Broadview
www.broadview.com.cn

以C++Builder 6.0为开发平台讲解OpenGL程序开发技巧
全方位展示笔者多年OpenGL程序开发经验
帮助读者理清学习思路，全面掌握OpenGL编程精髓

C++ Builder 6.0 下

OpenGL 编程技术

蒋勇 王介付 刘敬 著

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

C++ Builder 6.0 下

OpenGL 编程技术

蒋勇 王介付 刘敬 著

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内容简介

本书以 C++Builder 6.0 为开发平台，详细、系统地介绍了 OpenGL 程序研发入门知识。主要包括组件包安装、glut 库转化、基本图元、光照、颜色、键盘与鼠标操作等内容，最后提供了一个非常详细的应用程序开发模板供读者参考。本书是以 C++Builder 6.0 为开发平台进行 OpenGL 程序研发的工具书，很多章节将作者研发过程中遇到问题、寻求解决方法、多角度尝试、多次失败、最终解决问题的全过程在书中一一呈现，从这个意义上来讲，本书是作者最原始的研发笔记。作者希望本书不仅能为读者提供 OpenGL 技术指导，更能帮助读者理清研发思路、树立克难攻坚的信心与决心。

本书面向的读者主要有对计算机图形学比较感兴趣的高年级本科生、从事图形图像研究的研究生、对 OpenGL 3D 游戏开发比较感兴趣的编程爱好者，以及编写图形硬件驱动的程序开发人员，本书也适合于从事 3D 游戏开发的工作人员参考借鉴。

图书在版编目（CIP）数据

C++Builder 6.0 下 OpenGL 编程技术/蒋勇，王介付，刘敬著. —北京：电子工业出版社，2011.3
ISBN 978-7-121-12867-7

Ⅰ. ①C… Ⅱ. ①蒋… ②王… ③刘… Ⅲ. ①图形软件，OpenGL－程序设计②C 语言－程序设计
Ⅳ.①TP391.41②TP312

中国版本图书馆 CIP 数据核字（2011）第 015075 号

策划编辑：
责任编辑：李　冰
特约编辑：赵树刚
印　　刷：
装　　订：北京中新伟业印刷有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　邮编 100036
开　　本：787×980　1/16　印张：13.25　字数：297 千字
印　　次：2011 年 3 月第 1 次印刷
印　　数：4 000 册　　定价：39.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：（010）88258888。

前　言

C++ Builder 是由 Borland 公司继 Delphi 之后推出的一款高性能、可视化集成开发工具，C++ Builder 具有快速的可视化开发环境，它实现了可视化的编程环境和功能强大的编程语言（C++）的完美结合。OpenGL 是一个行业标准的跨平台应用程序编程接口（Application Programming Interface，简称 API）。OpenGL 与硬件无关，利用它所开发的程序可以在不同的平台之间进行移植。近年来，OpenGL 在 3D 游戏开发等领域应用广泛。作者结合几年来在人体头颅建模、纺织产品设计等方面科学研究的经验，将 C++ Builder 与 OpenGL 研发技术强强结合，精心编写本书，希望广大读者通过阅读本书来解决学习、工作、科研中遇到问题。

本书具体内容编排如下：

第 1 章详细介绍 TOpenGLB6 组件包的安装、glut 库的转换方法。该章是整本书的铺垫，从无到有的过程中，作者遇到了很多意想不到的问题，利用有限资源探寻解决问题办法的过程对读者有较高的借鉴意义。

第 2 章详细介绍 OpenGL 的基本图元，如点、线、三角形、四边形、多边形等。本章重点讲解了图形的基本元素，也是图形相关研发的基础。本章利用浅显的图形知识、普通常见的函数创造性地开发了几个有趣地实例，作者重点体会其中的创造性、发散性思维方法。

第 3 章详细介绍颜色、光照、纹理、位图的相关知识。本章内容多、信息量大，利用好颜色、光照、纹理、位图等知识是 OpenGL 程序研发的关键环节。建议读者在详细掌握本章的基础上，通过修改部分程序代码、修改部分数据等方法体会其中细微差别，以便寻求最佳显示效果。

第 4 章详细介绍动态效果，主要内容有平移、旋转、缩放等。图形的平移、旋转、缩放等是图形研发的基础性工作，是图形研发中模块化、程序化的过程，建议读者在学会合理利用平移、旋转、缩放等技术手段的基础上粗读即可。

第 5 章介绍高级技巧与综合实例，主要是对前 4 章的内容进行提高与加强。本章在整书中具有承上启下的作用，自由落体运动等小程序开发中细节的处理请读者多加体会。

第 6 章详细介绍如何从 VC++的 OpenGL 程序中提取有用信息应用到 C++ Builder 6.0 中。因国内现有 OpenGL 程序研发类书籍基本上都是以 VC++为研发平台，如何选择有用信息并加以运用对读者以后进行系统研发具有重要的意义，对程序跨平台移植也有一定的借鉴意义。

第 7 章详细介绍一个应用程序开发模板。此模板将 OpenGL 程序研发的基本元素加以整合，以此模板为基础，可以使研发人员从基础性、铺垫性的工作中解放出来，专心处理核心代码，对提高研发效率意义重大。当然，读者也可以以此模板为基础，研发出适合本部门、本领域的专业性模板。

入门的知识、核心的思想，这是作者编写本书的出发点与期望之所在。

南京信息工程大学　蒋勇

目　录

第1章 架好通向 OpenGL 的桥

本章主要是为以后进行的 OpenGL 编程进行一些铺垫工作。主要内容有：OpenGL 组件包的安装、如何从 VC++ 6.0 下的 OpenGL 程序中提取有用的信息、glut 库的设置。

1.1 由 C++Builder 6.0 通向 OpenGL

事实上，大家所用的操作系统 Windows XP、Windows 2000 与开发环境 C++Builder 6.0 已经将这座桥架好。我们现在所要做的工作就是将桥面铺平，以便我们走得更舒服。我们这里所有的设置是以 Windows XP 操作系统为例进行说明。

简单介绍一个组件：TOpenGLB 6。这个组件是由 Alan 开发的，用于 OpenGL 编程时进行背景设置。现在可以免费使用这个组件（这里有个非常有趣的故事，大家可以阅读本章后面的内容）。读者可以从 http://www.hellix.com/Alan/Computing/免费下载。

看过 OpenGL 方面书的读者都知道，所有的 OpenGL 程序都要设置一个背景，每次都要写：

```
#include<gl/gl.h>
#include<gl/glu.h>
```

还要进行屏幕的颜色、大小、长宽比例、在窗口中的位置等方面的设置。而且在 VC++ 中这些设置都需要特定的函数来实现。好了，现在我们可以把这项工作交给 TOpenGLB 6 组件去做，实现上面提到的效果只需用鼠标拖曳一下即可。

当然读者也可以使用其他功能更强大的组件包，但在网上找了一下并没有找到适用于

C++Builder 6.0 的 OpenGL 方面的组件（用于 C++Builder 4.0、C++Builder 5.0 倒有不少）。当然，如果读者对 C++Builder 6.0 和 OpenGL 都比较了解的话，可以自己开发最适合自己用的组件。

下面用 TOpenGLB 6 组件包来铺设 “桥面”。

考虑到一般人会将应用程序安装到 D 盘，以下均假定 C++Builder 6.0 的安装目录为 D:\Program Files\Borland\CBuilder 6。首先，将 TOpenGLB 6（里面包含了 OpenGL 组件包的所有文件）文件夹放到 CBuilder 6 文件夹下。工作正式开始：

❶ 打开 C++Builder 6.0。在菜单栏选择“组件”→“安装包”命令，如图 1-1 所示。

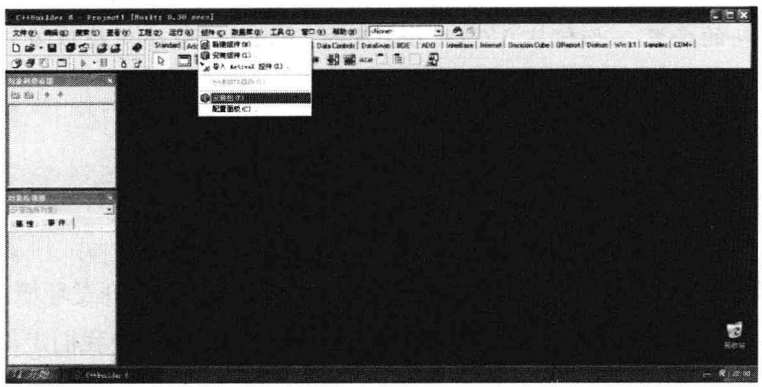

图 1-1　组件安装包略图

❷ 单击后，将看到如图 1-2 所示的界面，单击“添加”按钮。

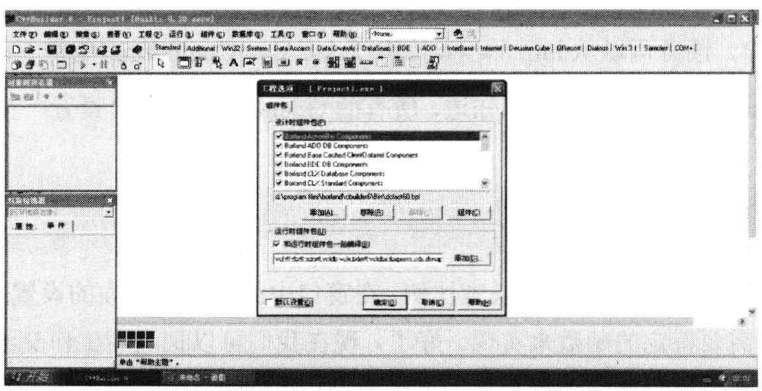

图 1-2　安装过程示意图

❸ 如图 1-3 所示，这里的对话框让你来选择组件包文件。打开文件夹 TOpenGLB 6，选择 OpenGLPackage.bpl 文件，单击“打开”按钮就可以了。

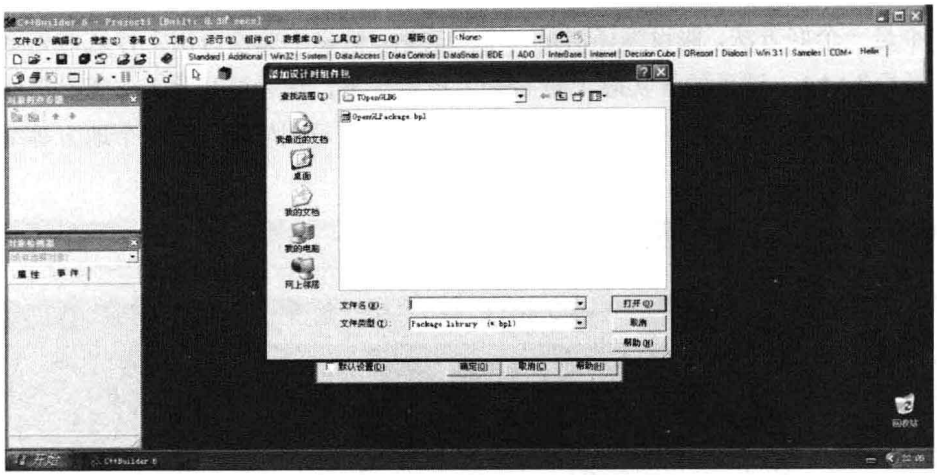

图 1-3 选择安装包

然后会出现如图 1-4 所示的界面。大家观察一下我们前面所做工作的效果。在 C++Builder 6.0 组件面板中，最后的位置多出了一个 Helix 标签。单击一下，你会发现它上面出现了一个 OpenGL 组件。而它就是我们所需要的，以后所有的程序就是用它来设置屏幕背景的。

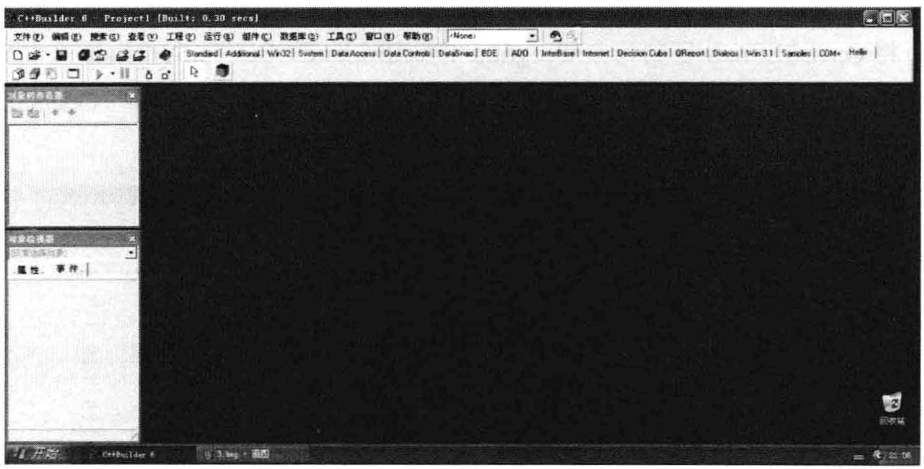

图 1-4 Helix 标签添加效果

现在来试一下这个组件好不好用。单击 图标，然后在 Form1 中单击，Form1 中就会出现一个黑色的矩形。用鼠标拖曳适当地调整矩形的大小（见图 1-5），这就是以后要用的 OpenGL 作图的背景了。调试一下，不能运行，为什么呢？这是因为编译器找不到 OpenGLSrc.h 这样一个文件。可能有人会想到修改路径，不错，这样是可以的。但我们认

为这并不是一个好方法，假设编译器是一个人的话，他愿不愿意每次工作之前先跑两个地方向“领导”（*.h，这里的 h 实际上是 head）报告一下，然后再开始工作呢？下面我们要做的工作是为编译器提供一些方便，说白了就是让“领导们”都呆在一个地方等编译器来报告。但问题是让“领导们”待在哪里呢？

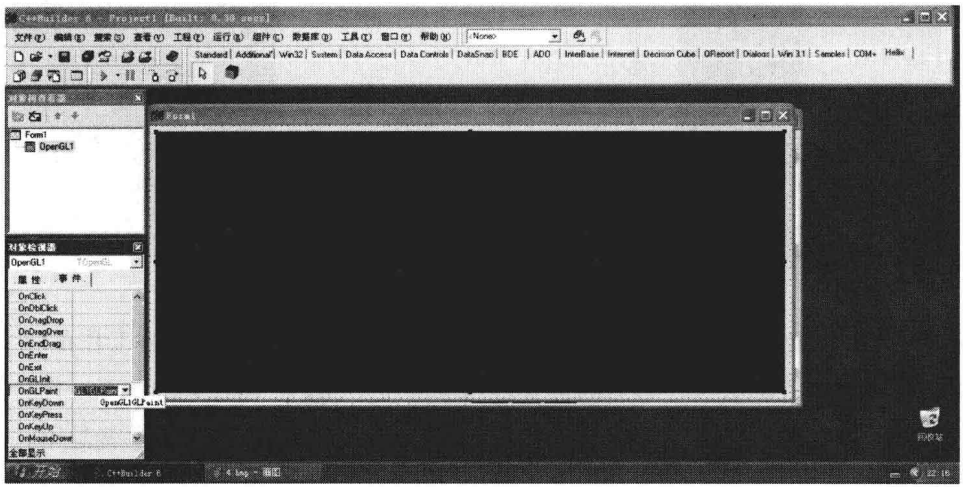

图 1-5　Form1 设置效果

❹ 将 OpenGLSrc.h 复制到 D:\Program Files\Borland\CBuilder6\Include 文件夹下，如图 1-6 所示。另外别的“领导”（*.h）都在这里“办公”，估计“新领导”OpenGLSrc.h 也会喜欢待在这里。

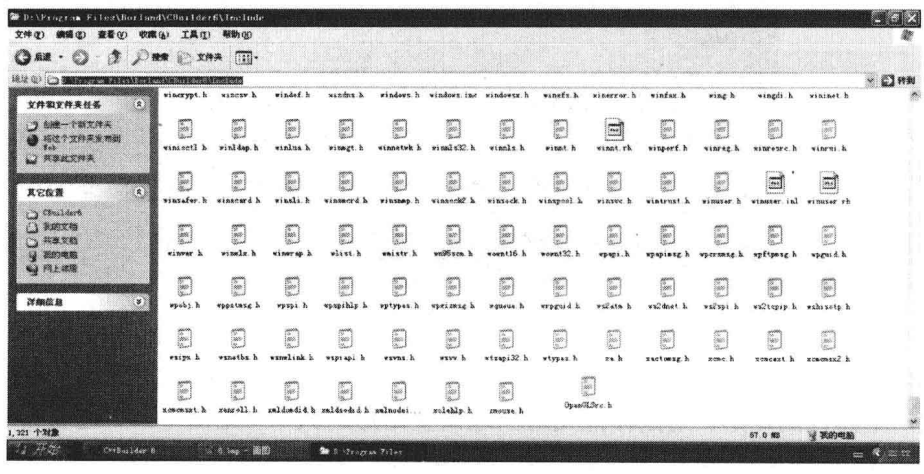

图 1-6　head“办公室”

另外，基于为编译器提供方便的理念，还需要让“秘书们”换一下地方。原则：到它们应该到的地方（什么意思呢？意思就是说让新秘书到秘书办公室工作）。

⑤ 将 OpenGLPackage.bpl 文件复制到 D:\Program Files\Borland\CBuilder6\Bin 文件夹下，如图 1-7、图 1-8 所示。同样的方法，将 OpenGLPackage.lib、OpenGLPackage.bpi 文件复制到 D:\Program Files\Borland\CBuilder6\Lib 文件夹下（注意：这个文件夹好像是*.lib 文件的“办公室”啊！原来大家都在这里！）。

图 1-7　复制

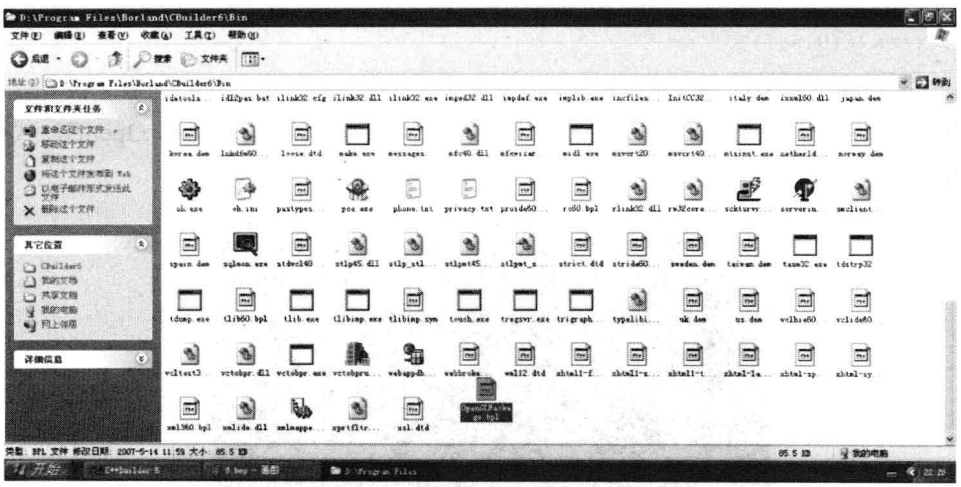

图 1-8　粘贴

好了，所有的设置都完成了。现在可以试一下我们的工作成果了。回到如图 1-5 所示的

界面，先单击一下 Form1 中的黑色区域，然后在左下角的“对象检视器”中，选择“事件”选项卡，找到“OnGLPaint”选项，在竖线的右侧双击，则会出现如图 1-9 所示的画面。

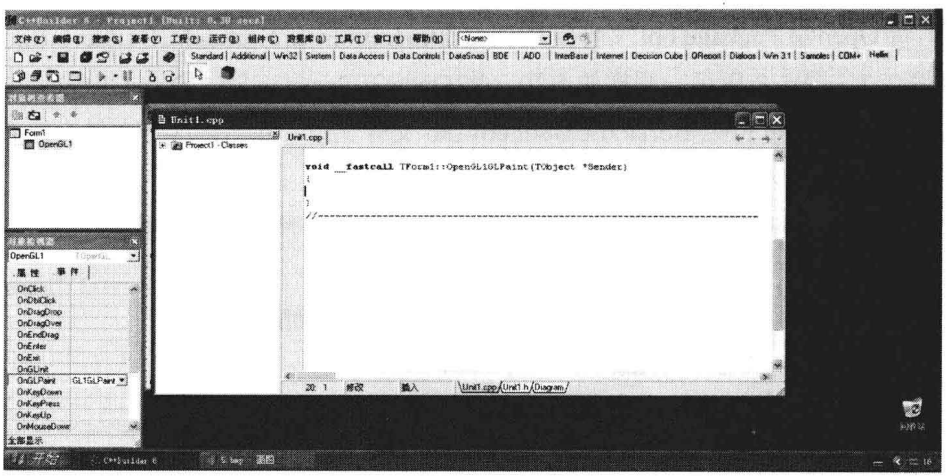

图 1-9 “对象检视器”设置

添加以下代码：

```
glLineWidth(20);
glColor3f(1,0,0);
glBegin(GL_LINE_STRIP);
glVertex2f(-0.8,1);
glVertex2f(0,-1);
glVertex2f(0.8,1);
glEnd();
```

调试运行，将出现如图 1-10 所示的画面。在这里我们用最简单的 OpenGL 代码，为第一阶段的工作写下一个大大的“V”字。

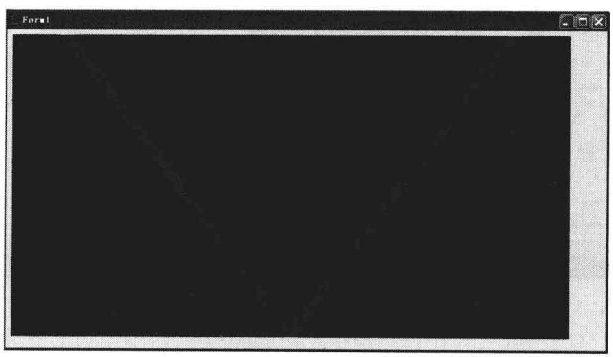

图 1-10 绘图效果

本节内容主是涉及 TOpenGLB 6 组件包的安装。之所以如此详细介绍，一方面是因为这是后面所有内容的基础，若此处设置不成功，后面的所有内容就没有任何价值；另一方面，熟悉了本节内容，有利于大家来安装其他 C++Builder 6.0 的组件。

1.2 由 VC++ 6.0 通向 C++Builder 6.0

本节为了比较 VC++ 6.0 下 OpenGL 编程同 C++Builder 6.0 下 OpenGL 编程的异同，将完整地展示几段 VC++ 6.0 下 OpenGL 的程序（这个例子来自互联网。原始出处不可查找，若无意间侵犯了作者的权利请原谅）。为了表示对原作者的尊重，我们这里用斜体进行标记。学习本节内容，要学会如何从 VC++ 6.0 下 OpenGL 的例子中寻找我们所需要的东西。

现在 VC++ 6.0 环境下进行 OpenGL 编程的书很多，差不多每一本书的第一、二个程序都是用来实现简单的 OpenGL 编程的。阅读过这方面书籍的人会对下面这个程序非常熟悉：

```
#include <Windows.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <gl/glaux.h>
#include <stdio.h>
#include <io.h>

HWND      hWnd;
HDC           hDC;
HGLRC     hRC=NULL;          //定义渲染环境
HINSTANCE hInstance;         //得到程序的例子

RECT      rect;

INT           SW = 640;
INT           SH = 480;

BOOL      FULLSCREEN  = 1;

GLFLOAT       ASPECT;

#PRAGMA COMMENT( LIB, "OPENGL32.LIB" )                    //链接时使用 OPENGL32.LIB
```

```
#pragma comment( lib, "glu32.lib" )                    / 链接时使用 GLu32.lib
#pragma comment( lib, "glaux.lib" )                    //链接时使用 GLaux.lib

void SceneInit(int w, int h)
{
 glShadeModel(GL_SMOOTH);                              //允许平滑着色
 glClearColor( 0.0, 0.0, 0.0, 0.5 );
 glClearDepth(1.0f);                                   //设置深度缓冲区
 glEnable(GL_DEPTH_TEST);
 glDepthFunc(GL_LEQUAL);                               //深度测试的类型
 glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);
}

void SceneResizeViewport(GLsizei w, GLsizei h)
{
 if(h==0)
 {
     h=1;
 }
 aspect = (GLfloat)w/(GLfloat)h;

 glViewport(0, 0, w, h);
 glMatrixMode(GL_PROJECTION);                          //选择投影矩阵
 glLoadIdentity();

 gluPerspective( 45.0f, aspect, 0.1f, 100.0f );
 glMatrixMode(GL_MODELVIEW);
 glLoadIdentity();
}

void SceneShow(GLvoid)                                 //这里进行所有的绘图工作
{
 glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);   //清屏和清除深度缓冲区
 glLoadIdentity();                                     //重置当前 Modelview 矩阵
}

void EnableOpenGL()
{
 PIXELFORMATDESCRIPTOR pfd;
 int iFormat;
```

```
    hDC = GetDC( hWnd );

    ZeroMemory( &pfd, sizeof( pfd ) );
    pfd.nSize = sizeof( pfd );
    pfd.nVersion = 1;
    pfd.dwFlags =  PFD_DRAW_TO_WINDOW |
                   PFD_SUPPORT_OPENGL |
                   PFD_DOUBLEBUFFER;
    pfd.iPixelType = PFD_TYPE_RGBA;
    pfd.cColorBits = 16;
    pfd.cDepthBits = 16;
    pfd.iLayerType = PFD_MAIN_PLANE;

    iFormat = ChoosePixelFormat( hDC, &pfd );

    SetPixelFormat( hDC, iFormat, &pfd );

    hRC = wglCreateContext( hDC );
    wglMakeCurrent( hDC, hRC );
}


// 取消 OpenGL
void DisableOpenGL()
{
    wglMakeCurrent( NULL, NULL );
    wglDeleteContext( hRC );
    ReleaseDC( hWnd, hDC );
}

bool ChangeResolution(int w, int h, int bitdepth)
{
    DEVMODE devMode;
    int   modeSwitch;
    int   closeMode = 0;

    EnumDisplaySettings(NULL, closeMode, &devMode);

    devMode.dmBitsPerPel = bitdepth;
    devMode.dmPelsWidth  = w;
    devMode.dmPelsHeight = h;
```

```
    devMode.dmFields = DM_BITSPERPEL | DM_PELSWIDTH | DM_PELSHEIGHT;

    modeSwitch = ChangeDisplaySettings(&devMode, CDS_FULLSCREEN);

    if(modeSwitch == DISP_CHANGE_SUCCESSFUL)
    {
        return true;
    }
    else
    {
        ChangeDisplaySettings(NULL, 0);
        return false;
    }
}

LRESULT CALLBACK WndProc( HWND hWnd, UINT message,
                          WPARAM wParam, LPARAM lParam )
{

    switch ( message )
    {
        case WM_CREATE:
            GetWindowRect(hWnd, &rect);
            sw = rect.right - rect.left;
            sh = rect.bottom - rect.top;
            SceneResizeViewport(sw, sh);
            return 0;

        case WM_SIZE:

            if(!fullscreen)
            {
                GetWindowRect(hWnd, &rect);
                sw = rect.right - rect.left;
                sh = rect.bottom - rect.top;
                if(sw>0 && sh>0)
                    SceneResizeViewport(sw, sh);
            }
            else
            {
                SceneResizeViewport(GetSystemMetrics( SM_CXSCREEN ),
```

```
                                        GetSystemMetrics( SM_CYSCREEN ));
            }

            return 0;

        case WM_CLOSE:
            ShowWindow (hWnd, SW_HIDE);
            PostQuitMessage( 0 );
            return 0;

        case WM_DESTROY:
            return 0;

        case WM_KEYDOWN:
            switch( wParam )
            {
            case VK_ESCAPE:
                PostMessage(hWnd, WM_CLOSE, 0, 0);
            break;
            }
            return 0;
        default:
        return DefWindowProc( hWnd,message, wParam, lParam );
    }
    }


    int APIENTRY WinMain(HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPSTR     lpCmdLine,
                     int       nCmdShow)
    {
        WNDCLASS wc;
    MSG msg;
    bool bQuit = false;

    if (MessageBox(NULL,"是否选择全屏显示模式?", "全屏方式运行?",MB_YESNO|MB_ICONQU
ESTION)==IDNO)
    {
        fullscreen=0;                           //窗口模式
    }
```