[**Flash与3D编程探秘（八）- 3D物体着色基础知识**](http://www.cnblogs.com/yangzhou1030/archive/2008/11/14/1333608.html)

日期：2008年11月

前面的文章讨论了如何使用线绘制物体的框架，可是往往模拟现实中的3D物体并不是只有框架。比如一本书或者是一块玻璃，它们都是具有填充的物体。虽 然在程序里能够（或者我应该说很是不实际）真正的给物体进行填充，但是可以通过给物体的表面着色这个方法，使物体看起来更加3D，而如何给物体表面着色将 是后面两篇文章讨论的重点。在这一篇文章中，我将介绍一些关于着色的基本知识，其中涉及到一些向量数学运算，如果你已经有这些数学背景的话，那么这些对你 来说非常容易。如果对你还是新课题的话，也不要放弃，只要你有一直读前面的文章，相信下面的内容对你来说应该不会困难。

先来看一个给物体着色的例子，运行动画你会看到一个透明的金字塔在舞台上旋转，虽然很简单，不过拿来热身非常合适。程序的框架和上一篇中的正方体的例子大致一样，所以我就只把需要改动代码的地方解释一下。

**透明的金字塔**

#### ****制作步骤****

1. 完全可以Copy前面例子的代码来使用，首先需要把前面例子正方体的点的定义删除，然后把构造这个金字塔的5个3D空间点添加到数组中。

// we calculate all the vertex  
var len = 50;                                                    // half of the bottom face width  
// now create the vertexes for the cube  
var points = [  
                //        x        y        z  
                vertex3d(0,    -len,     0),                  // top  
                  
                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);)

2. 写一个绘制函数，功能是把所有参数点连接起来，并且对连接后的区域着色。

// draw the face with args, the vertex of the facet  
function draw\_face(... args)  
{  
    with (scene.graphics)  
    {  
        lineStyle(.5, 0x7DBFC6, 1);  
        beginFill(0xB3DADD, .3);  
        moveTo(args[0].x, args[0].y);  
        for (var i = 1; i < args.length; i++)  
        {  
            lineTo(args[i].x, args[i].y);  
        }  
    }  
}

[复制代码](javascript:void(0);)

3. 在更新物体的update函数中，把前面画正方体的代码去掉，然后添加如下代码。按照顺序画出底面，正面，反面和两个侧面。

scene.graphics.clear();  
// now we start drawing the cube  
// bottom face  
draw\_face(pro[1], pro[2], pro[3], pro[4]);  
// front face  
draw\_face(pro[0], pro[1], pro[2]);  
// back face  
draw\_face(pro[0], pro[2], pro[3]);  
// left face  
draw\_face(pro[0], pro[3], pro[4]);  
// right face  
draw\_face(pro[0], pro[4], pro[1]);

[复制代码](javascript:void(0);)

#### ****建议****

你可以尝试制作一些更加复杂的模型来训练你的空间感，比如制作一个旋转的三角房子，或者是一颗钻石。

**一颗简单的钻石，点击选择是否填充表面**

#### ****注意****

通过上面的例子，你一定会注意到，设置场景的代码，project\_pts等函数这些代码是一成不变的，完全可以把它们写成类，使用的时候直接调用即可。

#### ****注意****

由于物体是透明的，所以有时候你会产生错觉，物体变形了，其实是人的大脑把表面看错位置了。

#### ****不透明物体****

Looks pretty cool！不过不知道你有没有发现，上面的例子中物体的表面都是透明的，因此不管物体的背面背对摄像机或者面对摄像机，程序都给它着色，降低了我们的工作 量，但是却增加了CPU的负荷。那么如果物体不是透明的怎么使用Flash绘制呢？先来分析一下一个不透明物体是如何出现在摄像机的镜头中的：当物体一个 表面背对着摄像机的时候，这个表面是不可见的，当这个面面对摄像机的时候，它是可见的。虽然这些道理我们都知道，但是程序并不知道哪个可见哪个不可见，也 不知道何时给表面着色。怎样让程序判断这个表面是不是可见呢？

#### ****背面筛选****

判断一个物体的表面是否可见叫做背面筛选（Backface Culling），这不是一个新的课题，有很多的办法实现它。一种方法是利用空间中三个点的位置关系来判断，虽然我不提倡，不过这种方法相对来说好理解， 适用于小规模程序。第二种方法是利用物体表面的法线与摄像机的视线夹角来判断，这种方法比较正统，也是后面主要讨论的。在这篇文章里，我主要讲述前两种， 以便于对比找出一种适合你的方法。

#### ****注意****

当然还有其他的一些方法和技巧，比如说把物体的每个表面都放在不同的层，每一次刷新画面的时候都对所有层进行排序，以达到目的，这种方法的好处在于你能够控制每一个表面，以便于做出鼠标或者键盘事件响应。

#### ****利用空间三个点的位置关系筛选****

第一点要清楚的是，空间的三个不共线的点确定一个面，这也是为什么使用三个点而不是四个的原因。下面的动画演示的是一个表面的三个点，你可以尝试拖动它们看看什么情况下这个面不可见。

**利用三个点的关系判断表面是否可见**

这种算法核心就是比较两个边的斜率，例如对于B点来说，（拖动）测试它沿BA和BC发射的两条直线是否重合，当它们重合时（BC斜率超过BA斜率时），变化表面BCA的可见性。

(b.y-a.y)/(b.x-a.x) < (c.y-a.y)/(c.x-a.x)

不过有一个问题，当三角形BCA旋转时，也会造成两条直线的斜率大小变化，于是再加上下面的判断：

a.x <= b.x == a.x > c.x

把上面两个结合起来便得到一个完整判断，你完全可以依赖这种方法计算背面筛选，虽然看起来很简单，但是我做很多测试，并没有发现问题。不过要注意在判断时 使用的ABC点的顺序，如果顺时针不对的话，那么使用逆时针CBA 顺序通常会解决问题。请注意，由于这篇文章篇幅过长，如何使用这个算法执行背面筛选将在下一篇中介绍。

if (Number((b.y-a.y)/(b.x-a.x) < (c.y-a.y)/(c.x-a.x)) ^ Number(a.x <= b.x == a.x > c.x))  
{  
     return true;  
}  
return false;

[复制代码](javascript:void(0);)

#### ****Bitwise XOR ^****

下面是一个例子：

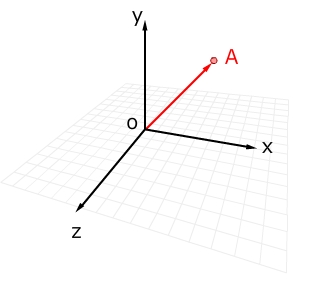
1001 == 1111 ^ 0110

我的解释是，当二进制运算两个位相同时，产生1，否则产生0，需要注意的是，在程序里适当的使用XOR等二进制运算时，会提高你的程序的运行速度。

上面的方法虽然很不错，不过精益求精，我还是要给大家介绍一个比较正统的背面筛选的方法，利用表面法线与视线夹角判断是表面否可见。但是由于这个课题有一些数学要求，因此我将放在下篇文章介绍。

#### ****向量****

当你步入3D图形编程时，会与向量经常打交道。因此在介绍表面法线与视线夹角判断背面筛选之前，我想给大家快速介绍一些关于向量计算的数学知识。我想毕竟 很多读者还不了解这些数学知识（笔者的确很笨的说，经常会想象大家也一样，真是非常抱歉），如果你对向量运算，矩阵运算非常熟悉的话，那么这后面的内容你 可以略过。（请注意这里公式都是程序书写方式，比如乘号是“\*“，除号是“/“）



**向量OA**

1. 首先你要清楚什么是向量（矢量，vector），空间中的两个点A和B，那么A->B就是一个向量，可以读成从A到B，它既有大小又有方向。同理假 设原点是O，给定空间中任意一点C，OC是从O到C的向量，我们把这个向量记为V。设点V的坐标为[a, b, c]（在文章中我将一直使用横板向量书写方式），那么使用下面的公式来表示从原点O到C的向量：

V = a\*i + b\*j + c\*k

其中i，j和k表示向量V在x，y和z轴的单位向量（unit vector）。

2. 在解释单位向量（unit vector）之前，你需要知道如何计算向量的大小，设向量V = [a, b, c]，那么向量的大小（magnitude）我们记作|V|，计算公式是：

|V| = mag = sqrt(a\*a + b\*b + c\*c)

要注意的是，mag是一个标量（scalar），它只有大小，没有方向。举一个例子，求向量V = [3, 4, 0]的大小：

|V| = sqrt(3\*3 + 4\*4 + 0\*0) = 5

3. 一个向量可以和一个标量相乘，产生一个新的向量，并且它们乘法遵循交换规则。设V = [a, b, c]，i是一个标量，那么它们相乘的公式是：

i V = [i\*a, i\*b, i\*c]

同理，一个向量除以一个标量，产生一个新的向量：

V / i = [a/i, b/i, c/i]

4. 知道了向量的大小和上面的除法公式后，V的单位向量就容易解决了：

Vu = V/mag = [a/mag, b/mag, c/mag]

我的解释是，把向量V在x，y和z轴上的分量分别除以mag，就得到一个新的向量Vu，这个向量就是V的单位向量，这个过程叫做normalize。（你可以反向思维，把单位向量Vu乘以mag，就得到向量V）  
  
刚才求向量大小时，点的参照是基于原点的，如果向量V是由点A = [a, b, c]到B = [x, y, z]时，向量V的大小也就是AB之间的距离，我们可以使用下面的公式求出它们的距离：

D = sqrt((a-x)\*(a-x) + (b-y)\*(b-y) + (c-z)\*(c-z))

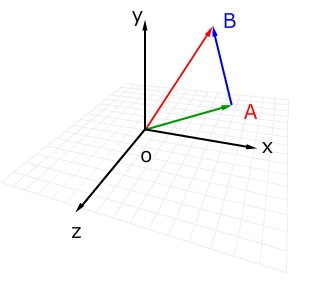
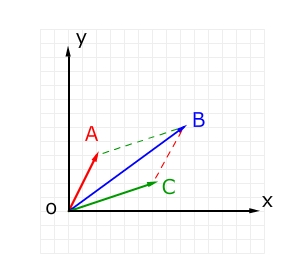
5. 两个向量可以进行加减运算（addition和substraction），设V = [a, b, c]，U = [x, y, z]，那么它的和与差分别是：

U + V = [a+x, b+y, c+z]  
U - V = [a-x, b-y, c-z]

[复制代码](javascript:void(0);)

如下图所示（左图），设从O到A的向量为U，从A到B的向量为V，那么向量U+V就是OB。再来看一个2D的图示（右图），设向量OA = [6, 2, 0]，向量OC = [2, 5.2, 0]，那么：

OB = OA + OC = [8, 7.2, 0]

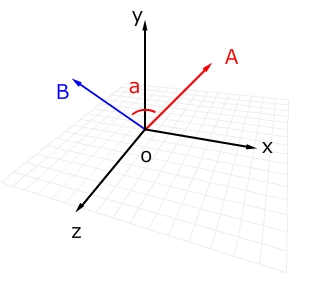
              

**向量相加**

能够看出，在OA与OC所确定的平面，以OA与OC为两个相邻边做一个平行四边形，对角线与OB重合。

6. 向量U和V的数量积（dot product，也称为标量积、点积、点乘或内积）数量积产生一个标量，运算公式如下：

U ・ V = |U| |V| cos(a)



**数量积**

其中a是U和V在3D空间中的夹角。如果已知两个向量，使用数量积我们就可以通过计算求得两个向量的夹角。如果，两个向量都是单位向量的话，它们的数量积就是它们夹角的余弦值：

Uu ・ Vu = cos(a)

举个例子，设U = [30, 40, 0]，V = [3, 4, 5]，那么U和V的数量积是：

U ・ V = 30\*3 + 40\*4 + 0\*5 = 250

U和V的大小分别是：

|U| = 50  
|V| = 7

[复制代码](javascript:void(0);)

那么得到：

cos (a) = 0.7  
a = 46

[复制代码](javascript:void(0);)

数量积满足以下的代数性质：  
交换率：

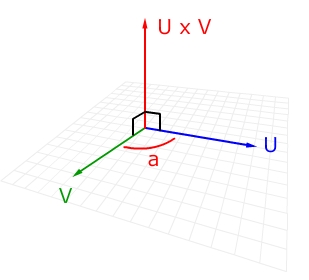
U ・ V = V ・ U

加法的分配率：

U ・ (V + P) = U ・ V + U ・ P

7. U和V的向量积（cross product，也称矢量积、叉积或者外积）产生一个向量，这个向量垂直于U和V，它的计算公式是：

U × V = n |U| |V| sin(a)



**向量积**

其中n为垂直于U和V的单位向量，a是U和V的夹角。向量积计算满足以下代数性质：

反交换率：

U × V = - (V × U)

加法的分配率：

U × (V + P) = U × V + U × P

与标量乘法相容：

s (U × V) = sU × V = U × sV

给定空间两个向量U = [a, b, c]，V = [x, y, z]，那么U和V的向量积是：

U × V = [b\*z - z\*b, c\*x - x\*c, a\*y - y\*a]

OK，这些就是关于向量的基本知识，是不是有点太多了，没关系，你完全不必记住所有的公式，只要你在使用时知道为什么使用这个公式就可以了。关于向量和矩阵的运算这里就不再列举，如果在后面文章用到的话，我会详细介绍。

#### ****补充一些名词解释****

法线（normal）的定义，三维平面上的法线是一条垂直于该平面的一个三维向量，法线可以通过平面上两条不平性的向量的向量积求得。

摄像机的视线是从摄像机的镜头到该平面某个点的向量，这里你可以理解为一束向量。

#### ****在程序里使用向量****

下面试着使用前面所讲述的内容，把向量这个概念加入到程序里，为下面的学习做好基础。下面这个例子，来制作一个3D的平面，并且让它旋转。在这个程序并没 有用到向量的运算，只不过我想让你把程序里所有与向量有关的Object都转换成Vector，这样你就能快速的明白使用Vector得必要性。基本的框 架还是和前面的例子一样，完全可以Copy本文中第一个例子的源代码，我只把需要更改的地方解释一下。

**引入向量概念**

#### ****制作步骤（部分）****

1. 首先要写一个向量类，把它命名为Vector.as，并且把它添加到工程里。我写了一个向量类，可以在附件中下载。你完全可以Copy使用，但是还是希望 你能够明白每一个函数表达的意思。注意copy函数是一个复制函数，剩下的函数就麻烦你和上面的向量运算数学知识一一对照了。

/\*   
 \* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR  
 \* IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,  
 \* FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE  
 \* AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER  
 \* LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,  
 \* OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE  
 \* SOFTWARE.  
 \*  
 \* This class can be used only if you keep this claim intact  
 \* Zhou Yang 2008.11 yangzhou1030@gmail.com  
 \*/  
package  
{  
// vector class  
public dynamic final class Vector  
{  
    public var x:Number;  
    public var y:Number;  
    public var z:Number;  
    // constructor  
    function Vector(x\_3d = 0.0, y\_3d = 0.0, z\_3d = 0.0)  
    {  
        x = x\_3d;  
        y = y\_3d;  
        z = z\_3d;  
    }  
    // copy the x y z value from another vector  
    // return false if parameter is not type of vector  
    public function copy(vector)                // return boolean  
    {  
        if (vector is Vector)  
        {  
            x = vector.x;  
            y = vector.y;  
            z = vector.z;  
            return 1;  
        }  
        return 0;  
    }  
    // vector addition  
    public function add(a)                        // return vector  
    {  
        var v = new Vector();  
        v.x = x + a.x;  
        v.y = y + a.y;  
        v.z = z + a.z;  
        return v;  
    }  
    // vector substraction  
    public function substract(a)                // return vector  
    {  
        var v = new Vector();  
        v.x = x - a.x;  
        v.y = y - a.y;  
        v.z = z - a.z;  
        return v;  
    }  
    // multiply vector by a scalar  
    public function multiply(scalar)            // return vector  
    {  
        var v = new Vector();  
        v.x = x\*scalar;  
        v.y = y\*scalar;  
        v.z = z\*scalar;  
        return v;  
    }  
    // devide vector by a scalar  
    public function devide(scalar)                // return vector  
    {  
        var v = new Vector();  
        v.x = x/scalar;  
        v.y = y/scalar;  
        v.z = z/scalar;  
        return v;  
    }  
    // vector dot product of v  
    public function dot(v)                        // return scalar  
    {  
        return x\*v.x + y\*v.y + z\*v.z;  
    }  
    // cross product  
    public function cross(a)                    // return vector  
    {  
        var v = new Vector();  
        v.x = y\*a.z - z\*a.y;  
        v.y = z\*a.x - x\*a.z;  
        v.z = x\*a.y - y\*a.x;  
        return v;  
    }  
    // distance from this to vector a  
    public function distance(a)                    // return scalar  
    {  
        var dx = x - a.x;  
        var dy = y - a.y;  
        var dz = z - a.z;  
        return Math.sqrt(dx\*dx + dy\*dy + dz\*dz);  
    }  
    // vector magnitude  
    public function mag()                        // return scalar  
    {  
        return Math.sqrt(x\*x+y\*y+z\*z);  
    }  
    // return normal vector  
    public function normal()                    // return vector  
    {  
        var v = new Vector();          
        var mag = this.mag();  
          
        if (mag == 0)  
            return 0;  
  
        v.x = x/mag;  
        v.y = y/mag;  
        v.z = z/mag;  
          
        return v;  
    }  
    // normalize vector this  
    // return false if magnitude is 0  
    public function normalize()  
    {  
        var mag = this.mag();  
        if (mag == 0)  
            return 0;  
  
        x /= mag;  
        y /= mag;  
        z /= mag;  
          
        return 1;  
    }  
}  
}

[复制代码](javascript:void(0);)

2. 把场景的原点定义为一个向量：

var origin = new Vector(stage.stageWidth/2, stage.stageHeight/2-30, 0);  
// create a scene to hold the polygon  
var scene = new Sprite();  
scene.x = origin.x;  
scene.y = origin.y;  
this.addChild(scene);

[复制代码](javascript:void(0);)

3. 把摄像机所在的点和平面的旋转角度各定义为一个向量：

var camera = new Vector(0, 0, 60);  
// this is the rotation of the polygon in 3d space  
var axis\_rotation = new Vector();        // default constructor init x y z to 0

[复制代码](javascript:void(0);)

4. 然后删除前面金字塔的顶点的定义，添加如下的代码，主要目的是定义空间中的四个点，当然这四个点在一个平面上，因为它们的z值都是0。

var vertexes = [  
               new Vector(-60, -60, 0),  
               new Vector(60, -60, 0),  
               new Vector(60, 60, 0),  
               new Vector(-60, 60, 0)  
               ];

[复制代码](javascript:void(0);)

5. 修改刷新画面的函数update，使用project函数把四个点映射到2D平面上，然后绘制两个组成四边形的三角形（绘制两个三角形是想让你明白，所有的物体表面都可以用三角形来绘制）。

with (polygon.graphics)  
{  
    clear();  
    lineStyle(.5, 0x000000, 0);                        // clear out what was previously drawn  
    beginFill(0x6A83A6, 1);                             // draw red triangle  
    draw\_face(pro[0], pro[1], pro[2]);             // notice the drawing order me used  
    endFill();  
    beginFill(0xBD5E53, 1);  
    draw\_face(pro[3], pro[2], pro[0]);  
    endFill();  
}

[复制代码](javascript:void(0);)

感谢你能够读到这里，写了这些，肯定会有疏忽和遗漏的地方，如果你有什么不明白的话，可以和我联系。另外，我相信基本的向量运算应该已经难不住你了，剩下的就是如何在程序里运用这些数学知识对背面筛选，将在下一篇文章中介绍。So keep it up!

http://www.cnblogs.com/yangzhou1030/archive/2008/11/14/1333608.html