GLUT教程 （一） 简介

为了用 GLUT 写一个 C 语言程序你需要有三个文件：

1：glut.h—这个头文件是要包含到你的代码里的去的。一般把这个文件放到、include/gl 文件夹里。

2：glut.lib 和 glut32.lib（glut.lib 是 SGI 的，glut32.lib 是 Microsoft 的。好像就是看你用的什么操作系统了。一般 down 的这两个文件都有）这个文件必须连接到你的程序。所以必须放在 lib 文件夹里。

3:glut32.dll（windows）和 glut.dll（SGI）--这个文件必须放在 system32 文件夹里。

在 VC/VC6.0 里的设置

有 Visual C/C++里建立工程可以有两个选择：控制台（console）和 Win32。第一个是最常用的，选第一个的话，应用程序将会有两个窗口，一个控制台窗口（就是命令行那样的窗口）一个 OpenGL 窗口。选择 Win32 的也有可能用 GLUT 和 windows 编程结合建立一个应用程序。所有你必须做的是改变一个设置。

主菜单中选择“工程”（project）->“设置”（setting）

对话框中选择“连接”（link）标签。

在“分类”（Category）组合框里选择“输出”（output）

再在“入口点”（Entry-point symbol）文本框里键入“mainCRTStartup”

对一个现有的控制台应用程序，有一个简单的办法把它转换成 Win32 应用程序，这样可以摆脱那个命令行窗口。

1：接着上面的添加入口点的那个标签。

2：在“工程选项”（Project options）文本框里把“subsystem:console”替换成“subsystem:windows”你也可以仅仅在你的代码的开头添加下面的这一行代码，而不进行上述设置。

```
#pragma comment(linker, "/subsystem:\"windows\" /entry:\"mainCRTStartup\"" )
```

现在这个应用程序就没有控制台窗口，只有 OpenGL 窗口。为了把 GLUT 连接到一个程序里，你还得进行以下几步。

1：选择“工程”（project）->“设置”（settings）。

2：选择“连接”（Link）标签。

3：增加下面的文件到“对象/库模块”(Object/library modules):OpenGL32.lib,glut32.lib,glu32.lib.(一般加一个 glut32.lib 就可以了，添加多个请用空格间隔开来)。

上面添加了 glut32.lib，和 opengl32.lib。这两个都是标准 OpenGL 的库，glu 是一个 API 来自标准的 OpenGL 扩充。

在 VS.NET 里使用 OpenGL

很多人在使用 VS.NET 来建立 OpenGL 应用程序时，都遇到了一个小问题：一个编译器错误。根据我所知道的，好像仅存在与 VS 2003 和 VS2005 的编译器里。下面是 VS2005 里产生的错误。

```
c:/programas/microsoft visual studio 8/vc/include/stdlib.h(406) :
    error C2381: 'exit' : redefinition; __declspec(noreturn) differs
    c:/opengl/toolkits/includes/gl/glut.h(146) : see declaration of 'exit'
```

这个问题好像是因为包含文件时 glut.h 在 stdlib.h 的前面。改下顺序就可以解决这个问题。
把
#include <GL/glut.h>
#include <stdlib.h>
改成
#include <stdlib.h>
#include <GL/glut.h>

GLUT教程（二） GLUT初始化

在这个部分我们将在我们的程序里建立一个 main 函数，这个 main 函数将完成必须的初始化和开启事件处理循环。所有的 GLUT 函数都有 glut 前缀并且那些完成一些初始化的函数有 glutInit 前缀。你首先要做的是调用函数 glutInit()。

Void glutInit(int*argc,char**argv)；

参数：

Argc：一个指针，指向从 main（）函数传递过来的没更改的 argc 变量。

Argv：一个指针，指向从 main（）函数传递过来的没更改的 argv 变量。

在初始化 GLUT 后，我们开始定义我们的窗口。首先确定窗口位置（它默认的是屏幕左上角），我们使用函数 glutInitWindowPosition（）。

Void glutInitWindowPositon(int x,int y);

参数：

X: 距离屏幕左边的像素数。－1 是默认值，意思就是由窗口管理程序决定窗口出现在哪里。如果不使用默认值，那你就自己设置一个值。

Y：距离屏幕上边的像素数。和 X 一样。

注意，参数仅仅是对窗口管理程序的一个建议。尽管你精心的设置了窗口位置，window 返回的可能是不同的位置。如果你设置了，一般会得到你想要的结果。接下来我们设置窗口大小，使用函数 glutInitWindowSize（）。

Void glutInitWindowSize(int width,int height);

参数：

Width：窗口的宽度。

Height：窗口的高度。

同样 width，height 也只是一个参考数字。避免使用负数。

接下来。你应该使用函数 glutInitDisplayMode()定义显示方式。

Void glutInitDisplayMode(unsighed int mode)

参数：

Mode——可以指定下列显示模式

Mode 参数是一个 GLUT 库里预定义的可能的布尔组合。你使用 mode 去指定颜色模式，数量和缓冲区类型。

指定颜色模式的预定义常量有：

1：GLUT_RGBA 或者 GLUT_RGB。指定一个 RGBA 窗口，这是一个默认的颜色模式。

2：GLUT_INDEX。指定颜色索引模式。

这个显示模式还允许你选择单缓冲区或双缓冲区窗口。

1：GLUT_SINGLE.单缓冲区窗口。

2：GLUT_BUFFER.双缓冲区窗口，这是产生流畅动画必须选的。

还可以指定更多，如果你想指定一组特殊的缓冲的话，用下面的变量：

1：GLUT_ACCUM.累积缓冲区。

2：GLUT_STENCIL.模板缓冲区。

3：GLUT_DEPTH.深度缓冲区。

假定你想要一个有单缓冲区，深度缓冲区的 RGB 窗口，你用“或“（|）操作符来建立你想要的显示模式。

glutInitDisplayMode(GLUT_RGB|GLUT_SINGLE|GLUT_DEPTH);

经过上面的这些步骤后，就可以调用函数 glutCreateWindow()来创建窗口了。

Int glutCreateWindow(char* title);

参数：

Title：设置窗口的标题。

glutCreateWindow（）的返回值是一个窗口标识符。后面你可以在 GLUT 里使用这个标识符，不过这个超出了本小节的范围。

现在就有一些代码来完成所有的初始化操作。

```
#include<gl/glut.h>
void main(int argc,char**argv)
{
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_DEPTH|GLUT_SINGLE|GLUT_RGBA);
    glutInitWindowPosition(100,100);
    glutInitWindowSize(320,320);
    glutCreateWindow(“GLUT Tutorial”);
}
```

如果你运行上述代码，你将会得到一个空的黑的控制台窗口，而没有 OpenGL 窗口。并且控制台窗口将很快消失。

在我们渲染一些东西前，还有两件事需要处理。第一告诉 GLUT 哪个函数负责渲染。我们创建一个简单的渲染的函数。下面的这个函数将会清除颜色缓冲区并画一个三角形。

```
void renderScene(void) {
     glClear(GL_COLOR_BUFFER_BIT);
     glBegin(GL_TRIANGLES);
          glVertex3f(-0.5,-0.5,0.0);
          glVertex3f(0.5,0.0,0.0);
          glVertex3f(0.0,0.5,0.0);
     glEnd();
     glFlush();
}
```

上面的函数的名字你可以自己取一个。现在你必须告诉 GLUT 使用我们上面的函数来进行渲染。这个叫寄存回调。让我们告诉 GLUT 这个函数 renderScene 应该被使用。当需要重画的时候 GLUT 有一个只传递一个函数名称参数的函数（以函数名为形参的函数）就会被调用。

void glutDisplayFunc(void (*func)(void));

参数：

func: 当窗口需要被重绘是调用的函数的名称。注意使用 NULL 作为实参是错误的。

最后一件事是告诉 GLUT 我们准备进入应用程序事件处理循环。GLUT 提供了一个函数让程序进入一个永不结束的循环。一直等待处理下一个事件。函数是 glutMainLoop（）。

void glutMainLoop(void)

到目前为止所有的代码都列在下面。如果你运行代码，将会得到一个控制台窗口，和一个画着一个白色三角形的 OpenGL 窗口，出现在你设置的位置，并有着你设置的尺寸。

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

 void renderScene(void) {
        glClear(GL_COLOR_BUFFER_BIT);
        glBegin(GL_TRIANGLES);
              glVertex3f(-0.5,-0.5,0.0);
              glVertex3f(0.5,0.0,0.0);
              glVertex3f(0.0,0.5,0.0);
        glEnd();
        glFlush();
}

void main(int argc, char **argv) {
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_DEPTH | GLUT_SINGLE | GLUT_RGBA);
        glutInitWindowPosition(100,100);
        glutInitWindowSize(320,320);
        glutCreateWindow("3D Tech- GLUT Tutorial");
        glutDisplayFunc(renderScene);
        glutMainLoop();
}
```

GLUT教程（三） GLUT窗口设置

下载下面的VC工程并运行它（glut0.zip）（这个就是上一节的工程）。你将看到两个窗口：一个控制台窗口，一个OpenGL窗口。现在改变窗口大小使高度与宽度不再相等，这时三角形发生变形。这会发生是因为你没有正确设置投影矩阵。默认的是透视投影矩阵且高宽比为1.因此高宽比改变了，投影就会变形。因此只要高宽比改变了，投影就应该重新计算。

GLUT 定义了当窗口大小改变时哪一个函数应该被调用。此外，这个函数还会在窗口初次被创建时调用，保证初始化窗口不是正方形的时候渲染也不会变形出错。

这个函数是 glutReshapeFunc()。

void glutReshapeFunc(void(*func)（int width,int height）);

参数：

func: 指负责设置正确投影的函数的名称。

因此我们必须做的第一件事是回到 main（）函数。在上一章的代码里加入对glutReshapeFunc()的调用，让我们把负责窗口尺寸的函数叫做 changeSize。现在的代码如下。

```
void main(int argc, char **argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DEPTH | GLUT_SINGLE | GLUT_RGBA);
    glutInitWindowPosition(100,100);
    glutInitWindowSize(320,320);
    glutCreateWindow("3D Tech- GLUT Tutorial");
    glutDisplayFunc(renderScene);

    // Here is our new entry in the main function
    glutReshapeFunc(changeSize);

    glutMainLoop();
}
```

下面我们需要做的就是定义函数 changeSize（）。从 glutReshapeFunc（）函数的声明可以看到，changSize（）函数有两个形参。。这两个参数代表新的窗口高度和宽度。

```
void changeSize(int w, int h) {
    // 防止除数即高度为 0
    // (你可以设置窗口宽度为 0).
    if(h == 0)
        h = 1;

    float ratio = 1.0* w / h;

    // 单位化投影矩阵。
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    // 设置视口大小为增个窗口大小
    glViewport(0, 0, w, h);

    // 设置正确的投影矩阵
    gluPerspective(45,ratio,1,1000);
   //下面是设置模型视图矩阵
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(0.0,0.0,5.0, 0.0,0.0,-1.0,0.0f,1.0f,0.0f);
}
```

我们在上一小段代码里引进了一些函数。下面让我们详细讲解，以免让你感到很迷茫。

第一步是计算高宽比（wight/height）。注意为了计算正确，我们必须保证高度不为 0。

接着，我们设置当前矩阵为投影矩阵，这个矩阵定义了一个可视空间（viewing volume）。我们调用一个单位矩阵来初始化投影矩阵。然后我们用函数 glViewport 把视口设置为整个窗口。你也可以设置不同的值。函数中（0，0）指定了视口的左下角，（w,h）指定了视口矩形

的大小。注意这个坐标是和客户区域有关的，而不是屏幕坐标。。

gluPerspective 函数是其他 OpenGL 库（GLU 库）里的一个函数。（gluPerspective 函数的参数，设置请参见其他书籍。我并不想在 GLUT 讲解里再加入其他一些 OpenGL 内容的讲解。）

最后就是设置模型观测矩阵。调用 GL_MODELVIEW 把当前矩阵设为模型观测矩阵。gluLookAt()也是 GLU 库里的一个函数，其参数详细设置参见其他书籍。

下面是这章的VC工程（glut1.zip），你可以自己设置大小，改变参数值。要真正弄懂程序究竟是怎样运行的。

GLUT教程（四） GLUT动画

到现在为止，我们有了一个画着一个白色三角形的 OpenGL 窗口，但一点也不激动人心。现在让我们在这节教程里，让这个三角形自己旋转起来。

让我们回到 main（）函数，增加些额外的设置。首先告诉 GLUT 我们想要一个双缓冲区。双缓冲区通过在后一个缓冲区里绘画，并不停交换前后缓冲区（可见缓冲区），来产生平滑的动画。使用双缓冲区可以预防闪烁。

glutInitDisplayMode(GL_DEPTH|GLUT_DOUBLE|GLUT_RGBA);

接着我们要做的是告诉 GLUT，当应用程序空闲的时候渲染函数应该被调用。这导致 GLUT 一直调用渲染函数而产生动画。GLUT 提供了一个函数：glutIdleFunc.这个函数使另一个函数在程序空闲的时候就会被调用。

void glutIdleFunc(void(*func)(void));

参数：

func：在程序空闲的时候就会被调用的函数的函数名。

按照我们的想法，当程序空闲时应该调用的函数是我们先前定义的渲染函数：renderScene。由于 OpenGL 默认没有开启深度测试，我们还要开启它，这样我们才能知道哪个物体在前面，哪个物体在后面。深度测试的开启在 main（）函数里，下面看看现在的 main 函数。

```
void main(int argc, char **argv) {
    glutInit(&argc, argv);

    // 在这里设置双缓冲区。
    glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);

    glutInitWindowPosition(100,100);
    glutInitWindowSize(320,320);
    glutCreateWindow("3D Tech- GLUT Tutorial");
    glutDisplayFunc(renderScene);

    // 这里让程序空闲时调用 renderScene，
```

```
        glutIdleFunc(renderScene);

        glutReshapeFunc(changeSize);

        //开启深度测试。
        glEnable(GL_DEPTH_TEST);
        glutMainLoop();
}
```

下面就是设置渲染函数 renderScene。我们定义了一个浮点型变量并初始化为 0.0，下面在 renderScene 函数加一些必须的东西。

```
float angle=0.0;

void renderScene(void) {

        //注意我们这里清除了深度缓冲区。
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        //保存当前模型视图矩阵。
        glPushMatrix();

        glRotatef(angle,0.0,1.0,0.0);
        glBegin(GL_TRIANGLES);
            glVertex3f(-0.5,-0.5,0.0);
            glVertex3f(0.5,0.0,0.0);
            glVertex3f(0.0,0.5,0.0);
        glEnd();

        // 弹出堆栈
        glPopMatrix();

        // 交换缓冲区
        glutSwapBuffers();

        // 让 angle 自动增加。
        angle++;
}
```

glutSwapBuffers 函数交换了前后缓冲区，函数原型如下：

```
void glutSwapBuffers();
```

好了，我们得到了一个旋转的三角形，你可以下载这个VC工程在这里（glut2.zip）。很棒吧？。但再次说下，我们不会渲染一些十分精美的画面，这是为了保持代码的简洁。也因为主要是学习GLUT。

GLUT教程（五） GLUT键盘控制

GLUT 允许我们编写程序，在里面加入键盘输入控制，包括了普通键，和其他特殊键（如F1,UP）。在这一章里我们将学习如何去检测哪个键被按下，可以从 GLUT 里得到些什么信息，和如何处理键盘输入。

到现在，你应该注意到了，只要你想控制一个事件的处理，你就必须提前告诉 GLUT，哪个函数将完成这个任务。到现在为止，我们已经使用 GLUT 告诉窗口系统，当窗口重绘时我们想调用哪个渲染函数，但系统空闲时，哪个函数被调用。和当窗口大小改变时，哪个函数又将被调用。

相似的，我们必须做同样的事来处理按键消息。我们必须使用 GLUT 通知窗口系统，当某个键被按下时，哪个函数将完成所要求的操作。我们同样是调用一个函数注册相关的回调函数。

当你按下一个键后，GLUT 提供了两个函数为这个键盘消息注册回调。第一个是glutKeyboardFunc。这个函数是告诉窗口系统，哪一个函数将会被调用来处理普通按键消息。普通键是指字母，数字，和其他可以用 ASCII 代码表示的键。函数原型如下：

void glutKeyboardFunc(void(*func)(unsigned char key,int x,int y));

参数：

func: 处理普通按键消息的函数的名称。如果传递 NULL，则表示 GLUT 忽略普通按键消息。

这个作为 glutKeyboardFunc 函数参数的函数需要有三个形参。第一个表示按下的键的ASCII 码，其余两个提供了，当键按下时当前的鼠标位置。鼠标位置是相对于当前客户窗口的左上角而言的。

一个经常的用法是当按下 ESCAPE 键时退出应用程序。注意，我们提到过，glutMainLoop 函数产生的是一个永无止境的循环。唯一的跳出循环的方法就是调用系统 exit 函数。这就是 我们函数要做的，当按下 ESCAPE 键调用 exit 函数终止应用程序（同时要记住在源代码包含头文件 stdlib.h）。下面就是这个函数的代码：

```
void processNormalKeys(unsigned char key,int x,int y)
{
    if(key==27)
        Exit(0);
}
```

下面让我们控制特殊键的按键消息。GLUT 提供函数 glutSpecialFunc 以便当有特殊键按下的消息时，你能注册你的函数。函数原型如下：

void glutSpecialFunc(void (*func)(int key,int x,int y));

参数：

func: 处理特殊键按下消息的函数的名称。传递 NULL 则表示 GLUT 忽略特殊键消息。

下面我们写一个函数，当一些特殊键按下的时候，改变我们的三角形的颜色。这个函数

使在按下 F1 键时三角形为红色，按下 F2 键时为绿色，按下 F3 键时为蓝色。

```
void processSpecialKeys(int key, int x, int y) {
        switch(key) {
                case GLUT_KEY_F1 :
                                red = 1.0;
                                green = 0.0;
                                blue = 0.0; break;
                case GLUT_KEY_F2 :
                                red = 0.0;
                                green = 1.0;
                                blue = 0.0; break;
                case GLUT_KEY_F3 :
                                red = 0.0;
                                green = 0.0;
                                blue = 1.0; break;
        }
}
```

上面的 GLUT_KEY_*在 glut.h 里已经被预定义为常量。这组常量如下：

```
GLUT_KEY_F1             F1 function key
GLUT_KEY_F2             F2 function key
GLUT_KEY_F3             F3 function key
GLUT_KEY_F4             F4 function key
GLUT_KEY_F5             F5 function key
GLUT_KEY_F6             F6 function key
GLUT_KEY_F7             F7 function key
GLUT_KEY_F8             F8 function key
GLUT_KEY_F9             F9 function key
GLUT_KEY_F10            F10 function key
GLUT_KEY_F11            F11 function key
GLUT_KEY_F12            F12 function key
GLUT_KEY_LEFT           Left function key
GLUT_KEY_RIGHT          Up function key
GLUT_KEY_UP             Right function key
GLUT_KEY_DOWN           Down function key
GLUT_KEY_PAGE_UP        Page Up function key
GLUT_KEY_PAGE_DOWN      Page Down function key
GLUT_KEY_HOME           Home function key
GLUT_KEY_END            End function key
GLUT_KEY_INSERT         Insert function key
```

为了让上面 processSpecialKeys 函数能过编译通过，我们还必须定义，red，green，blue 三个变量。此外为了得到我们想要的结果，我们还必须修改 renderScene 函数。

```
// 所有的变量被初始化为 1，表明三角形最开始是白色的。
float red=1.0, blue=1.0, green=1.0;
```

```
void renderScene(void) {
        glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
        glPushMatrix();
        glRotatef(angle,0.0,1.0,0.0);

        // glColor3f 设置绘制三角形的颜色。
        glColor3f(red,green,blue);

        glBegin(GL_TRIANGLES);
              glVertex3f(-0.5,-0.5,0.0);
              glVertex3f(0.5,0.0,0.0);
              glVertex3f(0.0,0.5,0.0);
        glEnd();
        glPopMatrix();
        angle++;
        glutSwapBuffers();
}
```

。下面我们就该告诉 GLUT，我们刚刚定义的函数用来处理，按键消息。也就是该调用 glutKeyboardFunc 和 glutSpecialFunc 函数。我们在 main 函数里调用它们。下面就是最新的 main 函数。

VC工程可以在这里下载（glut3.zip）

CTRL,ALT 和 SHIFT

一些时候我们想知道要是一个组合键（modifier key）也就是 CTRL,ALT 或者 SHIFT 被按下该如何处理。GLUT 提供了一个函数来检测时候有组合键被按下。这个函数仅仅只能在处理按键消息或者鼠标消息函数里被调用。函数原型如下：

```
int glutGetModifiers(void);
```

这个函数的返回值是三个 glut.h 里预定义的常量里的一个，或它们的或组合。这三个常量是：

1：GLUT_ACTIVE_SHIFT: 返回它，当按下 SHIFT 键或以按下 CAPS LOCK，注意两者同时按下时，不会返回这个值。

2：GLUT_ACTIVE_CTRL: 返回它，当按下 CTRL 键。

3：GLUT_ACTIVE_ATL:返回它,当按下 ATL 键。

注意，窗口系统可能会截取一些组合键（modifiers），这是就没有回调发生。现在让我们扩充 processNormalKeys，处理组合键。按下 r 键时 red 变量被设置为 0.0，当按下 ATL+r 时 red 被设置为 1.0。代码如下：

```
void processNormalKeys(unsigned char key, int x, int y) {
        if (key == 27)
              exit(0);
        else if (key=='r') {
              int mod = glutGetModifiers();
              if (mod == GLUT_ACTIVE_ALT)
                    red = 0.0;
              else
```

```
                red = 1.0;
        }
}
```

注意如果我们按下 R 键，将不会有什么发生，因为 R 与 r 键的 ASCII 码不同。即这是两个不同的键。最后就是如何检测按键 CTRL+ALT+F1?。这种情况下，我们必须同时 检测两个组合键，为了完成操作我们需要使用或操作符。下面的代码段，使你按下 CTRL+ALT+F1 时颜色改变为红色。

```
void processSpecialKeys(int key, int x, int y) {
        int mod;
        switch(key) {
                case GLUT_KEY_F1 :
                  mod = glutGetModifiers();
                   if (mod == (GLUT_ACTIVE_CTRL|GLUT_ACTIVE_ALT)) {
                        red = 1.0; green = 0.0; blue = 0.0;
                   }
                   break;
                case GLUT_KEY_F2 :
                  red = 0.0;
                  green = 1.0;
                  blue = 0.0; break;
                case GLUT_KEY_F3 :
                  red = 0.0;
                  green = 0.0;
                  blue = 1.0; break;
        }
}
```

GLUT教程（六） GLUT场景漫游

让我们看一个比较好的使用键盘控制的例子。这一章我们将建立一个应用程序。这个程序绘制了一个小的居住着雪人的世界。并且我们将用方向键来移动照相机（即移动视点在场景中漫游）。左右方向键，将照相机绕 y 轴旋转，上下方向键，将前后方向移动照相机。

这个例子的代码放在下面。首先我们处理初始状态。

```
#include <math.h>
#include <GL/glut.h>

#include <stdlib.h>

static float angle=0.0,ratio;
static float x=0.0f,y=1.75f,z=5.0f;
static float lx=0.0f,ly=0.0f,lz=-1.0f;
static GLint snowman_display_list;
```

注意我们包含了 math.h 头文件。我们需要计算旋转角。上面变量的含义到后面你就会

清楚了，但我们还是简单的描述下：
1：angle：绕 y 轴的旋转角，这个变量允许我们旋转照相机。
2：x,y,z：照相机位置。
3：lx,ly,lz：一个向量用来指示我们的视线方向。
4：ratio：窗口宽高比（width/height）。
5：snowman_display_list：一个雪人的显示列表索引。

注意：如果你不愿意用显示列表，你也可以忽略它，这并不影响，教程。

接下来，我们用一个公共的函数来处理窗口尺寸。唯一的区别是函数 glutLookAt 的参数用变量而不是固定的值。gluLookAt 函数提供了一个简单直观的方法来设置照相机的位置和方向。它有三组参数，每一组由三个浮点型数组成。前三个参数表明照相机的位置，第二组参数定义照相机观察的方向，最后一组表明向上的向量，这个通常设为（0.0，1.0，0.0）。也就是说照相机并没有倾斜。如你想看到所有的物体都是倒置的则可以设置为（0.0，-1.0，0.0）。

上面提到的变量 x,y,z 表示照相机位置，因此这三个变量也就对应着函数 gluLookAt 里的第一组向量。第二组参数观察方向，是通过定义视线的向量和照相机位置相加得到的：
Look At Point=Line Of Sight+ Camera Position

```
void changeSize(int w, int h)
	{
	// 防止被 0 除.
	if(h == 0)
		h = 1;

	ratio = 1.0f * w / h;
	// Reset the coordinate system before modifying
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();

	//设置视口为整个窗口大小
	 glViewport(0, 0, w, h);

	//设置可视空间
	gluPerspective(45,ratio,1,1000);
	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();
	gluLookAt(x, y, z,
		  x + lx,y + ly,z + lz,
		 0.0f,1.0f,0.0f);
	}
```

下面我们定义显示列表，绘制雪人，初始化场景，渲染场景。

```
void drawSnowMan() {
```

```
        glColor3f(1.0f, 1.0f, 1.0f);

//画身体
        glTranslatef(0.0f ,0.75f, 0.0f);
        glutSolidSphere(0.75f,20,20);


//  画头
        glTranslatef(0.0f, 1.0f, 0.0f);
        glutSolidSphere(0.25f,20,20);

//  画眼睛
        glPushMatrix();
        glColor3f(0.0f,0.0f,0.0f);
        glTranslatef(0.05f, 0.10f, 0.18f);
        glutSolidSphere(0.05f,10,10);
        glTranslatef(-0.1f, 0.0f, 0.0f);
        glutSolidSphere(0.05f,10,10);
        glPopMatrix();

//  画鼻子
        glColor3f(1.0f, 0.5f , 0.5f);
        glRotatef(0.0f,1.0f, 0.0f, 0.0f);
        glutSolidCone(0.08f,0.5f,10,2);
}

GLuint createDL() {
        GLuint snowManDL;

        //生成一个显示列表号
        snowManDL = glGenLists(1);

        //  开始显示列表
        glNewList(snowManDL,GL_COMPILE);

        // call the function that contains
        // the rendering commands
              drawSnowMan();

        // endList
        glEndList();

        return(snowManDL);
```

```
}

void initScene() {

	glEnable(GL_DEPTH_TEST);
	snowman_display_list = createDL();
}


void renderScene(void) {
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

	//画了一个地面

	glColor3f(0.9f, 0.9f, 0.9f);
	glBegin(GL_QUADS);
		glVertex3f(-100.0f, 0.0f, -100.0f);
		glVertex3f(-100.0f, 0.0f, 100.0f);
		glVertex3f( 100.0f, 0.0f, 100.0f);
		glVertex3f( 100.0f, 0.0f, -100.0f);
	glEnd();

	//画了 36 个雪人

	for(int i = -3; i < 3; i++)
		for(int j=-3; j < 3; j++) {
			glPushMatrix();
			glTranslatef(i*10.0,0,j * 10.0);
			glCallList(snowman_display_list);;
			glPopMatrix();
		}
	glutSwapBuffers();
}
```

这里我们建立函数，处理特殊键按下消息。使用左右方向键旋转照相机，也就是改变视线。上下方向键使照相机沿视线前后移动。

```
void inputKey(int key, int x, int y) {

	switch (key) {
		case GLUT_KEY_LEFT :
			angle -= 0.01f;
			orientMe(angle);break;
		case GLUT_KEY_RIGHT :
			angle +=0.01f;
			orientMe(angle);break;
```

```
        case GLUT_KEY_UP :
            moveMeFlat(1);break;
        case GLUT_KEY_DOWN :
            moveMeFlat(-1);break;
    }
```

当我们按下左右方向键时 angle 变量改变，并且 orientMe 被调用。这个函数将旋转照相机。函数 moveMeFlat 负责在 XZ 平面里沿着某一视线移动照相机。

函数 orientMe 接受一个参数 angle 并且为视线的 X,Z 计算出适当的值。新的 lx 和 lz 映射在一个 XZ 平面的单位圆上。因此给定一个角度 ang，新的 lx，lz 的值为：

$$Lx=\sin(ang)；$$

$$Lz=\cos(ang)；$$

就像我们把极坐标（ang，1）转换为欧几里德几何坐标一样。然后我们设定新的照相机方向。注意：照相机并未移动，照相机位置没变，仅仅改变了视线方向。

```
void orientMe(float ang) {

    lx = sin(ang);
    lz = -cos(ang);
    glLoadIdentity();
    gluLookAt(x, y, z,
              x + lx,y + ly,z + lz,
                0.0f,1.0f,0.0f);
}
```

下一个函数就是管理照相机移动的 moveMeFlat。我们想沿视线移动照相机。为了完成这个任务，我们把视线里的一小部分加入到我们的当前的位置。新的 X,Z 的值为：

$$X=x+direction(lx)*fraction$$

$$Z=z+direction*(lz)*fraction$$

方向是 1 或者-1，这取决于我们是前移还是后移。这个 fraction 可以加速视实现。我们知道（lx,lz）是一个整体的向量。因此如果 franction 是个常数那么移动速度也就是一个常量。增大 franction 我们就可以移动的更快。接下来的步骤和 orientMe 函数一样。

```
void moveMeFlat(int direction) {
    x = x + direction*(lx)*0.1;
    z = z + direction*(lz)*0.1;
    glLoadIdentity();
    gluLookAt(x, y, z,
              x + lx,y + ly,z + lz,
                0.0f,1.0f,0.0f);
}
```

这时 main 函数如下：

```
int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
    glutInitWindowPosition(100,100);
```

```
    glutInitWindowSize(640,360);
    glutCreateWindow("SnowMen from 3D-Tech");

    initScene();

    glutSpecialFunc(inputKey);

    glutDisplayFunc(renderScene);
    glutIdleFunc(renderScene);

    glutReshapeFunc(changeSize);

    glutMainLoop();

    return(0);
}
```

这节的VC工程你可以在这里下载（glut4.zip）

GLUT教程（七） GLUT高级键盘控制

这节我们将去介绍 4 个新的处理键盘输入的函数。

第一个函数允许我们去禁止 keyboard repeat。函数原型如下：

int glutSetKeyRepeat(int repeatMode);

参数：

repeatMode：开启，禁用，或恢复 auto repeat 模式，下面是它可能的取值。

RepeatMode 的可能取值如下：

GLUT_KEY_REPEAT_OFF：关闭 auto repeat 模式。

GLUT_KEY_REPEAT_ON：开启 auto repeat 模式。

GLUT_KEY_REPEAT_DEFAULT：把 auto repeat 模式恢复到默认状态。

注意这个函数，作用范围是全局性的。也就是，它会影响所有窗口的 repeat 模式。不仅仅是我们应用程序这一个。因此注意当使用这个函数关闭 auto repeat 模式后，有必要在程序结束时将 auto repeat 模式重设到默认模式。

GLUT 提供我们另外一个简单的函数，来禁用 keyboard repeat，这个让我们安全的忽视 keyboard repeat，而不会影响其他程序。函数原型如下：

Int glutIgnoreKeyRepeat(int repeatMode);

参数：

RepeatMode：传递 0，开启 auto repeat，非 0 则禁用 auto repeat。

在一些情况下，当 key repeat 发生时，我们将不接受函数回调。然而如果你想在一个 key 被按下后，执行一个动作，你就需要知道这个 key 什么时候松开。GLUT 提供了两个函数注

册相关的回调函数。

Void glutKeyboardUpFunc(void (*func)(unsigned char key,int x,int y));

Void glutSpecialUpFunc(void (*func)(int key,int x,int y));

参数：

Func：回调函数的函数名。

我们在下一节，提供一个程序也就是上一节的代码，来看看这些函数怎么工作。

（这章很不好翻译。好多都不知道怎么说，汗。- -|| 有需要的看原文。http://www.lighthouse3d.com/opengl/glut/index.php?7 我感觉原文写的也不咋清楚。还是看下一节的例子。 呵呵。）

GLUT教程（八） GLUT场景漫游II

这一节里，我们再来看看上次的例子，这次我们讲使用高级的键盘控制。

在初始化那部分，我们有两个变量：deltaAngle 和 deltaMode。这些变量控制旋转和移动照相机。当为非 0 时，照相机执行一些动作，当为 0 时，照相机就不动，这两个变量的初始值是 0，也就是说，照相机初始状态是不动的。

```
#include <math.h>
#include <GL/glut.h>

float angle=0.0,deltaAngle = 0.0,ratio;
float x=0.0f,y=1.75f,z=5.0f;
float lx=0.0f,ly=0.0f,lz=-1.0f;
GLint snowman_display_list;
int deltaMove = 0;


void changeSize(int w, int h)
    {
    // Prevent a divide by zero, when window is too short
    // (you cant make a window of zero width).
    if(h == 0)
        h = 1;

    ratio = 1.0f * w / h;
    // 重置投影矩阵
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    // 设置视口大小。
    glViewport(0, 0, w, h);
```

```
        //  设置可视空间
        gluPerspective(45,ratio,1,1000);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        gluLookAt(x, y, z,
                  x + lx,y + ly,z + lz,
                     0.0f,1.0f,0.0f);


        }


void drawSnowMan() {
        glColor3f(1.0f, 1.0f, 1.0f);
// Draw Body
        glTranslatef(0.0f ,0.75f, 0.0f);
        glutSolidSphere(0.75f,20,20);

// Draw Head
        glTranslatef(0.0f, 1.0f, 0.0f);
        glutSolidSphere(0.25f,20,20);

// Draw Eyes
        glPushMatrix();
        glColor3f(0.0f,0.0f,0.0f);
        glTranslatef(0.05f, 0.10f, 0.18f);
        glutSolidSphere(0.05f,10,10);
        glTranslatef(-0.1f, 0.0f, 0.0f);
        glutSolidSphere(0.05f,10,10);
        glPopMatrix();

// Draw Nose
        glColor3f(1.0f, 0.5f , 0.5f);
        glRotatef(0.0f,1.0f, 0.0f, 0.0f);
        glutSolidCone(0.08f,0.5f,10,2);
}

GLuint createDL() {                    //返回一个显示列表
        GLuint snowManDL;

        // Create the id for the list
        snowManDL = glGenLists(1);
```

```
        // start list
        glNewList(snowManDL,GL_COMPILE);

        // call the function that contains
        // the rendering commands
            drawSnowMan();

        // endList
        glEndList();

        return(snowManDL);
}

void initScene() {

        glEnable(GL_DEPTH_TEST);
        snowman_display_list = createDL();

}

void orientMe(float ang) {
        lx = sin(ang);
        lz = -cos(ang);
        glLoadIdentity();
        gluLookAt(x, y, z,
                x + lx,y + ly,z + lz,
                   0.0f,1.0f,0.0f);
}

void moveMeFlat(int i) {
        x = x + i*(lx)*0.1;
        z = z + i*(lz)*0.1;
        glLoadIdentity();
        gluLookAt(x, y, z,
                x + lx,y + ly,z + lz,
                   0.0f,1.0f,0.0f);
}
```

下面的代码和以前的有些不同了。下面这个函数最开始，检查我们定义的那两个变量是否为 0，如果不为 0 就调用相应的函数，来控制照相机的移动，和旋转。

```
void renderScene(void) {

        if (deltaMove)
            moveMeFlat(deltaMove);
        if (deltaAngle) {
```

```
            angle += deltaAngle;
            orientMe(angle);
        }
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
// Draw ground
        glColor3f(0.9f, 0.9f, 0.9f);
        glBegin(GL_QUADS);
            glVertex3f(-100.0f, 0.0f, -100.0f);
            glVertex3f(-100.0f, 0.0f, 100.0f);
            glVertex3f( 100.0f, 0.0f, 100.0f);
            glVertex3f( 100.0f, 0.0f, -100.0f);
        glEnd();
// Draw 36 SnowMen
        for(int i = -3; i < 3; i++)
            for(int j=-3; j < 3; j++) {
                glPushMatrix();
                glTranslatef(i*10.0,0,j * 10.0);
                glCallList(snowman_display_list);;
                glPopMatrix();
            }
        glutSwapBuffers();
}
```

下面的函数，分别是我们注册的特殊键控制的回调函数，和键的释放（键的松开）。

```
void pressKey(int key, int x, int y) {

        switch (key) {
            case GLUT_KEY_LEFT :
                deltaAngle = -0.01f;break;
            case GLUT_KEY_RIGHT :
                deltaAngle = 0.01f;break;
            case GLUT_KEY_UP :
                deltaMove = 1;break;
            case GLUT_KEY_DOWN :
                deltaMove = -1;break;
        }
}

void releaseKey(int key, int x, int y) {

        switch (key) {
            case GLUT_KEY_LEFT :
            case GLUT_KEY_RIGHT :
                deltaAngle = 0.0f;break;
            case GLUT_KEY_UP :
```

```
            case GLUT_KEY_DOWN :
                deltaMove = 0;break;
        }
}
```

Main 函数里，有三行新的代码行。

glutIgnoreKeyRepeat 函数，传递了一个非 0 值告诉 GLUT 不发送 keyborad repeat 消息，然后，glutSpecialUpFunc 和 glutKeyboardUpFunc 调用注册的回调函数。

```
int main(int argc, char **argv)
{
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
        glutInitWindowPosition(100,100);
        glutInitWindowSize(640,360);
        glutCreateWindow("SnowMen from 3D-Tech");

        initScene();

        glutIgnoreKeyRepeat(1);
        glutSpecialFunc(pressKey);
        glutSpecialUpFunc(releaseKey);

        glutDisplayFunc(renderScene);
        glutIdleFunc(renderScene);

        glutReshapeFunc(changeSize);

        glutMainLoop();

        return(0);
}
```

可以在这里下载这节的VC6.0 工程（glut7.zip）。还可以自己改下照相机设置的函数。

GLUT教程（九） GLUT鼠标

在前几节，我们看了怎么使用 GLUT 的 keyboard 函数，来增加一个 OpenGL 程序的交互性。现在，是时候研究下鼠标了。GLUT 的鼠标接口提供一些列的选项来增加鼠标的交互性。也就是检测鼠标单击，和鼠标移动。

检测鼠标 Clicks

和键盘处理一样，GLUT 为你的注册函数（也就是处理鼠标 clicks 事件的函数）提供了

一个方法。函数 glutMouseFunc,这个函数一般在程序初始化阶段被调用。函数原型如下：

void glutMouseFunc(void(*func)(int button,int state,int x,int y));

参数：

func：处理鼠标 click 事件的函数的函数名。

从上面可以看到到，处理鼠标 click 事件的函数，一定有 4 个参数。第一个参数表明哪个鼠标键被按下或松开，这个变量可以是下面的三个值中的一个：

GLUT_LEFT_BUTTON
GLUT_MIDDLE_BUTTON
GLUT_RIGHT_BUTTON

第二个参数表明，函数被调用发生时，鼠标的状态，也就是是被按下，或松开，可能取值如下：

GLUT_DOWN
GLUT_UP

当函数被调用时，state 的值是 GLUT_DOWN，那么程序可能会假定将会有个 GLUT_UP 事件，甚至鼠标移动到窗口外面，也如此。然而，如果程序调用 glutMouseFunc 传递 NULL 作为参数，那么 GLUT 将不会改变鼠标的状态。

剩下的两个参数（x,y）提供了鼠标当前的窗口坐标（以左上角为原点）。

检测动作（motion）

GLUT 提供鼠标 motion 检测能力。有两种 GLUT 处理的 motion：active motion 和 passive motion。Active motion 是指鼠标移动并且有一个鼠标键被按下。Passive motion 是指当鼠标移动时，并有没鼠标键按下。如果一个程序正在追踪鼠标，那么鼠标移动期间，没一帧将产生一个结果。

和以前一样，你必须注册将处理鼠标事件的函数（定义函数）。GLUT 让我们可以指定两个不同的函数，一个追踪 passive motion，另一个追踪 active motion

它们的函数原型，如下：

void glutMotionFunc(void(*func)(int x,int y));
void glutPassiveMotionFunc(void (*func)(int x,int y));

参数：

Func：处理各自类型 motion 的函数名。

处理 motion 的参数函数的参数（x,y）是鼠标在窗口的坐标。以左上角为原点。

检测鼠标进入或离开窗口

GLUT 还能检测鼠标鼠标离开，进入窗口区域。一个回调函数可以被定义去处理这两个事件。GLUT 里，调用这个函数的是 glutEntryFunc,函数原型如下：

void glutEntryFunc(void(*func)（int state）);

参数：

Func：处理这些事件的函数名。

上面函数的参数中，state 有两个值：

GLUT_LEFT
GLUT_ENTERED

表明，是离开，还是进入窗口。

把它们放一起

首先我们要做的是在 GLUT 里定义哪些函数将负责处理鼠标事件。因此我们将重写我们的 main 函数，让它包含所有必须的回调注册函数。我们将在程序里描述其他一些教程里没说清楚的地方。

```
void main(int argc, char **argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
    glutInitWindowPosition(100,100);
    glutInitWindowSize(320,320);
    glutCreateWindow("SnowMen");
    glutDisplayFunc(renderScene);
    glutIdleFunc(renderScene);
    glutReshapeFunc(changeSize);

    //adding here the mouse processing callbacks
    glutMouseFunc(processMouse);
    glutMotionFunc(processMouseActiveMotion);
    glutPassiveMotionFunc(processMousePassiveMotion);
    glutEntryFunc(processMouseEntry);

    glutMainLoop();
}
```

OK，现在做点有趣的。我们将定义那些将做一些不可思议事件的回调函数。当一个鼠标键和 alt 键都被按下，我们将改变三角形的颜色。鼠标左键使三角形变成红色，中间的将三角形变成绿色，鼠标右键将三角形变成蓝色。函数如下：

```
void processMouse(int button, int state, int x, int y) {


    specialKey = glutGetModifiers();
    //  当鼠标键和 alt 键都被按下
    if ((state == GLUT_DOWN) &&
            (specialKey == GLUT_ACTIVE_ALT)) {

        // set the color to pure red for the left button
        if (button == GLUT_LEFT_BUTTON) {
            red = 1.0; green = 0.0; blue = 0.0;
        }
        // set the color to pure green for the middle button
        else if (button == GLUT_MIDDLE_BUTTON) {
            red = 0.0; green = 1.0; blue = 0.0;
        }
        // set the color to pure blue for the right button
```

```
            else {
                red = 0.0; green = 0.0; blue = 1.0;
            }
        }
}
```

接下来有一个精细的颜色拾取方法。当一个鼠标键被按下，但 alt 键被被按下。我们把 blue 设为 0.0，并且让 red 和 green 分量的值取决于鼠标在窗口中的位置。。函数如下：

```
void processMouseActiveMotion(int x, int y) {

        // the ALT key was used in the previous function
        if (specialKey != GLUT_ACTIVE_ALT) {
            // setting red to be relative to the mouse
            // position inside the window
            if (x < 0)
                red = 0.0;
            else if (x > width)
                red = 1.0;
            else
                red = ((float) x)/height;
            // setting green to be relative to the mouse
            // position inside the window
            if (y < 0)
                green = 0.0;
            else if (y > width)
                green = 1.0;
            else
                green = ((float) y)/height;
            // removing the blue component.
            blue = 0.0;
        }
}
```

下面给 passive motion 添加一些动作。当 shift 键被按下，鼠标将在 x 轴上有一个旋转。我们不得不修改 renderScene 函数。函数如下：

```
float angleX = 0.0;
...
void renderScene(void) {
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glPushMatrix();
        glRotatef(angle,0.0,1.0,0.0);

        // This is the line we added for the
        // rotation on the X axis;
        glRotatef(angleX,1.0,0.0,0.0);
```

```
        glColor3f(red,green,blue);

        glBegin(GL_TRIANGLES);
            glVertex3f(-0.5,-0.5,0.0);
            glVertex3f(0.5,0.0,0.0);
            glVertex3f(0.0,0.5,0.0);
        glEnd();
        glPopMatrix();
        angle++;
        glutSwapBuffers();
}
```

现在我们的有个函数处理 passive motion 事件。函数将改变 angleX 的值。

```
void processMousePassiveMotion(int x, int y) {

        // User must press the SHIFT key to change the
        // rotation in the X axis
        if (specialKey != GLUT_ACTIVE_SHIFT) {

            // setting the angle to be relative to the mouse
            // position inside the window
            if (x < 0)
                angleX = 0.0;
            else if (x > width)
                angleX = 180.0;
            else
                angleX = 180.0 * ((float) x)/height;
        }
}
```

最后鼠标离开窗口将使动画停止，为了做到这，我们也需要改变函数 renderScene。

```
// initially define the increase of the angle by 1.0;
float deltaAngle = 1.0;
...
void renderScene(void) {
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glPushMatrix();
        glRotatef(angle,0.0,1.0,0.0);
        glRotatef(angleX,1.0,0.0,0.0);
        glColor3f(red,green,blue);

        glBegin(GL_TRIANGLES);
            glVertex3f(-0.5,-0.5,0.0);
            glVertex3f(0.5,0.0,0.0);
            glVertex3f(0.0,0.5,0.0);
        glEnd();
```

```
        glPopMatrix();
        // this is the new line
        // previously it was: angle++;
        angle+=deltaAngle;
        glutSwapBuffers();
}
```

processMouseEntry 是最后一个函数。注意，这个在微软操作系统下可能工作的不是很好。

```
void processMouseEntry(int state) {
        if (state == GLUT_LEFT)
                deltaAngle = 0.0;
        else
                deltaAngle = 1.0;
}
```

VC6.0 工程可以在这里下载（glut8.zip）。

（到这里位置，键盘，鼠标方面的控制讲完了，下面就是菜单了。）

GLUT教程（十） GLUT菜单

弹出式菜单（像点鼠标右键出来的菜单那样的）也是 GLUT 的一部分，虽然它不能实现我们经常看到的 windows 系统弹出式菜单的所有的功能，但是它也有很大的作用。给一个程序增加菜单提供了一个比键盘更简单的方法来和程序交互，选择不同选项，而不用去记那些按键。

我们首先要做的是创建菜单，创建菜单函数 glutCreateMenu 的原型如下：

int glutCreateMenu（void （*func）（int value））；

参数：

func：为新建的菜单处理菜单事件的函数名。

这个函数的返回值是菜单的标识符（menu identifier）。

我们的程序中，我们可以相加多少菜单就加多少菜单。对每个菜单我们要指定一个回调函数，而且我们可以指定相同的函数。下面为菜单增加一些条目（出来个空菜单也没什么用）。使用的函数是 glutAddMenuEntry：

void glutAddMenuEntry（char *name，int value）；

参数：

name：菜单名称的字符串。

value：当你选择菜单里的一项后，这个值就返回给上面的 glutCreateMenu 里调用的函数。

这个函数根据函数名来看，就是给菜单里添加条目的，可以一直添加（这里有个顺序，自己实验下就明白了的）。

好了现在有了一个弹出式菜单。但还有最后一件事要做，就是把菜单和一个鼠标键连接起来（attach）。因为我们必须指定菜单怎么出现，使用 GLUT 你可以在按下一个鼠标按键

后让菜单显示，函数是 glutAttachMenu：

void glutAttachMenu（int button）；

参数：

button: 一个整数，指定菜单和哪个鼠标键关联起来。

botton 可以去下面的值;

GLUT_LEFT_BUTTON

GLUT_MIDDLE_BUTTON

GLUT_RIGHT_BUTTON

下面就是一个应用了上面所有函数的例子。

```
...
#define RED 1
#define GREEN 2
#define BLUE 3
#define WHITE 4
...
void createGLUTMenus() {

    int menu;

    // 创建菜单并告诉 GLUT，processMenuEvents 处理菜单事件。
    menu = glutCreateMenu(processMenuEvents);

    //给菜单增加条目
    glutAddMenuEntry("Red",RED);
    glutAddMenuEntry("Blue",BLUE);
    glutAddMenuEntry("Green",GREEN);
    glutAddMenuEntry("White",WHITE);

    // 把菜单和鼠标右键关联起来。
    glutAttachMenu(GLUT_RIGHT_BUTTON);
}
```

注意 RED,BLUE,GREEN,和 WHITE 必须定义为整数，再就是你必须为每个选单（菜单里的条目）定义不同的 value，

下面我们写处理菜单事件的函数。我们将使用我们的菜单来设置三角形的颜色。函数如下：

```
void processMenuEvents(int option) {
    //option，就是传递过来的 value 的值。
    switch (option) {
        case RED :
            red = 1.0;
            green = 0.0;
            blue = 0.0; break;
```

```
            case GREEN :
                red = 0.0;
                green = 1.0;
                blue = 0.0; break;
            case BLUE :
                red = 0.0;
                green = 0.0;
                blue = 1.0; break;
            case WHITE :
                red = 1.0;
                green = 1.0;
                blue = 1.0; break;
        }
}
```

剩下来的就是把我们的 createGLUTMenus 函数放到 main 函数里。下面的代码就是当前的 main 函数。

```
void main(int argc, char **argv) {
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
        glutInitWindowPosition(100,100);
        glutInitWindowSize(320,320);
        glutCreateWindow("SnowMen");
        glutDisplayFunc(renderScene);
        glutIdleFunc(renderScene);
        glutReshapeFunc(changeSize);

        //调用我们的函数来创建菜单
        createGLUTMenus();

        glutMainLoop();
}
```

VC工程可以在这里下载（glut7.zip）。

下面我们还来看两个函数，第一个允许我们断开鼠标按键和一个菜单的关联。前面我们用 glutAttachMenu 来在鼠标和菜单间建立关联，但我们有时候需要断开这种关联。完成这个工作的函数是 glutDetachMenu。函数原型如下：

void glutDetachMenu（int button）；

参数：

button：要断开的鼠标按键。

Button 的取值和 glutAttachMenu 一样。因此，要是我们想断开关联我们可以这样：

glutDetachMenu(GLUT_RIGHT_BUTTON);

最后，如果你想恢复被菜单使用了的资源，我们可以销毁（destroy）它，相应的函数是 glutDestroyMenu，它的原型如下：

void glutDestroyMenu（int menuIdentifier）；

参数：

menuIdentifier：要销毁的菜单的标识符，它必须和函数 glutCreateMenu 返回的值相同。好了，到这里你已经知道了基本的在 GLUT 中建立菜单，下章我们将探索更多的弹出式菜单功能。