数字媒体技术专业计算机图形学推荐教材

Computer Graphics
OpenGL
Programming

C#版

计算机图形学

OpenGL三维渲染

赵辉　王晓玲　著

海洋出版社

数字媒体技术专业计算机图形学推荐教材

Computer Graphics
OpenGL
Programming

C#版

计算机图形学

OpenGL三维渲染

赵辉　王晓玲　著

海洋出版社
2016年·北京

内 容 简 介

“数字媒体技术专业计算机图形学”系列丛书服务于计算机图形学教学，并可满足从业技术需求。本书为其中的OpenGL三维渲染分册。

主要内容： 全书共13章，囊括了OpenGL点线面的绘制、光照模型、渲染材质、OpenGL状态和信息、OpenGL缓冲区、透明渲染、坐标变换、投影变换、四元数、鼠标界面等算法实现。

本书特色： 1. 为新形势下计算机图形学教学及从业需要打造。2. 精练OpenGL三维渲染的理论知识，便于入门。3. 技术要点与算法实现对应讲解，层次清晰。4. 选用C#语言编写代码，上手快捷，易于变通。5. 三维交互、虚拟现实类课程的基础。6. 可与三维模型处理算法同步学习。

读者对象：

- 中高等院校数字媒体技术及软件工程相关专业学生。
- 计算机图形、三维动画、虚拟现实领域从业人士及爱好者。

图书在版编目(CIP)数据

计算机图形学——OpenGL三维渲染（C#版）/ 赵辉，王晓玲著. —北京：海洋出版社，2016.1
ISBN 978-7-5027-9356-2

Ⅰ. ①计… Ⅱ. ①赵… ②王… Ⅲ. ①图形软件—程序设计 Ⅳ. ①TP391.41

中国版本图书馆CIP数据核字（2016）第020134号

总 策 划：张曌嫘
责任编辑：张曌嫘　张鹤凌
责任校对：肖新民
责任印制：赵麟苏
排　　版：申　彪

出版发行：海洋出版社

地　　址：北京市海淀区大慧寺路8号（707房间）100081
经　　销：新华书店
技术支持：（010）62100050　graphicsresearch@qq.com

发 行 部：（010）62174379（传真）（010）62132549
（010）68038093（邮购）（010）62100077
网　　址：www.oceanpress.com.cn
承　　印：北京画中画印刷有限公司
版　　次：2016年1月第1版
2016年1月第1次印刷
开　　本：889mm×1194mm　1/16
印　　张：13.5（全彩印刷）
字　　数：490千字
印　　数：1～3000册
定　　价：79.00元

前言 Foreword

2008年以来，全国各个高等院校纷纷在各自软件工程学科专业的基础上开设了数字媒体技术专业。数字媒体技术专业和计算机科学专业的区别是，前者主要学习与二维图像和三维图形相关的算法和应用开发，而后者还需要学习其他与计算机科学相关的知识。由于开设和建立时间短，各学校的数字媒体技术专业的教学工作都还处在摸索阶段，没有形成统一、成熟的教材体系。根据在数字媒体技术专业多年的教学实践经验，我们总结出本专业要以计算机三维图形学的理论和算法为基础，以三维应用开发为导向进行建设。

根据多年一线教学经验与反馈以及当前的三维图形学研究成果，我们编写了本套丛书。本套丛书涵盖了三维图形学算法的三大方面：模型、动画和渲染。内容根据数字媒体技术专业的教学特点分散到5本计算机图形学系列书中：《三维模型处理算法初步：理论与实现（C#版）》《三维模型处理算法进阶：理论与实现（C#版）》《三维模型参数化：理论与实现（C#版）》《OpenGL编程（C#版）》和《GLSL渲染编程》。通过本系列专业基础教材，再加上已有的成熟的计算机基础编程教材以及三维软件使用的教材，就可以完整地覆盖数字媒体技术专业的所有课程。

书里的代码采用C#编程语言。C#编程语言是一种结合了C++和Java优点的编程语言。C#语言相对于其他编程语言来说比较容易学习和掌握，但是本套丛书里讲述的原理和算法不仅限于C#语言。读者可以通过示例中的代码，采用自己熟悉的编程语言来进行编程。本套丛书包含了很多计算机图形学会议Siggraph论文里具有突破性的图形学算法分析与实现。

除了三维模型、动画的理论和算法实现之外，本套丛书也包含了三维渲染OpenGL和GLSL渲染编程的讲解，通过对渲染实例的讲解介绍如何调用OpenGL和GLSL以得到实时的渲染效果。相比于之前介绍OpenGL编程的作品，本书更加注重通过对实例的讲解，降低学习的难度。GLSL渲染编程需要在OpenGL编程之后学习。OpenGL编程和其他几门课程可以并行进行。三维模型处理算法进阶、三维模型参数化算法的学习需要在掌握三维模型算法初步的知识及操作基础后方可进行。

《OpenGL三维渲染（C#版）》介绍了OpenGL进行三维渲染的方法，并用大量实例来展示如何进行OpenGL编程。本书一共分为13章，详细讲述了OpenGL点、线、面的绘制；光照模型；渲染材质；OpenGL状态和信息；OpenGL缓冲区；透明

渲染；坐标变换；投影变换；四元数；鼠标界面等算法实现。本书的特点是以各种渲染实例为核心，而不是仅仅讲述OpenGL的函数。通过本书的学习可以快速掌握OpenGL的编程。

本书不仅可以作为数字媒体技术专业的专业基础课，还可以作为计算机学科和软件工程学科“数据结构和算法”“计算机图形学”等课程的教材和参考书。

赵　辉

2015年10月于美国哈佛大学

目录 Contents

1 OpenGL介绍

1.1 OpenGL特点

OpenGL是进行三维显示的函数库，与平台无关，能在Linux、Windows及Mac OS等操作系统上运行。另一个类似的函数库是DirectX，但它只能在微软的操作系统上运行。OpenGL是高性能图形显示的标准。包括ATT公司、UNIX软件实验室、IBM公司、DEC公司、SUN公司、HP公司、Microsoft公司和SGI公司在内的多家在计算机市场占主导地位的大公司都采用OpenGL图形标准。OpenGL在硬件、窗口及操作系统方面是相互独立的，也就是说，OpenGL能不依赖于任何硬件或者操作系统即可运行。OpenGL本身只定义了一个标准，因此只要符合这个标准的函数库实现，都可以称为OpenGL标准的函数库。除了通常使用的在Windows操作系统上运行的OpenGL函数库之外，Mesa3D也是一个常用的实现OpenGL标准的函数库，Mesa3D可以运行在Linux操作系统上。

OpenGL标准定义的函数库分为4个部分：OpenGL核心库、OpenGL实用库、OpenGL辅助库及OpenGL工具库。

（1）核心库

核心库包含115个函数，函数名的前缀为gl。这部分函数用于常规的、核心的图形处理。此函数封装在gl.dll中。同一个功能的函数可以输入不同类型的参数，因此从基本的115个函数中派生出来的函数达300多个。

（2）实用库

实用库包含43个函数，函数名的前缀为glu。OpenGL提供了强大的但是为数不多的绘图命令，所有三维图形的绘制都必须从点、线、面开始。glu函数通过调用核心库的函数，封装了OpenGL函数，为开发者提供了相对简单的用法，实现了一些较为复杂的操作。这些函数封装在glu.dll中。OpenGL中的核心库和实用库可以在所有的OpenGL平台上运行。

（3）辅助库

辅助库包含31个函数，函数名前缀为aux。 这部分函数提供窗口管理、输入输出处理以及绘制一些简单的三维物体。这些函数封装在glaux.dll中。

（4）工具库

工具库包含30多个函数，函数名前缀为glut。glut是不依赖于操作系统平台的OpenGL工具包，目的是隐藏不同操作系统平台API的复杂度。函数以glut开头，它们作为aux库功能更强的替代品，提供更为复杂的绘制功能，此函数由glut.dll来负责解释执行。由于glut中的窗口管理函数是不依赖于运行环境的，因此OpenGL中的工具库可以在各种操作系统中运行。

本书只讲解OpenGL的核心库，着重学习OpenGL如何对三维模型进行渲染。也就是说，如何把一个三维模型加载到内存后，用OpenGL来显示出各种不同的渲染效果。本书采用C#语言来讲解，主要原因是C#语言比C++更容易使用，从而可以使读者减轻学习的负担。 Windows上的OpenGL本身是C++版本的函数库，因此需要把OpenGL的C++库进行封装，从而提供给C#语言进行调用。 The Open Took Kit (OpenTK)就是对 OpenGL的一种跨平台的封装，使用 C# 编写，可以运行在 Windows、 Linux 以及 Mac

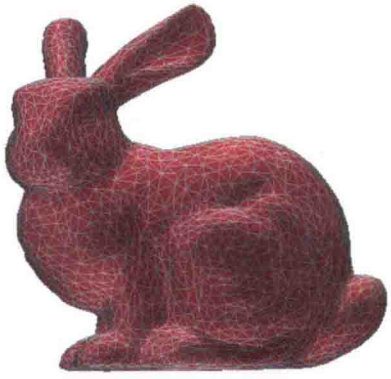
OS等操作系统平台上， 任何 .Net 语言都可以使用它做开发。 除OpenTK之外，还有其他的支持C#编程的OpenGL函数库。

OpenTK把OpenGL函数封装为3个dll，即OpenTK.dl、OpenTK.GLControl.dll和OpenTK.Compatibility.dll。OpenTK.dll包含OpenGL的核心库；OpenTK.GLControl.dll主要用于界面显示和鼠标键盘交互；OpenTK.Compatibility.dll提供了一些辅助的函数。本书集中讲述OpenGL核心库的使用。

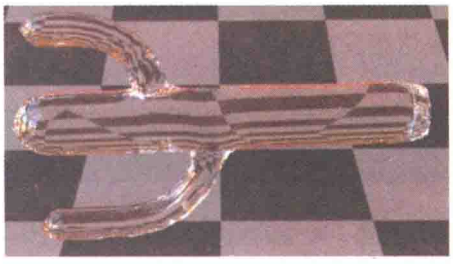
OpenGL函数库是一个实时渲染的函数库。与基于物理的光照不同，OpenGL函数渲染的光照是近似的光照，因此能够在短时间进行实时交互绘制；而基于物理的光照则需要长时间的绘制。基于物理光照的渲染引擎，有Luxrender和Indigo等， 3dsMax、Maya等三维软件中也提供了基于物理光照的渲染引擎。基于物理的光照渲染方法渲染出来的结果是照片级别的，也就是基于物理光照渲染方法不是用OpenGL来进行渲染，而是根据光线传播的规律来进行物理计算，从而借助某些视角看到整个的场景图像效果。由于要进行大量的数值计算，因此基于物理的光照渲染耗时长，主要用在不要求实时或者交互的应用中，如电影特效等。这些应用追求的是最终效果的真实感，而不是实时地得到渲染效果。基于物理的光照几乎可以渲染出任意真实的光照效果，得到比较逼真的或近似照相机拍摄的照片效果。 而OpenGL提供的是一个能够实时渲染的方式，是对光照传播定理的近似，也就是OpenGL的光照计算是对真实光照规律的一个特殊的近似，它计算速度非常快，但是也可以看出光照的效果。之所以定义OpenGL的标准，是因为这样可以减少重复开发，每个三维实时渲染的应用，都是基于同样的方式，因此这个方式可以形成一个标准，避免每个应用单独开发自己的实时渲染函数库。虽然基于物理的光照渲染比OpenGL渲染出来的效果要真实得多，但是OpenGL光照适用于游戏、虚拟现实等需要实时交互的应用。本书主要讲述OpenGL实时渲染方法和示例。

表1-1是各种三维模型OpenGL渲染结果与基于物理光照渲染效果对比。

表1-1　OpenGL渲染结果与基于物理光照渲染效果对比

OpenGL渲染结果	基于物理光照渲染效果
	
	

续表

OpenGL渲染结果	基于物理光照渲染效果
	
	

目前已经有了大量的OpenGL方面的图书，但是这些书大部分都是讲述OpenGL单个函数的功能和使用方法的。而要渲染出一个三维场景，需要把很多单个的OpenGL函数组合起来。从单个函数的使用，到一个渲染效果的实现还有一段很长的学习道路要走。本书不是简单的OpenGL函数的枚举、罗列，而是以大量的实例来显示如何把各种OpenGL函数组合来实现某个特定的渲染效果，从而使读者快速地掌握和学习如何使用OpenGL来进行三维场景的渲染。这是本书的一个特点。本书的另一个特点是不围绕OpenGL单个函数进行说明，这样可避免本书成为一本函数使用手册，而是通过围绕着一个显示效果，组合使用多个函数来进行讲述，这样可以使读者更好地理解三维渲染方法和OpenGL使用。 本书也讲述了如何对OpenGL进行封装，使外部代用OpenGL更加简单和方便；也就是在应用程序开发中，如何把基础的、不是面向对象的OpenGL函数库封装为面向对象的高一层函数库，从而不需要了解三维渲染的理论就可以把三维场景渲染出来。本书借助三维渲染的方法和原理，讲述OpenGL函数本身如何实现三维渲染，以及如何封装OpenGL函数库为三维渲染模块这两个层次，读者由此可以完整地掌握OpenGL三维渲染的编程。

1.2 渲染流程

在OpenGL里，所有的三维图形，如三维球、茶壶、兔子等都用顶点来描述，因此三维模型上的操作都可以针对每个顶点进行计算和操作，然后进行光栅化形成二维的像素。对于像素数据，像素操作结果被存储在纹理组装用的内存中，也就是说，对于OpenGL来说输入是三维的顶点，输出是二维的图像。对于OpenGL来说，一个核心的模块是变换，也就是把三维模型变换为二维图像；OpenGL的另一个核心的模块是显示，也就是通过各种设置把同样的三维模型显示为不同的二维图像。

在OpenGL中进行主要的图形操作直至在计算机屏幕上渲染绘制出三维图形景观的基本步骤如下。

第一步：输入三维模型要渲染的顶点坐标、面等几何信息。

第二步：把三维模型放到三维场景合适的位置，设置摄像头的位置和视角。

第三步：设置光照的位置、颜色、方向、类型等参数。

第四步：设置三维模型的颜色、纹理贴图等材质参数。

第五步：把上述三维信息转化为二维的图像。

在这些步骤的执行过程中，OpenGL可能执行其他的一些操作，如自动消隐处理等。三维模型转换为二维图像后，还可以根据需要对像素数据进行操作。使用OpenGL来进行三维渲染的主要工作就是调用

OpenGL函数进行各种输入和输出的设置，具体的渲染工作都是由OpenGL提供的函数本身来实现的。除了输入输出之外，OpenGL是基于状态的函数库，有很多全局的状态开关，通过这些开关来判断如何进行渲染。OpenGL另一个重要的模块是三维矩阵变换；也就是三维模型的移动、旋转和缩放，从三维转为二维都需要进行矩阵变换。这些变换是由OpenGL函数内部来实现的，外部调用的时候只需要设置好各种变换矩阵，或者变换参数，由OpenGL来自动生成变换矩阵。

虽然OpenGL把各种功能都内置封装到函数里面，只需要设置相应的参数就可以使用OpenGL进行渲染。但是只有了解了OpenGL封装函数实现的原理，才能够更好地掌握三维渲染的本质。本书将理论和应用结合起来，一方面讲述了三维渲染的原理，另一方面讲述如何调用OpenGL来实现，或者在不调用OpenGL的情况下，如何实现相应的功能。例如，对于变换来说，OpenGL提供了各种变换矩阵，外部只需要设置相应的参数就可以调用OpenGL的函数来自动得到变换矩阵，但是同时也可以不用OpenGL，而用代码生成这些变换矩阵。

1.3 函数分类

OpenGL包含了大量的函数，需要对这些函数进行分类，有助于快速学习。本书中列举的OpenGL函数名称都是OpenTK中封装过的符合C#命名规则的函数。OpenGL核心库中的函数主要可以分为以下10类。

第一类：绘制基本几何图元的函数。如绘制图元的函数GL.Begin()、GL.End()、GL.Normal*()、GL.Vertex*()。

第二类：矩阵操作、几何变换和投影变换的函数。如矩阵入栈函数GL.PushMatrix()，矩阵出栈函数GL.PopMatrix()，装载矩阵函数 GL.LoadMatrix()，矩阵相乘函数GL.MultMatrix()，当前矩阵函数GL.MatrixMode()，矩阵标准化函数GL.LoadIdentity()，几何变换函数GL.Translate*()、GL.Rotate*()和GL.Scale*()，投影变换函数GL.Ortho()、GL.Frustum()和视口变换函数GL.Viewport()等。

第三类：颜色、光照和材质的函数。如设置颜色模式函数GL.Color*()、GL.Index*()，设置光照效果函数GL.Light*() 和GL.LightModel*()，设置材质效果函数GL.Material()等。

第四类：显示列表函数。主要有创建、结束、生成、删除和调用显示列表的函数GL.NewList()、GL.EndList()、GL.GenLists()、GL.CallList()和GL.DeleteLists()。

第五类：纹理映射函数。主要有一维纹理函数GL.TexImage1D()，二维纹理函数GL.TexImage2D()，设置纹理参数、纹理环境和纹理坐标的函数GL.TexParameter*()、GL.TexEnv*()和GL.TetCoord*()等。

第六类：特殊效果函数。主要有融合函数GL.BlendFunc()，反走样函数GL.Hint()和雾化效果GL.Fog*()。

第七类：光栅化、像素操作函数。如像素位置GL.RasterPos*()，线型宽度GL.LineWidth()，多边形绘制模式GL.PolygonMode()，读取像素GL.ReadPixel()，复制像素GL.CopyPixel()等。

第八类：选择与反馈函数。主要有渲染模式GL.RenderMode()，选择缓冲区GL.SelectBuffer()和反馈缓冲区GL.FeedbackBuffer()等。

第九类：曲线与曲面的绘制函数。生成曲线或曲面的函数GL.Map*()和GL.MapGrid*()，求值器的函数GL.EvalCoord*()和GL.EvalMesh*()。

第十类：状态设置与查询函数。主要有GL.Get*()、GL.Enable()和GL.GetError()等。

由于OpenGL标准制定得比较早，不是面向对象的函数库，因此本书中为了使用方便，需要在OpenGL外面封装一层。同时为了能够在外部菜单界面灵活控制，也需要有界面的接口。整个OpenGL框架分为两个函数库：一个是与界面无关的核心函数库；另一个是OpenGL的界面库。这样把OpenGL封装后，外部的调用就更加方便和简单。即使不了解三维渲染的基本方法和原理，只需设置相关的参数，就可以成功地实现各种三维渲染的效果。目前很多游戏引擎在进行三维渲染模块设计时，也是把OpenGL封装为面向对象的高一层的函数库，从而可以不用了解底层的渲染机制。

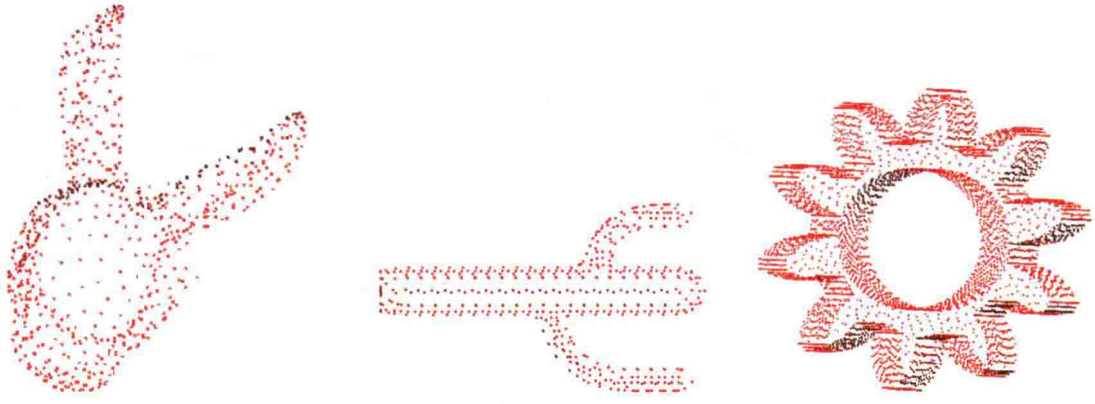
2 点线面基本绘制

2.1 OpenGL点线面概念

点、线、多边形（也称为面）是OpenGL的基本显示单元。复杂的三维模型都由基本显示单元构成。在OpenGL里，点、线、多边形的绘制都封装为OpenGL的内部函数，只需要在外部给OpenGL传递点的坐标，面的构成，以及设置需要绘制的方式；也就是说，在外部调用的时候，只需要告诉OpenGL函数库绘制的元素信息，以及如何对这些元素进行绘制，具体的绘制过程由OpenGL来实现。

1. 顶点

顶点（Vertex）用浮点数来表示。所有顶点在OpenGL内部计算时都作为三维点处理，顶点坐标用齐次坐标(x, y, z, w)表示，如果w不为0.0，这些齐次坐标表示的顶点即为三维空间点$(x/w, y/w, z/w)$。可以设定w值，但很少这样做。一般来说，w缺省为1.0。图2-1中是几个三维模型顶点的显示。



（a）兔子头　　（b）仙人掌　　（c）机械部件

图2-1　三维模型顶点显示

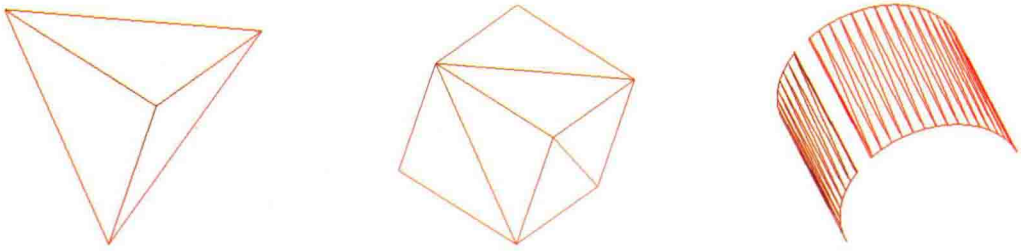
2. 线

在OpenGL中，线代表线段（Line Segment），不是数学意义上的那种沿轴两个方向无限延伸的线。这里的线由一系列顶点顺次连接而成，分闭合和不闭合两种，如图2-2所示。

（a）闭合线　　（b）不闭合线

图2-2　闭合和不闭合的线

图2-3是几种线条显示。



(a) 三棱锥　　(b) 立方体　　(c) 半圆柱

图2-3　线条形状

3. 多边形

OpenGL中的多边形（Polygon）可以是平面多边形，即所有顶点在一个平面上，也可以是空间多边形。OpenGL中定义的多边形是由一系列线段依次连接而成的封闭区域。这些多边形可以是三角形、四边形、五边形、六边形……

2.2　图形元素输入显示

2.2.1　元素输入

1. 输入顶点

在OpenGL中，顶点数据是所有几何模型输入的基础，所有几何物体最终都由有一定顺序的顶点集来描述。顶点数据的输入设置函数如下：

GL.Vertex3(0.0, 1.0, 0.0);

上面这个函数定义了空间中坐标为(0.0, 1.0, 0.0)的一个点。

GL.Vertex3(2.0, 4.0, 5.0);

这个函数定义了空间中坐标为(2.0, 4.0, 5.0)的一个点。

2. 输入图形

三维图形是由线条和面来构成的，这些线条和面又是由一组顶点构成的。因此为了输入线条和面，需要输入顶点。 并且同时需要设置这些顶点构成线条和面的规则，也就是确定哪几个点构成一个线条或者一个多边形。之后，OpenGL会根据设定的规则，把这些输入顶点绘制为相应的线条和面。

在OpenGL中，输入的顶点必须放在GL.Begin和GL.End()两个函数之间。GL.Begin表示顶点输入的开始，GL.End()表示顶点输入的结束。 如图2-4中的例子，定义一个正方形，这个正方形包含4个顶点。

```
public void DemoOne()
{
    GL.Begin(BeginMode.Polygon);
    GL.Vertex3(0.0, 0.0, 0.0);
    GL.Vertex3(0.0, 0.3, 0.0);
    GL.Vertex3(0.3, 0.3, 0.0);
    GL.Vertex3(0.3, 0.0, 0.0);
    GL.End();
}
```

(a) 代码设置

(b) 显示结果

图2-4　正方形显示

GL.Begin这个函数的输入参数是设置顶点的解析规则，OpenGL根据这里的设置，来解释下面的顶点按照怎样的规则来构成多边形、线条，或者不构成多边形，只是显示顶点本身。如图2-5的例子所示，虽然输入的顶点都一样，但是因为GL.Begin的输入参数设置不一样，OpenGL显示出的不是多边形，而是点。

```
public void DemoOne()
{
    GL.Begin(BeginMode.Points);
    GL.Vertex3(0.0, 0.0, 0.0);
    GL.Vertex3(0.0, 0.3, 0.0);
    GL.Vertex3(0.3, 0.3, 0.0);
    GL.Vertex3(0.3, 0.0, 0.0);
    GL.End();
}
```

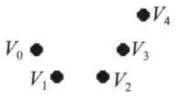
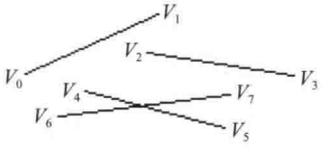
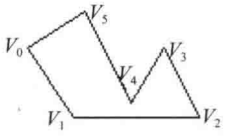
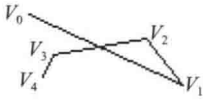
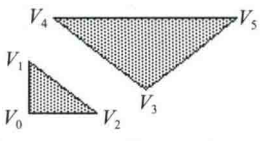
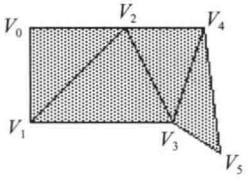
(a) 代码设置　　(b) 显示结果

图2-5　4个顶点显示

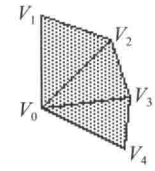
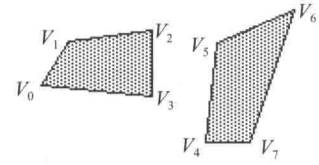
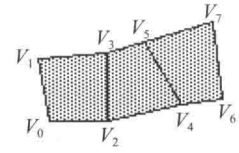
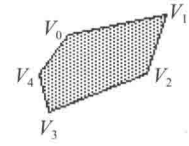
2.2.2　输入构成解析

GL.Begin函数根据输入参数的不同，把同样的输入显示为不同的输出结果，这个函数的输入参数有以下几种，见表2-1。表中V表示顶点。

表2-1　GL.Begin输入函数种类、含义及图示

参数名	含义	图示
Points	把每一个顶点作为一个点进行处理	
Lines	把每一个顶点作为一个独立的线段，顶点$2n-1$和$2n$之间共定义了n条线段，总共绘制$n/2$条线段	
LineLoop	绘制从第一个顶点到最后一个顶点依次相连的一组线段，然后最后一个顶点和第一个顶点相连，第n和第$n+1$个顶点定义了线段n，总共绘制n条线段	
LineStrip	绘制从第一个顶点到最后一个顶点依次相连的一组线段，第n和第$n+1$个顶点定义了线段n，总共绘制$n-1$条线段	
Triangles	把每个顶点作为一个独立的三角形，顶点$3n-2$、$3n-1$和$3n$定义了第n个三角形，总共绘制$n/3$个三角形	
TriangleStrip	绘制一组相连的三角形，对于奇数n，顶点n、$n+1$和$n+2$定义了第n个三角形；对于偶数n，顶点$n+1$、n和$n+2$定义了第n个三角形，总共绘制$n-2$个三角形	

续表

参数名	含义	图示
TriangleFan	绘制一组相连的三角形，三角形由第一个顶点及其后给定的顶点确定，顶点1、n+1和n+2定义了第n个三角形，总共绘制$n-2$个三角形	
Quads	绘制由4个顶点组成的一组单独的四边形。顶点$4n-3$、$4n-2$、$4n-1$和$4n$定义了第n个四边形。总共绘制$n/4$个四边形	
QuadStrip	绘制一组相连的四边形。每个四边形是由一对顶点及其后给定的一对顶点共同确定的。顶点$2n-1$、$2n$、$2n$+2和$2n$+1定义了第n个四边形，总共绘制$n/2-1$个四边形	
Polygon	绘制一个凸多边形。顶点1到顶点n定义了这个多边形	

和GL.Begin函数相对应的函数GL.End()标志顶点列表的结束。从中可看出，对于同样的顶点，可以采用许多方法构造几何图元，这些方法仅仅依赖于所给的顶点数据。 下面是BeginMode各种设置的示例。

[例1] 一共输入6个点，根据显示模式的不同，OpenGL显示不同的结果。在同一个图中，其中不同的颜色表示不同的点、线、面元素。

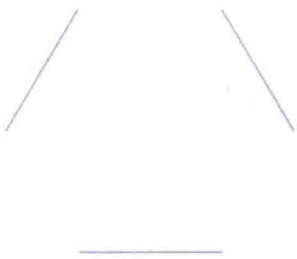
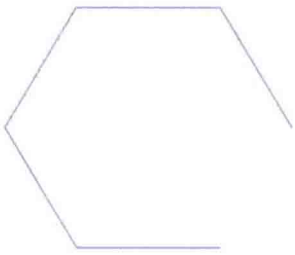
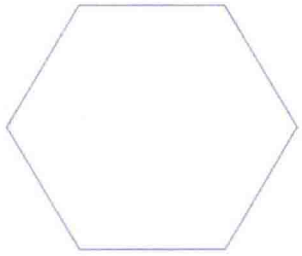
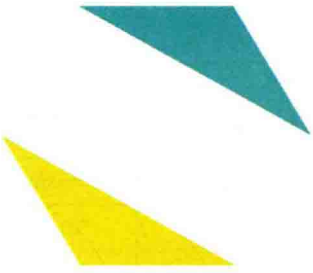
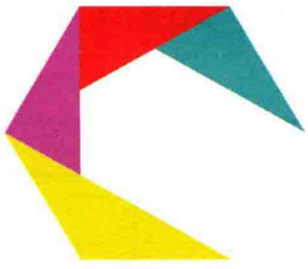
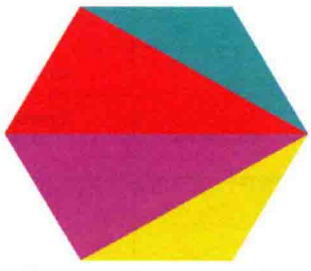
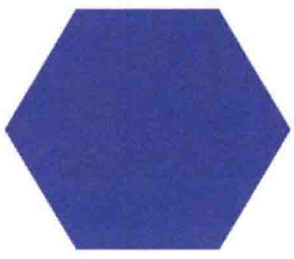
代码设置如下。

```
public void DemoOne()
    {
        GL.ShadeModel(this.shadingModel);
        GL.PolygonMode(MaterialFace.FrontAndBack, PolygonMode.Fill);
        GL.Enable(EnableCap.ColorMaterial);

        GL.Begin(this.beginMode);
        double r = 0.3;
        int n = 6;
        for (int i = 0; i < n; i++)
        {
            double x = r * Math.Cos(Math.PI * 2 / n * i);
            double y = r * Math.Sin(Math.PI * 2 / n * i);
            int color = i + 1;
            GL.Color3((float)(color / 4), (float)(color / 2 % 2), (float)(color % 2));
            GL.Vertex3(x, y, 0);
        }
        GL.End();
    }
```

显示结果见表2-2。

表2-2　例1显示结果

结果	图示	结果	图示
Points		Lines	
LineStrip		LineLoop	
Triangles		TriangleStrip	
TriangleFan		Quads	
QuadStrip		Polygon	

[例2] 一共6个点，根据参数的不同显示结果如表2-3所示。在同一个图中，不同的颜色表示不同的点、线、面元素。

代码设置如下。

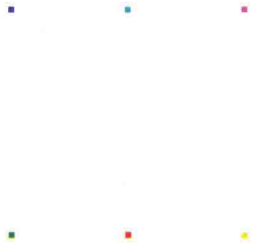
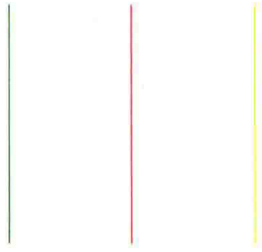
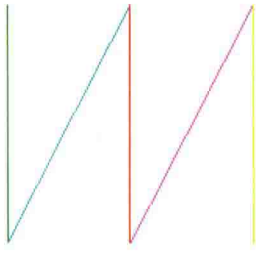
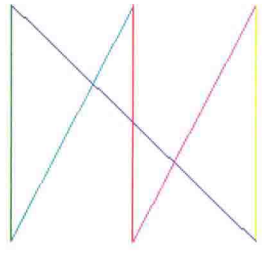
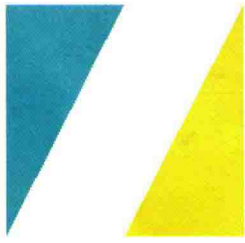
```
public void Demo1()
    {
```

```
            GL.ShadeModel(this.shadingModel);
            GL.PolygonMode(MaterialFace.FrontAndBack, PolygonMode.Fill);
            GL.Enable(EnableCap.ColorMaterial);

            GL.Begin(this.beginMode);
            double r = 0.3;
            GL.Color3(0f, 0f, 1f);
            GL.Vertex3(-r, r, 0);
            GL.Color3(0f, 1f, 0f);
            GL.Vertex3(-r, -r, 0);
            GL.Color3(0f, 1f, 1f);
            GL.Vertex3(0, r, 0);
            GL.Color3(1f, 0f, 0f);
            GL.Vertex3(0, -r, 0);
            GL.Color3(1f, 0f, 1f);
            GL.Vertex3(r, r, 0);
            GL.Color3(1f, 1f, 0f);
            GL.Vertex3(r, -r, 0);
            GL.End();
}
```

显示结果见表2-3。

表2-3 例2显示结果

结果	图示	结果	图示
Points		Lines	
LineStrip		LineLoop	
Triangles		TriangleStrip