[**Flash与3D编程探秘（七）- 3D物体框架**](http://www.cnblogs.com/yangzhou1030/archive/2008/11/09/1330015.html)

日期：2008年11月

从这篇文章开始，我将开始介绍3D物体及其在空间中运动和交互。这里提到的物体是指单个的实体，比如银河系中的一颗恒星，那么空间就是银河系了。不 过，所有的一切都是相对的，当一个分子作为例子中的实体的时候，那么一个细胞也可以作为3D的空间来看待（一个细胞是由很多的分子组成），同理你可以知道 细胞相对于一个生物（空间）来说也是一个物体。有些说多了，不过我想让你明白，我们用程序模拟一只小狗，或者一个人作为一个整体，但是不可能完全真实的模 拟它。因为，人体由数不清的细胞组成，每一个细胞都是一个物体，做着自己的运动，除非使用计算机真实模拟着人体的每一个细胞以及它的运动，否则永远不可能 得到一个真实模拟的人。但是使用现代的计算机科技是不可能模拟组成人体的所有细胞，那就更不用说组成每个细胞的分子。

还是言归正传来看一个3D物体的例子，这也是第一个绘制一个3D物体的例子。这个程序里，创建一个正方体并且让它围绕着正方体的对角线交点自转，不过这个正方体还是由8个好朋友小P组成，每个顶点站一个，由它们来勾勒这个正方体的框架。

**一个小P组成的正方体，鼠标掠过开始旋转**

#### ****动画制作步骤****

1. 首先在Flash IDE里绘制一个物体小P。

2. 开始设置还是和以前一样，原点，摄像机，焦距等等，另外不要忘记创建一个旋转角度object，存放物体在x，y和z轴的旋转角度变量。

// constants  
var PI = 3.1415926535897932384626433832795;  
  
// origin is the center of the view point in 3d space  
// everything scale around this point  
// these lines of code will shift 3d space origin to the center  
var origin = new Object();  
origin.x = stage.stageWidth/2;  
origin.y = stage.stageHeight/2;  
origin.z = 0;  
  
// focal length of viewer's camera  
var focal\_length = 300;  
  
// now create a scene object to hold the spinning box  
var scene = new Sprite();  
scene.x = origin.x;  
scene.y = origin.y  
this.addChild(scene);  
  
var axis\_rotation = new Object();  
axis\_rotation.x = 0;  
axis\_rotation.y = 0;  
axis\_rotation.z = 0;  
  
var camera = new Object();  
camera.x = 0;  
camera.y = 0;  
camera.z = 0;

[复制代码](javascript:void(0);)

3. 写一个函数，用它来创建空间中的一个点，scale\_point代表这个点在投射到2D平面上后位置缩放的比率。

// this function construct a 3d vertex  
function vertex3d(x, y, z, scale = 1):Object  
{  
    var point3d = new Object();  
    point3d.x = x;  
    point3d.y = y;  
    point3d.z = z;  
    point3d.scale\_point = scale;  
    return point3d;  
}

[复制代码](javascript:void(0);)

4. 下面发挥一下你的空间想象力，使用第3步的函数创建正方体的8个顶点，并且把它们添加到一个数组里。

// we calculate all the vertex  
var len = 50;                    // half of the cube width  
// now create the vertexes for the cube  
var points = [  
                //        x        y        z  
                vertex3d(-len,    -len,     -len),            // rear upper left  
                vertex3d(len,    -len,     -len),            // rear upper right  
                vertex3d(len,    -len,     len),            // front upper right  
                vertex3d(-len,    -len,     len),            // front upper left  
                  
                vertex3d(-len,    len,     -len),            // rear lower left  
                vertex3d(len,    len,     -len),            // rear lower right  
                vertex3d(len,    len,     len),            // front lower right  
                vertex3d(-len,    len,     len),            // front lower left  
            ];

[复制代码](javascript:void(0);)

5. 初始化8个小P，并且把它们放在8个顶点（映射到xy轴上的点）所在的x和y位置。

// init balls and put them on the screen  
for (var i = 0; i < points.length; i++)  
{  
    var ball = new Sphere();  
    ball.x = points[i].x;  
    ball.y = points[i].y;  
    ball.z = 0;  
    scene.addChild(ball);  
}

[复制代码](javascript:void(0);)

6. 这个函数你在摄像机空间旋转一篇文章中应该见过，函数的功能是把3D空间的点，映射到2D平面xy上。函数执行的步骤是这样的：

    a) 提前计算出x，y和z旋转角度的正余弦值。  
    b) 使用for loop遍历物体所有的顶点。  
    c) 使用计算出的正余弦和三角函数对三个轴的旋转分别进行计算，得出旋转后顶点的x，y和z。  
    d) 然后计算出物体在2D平面上映射后的x和y值。  
    e) 并且把这些2D点添加到一个新的数组里。

    f) 最后返回这个数组。

function project\_pts(points)  
{  
    var projected = [];  
    // declare some variable for saving function call  
    var sin\_x = Math.sin(axis\_rotation.x);  
    var cos\_x = Math.cos(axis\_rotation.x);  
    var sin\_y = Math.sin(axis\_rotation.y);  
    var cos\_y = Math.cos(axis\_rotation.y);  
    var sin\_z = Math.sin(axis\_rotation.z);  
    var cos\_z = Math.cos(axis\_rotation.z);  
      
    var x, y, z,                // 3d x, y, z  
        xy, xz,            // rotate about x axis  
        yx, yz,            // rotate about y axis  
        zx, zy,            // rotate about z axis  
        scale;            // 2d scale transform  
      
    for (var i = 0; i < points.length; i++)  
    {  
        x = points[i].x;  
        y = points[i].y;  
        z = points[i].z;  
          
        // here is the theroy:  
        // suppose a is the current angle, based on given current\_x, current\_y on a plane  
        // (can be x, y plane, or y, z plane or z, x plane), rotate angle b  
        // then the new x would be radius\*cos(a+b) and y would be  radius\*sin(a+b)  
        // radius\*cos(a+b) = radius\*cos(a)\*cos(b) - radius\*sin(a)\*sin(b)  
        // radius\*sin(a+b) = radius\*sin(a)\*cos(b) + radius\*cos(a)\*sin(b)

        // rotate about x axis  
        xy = cos\_x\*y - sin\_x\*z;  
        xz = sin\_x\*y + cos\_x\*z;  
        // rotate about y axis  
        yz = cos\_y\*xz - sin\_y\*x;  
        yx = sin\_y\*xz + cos\_y\*x;  
        // rotate about z axis  
        zx = cos\_z\*yx - sin\_z\*xy;  
        zy = sin\_z\*yx + cos\_z\*xy;  
        // scale it  
        scale = focal\_length/(focal\_length+yz-camera.z);  
        x = zx\*scale - camera.x;                // get x position in the view of camera  
        y = zy\*scale - camera.y;                // get x position in the view of camera  
          
        projected[i] = vertex3d(x, y, yz, scale);  
    }  
    return projected;  
}

[复制代码](javascript:void(0);)

这样就得到一个数组，包含所有需要的2D数据。并不困难，你完全可以把这一段代码叫做这个程序的3D引擎，它负责了所有点的数据在空间里旋转计算和输出。

7. 下面是动画执行的循环函数。需要注意的一点，在以后的文章中我都将使用基于时间的运动。在这里你只要知道下面的公式就可以了：旋转角度=角速度X时间。使 用上面的公式，递增物体围绕y轴和x轴的旋转角度。然后使用第6步的函数计算所有的3D顶点旋转后的位置并且得到映射后的2D点。剩下你应该能想到，就是 把相应的小P定位到这些定点上，并且对小球进行缩放比率scale\_point，最后不要忘记对小P进行z排序。

function move(e:Event):void  
{  
    // well we use time based movement in this tutorial  
    var current\_time = new Date().getTime();                // sampe the current time  
    // increment the rotation around y axis  
    axis\_rotation.y += 0.0008\*(current\_time-start\_time);  
    // increment the rotation around x axis  
    axis\_rotation.x += 0.0006\*(current\_time-start\_time);  
    start\_time = current\_time;                                // reset the start time  
      
    var projected = project\_pts(points);        // 3d ponts to 2d transformation           
      
    // now we have all the data we need to position the balls  
    for (var i = 0; i < scene.numChildren; i++)                // loop throught the scene  
    {  
        // positioning the ball  
        scene.getChildAt(i).x = projected[i].x;  
        scene.getChildAt(i).y = projected[i].y;  
        scene.getChildAt(i).z = projected[i].z;  
        scene.getChildAt(i).scaleX = scene.getChildAt(i).scaleY = projected[i].scale\_point;  
    }  
      
    swap\_depth(scene);                // sort out the depth   
}  
  
// bubble sort algo  
function swap\_depth(container:Sprite)  
{  
    for (var i = 0; i < container.numChildren - 1; i++)  
    {  
        for (var j = container.numChildren - 1; j > 0; j--)  
        {  
            if (Object(container.getChildAt(j-1)).z < Object(container.getChildAt(j)).z)  
            {  
                container.swapChildren(container.getChildAt(j-1), container.getChildAt(j));  
            }  
        }  
    }  
}  
  
// now add the event listener and spin the box  
this.addEventListener(Event.ENTER\_FRAME, move);

[复制代码](javascript:void(0);)

#### ****注意****

例子中物体沿着x和y轴旋转，但是并没有添加z轴旋转，你可以自己添加上，看看有什么不同。

#### ****注意****

例子中我们使用了两个for loop，第一次遍历所有的顶点把3D点转化为2D点，第二个把相应的小P定位到这些2D点的位置。虽然这样看起来会降低执行速度，但是这样会使程序的流程一目了然。如果你已经非常熟练，你可以试着修改这两个函数提高执行速度。

#### 使用Flash绘制API

上面的例子看起来不错，不过只有顶点有小球，我们看起来还不满意，那么接下来做一个动态绘制的正方体。这个例子里，基本的框架并没有什么变化，正方体所有的边都是用Flash的moveTo()和lineTo()来绘制。那么我就把需要更改代码的地方解释一下。

**一个正方体的框架，鼠标掠过开始旋转**

#### ****制作步骤****

1. 基本上的代码和前面是一样的，同样需要设置场景，创建正方体的顶点，注意不要再在舞台上添加小P。

2. 当把3D点映射到2D平面上后，使用黑线把正方体相邻的两个点连接起来。非常容易理解，不过注意，有很多种办法连接这些点，你可以先连接正方体顶面的点，然后底面的点，最后连接顶面和底面，不要局限于这几种，试着去发现新的方法，找到合适你思维方式的一种。

function move(e:Event):void  
{  
    // well we use time based movement in this tutorial  
    var current\_time = new Date().getTime();                // sampe the current time  
    // increment the rotation around y axis  
    axis\_rotation.y += 0.0008\*(current\_time-start\_time);  
    // increment the rotation around x axis  
    axis\_rotation.x += 0.0006\*(current\_time-start\_time);  
    start\_time = current\_time;                            // reset the start time  
      
    var projected = project\_pts(points);        // 3d ponts to 2d transformation           
      
    // now we start drawing the cube  
    with (scene.graphics)  
    {  
        clear();  
        lineStyle(0.5, 0x0F6F9F, 1);  
        // top face  
        moveTo(projected[0].x, projected[0].y);  
        lineTo(projected[1].x, projected[1].y);  
        lineTo(projected[2].x, projected[2].y);  
        lineTo(projected[3].x, projected[3].y);  
        lineTo(projected[0].x, projected[0].y);  
        // bottom face  
        moveTo(projected[4].x, projected[4].y);  
        lineTo(projected[5].x, projected[5].y);  
        lineTo(projected[6].x, projected[6].y);  
        lineTo(projected[7].x, projected[7].y);  
        lineTo(projected[4].x, projected[4].y);  
        // vertical lines  
        moveTo(projected[0].x, projected[0].y);  
        lineTo(projected[4].x, projected[4].y);  
        moveTo(projected[1].x, projected[1].y);  
        lineTo(projected[5].x, projected[5].y);  
        moveTo(projected[2].x, projected[2].y);  
        lineTo(projected[6].x, projected[6].y);  
        moveTo(projected[3].x, projected[3].y);  
        lineTo(projected[7].x, projected[7].y);  
    }  
}

[复制代码](javascript:void(0);)

3. 还有一个地方需要改动，因为不再对顶点的物体进行缩放，所以就必须要传递scale\_point这个属性。

// this function construct a 3d vertex  
function vertex3d(x, y, z):Object  
{  
    var point3d = new Object();  
    point3d.x = x;  
    point3d.y = y;  
    point3d.z = z;  
    return point3d;  
}

[复制代码](javascript:void(0);)

#### ****建议****

试着把上面的两种框架构建方式结合在一起，制作一个旋转的物体被线连着，试一试一条螺旋体的模型，或者如果你想增加难度的话，你还可以做一个DNA链。

**DNA链**

那么到目前为止，你已经知道如何使用框架构建一个方体，不过现实中物体总是有纹理和填充色的。你也许会想，使用Flash的beginFill()函数就 可以给物体加上填充色了，这不是很简单。Hum，很接近不过如果要给物体上色的话，还有很多工作要做，后面的文章中将重点开始介绍着色筛选和相关内容。

#### ****关于Time Based和Frame Based运动****

文章第一个例子中的制作步骤里，提到关于基于时间的运动公式（只要知道了物体运动的速度，那么根据牛顿第一运动定律就可以得出物体在某个时间点的位移）：

位移 = 时间 X 速度

回想一下，前面的几篇文章里使用的都是基于祯的运动，然而基于祯的运动是不稳定的，它的公式是：

位移 = 执行次数 X 速度

基于祯的运动不管程序执行流逝了多少时间，只在function执行的时候给物体的x或者y加减一定的值。这种运动是不稳定的，所以我建议大家使用 基于时间的运动，下面的两个动画分别用两种运动模式做成，点击一下动画就会在function执行时执行大量的junk运算，这时你就会看到两种运动的差 异。而基于时间的运动中，当速度恒定时，物体会处在正确的位置；基于祯的运动，你就会看到物体运动慢下来很多， 并不能达到物体在某个时间点应该到达的位置。由于这个页面里还有另外3个使用大量CPU运算的动画，所以我把下面的动画移到另外一个页面了，[点击这里到另外一个文章里查看下面的两个动画](http://www.cnblogs.com/yangzhou1030/archive/2008/11/09/1330101.html)，如果感觉动画还是不够连贯的话，那么你可以[下载这两个动画到本机察看](http://tw1hjq.bay.livefilestore.com/y1p812-Shz8pOd4iSKmj16lXpL7_OIWf9visHSYAkUCuPJxWnssNvLpnTT9alNGe13wF8UarUY6pptzGinqBrbzEg/time_based_frame_based.rar?download)。

http://www.cnblogs.com/yangzhou1030/archive/2008/11/09/1330015.html