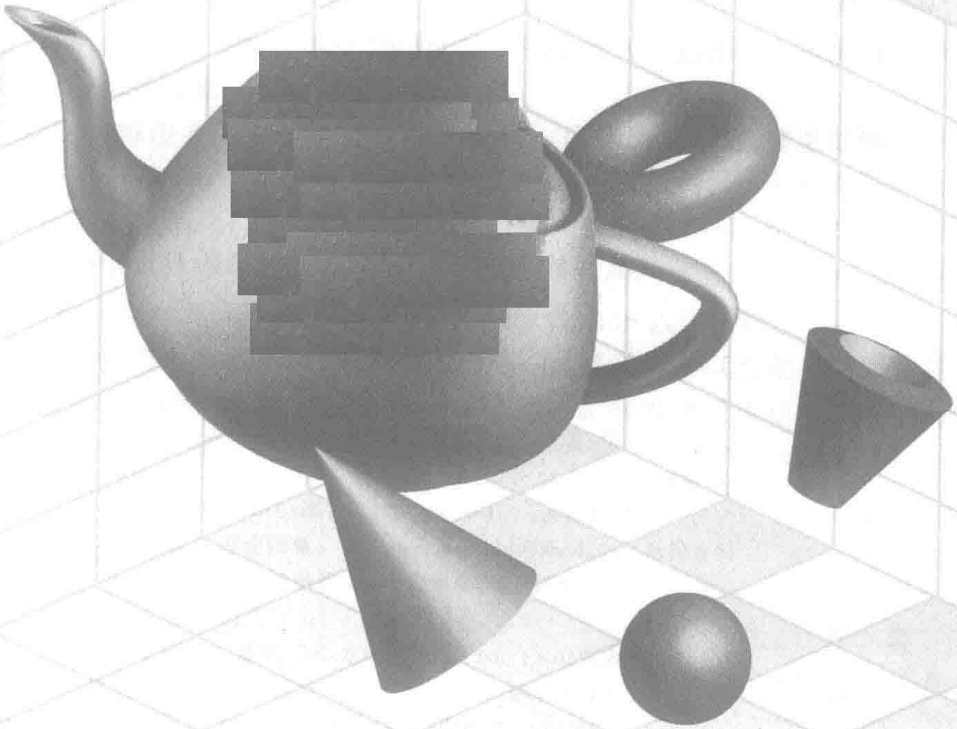
OpenGL简约笔记：用C#学面向对象的OpenGL

祝 威 编著

北京航空航天大学出版社
BEIHANG UNIVERSITY PRESS

OpenGL简约笔记：用C#学面向对象的OpenGL

祝 威 编著



北京航空航天大学出版社
BEIHANG UNIVERSITY PRESS

内 容 简 介

本书由浅入深地讲解 OpenGL 的概念和用法，通过一个个简单的实例使读者对各个知识点一目了然。书中包括渲染管道、着色器、缓存、纹理、矩阵、光照模型、阴影、帧缓存、拾取、文字、用户界面、体渲染、透明和通用计算等内容。读者掌握这些之后，就可以自由地设计和编写中等规模的三维图形程序，并且能够渲染百万数量级顶点的模型。本书相关内容将在 Github 上持续更新，读者可参阅更多资料。

本书适用于熟悉 C、C++、C# 或 Java 等任何面向对象编程语言的读者，本书会对这些相关的基础内容进行必要的介绍。读者只需认真实践，完全可以掌握本书内容。

图书在版编目(CIP)数据

OpenGL 简约笔记：用 C# 学面向对象的 OpenGL/祝威编著. -- 北京 ：北京航空航天大学出版社，2019.4

ISBN 978-7-5124-2735-8

Ⅰ. ①O… Ⅱ. ①祝… Ⅲ. ①图形软件 Ⅳ. ①TP391.412

中国版本图书馆 CIP 数据核字(2018)第 266706 号

OpenGL 简约笔记：用 C# 学面向对象的 OpenGL

编　　著　祝　威

责任编辑　剧艳婕

*

北京航空航天大学出版社出版发行

北京市海淀区学院路 37 号(邮编 100191)　http：//www.buaapress.com.cn

发行部电话：(010)82317024　传真：(010)82328026

读者信箱：emsbook@buaacm.com.cn　邮购电话：(010)82316936

涿州市新华有限公司印装　各地书店经销

*

开本：710×1 000　1/16　印张：20　字数：426 千字

2019 年 4 月第 1 版　2019 年 4 月第 1 次印刷　印数：3 000 册

ISBN 978-7-5124-2735-8　定价：69.00 元

前　言

本书将以C#为工具介绍OpenGL的相关知识。如果读者不太了解C#或面向对象程序设计，可参考本书最后的附录。

本书由浅入深地讲解OpenGL的概念和用法，通过一个个简单的实例让读者一目了然地认识OpenGL的各个知识点。本书包括渲染管道(Pipeline)、着色器(Shader)、缓存(Buffer)、纹理(Texture)、矩阵(Matrix)、光照模型(Lighting)、阴影(Shadow)、帧缓存(Framebuffer)、拾取(Picking)、文字(Text)、用户界面(UI)、体渲染(Volume Rendering)、透明(Transparency)和通用计算(Compute Shader)等内容。读者掌握这些内容后，就可以自由设计、编写中等规模的三维图形程序了，并且能够渲染百万数量级顶点的模型。

本书的网络资料包含了书中所有的完整代码，读者可以从北航出版社网站(www.buaapress.com.cn)的“下载专区”的相关页面获取。另外，读者也可以在CSharpGL托管页面(https://github.com/bitzhuwei/CSharpGL)找到本书的全部代码。如果读者对本书内容有任何疑问、指错或想进一步深入讨论，还可以在托管页面进行提问、讨论。本书最后的附录也提供了简单的Github入门教程。

感谢北京航空航天大学出版社提供的机会与平台，感谢家人以及业界的朋友在本书编写的过程中给予的支持和帮助，这里还要特别感谢编辑剧艳婕在本书出版的过程中提出的宝贵意见。希望本书能够帮助读者打开计算机图形学的大门。

作　者

2019年1月

目 录

第 1 章

Hello OpenGL

学完本章之后，读者将：

- 了解 OpenGL 是什么；
- 初次理解 OpenGL Pipeline 的工作流程；
- 使用 C# 编写简单的 OpenGL 程序。

1.1 从这里开始认识

大部分讲解 OpenGL 的书都是用 C\C++编写代码的，但本书全部使用 C# 编写 OpenGL 程序。这是因为，与 C\C++等语言相比，C# 需要的配置最少，运行环境最简单，调试环境最方便，最适合用来学习复杂的 OpenGL。

为了深入理解 OpenGL，尽可能发挥它的强大功能，本书使用了面向对象的编码方式，并涉及了一些数学和算法知识，因此本书要求读者具有普通程序员的编码技能。当然，本书会对相关的基础内容进行必要的介绍。读者只需认真实践，完全能够掌握本书内容。

学习一项技能，应该循序渐进、由浅入深。最初对 OpenGL 的认识，必然是不全面的，这很正常。因此，本书在对 OpenGL 最初的介绍中，会使用一些不全面甚至“不正确”的说法。这有利于读者逐步理解相关概念，降低学习难度。

无论是阅读本书还是代码，都应像剥洋葱一样，一层一层剥开，通读一遍，了解一些内容；再通读一遍，了解更多内容，直到彻底理解。

本书在中文段落中的代码一般会用黑框包围起来。但下列情况不会这样强调：

- 首次出现后又紧接着多次出现时；
- 要强调其他代码时。

1.2 OpenGL 是什么

用 OpenGL 可以编制多种三维图形程序（如 3D 游戏、工程仿真、石油勘探、影视

特效、科研教学等)。那 OpenGL 具体是什么呢?

从表面看,OpenGL 就是一系列函数声明(function declaration),合理地组织这些函数,就可以在某种画布上渲染出图形。例如,用 C 或 C# 语言描述 OpenGL 中用于清空画布的函数 glClear(..),如下:

```
// clear buffers to preset values
// mask: Bitwise OR of masks that indicate the buffers to be cleared
//The three masks are
//GL_COLOR_BUFFER_BIT, GL_DEPTH_BUFFER_BIT, and GL_STENCIL_BUFFER_BIT
void glClear(uint mask);
```

注意:OpenGL 只有函数的声明,而没有函数的定义(function definition)。也就是说,OpenGL 不负责实现这些函数(function implementation)。实现 OpenGL 给出的这些函数是显卡驱动的任务。当然,也可以自己编写一套 OpenGL 的实现,只是运行效率会比较低。

1.3 如何使用 OpenGL

1.3.1 准备工作

OpenGL 只有函数声明,显然不可能直接使用它。所谓的使用 OpenGL,意思是调用实现了 OpenGL 的显卡驱动程序。在能够正常地使用 OpenGL 绘制三维图形前,程序员需要做一些准备工作。

- 首先要找到 OpenGL 函数。

在 Windows 下,找到 OpenGL 函数很简单,做一个 extern 的函数声明即可。例如 C# 语言的 glClear(..)的声明如下:

```
[DllImport("opengl32.dll", SetLastError = true)] // C# Attribute
static extern void glClear(uint mask);
```

但如果是扩展的 OpenGL 函数(例如 `uint glCreateShader(uint shaderType);`),则要使用动态加载的方式,从系统文件 opengl32.dll 中找到函数指针。

- 接着,创建 Render Context。

没有这个 Context,任何对 OpenGL 函数的调用都是无效的。创建 Context 的步骤很繁琐,其中任何一点出错,就可能产生各种奇怪的问题,而且难以查找。

- 然后准备模型。

为了便于学习,可以只用一个三角形当模型,但在学习光照等环节时,没有恰当的模型很难体会到算法的精髓。

● 最后，要先领会很多孤立的函数和矩阵变换。

把模型数据正确地传送到显卡，并渲染出来，中间涉及几十个函数、各种神似的参数和复杂的绑定关系。其中任何一点出错，都很难调试，让人心态崩溃。矩阵也是OpenGL里很难理解的一个工具。为了实现各种方位变换、光照效果，矩阵变换也是绕不开的问题。

综上所述，晦涩繁琐、易错难改是初学OpenGL的最大困难。为了解决这些问题，本书以CSharpGL为基础，讲解如何开始使用OpenGL。其好处之一就是，上述准备工作都可以省了。

1.3.2 开源库

开源库(CSharpGL)是笔者设计编写维护的，它包含下述部分：

① CSharpGL：这是一个OpenGL库，封装了OpenGL的功能，编译出来的是一个库文件CSharpGL.dll；

② CSharpGL.Windows：封装了在Windows下使用OpenGL的准备工作。编译出来的是一个库文件CSharpGL.Windows.dll；

③ Infrastructures\：有一些辅助代码。既有必要出现，又不属于上述部分；

④ 其他：使用CSharpGL的示例集合。

有了CSharpGL，读者只需做如下准备工作：

● 下载安装Visual Studio 2013；

● 在本书网络资料中找到CSharpGL.zip，用Visual Studio 2013打开CSharpGL.sln解决方案；

● 完毕。读者可以直接运行各个示例项目，看看CSharpGL都能做些什么。

1.3.3 渲染流程

OpenGL应用程序开发者只需告知OpenGL想做些什么(渲染三维场景、执行并行计算等)，而不需知道OpenGL如何去完成这些任务。OpenGL会在一系列复杂的处理之后完成这些任务，这一系列复杂的处理过程就是OpenGL的渲染流程(Pipeline)。这类似《植物大战僵尸》这样的塔防游戏，玩家只需放置需要的植物(坚果墙、向日葵、豌豆炮、南瓜、磁力菇等)，它们就会自动发挥自己的作用，玩家不必关心豌豆是如何神奇地修炼成精并发出炮弹的。

这里来初次认识一下简化的Pipeline。OpenGL处理模型数据主要分3步，如图1-1所示。

1. 准备数据

三维世界里的物体，可以由组成物体的点以及点之间的连线(直线)来描述。例如图1-2就用点和连线描述了一个立方体和一个球体。

一个点最重要的数据就是其位置和颜色，该点就是OpenGL中的顶点(Ver-

图 1-1　简化的 OpenGL Pipeline

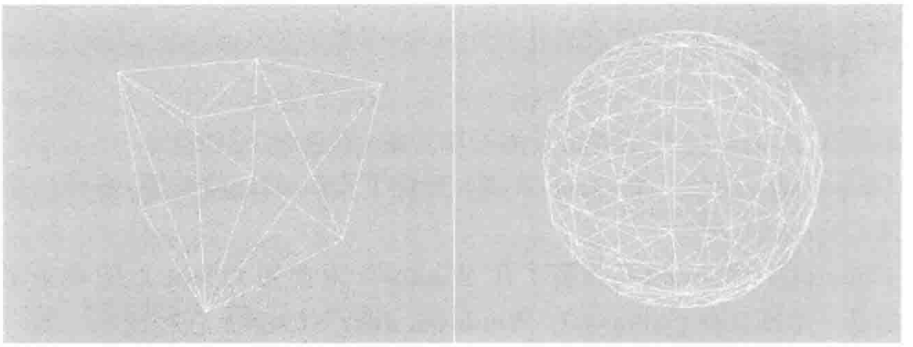


图 1-2　用点和连线描述的立方体和球体

tex)。用这种方式描述的物体称为一个模型(Model)。

OpenGL 处理的模型,是由一个个顶点组成的。对于每个顶点,都可能需要指定它的位置、颜色、法线和纹理坐标等属性(Attribute)。顶点的位置属性一般必须指定,而其他属性则未必。但是从程序的角度看,位置属性和其他属性是并列关系。

为了便于解释,这里以一个立方体 Cube 模型为例。立方体具有很特殊的性质,以后也将用它学习观察 OpenGL 的各种功能用法。CSharpGL 中用图 1-3 所示的注释来记录立方体的顶点顺序和坐标轴。这样的方式既直观,又不像图片那样占用大量的空间。

根据图 1-3,首先指定立方体 8 个顶点的位置属性:

```
private const float halfLength = 0.5f;
private static readonly vec3[] positions = new vec3[]
{
    new vec3( + halfLength,  + halfLength,  + halfLength),    // 0
    new vec3( + halfLength,  + halfLength,  - halfLength),    // 1
    new vec3( + halfLength,  - halfLength,  + halfLength),    // 2
    new vec3( + halfLength,  - halfLength,  - halfLength),    // 3
    new vec3( - halfLength,  + halfLength,  + halfLength),    // 4
    new vec3( - halfLength,  + halfLength,  - halfLength),    // 5
    new vec3( - halfLength,  - halfLength,  + halfLength),    // 6
    new vec3( - halfLength,  - halfLength,  - halfLength),    // 7
};
```

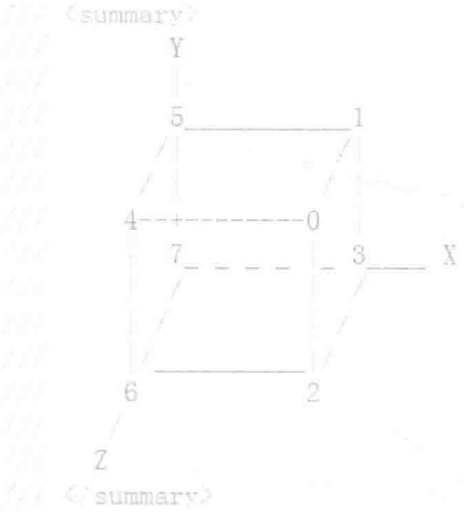


图 1-3　用字符画描述 Cube

这里只指定位置属性，其他属性暂不涉及。

有了位置，下面要指定这些顶点的连接关系。立方体有 6 个面，用 12 个三角形来描述这 6 个面，每个三角形需要指定 3 个顶点。用一个 uint[]类型的数组 indexes 指定 12 个三角形，代码如下：

```
private static readonly uint[] indexes = new uint[]
{
    0, 2, 1,  1, 2, 3, // +X faces.
    0, 1, 5,  0, 5, 4, // +Y faces.
    0, 4, 2,  2, 4, 6, // +Z faces.
    7, 6, 4,  7, 4, 5, // -X faces.
    7, 5, 3,  3, 5, 1, // -Z faces.
    7, 3, 2,  7, 2, 6, // -Y faces.
};
```

上述代码中，每 3 个数值描述一个三角形，每个数值指定此三角形的顶点在 positions 里的索引。例如，indexes[0]、indexes[1]、indexes[2]指定了第一个三角形，其顶点位置分别为 positions[indexes[0]]、positions[indexes[1]]和 positions[indexes[2]]。

2. 顶点着色器

着色器(Shader)是一段负责给图形上色(或为上色做准备)的程序，其形式类似于 C 语言。OpenGL 开始渲染后，首先会对每个顶点都执行一个简短的程序，这个程序就叫 Vertex Shader。

Vertex Shader 有 2 个用处：

- 指定顶点在 Clip Space 里的位置（即设置内置变量`vec4 gl_Position;`的值）。Clip Space 的相关内容将在矩阵章节具体介绍。现在只需知道，为立方体指定的顶点位置是在三维空间里的，而这最终要画到二维的窗口上，所以必然要进行坐标变换。坐标变换就是矩阵的功能。
- 向后续的 Fragment Shader 传递数据。为了实现各种光照效果，向 Fragment Shader 传递顶点的位置、颜色、纹理坐标等属性。

3. 线性插值

立方体只有 8 个顶点，然而 OpenGL 却要画出成片的三角形，这是通过 OpenGL 内部的插值功能实现的。在 Vertex Shader 完成后，OpenGL 会根据`indexes`生成所有的三角形，然后根据三角形的三个顶点的位置、颜色等属性，通过插值的方式，为三角形覆盖到的各个像素生成其位置、颜色等属性值。上文说明了 Vertex Shader 可以向 Fragment Shader 传递数据。这些数据，也将作为顶点的属性，一并进行插值。

具体的插值过程完全由 OpenGL 自动完成，OpenGL 应用程序不需也无法干预。

4. Fragment Shader

经过插值处理，画布上的一个像素所在的位置，每被三角形覆盖一次，就会产生一个富含顶点属性的片段（Fragment）。Fragment Shader 会对每个 Fragment 执行一次，其输出结果就是它所在像素的候选颜色值。

在 Fragment Shader 中，一个像素的颜色是由四维向量`vec4(r, g, b, a);`描述的。其中 r、g、b、a 的范围都是[0, 1]。如果 Shader 输出的值超出此范围，OpenGL 会自动将其裁切到范围内。

1.3.4 小 结

这是首次认识 Pipeline，因此省略了很多环节。真正完整的 Pipeline 要复杂得多，本书将逐步展开予以介绍。

Pipeline 是 OpenGL 的核心，是一个高度灵活的算法（可以自定义 Shader、控制开关）。充分利用 Pipeline，就可以实现光照、阴影、拾取、透明等各种效果和功能。

1.4 Hello OpenGL

本节一步步地介绍如何完成第一个 OpenGL 程序，画出一个立方体。读者可在本书网络资料上的 c01d00_Cube 项目中找到此示例的完整代码。建议读者先打开此项目，以逐步执行的方式调试运行几次，对项目代码建立一些感性的了解，再详细

阅读下面的章节。请注意随时记得 Pipeline。OpenGL 所做的一切都是围绕 Pipeline 进行的。

Pipeline 需要三类信息：模型、Shader 和状态开关。最后调用 OpenGL 的渲染命令即可。状态开关用于控制 OpenGL 的各种状态，例如控制多边形渲染模式的 glPolygonMode(..)，控制线宽度的 glLineWidth(..) 等。

1.4.1 新建项目

首先，在本书网络资料中找到 CSharpGL.zip 进行解压缩，用 Visual Studio 打开 CSharpGL.sln 解决方案。将解决方案中的 Demos 文件夹和 OpenGLviaCSharp 文件夹下的所有项目都删除，以便从零开始创建新项目。然后，新建一个项目，如图 1-4 所示。

图 1-4 新建项目 c01d00_Cube

新建一个“Windows 窗体应用程序”，项目命名为“c01d00_Cube”。然后，为该项目引用必要的 DLL 库项目，如图 1-5 所示。

只需引用 CSharpGL.dll 和 CSharpGL.Windows.dll。CSharpGL.dll 封装了 OpenGL 函数，类似于在 C++中引用了 gl.h 头文件。CSharpGL.Windows.dll 封装了在 Windows 下使用 OpenGL 的初始化工作，类似于在 C++中引用了 glut.h 头文件。

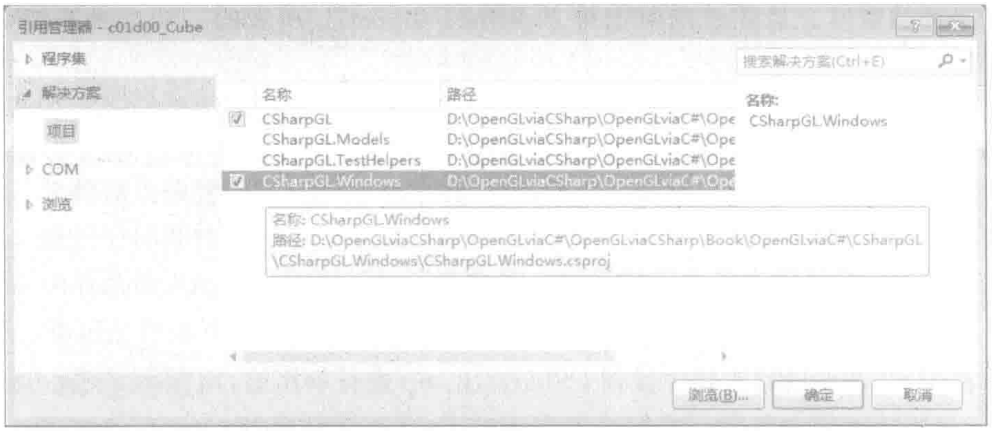


图 1-5　添加引用

1.4.2　使用 WinGLCanvas 控件

项目创建完成后,会自动提供一个 Form1 窗口。在这个窗口上添加一个 WinGLCanvas 控件。WinGLCanvas 控件有如下功能:

① OpenGL 渲染的所有内容都将显示在此控件范围内;

② 封装了 OpenGL 的初始化过程;

③ 提供渲染事件和两种渲染触发方式(自动或手动)。

用 Visual Studio 编译刚刚创建的项目,打开 Form1 窗口,就可以在工具箱里找到 WinGLCanvas 控件,如图 1-6 所示。

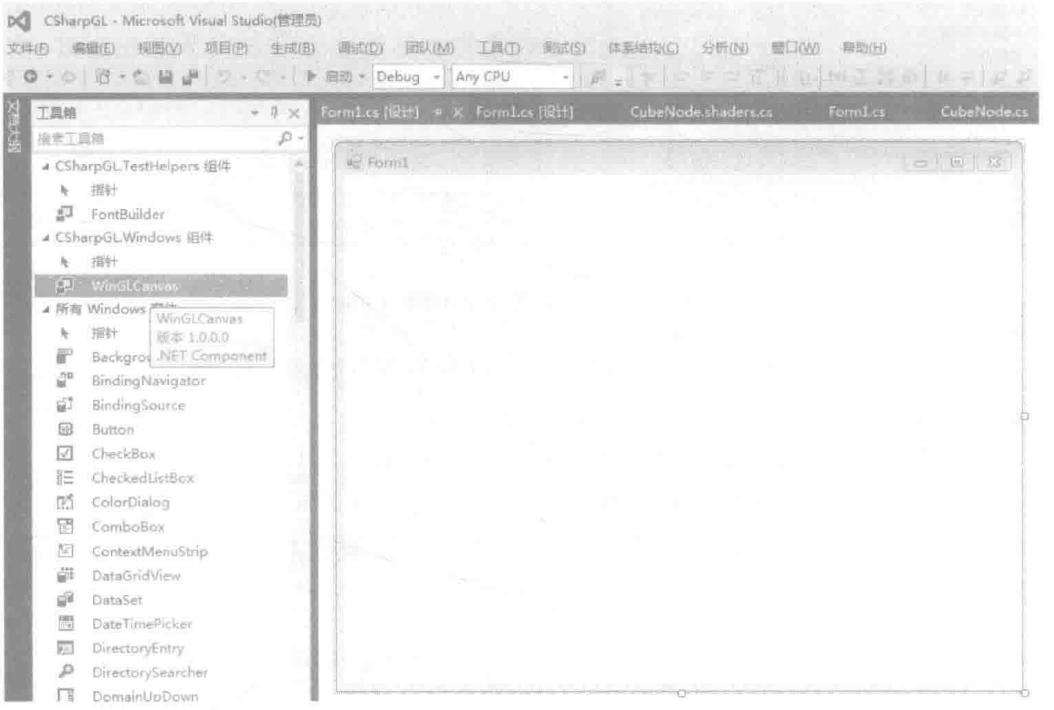


图 1-6　编译后在工具箱找到 WinGLCanvas 控件

从工具箱中拖拽一个 WinGLCanvas 控件到窗口 Form1 上，设置其 Dock 属性为 Fill，保存、编译，此时 Form1 里的控件是纯黑色的，如图 1-7 所示。

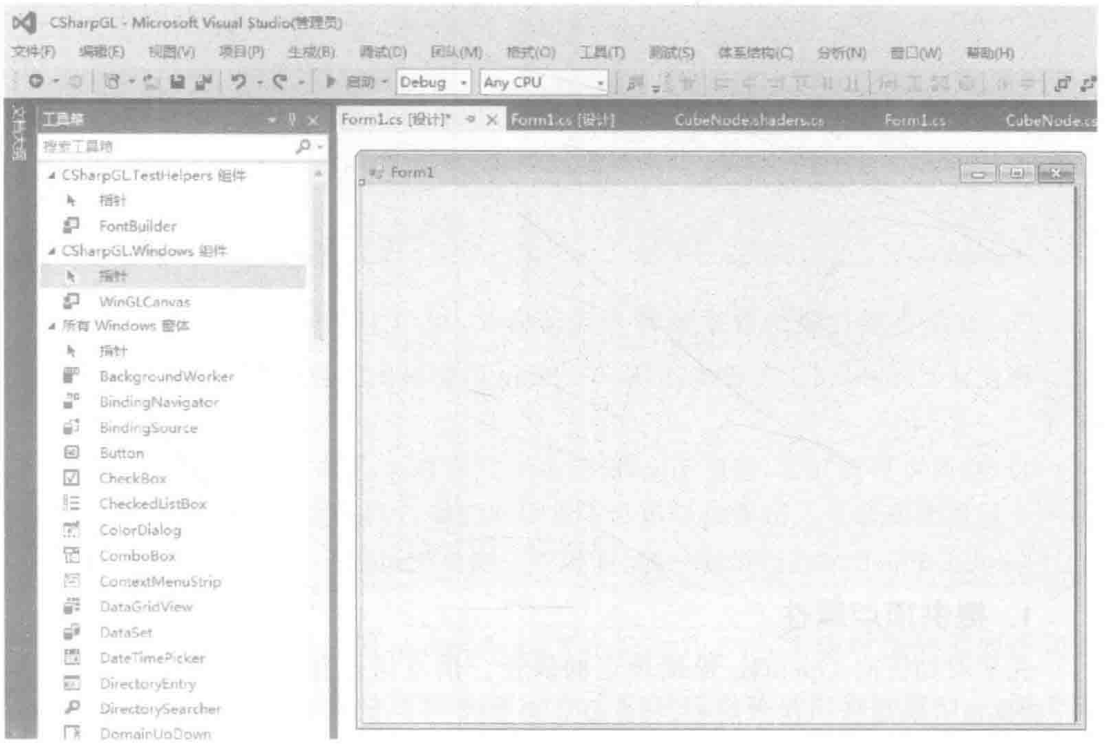


图 1-7　添加 WinGLCanvas 控件到窗口

1.4.3　模型数据 IBufferSource

项目、控件已经准备好，新建代码文件 CubeModel.cs，用以向 OpenGL 提供模型数据，代码如下：

```
namespace c01d00_Cube
{
    class CubeModel :IBufferSource
    {
        #region IBufferSource 成员

        public IEnumerable<VertexBuffer> GetVertexAttribute(string bufferName)
        {
            throw new NotImplementedException();
```