**电 子 科 技 大 学**

**实 验 报 告**

**学生姓名：XXXXX 学 号：XXXX 指导教师：XXXXX**

**实验地点：xxx 实验时间：XX.XX.XX**

**一、实验室名称：** 软件信息技术专业实验室

**二、实验项目名称：**bresenham算法模拟演示

**三、实验学时：** 4

**四、实验原理：**

计算机处理图形的方法是采用管线的方式，主要在GPU中进行。根据管线把图形的处理分为顶点处理阶段和像素处理阶段。其中由顶点投影变换到屏幕像素之后，就需要进行图元生成，最基本的就是直线段的生成以及其他曲线等。如何利用GPU加速来实现图元生成，一直都是图形学发展的基础算法。从函数方法到DDA方法，再到现在的bresenham算法。算法效率越来越高，通过学习这些算法，让学生对计算机处理图形元素（点、线、多边形）等有一个基本的处理技术概念。

在屏幕坐标上实现线段的生成，是一个整数坐标系下的近似采样问题。所产生的线段会产生明显的锯齿状效应，这就是我们通常所说的“走样”的问题。通过过取样和区域取样等方法可以通过控制每个像素点的亮度来实现消除锯齿状效果。

bresenham算法主要的目的是通过一个决策参数实现线段上所有像素的获取工作，从而避免了其他算法所需要的浮点乘法操作以及取整操作，而只需要进行简单的整数相加运算、逻辑判断等就可以进行像素点的生成。

**五、实验目的：**

1. 理解屏幕整数坐标以及基于整数坐标下图元生成的原则与问题
2. 掌握bresenham算法，编程实现
3. 掌握bresenham算法的各种边界条件处理。
4. 分析所产生的走样问题
5. 利用OpenGL实现一个交互式模拟演示

**六、实验内容：**

1. 基于glut的opengl绘制框架实现
2. 编写背景绘制函数void drawbackground()
   1. 绘制整数坐标网格，利用opengl模拟产生一个网格，以一个格子代表一个像素。产生一个20×20的网格，在视点观察方向，垂直于视点方向。
3. 编写bresenham算法函数，void drawline(point1, point2),能够实时动态产生整数坐标点，并绘制一个球替代该像素
4. 编写键盘相应函数，实现对线段两个端点位置的控制。键盘1，2，选择两个点作为当前点；键盘上下左右分别对点的坐标进行位移。

**七、实验器材（设备、元器件）：**

·多媒体计算机，vs2010， opengl库， glut库

**八、实验步骤：**

**步骤1 OpenGL工程项目配置**

拷贝opengl库和glut库，分别保存在GL Files下，对应dll目录是动态库以及相关纹理，include是头文件包含目录， lib是所需的库文件目录。

在vs中打开nehe例子中的第一个例子，会自动转换，从vc6的工程转换为vs2010的解决方案和工程，这些步骤都采用缺省设置。

在项目工程中，设置属性。针对vc++目录，分别设置头文件目录，加入GL Files下的include，以及库目录，加入GL Files下的lib。在工程的调试选项下，选择工作路径为GL Files的dll。具体操作如下图所示

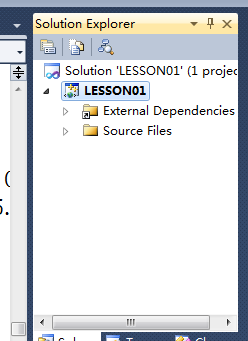


图1 转换后的vs2010工程

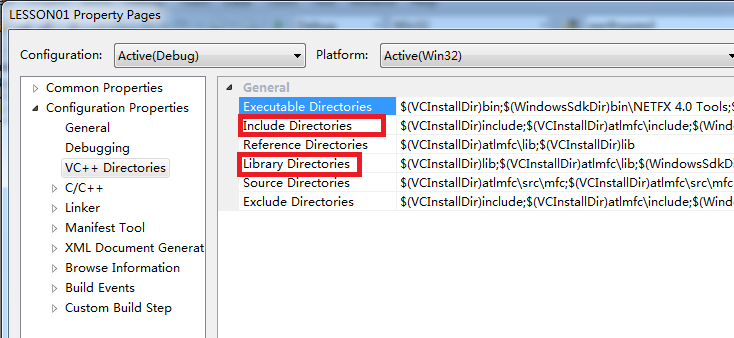


图2 配置头文件和库文件路径

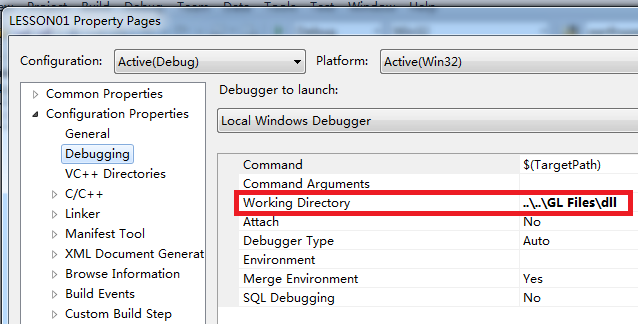


图3 配置调试工作路径

上述步骤之后，就可以编译项目，并执行。由于nehe的第一个例子什么也没有绘制，render函数中没有绘制任何内容，只有黑色的屏幕，所以结果是一个窗口，中间工作区为黑色。

**步骤2 glut框架调整**

调整nehe中glut框架的内容。在mian函数中删除game mode的对话框以及相应的全屏设置。结果代码如下

int main(int argc, char\*\* argv)

{

glutInit(&argc, argv); // GLUT Initializtion

glutInitDisplayMode(GLUT\_RGB | GLUT\_DOUBLE); // Display Mode (Rgb And Double Buffered)

glutInitWindowSize(500, 500); // Window Size If We Start In Windowed Mode

glutCreateWindow("My shiyan1"); // Window Title

init(); // Our Initialization

glutDisplayFunc(render); // Register The Display Function

glutIdleFunc(render);

glutReshapeFunc(reshape); // Register The Reshape Handler

glutKeyboardFunc(keyboard); // Register The Keyboard Handler

glutSpecialFunc(special\_keys); // Register Special Keys Handler

glutMainLoop(); // Go To GLUT Main Loop

return 0;

}

glut是创建OpenGL绘制窗口，管理各种消息响应回调的简单框架系统。具体函数说明如下

glutInit(&argc, argv);GLUT框架的初始化

glutInitDisplayMode(GLUT\_RGB | GLUT\_DOUBLE);是显示模式的设置

glutInitWindowSize(500, 500); 初始化OpenGL绘制窗口大小是500×500像素。

glutCreateWindow("My shiyan1");创建绘制窗口，并设置title为引号内的内容。

窗口设置好之后，下面就分别就窗口中的各种消息回调函数进行设置，这些函数需要在main函数前进行定义或者声明。

glutDisplayFunc(render); //设置主显示函数为render（）函数。

glutIdleFunc(render); //设置窗口Idle消息的时候调用render（）函数。

glutReshapeFunc(reshape) //设置窗口大小改变时，调用函数reshape（）函数。

glutKeyboardFunc(keyboard); //设置一般键盘和鼠标消息的时候调用keyboard（）函数。

glutSpecialFunc(special\_keys);//设置特殊功能键消息的时候调用special\_keys（）函数。

glutMainLoop(); //所有内容设置好之后，系统进入主循环中。

**步骤3 主循环代码编写**

前面glut消息设置中，设置主循环函数是render（）。

在该render（）函数中需要包括基本的OpenGL绘制框架，首先是清空，最后是切换buffer。中间才是我们需要绘制的内容。

void render(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT); // Clear Screen And Depth Buffer

glLoadIdentity();

glTranslatef(0,0,-5.0f);// Reset The Current Modelview Matrix

drawbackground();

bresenham(10,10,p2x,p2y);

// Swap The Buffers To Become Our Rendering Visible

glutSwapBuffers ( );

}

其中清空部分的函数为

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);//清空颜色buffer和深度buffer

glLoadIdentity() //清空矩阵

glutSwapBuffers ( );//绘制后的切换前后台buffer。

然后我们要在OpenGL三维环境中创建并绘制虚拟的屏幕网格坐标系，采用51根横线和51根水平线构成50×50的一个网格。然后利用bresenham算法动态生成各种线段。

**步骤4 绘制背景网格函数**

采用函数drawbackground来绘制背景网格。背景网格的计算应该根据视矩设置来确定。在本例中，视点设置为透视投影的方式，并且投影角度是60度，宽高比是1：1，近平面和原平面分别是1.0和5000.0。函数如下：

gluPerspective(60, (float)w/(float)h, 1.0, 5000.0);

这样网格在z=-5的平面，根据视角是60度，针对窗口是500×500情况，可以计算出铺满屏幕的xy坐标范围是5.6，也就是（-2.8～2.8）。

60度

z=-5

视点

图4 网格模型坐标计算

这样51根水平线和51根垂直线的绘制函数如下。

void drawbackground()

{

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

glBegin(GL\_LINES);

for(float xx=-2.8;xx<=2.8;xx+=step)

{

glVertex3f(xx,-2.8f,0);

glVertex3f(xx,2.8f,0);

}

for(float yy=-2.8;yy<=2.8;yy+=step)

{

glVertex3f(-2.8f,yy,0);

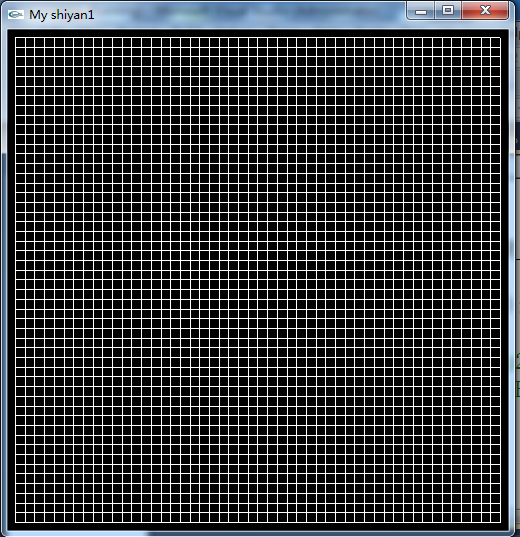
glVertex3f(2.8f,yy,0);

}

glEnd();

}

具体结果是



**步骤5 计算并绘制bresenham函数**

bresenham算法是利用一个决策参数来判断，每个像素位置。其中按照dx和dy的比较，然后确定是以x方向还是y方向取样，再利用对应的决策参数计算，判断另一个坐标的取值。在程序中会遇到各种情况组合，因此需要进行所有的边界判断。具体代码如下

void bresenham(int X1, int Y1, int X2, int Y2)

{

float x,y;

float xx1,yy1,xx2,yy2;

if(X1>X2)

{

xx1=X2;xx2=X1;

yy1=Y2;yy2=Y1;

}

else

{

xx1=X1;xx2=X2;

yy1=Y1;yy2=Y2;

}

float dx = fabs(xx2 - xx1), dy = fabs(yy2 - yy1);

float px = 2 \* dy - dx;//当斜率小于1时

float py = 2 \* dx - dy;//当斜率大于1时

x=xx1;y=yy1;

glPointSize(8.0);

if (dx>=dy)//当斜率绝对值小于1时

{

glBegin(GL\_POINTS);

while (x <= xx2)

{

drawpixel(x,y);

if (px < 0)

{

px += 2 \* dy;

}

else

{

px += 2 \* (dy - dx);

if(yy2>=yy1)

y++;

else

y--;

}

x++;

}

glEnd();

}else

{ //dx<dy

glBegin(GL\_POINTS);

while(inYrange(y,yy1,yy2))

{

drawpixel(x,y);

if (py < 0)

{

py += 2 \* dx;

}

else

{

py += 2 \* (dx - dy);

x++;

}

if(yy2>=yy1)

y++;

else

y--;

}

glEnd();

}

}

其中X2>X1，dx>=dy的情况是书上所讲述的公式推导情况，而其他边界情况还需要进行详细的判别处理。

对于X2>X1，通过下面代码

if(X1>X2)

{

xx1=X2;xx2=X1;

yy1=Y2;yy2=Y1;

}

else

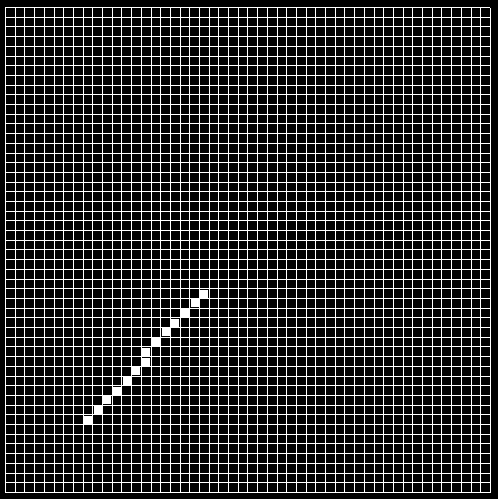
{

xx1=X1;xx2=X2;

yy1=Y1;yy2=Y2;

}

就可以统一算法的实现。而对于dx与dy的关系，就要对bresenham算法的的公式进行调整。这样得到的结果分别是：



**步骤6 键盘控制响应**

通过传入不同的参数到bresenham（int X1, int Y1, int X2, int Y2）函数中，可以显示不同的结果。这里通过固定第一个点x1=20，y1=20，然后利用键盘设置响应函数动态调整第二个点的位置，就可以观察到动态的效果变化。

设计思路如下：首先定义两个全局变量p2x和p2y，并设定一个初始值在（0，50）范围。在glut框架下所设定的特殊键盘响应函数为special\_keys(int a\_keys, int x, int y)。根据用户按键分别是上下左右，对第二个点的坐标位置进行修改。具体代码如下

void special\_keys(int a\_keys, int x, int y)

{

switch (a\_keys) {

case GLUT\_KEY\_F1:

break;

case GLUT\_KEY\_LEFT:

p2x--;

break;

case GLUT\_KEY\_UP:

p2y++;

break;

case GLUT\_KEY\_RIGHT:

p2x++;

break;

case GLUT\_KEY\_DOWN:

p2y--;

break;

default:

break;

}

}

**九、实验数据及结果分析：**

由于所有的绘制都是在原点（0，0，0）进行，而OpenGL缺省的视点是在原点（0，0，0）想z轴的负方向（0，0，-1）观察。所以还需要把图像向z轴负方向推送glTranslatef(0,0,-5.0f);。

bresenham算法在书上的推导是在一定限定条件下进行的。根据bresenham算法，生成线段的公式如下

得到决策参数的递推公式是：

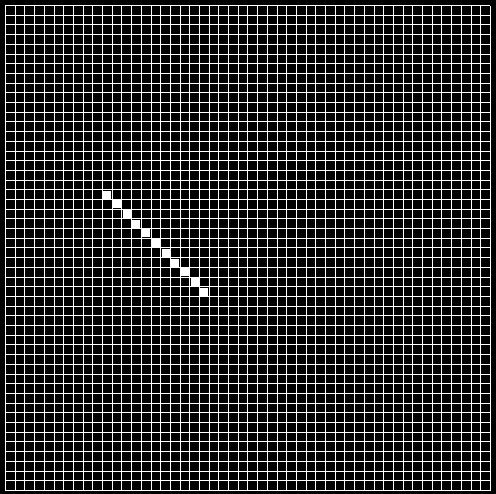
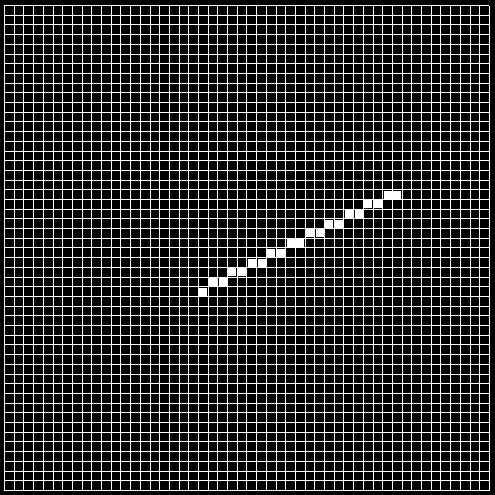
其中，

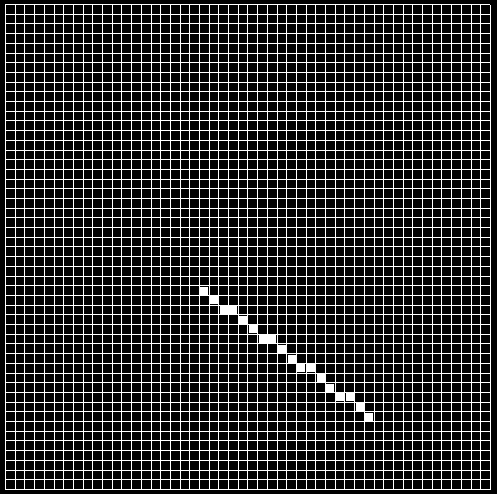
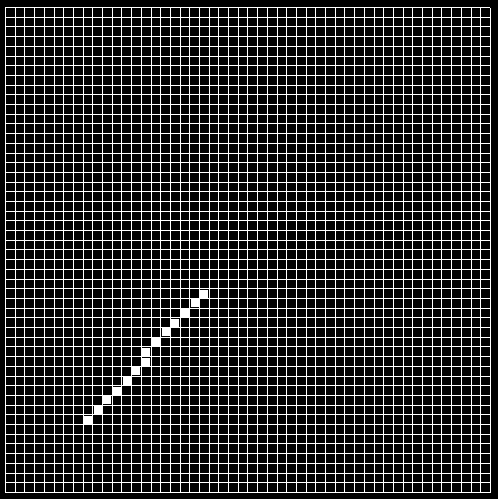
这个递推公式中，前提条件是dx>dy，而且x1<x2的情况。而当dy>dx的时候

决策参数的递推公式变为：

其中，

通过键盘分别能够实现对四个区域的位置的线段生成。





从结果图中可以看到，当图像在垂直和水平的时候，线段是比较规则，但当线段在倾斜的时候，锯齿状效果特别明显，这就是书中所说的走样问题，原因是屏幕整数坐标对线段的取样生成，低频取样造成的结果。

**十、实验结论：**

Bresenham算法经过决策参数的引入，大大提高了效率。线段生成中按照递推的情况，每次计算只涉及整数的加减，以及逻辑判断，所以效率非常好。由于低频取样，在线段生成的之后，产生了走样的效果。

**十一、总结及心得体会：**

利用OpenGL编程，本身就需要理解计算机处理图形的管线操作，要明白整个图形处理的流程和每个操作的意义。这里通过建立一个模拟屏幕坐标环境，模拟验证bresenham算法，通过交互的方法，得到更加直观和生动的展示。

**十二、对本实验过程及方法、手段的改进建议：**

可以在网络上分享自己的成果。

**报告评分： XX**

**指导教师签字：XXXX**