电子科技大学信息与软件工程学院

实 验 指 导 书

（实验）课程名称 图形与动画 II

电子科技大学教务处制表

目　　录

实验一 OpenGL 基础实验（一）

一、实验目的

1. 掌握 OpenGL 环境配置方法；
2. 掌握在空间中绘图的基本方法；
3. 掌握移动物体的基本方法；
4. 掌握生成简单动画的方法；
5. 掌握键盘事件响应的方法

二、实验原理

（一） OpenGL 简介

1. 什么是 OpenGL？

OpenGL 是一个 3D 图形和模型库，具有高度的可移植性，并且具有非常快的速度。使用 OpenGL，可以创建优质的 3D 图像。OpenGL 并不像 C 或 C++一样是门编程语言，它更像一个 C 运行时调用的函数库，提供一些预先设定好的功能。程序员可以使用任何语言来编写动画程序，在需要某项绘图功能时调用 OpenGL 所提供的函数即可。因此，当我们提到一个基于 OpenGL 的程序或者一个 OpenGL 应用程序时，实际是指这个程序是用其他编程语言(C,C++,java 等)编写，但它调用了 OpenGL 函数库。

2. OpenGL API 规范

2.1 数据类型

为了使 OpenGL 代码易于从一个平台移植到另一个平台，OpenGL 定义了它自己的数据类型。这些数据类型对应了 C 标准的数据类型。当然，也可以在 OpenGL 程序中使用标准的 C 数据类型。但是，各种编译器和操作系统可能会采用其规则定义各种 C 数据类型的大小和内存布局，从而使标准数据类型在不同的环境中存在一些差别。使用 OpenGL 定义的数据类型，可以避免平台不一致造成的影响。

OpenGL 数据类型和对应的 C 数据类型

OpenGL 数据类型	内部表示形式	对应的 C 类型	C 字面值后缀

GLbyte	8 位整数	signed char	b
GLshort	16 位整数	short	s
GLint,GLsizei	32 位整数	long	l
GLfloat, GLclampf	32 位浮点数	float	f
GLdouble，GLclampd	64 位浮点数	Double	d
Glubyte，GLboolean	8 位无符号整数	unsigned char	ub
GLushort	16 位无符号整数	unsigned short	us
GLuint,GLenum,GLbitfield	32 位无符号整数	unsigned long	ui

2.2 函数名约定

绝大多数 OpenGL 函数遵循一种命名约定，据此可以判断这个函数来自于哪个函数库，函数接受的参数个数和类型。

<函数库前缀><根命令><可选的参数数量><可选的参数类型>

如：glColor3f，gl 表示 gl 函数库，根名称 Color 表示这个命令和色彩有关，3 表示参数数量，f 表示参数类型为浮点型。

（二） GLUT 简介

GLUT 是 OpenGL 工具箱，全名为 OpenGL Utility Toolkit。作者 Mark J. Kilgard 把它设计成跨平台的库。GLUT 方便了在任何特定的平台下实现 GUI 编程。通过 GLUT 编写 OpenGL 通常只需要增加几个 GLUT 函数，如利用 GLUT 你可以 5 行代码实现一个 OpenGL 窗口,再用 3-4 行代码实现你的鼠标键盘事件，这样编程人员不需要知道每个不同操作系统处理窗口的函数，一切 GLUT 处理。GLUT 令代码变得简单。

所有 GLUT 函数都以 glut 作为开头，如 glutPostRedisplay()。

（三） GLUT 安装

1. 为了能够使用 GLUT 写程序，首先应该有最新版本的 GLUT 库。下载最新版本 glut 库，解压缩后包括如下文件：
    - glut.dll
    - glut32.dll
    - glut.h
    - glut.lib
    - glut32.lib
2. 将 **glut.dll** 和 **glut32.dll** 放入系统盘 **windows** 文件夹下；
3. 将 **glut.h** 放在 **vs** 安装路径下文件夹 **VC** 的 **include** 文件夹下（如果是 2017 版，放在目录 Program Files (x86)\Microsoft Visual Studio\2017\Community\VC\Auxiliary\VS\include）
4. 将 **glut.lib** 以及 **glut32.lib** 放在 **vs** 安装路径下文件夹 **VC** 的 **lib** 文件夹下；

(如果是 2017 版，放在目录 Program Files (x86)\Microsoft Visual Studio\2017\Community\VC\Auxiliary\VS\lib\x86)

5. 启动 vs，新建一个 project“Win32 Console Application”；
6. 新建一个空白 project；
7. 编写代码时，加入头文件#include <glut.h>即可。

注意：若编译链接出现“LINK : fatal error LNK1123: 转换到 COFF 期间失败: 文件无效或损坏”的错误，将 项目|项目属性|配置属性|链接器|清单文件|生成清单 “是”改为“否”。

（四） GLUT 初始化

这一节开始从 main 函数入手。第一步实现初始化 GLUT 库和创建窗口。

GLUT 进入事件处理循环之后会获得程序的控制权。GLUT 会等待**事件**(event)发生,然后检查有没有绑定的函数来处理它。所以在 GLUT 进入它的事件处理循环之前,我们要先告诉 GLUT 事件发生时需要调用哪个函数来处理。

那么什么是事件(event)？事件就是诸如键盘的键被按下，鼠标被移动，窗口需要绘制(到显示器屏幕)，还有窗口被改变大小，还有很多。

告诉 GLUT 一个事件对应哪个函数的方法是**注册回调函数**(callback function)。每一个事件类型 GLUT 都提供一个指定的函数来注册回调函数。

我们下面先从处理绘制事件开始。main 函数的框架如下:

```
int main(int argc, char **argv)
{

    // init GLUT and create window

    // register callbacks

    // enter GLUT event processing cycle

}
```

图 1

我们看到主函数实际由三个部分构成：

- **初始化 GLUT 和创建窗体；**
- **渲染函数和回调注册；**
- **GLUT 事件处理队列；**

接下来，看看这三部分分别实现什么功能，以及具体怎样实现。

1. 初始化 GLUT 和创建窗体

所有 GLUT 函数都以 glut 开头(作为名字前缀),而 GLUT 的初始化函数都以 glutInit 开头。第一步是调用 glutInit 函数,原型如下:

1.1 glutInit 函数

在主函数中首先调用函数 glutInit()初始化 GLUT，函数原型如下：

void **glutInit**(int*argc,char**argv);

参数：

- argc 和 argv 都是只读的， main 函数传入变量的指针

1.2 glutInitWindowPosition 函数

在初始化 GLUT 后，开始定义窗口的属性，即确定窗口**左上角**的位置。通过函数 glutInitWindowPosition（）实现，原型如下：

void **glutInitWindowPositon**(int x,int y);

参数：

- x: 屏幕左边的像素值。 -1 表示使用默认值,意味着它交由 window 管理器来决定窗体出现在何处,不想这样的话就给正值；
- y：屏幕顶部的像素值，同上；

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。

1.3 glutInitWindowSize 函数

接下来设置窗口大小，使用函数 glutInitWindowSize（），原型如下：

void **glutInitWindowSize**(int width,int height);

参数：

- width：窗口的宽度。
- height：窗口的高度。

1.4 glutInitDisplayMode 函数

接下来应该使用函数 glutInitDisplayMode()定义显示模式，原型如下：

void **glutInitDisplayMode**(unsighed int mode);

参数：

- Mode—可以指定多种显示模式；Mode 参数是一个 GLUT 库里预定义的可能的布尔组合。可使用 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。

制定**单缓冲区**或**双缓冲区**窗口：

1) GLUT_SINGLE - 单缓冲区窗口。
2) GLUT_DOUBLE - 双缓冲区窗口，这是产生流畅动画必须选的。

还可以指定更多的**缓冲集合**，如果你想指定一组特殊的缓冲的话，用下面的变量：

1) 1：GLUT_ACCUM - 累积缓冲区。
2) 2：GLUT_STENCIL - 模板缓冲区。
3) 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);

1.5 glutCreateWindow 函数

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

Int **glutCreateWindow**(char* title);

参数：

- Title：设置窗口的标题。

glutCreateWindow（）的返回值是一个窗口 ID，该 ID 后面几章会用到。至此，初始化代 GLUT 和创建窗体完成，完整代码见图 2。

```
// Main program entry point
int main(int argc, char* argv[])
{
    // init GLUT and create window
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition(100,100);
    glutInitWindowSize(320,320);
    glutCreateWindow("First GLUT");

    return 0;
}
```

图 2

注意：代码开头的 include 语法包含的头文件是“glut.h”。

2. 渲染函数和回调注册

如果运行图 2 中代码，将会得到一个空的黑色的命令行窗口，而没有绘制图形的 OpenGL 窗口，并且控制台窗口将很快消失。这是因为在渲染前，还有两件事需要处理：

第一告诉 GLUT 库渲染的响应函数（哪个函数负责渲染），该函数在每次窗口绘制或重绘的时候被 GLUT 调用。下面我们创建一个简单的渲染的函数。这个函数将会清除颜色缓冲区。

2.1 glutDisplayFunc 函数

绘制函数实现如下：

```
// Called to draw scene
void renderScene(void)
{
    // Clear the window with current clearing color
    glClear(GL_COLOR_BUFFER_BIT);


    // Flush drawing commands
    glFlush();
}
```

图 3

glFlush 函数告诉 OpenGL 应该处理提供给它的绘图指令。

该绘制函数的名字可以任意取。定义好绘制函数后，必须告诉 GLUT 使用我们定义的这个函数来进行渲染，实际上是把这个函数绑定到 GLUT 中，这个操作叫做**注册一个回调**。GLUT 会在需要渲染的时候调用这个函数。

现在就告诉 GLUT 库 renderScene 函数需要在窗体绘制的时候调用（图 4）。GLUT 有一个专门函数来做绑定回调函数的操作，这个函数需要在窗体创建的时候调用，原型如下：

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。

3. GLUT 事件处理队列

3.1 glutMainLoop 函数

最后一件事是告诉 GLUT 可以开始获取事件处理队列。GLUT 提供了一个函数以死循环的方式等待下一个要处理的下一个事件。该函数是 glutMainLoop，原型如下：

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黑色的 OpenGL 窗口，出现在你设置的位置，并有着你设置的尺寸。

```
int main(int argc, char *argv[]) {
    // init GLUT and create Window
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
    glutInitWindowPosition(100,100);
    glutInitWindowSize(320,320);
    glutCreateWindow("GLUT Tutorial");

    // register callbacks
    glutDisplayFunc(renderScene);

    //enter GLUT event processing cycle
    glutMainLoop();

    return 0;
}
```

图 4

3.2 绘图初始化

如果希望用某一颜色清除窗口，那么需要设置 glClearColor 函数，其原型为：

void **glClearColor**(GLclampf red, GLclampf green, GLclampf blue, GLclampf alpha);

在 OpenGL 中，一种颜色是由红、绿、蓝三种原色混合而成，每种成分的值范围从 0.0 到 1.0。最后一个参数是 alpha，用于混合产生一种特殊的效果，如半透明。

设置了 glClearColor 函数后，OpenGL 使用该颜色作为清除色，由 glClear(GL_COLOR_BUFFER_BIT)实现。

绘图初始化我们单独放在一个函数中：

```
// Setup the rendering state
void SetupRC(void)
{
    glClearColor(0.0f, 0.0f, 1.0f, 1.0f);
}
```

图 5

新的 main 函数如下：

```
int main(int argc, char *argv[]) {
    // init GLUT and create Window
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
    glutInitWindowPosition(100,100);
    glutInitWindowSize(320,320);
    glutCreateWindow("GLUT Tutorial");

    // register callbacks
    glutDisplayFunc(renderScene);

    SetupRC();

    //enter GLUT event processing cycle
    glutMainLoop();

    return 0;
}
```

图 6

（五） 绘制一个基本图元 – 三角形

图元是一组顶点的几何，构成屏幕上所绘制的形状。OpenGL 共有 10 种图元，从空间中简单的一个点到任意数量边的闭合多边形。

绘制图元的一种方式是使用 glBegin 和 glEnd 命令来指定一个图元的顶点，glBegin 和 glEnd 之间放置顶点坐标。

如：绘制两个**点**：

```
glBegin（GL_POINTS）;
    glVertex3f(0.0f,0.0f,0.0f);
    glVertex3f(50.0f,50.0f,50.0f);
glEnd();
```

绘制**直线**：

```
glBegin（GL_LINES）;
    glVertex3f(0.0f,0.0f,0.0f);
    glVertex3f(50.0f,50.0f,50.0f);
glEnd();
```

绘制**三角形**：

```
glBegin(GL_TRIANGLES);
    glVertex3f(-50.0,-50.0,0.0);
    glVertex3f(50.0,0.0,0.0);
    glVertex3f(0.0,50,0.0);
glEnd();
```

绘制**任意多边形**：

```
glBegin(GL_POLYGON);
    glVertex3f(…);
    glVertex3f(…);
    …
    glVertex3f(…);
glEnd();
```

接下来我们已有框架基础上绘制一个三角形。在修改 renderScene，加入如下代码：

```
// Called to draw scene
void renderScene(void)
{
    // Clear the window with current clearing color
    glClear(GL_COLOR_BUFFER_BIT);

    // Draw triangle
    glBegin(GL_TRIANGLES);
        glVertex3f(-1.0,-1.0,0.0);
        glVertex3f(1.0,0.0,0.0);
        glVertex3f(0.0,1.0,0.0);
    glEnd();

    // Flush drawing commands
    glFlush();
}
```

图 7

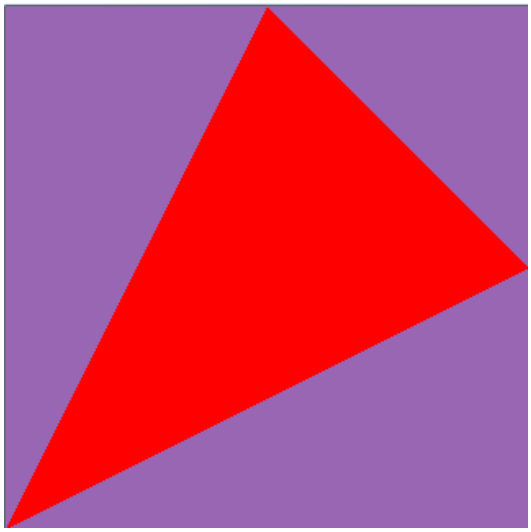
绘制三角形需要指定其顶点坐标，由glVertex3f实现。顶点坐标指定完后放置在glBegin(GL_TRIANGLES)和glEnd()内。

在 SetupRC 函数中用 glColor3f 设置图形颜色。

```
// Setup the rendering state
void SetupRC(void)
{
    glClearColor(0.6f, 0.4f, 0.7f, 1.0f);
    glColor3f(1.0f,0.0f,0.0f);
}
```

图 8

如果你运行代码，将会得到一个控制台窗口，和一个画着一个红色三角形的 OpenGL 窗口，出现在你设置的位置，并有着你设置的尺寸。



（六） glut 改变窗口大小

上一节中程序正确运行后将看到两个窗口：一个控制台窗口（命令行窗口），一个 OpenGL 窗口。当改变窗口大小使高度与宽度不再相等时，这时三角形发生变形。这是因为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。

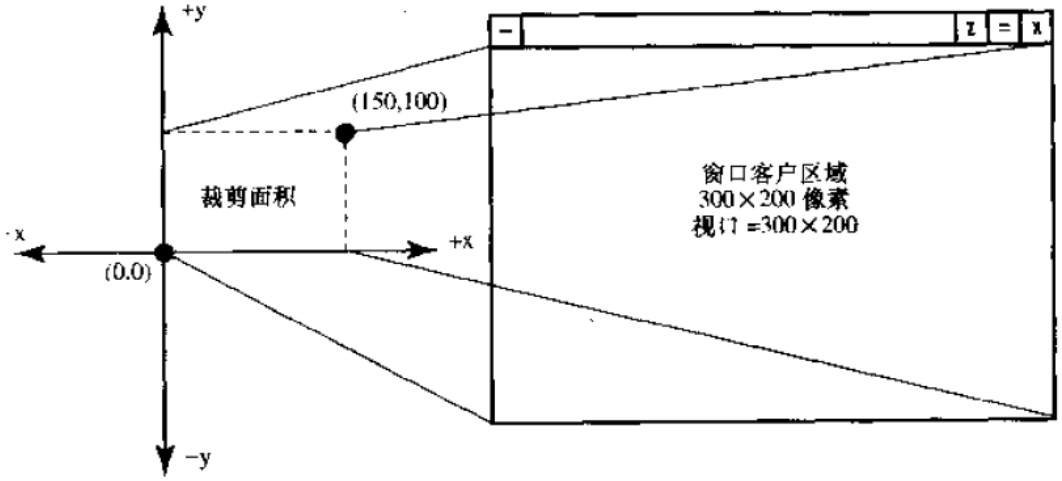


图 1.24　视口被定义为裁剪区域大小的两倍

GLUT 提供了一个回调接口给窗体大小改变事件。此外,该函数在窗体初始化创建的时候也会被调用,所以即便你初始化的窗体不是正方形看上去也不会有问题。原型如下:

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

参数：

func: 指向的函数名，就是该接口要绑定哪个函数作为窗体大小改变事件的响应函数。

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。我们要做的是在 main 函数中添加这个实现代码：

```
// Main program entry point
int main(int argc, char *argv[])
{
    // Init GLUT and create window
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
    glutInitWindowPosition(100,100);
    glutInitWindowSize(320,320);
    glutCreateWindow("GLUT Tutorial");

    // Register callbacks
    glutDisplayFunc(renderScene);
    glutReshapeFunc(changeSize);

    // Setup the rendering state
    SetupRC();

    // Enter GLUT event processing cycle
    glutMainLoop();

    return 0;
}
```

图 9

定义一个负责修改窗口尺寸的函数 changeSize，其实现有两种方式

方式一：通过正投影调整剪裁区域，代码如下：

```
void changeSize(int w, int h)
{
    GLfloat aspectRatio;

    // 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;

    // 设置视窗大小为整个窗口大小
    glViewport(0, 0, w, h);

    // 重置坐标系统
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    // 建立剪裁区域（左、右、底、顶、近、远）
    aspectRatio = (GLfloat)w / (GLfloat)h;

    if (w <= h)
        glOrtho(-1.0, 1.0, -1.0 / aspectRatio, 1.0 / aspectRatio, 1.0, -1.0);
    else
        glOrtho(-1.0 * aspectRatio, 1.0 * aspectRatio, -1.0, 1.0, 1.0, -1.0);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}
```

图 10

方式二：通过透视投影调整，代码如下：

```
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();

    // 设置视窗大小为整个窗口大小
    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(0.0,0.0,2.0, 0.0,0.0,-1.0,0.0f,1.0f,0.0f);
}
```

图 11

changeSize 函数详解：

1) 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；
2) 接着，我们设置当前矩阵为投影矩阵 glMatrixMode(GL_PROJECTION)，这个矩阵定义了一个可视空间（viewing volume）。 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。你可以试一下不同的值,看出来什么效果。注意坐标是和客户区域有关，而不是屏幕。如果你用不同值来试，记得要用新的宽高来计算比例。
3) gluPerspective 函数是其他 OpenGL 库（GLU 库）里的一个函数。gluPerspective 函数包含创建视觉的参数。第一个参数定义了可视角度域的 yz 平面。ratio 参数定义了视口的宽高比关系。后面两个参数定义远近的切面。介于最近值和最远值之间的所有东西都会从场景中减掉。设置了以上参数后,你就会什么都看不见了。
4) 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。

（七） 一个简单的动画

到现在为止， 我们有了一个画着一个红色三角形的 OpenGL 窗口，但一点也不激动人心。现在让我们在这节教程里，学习利用 GLUT 实现一个简单的动画：让三角形自己旋转。

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(**GLUT_DOUBLE**|GLUT_RGB);

双缓冲模式作为显示模式提供两种显示缓冲。当前正在显示的是前台缓冲，另外的是后台缓冲，后台缓冲在后台按设定绘图。当我们发送切换缓冲的绘制命令时(例如当驱动完成时，前后缓冲会切换)，然后下一帧会取代并渲染到屏幕。

glutSwapBuffers 函数促发前后缓冲的切换,就是把后台绘制好的缓冲图像绘制到屏幕，原型如下:

void **glutSwapBuffers**();

接着我们要做的是告诉 GLUT，每间隔一定时间绘制一帧，这样可以产生连续的动画。GLUT 提供了一个函数：glutTimerFunc 来让你注册一个回调函数方便地设置一个简单的动画序列：

void **glutTimerFunc**(unsigned int msecs, void(*func)(void), int value);

参数：

- msecs – GLUT 等待 msecs 毫秒；
- func –计时器函数（msecs 后被调用的函数的函数名）；
- value – 用户定义的值。

glutTimerFunc 只会触发一次，为了产生连续的动画，必须在计时器调用函数中重新设置计时器。计时器函数如下：

```
void TimerFunction(int value){
    angle += 2.0f;
    // Redraw the scene with new coordinates
    glutPostRedisplay();
    glutTimerFunc(23,TimerFunction, 1);
}
```

图 12

下面看看现在的 main 函数：

```
// Main program entry point
int main(int argc, char* argv[])
{
    // init GLUT and create window
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowPosition(100,100);
    glutInitWindowSize(320,320);
    glutCreateWindow("Rotate Triangle");

    // register callbacks
    glutDisplayFunc(renderScene);
    glutReshapeFunc(changeSize);
    glutTimerFunc(23, TimerFunction, 1);

    // Setup the rendering state
    SetupRC();

    // enter GLUT event processing cycle
    glutMainLoop();

    return 0;
}
```

图 13

下面就是设置渲染函数 renderScene。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：

```
GLfloat angle = 0.0;


void renderScene(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glPushMatrix();
    glRotatef(angle, 0.0f, 1.0f, 0.0f);

    glColor3f(1.0, 0.0, 0.0);
    glBegin(GL_TRIANGLES);
        glVertex3f(-1.0, -1.0, 0.0);
        glVertex3f(1.0, 0.0, 0.0);
        glVertex3f(0.0, 1.0, 0.0f);
    glEnd();

    glPopMatrix();

    glutSwapBuffers();
}
```

图 14

注意：旋转角度在计时器函数 TimerFunction 中增加（如图 12）。

（八） 键盘控制

GLUT 允许应用程序自动检测键盘输入，包括了普通键和其他特殊键（如 F1,UP）。本节我们来讨论怎样监测按键事件和如何响应，即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。

目前为止我们学习到了，需要 GLUT 处理对应的事件,必须先告知 GLUT 把事件绑定到指定函数。之前已经介绍了重绘事件和窗体更改大小事件，当窗口重绘时我们想调用哪个渲染函数。和当窗口大小改变时，哪个函数又将被调用。同样的,下面来介绍键盘事件，我们要告知 GLUT 哪个函数是响应按键处理的。我们同样是调用一个函数注册相关的回调函数。

1. 普通按键

glutKeyboardFunc 函数

当你按下一个键后，GLUT 提供了两个函数为这个键盘时间注册回调函数。第一个是 glutKeyboardFunc。用于告知窗体系统处理普通按键事件。**普通按键**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。

glutKeyboarFunc 绑定的函数必须返回三个结果值。第一个是按键对应的 ASCII 内码,其余两个是按钮触发时鼠标所在的位置。鼠标位置是相对于窗体客户端的左上角。

我们来看一个例子。当用户输入 esc 键的时候退出程序。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 esc 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);
}
```

图 15

2. 特殊按键

glutSpecialFunc 函数

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;
    }
}
```

图 16

常量 GLUT_KEY_*在 glut.h 头文件中预定义的。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	Right function key
GLUT_KEY_UP	Up function key
GLUT_KEY_DOWN	Down function key
GLUT_KEY_PAGE_UP	Page function key
GLUT_KEY_PAGE_DOWN	Page function key
GLUT_KEY_HOME	Home function key
GLUT_KEY_END	End function key
GLUT_KEY_INSERT	Insert function key

为配合自定义响应函数 processSpecialKeys，我们添加红绿蓝变量到代码头部。除此之外，我们要更改 renderScene 函数来达到渲染效果。

```
#include <glut.h>
#include <stdlib.h>

GLfloat angle = 0.0;
GLfloat red = 1.0f, green = 1.0f, blue = 1.0f;

void renderScene(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glPushMatrix();
    glRotatef(angle, 0.0f, 1.0f, 0.0f);

    glColor3f(red, green, blue);
    glBegin(GL_TRIANGLES);
        glVertex3f(-1.0, -1.0, 0.0);
        glVertex3f(1.0, 0.0, 0.0);
        glVertex3f(0.0, 1.0, 0.0f);
    glEnd();

    glPopMatrix();

    glutSwapBuffers();
}
```

图 17

现在已经定义好 glutKeyboardFunc 和 glutSpecialFunc 函数的代码.

接下来需要告诉 GLUT 键盘事件的响应函数（哪个函数负责处理键盘事件），该函数在每次键盘被按下的时候被 GLUT 调用。下面是加了键盘处理的 main 函数:

```
// Main program entry point
int main(int argc, char* argv[])
{
    // init GLUT and create window
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowPosition(100,100);
    glutInitWindowSize(320,320);
    glutCreateWindow("Rotate Triangle");

    // register callbacks
    glutDisplayFunc(renderScene);
    glutReshapeFunc(changeSize);
    glutTimerFunc(23, TimerFunction, 1);

    // here are the new entries
    glutKeyboardFunc(processNormalKeys);
    glutSpecialFunc(processSpecialKeys);

    // Setup the rendering state
    SetupRC();

    // enter GLUT event processing cycle
    glutMainLoop();

    return 0;
}
```

图 18

（九） 实现在窗口内运动的正方形

接下来综合运用前面知识，实现一个在窗口内匀速运动的正方形，在碰到窗口边界时实现反弹（速度方向，大小不变）。

1. 主函数

```
////////////////////////////////////////////////////////////////
// Main program entry point
int main(int argc, char* argv[])
    {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(800,600);
        glutCreateWindow("Bounce");
    glutDisplayFunc(RenderScene);
        glutReshapeFunc(ChangeSize);
    glutTimerFunc(33, TimerFunction, 1);

    SetupRC();

    glutMainLoop();

        return 0;
    }
```

图 19

2. 全局变量

```
// Initial square position and size
GLfloat x = 0.0f;
GLfloat y = 0.0f;
GLfloat rsize = 25;

// Step size in x and y directions
// (number of pixels to move each time)
GLfloat xstep = 1.0f;
GLfloat ystep = 1.0f;

// Keep track of windows changing width and height
GLfloat windowWidth;
GLfloat windowHeight;
```

图 20

3. 渲染函数

```
////////////////////////////////////////////////////////////////
// Called to draw scene
void RenderScene(void)
    {
    // Clear the window with current clearing color
    glClear(GL_COLOR_BUFFER_BIT);

    // Set current drawing color to red
    //         R     G     B
    glColor3f(1.0f, 0.0f, 0.0f);

    // Draw a filled rectangle with current color
    glRectf(x, y, x + rsize, y - rsize);

    // Flush drawing commands and swap
    glutSwapBuffers();
    }
```

图 21

```
////////////////////////////////////////////////////////////////
// Setup the rendering state
void SetupRC(void)
    {
    // Set clear color to blue
    glClearColor(0.0f, 0.0f, 1.0f, 1.0f);
    }
```

图 22

4. 计时函数

```
// Called by GLUT library when idle (window not being
// resized or moved)
void TimerFunction(int value)
    {
    // Reverse direction when you reach left or right edge
    if(x > windowWidth-rsize || x < -windowWidth)
        xstep = -xstep;

    // Reverse direction when you reach top or bottom edge
    if(y > windowHeight || y < -windowHeight + rsize)
        ystep = -ystep;

    // Actually move the square
    x += xstep;
    y += ystep;

    // Check bounds. This is in case the window is made
    // smaller while the rectangle is bouncing and the
    // rectangle suddenly finds itself outside the new
    // clipping volume
    if(x > (windowWidth-rsize + xstep))
        x = windowWidth-rsize-1;
    else if(x < -(windowWidth + xstep))
        x = -windowWidth -1;

    if(y > (windowHeight + ystep))
        y = windowHeight-1;
    else if(y < -(windowHeight - rsize + ystep))
        y = -windowHeight + rsize - 1;

     // Redraw the scene with new coordinates
    glutPostRedisplay();
    glutTimerFunc(33,TimerFunction, 1);
    }
```

图 23

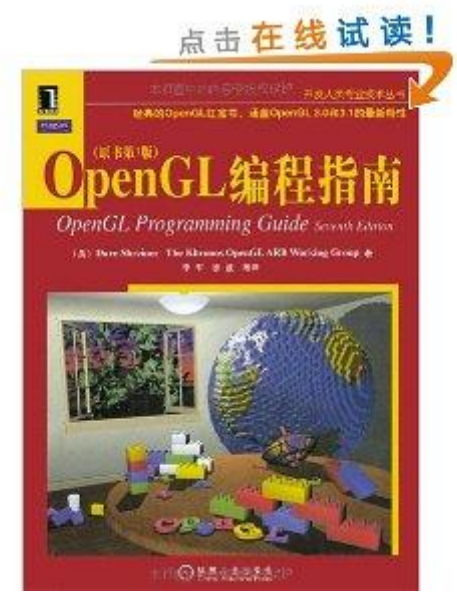
5. 窗口响应函数

```
// Called by GLUT library when the window has changed size
void ChangeSize(int w, int h)
    {
    GLfloat aspectRatio;

    // Prevent a divide by zero
    if(h == 0)
        h = 1;

    // Set Viewport to window dimensions
    glViewport(0, 0, w, h);

    // Reset coordinate system
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    // Establish clipping volume (left, right, bottom, top, near, far)
    aspectRatio = (GLfloat)w / (GLfloat)h;
    if (w <= h)
        {
        windowWidth = 100;
        windowHeight = 100 / aspectRatio;
        glOrtho (-100.0, 100.0, -windowHeight, windowHeight, 1.0, -1.0);
        }
    else
        {
        windowWidth = 100 * aspectRatio;
        windowHeight = 100;
        glOrtho (-windowWidth, windowWidth, -100.0, 100.0, 1.0, -1.0);
        }

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    }
```

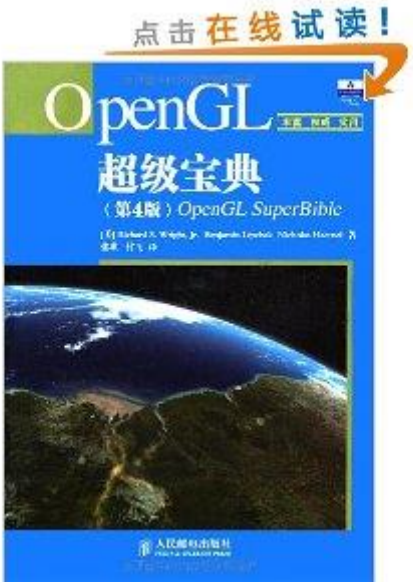
图 24

参考图书

1. 《OpenGL 编程指南》（俗称“红宝书 redbook”）第一到第二章



2. 《OpenGL 超级宝典》（俗称“蓝宝书 bluebook”）第一到第四章



三、实验内容

1. 设置编程环境，使用 GLUT；
2. 编写第一个程序:绘制一个三角形；
3. 实现可变窗口大小；
4. 生成一个简单动画，三角形绕固定轴匀速旋转；
5. 键盘控制三角形颜色和旋转速度；
6. 实现一个反弹运动的正方形的动画；

7. 仿照 6),生成一个可以反弹运动的三角形动画，键盘控制三角形移动速度、颜色。

四、实验步骤

1. 明确项目需求；
2. 编写代码；
3. 编译代码；
4. 测试程序；
5. 根据测试结果对程序进行调试改进。