OpenGL 三维图形库 编程指南

白燕斌 史惠康 等编

机械工业出版社

前　言

图形制作和显示一直是计算机技术的一大课题。图形尤其是三维图形在军事、航天、航空、医学、广告、仿真、虚拟现实等领域的应用越来越广泛。以前，三维图形只是工作站的专利，工作站高昂的价格，只能使微机用户望而却步。随着计算机技术的发展，微机性能的提高，使得在工作站上实现的图形功能移植到微机平台上成为历史的必然。

OpenGL 就是在这种前提下移植到微机平台的。OpenGL 是一种硬件和图形的软件接口，实际上就是一个三维图形和模型库。由于它在三维真实感图形制作中的优秀性能，所以，诸如 Microsoft 公司、SGI 公司、、IBM 公司、DEC 公司、SUN 公司、HP 公司等在计算机市场中占主导地位的大公司都将其作为自己的图形标准，从而使 OpenGL 成为新一代的三维图形工业标准。

此外，OpenGL 自身也是一个 API(Application Programming Interface)，它是一个与硬件无关的编程接口，而且可以在不同的硬件平台上方便地移植。

本书主要介绍基于 Windows 95、Windows NT 操作系统的 OpenGL 程序的编写。其中所有例程都是使用 VC++ 4.2 编写的。在本书第二章到第八章中主要介绍了 OpenGL 的基本功能，为了学习方便，这几章的所有例程都是使用 C 调用 OpenGL 函数实现的。在第九章中，详细地介绍了使用 MFC 编写 OpenGL 程序。本书选取的例子都具有一定的代表性，希望读者能亲自使用它们，相信会有很大的收获。

本书由齐侪创作室策划，第一章由钟建平编写，第二章到第六章由白燕斌编写，第七章由史惠康编写，第八章由韩友斌编写，第九章由张磊编写，第十章由贾志东编写，另外参加编写的还有张杰、鲁晓红、刘小明、何国梁、范刚、彭有为等，全书由周予滨和史惠康统稿。

由于编者水平有限，书中难免有误，希望读者批评指正。

编者

1998 年 8 月

目　录

第一章　OpenGL 简介

1.1　OpenGL 概述

OpenGL 是个硬件和图形的软件接口，实际上就是一个三维图形和模型库。由于它在三维真实感图形制作中具有优秀的性能，所以，诸如 Microsoft、SGI、IBM、DEC、SUN，HP 等在计算机市场中占主导地位的大公司，都将它做为自己的图形标准，从而使之成为新一代的三维图形工业标准。

OpenGL 不仅仅是一个图形库，更是一个 API(Application Programming Interface)。它本身是一个与硬件无关的编程接口，可以在不同的硬件平台上得到实现，也正是因为如此，OpenGL 中没有包含处理窗口和用户输入的命令。在 OpenGL 中不提供三维造型的高级命令，虽然 OpenGL 也是通过基本的几何图元——点、线、多边形来建立物体模型的，但更确切地说它应该被称为新一代的三维图形开发标准。现在有很多优秀的三维图形软件，如 3D max 等可以方便地建立模型，但难以对其进行控制。把这些模型转化为用 C 语言编写的 OpenGL 程序，就可以随心所欲地控制这些模型、制作 CAD、制作三维动画、实现虚拟仿真、制作商业广告、进行影视采辑，这些都使我们制作出的三维真实感图形更方便、更真实。

OpenGL 可以制作各种各样的三维图形，方便地实现三维图形的交互操作。但这些图形都是由一些基本操作实现的。OpenGL 提供的操作包括：

一、绘制物体(Drawing Object)

任何三维图形、三维场景都是由一些基本的图元——点、线、多边形组成的。实际上，一个图形系统，其性能是由它对这些基本图元的绘制操作决定的。OpenGL 提供了丰富的基本图元绘制命令，在第二章中将详细地介绍 OpenGL 的基本图元绘制函数。

二、变换(Transformer)

任何复杂的图形都是物体经过一系列变换来实现的。OpenGL 提供了一系列基本的变换，如投影变换定义了一个视景体；几何变换可以使物体在三维场景中平移、旋转和放缩；视点变换可以从不同的角度去观察物体，如果用特殊的显示硬件还可实现虚拟现实显示；裁剪变换可以定义除了裁剪体以外的裁剪平面；视口变换决定怎样把制作的图形映射屏幕上。另外，OpenGL 提供了一系列矩阵操作函数，利用这些函数，用户可以根据具体的需要，定义具体应用中的变换，这样有利于具体问题具体分析，消除了系统对应用的局限性。在本书的第三章详细地讲解了这些变换的应用以及矩阵操作函数。

三、着色（Rendering）

OpenGL 提供了两种颜色模式 RGBA 模式和颜色索引表模式和两种具体的物体着色方式。如果显示硬件允许的话，OpenGL 提供 16M 种颜色，基本上是自然界所有的颜色。在本书的第四章中将介绍 OpenGL 提供的颜色模式和具体的着色模式。

四、光照(Iighling)

光是真实感图形的必要组成部分。OpenGL 中认为光是由四部分组成的，这四部分是环境光、漫反射光、辐射光和镜面光。在定义光源的时候要分别定义这四部分。用户可以在应用中定义光源的属性，改变光源的位置。在 OpenGL 中，光源相当于一个几何体，用户可以像控制几何体一样地控制光源的位置、控制光照物体表面的属性以及可以定义聚光。在本书的第四章中将具体地介绍光照。

五、反走样(Antialiasing)

走样是计算机绘制图形过程中常见的问题之一，在 OpenGL 中，提供了点、线和多边形的反走样技术，在本书的第五章中将会做具体地介绍。

六、混合(Blending)

在一些逼真的高质量三维图形制作过程中，经常会遇到透明物体和半透明物体的处理。在 OpenGL 中，这种处理是通过混合技术来实现的。缺省状态下，所有的物体都是不透明的。物体的不透明度是由表示物体颜色的第四个分量 Alpha 来决定。OpenGL 提供了控制比例的混合的函数，用户可以根据自己的需要，选择在实际应用中最合适的混合函数。这部分内容在本书的第五章中将会做具体介绍。

七、雾(Fog)

在制作真实感图形的过程中，经常要有一些烟、雾等特殊效果的制作要求。在 OpenGL 用雾来实现这种自然现象，可以使所制作的三维图形更真实。这部分内容在本书的第五章中介绍。

八、位图和图像（Bitmap and Image)

OpenGL 提供了两种特殊的数据类型——位图和图像。OpenGL 提供了一系列的函数来实现位图操作，例如可以和显示列表结合，方便地实现英文字符的制作和显示。图像是图形制作中的一个重要方面。OpenGL 提供了一系列的函数来实现图像操作，如图像绘制、拷贝、放缩以及图像数据的转换、映射和存储。在本书的第六章将具体介绍。

九、纹理映射（Texture Map)

纹理映射是考察三维图形系统最重要的一个方面。在具体的三维模型制作过程中，在模型表面加上现实世界中物体的纹理，可使三维图形更生动、更自然。OpenGL 提供了纹理堆栈，可以使纹理粘贴在物体以前，对纹理进行变换，包括平移、旋转和缩放。在本书的第七章将要详细讲述 OpenGL 提供的纹理映射。

十、交互操作和动画（Interactive and Animation）

交互操作是考察一个三维图形系统的另一个重要方面，OpenGL 没有提供直接的交互操作函数，OpenGL 辅助库为使用标准 C 编写 OpenGL 程序提供了简单的消息响应函数。在本书的第一章到第八章的程序全部是用辅助库编写的 OpenGL 程序，交互的思想一直贯穿在所有的例程中。MFC 提供了丰富的窗口操作函数和消息响应处理函数，本书的第八章将要详细介绍使用 MFC 编写 OpenGL 程序。动画是考察一个三维图形系统性能的重要方面，OpenGL 中使用双缓存区技术实现动画绘制，可以实时地根据用户需求，按具体的交互效果，绘制出结果。本书的第十章将详细介绍使用 OpenGL 实现动画。

1.2 Windows 系统下的 OpenGL 函数

在 Windows 95 以及 Windows NT3.51 以上的操作系统中提供了 OpenGL 的动态库，在 VC++ 2.0 以上的版本中提供了 OpenGL 的静态库，所以，使用 OpenGL 编程，在微机上使用时，最好是在上述软件环境中编写 OpenGL 程序。

在微机版本中，OpenGL 提供了三个函数库，它们是基本库、实用库和辅助库。

OpenGL 的基本库是 OpenGL 的核心函数库，在这个函数库中，提供了一百多个函数，这些函数都是以“gl”为前缀，所有 OpenGL 提供的操作都可以使用这些函数来实现，而且，对于不同的软件和硬件平台，这些函数的使用是完全相同的，这个特性注定了 OpenGL 程序完美的可移植。

OpenGL 的实用库是 OpenGL 基本库的一套子程序，它提供了四十多个函数，这些函数都是以“glu”为前缀。基本的 OpenGL 不支持传统上同图形标准相关的一些几何对象，为了减少一些编程负担，OpenGL 提供了实用库。实用库中的所有函数全部是由 OpenGL 基本库函数来编写的，所以，在使用上和 OpenGL 基本库的使用是完全相同的，而且，用户也可以使用基本函数库来实现实用库的函数功能。在本书的第八章将详细地介绍 OpenGL 实用库。

OpenGL 的辅助库是为了方便用户用标准 C 编写 OpenGL 程序而编写的。OpenGL 是一个图形标准，所以，在 OpenGL 中没有提供窗口管理和消息事件响应的函数，这样使用标准 C 编写 OpenGL 程序是很不方便的，所以提供了辅助库。OpenGL 辅助库提供了一些基本的窗口管理函数、事件处理函数和一些简单模型的制作函数，例如，定义窗口的大小、处理键盘时间、鼠标击键事件、绘制多面体等等。在本书的第二章将详细地介绍 OpenGL 辅助库。

另外，对于编写 Windows 程序，OpenGL 也提供了一些相关的函数库，在本书使用中会做详细的说明。

第二章　OpenGL 的基本操作

2.1　OpenGL 的基本程序结构

用 OpenGL 编写应用程序，只是相当于在应用程序中填加了一个三维函数库。OpenGL 图形系统可以做很多事情，但一个 OpenGL 程序的基本结构是非常简单的。

一般 OpenGL 程序包括以下几部分：

(一) 定义窗口

即定义所制作的三维图形在 Windows 屏幕坐标系中的显示位置，定义所显示图形窗口的大小、窗口的性质。

(二) 初始化

包括初始化一些基本操作，如清缓存区（buffer）、定义光照模型、定义纹理映射、安装显示列表、定义雾等基本操作的初始化状态、设置三维视景体、定义视口。总之，是为下一步的 OpenGL 图形显示做一些准备工作，初始化 OpenGL 的状态变量，使显示三维 OpenGL 图形更方便，节省显示图形的时间。

(三) 绘制和显示图形

按照应用的具体要求，绘制和显示三维图形包括建立三维模型、设置物体在立体空间的运动轨迹、变更 OpenGL 的状态变量、协调合理地结合运用 OpenGL 各基本操作、实现完美的三维图形显示。

例 2.1 用一个简单的例子来说明 OpenGL 的程序结构。

例 2.1　OpenGL 例程 sample. c

```
#include <window.h>

#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glaux.h>

#include <stdio.h>

void myinit(void);
void CALLBACK myReshape(GLsizei w, GLsizei h);
void CALLBACK display(void);

/*  Initialize antialiasing for RGBA mode, including alpha
 *  blending, hint, and line width.
 */
void myinit(void)
```

```
{
    glClearColor(0.0, 0.0, 0.0, 0.0);
}

/*  display() draws an octahedron with a large alpha value, 1.0.
 */

void CALLBACK display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor4f(0.2, 0.6, 1.0, 1.0);
    glRotatef(60.0,1.0,1.0,1.0);
    auxWireCube(1.0);
    glFlush();
}

void CALLBACK myReshape(GLsizei w, GLsizei h)
{
    glViewport(0, 0, w, h);
}

/*  Main Loop
 *  Open window with initial window size, title bar,
 *  RGBA display mode, and handle input events.
 */

void main(void)
{
    auxInitDisplayMode (AUX_SINGLE | AUX_RGBA);
    auxInitPosition (0, 0, 400, 400);
    auxInitWindow ("sample.c");
    myinit();
    auxReshapeFunc (myReshape);
    auxMainLoop(display);
}
```

运行这个程序和运行标准 C 编写的 C 程序一样，只是在连接的时候要加入三个静态库，这三个静态库是 opengl32.lib、glu32.lib 和 glaux.lib。图 2-1 是 sample.c 的执行结果。为了让读者更容易掌握 OpenGL 编程的技巧，本书基本操作中的例程全部是用标准 C 调用

OpenGL 函数来实现的，窗口系统是用 OpenGL 附带的辅助库实现的，用 MFC 编写 OpenGL 程序将在第九章介绍。

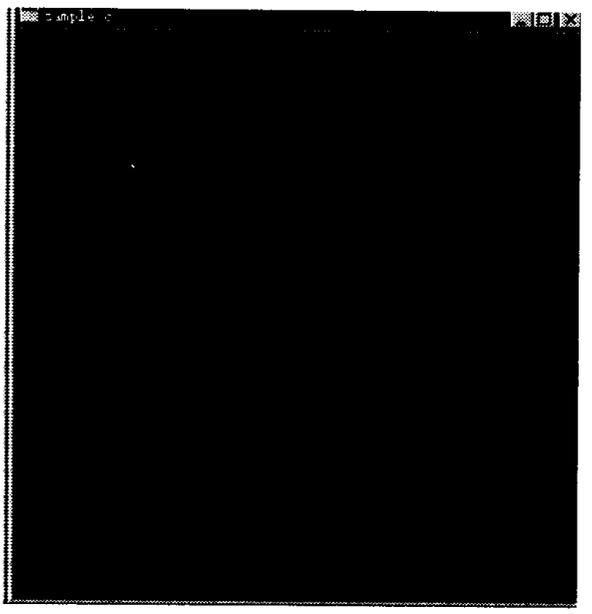

图 2-1 sample.c 的执行结果

下面将对 sample.c 程序进行详细说明。函数 main()是主控制函数。在这个函数中调用了五个以 aux 为前缀的函数，这些函数是辅助库中定义的函数，负责管理窗口和事件。

函数 auxInitPosition(0, 0,400,400)定义了程序执行窗口的位置和大小。即在当前的屏幕坐标系下开一个起始点在（0,0），宽和高分别为 400 和 400 的窗口。

函数 auxInitDisplayMode (AUX_SINGLE | AUX_RGBA)定义了窗口的显示属性。

函数 auxReshapeFunc (myReshape)调用一个用户定义的回调函数 myReshape()。当图形输出窗口的大小和尺寸发生变化时，主程序将自动调用这个函数，通常在 myReshape()中定义视口的大小、定义三维场景的视景体。函数 myReshape()有两个参数 w 和 h，表示当前执行应用程序窗口的宽和高，这样可以保证视口和视景体与窗口的一致性，不会因为窗口的变化而影响图形在窗口的显示。

函数 auxMainLoop(display)定义了一个回调函数 display()。在每次窗口建立、移动、改变形状或其它的一些事件发生时都需要重新绘制场景，函数 display()是用来说明绘制场景的程序的。当窗口需要修改时主程序将自动调用这个函数，由 display()重新绘制场景，所有使用 OpenGL 绘制和显示图形的操作在这个函数中完成。在例 sample.c 中，这个函数中有下列五条语句：

```
glClear(GL_COLOR_BUFFER_BIT);

glColor4f(0.2, 0.6, 1.0, 1.0);
glRotatef(60.0,1.0,1.0,1.0);
```

```
auxWireCube(1.0);
glFlush();
```

以上五条语句中第一条语句完成清屏，后三条语句完成立方体绘制，语句 glFlush()强制绘图完成。

函数 myinit()是用户自己定义的一个函数，主要做一些初始化工作，在程序中这个函数是调用了 OpenGL 的函数 glClearColor()，设定以后的清屏颜色。

这一节主要目的是说明标准 C 调用 OpenGL 函数的基本程序结构，在例 sample.c 中有些语句没有重点讲述，希望读者也不要深究，这些语句将在后面的章节中逐步讲述。通过以上一个简单的例程分析，希望读者对 OpenGL 程序结构有个大致的了解。无论多复杂的 OpenGL 程序设计都是由上述的三个模块组成的，而且这三部分是紧密相关的，希望读者在以后的学习中仔细体会三者的联系。

2.2 标准 C 调用 OpenGL 函数编程

OpenGL 三维图形制作不依赖于具体的硬件和软件平台，所以在 OpenGL 中也没法定义一些窗口管理函数和键盘、鼠标输入的事件响应函数。这样调用 OpenGL 的函数用简单的标准 C 集成环境编写 OpenGL 程序是非常困难的。OpenGL 只提供了简单的图元绘制命令——点、线、多边形。对初学者，运用复杂的建模软件建立物体模型，用 OpenGL 实现操作也是很困难的。

OpenGL 辅助库就是为此目的而设计的，它可以帮助初学者使用标准 C 尽快地掌握 OpenGL，而且用 OpenGL 辅助库编写的应用程序简洁明了。

OpenGL 辅助库的函数大致分为如下几类：

（一）窗口初始化和退出函数

（二）事件响应函数

（三）安装颜色索引函数

（四）三维模型绘制函数

（五）程序运行函数

下面几节将讲述如何使用这些函数。

2.2.1 窗口初始化和退出函数

void auxInitPosition(Glint x,Glint y,GLsizei Width,Glsizei Height)

功能：

指定所建立的窗口在屏幕坐标系中的位置和大小。

参数说明：

(x,y)：

是所开窗口在屏幕坐标系中的左上角的位置。

(Width, Height)：

是所开窗口的宽和高。

void auxInitDisplayMode(GLbitfield mask)

功能：

设置打开窗口的显示模式。

参数说明：

mask：

是所开的窗口的具体显示模式，它的可能值见表 2-1。

表 2-1 auxInitDisplayMod（）参数说明

Mask	参数定义
AUX_RGBA	采用 RGBA 颜色模式
AUX_INDEX	采用颜色索引表模式
AUX_SINGLE	采用单 buffer 模式
AUX_DOUBLE	采用双 buffer 模式
AUX_STENCIL	采用 Stencil
AUX_ACCUM	采用累积 buffer 模式
AUX_DEPTH	采用深度的 buffer 模式

以上参数初学者可能有一些不明白的地方，在以后的章节中将做更详尽的介绍。

以上窗口显示模式参数可以选一个也可同时选几个。但是 AUX_RBGA 和 AUX_INDEX 只可取其中之一，AUX_SINGLE 和 AUX_DOUBLE 也是只可取其中之一。

缺省窗口显示模式是 auxInitDisplayMode(AUX_INDEX|AUX_SINGLE)，这种模式也是通常的窗口模式。

void auxInitWidows(Glbyte *titlesting)

功能：

打开一个由 auxInitDisplayMode()和 auxInitPosition ()函数所定义的窗口。

参数说明：

titlesting：

是个字符串，显示在窗口的标题栏中。

用 auxInitWindow()打开的窗口缺省背景色在 RGBA 颜色模式下是黑色，按下 ESC 键可使程序中断且关闭窗口。

下面语句是一个简单的窗口初始化的例子：

```
auxInitDisplayMode (AUX_SINGLE | AUX_RGB);
auxInitPosition (120, 110, 400, 400);
auxInitWindow ("My Windows");
```

上面程序所定义的窗口见图 2-2。

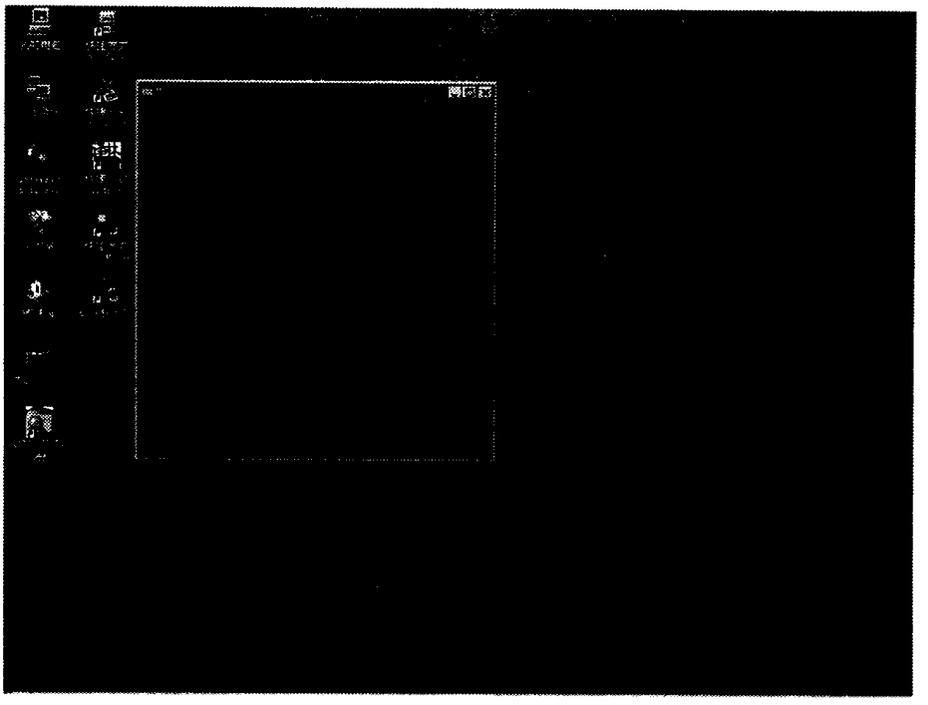

图 2-2 屏幕中的 My Windows 窗口

2.2.2 事件响应函数

void auxReshapFunc(void(*function)(GLsizei x,GLsizei y))

功能：

指定的窗口大小发生变化或者窗口被移动时主程序所调用的函数。

参数说明：

function(x,y)：

是一个用户自己定义的回调函数。当窗口事件发生时，主程序将自动调用这个函数。function()的两个参数是主程序自动返回当前窗口的宽和高。

void auxKeyFunc(GLint key,void (*function)(void))

功能：

当用 key 指定的键被压下后，主程序所做的事情是调用第二个参数所指定的函数。并且重新绘制窗口。

参数说明：

key：

是 OpenGL 辅助库定义的代表键盘键的一些常数。

如：AUX_A、AUX_B～AUX_Z 代表键盘的大写字母 A～Z 键。

AUX_a AUX_b～AUX_z 代表小写字母 a～z 键。

AUX_LEFT、AUX_RIGHT、AUX_UP、AUX_DOWN 代表左右上下键。

AUX_ESCAPE、AUX_RETURN 代表 ESCAPE 键和回车键。

当任何一个击键事件发生后，窗口将被自动的重新绘制。

function():

当所代表的击键事件发生后主程序将调用这个函数。

例如：

auxKeyFunc(AUX_A,left());

当键盘的 A 键被压下以后，主程序将自动调用函数 left()，且重新绘制窗口。

auxMouseFunc(GLint button,GLint mode,void (*function)(AUX_EVENTREC *EVEN))

功能：

当鼠标键被压下以后，主程序将自动调用函数 function()，并且重新绘制窗口。

参数说明：

button：

是 OpenGL 铺助库定义的常数表示鼠标的键。

如:AUX_LEFTBUTTON 代表鼠标左键；

AUX_MIDDLEBUTTON 代表鼠标的中键；

AUX_RIGHTBUTTON 代表鼠标的右键。

mode：

代表鼠标键的状态

AUX_MOUSEDOWN 鼠标键压下事件。

AUX_UPDOWN 鼠标键放开事件。

function：

指定一个函数指针。其参数是辅助库定义的 AUX_EVENTREC 结构。

auxMouseFunc()将自动为这个结构分配存贮空间。

例如：

auxMouseFunc(AUX_LEFTMOUSE,AUX_PRESS,press());

上面语句执行的结果是当鼠标左键被压下以后，主程序将自动调用函数 press()且重新绘制窗口。

void auxIdleFunc(void *Func)

功能：

当没有其它事件挂起时，Func 将被执行。如果这个函数的参数为零将什么也不做。

例：event.c 是一个调用事件响应函数的例程。

例：event.c

```
#include <windows.h>
#include <stdio.h>
#include <GL/gl.h>
#include <GL/glu.h>
```

```
#include <GL/glaux.h>

// disable data conversion warnings
#pragma warning(disable : 4244)     // MIPS
#pragma warning(disable : 4136)     // X86
#pragma warning(disable : 4051)     // ALPHA

void myinit(void);
void CALLBACK myReshape(GLsizei w, GLsizei h);
void CALLBACK display(void);
AUX_EVENTREC *event;
int sign = 1,angle = 30;
void myinit(void)
{
    glLineWidth(5.0);
    glClearColor(0.0, 0.0, 0.0, 0.0);
}

// if the LEFT key is pressed,left() is running
void CALLBACK left(void)
 {
  sign = 1;
}

// if the RIGHT key is pressed,right() is running
void CALLBACK right(void)
 {
  sign = 0;
}

// if the LEFT button of mouse is pressed,leftdown() is running
void CALLBACK leftdown(event){
  angle += 10;
  if(angle >=360) angle = 0;
}

// if the RIGHT button of mouse is pressed,rightdown() is running
void CALLBACK rightdown(event){
```

```
        if(sign)
         sign = 0;
        else
         sign = 1;
}

//display() draw a cube
void CALLBACK display(void)
{
      glClear(GL_COLOR_BUFFER_BIT );
      glColor4f (1.0, 0.6, 0.4, 1.0);
      glPushMatrix();
      glTranslatef(0.0,0.0,-4.0);       //    move object into view

      if(sign){
       glRotatef(angle,1.0,0.0,0.0);// rotate object
       auxWireCube(1.0);                          //draw cube
}
      glPopMatrix();
      glFlush();
}

void CALLBACK myReshape(GLsizei w, GLsizei h)
{
      // Prevent a divide by zero
      if(h == 0)
       h = 1;

       // Set Viewport to window dimensions

      glViewport(0, 0, w, h);
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      // Establish clipping volume
      gluPerspective (45.0, (GLfloat) w/(GLfloat) h, 3.0, 5.0);
      glMatrixMode(GL_MODELVIEW);
      glLoadIdentity ();
```