# 实验1 OpenGL与基本图形生成算法

## 一、实验目的与要求：

1. 掌握OpenGL的GLUT图形编程框架及绘制点、线、多边形等基本图形的方法；
2. 熟悉基于OpenGL绘制简单的图案
3. 掌握直线的扫描转换原理并程序实现方法；

## 二、实验内容与步骤：

### 1 OpenGL介绍

OpenGL（Open Graphics Library，开放图形库或者“开放式图形库”）是用于渲染2D、3D矢量图形的跨语言、跨平台的应用程序编程接口（API）。这个接口由近350个不同的函数调用组成，用来从简单的图形比特绘制复杂的三维景象。而另一种程序接口系统是仅用于Microsoft Windows上的Direct3D。OpenGL常用于CAD、虚拟实境、科学可视化程序和电子游戏开发。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Opengl32.lib <gl/gl.h> | 核心函数库 | gl | 115 | 最基本的绘制命令 |
| Glu32.lib  <gl/glu.h> | 实用库 | glu | 43 | 管理坐标变换、多边形镶嵌、绘制Nurbs曲线、曲面以及处理错误等 |
| Glaux.lib  <gl/glaux.h> | 辅助库 | aux | 31 | 提供窗口管理和消息响应函数及简单模型的制作 |
|  | Windows专用库 | wgl | 6 | 管理显示列表、字体位图、绘图描述表及扩展函数等 |

OpenGL的基本库是OpenGL的核心函数库，在这个函数库中，提供了115个函数，这些函数都是以“gl”为前缀。这类API的主要功能包括物体描述、平移、旋转、缩放、光照、纹理、材质、像素、位图、文字处理等。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程序完美的可移植性。

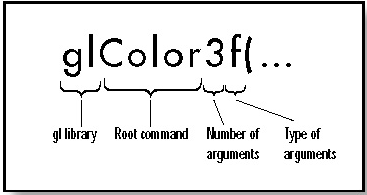
OpenGL的实用库是OpenGL基本库的一套子程序，它提供了43个函数，这些函数都是以”glu”为前缀。基本的OpenGL不支持传统上同图形标准相关的一些几何对象，为了减少一些编程负担，OpenGL提供了实用库。主要功能包括绘制二次曲面、NURBS曲线曲面、复杂多边形以及纹理、矩阵管理等。实用库中的所有函数全都是由OpenGL基本库函数来编写的，所以，在使用上和OpenGL基本库的使用是完全相同的，而且，用户也可以使用基本函数库来实现实用库的函数功能。

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。它提供了31个函数，这些函数都是以“aux”为前缀。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等，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。

GLUT代表OpenGL应用工具包，英文全称为OpenGL Utility Toolkit，是一个和窗口系统无关的软件包，它由Mark Kilgard在SGI时写的。作为AUX库的功能更强大的替代品，用于隐藏不同窗口系统API的复杂性。GLUT是专为构建中小型OpenGL程序。GLUT简单易学，是适合学习OpenGL和开发简单的OpenGL应用程序，但GLUT并不是一个功能全面的工具包所以大型应用程序需要复杂的用户界面最好使用本机窗口系统工具包。

OpenGL语法规则：

* 核心库常量以GL开头，均大写，如：GL\_COLOR\_MATERIAL
* <库前缀><根命令><可选的参数个数><可选的参数类型>



* OpenGL是一种直接模式的API,每条命令根据当前的渲染状态都会产生某种立即效果，渲染状态即各种标记，它们指出哪些特性是打开或关闭的。OpenGL的绘图方式是由一系列的状态决定的，如果设置了一种状态或模式而不改变它，OpenGL在绘图过程中将一直保持这种状态或模式。

#### 1.1 Visual Studio中的OpenGL环境设置

Visual Studio底层已经集成了OpenGL，仅需手动设置好glut库即可（glut文件可见群共享文件中的“OpenGL的glut库所需文件.rar”），其方法是：

Step 1.把glut.h放到Microsoft Visual Studio\VC98\Include\GL文件夹.

Step 2.把glut.lib和glut32.lib放到Microsoft Visual Studio\VC98\ lib文件夹

Step 3.把glut.dll和glut32.dll放到操作系统目录下面的system32文件夹内（典型的位置为：C:\Windows\System32）.(如果是win 7 64位操作系统，则是C:\WINDOWS\SysWOW64这个文件下)。

#### 1.2 搭建OpenGL程序框架

1）创建一个空的Win32 Console Application工程，添加源程序如example.cpp；

2）在源程序中添加 #include <gl/glut.h>

在int main(int argc, char\* argv[])函数中增加OpenGL窗口管理的函数语句：

* 初始化窗口

对GLUT进行初始化，这个函数必须在其它的GLUT使用之前调用一次

*glutInit(&argc, argv);*

* 设定窗口的显示模式（ glutInitDisplayMode）

设置显示方式，其中GLUT\_RGB表示使用RGB颜色，与之对应的还有GLUT\_INDEX（表示使用索引颜色）。GLUT\_SINGLE表示使用单缓冲，与之对应的还有GLUT\_DOUBLE（使用双缓冲）

*glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);*

* 指定窗口的位置和大小

glutInitWindowPosition (100,120)设置窗口在屏幕中的位置

glutInitWindowSize (400,300);设置窗口的大小

* 创建窗口

glutCreateWindow ("窗口标题")

根据前面设置的信息创建窗口。参数将被作为窗口的标题。注意：窗口被创建后，并不立即显示到屏幕上。需要调用glutMainLoop( )才能看到窗口

* 指定窗口的显示内容函数

glutDisplayFunc(*Display*);

设置绘制回调函数，当需要进行画图时，这个函数就会被调用。例如，绘制函数可为：

void Display(void){

glClear(GL\_COLOR\_BUFFER\_BIT); //用当前背景色填充窗口

glColor3f(1.0f, 0.0f, 0.0f); //设置当前的绘图颜色为红色

glRectf(50.0f, 100.0f, 150.0f, 50.0f); //绘制一个矩形

glFlush(); //处理所有的OpenGL程序

}

* 初始化OpenGL绘图场景

glClearColor(1.0f, 1.0f, 1.0f, 1.0f); //设置窗口背景颜色为白色

glMatrixMode(GL\_PROJECTION); //设置投影参数

gluOrtho2D(0.0,200.0,0.0,150.0); //定义屏幕区域对应的模型坐标范围

* 启动事件循环

glutMainLoop(); //启动主GLUT事件处理循环

进行一个消息循环 .这个函数可以显示窗口，并且等待窗口

#### 1.3 一个简单的OpenGL程序框架

例如搭建一个OpenGL程序框架example.cpp源程序如下：

#include <gl/glut.h>

#include <math.h>

// 初始化OpenGL场景

void Initial( ){

glClearColor(1.0f, 1.0f, 1.0f, 1.0f); //设置窗口背景颜色为白色

glMatrixMode(GL\_PROJECTION); //设置投影参数

gluOrtho2D(-400.0, 400.0, -300, 300.0); // 投影面上的模型坐标范围

}

// 显示图形

void Display(void){

glClear(GL\_COLOR\_BUFFER\_BIT); //用当前背景色填充窗口

**// 此处需增加调用基本图形生成函数**

**// 例如:**

***glColor3f(1.0f, 0.0f, 0.0f);*** *//设置当前的绘图颜色为红色*

***glRectf(50.0f, 100.0f, 150.0f, 50.0f);*** *//绘制一个矩形*

glFlush(); //处理所有的OpenGL程序

}

int main(int argc, char\* argv[ ]){

glutInit(&argc, argv); // glut初始化

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);//初始窗口显示模式

glutInitWindowSize(800, 600); //设置窗口的尺寸

glutInitWindowPosition(200, 200); //设置窗口的位置

glutCreateWindow("基本图形生成"); //创建一个窗口

glutDisplayFunc(Display); //设置当前窗口的显示回调函数

Initial(); //完成窗口初始化

glutMainLoop(); //启动主GLUT事件处理循环

return 0;

}

搭好框架后，可根据基本图形生成算法的原理和伪代码，编写直线、圆、椭圆、多边形等的扫描转换程序，在Display函数中调用这些生成函数实现基本图形的扫描转换及绘制。

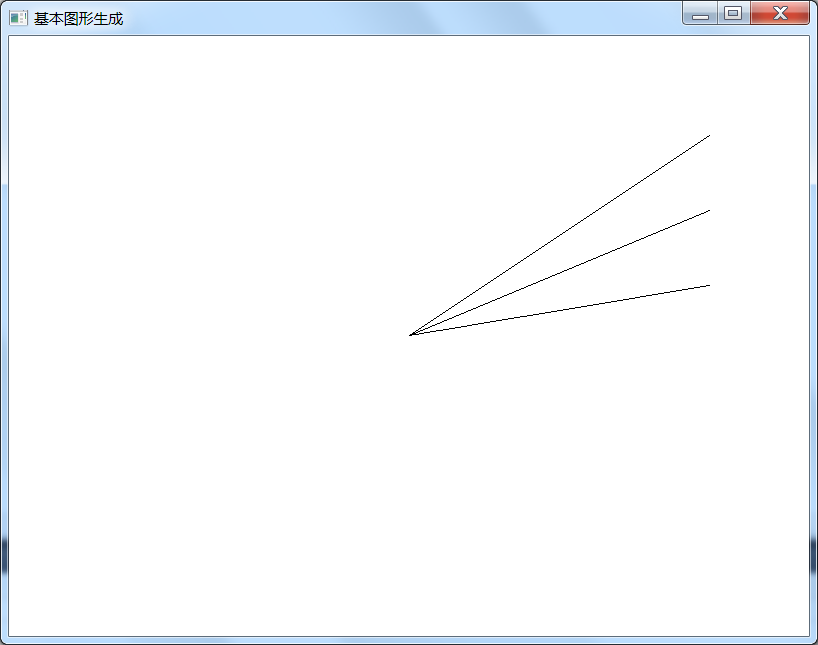
*// 例如：此处需增加调用基本图形生成函数*

*DDAline(0, 0, 300, 50);*

*MidBhline(0, 0, 300, 125);*

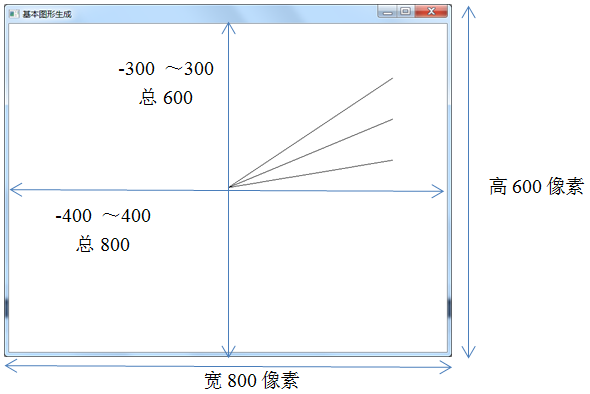
*Bhline(0, 0, 300, 200);*

绘制效果如下图：



具体程序代码见：

要注意窗口的尺寸大小要与OpenGL投影面设置的坐标范围尺寸一致，以保证OpenGL画点的整数坐标与像素对应：



#### 1.4 OpenGL图形绘制

注意：以下仅列出常见的OpenGL图形绘制函数，更详细内容可见群共享文件的《OpenGL编程指南》和《OpenGL超级宝典》。

1）顶点绘制

* 顶点坐标
  + OpenGL采用有序排列的顶点集合来构造几何图元。 如：

glVertex2i(2, 5); //整数定义的二维坐标

glVertex3f(2.0f, 5.0f, 7.0f); //浮点定义的三维坐标

* 顶点关系
  + 在OpenGL中，同一个几何图元的所有被定义的顶点一起放在glBegin()和glEnd()函数之间，同时定义这些顶点之间的关系。如：

glBegin(GL\_POLYGON);

glVertex2i(0,0); glVertex2i(0,11);

glVertex2i(11,14); glVertex2i(14,7);

glEnd();

glBegin中的绘制模式主要有：

* GL\_POINTS 单个顶点集
* GL\_LINES 多组双顶点线段
* GL\_LINE\_LOOP 闭合折线
* GL\_LINE\_STRIP 不闭合折线
* GL\_TRAINGLES 多组独立填充三角形
* GL\_TRAINGLE\_STRIP 线型连续填充三角形串
* GL\_TRAINGLE\_FAN 扇形连续填充三角形串
* GL\_QUADS 多组独立填充四边形
* GL\_QUAD\_STRIP 连续填充四边形串
* GL\_POLYGON 单个简单填充凸多边形

2）OpenGL绘制点

* 点的绘制

glBegin(GL\_POINTS)；

glVertex3f(0.0f, 0.0f, 0.0f);

glVertex3f(10.0f, 0.0f, 0.0f);

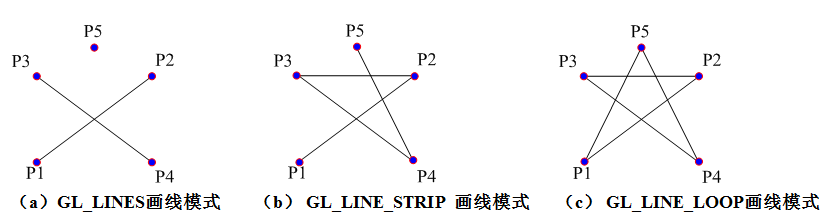
glEnd()；

* 点的属性（大小）

void glPointSize(GLfloat size)；

3）OpenGL绘制线

* 直线的绘制模式
  + GL\_LINES
  + GL\_LINE\_STRIP
  + GL\_LINE\_LOOP

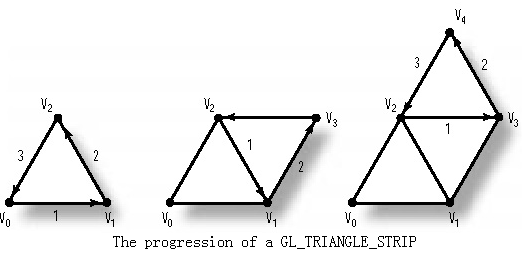
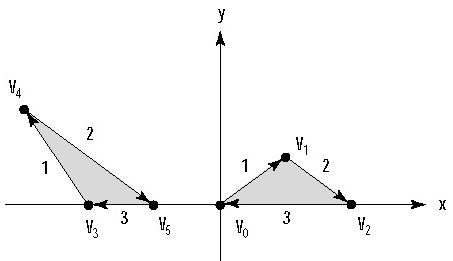


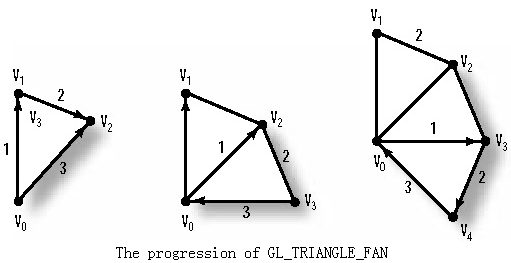
* 直线的属性：线宽

void glLineWidth(GLfloat width)

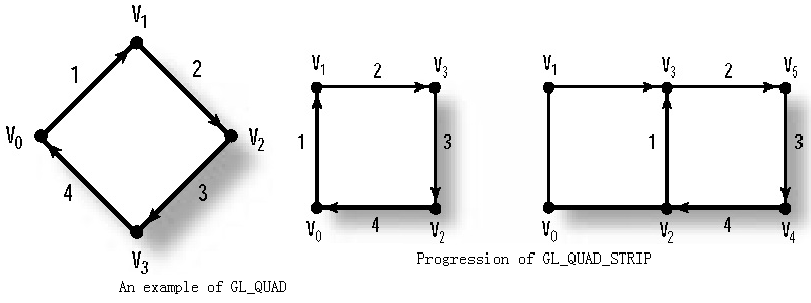
4）OpenGL绘制面

* 三角形面的绘制
  + GL\_TRIANGLES
  + GL\_TRIANGLE\_STRIP
  + GL\_TRIANGLE\_FAN

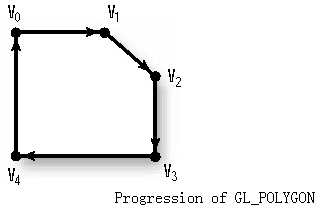




* 四边形面的绘制
  + GL\_QUADS
  + GL\_QUAD\_STRIP



* 多边形面的绘制（GL\_POLYGON）



* 多边形面的绘制规则
  + 所有多边形都必须是平面的。
  + 多边形的边缘决不能相交，而且多边形必须是凸的。

解决：对于非凸多边形，可以把它分割成几个凸多边形（通常是三角形），再将它绘制出来。

* 多边形面的正反属性（绕法）：指定顶点时顺序和方向的组合称为“绕法”。绕法是任何多边形图元的一个重要特性。一般默认情况下，OpenGL认为逆时针绕法的多边形是正对着的。

glFrontFace(GL\_CW)；

glFrontFace(GL\_CCW)；

* 多边形面的颜色
  + glShadeModel(GL\_FLAT) 用指定多边形最后一个顶点时的当前颜色作为填充多边形的纯色，唯一例外是GL\_POLYGON图元，它采用的是第一个顶点的颜色。
  + glShadeModel(GL\_SMOOTH) 从各个顶点给三角形投上光滑的阴影，为各个顶点指定的颜色之间进行插值。
* 多边形面的显示模式

glPolygonMode(GLenum face，GLenum mode)；

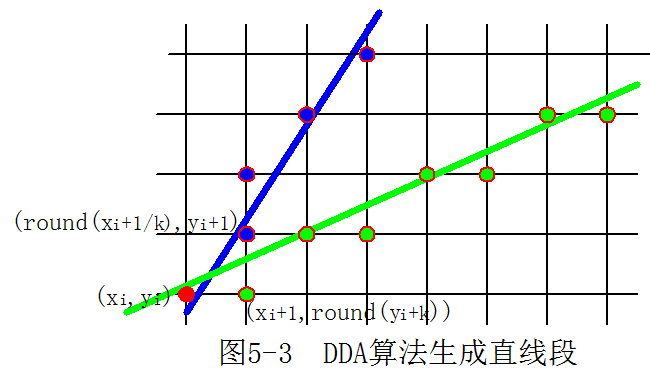
参数face用于指定多边形的哪一个面（GL\_FRONT/GL\_BACK/GL\_FRONT\_AND\_BACK）受到模式改变的影响。

参数mode用于指定新的绘图模式（GL\_POINT/ GL\_LINE/GL\_FILL）。

### 2 直线的扫描转换

注意：为了深刻理解CG原理和方法，后续的直线、圆、椭圆及多边形等的扫描转换**要求通过OpenGL画点的途径实现，不能直接使用OpenGL绘制图元函数**。

#### 2.1 DDA算法



// 数值微分方法绘制直线段

void DDAline(int x0, int y0, int x1, int y1){

int dx=x1-x0, dy=y1-y0, epsl, k; // 横向和纵向上的距离

float x=x0, y=y0, xIncre, yIncre;

if(abs(dx)>abs(dy)) epsl=abs(dx);

else epsl=abs(dy);

xIncre=(float)(dx)/epsl;

yIncre=(float)(dy)/epsl;

glBegin(GL\_POINTS);

for(k=0;k<=epsl;k++){

glVertex2i((int)(x+0.5), (int)(y+0.5)); // 舍入取整

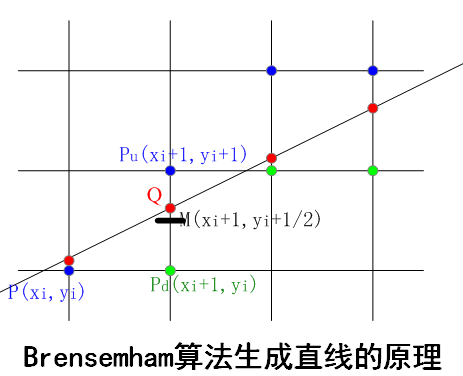
x+= xIncre; y+= yIncre; // 每次叠加一个增量

}

glEnd();

}

#### 2.2 中点BH算法



// 中点Bresenham算法绘制直线段(0≤k≤1)

void MidBhline(int x0, int y0, int x1, int y1){

int dx, dy, d, UpIncre, DownIncre, x, y;

if(x0>x1){

x=x1; x1=x0; x0=x; y=y1; y1=y0; y0=y;

}

x=x0; y=y0; dx=x1-x0; dy=y1-y0; d=dx-2\*dy;

UpIncre=2\*dx-2\*dy;

DownIncre=-2\*dy;

glBegin(GL\_POINTS);

while(x<=x1){

glVertex2i(x, y);

x++;

if(d<0) {

y++;

d+= UpIncre;

}

else

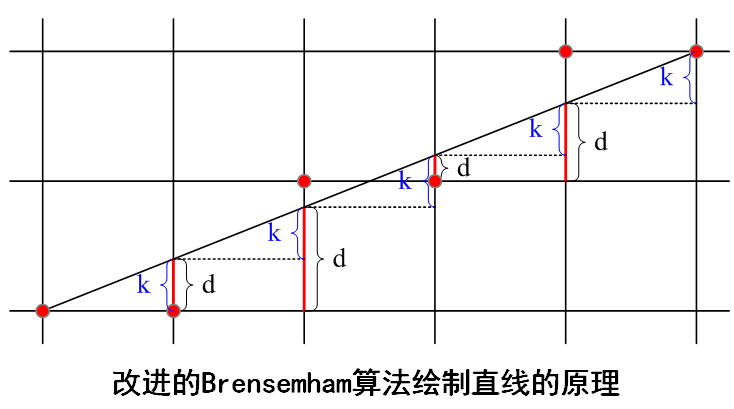
d+= DownIncre;

}

glEnd();

}

#### 2.3 改进的BH算法



// 改进的Bresenham算法绘制直线段(0≤k≤1)

void Bhline(int x0, int y0, int x1, int y1){

int x, y, dx, dy, e;

if(x0>x1){

x=x1; x1=x0; x0=x; y=y1; y1=y0; y0=y;

}

dx=x1-x0; dy=y1-y0; x=x0; y=y0;

e=-dx;

glBegin(GL\_POINTS);

while(x<=x1){

glVertex2i(x, y);

x++;

e=e+2\*dy;

if(e>0) {

y++;

e=e-2\*dx;

}

}

glEnd();

}

## 三、实验练习：

###### 1 使用OpenGL绘制不同属性、颜色和坐标的基本图元对象；

###### 2 验证教材中的OpenGL例子（见群共享文件的“课本中OpenGL程序.rar”）；

###### 3 编写斜率k在[0,1]间的DDA/Bresenham绘制直线程序，并用逐点生成方式在OpenGL中显示。

###### 4 推导并试验中点BH算法对任意斜率的直线扫描转换方法；

###### 5 学习并试验圆和椭圆多边形的扫描转换方法（例如：边界用红色，内部用蓝色填充）