[转载]从零开始学习OpenGL ES之二 - 简单绘图概述

2011-9-14 21:52 | 发布者: Iven | 查看: 29168 | 评论: 28

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

还有许多理论知识需要讨论，但与其花许多时间在复杂的数学公式或难以理解的概念上，还不如让我们开始熟悉OpenGL ES的基本绘图功能。

请下载OpenGL Xcode项目模板。我们使用此模板而不是Apple提供的模板。你可以解压到下面目录来安装它：

```
/Developer/Platforms/iPhoneOS.platform/Developer/Library/Xcode/Project Templates/Application/
```

此模板用于全屏OpenGL程序，它具有一个OpenGL视图以及相应的视图控制器。 大部分时候你不需要动到此视图。此视图用于处理一些诸如缓存切换之类的事物，但在两处调用了其控制器类。

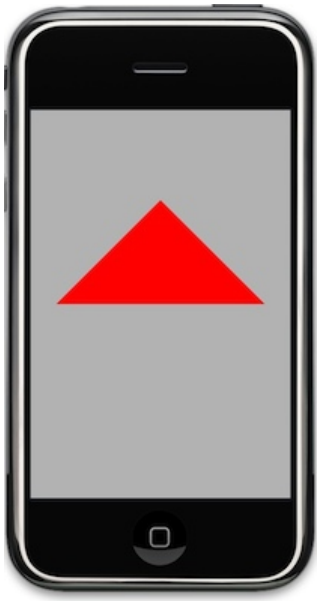
首先，当设定视图时，调用了一次控制器。调用视图控制器的 `setupView:` 方法使控制器有机会增加所需的设定工作。这里是你设定视口，添加光源以及进行其他项目相关设定的地方。现在我们将忽略此方法。此方法中已经有非常基本的设定以允许你进行简单地绘图。

控制器的 `drawView:` 方法根据常数kRenderingFrequency的值定期地被调用。kRenderingFrequency的初始值为15.0，表示 `drawView:` 方法每秒钟被调用15次。如果你希望改变渲染的频率，你可以在ConstantsAndMacros.h中找到此常数的定义。

首先加入下列代码到GLViewController.m的`drawView:`方法中：

```
- (void)drawView:(GLView*)view;
{
    Vertex3D    vertex1 = Vertex3DMake(0.0, 1.0, -3.0);
    Vertex3D    vertex2 = Vertex3DMake(1.0, 0.0, -3.0);
    Vertex3D    vertex3 = Vertex3DMake(-1.0, 0.0, -3.0);
    Triangle3D  triangle = Triangle3DMake(vertex1, vertex2, vertex3);

    glLoadIdentity();
    glClearColor(0.7, 0.7, 0.7, 1.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glEnableClientState(GL_VERTEX_ARRAY);
    glColor4f(1.0, 0.0, 0.0, 1.0);
    glVertexPointer(3, GL_FLOAT, 0, &triangle);
    glDrawArrays(GL_TRIANGLES, 0, 9);
    glDisableClientState(GL_VERTEX_ARRAY);
}
```

在讨论我们到底做了什么之前，先运行一下，你应该看到以下画面：



这是个简单的方法；如果你试过了，你可能已经知道我们到底做了什么，但这里我们还是一起过一遍。因为我们的任务是画三角形，所以需要三个顶点，因此我们创建三个上一篇文章中讨论过的Vertex3D对象：

```
Vertex3D    vertex1 = Vertex3DMake(0.0, 1.0, -3.0);
Vertex3D    vertex2 = Vertex3DMake(1.0, 0.0, -3.0);
Vertex3D    vertex3 = Vertex3DMake(-1.0, 0.0, -3.0);
```

你应该注意到了三个顶点的z值是一样的，其值(-3.0)是处于原点 “之后”的。因为我们还没有做任何改变，所以我们是站在原点上观察虚拟世界的，这是默认的起点位置。将三角形放置在z值为-3处，可以保证我们可以在屏幕上看到它。
随后，我们创建一个由这三个顶点构成的三角形。

```
Triangle3D  triangle = Triangle3DMake(vertex1, vertex2, vertex3);
```

这些代码很容易理解，对吗？但是，在幕后，电脑是将其视为一个包含9个 GLfloat 的数组。如下：

```
GLfloat  triangle[] = {0.0, 1.0, -3.0, 1.0, 0.0, -3.0, -1.0, 0.0, -3.0};
```

并不是完全相同 – 这里有一个很小但很重要的区别。在我们的示例中，我们传递给OpenGL的是 Triangle3D 对象的地址（即 &triangle ），但在第二个使用数组的示例中，由于C数组是指针，我们传递的是数组。现在不需要考虑太多，因为这将是最后一次我用这种方式（第二种方法）定义一个 Triangle3D 对象。等一下我将解释原因，但现在让我们先过一遍代码。下一步我们做的是加载单位矩阵。我将花至少一整篇文章讨论变换矩阵。我们暂且将其视为OpenGL的“复位开关”。它将清除虚拟世界中的一切旋转，移动或其他变化并将观察者置于原点。

```
glLoadIdentity();
```

之后，我们告诉OpenGL所有的绘制工作是在一个灰色背景上进行的。OpenGL通常需要用四个钳位值来定义颜色。上一篇文章中有提过，钳位浮点数是0.0 到 1.0之间的浮点数。我们通过定义红，绿，蓝以及alpha元素来定义颜色， alpha值定义了颜色之后物体的透视程度。现在暂时不用管它，将其设为1.0，代表完全不透明。

```
glClearColor(0.7, 0.7, 0.7, 1.0);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

在OpenGL中要定义白色，我们需要将四个元素全部设为1.0。要定义不透明的黑色，则定义红，绿，蓝为0.0，alpha为1.0。上例代码的第二行是通知OpenGL清除以前的一切图形并将其设为clear颜色。
你可能想知道 glClear()的两个参数是什么意思。简单地说，它们是存储与位域中的常量。OpenGL 保存了一系列 **缓存（buffers）**，即用于绘图各方面的内存块。将这两个值进行逻辑或是通知OpenGL清除两个不同的缓存 – **颜色缓存（color buffer）** 和 **深度缓存（depth buffer）**。 颜色缓存保存当前帧各像素的颜色。基本上就是你在屏幕上看到的。深度缓存（有时也称为“z-buffer”）保存每个潜在像素离观察者距离的信息。使用此信息可以确定

一个像素是否需要被绘制出来。这两个缓存是OpenGL中最常见的缓存。还有其他一些缓存，如模板缓存（stencil buffer）和累积缓存（accumulation buffer），但现在我们暂时不讨论这些。我们现在只需记住在绘制一帧之前，必须清除这两个缓存以保证不会和以前的内容混杂。

然后，我们要启动OpenGL的一项称为**vertex arrays**（顶点数组）的特性。此特性可能只需要`setupView:`方法中启动一次，但作为基本准则，我喜欢启动和禁止我使用的功能。你永远也不会知道是否另一段代码会做不同处理。如果你打开你需要的功能然后关闭它，产生问题的几率将大为减小。就本例来说，如果另一个类不使用顶点数组而使用顶点缓存的话，任何一段代码遗留了启动了的特性或没有显性启动其需要的特性，这一段或两段代码都会导致不可预知的结果。

```
    glEnableClientState(GL_VERTEX_ARRAY);
```

接下来我们设置了绘图时所需的颜色。此行代码将绘图颜色设为鲜艳的红色。

```
    glColor4f(1.0, 0.0, 0.0, 1.0);
```

现在，直到下次调用 `glColor4f()`前所有的图形都是以红色绘制。有一些例外的情况，例如绘制纹理形状时，但基本上，这样设定颜色可以使颜色保持。

由于我们使用顶点数组，我们必须通知OpenGL顶点的数组在什么地方。记住，顶点数组只是一个`GLfloat`的C数组，每三个值代表一个顶点。我们创建了`Triangle3D` 对象，但在内存中，它完全等同于9个连续的`GLfloat`，所以我们可以传递此三角形对象的地址。

```
    glVertexPointer(3, GL_FLOAT, 0, &triangle);
```

`glVertexPointer()` 的第一个参数指示了多少个`GLfloat`代表一个顶点。根据你是在进行二维或三维绘图，你可以传递2或者3。尽管我们的物体是存在于一个平面的，我们仍然将其绘制在三维虚拟世界中，因此每个顶点用三个数值表示，所以我们传递3给函数。然后，我们传递一个枚举值告诉OpenGL顶点是由`GLfloat`构成。

`OpenGL ES`允许你在当地数组中使用大部分的数据类型，但除`GL_FLOAT`外，其他都很少见。下一个参数… 现在不需要考虑下一个参数。那是以后讨论的主题。现在，它始终为0。在以后的文章中，我将讨论怎样使用此参数将同一对象以不同的数据类型混杂在一个数据结构中。

随后，我们通知OpenGL通过刚才提交的顶点数组来绘制三角形。

```
    glDrawArrays(GL_TRIANGLES, 0, 9);
```

你可能已经可以猜到，第一个枚举值是告诉OpenGL绘制什么。尽管OpenGL ES不支持绘制三角形之外的四边形或其他多边形，但它仍然支持一些其他绘图模式，如绘制点，线，线回路，三角形条和三角形扇。稍后我们将讨论这些绘图模式。

最后，我们要禁止先前启动了的特性以保证不会被其他地方的代码弄混。本例中没有其他的代码了，但通常你可以使用OpenGL绘制多个物体。

```
    glDisableClientState(GL_VERTEX_ARRAY);
```

好了，我们的代码可以工作了尽管它不是那么引人入胜而且不是十分高效，但它确确实实可以工作。每秒钟我们的代码被调用数次。不相信？加入下列黑体代码再次运行：

```
- (void)drawView:(GLView*)view;
{
    static      GLfloat rotation = 0.0;

    Vertex3D    vertex1 = Vertex3DMake(0.0, 1.0, -3.0);
    Vertex3D    vertex2 = Vertex3DMake(1.0, 0.0, -3.0);
    Vertex3D    vertex3 = Vertex3DMake(-1.0, 0.0, -3.0);
    Triangle3D  triangle = Triangle3DMake(vertex1, vertex2, vertex3);

    glLoadIdentity();
    glRotatef(rotation, 0.0, 0.0, 1.0);
```

```
    glClearColor(0.7, 0.7, 0.7, 1.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glEnableClientState(GL_VERTEX_ARRAY);
    glColor4f(1.0, 0.0, 0.0, 1.0);
    glVertexPointer(3, GL_FLOAT, 0, &triangle);
    glDrawArrays(GL_TRIANGLES, 0, 9);
    glDisableClientState(GL_VERTEX_ARRAY);

    rotation+= 0.5;
}
```

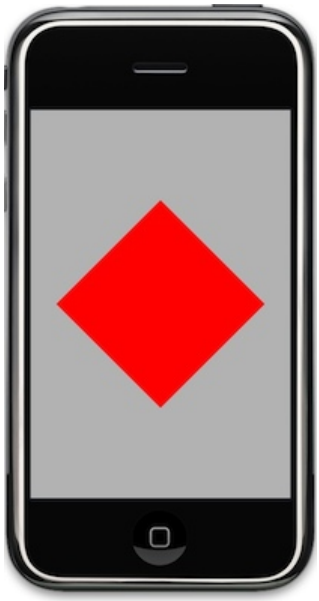
当你再次运行时，三角形将沿着原点缓缓转动。先不需要关注太多旋转的逻辑。我只是想告诉你我们的代码每秒钟被调用了多次。

如果你想画正方形怎么办？OpenGL ES并不支持正方形，所以我们只能通过三角形来定义正方形。这很简单 – 一个正方形可以通过两个三角形构成。我们要怎样调整上叙代码来绘制两个三角形？是不是可以创建两个Triangle3D？是的，你可以这样做，但那没有效率。我们最好将两个三角形置入同一个顶点数组中。我们可以通过定义一个包含两个Triangle3D对象的数组，或分配大小等于两个Triangle3D对象或18个GLfloat的内存.

这是一种方法:

```
- (void)drawView:(GLView*)view;
{
    Triangle3D  triangle[2];
    triangle[0].v1 = Vertex3DMake(0.0, 1.0, -3.0);
    triangle[0].v2 = Vertex3DMake(1.0, 0.0, -3.0);
    triangle[0].v3 = Vertex3DMake(-1.0, 0.0, -3.0);
    triangle[1].v1 = Vertex3DMake(-1.0, 0.0, -3.0);
    triangle[1].v2 = Vertex3DMake(1.0, 0.0, -3.0);
    triangle[1].v3 = Vertex3DMake(0.0, -1.0, -3.0);

    glLoadIdentity();
    glClearColor(0.7, 0.7, 0.7, 1.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glEnableClientState(GL_VERTEX_ARRAY);
    glColor4f(1.0, 0.0, 0.0, 1.0);
    glVertexPointer(3, GL_FLOAT, 0, &triangle);
    glDrawArrays(GL_TRIANGLES, 0, 18);
    glDisableClientState(GL_VERTEX_ARRAY);
}
```

运行，你将看到如下屏幕:



由于Vertex3DMake()方法在堆中创建新的Vertex3D然后复制其值到数组中而造成额外的内存被占用，因此上述代码并不理想。

对于这样简单的情况是没有什么问题的，但是在一个更为复杂的情况下，如果定义的3D物体很大时，那么你不会希望将其分配在堆中而且你不会希望对一个顶点不止一次地进行内存分配，所以最好是养成习惯通过我们的老朋友 malloc() （我更喜欢用 calloc()，因为它会将所有值设为0，比较易于查找错误）将顶点分配在栈中。 首先我们需要一个函数设定现存顶点的值而不是像Vertex3DMake()一样创建一个新对象。如下：

```
static inline void Vertex3DSet(Vertex3D *vertex, CGFloat inX, CGFloat inY, CGFloat inZ)

{
    vertex->x = inX;
    vertex->y = inY;
    vertex->z = inZ;
}
```

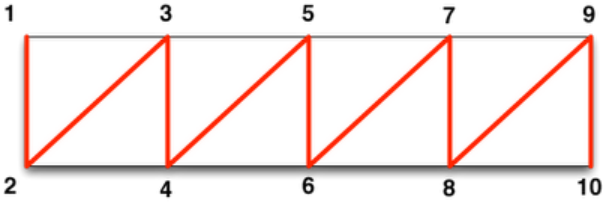
现在，我们使用新方法将两个三角形分配在栈中，重写代码：

```
- (void)drawView:(GLView*)view;
{
    Triangle3D  *triangles = malloc(sizeof(Triangle3D) * 2);

    Vertex3DSet(&triangles[0].v1, 0.0, 1.0, -3.0);
    Vertex3DSet(&triangles[0].v2, 1.0, 0.0, -3.0);
    Vertex3DSet(&triangles[0].v3, -1.0, 0.0, -3.0);
    Vertex3DSet(&triangles[1].v1, -1.0, 0.0, -3.0);
    Vertex3DSet(&triangles[1].v2, 1.0, 0.0, -3.0);
    Vertex3DSet(&triangles[1].v3, 0.0, -1.0, -3.0);

    glLoadIdentity();
    glClearColor(0.7, 0.7, 0.7, 1.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glEnableClientState(GL_VERTEX_ARRAY);
    glColor4f(1.0, 0.0, 0.0, 1.0);
    glVertexPointer(3, GL_FLOAT, 0, triangles);
    glDrawArrays(GL_TRIANGLES, 0, 18);
    glDisableClientState(GL_VERTEX_ARRAY);
```

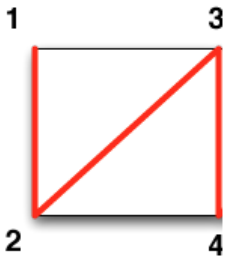
```
    if (triangles != NULL)
        free(triangles);
}
```

好了，我们已经讨论了许多基础知识，我们现在更深入一点。记住我说过OpenGL ES不止一种绘图方式吗？现在正方形需要6个顶点(18个 GLfloat)，实际上我们可以使用 **triangle strips** (GL_TRIANGLE_STRIP)方法通过四个顶点(12个 GLfloat)来绘制正方形。

这里是三角形条的基本概念：第一个三角形条是由前三个顶点构成(索引0, 1, 2)。第二个三角形条是由前一个三角形的两个顶点加上数组中的下一个顶点构成，继续直到整个数组结束。看下图更清楚 – 第一个三角形由顶点 1, 2, 3构成，下一个三角形由顶点 2, 3, 4构成，等等：



所以，我们的正方形是这样构成的：



代码如下：

```
- (void)drawView:(GLView*)view;
{
    Vertex3D  *vertices = malloc(sizeof(Vertex3D) * 4);

    Vertex3DSet(&vertices[0], 0.0, 1.0, -3.0);
    Vertex3DSet(&vertices[1], 1.0, 0.0, -3.0);
    Vertex3DSet(&vertices[2], -1.0, 0.0, -3.0);
    Vertex3DSet(&vertices[3], 0.0, -1.0, -3.0);

    glLoadIdentity();

    glClearColor(0.7, 0.7, 0.7, 1.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glEnableClientState(GL_VERTEX_ARRAY);
    glColor4f(1.0, 0.0, 0.0, 1.0);
    glVertexPointer(3, GL_FLOAT, 0, vertices);
    glDrawArrays(GL_TRIANGLE_STRIP, 0, 12);
    glDisableClientState(GL_VERTEX_ARRAY);

    if (vertices != NULL)
        free(vertices);
}
```

我们再返回到第一段代码看看。记住我们是怎样绘制第一个三角形吗？我们使用glColor4f()设置颜色并且说设置的颜色一直适用于随后的代码。那意味着定义于顶点数组中的物体必须用同一种颜色绘制。很有局限性，对吗？

并非如此。正如 OpenGL ES 允许你将所有顶点置于一个数组中，它还允许你将每个顶点使用的颜色置于一个颜

色数组（color array）中。如果你选择使用颜色数组，那么你需要为每个顶点设置颜色（四个GLfloat值）。

通过下面方法启动颜色数组：

```
glEnableClientState(GL_COLOR_ARRAY);
```

我们可以象顶点数组一样定义一个包含四个GLfloat成员的 Color3D结构。下面是怎样为原始三角形分配不同颜色的示例：

```
- (void)drawView:(GLView*)view;
{
    Vertex3D    vertex1 = Vertex3DMake(0.0, 1.0, -3.0);
    Vertex3D    vertex2 = Vertex3DMake(1.0, 0.0, -3.0);
    Vertex3D    vertex3 = Vertex3DMake(-1.0, 0.0, -3.0);
    Triangle3D  triangle = Triangle3DMake(vertex1, vertex2, vertex3);

    Color3D     *colors = malloc(sizeof(Color3D) * 3);
    Color3DSet(&colors[0], 1.0, 0.0, 0.0, 1.0);
    Color3DSet(&colors[1], 0.0, 1.0, 0.0, 1.0);
    Color3DSet(&colors[2], 0.0, 0.0, 1.0, 1.0);

    glLoadIdentity();
    glClearColor(0.7, 0.7, 0.7, 1.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glEnableClientState(GL_VERTEX_ARRAY);
    glEnableClientState(GL_COLOR_ARRAY);
    glColor4f(1.0, 0.0, 0.0, 1.0);
    glVertexPointer(3, GL_FLOAT, 0, &triangle);
    glColorPointer(4, GL_FLOAT, 0, colors);
    glDrawArrays(GL_TRIANGLES, 0, 9);
    glDisableClientState(GL_VERTEX_ARRAY);
    glDisableClientState(GL_COLOR_ARRAY);

    if (colors != NULL)
        free(colors);
}
```

运行，屏幕如下：

今天我们还要讨论一个话题。如果我们不止一次使用一个顶点（三角形条或三角形扇的相邻顶点除外），我们至今使用的方法存在一个问题，我们必须多次向OpenGL传递同一顶点。那不是什么大问题，但通常我们需要尽量减少向OpenGL传递的数据量，所以一次又一次地传递同一个4字节浮点数是非常地不理想。在一些物体中，某个顶点会用在七个甚至更多的不同三角形中，因此你的顶点数组可能会增大许多倍。

当处理这类复杂的几何体时，有一种方法可以避免多次传递同一个顶点，就是使用通过顶点对应于顶点数组中的索引的方法，此方法称之为**元素（elements）**。其原理是创建一个每个顶点只使用一次的数组。然后使用另一个使用最小的无符号整型数的数组来保存所需的唯一顶点号。换句话说，如果顶点数组具有小于256个顶点，那么你应创建一个GLubyte数组，如果大于 256，但小于 65,536，应使用 GLushort。你可以通过映射顶点在第一个数组中的索引值来创建三角形（或其他形状）。所以，如果你创建了一个具有12个顶点的数组，那么数组中的第一个顶点为0。你可以按以前一样的方法绘制图形，只不过不是调用glDrawArrays()，而是调用不同的函数 glDrawElements()并传递整数数组。

让我们以一个真实的，如假包换的3D形状来结束我们的教程：一个二十面体。每个人都使用正方体，但我们要更怪异一点，我们要画一个二十面体。替换drawView：

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

    static GLfloat rot = 0.0;

    // 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,
    };

    glLoadIdentity();
    glTranslatef(0.0f,0.0f,-3.0f);
    glRotatef(rot,1.0f,1.0f,1.0f);
    glClearColor(0.7, 0.7, 0.7, 1.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glEnableClientState(GL_VERTEX_ARRAY);
    glEnableClientState(GL_COLOR_ARRAY);
    glVertexPointer(3, GL_FLOAT, 0, vertices);
    glColorPointer(4, GL_FLOAT, 0, colors);

    glDrawElements(GL_TRIANGLES, 60, GL_UNSIGNED_BYTE, icosahedronFaces);

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

运行，屏幕上将看到一个漂亮的旋转物体：

因为我们没有使用光源而且即使使用了光源我们也没有告诉OpenGL怎样进行光源反射，所以它看上去还不是一个完美的3D物体。有关光线的部分将在后续文章中讨论。

这里我们做了什么？首先，我们建立了一个静态变量来跟踪物体的旋转。

```
static GLfloat rot = 0.0;
```

然后我们定义顶点数组。我们使用了一个与前不同的方法，但结果是一样的。由于我们的几何体根本不会变化，所以我们将其定义为const，这样就不需要每一帧都分配/清除内存：

```
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]
};
```

然后用同样方法建立一个颜色数组 。创建一个Color3D 对象数组，每项对应于前一个数组的顶点：

```
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}
    };
```

最后，创建二十面体。上述十二个顶点本身并未描述形状。OpenGL需要知道怎样将它们联系在一起，所以我们创建了一个整型数组（ GLubyte）指向构成各三角形的顶点。

```
    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,
    };
```

二十面体的第一个面的三个数是1,2,6，代表绘制处于索引1 (0.850651, 0, 0.525731)， 2 (0.850651, 0, 0.525731)， 和6 (0.525731, 0.850651, 0)之间的三角形。
下面一段代码没有新内容，只是加载单元矩阵（所以变换复位），移动并旋转几何体，设置背景色，清除缓存，启动顶点和颜色数组，然后提供顶点数组数据给OpenGL。所有这些在前一个例子中都有涉及。

```
    glLoadIdentity();
    glTranslatef(0.0f,0.0f,-3.0f);
    glRotatef(rot,1.0f,1.0f,1.0f);
    glClearColor(0.7, 0.7, 0.7, 1.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glEnableClientState(GL_VERTEX_ARRAY);
    glEnableClientState(GL_COLOR_ARRAY);
    glColor4f(1.0, 0.0, 0.0, 1.0);
    glVertexPointer(3, GL_FLOAT, 0, vertices);
    glColorPointer(4, GL_FLOAT, 0, colors);
```

然后，我们没有使用glDrawArrays()。而是调用glDrawElements()：

```
    glDrawElements(GL_TRIANGLES, 60, GL_UNSIGNED_BYTE, icosahedronFaces);
```

接着，执行禁止功能，根据距上一帧绘制的时间增加旋转变量值：

```
    glDisableClientState(GL_VERTEX_ARRAY);
    glDisableClientState(GL_COLOR_ARRAY);
```

```
    static NSTimeInterval lastDrawTime;
    if (lastDrawTime)
    {
        NSTimeInterval timeSinceLastDraw = [NSDate timeIntervalSinceReferenceDate]
- lastDrawTime;
        rot+=50 * timeSinceLastDraw;
    }
    lastDrawTime = [NSDate timeIntervalSinceReferenceDate];
```

记住：如果你按绘制的正确次序提供顶点，那么你应该使用`glDrawArrays()`，但是如果你提供一个数组然后用另一个以索引值区分顶点次序的数组的话，那么你应该使用`glDrawElements()`。

请花些时间测试绘图代码，添加更多的多边形，改变颜色等。OpenGL绘图功能远超过本文涉及的内容，但你应该清楚iPhone上3D物体绘制的基本概念了：创建一块内存保存所有的顶点，传递顶点数组给OpenGL，然后由OpenGL绘制出来。

源码下载

最新评论 发表评论

Mr_yl 2014-3-5 10:51 引用

为什么我点击下载就跳到便民导航了

天人合一001 2014-2-7 16:01 引用

顶一下！

luozhonglan 2013-12-13 10:15 引用

源码不能下载，求发邮箱409347461@qq.com

々枫々 2013-9-25 11:43 引用

顶顶顶~~~~受益匪浅～～～

sunwulovelove1 2013-7-8 15:11 引用

新人来了！顶你下。多谢！

rainselect 2013-3-1 21:01 引用

模板怎么下载呀？下载不了呀？