|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 作业1 | 作业2 | 作业3 | 作业4 | 作业5 | 作业6 | 作业7 |
| 成绩 |  |  |  |  |  |  |  |

**作业1**

第3章 基本图元的扫描转换

**题目描述：**

使用扫描转换算法绘制完整的圆。

1. 为了要绘制一个完整的圆必须要对坐标轴进行转换。
2. 绘制一个笛卡尔系坐标轴。
3. 使用扫描转换在笛卡尔系中做圆。

**程序：**

C++代码如下：

// CTest1View 绘图

void CTest1View::OnDraw(CDC\* pDC)

{

CTest1Doc\* pDoc = GetDocument();

ASSERT\_VALID(pDoc);

if (!pDoc)

return;

// 获取视图矩形区域

CRect rect;

GetClientRect(&rect);

// 绘制笛卡尔坐标系

pDC->MoveTo(rect.left, rect.Height() / 2);

pDC->LineTo(rect.right, rect.Height() / 2);

pDC->MoveTo(rect.Width() / 2, rect.top);

pDC->LineTo(rect.Width() / 2, rect.bottom);

// 圆心坐标

int centerX = rect.Width() / 2;

int centerY = rect.Height() / 2;

// 半径

int radius = min(rect.Width(), rect.Height()) / 4;

int x = 0, y = radius;

int d = 1 - radius;

// 使用扫描转换绘制圆

while (y >= x)

{

pDC->SetPixel(centerX + x, centerY + y, RGB(0, 0, 0));

pDC->SetPixel(centerX + y, centerY + x, RGB(0, 0, 0));

pDC->SetPixel(centerX - x, centerY + y, RGB(0, 0, 0));

pDC->SetPixel(centerX - y, centerY + x, RGB(0, 0, 0));

pDC->SetPixel(centerX + x, centerY - y, RGB(0, 0, 0));

pDC->SetPixel(centerX + y, centerY - x, RGB(0, 0, 0));

pDC->SetPixel(centerX - x, centerY - y, RGB(0, 0, 0));

pDC->SetPixel(centerX - y, centerY - x, RGB(0, 0, 0));

x++;

if (d < 0)

d += 2 \* x + 1;

else

{

y--;

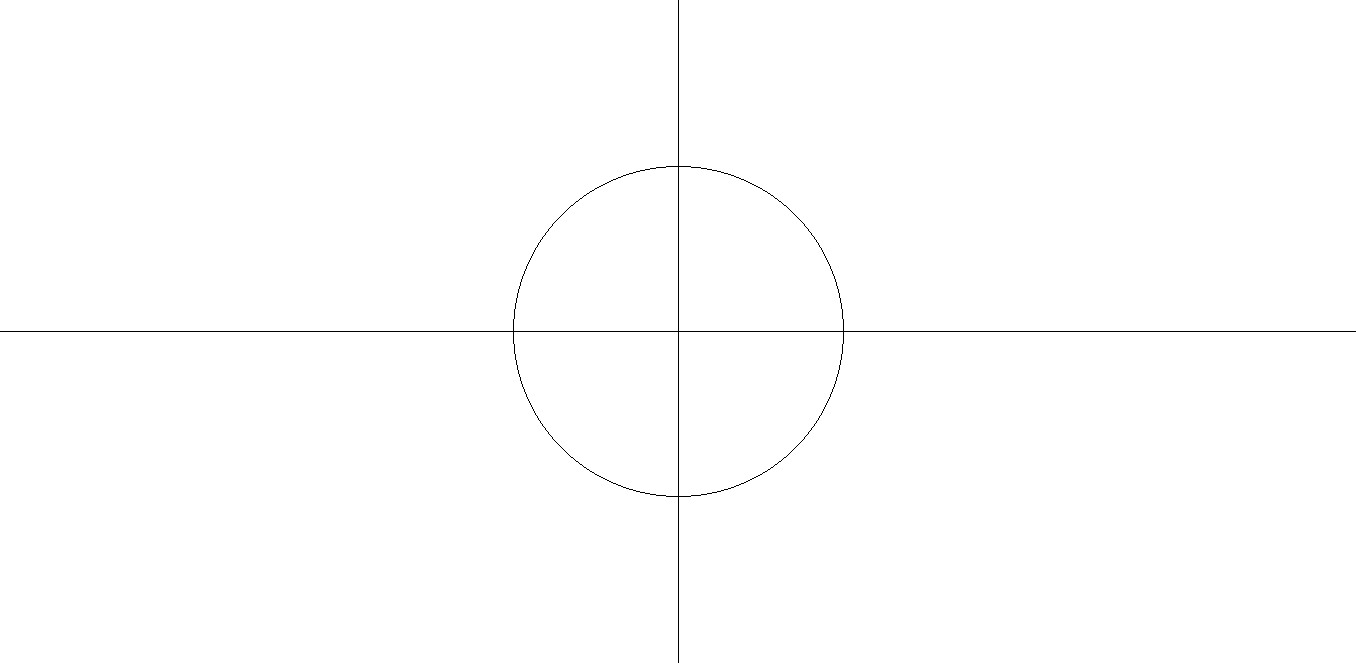
d += 2 \* (x - y) + 1;

}

}

}

**运行结果：**

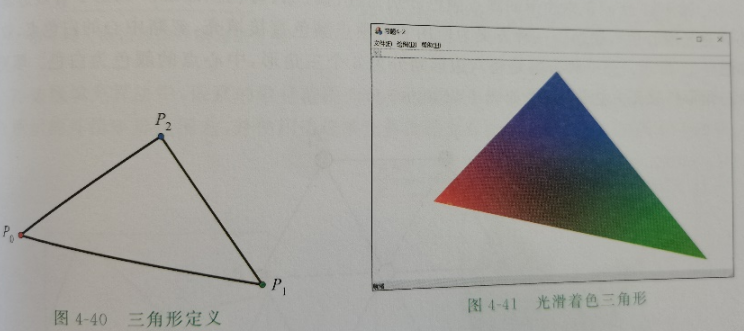


**作业2**

第4章 多边形填充

**题目描述：**

使用有效边表填充三角形，使得填充效果如下。



**程序：**

C++代码如下：

// CTest2View 消息处理程序

void CTest2View::OnMyDraw()

{

GdiplusStartupInput gdiplusStartupInput;

ULONG\_PTR gdiplusToken;//标识符号

GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, NULL);

{

// 创建 GDI+ 绘图对象

Graphics graphics(GetSafeHwnd());

// 定义三个渐变色

Color color1(255, 255, 0); // 黄色

Color color2(0, 255, 255); // 青色

Color color3(255, 0, 255); // 粉色

// 创建渐变画笔

LinearGradientBrush brush(Point(0, 0), Point(800, 400), color1, color2);

REAL positions[] = { 0.0f, 1.0f };

Color colors[] = { color1, color2 };

brush.SetInterpolationColors(colors, positions, 2);

// 创建画笔

Pen pen(&brush, 3);

// 定义三角形的三个顶点

PointF points[4] = { PointF(400.0f, 100.0f), PointF(800.0f, 100.0f), PointF(400.0f, 400.0f), PointF(400.0f, 100.0f) };

// 使用画笔绘制空心的渐变色边的三角形

graphics.DrawLines(&pen, points, 4);

}

// 关闭 GDI+

GdiplusShutdown(gdiplusToken);

}

void CTest2View::OnMyDraw3()

{

// 创建 GDI+ 绘图对象

Gdiplus::Graphics graphics(GetSafeHwnd());

// 定义三角形的三个顶点

Gdiplus::PointF points[3] = { Gdiplus::PointF(200.0f, 200.0f), Gdiplus::PointF(400.0f, 100.0f), Gdiplus::PointF(300.0f, 400.0f) };

// 确定边界框

REAL minXCoord = points[0].X;

REAL minYCoord = points[0].Y;

REAL maxXCoord = points[0].X;

REAL maxYCoord = points[0].Y;

for (int i = 1; i < 3; ++i) {

if (points[i].X < minXCoord) minXCoord = points[i].X;

if (points[i].X > maxXCoord) maxXCoord = points[i].X;

if (points[i].Y < minYCoord) minYCoord = points[i].Y;

if (points[i].Y > maxYCoord) maxYCoord = points[i].Y;

}

// 扫描线填充算法

for (int y = (int)minYCoord; y <= (int)maxYCoord; ++y) {

std::vector<int> intersections;

// 找到与扫描线相交的边

for (int i = 0; i < 3; ++i) {

int next = (i + 1) % 3;

if ((int)points[i].Y < y && (int)points[next].Y >= y ||

(int)points[next].Y < y && (int)points[i].Y >= y) {

float x = points[i].X + (y - points[i].Y) / (points[next].Y - points[i].Y) \* (points[next].X - points[i].X);

intersections.push\_back((int)x);

}

}

// 对交点排序

std::sort(intersections.begin(), intersections.end());

// 填充像素

for (size\_t i = 0; i < intersections.size(); i += 2) {

int startX = (intersections[i] > minXCoord) ? intersections[i] : minXCoord;

int endX = (intersections[i + 1] < maxXCoord) ? intersections[i + 1] : maxXCoord;

// 计算渐变色

Gdiplus::Color startColor(255, 255, 0); // 黄色

Gdiplus::Color endColor(0, 0, 255); // 蓝色

Gdiplus::LinearGradientBrush brush(Gdiplus::PointF(startX, y), Gdiplus::PointF(endX, y), startColor, endColor);

// 填充像素

for (int x = startX; x <= endX; ++x) {

graphics.FillRectangle(&brush, x, y, 1, 1); // 填充渐变色矩形

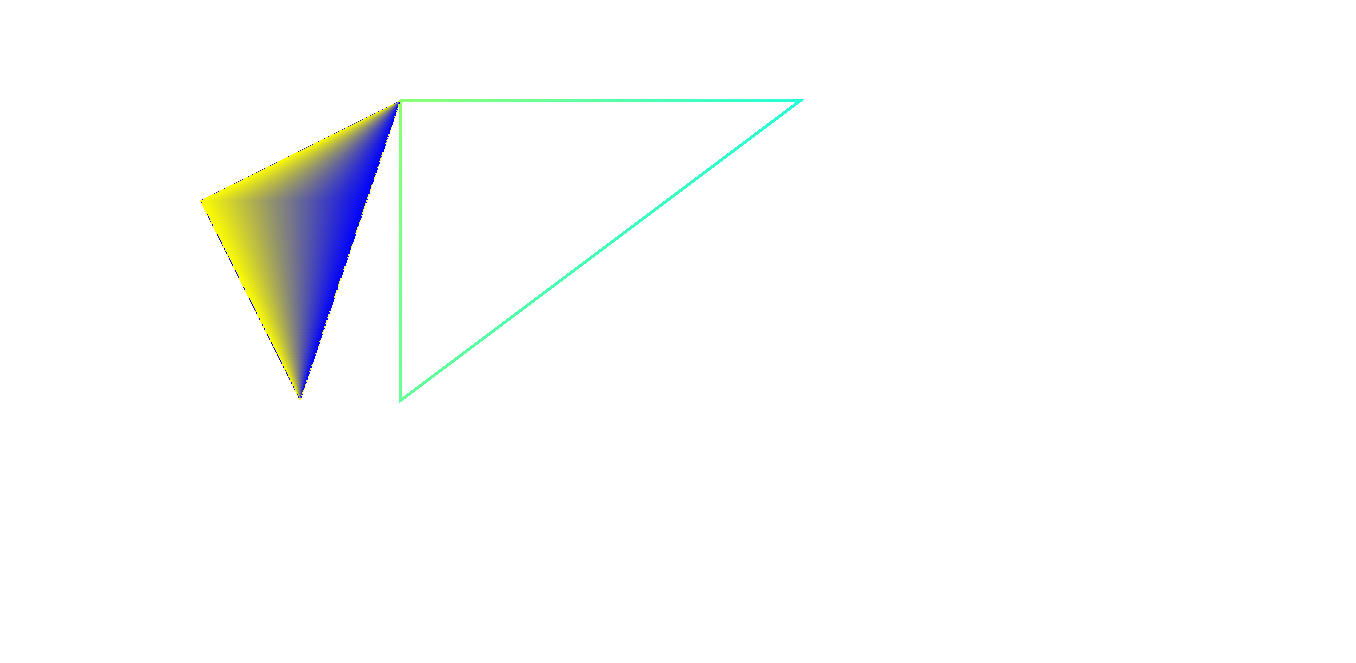
}

}

}

}

**运行结果：**



**作业3**

第5章 二维变换与裁剪

**题目描述：**

实现三角形的旋转

1. 首先要建立一个坐标系
2. 可以手动自己取点得到三角形
3. 自己决定绕哪一点的旋转

程序代码：

void CcgTest1View::symx()

{

CDC \*pDC = GetWindowDC();

for (int i = 0; i < points.size(); i++)

points[i].y = 2\*cy - points[i].y;

for (int i = 0; i < points.size(); i++)

{

int t;

if (i < points.size() - 1)

t = i + 1;

else

t = 0;

pDC->MoveTo(points[i]);

pDC->LineTo(points[t]);

}

}

void CcgTest1View::symy()

{

CDC \*pDC = GetWindowDC();

for (int i = 0; i < points.size(); i++)

points[i].x = 2 \* cx - points[i].x;

for (int i = 0; i < points.size(); i++)

{

int t;

if (i < points.size() - 1)

t = i + 1;

else

t = 0;

pDC->MoveTo(points[i]);

pDC->LineTo(points[t]);

}

}

void CcgTest1View::symo()

{

CDC \*pDC = GetWindowDC();

for (int i = 0; i < points.size(); i++)

{

points[i].y = 2 \* cy - points[i].y;

points[i].x = 2 \* cx - points[i].x;

}

for (int i = 0; i < points.size(); i++)

{

int t;

if (i < points.size() - 1)

t = i + 1;

else

t = 0;

pDC->MoveTo(points[i]);

pDC->LineTo(points[t]);

}

}

void CcgTest1View::symxy()

{

CDC \*pDC = GetWindowDC();

for (int i = 0; i < points.size(); i++)

{

int temp = points[i].x;

points[i].x = cx + cy - points[i].y;

points[i].y= cx + cy - temp;

}

for (int i = 0; i < points.size(); i++)

{

int t;

if (i < points.size() - 1)

t = i + 1;

else

t = 0;

pDC->MoveTo(points[i]);

pDC->LineTo(points[t]);

}

}

void CcgTest1View::protate()

{

CDC\* pDC = GetWindowDC();

double angle = M\_PI / 2; // 90度对应的弧度

for (int i = 0; i < points.size(); i++)

{

int x = points[i].x - cx;

int y = points[i].y - cy;

points[i].x = cos(angle) \* x - sin(angle) \* y + cx;

points[i].y = sin(angle) \* x + cos(angle) \* y + cy;

}

for (int i = 0; i < points.size(); i++)

{

int t;

if (i < points.size() - 1)

t = i + 1;

else

t = 0;

pDC->MoveTo(points[i]);

pDC->LineTo(points[t]);

}

}

void CcgTest1View::nrotate()

{

CDC \*pDC = GetWindowDC();

for (int i = 0; i < points.size(); i++)

{

int x = points[i].x -cx;

int y = points[i].y -cy;

points[i].x = cos(-0.1)\*x - sin(-0.1)\*y +cx;

points[i].y = sin(-0.1)\*x + cos(-0.1)\*y +cy;

}

for (int i = 0; i < points.size(); i++)

{

int t;

if (i < points.size() - 1)

t = i + 1;

else

t = 0;

pDC->MoveTo(points[i]);

pDC->LineTo(points[t]);

}

}

void CcgTest1View::smaller()

{

CDC \*pDC = GetWindowDC();

for (int i = 0; i < points.size(); i++)

{

int x = points[i].x - cx;

int y = points[i].y - cy;

points[i].x = 0.5\*x + cx;

points[i].y = 0.5\*y + cy;

}

for (int i = 0; i < points.size(); i++)

{

int t;

if (i < points.size() - 1)

t = i + 1;

else

t = 0;

pDC->MoveTo(points[i]);

pDC->LineTo(points[t]);

}

}

void CcgTest1View::larger()

{

CDC \*pDC = GetWindowDC();

for (int i = 0; i < points.size(); i++)

{

int x = points[i].x - cx;

int y = points[i].y - cy;

points[i].x = 2\*x + cx;

points[i].y = 2\*y + cy;

}

for (int i = 0; i < points.size(); i++)

{

int t;

if (i < points.size() - 1)

t = i + 1;

else

t = 0;

pDC->MoveTo(points[i]);

pDC->LineTo(points[t]);

}

}

void CcgTest1View::OnAnm()

{

SetTimer(1, 50, nullptr);

}

void CcgTest1View::OnTimer(UINT\_PTR nIDEvent)

{

CDC \*pDC = GetWindowDC();

RedrawWindow();

pDC->Rectangle(cx1,cy1,cx2,cy2);

cx1 += xs;

cx2 += xs;

cy1 += ys;

cy2 += ys;

if (cx1 > 1000 || cx1 < 100)

xs = -xs;

if (cy1 > 500 || cy1 < 100)

ys = -ys;

CView::OnTimer(nIDEvent);

}

void CcgTest1View::symxny()

{

CDC \*pDC = GetWindowDC();

for (int i = 0; i < points.size(); i++)

{

int temp = points[i].x;

points[i].x = cx - cy + points[i].y;

points[i].y = -cx + cy + temp;

}

for (int i = 0; i < points.size(); i++)

{

int t;

if (i < points.size() - 1)

t = i + 1;

else

t = 0;

pDC->MoveTo(points[i]);

pDC->LineTo(points[t]);

}

}

void CcgTest1View::OnTriangleP1()

{

CDC\* pDC = GetWindowDC();

double angle = M\_PI / 2; // 90度对应的弧度

double centerx, centery;

centerx = points[0].x;

centery = points[0].y;

for (int i = 0; i < points.size(); i++)

{

int x = points[i].x - centerx;

int y = points[i].y - centery;

points[i].x = cos(angle) \* x - sin(angle) \* y + centerx;

points[i].y = sin(angle) \* x + cos(angle) \* y + centery;

}

for (int i = 0; i < points.size(); i++)

{

int t;

if (i < points.size() - 1)

t = i + 1;

else

t = 0;

pDC->MoveTo(points[i]);

pDC->LineTo(points[t]);

}

// TODO: 在此添加命令处理程序代码

}

void CcgTest1View::rotateTriangle(double angle) {

// 计算旋转角度对应的弧度

double radians = angle \* M\_PI / 2;

// 假设points中有三个顶点，分别是p1、p2、p3

CPoint p1 = points[0];

CPoint p2 = points[1];

CPoint p3 = points[2];

//// 计算三角形中心点的坐标

//int centerX = (p1.x + p2.x + p3.x) / 3;

//int centerY = (p1.y + p2.y + p3.y) / 3;

// 计算三角形中心点的坐标

int centerX = p1.x;

int centerY = p1.y;

// 分别围绕三个顶点旋转

rotateAroundPoint(p1, radians, centerX, centerY);

//rotateAroundPoint(p2, radians, centerX, centerY);

//rotateAroundPoint(p3, radians, centerX, centerY);

// 更新三角形的顶点坐标

points[0] = p1;

points[1] = p2;

points[2] = p3;

}

void CcgTest1View::rotateAroundPoint(CPoint& point, double angle, int centerX, int centerY) {

double sinVal = sin(angle);

double cosVal = cos(angle);

// 将点移动到中心

int x = point.x - centerX;

int y = point.y - centerY;

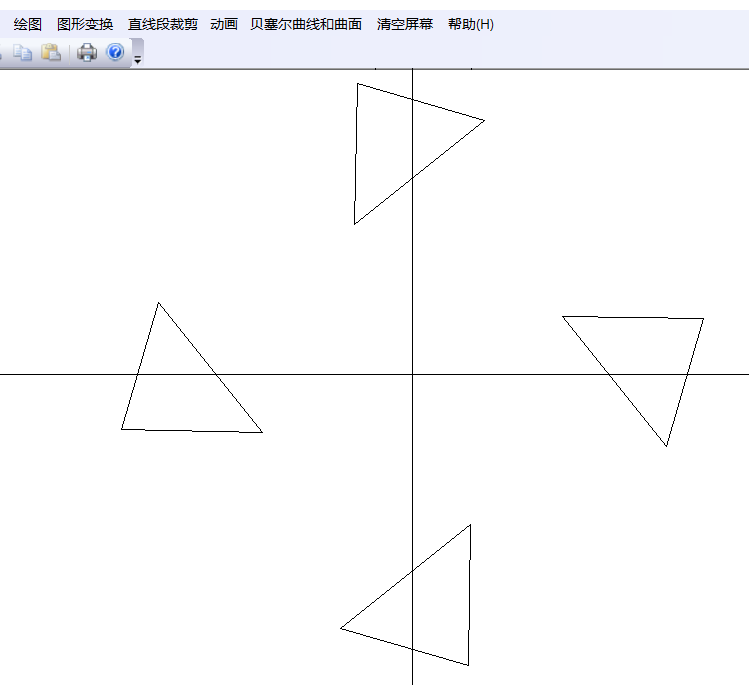
// 应用旋转矩阵

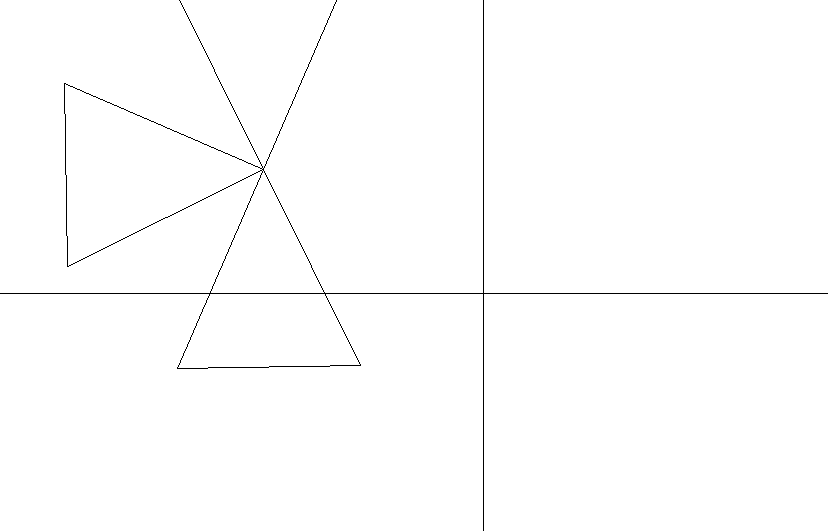
point.x = static\_cast<int>(x \* cosVal - y \* sinVal + centerX);

point.y = static\_cast<int>(x \* sinVal + y \* cosVal + centerY);

}

运行结果：





**作业4**

第6章 三维变换与投影

**题目描述：**

对任意一个三棱锥的透视投影

程序代码：

//计算矩阵乘

void CcgTest1View::matrix(double b[4][4]/\*旋转变换矩阵\*/, double m)

{

int d[4][4];//把点变成齐次的

for (int i = 0; i < 4; i++) {

d[i][0] = a[i][0];

d[i][1] = a[i][1];

d[i][2] = a[i][2];

d[i][3] = 1;

}

//与透视投影整体变换矩阵相乘

for (int i = 0; i < 4; i++) {

double s2 = 0;

//使得二维点的齐次最后一个为1

s2 = d[i][0] \* b[0][3] + d[i][1] \* b[1][3] + d[i][2] \* b[2][3] + d[i][3] \* b[3][3];

z[i][2] = s2 \* m;

for (int j = 0; j < 2; j++) {

double s = 0;

s = d[i][0] \* b[0][j] + d[i][1] \* b[1][j] + d[i][2] \* b[2][j] + d[i][3] \* b[3][j];

c[i][j] = s / s2;

z[i][j] = s;

s = 0;

}

s2 = 0;

}

}

//透视投影

void CcgTest1View::perspect(double then, double fine, double d, double r)

{

//角度转变

then = then \* 3.1415926;

fine = fine \* 3.1415926;

//首先构造变换矩阵

double k1, k2, k3, k4, k5, k6, k7, k8;

k1 = sin(then), k2 = sin(fine), k3 = cos(then), k4 = cos(fine), k5 = k2 \* k3, k6 = k1 \* k2, k7 = k3 \* k4, k8 = k4 \* k1;

double b[4][4];

b[0][0] = k3, b[0][1] = -k8, b[0][2] = 0, b[0][3] = -k6 / d;

b[1][0] = 0, b[1][1] = k2, b[1][2] = 0, b[1][3] = -k4 / d;

b[2][0] = -k1, b[2][1] = -k7, b[2][2] = 0, b[2][3] = -k5 / d;

b[3][0] = 0, b[3][1] = 0, b[3][2] = 0, b[3][3] = r / d;

matrix(b, d);

//移到中心

for (int i = 0; i < 4; i++) {

c[i][0] += 250;

c[i][1] = 300 - c[i][1];

}

//画出透视投影

CDC\* pDC = GetDC();

pDC->MoveTo(c[0][0], c[0][1]);

pDC->LineTo(c[1][0], c[1][1]);

pDC->LineTo(c[3][0], c[3][1]);//到顶点的

pDC->MoveTo(c[1][0], c[1][1]);

pDC->LineTo(c[2][0], c[2][1]);

pDC->LineTo(c[3][0], c[3][1]);//到顶点的

pDC->MoveTo(c[2][0], c[2][1]);

pDC->LineTo(c[0][0], c[0][1]);

pDC->LineTo(c[3][0], c[3][1]);//到顶点的

pDC->MoveTo(c[0][0], c[0][1]);

ReleaseDC(pDC);

}

//从这里开始就直接使用隐线算法，上面是透视投影

//

//

//计算外矢量

void CcgTest1View::acclu(int v[3]/\*三个点\*/)

{

//以第一个点为参考点

int x[3], y[3];

for (int i = 0; i < 3; i++) {

x[i] = z[v[1]][i] - z[v[0]][i];

y[i] = z[v[2]][i] - z[v[0]][i];

}

N[0] = x[1] \* y[2] - x[2] \* y[1];

N[1] = x[2] \* y[0] - x[0] \* y[2];

N[2] = x[0] \* y[1] - x[1] \* y[0];

}

//判断是否可视

void CcgTest1View::vision(double then, double fine, double r, int v[3])

{

//视矢量

double s[3];

s[0] = r \* sin(fine \* 3.1415926) \* sin(then \* 3.1415926) - z[v[0]][0];

s[1] = r \* cos(fine \* 3.1415926) - z[v[0]][1];

s[2] = r \* sin(fine \* 3.1415926) \* cos(then \* 3.1415926) - z[v[0]][2];

acclu(v);

double jug = 0;

for (int i = 0; i < 3; i++) {

jug += s[i] \* N[i];

}

if (jug > 0) {

h[v[0]][v[1]] = h[v[1]][v[0]] = 1;

h[v[0]][v[2]] = h[v[2]][v[0]] = 1;

h[v[2]][v[1]] = h[v[1]][v[2]] = 1;

}

else if (jug == 0) {

h[v[2]][v[1]] = h[v[1]][v[2]] = 1;

}

}

//计算边表

void CcgTest1View::xiaoyin(double then, double fine, double r/\*视径\*/)

{

//其实边表是一个对称矩阵，在连边时只需要看下三角

//先进行边表初始化

for (int i = 0; i < 4; i++) {

for (int j = 0; j < 4; j++) {

h[i][j] = 0;

}

}

//第一个面

int v[3];

v[0] = 1, v[1] = 3, v[2] = 2;

vision(then, fine, r, v);

//第二个面

v[0] = 0, v[1] = 3, v[2] = 1;

vision(then, fine, r, v);

//第三个面

v[0] = 0, v[1] = 2, v[2] = 3;

vision(then, fine, r, v);

//第四个面

v[0] = 0, v[1] = 1, v[2] = 2;

vision(then, fine, r, v);

}

void CcgTest1View::onDrawPsm()

{

/\*在这里定义三棱锥的四个顶点\*/

for (int i = 0; i < 3; i++) {

a[i][2] = 100;

}

a[3][2] = 600;

a[0][0] = 100, a[1][0] = 100, a[2][0] = 400, a[3][0] = 100;

a[0][1] = 100, a[1][1] = 400, a[2][1] = 200, a[3][1] = 300;

//输入参数

double str1, str2, str3, str4;

str1 = 0.25;

str2 = 0.25;

str3 = 60;

str4 = 120;

perspect(str1, str2,str3,str4);

xiaoyin(str1,str2, str4);

// TODO: 在此添加控件通知处理程序代码

// TODO: 在此添加命令处理程序代码

}

void CcgTest1View::onDrawPsx()

{

//移动

for (int i = 0; i < 4; i++) {

c[i][0] += 300;

}

CDC\* pDC = GetDC();

for (int i = 0; i < 4; i++) {

for (int j = 0; j <= i - 1; j++) {

if (h[i][j] == 1) {

pDC->MoveTo(c[i][0], c[i][1]);

pDC->LineTo(c[j][0], c[j][1]);

}

}

}

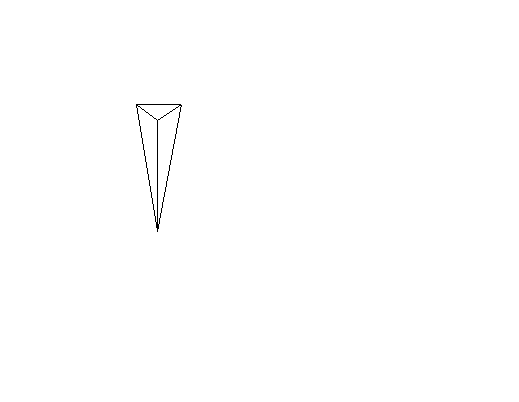
ReleaseDC(pDC);

// TODO: 在此添加控件通知处理程序代码

// TODO: 在此添加命令处理程序代码

}

运行结果：



**作业5**

第7章 自由曲线与曲面

**题目描述：**

根据三次Bezier曲线的基函数绘制三次Bezier曲线

**程序：**

void DrawCubicBezierCurve(Graphics& graphics, const Point& startPoint, const Point& endPoint,

const Point& controlPoint1, const Point& controlPoint2)

{

Pen pen(Color(255, 0, 0, 255)); // 定义红色画笔

// 绘制三次贝塞尔曲线

graphics.DrawBezier(&pen, startPoint.X, startPoint.Y,

controlPoint1.X, controlPoint1.Y,

controlPoint2.X, controlPoint2.Y,

endPoint.X, endPoint.Y);

}

void CcgTest1View::threeDraw()

{

GdiplusStartupInput gdiplusStartupInput;

ULONG\_PTR gdiplusToken;

GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, NULL);

CDC\* pDC = GetWindowDC();

Graphics graphics(pDC->m\_hDC);

// 定义曲线的端点和控制点

Point startPoint(100, 100);

Point endPoint(300, 300);

Point controlPoint1(150, 50);

Point controlPoint2(250, 250);

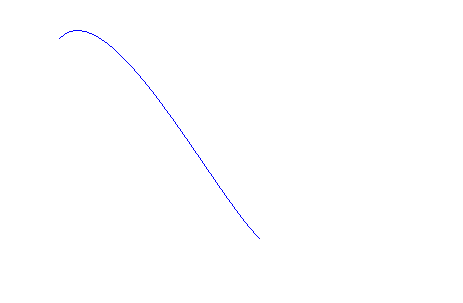
// 绘制三次贝塞尔曲线

DrawCubicBezierCurve(graphics, startPoint, endPoint, controlPoint1, controlPoint2);

// TODO: 在此添加命令处理程序代码

}

程序结果：



**作业6**

第8章 建模与消隐

**题目描述：**

实现正四面体的透视投影的动态隐线算法

（在算法中，一 要注意叉乘的方向根据实际情况来；2 要注意坐标系是观察坐标系）

程序代码：

//计算矩阵乘

void CcgTest1View::matrix(double b[4][4]/\*旋转变换矩阵\*/, double m)

{

int d[4][4];//把点变成齐次的

for (int i = 0; i < 4; i++) {

d[i][0] = a[i][0];

d[i][1] = a[i][1];

d[i][2] = a[i][2];

d[i][3] = 1;

}

//与透视投影整体变换矩阵相乘

for (int i = 0; i < 4; i++) {

double s2 = 0;

//使得二维点的齐次最后一个为1

s2 = d[i][0] \* b[0][3] + d[i][1] \* b[1][3] + d[i][2] \* b[2][3] + d[i][3] \* b[3][3];

z[i][2] = s2 \* m;

for (int j = 0; j < 2; j++) {

double s = 0;

s = d[i][0] \* b[0][j] + d[i][1] \* b[1][j] + d[i][2] \* b[2][j] + d[i][3] \* b[3][j];

c[i][j] = s / s2;

z[i][j] = s;

s = 0;

}

s2 = 0;

}

}

//透视投影

void CcgTest1View::perspect(double then, double fine, double d, double r)

{

//角度转变

then = then \* 3.1415926;

fine = fine \* 3.1415926;

//首先构造变换矩阵

double k1, k2, k3, k4, k5, k6, k7, k8;

k1 = sin(then), k2 = sin(fine), k3 = cos(then), k4 = cos(fine), k5 = k2 \* k3, k6 = k1 \* k2, k7 = k3 \* k4, k8 = k4 \* k1;

double b[4][4];

b[0][0] = k3, b[0][1] = -k8, b[0][2] = 0, b[0][3] = -k6 / d;

b[1][0] = 0, b[1][1] = k2, b[1][2] = 0, b[1][3] = -k4 / d;

b[2][0] = -k1, b[2][1] = -k7, b[2][2] = 0, b[2][3] = -k5 / d;

b[3][0] = 0, b[3][1] = 0, b[3][2] = 0, b[3][3] = r / d;

matrix(b, d);

//移到中心

for (int i = 0; i < 4; i++) {

c[i][0] += 250;

c[i][1] = 300 - c[i][1];

}

//画出透视投影

CDC\* pDC = GetDC();

pDC->MoveTo(c[0][0], c[0][1]);

pDC->LineTo(c[1][0], c[1][1]);

pDC->LineTo(c[3][0], c[3][1]);//到顶点的

pDC->MoveTo(c[1][0], c[1][1]);

pDC->LineTo(c[2][0], c[2][1]);

pDC->LineTo(c[3][0], c[3][1]);//到顶点的

pDC->MoveTo(c[2][0], c[2][1]);

pDC->LineTo(c[0][0], c[0][1]);

pDC->LineTo(c[3][0], c[3][1]);//到顶点的

pDC->MoveTo(c[0][0], c[0][1]);

ReleaseDC(pDC);

}

//从这里开始就直接使用隐线算法，上面是透视投影

//

//

//计算外矢量

void CcgTest1View::acclu(int v[3]/\*三个点\*/)

{

//以第一个点为参考点

int x[3], y[3];

for (int i = 0; i < 3; i++) {

x[i] = z[v[1]][i] - z[v[0]][i];

y[i] = z[v[2]][i] - z[v[0]][i];

}

N[0] = x[1] \* y[2] - x[2] \* y[1];

N[1] = x[2] \* y[0] - x[0] \* y[2];

N[2] = x[0] \* y[1] - x[1] \* y[0];

}

//判断是否可视

void CcgTest1View::vision(double then, double fine, double r, int v[3])

{

//视矢量

double s[3];

s[0] = r \* sin(fine \* 3.1415926) \* sin(then \* 3.1415926) - z[v[0]][0];

s[1] = r \* cos(fine \* 3.1415926) - z[v[0]][1];

s[2] = r \* sin(fine \* 3.1415926) \* cos(then \* 3.1415926) - z[v[0]][2];

acclu(v);

double jug = 0;

for (int i = 0; i < 3; i++) {

jug += s[i] \* N[i];

}

if (jug > 0) {

h[v[0]][v[1]] = h[v[1]][v[0]] = 1;

h[v[0]][v[2]] = h[v[2]][v[0]] = 1;

h[v[2]][v[1]] = h[v[1]][v[2]] = 1;

}

else if (jug == 0) {

h[v[2]][v[1]] = h[v[1]][v[2]] = 1;

}

}

//计算边表

void CcgTest1View::xiaoyin(double then, double fine, double r/\*视径\*/)

{

//其实边表是一个对称矩阵，在连边时只需要看下三角

//先进行边表初始化

for (int i = 0; i < 4; i++) {

for (int j = 0; j < 4; j++) {

h[i][j] = 0;

}

}

//第一个面

int v[3];

v[0] = 1, v[1] = 3, v[2] = 2;

vision(then, fine, r, v);

//第二个面

v[0] = 0, v[1] = 3, v[2] = 1;

vision(then, fine, r, v);

//第三个面

v[0] = 0, v[1] = 2, v[2] = 3;

vision(then, fine, r, v);

//第四个面

v[0] = 0, v[1] = 1, v[2] = 2;

vision(then, fine, r, v);

}

void CcgTest1View::onDrawPsm()

{

/\*在这里定义三棱锥的四个顶点\*/

for (int i = 0; i < 3; i++) {

a[i][2] = 100;

}

a[3][2] = 600;

a[0][0] = 100, a[1][0] = 100, a[2][0] = 400, a[3][0] = 100;

a[0][1] = 100, a[1][1] = 400, a[2][1] = 200, a[3][1] = 300;

//输入参数

double str1, str2, str3, str4;

str1 = 0.25;

str2 = 0.25;

str3 = 60;

str4 = 120;

perspect(str1, str2,str3,str4);

xiaoyin(str1,str2, str4);

// TODO: 在此添加控件通知处理程序代码

// TODO: 在此添加命令处理程序代码

}

void CcgTest1View::onDrawPsx()

{

//移动

for (int i = 0; i < 4; i++) {

c[i][0] += 300;

}

CDC\* pDC = GetDC();

for (int i = 0; i < 4; i++) {

for (int j = 0; j <= i - 1; j++) {

if (h[i][j] == 1) {

pDC->MoveTo(c[i][0], c[i][1]);

pDC->LineTo(c[j][0], c[j][1]);

}

}

}

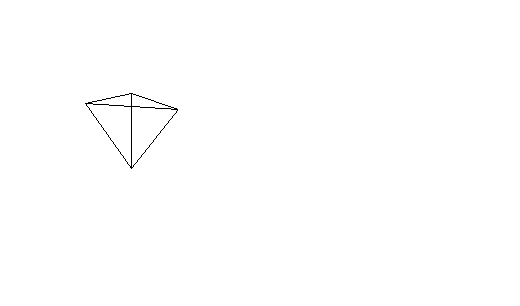
ReleaseDC(pDC);

// TODO: 在此添加控件通知处理程序代码

// TODO: 在此添加命令处理程序代码

}

运行结果：



**作业7**

第9章 光照模型

**题目描述：**

建立RGB几何模型，使用斜等测投影绘制RGB光滑着色模型消隐后的图像

程序代码：

void CcgTest1View::juzheng1(double b[4][4])

{

int d[8][4];

for (int i = 0; i < 8; i++) {

d[i][3] = 1;

d[i][0] = a1[i][0];

d[i][1] = a1[i][1];

d[i][2] = a1[i][2];

}

for (int i = 0; i < 8; i++) {

for (int j = 0; j < 2; j++) {

double s = 0;

s = d[i][0] \* b[0][j] + d[i][1] \* b[1][j] + d[i][2] \* b[2][j] + d[i][3] \* b[3][j];

c1[i][j] = s;

s = 0;

}

}

}

//等测投影

void CcgTest1View::EQUT()

{

double b[4][4];

b[0][0] = 1, b[0][1] = 0, b[0][2] = 0, b[0][3] = 0;

b[1][0] = 0, b[1][1] = 1, b[1][2] = 0, b[1][3] = 0;

b[2][0] = -cos(45), b[2][1] = -sin(45), b[2][2] = 0, b[2][3] = 0;

b[3][0] = 0, b[3][1] = 0, b[3][2] = 0, b[3][3] = 1;

//调用矩阵计算并直接将结果记录

juzheng1(b);

}

//对边进行进行光滑着色

void CcgTest1View::color(int i, int j)

{

CDC\* pDC = GetDC();

if (i != j) {

double k;

if (c1[i][0] != c1[j][0])

k = (double)(c1[i][1] - c1[j][1]) / (c1[i][0] - c1[j][0]);

else

k = 999;

if (abs(k) > 1) {

double x = c1[i][1] < c1[j][1] ? c1[i][0] : c1[j][0];

for (double y = c1[i][1] < c1[j][1] ? c1[i][1] : c1[j][1]; y < max(c1[i][1], c1[j][1]); y++) {

double d1 = (double)(y - c1[j][1]) / (c1[i][1] - c1[j][1]), d2 = (double)(y - c1[i][1]) / (c1[j][1] - c1[i][1]);

if (k != 999) {

x += 1 / k;

pDC->SetPixel(x, y, RGB(255 \* (h1[i][0] \* d1 + h1[j][0] \* d2), 255 \* (h1[i][1] \* d1 + h1[j][1] \* d2), 255 \* (h1[i][2] \* d1 + h1[j][2] \* d2)));

}

else {

pDC->SetPixel(c1[i][0], y, RGB(255 \* (h1[i][0] \* d1 + h1[j][0] \* d2), 255 \* (h1[i][1] \* d1 + h1[j][1] \* d2), 255 \* (h1[i][2] \* d1 + h1[j][2] \* d2)));

}

}

}

else {

double y = c1[i][0] < c1[j][0] ? c1[i][1] : c1[j][1];

for (double x = c1[i][0] < c1[j][0] ? c1[i][0] : c1[j][0]; x < max(c1[i][0], c1[j][0]); x++) {

double d1 = (double)(x - c1[j][0]) / (c1[i][0] - c1[j][0]), d2 = (double)(x - c1[i][0]) / (c1[j][0] - c1[i][0]);

y += k;

pDC->SetPixel(x, y, RGB(255 \* (h1[i][0] \* d1 + h1[j][0] \* d2), 255 \* (h1[i][1] \* d1 + h1[j][1] \* d2), 255 \* (h1[i][2] \* d1 + h1[j][2] \* d2)));

}

}

}

ReleaseDC(pDC);

}

//x主方向填充

void CcgTest1View::fill(double m[2][2], int n[4])

{

CDC\* pDC = GetDC();

double l[2][3];

//求出各自的光强

for (int i = 0; i < 3; i++) {

l[0][i] = h1[n[0]][i] \* ((double)(m[0][1] - c1[n[1]][1]) / (c1[n[0]][1] - c1[n[1]][1])) + h1[n[1]][i] \* ((double)(m[0][1] - c1[n[0]][1]) / (c1[n[1]][1] - c1[n[0]][1]));

}

for (int i = 0; i < 3; i++) {

l[1][i] = h1[n[2]][i] \* ((double)(m[1][1] - c1[n[3]][1]) / (c1[n[2]][1] - c1[n[3]][1])) + h1[n[3]][i] \* ((double)(m[1][1] - c1[n[2]][1]) / (c1[n[3]][1] - c1[n[2]][1]));

}

for (int x = m[0][0] + 1; x < m[1][0]; x++) {

double d1 = (double)(x - m[1][0]) / (m[0][0] - m[1][0]), d2 = (double)(x - m[0][0]) / (m[1][0] - m[0][0]);

pDC->SetPixel(x, m[1][1], RGB(255 \* (l[0][0] \* d1 + l[1][0] \* d2), 255 \* (l[0][1] \* d1 + l[1][1] \* d2), 255 \* (l[0][2] \* d1 + l[1][2] \* d2)));

}

ReleaseDC(pDC);

}

//y主方向填充

void CcgTest1View::fill2(double m[2][2], int n[4])

{

CDC\* pDC = GetDC();

double l[2][3];

//求出各自的光强

for (int i = 0; i < 3; i++) {

l[0][i] = h1[n[0]][i] \* ((double)(m[0][1] - c1[n[1]][1]) / (c1[n[0]][1] - c1[n[1]][1])) + h1[n[1]][i] \* ((double)(m[0][1] - c1[n[0]][1]) / (c1[n[1]][1] - c1[n[0]][1]));

}

for (int i = 0; i < 3; i++) {

l[1][i] = h1[n[2]][i] \* ((double)(m[1][1] - c1[n[3]][1]) / (c1[n[2]][1] - c1[n[3]][1])) + h1[n[3]][i] \* ((double)(m[1][1] - c1[n[2]][1]) / (c1[n[3]][1] - c1[n[2]][1]));

}

for (int y = m[0][1] + 1; y < m[1][1]; y++) {

double d1 = (double)(y - m[1][1]) / (m[0][1] - m[1][1]), d2 = (double)(y - m[0][1]) / (m[1][1] - m[0][1]);

pDC->SetPixel(m[1][0], y, RGB(255 \* (l[0][0] \* d1 + l[1][0] \* d2), 255 \* (l[0][1] \* d1 + l[1][1] \* d2), 255 \* (l[0][2] \* d1 + l[1][2] \* d2)));

}

ReleaseDC(pDC);

}

void CcgTest1View::OnDrawRpg()

{

//初始化点

a1[0][0] = a1[0][1] = a1[0][2] = 200;

a1[1][0] = 200, a1[1][1] = 400, a1[1][2] = 200;

a1[2][0] = 200, a1[2][1] = 400, a1[2][2] = 400;

a1[3][0] = 200, a1[3][1] = 200, a1[3][2] = 400;

a1[4][0] = 400, a1[4][1] = 200, a1[4][2] = 200;

a1[5][0] = 400, a1[5][1] = 400, a1[5][2] = 200;

a1[6][0] = 400, a1[6][1] = 400, a1[6][2] = 400;

a1[7][0] = 400, a1[7][1] = 200, a1[7][2] = 400;

//

EQUT();

CDC\* pDC = GetDC();

//对投影进行坐标变换

for (int i = 0; i < 8; i++) {

c1[i][0] += 300;

c1[i][1] = 350 - c1[i][1];

}

//设置点的光强（h1[8][3]在头文件定义了）

h1[0][0] = 0, h1[0][1] = 0, h1[0][2] = 0;

h1[4][0] = 1, h1[4][1] = 0, h1[4][2] = 0;

h1[5][0] = 1, h1[5][1] = 1, h1[5][2] = 0;

h1[1][0] = 0, h1[1][1] = 1, h1[1][2] = 0;

h1[3][0] = 0, h1[3][1] = 0, h1[3][2] = 1;

h1[7][0] = 1, h1[7][1] = 0, h1[7][2] = 1;

h1[6][0] = 1, h1[6][1] = 1, h1[6][2] = 1;

h1[2][0] = 0, h1[2][1] = 1, h1[2][2] = 1;

//

//首先对边进行光滑着色

color(5, 4);

color(5, 1);

color(7, 3);

color(7, 6);

color(6, 2);

color(2, 3);

color(1, 2);

color(5, 6);

color(4, 7);

//

//扫面线进行填充

double m[2][2];

//前面

int n[4];

n[0] = 2, n[1] = 3, n[2] = 6, n[3] = 7;

for (int i = min(c1[2][1], c1[3][1]) + 1; i < max(c1[2][1], c1[3][1]); i++) {

m[0][0] = c1[2][0], m[1][0] = c1[6][0];

m[0][1] = m[1][1] = i;

fill(m, n);

}

//上面

n[0] = 1, n[1] = 2, n[2] = 5, n[3] = 6;

double k = (double)(c1[1][1] - c1[2][1]) / (c1[1][0] - c1[2][0]);

m[0][0] = c1[1][0], m[1][0] = c1[5][0];

for (int i = min(c1[2][1], c1[1][1]) + 1; i < max(c1[2][1], c1[1][1]); i++) {

m[0][1] = m[1][1] = i;

m[0][0] += 1 / k, m[1][0] += 1 / k;

fill(m, n);

}

//侧面

n[0] = 6, n[1] = 5, n[2] = 7, n[3] = 4;

k = (double)(c1[5][1] - c1[6][1]) / (c1[5][0] - c1[6][0]);

m[0][1] = c1[6][1], m[1][1] = c1[7][1];

for (int i = min(c1[6][0], c1[5][0]) + 1; i < max(c1[6][0], c1[5][0]); i++) {

m[0][1] += k, m[1][1] += k;

m[0][0] = m[1][0] = i;

fill2(m, n);

}

//

// TODO: 在此添加命令处理程序代码

}

运行结果：

