table>
<h4>环境光:</h4>
<table style="border: 0; padding: 10px;">
    <tr>
        <td><b>颜色:</b></td>
        <td>R: <input type="text" id="ambientR" value="0.2" /></td>
        <td>G: <input type="text" id="ambientG" value="0.2" /></td>
        <td>B: <input type="text" id="ambientB" value="0.2" /></td>
```

```
        </tr>
    </table>
    <br/>
    镀锌纹理来自：
    <a href="http://www.arroway-textures.com/">
        Arroway Textures
    </a>.
    <br/>
    地球纹理来自：
    <a href="http://www.esa.int/esaEO/SEMGSY2IU7E_index_0.html">
        the European Space Agency/Envisat
    </a>.
    <br/>

    <script id = "per-fragment-lighting-vs" type = "x-shader/x-vertex">
        attribute vec3 aVertexPosition;
        attribute vec3 aVertexNormal;
        attribute vec2 aTextureCoord;

        uniform mat4 uMVMatrix;
        uniform mat4 uPMatrix;
        uniform mat3 uNMatrix;

        varying vec2 vTextureCoord;
        varying vec3 vTransformedNormal;
        varying vec4 vPosition;


        void main(void)
        {
            vPosition = uMVMatrix * vec4(aVertexPosition, 1.0);
            gl_Position = uPMatrix * vPosition;
            vTextureCoord = aTextureCoord;
            vTransformedNormal = uNMatrix * aVertexNormal;
        }
    </script>
    <script id = "per-fragment-lighting-fs" type = "x-shader/x-fragment">
        precision mediump float;

        varying vec2 vTextureCoord;
        varying vec3 vTransformedNormal;
        varying vec4 vPosition;

        uniform float uMaterialShininess;
        uniform bool uShowSpecularHighlights;

        uniform bool uUseLighting;
        uniform bool uUseTextures;

        uniform vec3 uAmbientColor;

        uniform vec3 uPointLightingLocation;
        uniform vec3 uPointLightingSpecularColor;
        uniform vec3 uPointLightingDiffuseColor;

        uniform sampler2D uSampler;

        void main(void)
        {
```

```
        vec3 lightWeighting;
        if (!uUseLighting)
        {
            lightWeighting = vec3(1.0, 1.0, 1.0);
        }
        else
        {
            vec3 lightDirection =
                normalize(uPointLightingLocation - vPosition.xyz);
            vec3 normal = normalize(vTransformedNormal);

            float specularLightWeighting = 0.0;

            if(uShowSpecularHighlights)
            {
                vec3 eyeDirection = normalize(-vPosition.xyz);
                vec3 reflectionDirection = reflect(-lightDirection, normal);

                specularLightWeighting =
                    pow(max(dot(reflectionDirection, eyeDirection), 0.0),
                    uMaterialShininess);
            }

            float diffuseLightWeighting = max(dot(normal, lightDirection), 0.0);
            lightWeighting = uAmbientColor +
                uPointLightingSpecularColor * specularLightWeighting +
                uPointLightingDiffuseColor * diffuseLightWeighting;
        }

        vec4 fragmentColor;
        if (uUseTextures)
        {
            fragmentColor =
                texture2D(uSampler, vec2(vTextureCoord.s, vTextureCoord.t));
        }
        else
        {
            fragmentColor = vec4(1.0, 1.0, 1.0, 1.0);
        }
        gl_FragColor = vec4(fragmentColor.rgb * lightWeighting, fragmentColor.a);
    }
</script>
<script type="text/javascript">
    requestAnimFrame = window.requestAnimationFrame ||
        window.mozRequestAnimationFrame ||
        window.webkitRequestAnimationFrame ||
        window.msRequestAnimationFrame ||
        window.oRequestAnimationFrame ||
        function(callback) { setTimeout(callback, 1000 / 60); };
</script>
<script type="text/javascript">
$(document).ready(function ()
{
    webGLStart();
});

function webGLStart()
{
    var canvas = $("#test14-canvas");
```

```
    initGL(canvas[0]);
    initShaders();
    initTextures();

    loadTeapot();

    gl.clearColor(0.0, 0.0, 0.0, 1.0);
    gl.enable(gl.DEPTH_TEST);

    waitTexture = setInterval("tick()", 100);
}

var textureFlag = [0, 0];
var startTick = false;
var waitTick;
function tick()
{
    if(textureFlag[0] == 1 && textureFlag[1] == 1)
    {
        startTick = true;
        textureFlag[0] = 2;
        textureFlag[1] = 2;
        clearInterval(waitTexture);
    }
    if(startTick)
    {
        requestAnimFrame(tick);
        drawScene();
        animate();
    }
}
var gl;
function initGL(canvas)
{
    try
    {
        gl = canvas.getContext("webgl") || canvas.getContext("experimental-webgl");
        gl.viewportWidth = canvas.width;
        gl.viewportHeight = canvas.height;
    }catch(e){}
    if(!gl)
    {
        alert("无法初始化“WebGL”。");
    }
}

var earthTexture;
var galvanizedTexture;
function initTextures()
{
    earthTexture = gl.createTexture();
    earthTexture.image = new Image();
    earthTexture.image.onload = function ()
    {
        handleLoadedTexture(earthTexture)
        textureFlag[0] = true;
    }
    earthTexture.image.src = "/Public/image/earth.jpg";
```

```
    galvanizedTexture = gl.createTexture();
    galvanizedTexture.image = new Image();
    galvanizedTexture.image.onload = function ()
    {
        handleLoadedTexture(galvanizedTexture)
        textureFlag[1] = true;
    }
    galvanizedTexture.image.src =
        "/Public/image/arroway.de_metal+structure+06_d100_flat.jpg";
}
function handleLoadedTexture(texture)
{
    gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, 1);


    gl.bindTexture(gl.TEXTURE_2D, texture);
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA,
        gl.UNSIGNED_BYTE, texture.image);
    gl.texParameteri(gl.TEXTURE_2D,
        gl.TEXTURE_MAG_FILTER, gl.LINEAR);
    gl.texParameteri(gl.TEXTURE_2D,
        gl.TEXTURE_MIN_FILTER, gl.LINEAR_MIPMAP_NEAREST);

    gl.generateMipmap(gl.TEXTURE_2D);

    gl.bindTexture(gl.TEXTURE_2D, null);
}
function getShader(gl, id)
{
    var shaderScript = $("#" + id);
    if(!shaderScript.length)
    {
        return null;
    }
    var str = shaderScript.text();

    var shader;
    if(shaderScript[0].type == "x-shader/x-fragment")
    {
        shader = gl.createShader(gl.FRAGMENT_SHADER);
    }
    else if(shaderScript[0].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;
}
```

```
var shaderProgram;

function initShaders()
{
    var vertexShader = getShader(gl, "per-fragment-lighting-vs");
    var fragmentShader = getShader(gl, "per-fragment-lighting-fs");

    shaderProgram  = gl.createProgram();
    gl.attachShader(shaderProgram, vertexShader);
    gl.attachShader(shaderProgram, fragmentShader);
    gl.linkProgram (shaderProgram );
    if(!gl.getProgramParameter(shaderProgram, gl.LINK_STATUS))
    {
        alert("无法初始化“Shader”。");
    }
    gl.useProgram(shaderProgram);
    shaderProgram.vertexPositionAttribute =
        gl.getAttribLocation(shaderProgram, "aVertexPosition");
    gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);

    shaderProgram.vertexNormalAttribute =
        gl.getAttribLocation(shaderProgram, "aVertexNormal");
    gl.enableVertexAttribArray(shaderProgram.vertexNormalAttribute);

    shaderProgram.textureCoordAttribute =
        gl.getAttribLocation(shaderProgram, "aTextureCoord");
    gl.enableVertexAttribArray(shaderProgram.textureCoordAttribute);

    shaderProgram.pMatrixUniform =
        gl.getUniformLocation(shaderProgram, "uPMatrix");
    shaderProgram.mvMatrixUniform =
        gl.getUniformLocation(shaderProgram, "uMVMatrix");
    shaderProgram.nMatrixUniform  =
        gl.getUniformLocation(shaderProgram, "uNMatrix");
    shaderProgram.samplerUniform =
        gl.getUniformLocation(shaderProgram, "uSampler");
    shaderProgram.materialShininessUniform =
        gl.getUniformLocation(shaderProgram, "uMaterialShininess");
    shaderProgram.showSpecularHighlightsUniform =
        gl.getUniformLocation(shaderProgram, "uShowSpecularHighlights");
    shaderProgram.useTexturesUniform =
        gl.getUniformLocation(shaderProgram, "uUseTextures");
    shaderProgram.useLightingUniform =
        gl.getUniformLocation(shaderProgram, "uUseLighting");
    shaderProgram.ambientColorUniform =
        gl.getUniformLocation(shaderProgram, "uAmbientColor");
    shaderProgram.pointLightingLocationUniform  =
        gl.getUniformLocation(shaderProgram, "uPointLightingLocation");
    shaderProgram.pointLightingSpecularColorUniform =
        gl.getUniformLocation(shaderProgram, "uPointLightingSpecularColor");
    shaderProgram.pointLightingDiffuseColorUniform =
        gl.getUniformLocation(shaderProgram, "uPointLightingDiffuseColor");
}

var mvMatrix = mat4.create();

var mvMatrixStack = [];

var pMatrix = mat4.create();
```

```
function mvPushMatrix()
{
    var copy = mat4.clone(mvMatrix);
    mvMatrixStack.push(copy);
}
function mvPopMatrix()
{
    if(mvMatrixStack.length == 0)
    {
        throw "不合法的矩阵出栈操作!";
    }
    mvMatrix = mvMatrixStack.pop();
}

function setMatrixUniforms()
{
    gl.uniformMatrix4fv(shaderProgram.pMatrixUniform, false, pMatrix);
    gl.uniformMatrix4fv(shaderProgram.mvMatrixUniform, false, mvMatrix);

    var normalMatrix = mat3.create();

    mat3.fromMat4(normalMatrix, mvMatrix);
    mat3.invert(normalMatrix, normalMatrix);
    mat3.transpose(normalMatrix, normalMatrix);

    gl.uniformMatrix3fv(shaderProgram.nMatrixUniform, false, normalMatrix);
}

var teapotVertexPositionBuffer;
var teapotVertexNormalBuffer;
var teapotVertexTextureCoordBuffer;
var teapotVertexIndexBuffer;
function loadTeapot()
{
    $.getJSON(
        "/Public/json/Teapot.json",
        function(data)
        {
            handleLoadedTeapot(data);
        }
    );
}
function handleLoadedTeapot(teapotData)
{
    teapotVertexNormalBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, teapotVertexNormalBuffer);
    gl.bufferData(gl.ARRAY_BUFFER,
        new Float32Array(teapotData.vertexNormals), gl.STATIC_DRAW);
    teapotVertexNormalBuffer.itemSize = 3;
    teapotVertexNormalBuffer.numItems =
        teapotData.vertexNormals.length / 3;
    teapotVertexTextureCoordBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, teapotVertexTextureCoordBuffer);
    gl.bufferData(gl.ARRAY_BUFFER,
        new Float32Array(teapotData.vertexTextureCoords), gl.STATIC_DRAW);
    teapotVertexTextureCoordBuffer.itemSize = 2;
    teapotVertexTextureCoordBuffer.numItems =
        teapotData.vertexTextureCoords.length / 2;
```

```
    teapotVertexPositionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, teapotVertexPositionBuffer);
    gl.bufferData(gl.ARRAY_BUFFER,
        new Float32Array(teapotData.vertexPositions), gl.STATIC_DRAW);
    teapotVertexPositionBuffer.itemSize = 3;
    teapotVertexPositionBuffer.numItems =
        teapotData.vertexPositions.length / 3;

    teapotVertexIndexBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, teapotVertexIndexBuffer);
    gl.bufferData(gl.ELEMENT_ARRAY_BUFFER,
        new Uint16Array(teapotData.indices), gl.STATIC_DRAW);
    teapotVertexIndexBuffer.itemSize = 1;
    teapotVertexIndexBuffer.numItems =
        teapotData.indices.length;

    $("#loadingtext").text("");
}

var teapotAngle = 180;

function drawScene()
{
    gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    if (teapotVertexPositionBuffer == null ||
        teapotVertexNormalBuffer == null ||
        teapotVertexTextureCoordBuffer == null ||
        teapotVertexIndexBuffer == null)
    {
        return;
    }

    mat4.perspective(pMatrix, 45,
        gl.viewportWidth / gl.viewportHeight, 0.1, 100.0);

    var specularHighlights = $("#specular").is(":checked");
    gl.uniform1i(shaderProgram.showSpecularHighlightsUniform, specularHighlights);

    var lighting = $("#lighting").is(":checked");
    gl.uniform1i(shaderProgram.useLightingUniform, lighting);
    if(lighting)
    {
        gl.uniform3f(
            shaderProgram.ambientColorUniform,
            parseFloat($("#ambientR").val()),
            parseFloat($("#ambientG").val()),
            parseFloat($("#ambientB").val())
            );
        gl.uniform3f(
            shaderProgram.pointLightingLocationUniform,
            parseFloat($("#lightPositionX").val()),
            parseFloat($("#lightPositionY").val()),
            parseFloat($("#lightPositionZ").val())
        );

        gl.uniform3f(
```

```
            shaderProgram.pointLightingSpecularColorUniform,
            parseFloat($("#specularR").val()),
            parseFloat($("#specularG").val()),
            parseFloat($("#specularB").val())
        );

        gl.uniform3f(
            shaderProgram.pointLightingDiffuseColorUniform,
            parseFloat($("#diffuseR").val()),
            parseFloat($("#diffuseG").val()),
            parseFloat($("#diffuseB").val())
        );

    }

    var texture = $("#texture").val();
    gl.uniform1i(shaderProgram.useTexturesUniform, texture != "none");

    mat4.identity(mvMatrix);
    mat4.translate(mvMatrix, mvMatrix, [0, 0, -40]);
    mat4.rotate(mvMatrix, mvMatrix, degToRad(23.4), [1, 0, -1]);
    mat4.rotate(mvMatrix, mvMatrix, degToRad(teapotAngle), [0, 1, 0]);

    gl.activeTexture(gl.TEXTURE0);
    if(texture == "earth")
    {
        gl.bindTexture(gl.TEXTURE_2D, earthTexture);
    }
    else if(texture == "galvanized")
    {
        gl.bindTexture(gl.TEXTURE_2D, galvanizedTexture);
    }
    gl.uniform1i(shaderProgram.samplerUniform, 0);
    gl.uniform1f(shaderProgram.materialShininessUniform,
        parseFloat($("#shininess").val()));
    gl.bindBuffer(gl.ARRAY_BUFFER, teapotVertexTextureCoordBuffer);
    gl.vertexAttribPointer(shaderProgram.textureCoordAttribute,
        teapotVertexTextureCoordBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, teapotVertexPositionBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
        teapotVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, teapotVertexNormalBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexNormalAttribute,
        teapotVertexNormalBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, teapotVertexIndexBuffer);
    setMatrixUniforms();
    gl.drawElements(gl.TRIANGLES,
        teapotVertexIndexBuffer.numItems, gl.UNSIGNED_SHORT, 0);
}

function degToRad(degrees)
{
    return degrees * Math.PI / 180;
}
var lastTime = 0;
function animate()
```

```
{
    var timeNow = new Date().getTime();
    if(lastTime != 0)
    {
        var elapsed = timeNow - lastTime;

         teapotAngle += 0.05 * elapsed;
    }
    lastTime = timeNow;
}
</script>
```

镜面地图

```
<div class="page-header"><h3>15、镜面地图</h3></div>

<canvas id = "test15-canvas" width = "800" height = "600"></canvas>
<br/>
<input type="checkbox" id="color-map" checked /> 使用纹理
<br/>
<input type="checkbox" id="specular-map" checked /> 使用高光标记图
<br/>
<input type="checkbox" id="lighting" checked /> 使用光照
<br/>

<h4>点光:</h4>
<table>
    <tr>
        <td><b>位置:</b></td>
        <td>X: <input type="text" id="lightPositionX" value="-10.0" />
        </td>
        <td>Y: <input type="text" id="lightPositionY" value="4.0" /></td>
        <td>Z: <input type="text" id="lightPositionZ" value="-20.0" /></td>
    </tr>
    <tr>
        <td><b>镜面反射颜色:</b></td>
        <td>R: <input type="text" id="specularR" value="5.0" /></td>
        <td>G: <input type="text" id="specularG" value="5.0" /></td>
        <td>B: <input type="text" id="specularB" value="5.0" /></td>
    </tr>
    <tr>
        <td><b>漫反射颜色:</b>
        <td>R: <input type="text" id="diffuseR" value="0.8" />
        <td>G: <input type="text" id="diffuseG" value="0.8" />
        <td>B: <input type="text" id="diffuseB" value="0.8" />
    </tr>
</table>
<h4>环境光:</h4>
<table style="border: 0; padding: 10px;">
    <tr>
        <td><b>颜色:</b></td>
        <td>R: <input type="text" id="ambientR" value="0.4" /></td>
        <td>G: <input type="text" id="ambientG" value="0.4" /></td>
        <td>B: <input type="text" id="ambientB" value="0.4" /></td>
    </tr>
</table>
<br/>
地球纹理来自：
<a href="http://www.esa.int/esaEO/SEMGSY2IU7E_index_0.html">
    the European Space Agency/Envisat
</a>.
<br/>

<script id = "per-fragment-lighting-vs" type = "x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    attribute vec3 aVertexNormal;
    attribute vec2 aTextureCoord;
```

```
    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;
    uniform mat3 uNMatrix;

    varying vec2 vTextureCoord;
    varying vec3 vTransformedNormal;
    varying vec4 vPosition;


    void main(void)
    {
        vPosition = uMVMatrix * vec4(aVertexPosition, 1.0);
        gl_Position = uPMatrix * vPosition;
        vTextureCoord = aTextureCoord;
        vTransformedNormal = uNMatrix * aVertexNormal;
    }
</script>
<script id = "per-fragment-lighting-fs" type = "x-shader/x-fragment">
    precision mediump float;

    varying vec2 vTextureCoord;
    varying vec3 vTransformedNormal;
    varying vec4 vPosition;

    uniform bool uUseColorMap;
    uniform bool uUseSpecularMap;
    uniform bool uUseLighting;

    uniform vec3 uAmbientColor;

    uniform vec3 uPointLightingLocation;
    uniform vec3 uPointLightingSpecularColor;
    uniform vec3 uPointLightingDiffuseColor;

    uniform sampler2D uColorMapSampler;
    uniform sampler2D uSpecularMapSampler;


    void main(void)
    {
        vec3 lightWeighting;
        if (!uUseLighting)
        {
            lightWeighting = vec3(1.0, 1.0, 1.0);
        }
        else
        {
            vec3 lightDirection =
                normalize(uPointLightingLocation - vPosition.xyz);
            vec3 normal = normalize(vTransformedNormal);

            float specularLightWeighting = 0.0;
            float shininess = 32.0;
            if (uUseSpecularMap)
            {
                shininess =
                    texture2D(uSpecularMapSampler,
                    vec2(vTextureCoord.s, vTextureCoord.t)).r * 255.0;
            }
```

```
            if (shininess < 255.0)
            {
                vec3 eyeDirection = normalize(-vPosition.xyz);
                vec3 reflectionDirection = reflect(-lightDirection, normal);

                specularLightWeighting =
                    pow(max(dot(reflectionDirection, eyeDirection), 0.0),
                    shininess);
            }

            float diffuseLightWeighting = max(dot(normal, lightDirection), 0.0);
            lightWeighting = uAmbientColor +
                             uPointLightingSpecularColor * specularLightWeighting +
                             uPointLightingDiffuseColor * diffuseLightWeighting;
        }

        vec4 fragmentColor;
        if (uUseColorMap)
        {
            fragmentColor =
                texture2D(uColorMapSampler,
                vec2(vTextureCoord.s, vTextureCoord.t));
        }
        else
        {
            fragmentColor = vec4(1.0, 1.0, 1.0, 1.0);
        }
        gl_FragColor = vec4(fragmentColor.rgb * lightWeighting, fragmentColor.a);
    }
</script>
<script type="text/javascript">
    requestAnimFrame = window.requestAnimationFrame ||
        window.mozRequestAnimationFrame ||
        window.webkitRequestAnimationFrame ||
        window.msRequestAnimationFrame ||
        window.oRequestAnimationFrame ||
        function(callback) { setTimeout(callback, 1000 / 60); };
</script>
<script type="text/javascript">
$(document).ready(function ()
{
    webGLStart();
});

function webGLStart()
{
    var canvas = $("#test15-canvas");
    initGL(canvas[0]);
    initShaders();
    initTextures();
    initBuffers();

    gl.clearColor(0.0, 0.0, 0.0, 1.0);
    gl.enable(gl.DEPTH_TEST);

    waitTexture = setInterval("tick()", 100);
}

var textureFlag = [0, 0];
```

```
var startTick = false;
var waitTick;
function tick()
{
    if(textureFlag[0] == 1 && textureFlag[1] == 1)
    {
        startTick = true;
        textureFlag[0] = 2;
        textureFlag[1] = 2;
        clearInterval(waitTexture);
    }
    if(startTick)
    {
        requestAnimFrame(tick);
        drawScene();
        animate();
    }
}
var gl;
function initGL(canvas)
{
    try
    {
        gl = canvas.getContext("webgl") || canvas.getContext("experimental-webgl");
        gl.viewportWidth = canvas.width;
        gl.viewportHeight = canvas.height;
    }catch(e){}
    if(!gl)
    {
        alert("无法初始化“WebGL”。");
    }
}

var earthColorMapTexture;
var earthSpecularMapTexture;
function initTextures()
{
    earthColorMapTexture = gl.createTexture();
    earthColorMapTexture.image = new Image();
    earthColorMapTexture.image.onload = function ()
    {
        handleLoadedTexture(earthColorMapTexture)
        textureFlag[0] = true;
    }
    earthColorMapTexture.image.src = "/Public/image/earth.jpg";

    earthSpecularMapTexture  = gl.createTexture();
    earthSpecularMapTexture .image = new Image();
    earthSpecularMapTexture .image.onload = function ()
    {
        handleLoadedTexture(earthSpecularMapTexture)
        textureFlag[1] = true;
    }
    earthSpecularMapTexture .image.src = "/Public/image/earth-specular.gif";
}
function handleLoadedTexture(texture)
{
    gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, 1);
```

```
    gl.bindTexture(gl.TEXTURE_2D, texture);
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA,
        gl.UNSIGNED_BYTE, texture.image);
    gl.texParameteri(gl.TEXTURE_2D,
        gl.TEXTURE_MAG_FILTER, gl.LINEAR);
    gl.texParameteri(gl.TEXTURE_2D,
        gl.TEXTURE_MIN_FILTER, gl.LINEAR_MIPMAP_NEAREST);

    gl.generateMipmap(gl.TEXTURE_2D);

    gl.bindTexture(gl.TEXTURE_2D, null);
}
function getShader(gl, id)
{
    var shaderScript = $("#" + id);
    if(!shaderScript.length)
    {
        return null;
    }
    var str = shaderScript.text();

    var shader;
    if(shaderScript[0].type == "x-shader/x-fragment")
    {
        shader = gl.createShader(gl.FRAGMENT_SHADER);
    }
    else if(shaderScript[0].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;
}

var shaderProgram;

function initShaders()
{
    var vertexShader = getShader(gl, "per-fragment-lighting-vs");
    var fragmentShader = getShader(gl, "per-fragment-lighting-fs");

    shaderProgram  = gl.createProgram();
    gl.attachShader(shaderProgram, vertexShader);
    gl.attachShader(shaderProgram, fragmentShader);
    gl.linkProgram (shaderProgram );
    if(!gl.getProgramParameter(shaderProgram, gl.LINK_STATUS))
    {
        alert("无法初始化“Shader”。");
```

```
    }
    gl.useProgram(shaderProgram);
    shaderProgram.vertexPositionAttribute =
        gl.getAttribLocation(shaderProgram, "aVertexPosition");
    gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);

    shaderProgram.vertexNormalAttribute =
        gl.getAttribLocation(shaderProgram, "aVertexNormal");
    gl.enableVertexAttribArray(shaderProgram.vertexNormalAttribute);

    shaderProgram.textureCoordAttribute =
        gl.getAttribLocation(shaderProgram, "aTextureCoord");
    gl.enableVertexAttribArray(shaderProgram.textureCoordAttribute);

    shaderProgram.pMatrixUniform =
        gl.getUniformLocation(shaderProgram, "uPMatrix");
    shaderProgram.mvMatrixUniform =
        gl.getUniformLocation(shaderProgram, "uMVMatrix");
    shaderProgram.nMatrixUniform  =
        gl.getUniformLocation(shaderProgram, "uNMatrix");
    shaderProgram.colorMapSamplerUniform =
        gl.getUniformLocation(shaderProgram, "uColorMapSampler");
    shaderProgram.specularMapSamplerUniform =
        gl.getUniformLocation(shaderProgram, "uSpecularMapSampler");
    shaderProgram.useColorMapUniform =
        gl.getUniformLocation(shaderProgram, "uUseColorMap");
    shaderProgram.useSpecularMapUniform =
        gl.getUniformLocation(shaderProgram, "uUseSpecularMap");
    shaderProgram.useLightingUniform =
        gl.getUniformLocation(shaderProgram, "uUseLighting");
    shaderProgram.ambientColorUniform =
        gl.getUniformLocation(shaderProgram, "uAmbientColor");
    shaderProgram.pointLightingLocationUniform =
        gl.getUniformLocation(shaderProgram, "uPointLightingLocation");
    shaderProgram.pointLightingSpecularColorUniform =
        gl.getUniformLocation(shaderProgram, "uPointLightingSpecularColor");
    shaderProgram.pointLightingDiffuseColorUniform =
        gl.getUniformLocation(shaderProgram, "uPointLightingDiffuseColor");
}

var mvMatrix = mat4.create();

var mvMatrixStack = [];

var pMatrix = mat4.create();

function mvPushMatrix()
{
    var copy = mat4.clone(mvMatrix);
    mvMatrixStack.push(copy);
}
function mvPopMatrix()
{
    if(mvMatrixStack.length == 0)
    {
        throw "不合法的矩阵出栈操作!";
    }
    mvMatrix = mvMatrixStack.pop();
}
```

```
function setMatrixUniforms()
{
    gl.uniformMatrix4fv(shaderProgram.pMatrixUniform, false, pMatrix);
    gl.uniformMatrix4fv(shaderProgram.mvMatrixUniform, false, mvMatrix);

    var normalMatrix = mat3.create();

    mat3.fromMat4(normalMatrix, mvMatrix);
    mat3.invert(normalMatrix, normalMatrix);
    mat3.transpose(normalMatrix, normalMatrix);

    gl.uniformMatrix3fv(shaderProgram.nMatrixUniform, false, normalMatrix);
}

var sphereVertexPositionBuffer;
var sphereVertexNormalBuffer;
var sphereVertexTextureCoordBuffer;
var sphereVertexIndexBuffer;

function initBuffers()
{
    var latitudeBands = 30;
    var longitudeBands = 30;
    var radius = 13;

    var vertexPositionData = [];
    var normalData = [];
    var textureCoordData = [];
    for(var latNumber = 0; latNumber <= latitudeBands; latNumber ++)
    {
        var theta = latNumber * Math.PI / latitudeBands;
        var sinTheta = Math.sin(theta);
        var cosTheta = Math.cos(theta);
        for(var longNumber = 0; longNumber <= longitudeBands; longNumber ++)
        {
            var phi = longNumber * 2 * Math.PI / longitudeBands;
            var sinPhi = Math.sin(phi);
            var cosPhi = Math.cos(phi);
            var x = cosPhi * sinTheta;
            var y = cosTheta;
            var z = sinPhi * sinTheta;
            var u = 1 - (longNumber / longitudeBands);
            var v = 1 - (latNumber / latitudeBands);

            normalData.push(x);
            normalData.push(y);
            normalData.push(z);
            textureCoordData.push(u);
            textureCoordData.push(v);
            vertexPositionData.push(radius * x);
            vertexPositionData.push(radius * y);
            vertexPositionData.push(radius * z);
        }
    }
    var indexData = [];
    for(var latNumber = 0; latNumber < latitudeBands; latNumber ++)
    {
        for(var longNumber = 0; longNumber < longitudeBands; longNumber ++)
```

```
            {
                var first = (latNumber * (longitudeBands + 1)) + longNumber;
                var second = first + longitudeBands + 1;
                indexData.push(first);
                indexData.push(second);
                indexData.push(first + 1);
                indexData.push(second);
                indexData.push(second + 1);
                indexData.push(first + 1);
            }
        }
        sphereVertexNormalBuffer = gl.createBuffer();
        gl.bindBuffer(gl.ARRAY_BUFFER, sphereVertexNormalBuffer);
        gl.bufferData(gl.ARRAY_BUFFER,
            new Float32Array(normalData), gl.STATIC_DRAW);
        sphereVertexNormalBuffer.itemSize = 3;
        sphereVertexNormalBuffer.numItems = normalData.length / 3;

        sphereVertexTextureCoordBuffer = gl.createBuffer();
        gl.bindBuffer(gl.ARRAY_BUFFER, sphereVertexTextureCoordBuffer);
        gl.bufferData(gl.ARRAY_BUFFER,
            new Float32Array(textureCoordData), gl.STATIC_DRAW);
        sphereVertexTextureCoordBuffer.itemSize = 2;
        sphereVertexTextureCoordBuffer.numItems = textureCoordData.length / 2;

        sphereVertexPositionBuffer = gl.createBuffer();
        gl.bindBuffer(gl.ARRAY_BUFFER, sphereVertexPositionBuffer);
        gl.bufferData(gl.ARRAY_BUFFER,
            new Float32Array(vertexPositionData), gl.STATIC_DRAW);
        sphereVertexPositionBuffer.itemSize = 3;
        sphereVertexPositionBuffer.numItems = vertexPositionData.length / 3;

        sphereVertexIndexBuffer = gl.createBuffer();
        gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, sphereVertexIndexBuffer);
        gl.bufferData(gl.ELEMENT_ARRAY_BUFFER,
            new Uint16Array(indexData), gl.STATIC_DRAW);
        sphereVertexIndexBuffer.itemSize = 1;
        sphereVertexIndexBuffer.numItems = indexData.length;
    }

    var earthAngle = 180;

    function drawScene()
    {
        gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
        gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

        mat4.perspective(pMatrix, 45,
            gl.viewportWidth / gl.viewportHeight, 0.1, 100.0);

        var useColorMap  = $("#color-map").is(":checked");
        gl.uniform1i(shaderProgram.useColorMapUniform, useColorMap);

        var useSpecularMap = $("#specular-map").is(":checked");
        gl.uniform1i(shaderProgram.useSpecularMapUniform, useSpecularMap );

        var lighting = $("#lighting").is(":checked");
        gl.uniform1i(shaderProgram.useLightingUniform, lighting);
        if(lighting)
```

```
    {
        gl.uniform3f(
            shaderProgram.ambientColorUniform,
            parseFloat($("#ambientR").val()),
            parseFloat($("#ambientG").val()),
            parseFloat($("#ambientB").val())
            );
        gl.uniform3f(
            shaderProgram.pointLightingLocationUniform,
            parseFloat($("#lightPositionX").val()),
            parseFloat($("#lightPositionY").val()),
            parseFloat($("#lightPositionZ").val())
        );

        gl.uniform3f(
            shaderProgram.pointLightingSpecularColorUniform,
            parseFloat($("#specularR").val()),
            parseFloat($("#specularG").val()),
            parseFloat($("#specularB").val())
        );

        gl.uniform3f(
            shaderProgram.pointLightingDiffuseColorUniform,
            parseFloat($("#diffuseR").val()),
            parseFloat($("#diffuseG").val()),
            parseFloat($("#diffuseB").val())
        );

    }

    mat4.identity(mvMatrix);
    mat4.translate(mvMatrix, mvMatrix, [0, 0, -40]);
    mat4.rotate(mvMatrix, mvMatrix, degToRad(23.4), [1, 0, -1]);
    mat4.rotate(mvMatrix, mvMatrix, degToRad(earthAngle), [0, 1, 0]);

    gl.activeTexture(gl.TEXTURE0);
    gl.bindTexture(gl.TEXTURE_2D, earthColorMapTexture);
    gl.uniform1i(shaderProgram.specularMapSamplerUniform, 0);

    gl.activeTexture(gl.TEXTURE1);
    gl.bindTexture(gl.TEXTURE_2D, earthSpecularMapTexture);
    gl.uniform1i(shaderProgram.specularMapSamplerUniform, 1);

    gl.bindBuffer(gl.ARRAY_BUFFER, sphereVertexPositionBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
        sphereVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, sphereVertexNormalBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexNormalAttribute,
        sphereVertexNormalBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, sphereVertexTextureCoordBuffer);
    gl.vertexAttribPointer(shaderProgram.textureCoordAttribute,
        sphereVertexTextureCoordBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, sphereVertexIndexBuffer);
    setMatrixUniforms();
    gl.drawElements(gl.TRIANGLES,
        sphereVertexIndexBuffer.numItems, gl.UNSIGNED_SHORT, 0);
```

```
}

function degToRad(degrees)
{
    return degrees * Math.PI / 180;
}
var lastTime = 0;
function animate()
{
    var timeNow = new Date().getTime();
    if(lastTime != 0)
    {
        var elapsed = timeNow - lastTime;

         earthAngle += 0.05 * elapsed;
    }
    lastTime = timeNow;
}
</script>
```

渲染到纹理

```
<div class="page-header"><h3>16、渲染到纹理</h3></div>

<canvas id = "test16-canvas" width = "800" height = "600"></canvas>
<br/>
笔记本模型来自：
<a href="http://www.turbosquid.com/3d-models/apple-macbook-max-free/391534">
    this 3DS Max model by Xedium
</a>
<br/>
月球表面纹理图片来自：
<a href="http://maps.jpl.nasa.gov/">the Jet Propulsion Laboratory</a>.
<br/>

<script id = "per-fragment-lighting-vs" type = "x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    attribute vec3 aVertexNormal;
    attribute vec2 aTextureCoord;

    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;
    uniform mat3 uNMatrix;

    varying vec2 vTextureCoord;
    varying vec3 vTransformedNormal;
    varying vec4 vPosition;


    void main(void)
    {
        vPosition = uMVMatrix * vec4(aVertexPosition, 1.0);
        gl_Position = uPMatrix * vPosition;
        vTextureCoord = aTextureCoord;
        vTransformedNormal = uNMatrix * aVertexNormal;
    }
</script>
<script id = "per-fragment-lighting-fs" type = "x-shader/x-fragment">
    precision mediump float;

    varying vec2 vTextureCoord;
    varying vec3 vTransformedNormal;
    varying vec4 vPosition;

    uniform vec3 uMaterialAmbientColor;
    uniform vec3 uMaterialDiffuseColor;
    uniform vec3 uMaterialSpecularColor;
    uniform float uMaterialShininess;
    uniform vec3 uMaterialEmissiveColor;

    uniform bool uShowSpecularHighlights;
    uniform bool uUseTextures;

    uniform vec3 uAmbientLightingColor;

    uniform vec3 uPointLightingLocation;
```

```
    uniform vec3 uPointLightingDiffuseColor;
    uniform vec3 uPointLightingSpecularColor;

    uniform sampler2D uSampler;


    void main(void)
    {
        vec3 ambientLightWeighting = uAmbientLightingColor;

        vec3 lightDirection = normalize(uPointLightingLocation - vPosition.xyz);
        vec3 normal = normalize(vTransformedNormal);

        vec3 specularLightWeighting = vec3(0.0, 0.0, 0.0);
        if (uShowSpecularHighlights)
        {
            vec3 eyeDirection = normalize(-vPosition.xyz);
            vec3 reflectionDirection = reflect(-lightDirection, normal);

            float specularLightBrightness =
                pow(max(dot(reflectionDirection, eyeDirection), 0.0),
                uMaterialShininess);
            specularLightWeighting =
                uPointLightingSpecularColor * specularLightBrightness;
        }

        float diffuseLightBrightness = max(dot(normal, lightDirection), 0.0);
        vec3 diffuseLightWeighting =
            uPointLightingDiffuseColor * diffuseLightBrightness;

        vec3 materialAmbientColor = uMaterialAmbientColor;
        vec3 materialDiffuseColor = uMaterialDiffuseColor;
        vec3 materialSpecularColor = uMaterialSpecularColor;
        vec3 materialEmissiveColor = uMaterialEmissiveColor;
        float alpha = 1.0;
        if (uUseTextures)
        {
            vec4 textureColor =
                texture2D(uSampler, vec2(vTextureCoord.s, vTextureCoord.t));
            materialAmbientColor = materialAmbientColor * textureColor.rgb;
            materialDiffuseColor = materialDiffuseColor * textureColor.rgb;
            materialEmissiveColor = materialEmissiveColor * textureColor.rgb;
            alpha = textureColor.a;
        }
        gl_FragColor = vec4(
                            materialAmbientColor * ambientLightWeighting +
                            materialDiffuseColor * diffuseLightWeighting +
                            materialSpecularColor * specularLightWeighting +
                            materialEmissiveColor,
                            alpha
                        );
    }
</script>
<script type="text/javascript">
    requestAnimFrame = window.requestAnimationFrame ||
        window.mozRequestAnimationFrame ||
        window.webkitRequestAnimationFrame ||
        window.msRequestAnimationFrame ||
        window.oRequestAnimationFrame ||
```

```
        function(callback) { setTimeout(callback, 1000 / 60); };
</script>
<script type="text/javascript">
$(document).ready(function ()
{
    webGLStart();
});

function webGLStart()
{
    var canvas = $("#test16-canvas");
    initGL(canvas[0]);
    initTextureFramebuffer();
    initShaders();
    initTextures();
    initBuffers();

    loadLaptop();

    gl.clearColor(0.0, 0.0, 0.0, 1.0);
    gl.enable(gl.DEPTH_TEST);

    waitTexture = setInterval("tick()", 100);
}

var textureFlag = [0, 0, 0];
var startTick = false;
var waitTick;
function tick()
{
    if(textureFlag[0] == 1 && textureFlag[1] == 1 && textureFlag[2] == 1)
    {
        startTick = true;
        textureFlag[0] = 2;
        textureFlag[1] = 2;
        textureFlag[2] = 2;
        clearInterval(waitTexture);
    }
    if(startTick)
    {
        requestAnimFrame(tick);
        drawScene();
        animate();
    }
}
var gl;
function initGL(canvas)
{
    try
    {
        gl = canvas.getContext("webgl") || canvas.getContext("experimental-webgl");
        gl.viewportWidth = canvas.width;
        gl.viewportHeight = canvas.height;
    }catch(e){}
    if(!gl)
    {
        alert("无法初始化“WebGL”。");
    }
}
```

```
var rttFramebuffer;
var rttTexture;
function initTextureFramebuffer()
{
    rttFramebuffer = gl.createFramebuffer();
    gl.bindFramebuffer(gl.FRAMEBUFFER, rttFramebuffer);
    rttFramebuffer.width = 512;
    rttFramebuffer.height = 512;

    rttTexture = gl.createTexture();
    gl.bindTexture(gl.TEXTURE_2D, rttTexture);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
    // gl.generateMipmap(gl.TEXTURE_2D);

    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA,
        rttFramebuffer.width, rttFramebuffer.height,
        0, gl.RGBA, gl.UNSIGNED_BYTE, null);

    var renderbuffer = gl.createRenderbuffer();
    gl.bindRenderbuffer(gl.RENDERBUFFER, renderbuffer);
    gl.renderbufferStorage(gl.RENDERBUFFER, gl.DEPTH_COMPONENT16,
        rttFramebuffer.width, rttFramebuffer.height);

    gl.framebufferTexture2D(
        gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0, gl.TEXTURE_2D, rttTexture, 0);
    gl.framebufferRenderbuffer(
        gl.FRAMEBUFFER, gl.DEPTH_ATTACHMENT, gl.RENDERBUFFER, renderbuffer);

    gl.bindTexture(gl.TEXTURE_2D, null);
    gl.bindRenderbuffer(gl.RENDERBUFFER, null);
    gl.bindFramebuffer(gl.FRAMEBUFFER, null);
}
var moonTexture;
var crateTexture;
function initTextures()
{
    moonTexture = gl.createTexture();
    moonTexture.image = new Image();
    moonTexture.image.onload = function ()
    {
        handleLoadedTexture(moonTexture)
        textureFlag[0] = 1;
    }
    moonTexture.image.src = "/Public/image/moon.gif";

    crateTexture  = gl.createTexture();
    crateTexture.image = new Image();
    crateTexture.image.onload = function ()
    {
        handleLoadedTexture(crateTexture)
        textureFlag[1] = 1;
    }
    crateTexture.image.src = "/Public/image/crate.gif";
}
function handleLoadedTexture(texture)
{
    gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, 1);
```

```
    gl.bindTexture(gl.TEXTURE_2D, texture);
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA,
        gl.UNSIGNED_BYTE, texture.image);
    gl.texParameteri(gl.TEXTURE_2D,
        gl.TEXTURE_MAG_FILTER, gl.LINEAR);
    gl.texParameteri(gl.TEXTURE_2D,
        gl.TEXTURE_MIN_FILTER, gl.LINEAR_MIPMAP_NEAREST);

    gl.generateMipmap(gl.TEXTURE_2D);

    gl.bindTexture(gl.TEXTURE_2D, null);
}
function getShader(gl, id)
{
    var shaderScript = $("#" + id);
    if(!shaderScript.length)
    {
        return null;
    }
    var str = shaderScript.text();

    var shader;
    if(shaderScript[0].type == "x-shader/x-fragment")
    {
        shader = gl.createShader(gl.FRAGMENT_SHADER);
    }
    else if(shaderScript[0].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;
}

var shaderProgram;

function initShaders()
{
    var vertexShader = getShader(gl, "per-fragment-lighting-vs");
    var fragmentShader = getShader(gl, "per-fragment-lighting-fs");

    shaderProgram  = gl.createProgram();
    gl.attachShader(shaderProgram, vertexShader);
    gl.attachShader(shaderProgram, fragmentShader);
    gl.linkProgram (shaderProgram );
    if(!gl.getProgramParameter(shaderProgram, gl.LINK_STATUS))
    {
        alert("无法初始化“Shader”。");
```

```
    }
    gl.useProgram(shaderProgram);
    shaderProgram.vertexPositionAttribute =
        gl.getAttribLocation(shaderProgram, "aVertexPosition");
    gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);

    shaderProgram.vertexNormalAttribute =
        gl.getAttribLocation(shaderProgram, "aVertexNormal");
    gl.enableVertexAttribArray(shaderProgram.vertexNormalAttribute);

    shaderProgram.textureCoordAttribute =
        gl.getAttribLocation(shaderProgram, "aTextureCoord");
    gl.enableVertexAttribArray(shaderProgram.textureCoordAttribute);

    shaderProgram.pMatrixUniform =
        gl.getUniformLocation(shaderProgram, "uPMatrix");
    shaderProgram.mvMatrixUniform =
        gl.getUniformLocation(shaderProgram, "uMVMatrix");
    shaderProgram.nMatrixUniform  =
        gl.getUniformLocation(shaderProgram, "uNMatrix");
    shaderProgram.samplerUniform =
        gl.getUniformLocation(shaderProgram, "uSampler");

    shaderProgram.materialAmbientColorUniform =
        gl.getUniformLocation(shaderProgram, "uMaterialAmbientColor");
    shaderProgram.materialDiffuseColorUniform =
        gl.getUniformLocation(shaderProgram, "uMaterialDiffuseColor");
    shaderProgram.materialSpecularColorUniform =
        gl.getUniformLocation(shaderProgram, "uMaterialSpecularColor");
    shaderProgram.materialShininessUniform =
        gl.getUniformLocation(shaderProgram, "uMaterialShininess");
    shaderProgram.materialEmissiveColorUniform =
        gl.getUniformLocation(shaderProgram, "uMaterialEmissiveColor");
    shaderProgram.showSpecularHighlightsUniform =
        gl.getUniformLocation(shaderProgram, "uShowSpecularHighlights");
    shaderProgram.useTexturesUniform =
        gl.getUniformLocation(shaderProgram, "uUseTextures");
    shaderProgram.ambientLightingColorUniform =
        gl.getUniformLocation(shaderProgram, "uAmbientLightingColor");
    shaderProgram.pointLightingLocationUniform =
        gl.getUniformLocation(shaderProgram, "uPointLightingLocation");
    shaderProgram.pointLightingSpecularColorUniform =
        gl.getUniformLocation(shaderProgram, "uPointLightingSpecularColor");
    shaderProgram.pointLightingDiffuseColorUniform =
        gl.getUniformLocation(shaderProgram, "uPointLightingDiffuseColor");
}

var mvMatrix = mat4.create();

var mvMatrixStack = [];

var pMatrix = mat4.create();

function mvPushMatrix()
{
    var copy = mat4.clone(mvMatrix);
    mvMatrixStack.push(copy);
}
function mvPopMatrix()
```

```
{
    if(mvMatrixStack.length == 0)
    {
        throw "不合法的矩阵出栈操作!";
    }
    mvMatrix = mvMatrixStack.pop();
}

function setMatrixUniforms()
{
    gl.uniformMatrix4fv(shaderProgram.pMatrixUniform, false, pMatrix);
    gl.uniformMatrix4fv(shaderProgram.mvMatrixUniform, false, mvMatrix);

    var normalMatrix = mat3.create();

    mat3.fromMat4(normalMatrix, mvMatrix);
    mat3.invert(normalMatrix, normalMatrix);
    mat3.transpose(normalMatrix, normalMatrix);

    gl.uniformMatrix3fv(shaderProgram.nMatrixUniform, false, normalMatrix);
}

var laptopVertexPositionBuffer;
var laptopVertexNormalBuffer;
var laptopVertexTextureCoordBuffer;
var laptopVertexIndexBuffer;
function loadLaptop()
{
    $.getJSON(
        "/Public/json/macbook.json",
        function(data)
        {
            handleLoadedLaptop(data);
            textureFlag[2] = 1;
        }
    );
}
function handleLoadedLaptop(laptopData)
{
    laptopVertexNormalBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, laptopVertexNormalBuffer);
    gl.bufferData(gl.ARRAY_BUFFER,
        new Float32Array(laptopData.vertexNormals), gl.STATIC_DRAW);
    laptopVertexNormalBuffer.itemSize = 3;
    laptopVertexNormalBuffer.numItems = laptopData.vertexNormals.length / 3;

    laptopVertexTextureCoordBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, laptopVertexTextureCoordBuffer);
    gl.bufferData(gl.ARRAY_BUFFER,
        new Float32Array(laptopData.vertexTextureCoords), gl.STATIC_DRAW);
    laptopVertexTextureCoordBuffer.itemSize = 2;
    laptopVertexTextureCoordBuffer.numItems =
    laptopData.vertexTextureCoords.length / 2;

    laptopVertexPositionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, laptopVertexPositionBuffer);
    gl.bufferData(gl.ARRAY_BUFFER,
        new Float32Array(laptopData.vertexPositions), gl.STATIC_DRAW);
    laptopVertexPositionBuffer.itemSize = 3;
```

```
    laptopVertexPositionBuffer.numItems = laptopData.vertexPositions.length / 3;

    laptopVertexIndexBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, laptopVertexIndexBuffer);
    gl.bufferData(gl.ELEMENT_ARRAY_BUFFER,
        new Uint16Array(laptopData.indices), gl.STREAM_DRAW);
    laptopVertexIndexBuffer.itemSize = 1;
    laptopVertexIndexBuffer.numItems = laptopData.indices.length;
}

var cubeVertexPositionBuffer;
var cubeVertexNormalBuffer;
var cubeVertexTextureCoordBuffer;
var cubeVertexIndexBuffer;

var moonVertexPositionBuffer;
var moonVertexNormalBuffer;
var moonVertexTextureCoordBuffer;
var moonVertexIndexBuffer;

var laptopScreenVertexPositionBuffer;
var laptopScreenVertexNormalBuffer;
var laptopScreenVertexTextureCoordBuffer;

function initBuffers()
{
    //箱子
    cubeVertexPositionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
    vertices = [
                // 正面
                -1.0, -1.0,  1.0,
                 1.0, -1.0,  1.0,
                 1.0,  1.0,  1.0,
                -1.0,  1.0,  1.0,

                // 背面
                -1.0, -1.0, -1.0,
                -1.0,  1.0, -1.0,
                 1.0,  1.0, -1.0,
                 1.0, -1.0, -1.0,

                // 顶部
                -1.0,  1.0, -1.0,
                -1.0,  1.0,  1.0,
                 1.0,  1.0,  1.0,
                 1.0,  1.0, -1.0,

                // 底部
                -1.0, -1.0, -1.0,
                 1.0, -1.0, -1.0,
                 1.0, -1.0,  1.0,
                -1.0, -1.0,  1.0,

                // 右侧面
                 1.0, -1.0, -1.0,
                 1.0,  1.0, -1.0,
                 1.0,  1.0,  1.0,
                 1.0, -1.0,  1.0,
```

```
                // 左侧面
                -1.0, -1.0, -1.0,
                -1.0, -1.0,  1.0,
                -1.0,  1.0,  1.0,
                -1.0,  1.0, -1.0,
                ];
    gl.bufferData(gl.ARRAY_BUFFER,
        new Float32Array(vertices), gl.STATIC_DRAW);
    cubeVertexPositionBuffer.itemSize = 3;
    cubeVertexPositionBuffer.numItems = 24;

    cubeVertexNormalBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexNormalBuffer);
    var vertexNormals = [
                    // 正面
                      0.0,  0.0,  1.0,
                      0.0,  0.0,  1.0,
                      0.0,  0.0,  1.0,
                      0.0,  0.0,  1.0,

                    // 背面
                      0.0,  0.0, -1.0,
                      0.0,  0.0, -1.0,
                      0.0,  0.0, -1.0,
                      0.0,  0.0, -1.0,

                    // 顶部
                      0.0,  1.0,  0.0,
                      0.0,  1.0,  0.0,
                      0.0,  1.0,  0.0,
                      0.0,  1.0,  0.0,

                    // 底部
                      0.0, -1.0,  0.0,
                      0.0, -1.0,  0.0,
                      0.0, -1.0,  0.0,
                      0.0, -1.0,  0.0,

                    // 右侧面
                      1.0,  0.0,  0.0,
                      1.0,  0.0,  0.0,
                      1.0,  0.0,  0.0,
                      1.0,  0.0,  0.0,

                    // 左侧面
                    -1.0,  0.0,  0.0,
                    -1.0,  0.0,  0.0,
                    -1.0,  0.0,  0.0,
                    -1.0,  0.0,  0.0,
                  ];
    gl.bufferData(gl.ARRAY_BUFFER,
        new Float32Array(vertexNormals), gl.STATIC_DRAW);
    cubeVertexNormalBuffer.itemSize = 3;
    cubeVertexNormalBuffer.numItems = 24;

    cubeVertexTextureCoordBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexTextureCoordBuffer);
    textureCoords = [
```

```
                    // 正面
                    0.0, 0.0,
                    1.0, 0.0,
                    1.0, 1.0,
                    0.0, 1.0,

                    // 背面
                    1.0, 0.0,
                    1.0, 1.0,
                    0.0, 1.0,
                    0.0, 0.0,

                    // 顶部
                    0.0, 1.0,
                    0.0, 0.0,
                    1.0, 0.0,
                    1.0, 1.0,

                    // 底部
                    1.0, 1.0,
                    0.0, 1.0,
                    0.0, 0.0,
                    1.0, 0.0,

                    // 右侧面
                    1.0, 0.0,
                    1.0, 1.0,
                    0.0, 1.0,
                    0.0, 0.0,

                    // 左侧面
                    0.0, 0.0,
                    1.0, 0.0,
                    1.0, 1.0,
                    0.0, 1.0,
                   ];
    gl.bufferData(gl.ARRAY_BUFFER,
        new Float32Array(textureCoords), gl.STATIC_DRAW);
    cubeVertexTextureCoordBuffer.itemSize = 2;
    cubeVertexTextureCoordBuffer.numItems = 24;

    cubeVertexIndexBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);
    var cubeVertexIndices =
        [
          0, 1, 2,        0, 2, 3,    // 正面
          4, 5, 6,        4, 6, 7,    // 背面
          8, 9, 10,        8, 10, 11,  // 顶部
          12, 13, 14,    12, 14, 15, // 底部
          16, 17, 18,    16, 18, 19, // 右侧面
          20, 21, 22,    20, 22, 23  // 左侧面
        ];
    gl.bufferData(gl.ELEMENT_ARRAY_BUFFER,
        new Uint16Array(cubeVertexIndices), gl.STATIC_DRAW);
    cubeVertexIndexBuffer.itemSize = 1;
    cubeVertexIndexBuffer.numItems = 36;

    //月亮
    var latitudeBands = 30;
```

```
    var longitudeBands = 30;
    var radius = 1;
    var vertexPositionData = [];
    var normalData = [];
    var textureCoordData = [];
    for(var latNumber = 0; latNumber <= latitudeBands; latNumber ++)
    {
        var theta = latNumber * Math.PI / latitudeBands;
        var sinTheta = Math.sin(theta);
        var cosTheta = Math.cos(theta);
        for(var longNumber = 0; longNumber <= longitudeBands; longNumber ++)
        {
            var phi = longNumber * 2 * Math.PI / longitudeBands;
            var sinPhi = Math.sin(phi);
            var cosPhi = Math.cos(phi);
            var x = cosPhi * sinTheta;
            var y = cosTheta;
            var z = sinPhi * sinTheta;
            var u = 1 - (longNumber / longitudeBands);
            var v = 1 - (latNumber / latitudeBands);

            normalData.push(x);
            normalData.push(y);
            normalData.push(z);
            textureCoordData.push(u);
            textureCoordData.push(v);
            vertexPositionData.push(radius * x);
            vertexPositionData.push(radius * y);
            vertexPositionData.push(radius * z);
        }
    }
    var indexData = [];
    for(var latNumber = 0; latNumber < latitudeBands; latNumber ++)
    {
        for(var longNumber = 0; longNumber < longitudeBands; longNumber ++)
        {
            var first = (latNumber * (longitudeBands + 1)) + longNumber;
            var second = first + longitudeBands + 1;
            indexData.push(first);
            indexData.push(second);
            indexData.push(first + 1);
            indexData.push(second);
            indexData.push(second + 1);
            indexData.push(first + 1);
        }
    }
    moonVertexNormalBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, moonVertexNormalBuffer);
    gl.bufferData(gl.ARRAY_BUFFER,
        new Float32Array(normalData), gl.STATIC_DRAW);
    moonVertexNormalBuffer.itemSize = 3;
    moonVertexNormalBuffer.numItems = normalData.length / 3;

    moonVertexTextureCoordBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, moonVertexTextureCoordBuffer);
    gl.bufferData(gl.ARRAY_BUFFER,
        new Float32Array(textureCoordData), gl.STATIC_DRAW);
    moonVertexTextureCoordBuffer.itemSize = 2;
    moonVertexTextureCoordBuffer.numItems = textureCoordData.length / 2;
```

```
    moonVertexPositionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, moonVertexPositionBuffer);
    gl.bufferData(gl.ARRAY_BUFFER,
        new Float32Array(vertexPositionData), gl.STATIC_DRAW);
    moonVertexPositionBuffer.itemSize = 3;
    moonVertexPositionBuffer.numItems = vertexPositionData.length / 3;

    moonVertexIndexBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, moonVertexIndexBuffer);
    gl.bufferData(gl.ELEMENT_ARRAY_BUFFER,
        new Uint16Array(indexData), gl.STATIC_DRAW);
    moonVertexIndexBuffer.itemSize = 1;
    moonVertexIndexBuffer.numItems = indexData.length;

    laptopScreenVertexPositionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, laptopScreenVertexPositionBuffer);
    vertices = [
         0.580687, 0.659, 0.813106,
        -0.580687, 0.659, 0.813107,
         0.580687, 0.472, 0.113121,
        -0.580687, 0.472, 0.113121,
        ];
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices), gl.STATIC_DRAW);
    laptopScreenVertexPositionBuffer.itemSize = 3;
    laptopScreenVertexPositionBuffer.numItems = 4;

    laptopScreenVertexNormalBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, laptopScreenVertexNormalBuffer);
    var vertexNormals = [
         0.000000, -0.965926, 0.258819,
         0.000000, -0.965926, 0.258819,
         0.000000, -0.965926, 0.258819,
         0.000000, -0.965926, 0.258819,
    ];
    gl.bufferData(gl.ARRAY_BUFFER,
        new Float32Array(vertexNormals), gl.STATIC_DRAW);
    laptopScreenVertexNormalBuffer.itemSize = 3;
    laptopScreenVertexNormalBuffer.numItems = 4;

    laptopScreenVertexTextureCoordBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, laptopScreenVertexTextureCoordBuffer);
    var textureCoords = [
        1.0, 1.0,
        0.0, 1.0,
        1.0, 0.0,
        0.0, 0.0,
    ];
    gl.bufferData(gl.ARRAY_BUFFER,
        new Float32Array(textureCoords), gl.STATIC_DRAW);
    laptopScreenVertexTextureCoordBuffer.itemSize = 2;
    laptopScreenVertexTextureCoordBuffer.numItems = 4;
}
var laptopScreenAspectRatio = 1.66;

var moonAngle = 180;
var cubeAngle = 0;
function drawSceneOnLaptopScreen()
{
```

```
    gl.viewport(0, 0, rttFramebuffer.width, rttFramebuffer.height);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    mat4.perspective(pMatrix, 45, laptopScreenAspectRatio, 0.1, 100.0);

    gl.uniform1i(shaderProgram.showSpecularHighlightsUniform, false);
    gl.uniform3f(shaderProgram.ambientLightingColorUniform, 0.2, 0.2, 0.2);
    gl.uniform3f(shaderProgram.pointLightingLocationUniform, 0, 0, -5);
    gl.uniform3f(shaderProgram.pointLightingDiffuseColorUniform, 0.8, 0.8, 0.8);

    gl.uniform1i(shaderProgram.showSpecularHighlightsUniform, false);
    gl.uniform1i(shaderProgram.useTexturesUniform, true);

    gl.uniform3f(shaderProgram.materialAmbientColorUniform, 1.0, 1.0, 1.0);
    gl.uniform3f(shaderProgram.materialDiffuseColorUniform, 1.0, 1.0, 1.0);
    gl.uniform3f(shaderProgram.materialSpecularColorUniform, 0.0, 0.0, 0.0);
    gl.uniform1f(shaderProgram.materialShininessUniform, 0);
    gl.uniform3f(shaderProgram.materialEmissiveColorUniform, 0.0, 0.0, 0.0);

    mat4.identity(mvMatrix);

    mat4.translate(mvMatrix, mvMatrix, [0, 0, -5]);
    mat4.rotate(mvMatrix, mvMatrix, degToRad(30), [1, 0, 0]);

    mvPushMatrix();
    mat4.rotate(mvMatrix, mvMatrix, degToRad(moonAngle), [0, 1, 0]);
    mat4.translate(mvMatrix, mvMatrix, [2, 0, 0]);
    gl.activeTexture(gl.TEXTURE0);
    gl.bindTexture(gl.TEXTURE_2D, moonTexture);
    gl.uniform1i(shaderProgram.samplerUniform, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, moonVertexPositionBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
        moonVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, moonVertexTextureCoordBuffer);
    gl.vertexAttribPointer(shaderProgram.textureCoordAttribute,
        moonVertexTextureCoordBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, moonVertexNormalBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexNormalAttribute,
        moonVertexNormalBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, moonVertexIndexBuffer);
    setMatrixUniforms();
    gl.drawElements(gl.TRIANGLES,
        moonVertexIndexBuffer.numItems, gl.UNSIGNED_SHORT, 0);
    mvPopMatrix();


    mvPushMatrix();
    mat4.rotate(mvMatrix, mvMatrix, degToRad(cubeAngle), [0, 1, 0]);
    mat4.translate(mvMatrix, mvMatrix, [1.25, 0, 0]);
    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
        cubeVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexNormalBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexNormalAttribute,
```

```
        cubeVertexNormalBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexTextureCoordBuffer);
    gl.vertexAttribPointer(shaderProgram.textureCoordAttribute,
        cubeVertexTextureCoordBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.activeTexture(gl.TEXTURE0);
    gl.bindTexture(gl.TEXTURE_2D, crateTexture);
    gl.uniform1i(shaderProgram.samplerUniform, 0);

    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);
    setMatrixUniforms();
    gl.drawElements(gl.TRIANGLES,
        cubeVertexIndexBuffer.numItems, gl.UNSIGNED_SHORT, 0);
    mvPopMatrix();

    gl.bindTexture(gl.TEXTURE_2D, rttTexture);
    gl.generateMipmap(gl.TEXTURE_2D);
    gl.bindTexture(gl.TEXTURE_2D, null);
}
var laptopAngle = 0;

function drawScene()
{
    gl.bindFramebuffer(gl.FRAMEBUFFER, rttFramebuffer);
    drawSceneOnLaptopScreen();
    gl.bindFramebuffer(gl.FRAMEBUFFER, null);

    gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    mat4.perspective(pMatrix, 45,
        gl.viewportWidth / gl.viewportHeight, 0.1, 100.0);

    mat4.identity(mvMatrix);

    mvPushMatrix();

    mat4.translate(mvMatrix, mvMatrix, [0, -0.4, -2.2]);
    mat4.rotate(mvMatrix, mvMatrix, degToRad(laptopAngle), [0, 1, 0]);
    mat4.rotate(mvMatrix, mvMatrix, degToRad(-90), [1, 0, 0]);

    gl.uniform1i(shaderProgram.showSpecularHighlightsUniform, true);
    gl.uniform3f(shaderProgram.pointLightingLocationUniform, -1, 2, -1);
    gl.uniform3f(shaderProgram.ambientLightingColorUniform, -1, 2, -1);

    gl.uniform3f(shaderProgram.ambientLightingColorUniform, 0.2, 0.2, 0.2);
    gl.uniform3f(shaderProgram.pointLightingDiffuseColorUniform, 0.8, 0.8, 0.8);
    gl.uniform3f(shaderProgram.pointLightingSpecularColorUniform, 0.8, 0.8, 0.8);


    gl.uniform3f(shaderProgram.materialAmbientColorUniform, 1.0, 1.0, 1.0);
    gl.uniform3f(shaderProgram.materialDiffuseColorUniform, 1.0, 1.0, 1.0);
    gl.uniform3f(shaderProgram.materialSpecularColorUniform, 1.5, 1.5, 1.5);
    gl.uniform1f(shaderProgram.materialShininessUniform, 5);
    gl.uniform3f(shaderProgram.materialEmissiveColorUniform, 0.0, 0.0, 0.0);
    gl.uniform1i(shaderProgram.useTexturesUniform, false);

    if (laptopVertexPositionBuffer)
```

```
    {
        gl.bindBuffer(gl.ARRAY_BUFFER, laptopVertexPositionBuffer);
        gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
            laptopVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);

        gl.bindBuffer(gl.ARRAY_BUFFER, laptopVertexTextureCoordBuffer);
        gl.vertexAttribPointer(shaderProgram.textureCoordAttribute,
            laptopVertexTextureCoordBuffer.itemSize, gl.FLOAT, false, 0, 0);

        gl.bindBuffer(gl.ARRAY_BUFFER, laptopVertexNormalBuffer);
        gl.vertexAttribPointer(shaderProgram.vertexNormalAttribute,
            laptopVertexNormalBuffer.itemSize, gl.FLOAT, false, 0, 0);

        gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, laptopVertexIndexBuffer);
        setMatrixUniforms();
        gl.drawElements(gl.TRIANGLES,
            laptopVertexIndexBuffer.numItems, gl.UNSIGNED_SHORT, 0);
    }

    gl.uniform3f(shaderProgram.materialAmbientColorUniform, 0.0, 0.0, 0.0);
    gl.uniform3f(shaderProgram.materialDiffuseColorUniform, 0.0, 0.0, 0.0);
    gl.uniform3f(shaderProgram.materialSpecularColorUniform, 0.5, 0.5, 0.5);
    gl.uniform1f(shaderProgram.materialShininessUniform, 20);
    gl.uniform3f(shaderProgram.materialEmissiveColorUniform, 1.5, 1.5, 1.5);
    gl.uniform1i(shaderProgram.useTexturesUniform, true);

    gl.bindBuffer(gl.ARRAY_BUFFER, laptopScreenVertexPositionBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
        laptopScreenVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, laptopScreenVertexNormalBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexNormalAttribute,
        laptopScreenVertexNormalBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, laptopScreenVertexTextureCoordBuffer);
    gl.vertexAttribPointer(shaderProgram.textureCoordAttribute,
        laptopScreenVertexTextureCoordBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.activeTexture(gl.TEXTURE0);
    gl.bindTexture(gl.TEXTURE_2D, rttTexture);
    gl.uniform1i(shaderProgram.samplerUniform, 0);

    setMatrixUniforms();
    gl.drawArrays(gl.TRIANGLE_STRIP, 0,
        laptopScreenVertexPositionBuffer.numItems);

    mvPopMatrix();
}

function degToRad(degrees)
{
    return degrees * Math.PI / 180;
}
var lastTime = 0;
function animate()
{
    var timeNow = new Date().getTime();
    if(lastTime != 0)
    {
```

```
            var elapsed = timeNow - lastTime;

            moonAngle += 0.05 * elapsed;
            cubeAngle += 0.05 * elapsed;

            laptopAngle -= 0.005 * elapsed;
        }
        lastTime = timeNow;
    }
    </script>
```