|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **实验一** | **实验二** | **实验三** | **总评** |
| **成绩** |  |  |  |  |

**江苏科技大学**

**计算机图形学课内实验报告**

**（2022/2023学年第2学期）**

**学生姓名： 陈余**

**学生学号： 212241807213**

**院 系： 计算机学院**

**专 业： 计算机科学与技术**

**考核得分：**

2024 **年 6 月 30 日**

目录

[目录 1](#_Toc135208528)

[项目1图形的几何变换与图形裁剪 1](#_Toc135208529)

[1. 问题 1](#_Toc135208530)

[2. 模块设计 1](#_Toc135208531)

[3. 函数调用关系 1](#_Toc135208532)

[4. 实验结果及分析 1](#_Toc135208533)

[5. 附录 1](#_Toc135208534)

[项目2图形的区域填充 1](#_Toc135208535)

[1. 问题 1](#_Toc135208536)

[2. 模块设计 1](#_Toc135208537)

[3. 函数调用关系（如果下表行列不够，请自行扩充） 2](#_Toc135208538)

[4. 实验结果及分析 2](#_Toc135208539)

[5. 附录 2](#_Toc135208540)

[项目3曲线曲面的造型 2](#_Toc135208541)

[1. 问题 2](#_Toc135208542)

[2. 模块设计 2](#_Toc135208543)

[3. 函数调用关系（如果下表行列不够，请自行扩充） 2](#_Toc135208544)

[4. 实验结果及分析 3](#_Toc135208545)

[5. 附录 3](#_Toc135208546)

项目1图形的几何变换与图形裁剪

1. 问题

开发带有可视化图形交互界面的程序，实现多边形的几何变换和图形裁剪。

1. 模块设计

该实验程序包含以下模块：

①建立坐标系

②双击右键取点

③画出取点后的多边形

④矩阵相乘计算

⑤错切变换

⑥进行旋转变换

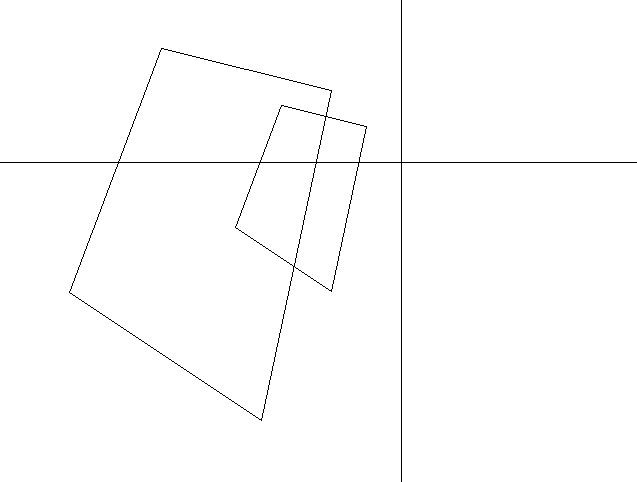
⑦平移操作

⑧比例变换操作

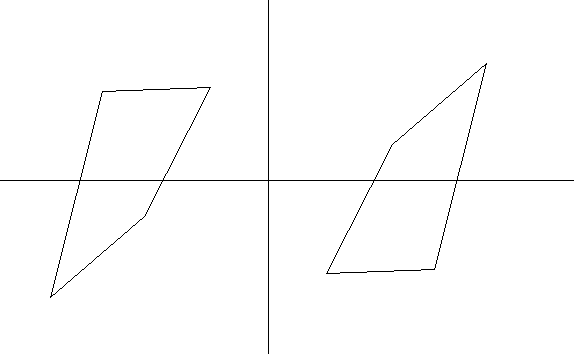
⑨关于原点的反射

⑩对称变换

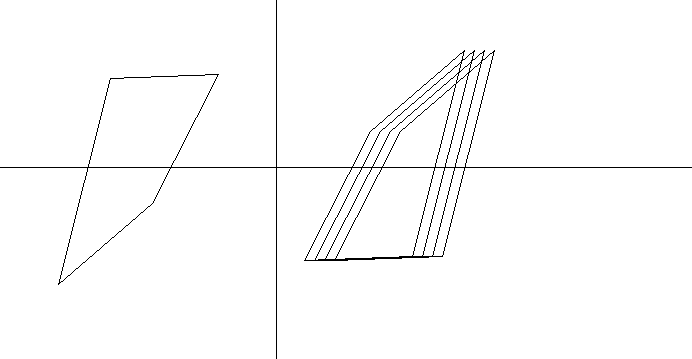
1. 实验结果及分析
2. 比例变换：



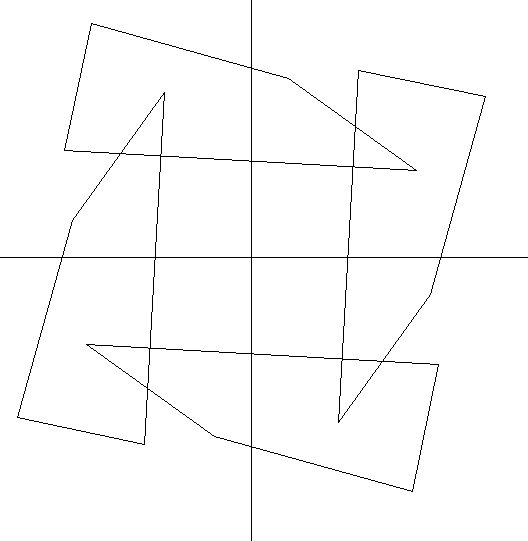
②对称（包含原点、x、y轴）：



1. 平移：



1. 旋转：



**5、附录**

**代码如下：**

//构建坐标系

void CcgTest1View::buildCoordinate()

{

type = BuildCoordinate;

RedrawWindow();

}

//取点

void CcgTest1View::drawPol()

{

finish = false;

type = DrawPol;

points.clear();

}

//平移

void CcgTest1View::moveUp()

{

CDC \*pDC = GetWindowDC();

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

points[i].y -= 10;

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::moveDown()

{

CDC \*pDC = GetWindowDC();

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

points[i].y += 10;

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::moveLeft()

{

CDC \*pDC = GetWindowDC();

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

points[i].x -= 10;

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::moveRight()

{

CDC \*pDC = GetWindowDC();

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

points[i].x += 10;

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]);

}

}

//平移

//图形变换部分

//对称部分

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::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]);

}

}

//对称部分

//旋转

//坐标原点旋转

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]);

}

}

//坐标原点旋转

//缩小

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]);

}

}

//放大

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);

}

//基于三角形（多边形）顶点的旋转

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

}

//顺时针

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);

}

void CcgTest1View::OnTriangleP2()

{

// TODO: 在此添加命令处理程序代码

}

void CcgTest1View::OnTriangleP3()

{

// TODO: 在此添加命令处理程序代码

}

//错切变化

//水平错切

void CcgTest1View::HorizontalCut()

{

CDC\* pDC = GetWindowDC();

double shx = 0.5;

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

// 计算变换后的 x 坐标

double newX = points[j].x + shx \* points[j].y;

points[j].x = newX;

}

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::VerticalCut()

{

CDC\* pDC = GetWindowDC();

double shy = 0.5;

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

// 计算变换后的 x 坐标

double newY = points[j].y + shy \* points[j].x;

points[j].y = newY;

}

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

}

项目2图形的区域填充

1. 问题

开发带有可视化图形交互界面的程序，实现各种图形的填充。

1. 模块设计

该实验程序包含以下模块：

①自己鼠标取点

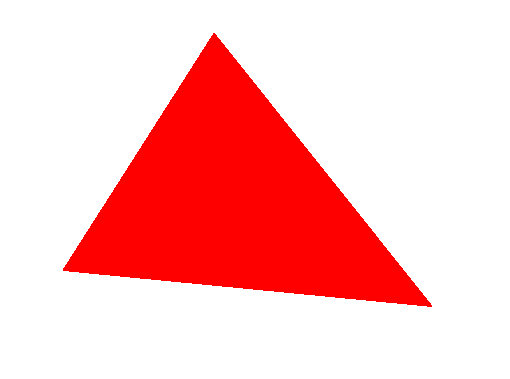
②画出去点后的图形

③对图形进行填充

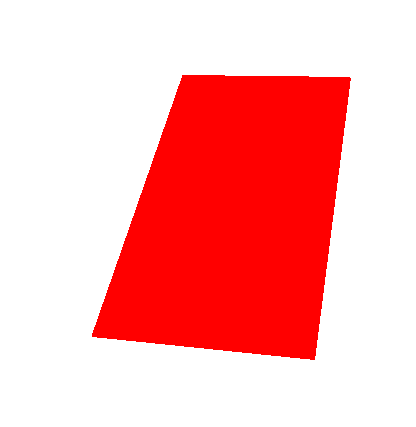
④填充点击事件

1. 实验结果及分析

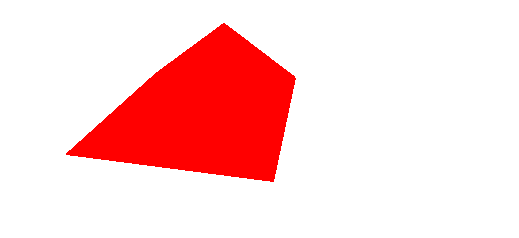
三角形：



四边形：



五边形：



1. 附录

代码如下：

//多边形的绘画和填充

void CcgTest1View::OnPolygon()

{

finish = false;

points.clear();

type = DrawPolygon;

}

void CcgTest1View::OnFillpolygon()

{

CDC \*pDC = GetWindowDC();

double y = 0.5;//扫描算法

for (; y < 1000; y++)

{

vector<int> c;

for (int i = 0; i < m\_pointNum; i++)

{

int t;

if (i < m\_pointNum - 1)

t = i + 1;

else

t = 0;

CPoint p1 = points[i];

CPoint p2 = points[t];

if ((y<p1.y&&y>p2.y) || (y > p1.y&&y < p2.y))

{

int x = (y - p2.y)\*(p1.x - p2.x) / (p1.y - p2.y) + p2.x;

c.push\_back(x);

}

}

sort(c.begin(), c.end());

for (int i = 0, j = 1; j < c.size(); i += 2, j += 2)

{

for (int x = c[i]; x <= c[j]; x++)

pDC->SetPixel(CPoint(x, (int)y), RGB(255, 0, 0));

}

}

}

void CcgTest1View::OnLButtonDown(UINT nFlags, CPoint point)

{

CDC \*pDC = GetWindowDC();

if (type == DrawPolygon)

{

points.push\_back(point);

m\_pointNum++;

}

else if (type == DrawLine)

{

if (!finish)

{

line = new pair<CPoint, CPoint>;

line->first = point;

finish = true;

}

else

{

line->second = point;

finish = false;

CPen pen(PS\_SOLID, 1, RGB(255, 0, 0));

CPen\* pOldPen = pDC->SelectObject(&pen);

pDC->MoveTo(line->first);

pDC->LineTo(line->second);

pDC->SelectObject(pOldPen);

lines.push\_back(line);

}

}

else if (type == DrawRect)

{

if (!finish)

{

rect.first = point;

finish = true;

}

else

{

rect.second = point;

pDC->Rectangle(rect.first.x, rect.first.y, rect.second.x, rect.second.y);

type = STOP;

}

}

else if (type == BuildCoordinate)

{

cx = point.x;

cy = point.y;

pDC->MoveTo(cx, 0);

pDC->LineTo(cx, 2000);

pDC->MoveTo(0, cy);

pDC->LineTo(2000, cy);

type = STOP;

}

else if (type==DrawPol)

{

points.push\_back(point);

}

CView::OnLButtonDown(nFlags, point);

}

void CcgTest1View::OnRButtonDown(UINT nFlags, CPoint point)

{

CDC \*pDC = GetWindowDC();

if (type == DrawPolygon)

{

int i;

for (i = 0; i < m\_pointNum; i++)

{

int t;

if (i < m\_pointNum - 1)

t = i + 1;

else

t = 0;

drawLineMid(points[i], points[t]);

}

type = STOP;

}

else if (type == DrawPol)

{

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]);

}

type = STOP;

}

CView::OnRButtonDown(nFlags, point);

}

void CcgTest1View::drawLineMid(CPoint p1, CPoint p2)

{

CDC \*pDC = GetWindowDC();

if (p1.x > p2.x)

{

CPoint t = p1;

p1 = p2;

p2 = t;

}

bool convert\_y = false, convert\_xy = false;

if (p1.y > p2.y)

{

convert\_y = true;

p1.y = -p1.y;

p2.y = -p2.y;

}

if (p1.x == p2.x)

{

for (int i = p1.y; i <= p2.y; i++)

pDC->SetPixel(CPoint(p1.x, i), RGB(255, 0, 0));

return;

}

if (p1.y == p2.y)

{

for (int i = p1.x; i <= p2.x; i++)

pDC->SetPixel(CPoint(i, p1.y), RGB(255, 0, 0));

return;

}

if (p2.y - p1.y > p2.x - p1.x)

{

convert\_xy = true;

int t = p1.x;

p1.x = p1.y;

p1.y = t;

t = p2.x;

p2.x = p2.y;

p2.y = t;

}

int a = p1.y - p2.y;

int b = p2.x - p1.x;

int d = 2 \* a + b;

int deta1 = 2 \* a;

int deta2 = 2 \* (a + b);

int x = p1.x, y = p1.y, px = x, py = y;

while (x < p2.x)

{

px = x;

py = y;

if (convert\_xy)

{

int t = px;

px = py;

py = t;

}

if (convert\_y)

{

py = -py;

}

pDC->SetPixel(CPoint(px, py), RGB(255, 0, 0));

if (d < 0)

{

x = x + 1;

y = y + 1;

d = d + deta2;

}

else

{

x = x + 1;

d = d + deta1;

}

}

}

void CcgTest1View::OnChangeDc()

{

// TODO: 在此添加命令处理程序代码

CDC \*pDC = GetWindowDC();

pDC->LineTo(-100, 100);

}

项目3曲线曲面的造型

1. 问题

开发具有可视化界面的软件，实现以下功能：（1）由鼠标在二维平面下选取一组n个控制点；（2）由该组控制点绕y轴旋转，得到三维空间中1080n个控制点；（3）由该组控制点每个由经线和纬线组成的小四边形顶点计算B样条曲面，得到3维几何造型。

1. 模块设计

该实验程序包含以下模块：

①用来计算出所有的控制点

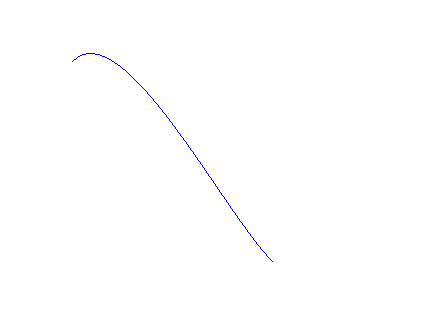
②用来记录每一个面的控制点

③进行斜投影并画出网格图

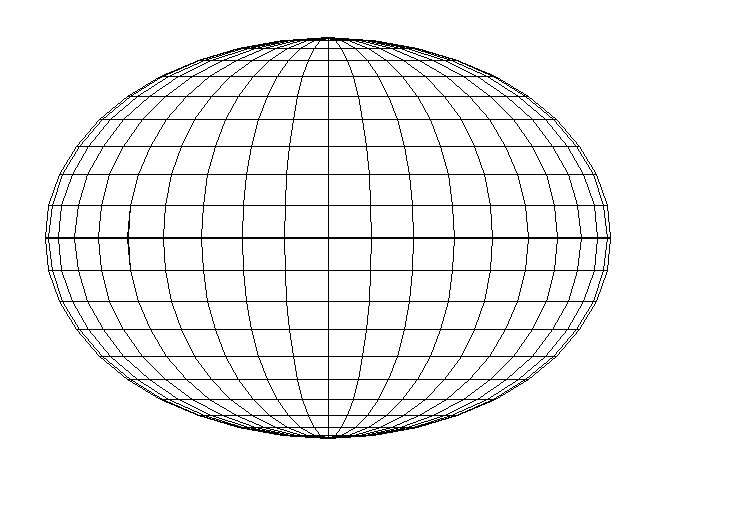
④求出双三次Bezier曲面上的点

1. 给出分块矩阵每个面分时使用双三次Bezier曲面
2. 实验结果及分析

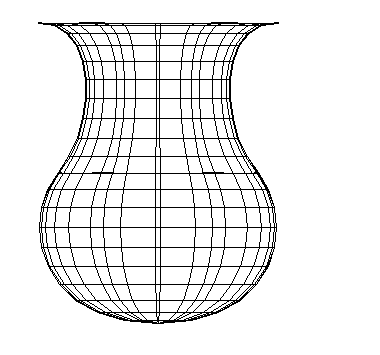
曲线



给定一个球的半径圆心，计算出他的控制点，再画出所有的双三次b样条曲面:



花瓶：



1. 附录

void CcgTest1View::SetBezierPoint()

{

double r = 200;//球的半径

const double m = 0.5523;//球半径计算的魔术常数

//第一卦限控制点

Vertex[0][0] = 0.0, Vertex[0][1] = r, Vertex[0][2] = 0.0;

Vertex[1][0] = 0.0, Vertex[1][1] = r, Vertex[1][2] = m \* r;

Vertex[2][0] = 0.0, Vertex[2][1] = m \* r, Vertex[2][2] = r;

Vertex[3][0] = 0.0, Vertex[3][1] = 0.0, Vertex[3][2] = r;

Vertex[4][0] = m \* m \* r, Vertex[4][1] = r, Vertex[4][2] = m \* r;

Vertex[5][0] = m \* r, Vertex[5][1] = m \* r, Vertex[5][2] = r;

Vertex[6][0] = m \* r, Vertex[6][1] = 0, Vertex[6][2] = r;

Vertex[7][0] = m \* r, Vertex[7][1] = r, Vertex[7][2] = m \* m \* r;

Vertex[8][0] = r, Vertex[8][1] = m \* r, Vertex[8][2] = m \* r;

Vertex[9][0] = r, Vertex[9][1] = 0.0, Vertex[9][2] = m \* r;

Vertex[10][0] = m \* r, Vertex[10][1] = r, Vertex[10][2] = 0.0;

Vertex[11][0] = r, Vertex[11][1] = m \* r, Vertex[11][2] = 0.0;

Vertex[12][0] = r, Vertex[12][1] = 0.0, Vertex[12][2] = 0.0;

//第二卦限控制点

Vertex[13][0] = m \* r, Vertex[13][1] = r, Vertex[13][2] = -m \* m \* r;

Vertex[14][0] = r, Vertex[14][1] = m \* r, Vertex[14][2] = -m \* r;

Vertex[15][0] = r, Vertex[15][1] = 0.0, Vertex[15][2] = -m \* r;

Vertex[16][0] = m \* m \* r, Vertex[16][1] = r, Vertex[16][2] = -m \* r;

Vertex[17][0] = m \* r, Vertex[17][1] = m \* r, Vertex[17][2] = -r;

Vertex[18][0] = m \* r, Vertex[18][1] = 0.0, Vertex[18][2] = -r;

Vertex[19][0] = 0.0, Vertex[19][1] = r, Vertex[19][2] = -m \* r;

Vertex[20][0] = 0.0, Vertex[20][1] = m \* r, Vertex[20][2] = -r;

Vertex[21][0] = 0.0, Vertex[21][1] = 0.0, Vertex[21][2] = -r;

//第三卦限控制点

Vertex[22][0] = -m \* m \* r, Vertex[22][1] = r, Vertex[22][2] = -m \* r;

Vertex[23][0] = -m \* r, Vertex[23][1] = m \* r, Vertex[23][2] = -r;

Vertex[24][0] = -m \* r, Vertex[24][1] = 0.0, Vertex[24][2] = -r;

Vertex[25][0] = -m \* r, Vertex[25][1] = r, Vertex[25][2] = -m \* m \* r;

Vertex[26][0] = -r, Vertex[26][1] = m \* r, Vertex[26][2] = -m \* r;

Vertex[27][0] = -r, Vertex[27][1] = 0.0, Vertex[27][2] = -m \* r;

Vertex[28][0] = -m \* r, Vertex[28][1] = r, Vertex[28][2] = 0.0;

Vertex[29][0] = -r, Vertex[29][1] = m \* r, Vertex[29][2] = 0.0;

Vertex[30][0] = -r, Vertex[30][1] = 0.0, Vertex[30][2] = 0.0;

//第四卦限控制点

Vertex[31][0] = -m \* r, Vertex[31][1] = r, Vertex[31][2] = m \* m \* r;

Vertex[32][0] = -r, Vertex[32][1] = m \* r, Vertex[32][2] = m \* r;

Vertex[33][0] = -r, Vertex[33][1] = 0.0, Vertex[33][2] = m \* r;

Vertex[34][0] = -m \* m \* r, Vertex[34][1] = r, Vertex[34][2] = m \* r;

Vertex[35][0] = -m \* r, Vertex[35][1] = m \* r, Vertex[35][2] = r;

Vertex[36][0] = -m \* r, Vertex[36][1] = 0.0, Vertex[36][2] = r;

//第五卦限控制点

Vertex[37][0] = 0.0, Vertex[37][1] = -m \* r, Vertex[37][2] = r;

Vertex[38][0] = 0.0, Vertex[38][1] = -r, Vertex[38][2] = m \* r;

Vertex[39][0] = m \* r, Vertex[39][1] = -m \* r, Vertex[39][2] = r;

Vertex[40][0] = m \* m \* r, Vertex[40][1] = -r, Vertex[40][2] = m \* r;

Vertex[41][0] = r, Vertex[41][1] = -m \* r, Vertex[41][2] = m \* r;

Vertex[42][0] = m \* r, Vertex[42][1] = -r, Vertex[42][2] = m \* m \* r;

Vertex[43][0] = r, Vertex[43][1] = -m \* r, Vertex[43][2] = 0.0;

Vertex[44][0] = m \* r, Vertex[44][1] = -r, Vertex[44][2] = 0.0;

//第六卦限控制点

Vertex[45][0] = r, Vertex[45][1] = -m \* r, Vertex[45][2] = -m \* r;

Vertex[46][0] = m \* r, Vertex[46][1] = -r, Vertex[46][2] = -m \* m \* r;

Vertex[47][0] = m \* r, Vertex[47][1] = -m \* r, Vertex[47][2] = -r;

Vertex[48][0] = m \* m \* r, Vertex[48][1] = -r, Vertex[48][2] = -m \* r;

Vertex[49][0] = 0.0, Vertex[49][1] = -m \* r, Vertex[49][2] = -r;

Vertex[50][0] = 0.0, Vertex[50][1] = -r, Vertex[50][2] = -m \* r;

//第七卦限控制点

Vertex[51][0] = -m \* r, Vertex[51][1] = -m \* r, Vertex[51][2] = -r;

Vertex[52][0] = -m \* m \* r, Vertex[52][1] = -r, Vertex[52][2] = -m \* r;

Vertex[53][0] = -r, Vertex[53][1] = -m \* r, Vertex[53][2] = -m \* r;

Vertex[54][0] = -m \* r, Vertex[54][1] = -r, Vertex[54][2] = -m \* m \* r;

Vertex[55][0] = -r, Vertex[55][1] = -m \* r, Vertex[55][2] = 0.0;

Vertex[56][0] = -m \* r, Vertex[56][1] = -r, Vertex[56][2] = 0.0;

//第八卦限控制点

Vertex[57][0] = -r, Vertex[57][1] = -m \* r, Vertex[57][2] = m \* r;

Vertex[58][0] = -m \* r, Vertex[58][1] = -r, Vertex[58][2] = m \* m \* r;

Vertex[59][0] = -m \* r, Vertex[59][1] = -m \* r, Vertex[59][2] = r;

Vertex[60][0] = -m \* m \* r, Vertex[60][1] = -r, Vertex[60][2] = m \* r;

Vertex[61][0] = 0.0, Vertex[61][1] = -r, Vertex[61][2] = 0.0;

}

void CcgTest1View::Getv()

{

//第一卦限的曲面片

F1[0][0][0] = 0, F1[0][0][1] = 1, F1[0][0][2] = 2, F1[0][0][3] = 3;

F1[0][1][0] = 0, F1[0][1][1] = 4, F1[0][1][2] = 5, F1[0][1][3] = 6;

F1[0][2][0] = 0, F1[0][2][1] = 7, F1[0][2][2] = 8, F1[0][2][3] = 9;

F1[0][3][0] = 0, F1[0][3][1] = 10, F1[0][3][2] = 11, F1[0][3][3] = 12;

//第二卦限的曲面片

F1[1][0][0] = 0, F1[1][0][1] = 10, F1[1][0][2] = 11, F1[1][0][3] = 12;

F1[1][1][0] = 0, F1[1][1][1] = 13, F1[1][1][2] = 14, F1[1][1][3] = 15;

F1[1][2][0] = 0, F1[1][2][1] = 16, F1[1][2][2] = 17, F1[1][2][3] = 18;

F1[1][3][0] = 0, F1[1][3][1] = 19, F1[1][3][2] = 20, F1[1][3][3] = 21;

//第三卦限的曲面片

F1[2][0][0] = 0, F1[2][0][1] = 19, F1[2][0][2] = 20, F1[2][0][3] = 21;

F1[2][1][0] = 0, F1[2][1][1] = 22, F1[2][1][2] = 23, F1[2][1][3] = 24;

F1[2][2][0] = 0, F1[2][2][1] = 25, F1[2][2][2] = 26, F1[2][2][3] = 27;

F1[2][3][0] = 0, F1[2][3][1] = 28, F1[2][3][2] = 29, F1[2][3][3] = 30;

//第四卦限的曲面片

F1[3][0][0] = 0, F1[3][0][1] = 28, F1[3][0][2] = 29, F1[3][0][3] = 30;

F1[3][1][0] = 0, F1[3][1][1] = 31, F1[3][1][2] = 32, F1[3][1][3] = 33;

F1[3][2][0] = 0, F1[3][2][1] = 34, F1[3][2][2] = 35, F1[3][2][3] = 36;

F1[3][3][0] = 0, F1[3][3][1] = 1, F1[3][3][2] = 2, F1[3][3][3] = 3;

//第五卦限的曲面片

F1[4][0][0] = 3, F1[4][0][1] = 37, F1[4][0][2] = 38, F1[4][0][3] = 61;

F1[4][1][0] = 6, F1[4][1][1] = 39, F1[4][1][2] = 40, F1[4][1][3] = 61;

F1[4][2][0] = 9, F1[4][2][1] = 41, F1[4][2][2] = 42, F1[4][2][3] = 61;

F1[4][3][0] = 12, F1[4][3][1] = 43, F1[4][3][2] = 44, F1[4][3][3] = 61;

//第六卦限的曲面片

F1[5][0][0] = 12, F1[5][0][1] = 43, F1[5][0][2] = 44, F1[5][0][3] = 61;

F1[5][1][0] = 15, F1[5][1][1] = 45, F1[5][1][2] = 46, F1[5][1][3] = 61;

F1[5][2][0] = 18, F1[5][2][1] = 47, F1[5][2][2] = 48, F1[5][2][3] = 61;

F1[5][3][0] = 21, F1[5][3][1] = 49, F1[5][3][2] = 50, F1[5][3][3] = 61;

//第七卦限的曲面片

F1[6][0][0] = 21, F1[6][0][1] = 49, F1[6][0][2] = 50, F1[6][0][3] = 61;

F1[6][1][0] = 24, F1[6][1][1] = 51, F1[6][1][2] = 52, F1[6][1][3] = 61;

F1[6][2][0] = 27, F1[6][2][1] = 53, F1[6][2][2] = 54, F1[6][2][3] = 61;

F1[6][3][0] = 30, F1[6][3][1] = 55, F1[6][3][2] = 56, F1[6][3][3] = 61;

//第八卦限的曲面片

F1[7][0][0] = 30, F1[7][0][1] = 55, F1[7][0][2] = 56, F1[7][0][3] = 61;

F1[7][1][0] = 33, F1[7][1][1] = 57, F1[7][1][2] = 58, F1[7][1][3] = 61;

F1[7][2][0] = 36, F1[7][2][1] = 59, F1[7][2][2] = 60, F1[7][2][3] = 61;

F1[7][3][0] = 3, F1[7][3][1] = 37, F1[7][3][2] = 38, F1[7][3][3] = 61;

}

void CcgTest1View::away()

{

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 / 180), b[2][1] = -sin(45 / 180), b[2][2] = 0, b[2][3] = 0;

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

CDC\* pDC = GetDC();

double uv2[11][11][2];

for (int i = 0; i < 11; i++) {

for (int j = 0; j < 11; j++) {

uv2[i][j][0] = uv[i][j][0] \* b[0][0] + uv[i][j][1] \* b[1][0] + uv[i][j][2] \* b[2][0] + b[3][0];

uv2[i][j][1] = uv[i][j][0] \* b[0][1] + uv[i][j][1] \* b[1][1] + uv[i][j][2] \* b[2][1] + b[3][1];

}

}

for (int i = 0; i < 11; i++) {

//连接v方向

pDC->MoveTo(uv2[i][0][0] + 400, 300 - uv2[i][0][1]);

for (int j = 1; j < 11; j++) {

pDC->LineTo(uv2[i][j][0] + 400, 300 - uv2[i][j][1]);

}

//连接u方向

pDC->MoveTo(uv2[0][i][0] + 400, 300 - uv2[0][i][1]);

for (int j = 1; j < 11; j++) {

pDC->LineTo(uv2[j][i][0] + 400, 300 - uv2[j][i][1]);

}

}

ReleaseDC(pDC);

}

void CcgTest1View::Dbezier()

{

int i = 0;

for (double u = 0; u <= 1; u += 0.1) {

double u1 = (1 - u) \* (1 - u) \* (1 - u);

double u2 = 3 \* (1 - u) \* (1 - u) \* u;

double u3 = 3 \* (1 - u) \* u \* u;

double u4 = u \* u \* u;

double q[4][3];

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

q[k][0] = F[0][k][0] \* u1 + F[1][k][0] \* u2 + F[2][k][0] \* u3 + F[3][k][0] \* u4;

q[k][1] = F[0][k][1] \* u1 + F[1][k][1] \* u2 + F[2][k][1] \* u3 + F[3][k][1] \* u4;

q[k][2] = F[0][k][2] \* u1 + F[1][k][2] \* u2 + F[2][k][2] \* u3 + F[3][k][2] \* u4;

}

int j = 0;

for (double v = 0; v <= 1; v += 0.1) {

double v1 = (1 - v) \* (1 - v) \* (1 - v);

double v2 = 3 \* (1 - v) \* (1 - v) \* v;

double v3 = 3 \* (1 - v) \* v \* v;

double v4 = v \* v \* v;

uv[i][j][0] = q[0][0] \* v1 + q[1][0] \* v2 + q[2][0] \* v3 + q[3][0] \* v4;

uv[i][j][1] = q[0][1] \* v1 + q[1][1] \* v2 + q[2][1] \* v3 + q[3][1] \* v4;

uv[i][j][2] = q[0][2] \* v1 + q[1][2] \* v2 + q[2][2] \* v3 + q[3][2] \* v4;

j++;

}

i++;

j = 0;

}

//求出曲面上的点后，进行连接斜投影连接

away();

}

//分块点矩阵

void CcgTest1View::continuity()

{

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

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

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

F[j][k][0] = Vertex[F1[i][j][k]][0], F[j][k][1] = Vertex[F1[i][j][k]][1], F[j][k][2] = Vertex[F1[i][j][k]][2];

}

}

Dbezier();

}

}

/\*

//画坐标轴

void CcgTest1View::OnBnClickedButton1()

{

ycoordinate();

// TODO: 在此添加控件通知处理程序代码

}

\*/

void CcgTest1View::BezierLine()

{

//基函数系数

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

b[1][0] = 3, b[1][1] = -6, b[1][2] = 3, b[1][3] = 0;

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

b[3][0] = 1, b[3][1] = 4, b[3][2] = 1, b[3][3] = 0;

SetBezierPoint();

Getv();

continuity();

/\*F[0][0][0] = 20, F[0][0][1] = 0, F[0][0][2] = 200;

F[0][1][0] = 0, F[0][1][1] = 100, F[0][1][2] = 150;

F[0][2][0] = -130, F[0][2][1] = 100, F[0][2][2] = 50;

F[0][3][0] = -250, F[0][3][1] = 50, F[0][3][2] = 0;

F[1][0][0] = 100, F[1][0][1] = 100, F[1][0][2] = 150;

F[1][1][0] = 30, F[1][1][1] = 100, F[1][1][2] = 100;

F[1][2][0] = -40, F[1][2][1] = 100, F[1][2][2] = 50;

F[1][3][0] = -110, F[1][3][1] = 100, F[1][3][2] = 0;

F[2][0][0] = 280, F[2][0][1] = 90, F[2][0][2] = 140;

F[2][1][0] = 110, F[2][1][1] = 120, F[2][1][2] = 80;

F[2][2][0] = 30, F[2][2][1] = 130, F[2][2][2] = 30;

F[2][3][0] = -100, F[2][3][1] = 150, F[2][3][2] = -50;

F[3][0][0] = 350, F[3][0][1] =30, F[3][0][2] = 150;

F[3][1][0] = 200, F[3][1][1] = 150, F[3][1][2] = 50;

F[3][2][0] = 50, F[3][2][1] =200, F[3][2][2] = 0;

F[3][3][0] = 0, F[3][3][1] = 100, F[3][3][2] = -70;

Dbezier();\*/

// TODO: 在此添加控件通知处理程序代码

}

void CcgTest1View::setpoint()

{

double m = 0.5523;

FVertex[0][0] = 120, FVertex[0][1] = 300, FVertex[0][2] = 0;

FVertex[1][0] = 60, FVertex[1][1] = 300, FVertex[1][2] = 0;

FVertex[2][0] = 60, FVertex[2][1] = 200, FVertex[2][2] = 0;

FVertex[3][0] = 100, FVertex[3][1] = 150, FVertex[3][2] = 0;

FVertex[4][0] = 140, FVertex[4][1] = 100, FVertex[4][2] = 0;

FVertex[5][0] = 120, FVertex[5][1] = 0, FVertex[5][2] = 0;

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

FVertex[7][0] = 120 \* cos(30 / 180), FVertex[7][1] = 300, FVertex[7][2] = 120 \* sin(30 / 180);

FVertex[8][0] = 60 \* cos(30 / 180), FVertex[8][1] = 300, FVertex[8][2] = 60 \* sin(30 / 180);

FVertex[9][0] = 60 \* cos(30 / 180), FVertex[9][1] = 200, FVertex[9][2] = 60 \* sin(30 / 180);

FVertex[10][0] = 100 \* cos(30 / 180), FVertex[10][1] = 150, FVertex[10][2] = 100 \* sin(30 / 180);

FVertex[11][0] = 140 \* cos(30 / 180), FVertex[11][1] = 100, FVertex[11][2] = 140 \* sin(30 / 180);

FVertex[12][0] = 120 \* cos(30 / 180), FVertex[12][1] = 0, FVertex[12][2] = 120 \* sin(30 / 180);

FVertex[13][0] = 0, FVertex[13][1] = 0, FVertex[13][2] = 0;

FVertex[14][0] = 120 \* cos(60 / 180), FVertex[14][1] = 300, FVertex[14][2] = 120 \* sin(60 / 180);

FVertex[15][0] = 60 \* cos(60 / 180), FVertex[15][1] = 300, FVertex[15][2] = 60 \* sin(60 / 180);

FVertex[16][0] = 60 \* cos(60 / 180), FVertex[16][1] = 200, FVertex[16][2] = 60 \* sin(60 / 180);

FVertex[17][0] = 100 \* cos(60 / 180), FVertex[17][1] = 150, FVertex[17][2] = 100 \* sin(60 / 180);

FVertex[18][0] = 140 \* cos(60 / 180), FVertex[18][1] = 100, FVertex[18][2] = 140 \* sin(60 / 180);

FVertex[19][0] = 120 \* cos(60 / 180), FVertex[19][1] = 0, FVertex[19][2] = 120 \* sin(60 / 180);

FVertex[20][0] = 0, FVertex[20][1] = 0, FVertex[20][2] = 0;

FVertex[21][0] = 0, FVertex[21][1] = 300, FVertex[21][2] = 120;

FVertex[22][0] = 0, FVertex[22][1] = 300, FVertex[22][2] = 60;

FVertex[23][0] = 0, FVertex[23][1] = 200, FVertex[23][2] = 60;

FVertex[24][0] = 0, FVertex[24][1] = 150, FVertex[24][2] = 100;

FVertex[25][0] = 0, FVertex[25][1] = 100, FVertex[25][2] = 140;

FVertex[26][0] = 0, FVertex[26][1] = 0, FVertex[26][2] = 120;

FVertex[27][0] = 0, FVertex[27][1] = 0, FVertex[27][2] = 0;

for (int i = 28; i < 56; i++) {

FVertex[i][0] = -FVertex[i - 28][0];

FVertex[i][1] = FVertex[i - 28][1];

FVertex[i][2] = FVertex[i - 28][2];

}

for (int i = 56; i < 84; i++) {

FVertex[i][0] = FVertex[i - 56][0];

FVertex[i][1] = FVertex[i - 56][1];

FVertex[i][2] = -FVertex[i - 56][2];

}

for (int i = 84; i < 112; i++) {

FVertex[i][0] = -FVertex[i - 84][0];

FVertex[i][1] = FVertex[i - 84][1];

FVertex[i][2] = -FVertex[i - 84][2];

}

}

void CcgTest1View::FGetv()

{

FF1[0][0][0] = 0, FF1[0][0][1] = 1, FF1[0][0][2] = 2, FF1[0][0][3] = 3;

FF1[0][1][0] = 7, FF1[0][1][1] = 8, FF1[0][1][2] = 9, FF1[0][1][3] = 10;

FF1[0][2][0] = 14, FF1[0][2][1] = 15, FF1[0][2][2] = 16, FF1[0][2][3] = 17;

FF1[0][3][0] = 21, FF1[0][3][1] = 22, FF1[0][3][2] = 23, FF1[0][3][3] = 24;

FF1[1][0][0] = 3, FF1[1][0][1] = 4, FF1[1][0][2] = 5, FF1[1][0][3] = 6;

FF1[1][1][0] = 10, FF1[1][1][1] = 11, FF1[1][1][2] = 12, FF1[1][1][3] = 13;

FF1[1][2][0] = 17, FF1[1][2][1] = 18, FF1[1][2][2] = 19, FF1[1][2][3] = 20;

FF1[1][3][0] = 24, FF1[1][3][1] = 25, FF1[1][3][2] = 26, FF1[1][3][3] = 27;

for (int i = 1; i < 4; i++) {

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

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

FF1[i \* 2][j][k] = FF1[0][j][k] + i \* 28;

FF1[i \* 2 + 1][j][k] = FF1[1][j][k] + i \* 28;

}

}

}

}

void CcgTest1View::Faway()

{

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

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

Fb[2][0] = -cos(45 / 180), Fb[2][1] = -sin(45 / 180), Fb[2][2] = 0, Fb[2][3] = 0;

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

CDC\* pDC = GetDC();

double uv2[11][11][2];

for (int i = 0; i < 11; i++) {

for (int j = 0; j < 11; j++) {

uv2[i][j][0] = uv[i][j][0] \* Fb[0][0] + uv[i][j][1] \* Fb[1][0] + uv[i][j][2] \* Fb[2][0] + Fb[3][0];

uv2[i][j][1] = uv[i][j][0] \* Fb[0][1] + uv[i][j][1] \* Fb[1][1] + uv[i][j][2] \* Fb[2][1] + Fb[3][1];

}

}

for (int i = 0; i < 11; i++) {

//连接v方向

pDC->MoveTo(uv2[i][0][0] + 400, 350 - uv2[i][0][1]);

for (int j = 1; j < 11; j++) {

pDC->LineTo(uv2[i][j][0] + 400, 350 - uv2[i][j][1]);

}

//连接u方向

pDC->MoveTo(uv2[0][i][0] + 400, 350 - uv2[0][i][1]);

for (int j = 1; j < 11; j++) {

pDC->LineTo(uv2[j][i][0] + 400, 350 - uv2[j][i][1]);

}

}

ReleaseDC(pDC);

}

void CcgTest1View::FDbezier()

{

int i = 0;

for (double u = 0; u <= 1; u += 0.1) {

double u1 = (1 - u) \* (1 - u) \* (1 - u);

double u2 = 3 \* (1 - u) \* (1 - u) \* u;

double u3 = 3 \* (1 - u) \* u \* u;

double u4 = u \* u \* u;

double q[4][3];

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

q[k][0] = F[0][k][0] \* u1 + F[1][k][0] \* u2 + F[2][k][0] \* u3 + F[3][k][0] \* u4;

q[k][1] = F[0][k][1] \* u1 + F[1][k][1] \* u2 + F[2][k][1] \* u3 + F[3][k][1] \* u4;

q[k][2] = F[0][k][2] \* u1 + F[1][k][2] \* u2 + F[2][k][2] \* u3 + F[3][k][2] \* u4;

}

int j = 0;

for (double v = 0; v <= 1; v += 0.1) {

double v1 = (1 - v) \* (1 - v) \* (1 - v);

double v2 = 3 \* (1 - v) \* (1 - v) \* v;

double v3 = 3 \* (1 - v) \* v \* v;

double v4 = v \* v \* v;

uv[i][j][0] = q[0][0] \* v1 + q[1][0] \* v2 + q[2][0] \* v3 + q[3][0] \* v4;

uv[i][j][1] = q[0][1] \* v1 + q[1][1] \* v2 + q[2][1] \* v3 + q[3][1] \* v4;

uv[i][j][2] = q[0][2] \* v1 + q[1][2] \* v2 + q[2][2] \* v3 + q[3][2] \* v4;

j++;

}

i++;

j = 0;

}

//求出曲面上的点后，进行连接斜投影连接

Faway();

}

void CcgTest1View::Fcontinuity()

{

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

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

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

F[j][k][0] = FVertex[FF1[i][j][k]][0], F[j][k][1] = FVertex[FF1[i][j][k]][1], F[j][k][2] = FVertex[FF1[i][j][k]][2];

}

}

FDbezier();

}

}

void CcgTest1View::BezierSurface()

{

//基函数系数

Fb[0][0] = -1, Fb[0][1] = 3, Fb[0][2] = -3, Fb[0][3] = 1;

Fb[1][0] = 3, Fb[1][1] = -6, Fb[1][2] = 3, Fb[1][3] = 0;

Fb[2][0] = -3, Fb[2][1] = 0, Fb[2][2] = 3, Fb[2][3] = 0;

Fb[3][0] = 1, Fb[3][1] = 4, Fb[3][2] = 1, Fb[3][3] = 0;

setpoint();

FGetv();

Fcontinuity();

// TODO: 在此添加命令处理程序代码

}

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

}

//贝塞尔曲线部分