# 第5课 动画

本节课要介绍的内容是：

* 使用requestAnimationFrame()方法实现动画；
* 创建FPS计数器。

程序截图如下：



其实就是把[Direct3D 10教程5：3D变换](http://shiba.hpe.cn/jiaoyanzu/wuli/showArticle.aspx?articleId=946&classId=5)的代码移植到了WebGL平台。

为了使模型在场景中移动，需要重复绘制场景，而且在两次绘制场景间隔，需要更新动画模型的位置。

## setInterval()和setTimeout()的使用

传统上，基于脚本的动画一直以来都是用setTimeout()和setlnterval()这两个JavaScript方法之一创建的，Canvas2D用的就是这种方法，可参见[21 创建动画循环](http://shiba.hpe.cn/jiaoyanzu/wuli/showArticle.aspx?articleId=1076&classId=5)。这两个方法用来定义回调函数，我们希望在未来规定的时间（单位为毫秒）内调用这些回调函数。在回调函数中更新动画，然后再次绘制。重复地执行此过程就可以生成一个动画。这两个方法函数原型如下：

setTimeout(codeToCall,timeoutInMilliseconds)

setInterval(codeToCall, timeoutlnMillisecond)

setTimeout()方法在规定的时间后（单位为毫秒，由它的第二个参数确定）调用第一个参数表示的代码或函数。这个方法有一个返回值，如果把这个返回值作为参数传给clearInterval()函数，则可取消回调函数或代码的调用。

setInterval()方法与setTimeout()方法相似，但是它每隔一定的时间（由第二个参数定义）反复调用第一个参数中的代码或函数。此外，setInterval()方法也返回一个值，如果把这个值作为参数送给clearInterval()函数，则可以取消回调函数或代码的执行。

例如下面这个例子，展示了一个非常简单的绘图循环结构，它建立在setInterval()之上：

fuction draw(){

 // 绘制代码

}

function startup(){

 setInterval(draw,16.7);

}

startup()通常是在载文档后调用的（或者在我们希望启洞动画的任何时刻调用）。在该函数中调用setInterval()，并规定在一定的时间间隔内（本例中的16.7毫秒）调用draw()函数。在这个示例中之所以选择16.7毫秒是因为通常LCD显示器的刷新频率不会大于60Hz，根据这个值就可以得到帧时间的近似值，即1/60Hz＝0.0167秒＝16.7毫秒。

## 使用requestAnimationFrame()函数

requestAnimationFrame()函数是专门为创建脚本式动画而设计的。有了一个专用的动画创建方法，浏览器供应商可以努力优化这个方法，使它比传统的setInterval()和setTimeout()方法更适合于创建动画。

用setlnterval()或setTimeout()方法创建动画时，需要确定动画更新的最佳频率。但是这个最佳频率对于动画的的设计人员来说是很难确定的。

但是，浏览器是比较容易确定这个最佳频率的。浏览器上可能会同时运行多个动画，这可能会影响这个帧频。在这种情形下，浏览器会降低所有动画的帧频，这样它们就以流畅但稍低的频率执行动画。

浏览器也可以减速更新或暂停更新在某个不可见选项卡中正在运行的动画。这样，用这个方法不仅可以改善动画的性能，还可以节省电能，后者对于移动设备尤为重要，电池寿命通常是移动设备的一个重要因素。

但requestAnimationFrame()方法的实现会有一个问题：因为不同的浏览器使用的这个函数名称稍有不同。一个实际的解决方法是使用JavaScript实用工具库webgl-utils.js中的一个小程序，这个实用工具库最早由Google公司的开发人员编写的。

在webgl-utils.js中，requestAnimationFrame()的支持跨平台的小程序采用如下格式：

/\*\*

 \* Provides requestAnimationFrame in a cross browser way.

 \*/

window.requestAnimFrame = (function() {

 return window.requestAnimationFrame ||

 window.webkitRequestAnimationFrame ||

 window.mozRequestAnimationFrame ||

 window.oRequestAnimationFrame ||

 window.msRequestAnimationFrame ||

 function(/\* function FrameRequestCallback \*/ callback, /\* DOMElement Element \*/ element) {

 window.setTimeout(callback, 1000/60);

 };

})();

可以看出，如果没有找到其他方法，则使用setTimeout()回退函数。这个实用工具库命名该方法为requestAnimFrame()而非requestAnimationFrame()，其原因是，此标准只是W3C的工作草案，还需要进一步修订。

## 具体代码的实现

这节课的代码以第4课为基础，不同的地方在于draw()方法中需要添加调用requestAnimFrame()函数的代码：

function draw() {

 requestAnimFrame(draw);

 var currentTime = Date.now();

 elapsedTime = currentTime - previousTime;

 previousTime = currentTime;

totalTime += elapsedTime / 1000;

// 以下代码省略

}

requestAnimFrame()方法很容易使用。只需要把绘制回调函数作为它的第一个参数传入，就可以调用这个回调函数。即：

requestAnimFrame(draw);

还设置了三个全局变量：

var previousTime; // 上一帧的开始时刻

var elapsedTime = 0; // 每帧流逝的时间（毫秒）

var totalTime = 0; // 动画开始起的总时间

这些时间变量在动画循环中是很有用的。比方说下draw()方法的下面的代码中就使用了totalTime作为矩阵变换的参数。代码如下：

 // 绘制第一个立方体，在原地绕Y轴旋转

 pushModelViewMatrix();

 mat4.rotateY(modelViewMatrix, -totalTime);

 uploadModelViewMatrixToShader();

 drawCube();

 popModelViewMatrix();

 // 绘制第二个立方体，缩小为原来大小的0.3倍，自转为绕Z轴旋转，

 // 公转为绕第一个立方体旋转

 pushModelViewMatrix();

 mat4.rotateY(modelViewMatrix, totalTime \* 2);

 mat4.translate(modelViewMatrix, [2.0, 0.0, 0.0]);

 mat4.rotateZ(modelViewMatrix, totalTime);

 mat4.scale(modelViewMatrix, [0.3, 0.3, 0.3]);

 uploadModelViewMatrixToShader();

 drawCube();

 popModelViewMatrix();

}

前面我已提到，这个示例是移植自[Direct3D 10教程5：3D变换](http://shiba.hpe.cn/jiaoyanzu/wuli/showArticle.aspx?articleId=946&classId=5)，如果你看一下那篇文章中的矩阵变换，会发现有所不同：

第一个立方体绕Y轴旋转的值，在DirectX程序中是正的，在WebGL程序中是负的，这是因为DirectX是左手坐标系，OpenGL是右手坐标系，正负相反才会得到同样的效果：绕Y轴逆时针旋转。

第二个立方体的矩阵变换较复杂，在DirectX中顺序是：mScale、mSpin、mTranslate、mOrbit，而在WebGL中恰相反：mOrbit、mTranslate、mSpin、mScale，这是因为OpenGL的矩阵默认是列主序，变换矩阵是左乘的，而DirectX的矩阵是行主序的，变换矩阵是右乘的。

## 创建FPS计数器

定义每秒绘制多少帧（Frames Per Second，FPS）可以衡量动画的性能或动画的平稳程度。较高的FPS意味着较好的性能，即一般所谓的动画比较平稳。

理论上，FPS值比较容易计算。找们只需要测量绘制每帧所需要的时间。例如，在一帧的开始时检查时间并保存下求。在下一帧中，回到代码中的同一个地方，再次检查时间。第二次时间与第一次时间之差就是帧时间，其实就是上面代码中的elapsedTime变量，而FPS值是帧时间的倒数。

每更新一帧就计算FPS值的缺点是它可能变化太频繁。通常是计数每秒内绘制的帧，当一秒时间过去后，则把显示的帧数作为FPS值并显示。然后重新开始计数帧，又经过一秒后，再显示这个计数值，以此类推。

具体代码在draw()中：

var previousSecondTimeStamp; // 上一秒的开始时刻

var FPS = 0; // 帧数

function draw() {

requestAnimFrame(draw);

 var currentTime = Date.now();

 // 经过1秒就重新更新一次FPS

 if (currentTime - previousSecondTimeStamp >= 1000) {

 fpsCounter.innerHTML = FPS;

 FPS = 0;

 previousSecondTimeStamp = currentTime;

}

// 以下代码省略

…

 // 更新FPS

 FPS++;

}

在1秒钟内，利用fpsCounter的innerHTML属性把FPS的值设置为在过去一秒钟时间内绘制的帧数。

你还需要在startup()函数中初始化计算FPS所需要的量个变量。另外，在HTML代码中添加一个<span>无素，它有一个唯一的id。通过它来显示FPS值，即利用documentgetElementByld()函数得到这个<span>的引用，然后利用innerHTML属性设置FPS值，如下所示。

<script>

…

 function startup() {

 …

 previousTime = previousSecondTimeStamp = Date.now();

 FPS = 0;

…

 }

</script>

</head>

<body onload="startup();">

 <canvas id="lesson5-canvas" width="640" height="480"></canvas>

 <div id="fps-counter">

 FPS: <span id="fps">--</span></div>

</body>