[转载]从零开始学习OpenGL ES之七 - 变换和矩阵

2011-9-15 00:01 | 发布者: Iven | 查看: 15881 | 评论: 13

摘要: 图形图像, 编程,编程, OpenGL ES, 教程,OpenGL ES 3D

自定义旋转

旋转稍微有点难度。我们可以创建绕各轴旋转的矩阵。我们已经知道绕Z轴旋转的矩阵像这样：

$$\begin{bmatrix} \cos(n) & \sin(n) & 0 \\ -\sin(n) & \cos(n) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

绕X轴像这样：

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(n) & -\sin(n) \\ 0 & \sin(n) & \cos(n) \end{bmatrix}$$

绕 Y轴旋转：

$$\begin{bmatrix} \cos(n) & 0 & \sin(n) \\ 0 & 1 & -0 \\ -\sin(n) & 0 & \cos(n) \end{bmatrix}$$

这三种旋转写成函数：

```
static inline void Matrix3DSetXRotationUsingRadians(Matrix3D matrix, GLfloat degree
s)
{
    matrix[0] = matrix[15] = 1.0;
    matrix[1] = matrix[2] = matrix[3] = matrix[4] = 0.0;
    matrix[7] = matrix[8] = 0.0;
    matrix[11] = matrix[12] = matrix[13] = matrix[14] = 0.0;

    matrix[5] = cosf(degrees);
    matrix[6] = -fastSinf(degrees);
    matrix[9] = -matrix[6];
    matrix[10] = matrix[5];
}
static inline void Matrix3DSetXRotationUsingDegrees(Matrix3D matrix, GLfloat degree
s)
{
    Matrix3DSetXRotationUsingRadians(matrix, degrees * M_PI / 180.0);
}
static inline void Matrix3DSetYRotationUsingRadians(Matrix3D matrix, GLfloat degree
```

```
s)
{
    matrix[0] = cosf(degrees);
    matrix[2] = fastSinf(degrees);
    matrix[8] = -matrix[2];
    matrix[10] = matrix[0];
    matrix[1] = matrix[3] = matrix[4] = matrix[6] = matrix[7] = 0.0;
    matrix[9] = matrix[11] = matrix[13] = matrix[12] = matrix[14] = 0.0;
    matrix[5] = matrix[15] = 1.0;
}
static inline void Matrix3DSetYRotationUsingDegrees(Matrix3D matrix, GLfloat degree
s)
{
    Matrix3DSetYRotationUsingRadians(matrix, degrees * M_PI / 180.0);
}
static inline void Matrix3DSetZRotationUsingRadians(Matrix3D matrix, GLfloat degree
s)
{
    matrix[0] = cosf(degrees);
    matrix[1] = fastSinf(degrees);
    matrix[4] = -matrix[1];
    matrix[5] = matrix[0];
    matrix[2] = matrix[3] = matrix[6] = matrix[7] = matrix[8] = 0.0;
    matrix[9] = matrix[11] = matrix[12] = matrix[13] = matrix[14] = 0.0;
    matrix[10] = matrix[15] = 1.0;
}
static inline void Matrix3DSetZRotationUsingDegrees(Matrix3D matrix, GLfloat degree
s)
{
    Matrix3DSetZRotationUsingRadians(matrix, degrees * M_PI / 180.0);
}
```

对于各轴的旋转有两种方法，一种是使用弧度，另一种是使用角度。这三个矩阵被称为Eular angle 。有一个问题是我们必须按顺序对多个轴进行旋转，当我们对这三个角度进行旋转时，我们可能会遇到一种被称作gimbal lock（框架自锁）的现像，它导致其中一个轴的旋转被锁定。为防止这种现象的发生，我们必须创建一个可以处理多轴旋转的矩阵。除了可以消除自锁的问题，它还能在物体需要绕多轴旋转时节省处理器的开销。

说实在的，我并不打算假装理解了此方法后面的数学机制。我读过一篇有关此机制的博士论文（四元数的矩阵表示），但我无法完全理解所有的数学原理，所以我们就基于信任此多旋转矩阵可以正常工作（它确实如此）。此矩阵假定了一个指定的角度N以及一个由3个浮点数值表示的向量。此向量的各元素将被N乘，从而导致绕对应轴的旋转：

$$\begin{bmatrix} X^2(1-\cos(n))+\cos(n) & XY(1-\cos(n))-Z\sin(n) & XZ(1-\cos(n))+Y\sin(n) \\ YX(1-\cos(n))+Z\sin(N) & Y^2(1-\cos(n))+\cos(n) & YZ(1-\cos(n))-X\sin(n) \\ XZ(1-\cos(n))+Y\sin(N) & YZ(1-\cos(n))+X\sin(N) & Z^2(1-\cos(n))+\cos(n) \end{bmatrix}$$

此矩阵要求向量是以单元向量（即法线）方式被传递的，所以我们在为矩阵赋值前必须保证这点。表示为OpenGL矩阵，像这样：

```
static inline void Matrix3DSetRotationByRadians(Matrix3D matrix, GLfloat angle,
    GLfloat x, GLfloat y, GLfloat z)
{
    GLfloat mag = sqrtf((x*x) + (y*y) + (z*z));
    if (mag == 0.0)
    {
        x = 1.0;
        y = 0.0;
        z = 0.0;
```

```
    }
    else if (mag != 1.0)
    {
        x /= mag;
        y /= mag;
        z /= mag;
    }

    GLfloat c = cosf(angle);
    GLfloat s = fastSinf(angle);
    matrix[3] = matrix[7] = matrix[11] = matrix[12] = matrix[13] = matrix[14] = 0.0
;
    matrix[15] = 1.0;

    matrix[0] = (x*x)*(1-c) + c;
    matrix[1] = (y*x)*(1-c) + (z*s);
    matrix[2] = (x*z)*(1-c) - (y*s);
    matrix[4] = (x*y)*(1-c)-(z*s);
    matrix[5] = (y*y)*(1-c)+c;
    matrix[6] = (y*z)*(1-c)+(x*s);
    matrix[8] = (x*z)*(1-c)+(y*s);
    matrix[9] = (y*z)*(1-c)-(x*s);
    matrix[10] = (z*z)*(1-c)+c;

}
static inline void Matrix3DSetRotationByDegrees(Matrix3D matrix, GLfloat angle,
    GLfloat x, GLfloat y, GLfloat z)
{
    Matrix3DSetRotationByRadians(matrix, angle * M_PI / 180.0, x, y, z);
}
```

此多轴旋转的版本以glRotatef()一样地工作。

现在我们替换了3个内嵌函数，下面是使用自定义矩阵和变换函数的 drawView: 方法。新矩阵代码以**粗体**表示：

```
- (void)drawView:(GLView*)view;
{

    static GLfloat  rot = 0.0;
    static GLfloat  scale = 1.0;
    static GLfloat  yPos = 0.0;
    static BOOL     scaleIncreasing = YES;

    // This is the same result as using Vertex3D, just faster to type and
    // can be made const this way
    static const Vertex3D vertices[]= {
        {0, -0.525731, 0.850651},             // vertices[0]
        {0.850651, 0, 0.525731},              // vertices[1]
        {0.850651, 0, -0.525731},             // vertices[2]
        {-0.850651, 0, -0.525731},            // vertices[3]
        {-0.850651, 0, 0.525731},             // vertices[4]
        {-0.525731, 0.850651, 0},             // vertices[5]
        {0.525731, 0.850651, 0},              // vertices[6]
        {0.525731, -0.850651, 0},             // vertices[7]
        {-0.525731, -0.850651, 0},            // vertices[8]
```

```
        {0, -0.525731, -0.850651},            // vertices[9]
        {0, 0.525731, -0.850651},             // vertices[10]
        {0, 0.525731, 0.850651}               // vertices[11]
    };

    static const Color3D colors[] = {
        {1.0, 0.0, 0.0, 1.0},
        {1.0, 0.5, 0.0, 1.0},
        {1.0, 1.0, 0.0, 1.0},
        {0.5, 1.0, 0.0, 1.0},
        {0.0, 1.0, 0.0, 1.0},
        {0.0, 1.0, 0.5, 1.0},
        {0.0, 1.0, 1.0, 1.0},
        {0.0, 0.5, 1.0, 1.0},
        {0.0, 0.0, 1.0, 1.0},
        {0.5, 0.0, 1.0, 1.0},
        {1.0, 0.0, 1.0, 1.0},
        {1.0, 0.0, 0.5, 1.0}
    };

    static const GLubyte icosahedronFaces[] = {
        1, 2, 6,
        1, 7, 2,
        3, 4, 5,
        4, 3, 8,
        6, 5, 11,
        5, 6, 10,
        9, 10, 2,
        10, 9, 3,
        7, 8, 9,
        8, 7, 0,
        11, 0, 1,
        0, 11, 4,
        6, 2, 10,
        1, 6, 11,
        3, 5, 10,
        5, 4, 11,
        2, 7, 9,
        7, 1, 0,
        3, 9, 8,
        4, 8, 0,
    };

    static const Vector3D normals[] = {
        {0.000000, -0.417775, 0.675974},
        {0.675973, 0.000000, 0.417775},
        {0.675973, -0.000000, -0.417775},
        {-0.675973, 0.000000, -0.417775},
        {-0.675973, -0.000000, 0.417775},
        {-0.417775, 0.675974, 0.000000},
        {0.417775, 0.675973, -0.000000},
        {0.417775, -0.675974, 0.000000},
        {-0.417775, -0.675974, 0.000000},
        {0.000000, -0.417775, -0.675973},
        {0.000000, 0.417775, -0.675974},
```

```
        {0.000000, 0.417775, 0.675973},
    };

    static Matrix3D    translateMatrix;
    Matrix3DSetTranslation(translateMatrix, 0.0, yPos, -3.0);
    static Matrix3D    scaleMatrix;
    Matrix3DSetScaling(scaleMatrix, scale, scale, scale);
    static Matrix3D     tempMatrix;
    Matrix3DMultiply(translateMatrix, scaleMatrix, tempMatrix);
    static Matrix3D    rotationMatrix;
    Matrix3DSetRotationByDegrees(rotationMatrix, rot, 1.0f, 1.0f, 1.0f);
    static Matrix3D    finalMatrix;
    Matrix3DMultiply(tempMatrix, rotationMatrix, finalMatrix);
    glLoadMatrixf(finalMatrix);

    glClearColor(0.0, 0.0, 0.05, 1.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glEnableClientState(GL_VERTEX_ARRAY);
    glEnableClientState(GL_COLOR_ARRAY);
    glEnable(GL_COLOR_MATERIAL);
    glEnableClientState(GL_NORMAL_ARRAY);
    glVertexPointer(3, GL_FLOAT, 0, vertices);
    glColorPointer(4, GL_FLOAT, 0, colors);
    glNormalPointer(GL_FLOAT, 0, normals);
    glDrawElements(GL_TRIANGLES, 60, GL_UNSIGNED_BYTE, icosahedronFaces);

    glDisableClientState(GL_VERTEX_ARRAY);
    glDisableClientState(GL_COLOR_ARRAY);
    glDisableClientState(GL_NORMAL_ARRAY);
    glDisable(GL_COLOR_MATERIAL);
    static NSTimeInterval lastDrawTime;
    if (lastDrawTime)
    {
        NSTimeInterval timeSinceLastDraw =
            [NSDate timeIntervalSinceReferenceDate] - lastDrawTime;
        rot+=50 * timeSinceLastDraw;

        if (scaleIncreasing)
        {
            scale += timeSinceLastDraw;
            yPos += timeSinceLastDraw;
            if (scale > 2.0)
                scaleIncreasing = NO;
        }
        else
        {
            scale -= timeSinceLastDraw;
            yPos -= timeSinceLastDraw;
            if (scale < 1.0)
                scaleIncreasing = YES;

        }
    }
    lastDrawTime = [NSDate timeIntervalSinceReferenceDate];
}
```

诡异但很奇妙的自定义矩阵

到目前为止，我仅仅展示了怎样重建OpenGL存在的功能，我们所作的只是基于矩阵相乘是硬件加速的这一原理对性能进行了一番小小的提升，但在99%的情况下，还不足以让我们如此大费周章。

但是，手工处理矩阵变换还有其他好处。例如，你可以创建OpenGL ES本身不支持的变换。比如，你可以进行剪切变换。 剪切本质上就是将物体沿两轴扭转。如果你对一个塔的轴进行扭转，那么它就会成为比萨斜塔。下面是剪切矩阵：

$$\begin{bmatrix} 1 & yShear & 0 \\ xShear & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

下面是代码：

```
static inline void Matrix3DSetShear(Matrix3D matrix, GLfloat xShear, GLfloat yShear
)
{
    matrix[0] = matrix[5] =  matrix[10] = matrix[15] = 1.0;
    matrix[1] = matrix[2] = matrix[3] = 0.0;
    matrix[6] = matrix[7] = matrix[8] = matrix[9] = 0.0;
    matrix[11] = matrix[12] = matrix[13] = matrix[14] = 0.0;
    matrix[1] = xShear;
    matrix[4] = yShear;
}
```

如果我们应用剪切矩阵，我们将得到：

尝试一下用内嵌函数实现。你可以组合矩阵调用。例如，创建一个函数不需要任何矩阵乘法生成转移和尺寸变换。

结论

矩阵是一个很大而且经常让人误解的课题，许多人（包括我自己）都在殚精竭虑去理解它。希望本文能给你足够的帮助，而且还提供了一个矩阵相关的库，你可以用在自己的项目中。如果你希望下载自己尝试一下，请下载feePart6ProjectI。我定义了两个常量，一个用来打开/关闭自定义变换，另一个用来打开/关闭剪切变换。它们在GLViewController.h 中：

```
#define USE_CUSTOM_MATRICES 1
#define USE_SHEAR_TRANSFORM 1
```

1为打开，0为关闭。另外我还更新了 OpenGL ES Xcode TemEmpty%20OpenGL%20ES%20Application 项目使其

包括了新的矩阵函数，包括向量化矩阵乘法函数。如果你无法完全消化，你也无需担心。对于开发OpenGL ES程序的99%的人来说，你不需要完全理解透视空间，同质坐标或线性变换，只需大体上的了解即可。

感谢 Snappy Touch 的 Noel Llopis。

注脚

1. 实际上，当你调用glLoadIdentity()时，你加载了一个 4×4 的单元矩阵。
2. 如果你希望使用 struct 而不是 typedef 时，你必须记住要以引用方式传递参数。struct并不会像数组一样自动以引用方式被传递。
3. 使用Shark测试，drawView: 方法从.7% 的处理时间降低为 .1% 的处理时间。对于此方法而言，速度有本质的提高，但对程序总体而言，影响不大。

原文见：OpenGL ES From the Ground Up, Part 7: Transformations and Matrices

最新评论 发表评论

449088680 2014-8-28 15:07 引用

还是看英文原帖舒服点,英文能力强的同学,可以点击以下链接去看英文原帖,这个教程写得真心不错：
http://iphonedevelopment.blogspot.com/2009/05/opengl-es-from-ground-up-table-of.html

luozhonglan 2014-1-11 10:20 引用

这套教程的书名叫什么，是老外的博客么？

icyttea 2013-7-14 22:09 引用

很好，受教了

VITO 2013-1-23 18:41 引用

有个问题，为什么要自己定义一套scaling rotation之类的方法而不用API的函数呢？

ssff169 2013-1-1 23:30 引用

这个好像 "自定义转移"后面的没有讲完吧??

cxjwin 2012-8-14 15:39 引用

帖子讲的很透彻

myhapi 2012-2-22 09:04 引用

还不错，正是想要的。

halley 2012-2-19 14:26 引用