[转载]从零开始学习OpenGL ES之一 - 基本概念

2011-9-13 22:27 | 发布者: Iven | 查看: 45283 | 评论: 41

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

我曾写过一些文章介绍iPhone OpenGL ES编程，但大部分针对的是已经至少懂得一些3D编程知识的人。

作为起点，请下载我的OpenGL Xcode项目模板，而不要使用Apple提供的模板。你可以解压到下面位置安装此模板：

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

已经有大量有关OpenGL的好教程和书籍。但是，却没有多少是关于OpenGL ES，而且没有（至少在我撰写此文时）是专门针对学习iPhone上3D编程的。因为大部分有关学习OpenGL的材料是从所谓"直接模式（direct mode）"开始的，而OpenGL ES并不支持此模式，对于没有3D背景知识的iPhone开发者而言，使用现有的书籍和教程是十分困难的。为满足一些开发者的要求，我决定撰写一个针对3D初学者的博文系列。这是此系列的第一篇文章。

OpenGL 数据类型

首先我们要讨论的是OpenGL的数据类型。因为OpenGL是一个跨平台的API，数据类型的大小会随使用的编程语言以及处理器（64位，32位，16位）等的不同而不同，所以OpenGL定义了自己的数据类型。当传递数据到OpenGL时，你应该坚持使用这些OpenGL的数据类型，从而保证传递数据的尺寸和精度正确。不这样做的后果是可能会导致无法预料的结果或由于运行时的数据转换造成效率低下。不论平台或语言实现的OpenGL都采用这种方式定义数据类型以保证在各平台上数据的尺寸一致，并使平台间OpenGL代码移植更为容易。

下面是OpenGL的各种数据类型:

- **GLenum**: 用于GL枚举的无符号整型。通常用于通知OpenGL由指针传递的存储于数组中数据的类型（例如，GL_FLOAT用于指示数组由GLfloat组成）。
- **GLboolean**: 用于单布尔值。OpenGL ES还定义了其自己的"真"和"假"值（GL_TRUE和GL_FALSE）以避免平台和语言的差别。当向OpenGL传递布尔值时，请使用这些值而不是使用YES或NO（尽管由于它们的定义实际没有区别，即使你不小心使用了YES或NO。但是，使用GL-定义值是一个好的习惯。）
- **GLbitfield**: 用于将多个布尔值（最多32个）打包到单个使用位操作变量的四字节整型。我们将在第一次使用位域变量时详细介绍，请参阅 wikipedia
- **GLbyte**: 有符号单字节整型，包含数值从-128 到 127
- **GLshort**: 有符号双字节整型，包含数值从−32,768 到 32,767
- **GLint**: 有符号四字节整型，包含数值从−2,147,483,648 到 2,147,483,647
- **GLsizei**: 有符号四字节整型，用于代表数据的尺寸（字节），类似于C中的`size_t`
- **GLubyte**: 无符号单字节整型，包含数值从0 到 255。
- **GLushort**: 无符号双字节整型，包含数值从0 到 65,535
- **GLuint**: 无符号四字节整型，包含数值从0 到 4,294,967,295
- **GLfloat**: 四字节精度IEEE 754-1985 浮点数
- **GLclampf**: 这也是四字节精度浮点数，但OpenGL使用GLclampf特别表示数值为0.0 到 1.0
- **GLvoid**:void值用于指示一个函数没有返回值，或没有参数
- **GLfixed**: 定点数 使用整型数存储实数。由于大部分计算机处理器在处理整型数比处理浮点数快很多，这通常是对3D系统的优化方式。但因为iPhone具有用于浮点运算的矢量处理器，我们将不讨论定点运算或**GLfixed数据类型。**
- **GLclampx**: 另一种定点型，用于使用定点运算来表示0.0 到 1.0之间的实数。正如**GLfixed**，我们不会讨论或使用它。

OpenGL ES （至少iPhone目前所使用的版本）不支持8字节（64位）数据类型，如long或double。OpenGL 其实具

相关分类

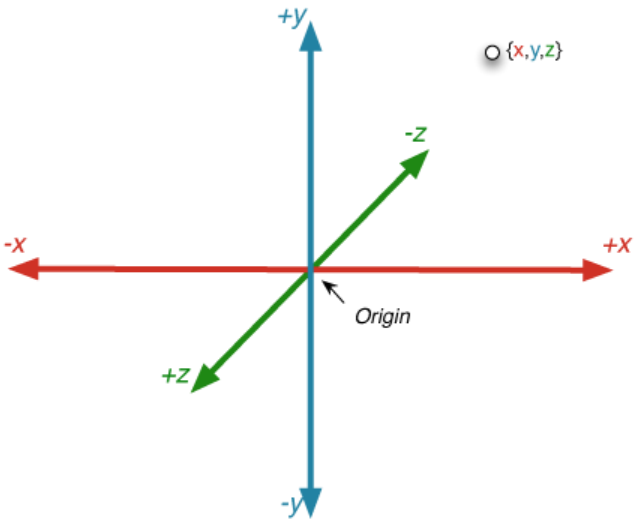
[iTyran原 [翻译]OpenGL ES
[子龙山人翻 从零开始学习

有这些大型数据类型，但考虑到大部分嵌入式设备屏幕尺寸以及可能为它们所写的程序类型而且使用它们有可能对性能造成不利的影响，最后的决定是在OpenGL ES中排除这些数据类型。

点或顶点

3D图像的最小单位称为 **点（point）** 或者 **顶点vertex**。它们代表三维空间中的一个点并用来建造更复杂的物体。多边形就是由点构成，而物体是由多个多边形组成。尽管通常OpenGL支持多种多边形，但OpenGL ES只支持三边形（即三角形）。

如果你回忆高中学过的几何学，你可能会记得所谓笛卡尔坐标。 基本概念是在空间中任选一点，称作**原点**。 然后你可以通过参照原点并使用三个代表三维的数值指定空间中的任意一点，坐标是由三个想象的通过原点线表示的。从左至右的想象直线叫x-轴。沿着x-轴从左至右数值变大，向左移动数值变小。原点左方x为负值，右边为正值。另外两轴同理。沿y轴向上，y值增加，向下y值减小。原点上方y为正，原点下方为负。对于z轴，当物体离开观察者，数值变小，向观察者移动（或超出观察者），数值变大。原点前方z值为正，原点之后为负。下图帮助说明了这一点：



> **Note:** iPhone上另一种绘图框架Core Graphics使用了稍微不同的坐标系统，当向屏幕上方移动时y值减小，而向下移动y值增加。

沿各轴增加或减小的数值是以任意刻度进行的 – 它们不代表任何真实单位，如英尺，英寸或米等。你可以选择任何对你的程序有意义的刻度。如果你想设计的游戏以英尺为单位，你可以那样做。如果你希望单位为毫米，同样可行。OpenGL不管它对最终用户代表什么，只是将它作为单位处理，保证它们具有相同的距离。

由于任何物体在三维空间中的方位可以由三个数值表示，物体的位置通常在OpenGL中由使用一个三维数组的三个GLfloat变量表示，数组中的第一项（索引0）为x位置，第二项（索引1）为y位置，第三项（索引2）为z位置。下面是一个创建OpenGL ES顶点的简单例子：

```
GLfloat vertex[3];
vertex[0] = 10.0;       // x
vertex[1] = 23.75;      // y
vertex[2] = -12.532;    // z
```

在OpenGL ES中，通常将场景中所有构成所有或部分物体的提交为**顶点数组**。一个顶点数组是包括场景中部分或所有顶点数据的简单数组。我将在系列的下一篇教程中讨论，有关顶点数组要记住的是它们的大小是基于呈现的顶点数乘以三（三维空间绘图）或二（二维空间绘图）。所以一个包含六个三维空间中的三角形的顶点数组由54个GLfloat组成，因为每个三角形有三个顶点，而每个顶点有三个坐标，即6 x 3 x 3 = 54。

处理所有这些**GLfloat**是很痛苦的事情。幸运的是，有一个容易的方法。我们可以定义一个数据结构了保存多个顶点，像这样：

```
typedef struct {
    GLfloat x;
    GLfloat y;
    GLfloat z;
} Vertex3D;
```

通过这样做，我们的代码可读性更强：

```
Vertex3D vertex;
vertex.x = 10.0;
```

```
vertex.y = 23.75;
vertex.z = -12.532;
```

现在由于Vertex3D由三个GLfloat组成，向Vertex3D传递指针与向数组传递一个包含三个GLfloat的数组的指针完全一样。对于电脑而言毫无分别；两者具有同样的尺寸和同样的字节数以及OpenGL需要的同样的顺序。将数据分组到数据结构只是让程序员感到更容易，处理起来更方便。如果你下载了文章开头处的Xcode模板，你会发现此数据结构以及我后面将讨论的各种函数都定义在文件OpenGLCommon.h中。还有一个内联函数用于创建单个顶点：

```
static inline Vertex3D Vertex3DMake(CGFloat inX, CGFloat inY, CGFloat inZ)
{
    Vertex3D ret;
    ret.x = inX;
    ret.y = inY;
    ret.z = inZ;
    return ret;
}
```

如果你回忆起几何学（如果不记得也不要紧）的内容，你会知道空间中两点间的距离是使用下面公式计算的：

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (Z_2 - Z_1)^2}$$

我们可以在一个简单的内联函数中实现这个公式来计算三维空间中任何两点间的直线距离：

```
static inline GLfloat Vertex3DCalculateDistanceBetweenVertices (Vertex3D first, Vertex3D second)
{
    GLfloat deltaX = second.x - first.x;
    GLfloat deltaY = second.y - first.y;
    GLfloat deltaZ = second.z - first.z;
    return sqrtf(deltaX*deltaX + deltaY*deltaY + deltaZ*deltaZ );
};
```

三角形

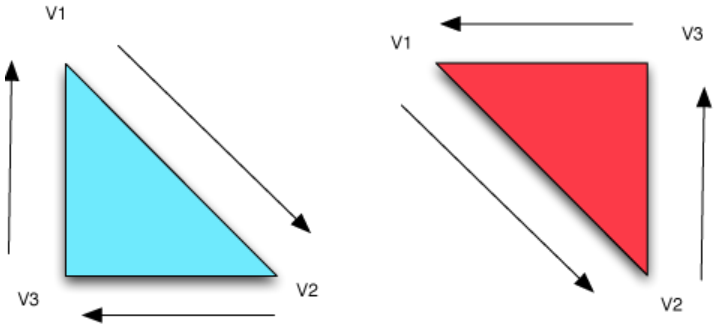
由于OpenGL ES仅支持三角形，因此我们可以通过创建一个数据结构将三个顶点组合成一个三角形物体。

```
typedef struct {
    Vertex3D v1;
    Vertex3D v2;
    Vertex3D v3;
} Triangle3D;
```

一个 Triangle3D实际上与一个九个GLfloat构成的数组是完全一样的，因为我们通过顶点和三角形而不是GLfloat数组来构建物体，所以它能帮助我们更容易地处理我们的代码。

然而关于三角形你需要知道更多的事情。在OpenGL中有一个概念叫**卷绕(winding)**， 它表示顶点绘制的次序是重要的。不像真实世界中的物体，OpenGL中的多边形通常都不会有两面。它们只有一面，被当做**front face（前面）**， 三角形只有其front face面对观察者时才可见。可以设置OpenGL将多边形作为两面处理，但默认状态下，三角形只有一个可见面。通过知道哪一个面是多边形的前面或可见面，才能使OpenGL只做一半的计算。

尽管有时多边形也可以独立存在，需要绘制其背面，但通常三角形是一个大物体的一部分，其面对物体内部的一面永远也不可见。不被绘制的一面称为**backface（背面）**，OpenGL是通过观察顶点的绘制次序来确定front face和backface的。以反时针次序绘制顶点的构成的面是frontface（默认，可以改变）。由于OpenGL可以很容易确定哪个三角形对用户可见，所以它使用了一种称为**Backface Culling（隐面消除）** 的技术来避免绘制视窗中多边形的不可见面。下一篇文章将讨论视窗，现在你可将其想象成一个虚拟摄像或观察OpenGL世界的虚拟窗口。



上图中，左边青色的三角形是backface，因此将不可见。而右方的三角形是frontface，所以将被绘制。

本系列的下一篇文章将设定一个OpenGL的虚拟世界并使用`Vertex3D` 和 `Triangle3D`进行一些基本绘图。再后，我们将讨论**变换**，它使用线性代数在虚拟世界中移动物体。

36

23

1 1

鲜花 握手 雷人 路过 鸡蛋

刚表态过的朋友 (61 人)

Jun0631 dtzhang parrot 花街无心 x1312520 jnetzhou RussellWait 浅底 reader づ牛拦痴女

々枫々 mo_myth iskyos wzx11011... 蝈蝈2011 cgw0827 怀旧 舍得333 度娘818 伊然

09jianfeng xiupoman yuxiang11... yuzhou164

邀请 收藏

最新评论 发表评论

烧饵块 2014-12-10 23:35 引用

编译不过，求解答！

parrot 2014-5-5 11:23 引用

开始学习下

花街无心 2014-3-31 20:50 引用

很基础，很好

紫色空间 2014-1-8 10:37 引用

顶一个

々枫々 2013-9-25 09:52 引用

若模板编译不过，出现：“ARC forbids explicit message send of release”错误！可参考下小弟的博文：http://blog.csdn.net/rexuefengye/article/details/11991183。

熊猫正正 2013-5-24 00:12 引用

学习，学习～～

白色恋歌 2013-4-29 14:07 引用

模板编译不过啊、、

cnchenjie 2013-4-12 11:35 引用