实验三 贝塞尔曲线与曲面

1. **贝塞尔曲线：使用三次贝塞尔曲线绘制圆**

    (1)　采用三次贝塞尔曲线绘制 1/4 圆弧（如图 1（a））

    (2)　采用三次贝塞尔曲线绘制近似圆（如图 1（b））

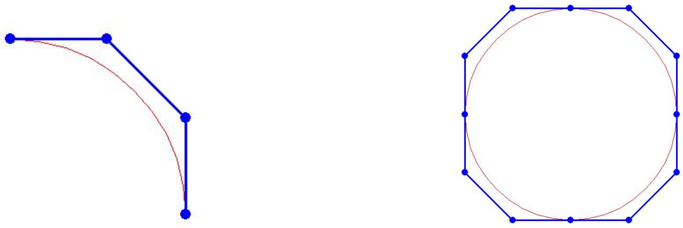

图 1 三次贝塞尔曲线（左：(a) 1/4 圆弧，右：(b) 近似圆）

【实验过程及编码】

（一）实验步骤

（1）设计三次贝塞尔曲线类 CBezierCurve

（2）设计圆类 CCircle

（3）初始化圆对象

（4）绘制圆

（二）实验编码

（1）设计三次贝塞尔曲线类 CBezierCurve

- CBezierCurve.h

```
#include"CP2.h"
class CBezierCurve  //三次贝塞尔曲线
{
public:
     CBezierCurve();
     virtual ~CBezierCurve();
     void ReadPoint(CP2 *P);    //读入控制点
     void DrawCurve(CDC *pDC); //绘制曲线
     void DrawPolygon(CDC* pDC); //绘制控制多边形
private:
     CP2 P[4]; //控制点数组
};
```

- CBezierCurve.cpp

```
#include "pch.h"
#include "CBezierCurve.h"
#define ROUND(d) int(d + 0.5)//四舍五入宏定义
CBezierCurve::CBezierCurve(void)
{
}
CBezierCurve::~CBezierCurve(void)
{
}
```

```
void CBezierCurve::ReadPoint(CP2* P)
{
    for (int i = 0; i < 4; i++)
        this->P[i] = P[i];
}
void CBezierCurve::DrawCurve(CDC* pDC)//de Casteljau 递推算法
{
    CPen NewPen, * pOldPen;
    NewPen.CreatePen(PS_SOLID, 1, RGB(255, 0, 0));
    pOldPen = pDC->SelectObject(&NewPen);
    CP2 p00 = P[0], p10 = P[1], p20 = P[2], p30 = P[3];
    CP2 p01, p11, p21, p02, p12, p03;
    double tStep = 0.1;//步长
    pDC->MoveTo(ROUND(P[0].x), ROUND(P[0].y));
    for (double t = 0.0; t < 1; t += tStep)
    {
        p01 = (1 - t) * p00 + t * p10;
        p11 = (1 - t) * p10 + t * p20;
        p21 = (1 - t) * p20 + t * p30;
        p02 = (1 - t) * p01 + t * p11;
        p12 = (1 - t) * p11 + t * p21;
        p03 = (1 - t) * p02 + t * p12;
        pDC->LineTo(ROUND(p03.x), ROUND(p03.y));
    }
    pDC->LineTo(ROUND(P[3].x), ROUND(P[3].y));
    pDC->SelectObject(pOldPen);
    NewPen.DeleteObject();
}
void CBezierCurve::DrawPolygon(CDC* pDC)//绘制控制多边形
{
    CPen pen(PS_SOLID, 3, RGB(0, 0, 255));
    CPen* pOldPen = pDC->SelectObject(&pen);
    CBrush brush(RGB(0, 0, 255));
    CBrush* pOldBrush = pDC->SelectObject(&brush);
    for (int i = 0; i < 4; i++)
    {
        if (0 == i)
        {
            pDC->MoveTo(ROUND(P[i].x), ROUND(P[i].y));
    pDC->Ellipse(ROUND(P[i].x) - 5, ROUND(P[i].y) - 5, ROUND(P[i].x) + 5, ROUND(P[i].y) + 5);
        }
        else
        {
            pDC->LineTo(ROUND(P[i].x), ROUND(P[i].y));
    pDC->Ellipse(ROUND(P[i].x) - 5, ROUND(P[i].y) - 5, ROUND(P[i].x) + 5, ROUND(P[i].y) + 5);
        }
    }
    pDC->SelectObject(pOldBrush);
    pDC->SelectObject(pOldPen);
}
```

（2）设计圆类 CCircle

➢ CCircle.h

```
#include"CBezierCurve.h"
class CCircle
{
public:
    CCircle(void);
    virtual ~CCircle(void);
    void ReadPoint(void);//读入控制点表
    CP2* GetVertexArrayName(void);//得到顶点数组名
    void Draw(CDC* pDC);//绘制曲线
private:
    CP2 P[12];//控制点数组
    CBezierCurve bezier[4];//曲线段
};
```

➢ CCircle.cpp

```
#include "pch.h"
#include "CCircle.h"
CCircle::CCircle(void)
{
}
CCircle::~CCircle(void)
{
}
CP2* CCircle::GetVertexArrayName(void)
{
    return   P;
}
void CCircle::ReadPoint(void)
{
    double m = 0.5523;
    P[0].x = 1, P[0].y = 0;
    P[1].x = 1, P[1].y = m;
    P[2].x = m, P[2].y = 1;
    P[3].x = 0, P[3].y = 1;
    P[4].x = -m, P[4].y = 1;
    P[5].x = -1, P[5].y = m;
    P[6].x = -1, P[6].y = 0;
    P[7].x = -1, P[7].y = -m;
    P[8].x = -m, P[8].y = -1;
    P[9].x = 0, P[9].y = -1;
    P[10].x = m, P[10].y = -1;
    P[11].x = 1, P[11].y = -m;
}
void CCircle::Draw(CDC* pDC)
{
    CP2 CtrP[4];//三次 Bezier 曲线控制点
    //第一段
    CtrP[0] = P[0], CtrP[1] = P[1], CtrP[2] = P[2], CtrP[3] = P[3];
    bezier[0].ReadPoint(CtrP);
    bezier[0].DrawCurve(pDC);
    bezier[0].DrawPolygon(pDC);
    //第二段
    CtrP[0] = P[3], CtrP[1] = P[4], CtrP[2] = P[5], CtrP[3] = P[6];
    bezier[1].ReadPoint(CtrP);
    bezier[1].DrawCurve(pDC);
    bezier[1].DrawPolygon(pDC);
    //第三段
    CtrP[0] = P[6], CtrP[1] = P[7], CtrP[2] = P[8], CtrP[3] = P[9];
    bezier[2].ReadPoint(CtrP);
    bezier[2].DrawCurve(pDC);
    bezier[2].DrawPolygon(pDC);
    //第四段
    CtrP[0] = P[9], CtrP[1] = P[10], CtrP[2] = P[11], CtrP[3] = P[0];
    bezier[3].ReadPoint(CtrP);
    bezier[3].DrawCurve(pDC);
    bezier[3].DrawPolygon(pDC);
}
```

（3）初始化圆对象

➢ CTestBezierCircleView.h

在类声明中加入两个数据成员：

```
    CCircle circle;
    CTransform2 transform;
```

➢ CTestBezierCircleView.cpp

```
CTestBezierCircleView::CTestBezierCircleView() noexcept
{
    // TODO: 在此处添加构造代码
    circle.ReadPoint();
    transform.SetMatrix(circle.GetVertexArrayName(), 12);
    double R = 200;
    transform.Scale(R, R);//对单位圆进行比例放大
}
```

（4）绘制圆

- ➢ CTestBezierCircleView.cpp

```
void CTestBezierCircleView::OnDraw(CDC* pDC)
{
	CTestBezierCircleDoc* pDoc = GetDocument();
	ASSERT_VALID(pDoc);
	if (!pDoc)
		return;
	// TODO: 在此处为本机数据添加绘制代码
	CRect rect;
	GetClientRect(&rect);//自定义二维坐标系
	pDC->SetMapMode(MM_ANISOTROPIC);
	pDC->SetWindowExt(rect.Width(), rect.Height());
	pDC->SetViewportExt(rect.Width(), -rect.Height());
	pDC->SetViewportOrg(rect.Width() / 2, rect.Height() / 2);
	circle.Draw(pDC); //绘制圆
}
```

2. 贝塞尔曲面：使用双三次贝塞尔曲面构造球面

将位于 xOy 平面内 x 轴正向的半圆绕 y 轴回转一圈（如图 2（a）所示），构造双三次贝塞尔球面（如图 2（b））。

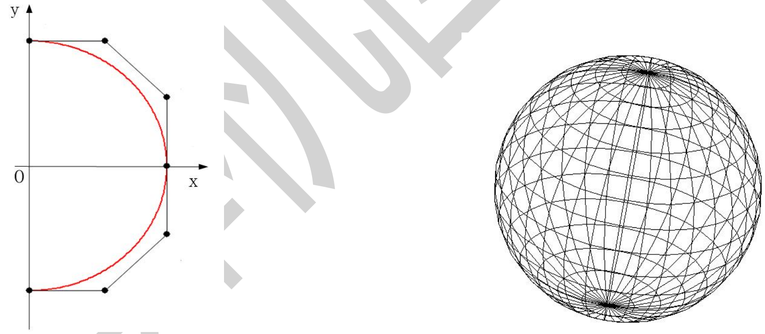

图 2 贝塞尔曲面（左：(a) 位于 xOy 平面内的半圆，右：(b) 双三次贝塞尔球面）

【实验过程及编码】

（一）实验步骤

（1）设计参数点类 CT2

（2）设计细分曲面类 CMesh

（3）设计双三次贝塞尔曲面类 CBezierPatch

（4）设计回转类 CRevolution

（5）初始化曲线段

（6）绘制线框球

（三）实验编码

（1）设计参数点类 CT2

- ➢ CT2.h

```
class CT2
{
public:
    CT2(void);
    virtual ~CT2(void);
    CT2(double u, double v);
    friend CT2 operator + (const CT2 &t0, const CT2 &t1);//运算符重载
    friend CT2 operator - (const CT2 &t0, const CT2 &t1);
    friend CT2 operator * (const CT2 &t, double scalar);
    friend CT2 operator * (double scalar, const CT2 &t);
    friend CT2 operator / (const CT2 &t,double scalar);
    friend CT2 operator += (CT2 &t0, CT2 &t1);
    friend CT2 operator -= (CT2 &t0, CT2 &t1);
    friend CT2 operator *= (CT2 &t, double scalar);
    friend CT2 operator /= (CT2 &t, double scalar);
public:
    double u, v;
}
```

- CT2.cpp

```
#include "CT2.h"
CT2::CT2(void)
{
    u = 0.0;
    v = 0.0;
}
CT2::CT2(double u, double v)
{
    this->u = u;
    this->v = v;
}
CT2::~CT2(void)
{
}
CT2 operator + (const CT2 &t0, const CT2 &t1)//和
{
    CT2 result;
    result.u = t0.u + t1.u;
    result.v = t0.v + t1.v;
    return result;
}
CT2 operator - (const CT2 &t0, const CT2 &t1)//差
{
    CT2 result;
    result.u = t0.u - t1.u;
    result.v = t0.v - t1.v;
    return result;
}
CT2 operator * (const CT2 &t, double scalar)//点和常量的积
{
    return CT2(t.u * scalar, t.v * scalar);
}
CT2 operator * (double scalar, const CT2 &t)//常量和点的积
{
    return CT2(t.u * scalar, t.v * scalar);
}
CT2 operator / (const CT2 &t,double scalar)//数除
{
    if(fabs(scalar) < 1e-4)
        scalar = 1.0;
    CT2 result;
    result.u = t.u/scalar;
    result.v = t.v/scalar;
    return result;
}
CT2 operator += (CT2 &t0, CT2 &t1)
{
    t0.u = t0.u + t1.u;
    t0.v = t0.v + t1.v;
    return t0;
}
CT2 operator -= (CT2 &t0, CT2 &t1)
{
    t0.u = t0.u - t1.u;
    t0.v = t0.v - t1.v;
    return t0;
}
CT2 operator *= (CT2 &t, double scalar)
{
    t.u = t.u * scalar;
    t.v = t.v * scalar;
    return t;
}
CT2 operator /= (CT2 &t, double scalar)
{
    if(fabs(scalar) < 1e-4)
        scalar = 1.0;
    t.u = t.u/scalar;
    t.v = t.v/scalar;
    return t;
}
```

（2）设计细分曲面类 CMesh

- CMesh.h

```
#include"CT2.h"
class CMesh
{
public:
    CMesh(void){}
    virtual ~CMesh(void){}
public:
    CT2 BL, BR, TR, TL;//四边形的 4 个角点坐标
};
```

（3）设计双三次贝塞尔曲面类 CBezierPatch

- CBezierPatch.h

```
#include"CMesh.h"
#include"CP3.h"
class CBezierPatch
{
public:
    CBezierPatch(void);
    virtual ~CBezierPatch(void);
    void ReadControlPoint(CP3 CtrPt[4][4], int ReNumber);//读入 16 个控制点和递归深度
    void DrawCurvedPatch(CDC* pDC);//绘制曲面
    void DrawControlGrid(CDC* pDC);//绘制控制网格
private:
    void Recursion(CDC* pDC, int ReNumber, CMesh Mesh);//递归函数
    void Tessellation(CMesh Mesh);//细分函数
    void DrawFacet(CDC* pDC);//绘制四边形网格
    void LeftMultiplyMatrix(double M[4][4], CP3 P[4][4]);//左乘顶点矩阵
    void RightMultiplyMatrix(CP3 P[4][4], double M[4][4]);//右乘顶点矩阵
    void TransposeMatrix(double M[4][4]);//转置矩阵
private:
    int ReNumber;//递归深度
    CP3 quadrP[4];//四边形网格点
    CP3 CtrPt[4][4];//曲面的 16 个控制点
};
```

- CBezierPatch.cpp

```
#include "pch.h"
#include "CBezierPatch.h"
#define ROUND(d) int(d + 0.5)
CBezierPatch::CBezierPatch(void)
{
    ReNumber = 0;
```

```
}
CBezierPatch::~CBezierPatch(void)
{
}
void CBezierPatch::ReadControlPoint(CP3 CtrPt[4][4], int ReNumber)
{
    for (int i = 0; i< 4; i++)
        for (int j = 0; j < 4; j++)
            this->CtrPt[i][j] = CtrPt[i][j];
    this->ReNumber = ReNumber;
}
void CBezierPatch::DrawCurvedPatch(CDC* pDC)
{
    CMesh Mesh;
    Mesh.BL = CT2(0, 0), Mesh.BR = CT2(1, 0);//初始化uv
    Mesh.TR = CT2(1, 1), Mesh.TL = CT2(0, 1);
    Recursion(pDC, ReNumber, Mesh);//递归函数
}
void CBezierPatch::Recursion(CDC* pDC, int ReNumber, CMesh Mesh)//递归函数
{
    if(0 == ReNumber)
    {
        Tessellation(Mesh);//细分曲面，将（u，v）点转换为（x，y）点
        DrawFacet(pDC);//绘制小平面
        return;
    }
    else
    {
        CT2 Mid = (Mesh.BL + Mesh.TR) / 2.0;
        CMesh SubMesh[4];//一分为四个
        //左下子长方形
        SubMesh[0].BL = Mesh.BL;
        SubMesh[0].BR = CT2(Mid.u,Mesh.BL.v);
        SubMesh[0].TR = CT2(Mid.u, Mid.v);
        SubMesh[0].TL = CT2(Mesh.BL.u, Mid.v);
        //右下子长方形
        SubMesh[1].BL = SubMesh[0].BR;
        SubMesh[1].BR = Mesh.BR;
        SubMesh[1].TR = CT2(Mesh.BR.u, Mid.v);
        SubMesh[1].TL = SubMesh[0].TR;
        //右上子长方形
        SubMesh[2].BL = SubMesh[1].TL;
        SubMesh[2].BR = SubMesh[1].TR;
        SubMesh[2].TR = Mesh.TR;
        SubMesh[2].TL = CT2(Mid.u, Mesh.TR.v);
        //左上子长方形
        SubMesh[3].BL = SubMesh[0].TL;
        SubMesh[3].BR = SubMesh[2].BL;
        SubMesh[3].TR = SubMesh[2].TL;
        SubMesh[3].TL = Mesh.TL;
        Recursion(pDC, ReNumber - 1, SubMesh[0]);//递归绘制 4 个子曲面
        Recursion(pDC, ReNumber - 1, SubMesh[1]);
        Recursion(pDC, ReNumber - 1, SubMesh[2]);
        Recursion(pDC, ReNumber - 1, SubMesh[3]);
    }
}
void CBezierPatch::Tessellation(CMesh Mesh)//细分曲面函数
{
    double M[4][4];//系数矩阵 M
    M[0][0] =-1, M[0][1] = 3, M[0][2] =-3, M[0][3] = 1;
    M[1][0] = 3, M[1][1] =-6, M[1][2] = 3, M[1][3] = 0;
    M[2][0] =-3, M[2][1] = 3, M[2][2] = 0, M[2][3] = 0;
    M[3][0] = 1, M[3][1] = 0, M[3][2] = 0, M[3][3] = 0;
    CP3 P3[4][4];//曲线计算用控制点数组
    for(int i = 0; i < 4; i++)
        for(int j = 0; j < 4; j++)
            P3[i][j] = CtrPt[i][j];
    LeftMultiplyMatrix(M, P3);//系数矩阵左乘三维点矩阵
    TransposeMatrix(M);//计算转置矩阵
    RightMultiplyMatrix(P3, M);//系数矩阵右乘三维点矩阵
    double u0, u1, u2, u3, v0, v1, v2, v3;//u、v 参数的幂
    double u[4] = { Mesh.BL.u,Mesh.BR.u ,Mesh.TR.u ,Mesh.TL.u };
    double v[4] = { Mesh.BL.v,Mesh.BR.v ,Mesh.TR.v ,Mesh.TL.v };
    for(int i = 0;i < 4; i++)
    {
        u3 = pow(u[i], 3.0), u2 = pow(u[i], 2.0), u1 = u[i], u0 = 1;
        v3 = pow(v[i], 3.0), v2 = pow(v[i], 2.0), v1 = v[i], v0 = 1;
        CP3 Pt = (u3 * P3[0][0] + u2 * P3[1][0] + u1 * P3[2][0] + u0 * P3[3][0]) * v3
               + (u3 * P3[0][1] + u2 * P3[1][1] + u1 * P3[2][1] + u0 * P3[3][1]) * v2
               + (u3 * P3[0][2] + u2 * P3[1][2] + u1 * P3[2][2] + u0 * P3[3][2]) * v1
               + (u3 * P3[0][3] + u2 * P3[1][3] + u1 * P3[2][3] + u0 * P3[3][3]) * v0;
        quadrP[i] = Pt;
    }
}
void CBezierPatch::DrawFacet(CDC* pDC)
{
    CP2 ScreenPoint[4];//二维投影点
    for(int nPoint = 0; nPoint < 4; nPoint++)
```

```
        ScreenPoint[nPoint] = quadrP[nPoint];//正交投影
    pDC->MoveTo(ROUND(ScreenPoint[0].x), ROUND(ScreenPoint[0].y));
    pDC->LineTo(ROUND(ScreenPoint[1].x), ROUND(ScreenPoint[1].y));
    pDC->LineTo(ROUND(ScreenPoint[2].x), ROUND(ScreenPoint[2].y));
    pDC->LineTo(ROUND(ScreenPoint[3].x), ROUND(ScreenPoint[3].y));
    pDC->LineTo(ROUND(ScreenPoint[0].x), ROUND(ScreenPoint[0].y));
}
void CBezierPatch::LeftMultiplyMatrix(double M[4][4], CP3 P[4][4])//左乘矩阵 M*P
{
    CP3 PTemp[4][4];//临时矩阵
    for (int i = 0; i < 4; i++)
        for (int j = 0; j < 4; j++)
            PTemp[i][j] = M[i][0] * P[0][j] + M[i][1] * P[1][j] + M[i][2] * P[2][j] + M[i][3] * P[3][j];
    for (int i = 0; i < 4; i++)
        for (int j = 0; j < 4; j++)
            P[i][j] = PTemp[i][j];
}
void CBezierPatch::RightMultiplyMatrix(CP3 P[4][4], double M[4][4])//右乘矩阵 P*M
{
    CP3 PTemp[4][4];//临时矩阵
    for (int i = 0; i < 4; i++)
        for (int j = 0; j < 4; j++)
            PTemp[i][j] = P[i][0] * M[0][j] + P[i][1] * M[1][j] + P[i][2] * M[2][j] + P[i][3] * M[3][j];
    for (int i = 0; i < 4; i++)
        for (int j = 0; j < 4; j++)
            P[i][j] = PTemp[i][j];
}
void CBezierPatch::TransposeMatrix(double M[4][4])//转置矩阵
{
    double PTemp[4][4];//临时矩阵
    for(int i = 0; i < 4; i++)
        for(int j = 0; j < 4; j++)
            PTemp[j][i] = M[i][j];
    for(int i = 0; i < 4; i++)
        for(int j = 0; j < 4; j++)
            M[i][j] = PTemp[i][j];
}
void CBezierPatch::DrawControlGrid(CDC* pDC)//绘制控制网格
{
    CP2 P2[4][4];//二维控制点
    for(int i = 0; i < 4; i++)
        for(int j = 0; j < 4; j++)
```

```
            P2[i][j] = CtrPt[i][j];//正交投影
    CPen NewPen, *pOldPen;
    NewPen.CreatePen(PS_SOLID, 3, RGB(0, 128, 0));
    pOldPen = pDC->SelectObject(&NewPen);
    for(int i = 0; i < 4; i++)
    {
        pDC->MoveTo(ROUND(P2[i][0].x), ROUND(P2[i][0].y));
        for(int j = 1; j < 4; j++)
            pDC->LineTo(ROUND(P2[i][j].x),ROUND(P2[i][j].y));
    }
    for(int j = 0; j < 4; j++)
    {
        pDC->MoveTo(ROUND(P2[0][j].x), ROUND(P2[0][j].y));
        for(int i = 1; i < 4; i++)
            pDC->LineTo(ROUND(P2[i][j].x), ROUND(P2[i][j].y));
    }
    pDC->SelectObject(pOldPen);
    NewPen.DeleteObject();
}
```

（4）设计回转类 CRevolution

- CRevolution.h

```
#include "Patch.h"
#include"BezierPatch.h"
class CRevolution  //回转类
{
public:
    CRevolution(void);
    virtual ~CRevolution(void);
    void ReadCubicBezierControlPoint(CP3 P[4]);//曲线顶点初始化
    CP3*  GetVertexArrayName(void);//得到顶点数组名
    void DrawRevolutionSurface(CDC* pDC);//绘制回转体
private:
    void ReadVertex(void);//读入回转体控制多边形顶点
    void ReadPatch(void);//读入回转体双三次曲面片
private:
    CP3 P[4];//来自曲线的 4 个三维控制点
    CP3 V[48];//回转体总顶点数组(4 个面,共 48 个点)
    CPatch S[4];//回转体总曲面数组（一圈 4 个面）
};
```

- CRevolution.cpp

```
#include "pch.h"
#include "Revolution.h"
CRevolution::CRevolution(void)
{
}
CRevolution::~CRevolution(void)
{
}
void CRevolution::ReadCubicBezierControlPoint(CP3 P[4])//三次 Bezier 曲线 4 个控制点初始化
{
    for (int i = 0; i < 4; i++)
        this->P[i] = P[i];
    //读入回转体的数据结构
    ReadVertex();
    ReadPatch();
}
CP3* CRevolution::GetVertexArrayName(void)//得到顶点数组名
{
    return   V;
}
void CRevolution::ReadVertex(void)//读入回转体的所有控制点
{
    const double m = 0.5523;//魔术常数
    //回转一圈需要 4 个面,第一个面 16 个点,第二个面 12 个点， 第三个面 12 个点，第四个面 8 个点，
共 48 个点
    //第一块面片 yo
    V[0] = P[0];
    V[1] = P[1];
    V[2] = P[2];
    V[3] = P[3];
    V[4].x = V[0].x,      V[4].y = V[0].y,  V[4].z = -V[0].x * m;
    V[5].x = V[1].x,      V[5].y = V[1].y,  V[5].z = -V[1].x * m;
    V[6].x = V[2].x,      V[6].y = V[2].y,  V[6].z = -V[2].x * m;
    V[7].x = V[3].x,      V[7].y = V[3].y,  V[7].z = -V[3].x * m;
    V[8].x = V[0].x * m,  V[8].y = V[0].y,  V[8].z = -V[0].x;
    V[9].x = V[1].x * m,  V[9].y = V[1].y,  V[9].z = -V[1].x;
    V[10].x =V[2].x * m,  V[10].y = V[2].y, V[10].z =-V[2].x;
    V[11].x =V[3].x * m,  V[11].y = V[3].y, V[11].z =-V[3].x;
    V[12].x = V[0].z,     V[12].y = V[0].y, V[12].z =-V[0].x;
    V[13].x = V[1].z,     V[13].y = V[1].y, V[13].z =-V[1].x;
    V[14].x = V[2].z,     V[14].y = V[2].y, V[14].z =-V[2].x;
    V[15].x = V[3].z,     V[15].y = V[3].y, V[15].z =-V[3].x;
    //第二块面片
    V[16].x =-V[0].x * m, V[16].y = V[0].y, V[16].z =-V[0].x;
    V[17].x =-V[1].x * m, V[17].y = V[1].y, V[17].z =-V[1].x;
    V[18].x =-V[2].x * m, V[18].y = V[2].y, V[18].z =-V[2].x;
    V[19].x =-V[3].x * m, V[19].y = V[3].y, V[19].z =-V[3].x;
    V[20].x =-V[0].x,     V[20].y = V[0].y, V[20].z =-V[0].x * m;
    V[21].x =-V[1].x,     V[21].y = V[1].y, V[21].z =-V[1].x * m;
    V[22].x =-V[2].x,     V[22].y = V[2].y, V[22].z =-V[2].x * m;
    V[23].x =-V[3].x,     V[23].y = V[3].y, V[23].z =-V[3].x * m;
    V[24].x =-V[0].x,     V[24].y = V[0].y, V[24].z = V[0].z;
    V[25].x =-V[1].x,     V[25].y = V[1].y, V[25].z = V[1].z;
    V[26].x =-V[2].x,     V[26].y = V[2].y, V[26].z = V[2].z;
    V[27].x =-V[3].x,     V[27].y = V[3].y, V[27].z = V[3].z;
    //第三块面片
    V[28].x =-V[0].x,     V[28].y = V[0].y, V[28].z = V[0].x * m;
    V[29].x =-V[1].x,     V[29].y = V[1].y, V[29].z = V[1].x * m;
    V[30].x =-V[2].x,     V[30].y = V[2].y, V[30].z = V[2].x * m;
    V[31].x =-V[3].x,     V[31].y = V[3].y, V[31].z = V[3].x * m;
    V[32].x =-V[0].x * m, V[32].y = V[0].y, V[32].z = V[0].x;
    V[33].x =-V[1].x * m, V[33].y = V[1].y, V[33].z = V[1].x;
    V[34].x =-V[2].x * m, V[34].y = V[2].y, V[34].z = V[2].x;
    V[35].x =-V[3].x * m, V[35].y = V[3].y, V[35].z = V[3].x;
    V[36].x = V[0].z,     V[36].y = V[0].y, V[36].z = V[0].x;
    V[37].x = V[1].z,     V[37].y = V[1].y, V[37].z = V[1].x;
    V[38].x = V[2].z,     V[38].y = V[2].y, V[38].z = V[2].x;
    V[39].x = V[3].z,     V[39].y = V[3].y, V[39].z = V[3].x;
    //第四块面片
    V[40].x = V[0].x * m, V[40].y = V[0].y, V[40].z = V[0].x;
    V[41].x = V[1].x * m, V[41].y = V[1].y, V[41].z = V[1].x;
    V[42].x = V[2].x * m, V[42].y = V[2].y, V[42].z = V[2].x;
    V[43].x = V[3].x * m, V[43].y = V[3].y, V[43].z = V[3].x;
    V[44].x = V[0].x,     V[44].y = V[0].y, V[44].z = V[0].x * m;
    V[45].x = V[1].x,     V[45].y = V[1].y, V[45].z = V[1].x * m;
    V[46].x = V[2].x,     V[46].y = V[2].y, V[46].z = V[2].x * m;
    V[47].x = V[3].x,     V[47].y = V[3].y, V[47].z = V[3].x * m;
}
void CRevolution::ReadPatch(void)//曲面片表
{
    //第1卦限面片
S[0].pIndex[0][0] = 0; S[0].pIndex[0][1] = 1; S[0].pIndex[0][2] = 2;  S[0].pIndex[0][3] = 3;
S[0].pIndex[1][0] = 4; S[0].pIndex[1][1] = 5; S[0].pIndex[1][2] = 6;  S[0].pIndex[1][3] = 7;
S[0].pIndex[2][0] = 8; S[0].pIndex[2][1] = 9; S[0].pIndex[2][2] = 10; S[0].pIndex[2][3] = 11;
S[0].pIndex[3][0] = 12;S[0].pIndex[3][1] = 13;S[0].pIndex[3][2] = 14; S[0].pIndex[3][3] = 15;
    //第2卦限面片
S[1].pIndex[0][0] = 12; S[1].pIndex[0][1] = 13; S[1].pIndex[0][2] = 14; S[1].pIndex[0][3] = 15;
S[1].pIndex[1][0] = 16; S[1].pIndex[1][1] = 17; S[1].pIndex[1][2] = 18; S[1].pIndex[1][3] = 19;
S[1].pIndex[2][0] = 20; S[1].pIndex[2][1] = 21; S[1].pIndex[2][2] = 22; S[1].pIndex[2][3] = 23;
S[1].pIndex[3][0] = 24; S[1].pIndex[3][1] = 25; S[1].pIndex[3][2] = 26; S[1].pIndex[3][3] = 27;
    //第3卦限面片
S[2].pIndex[0][0] = 24; S[2].pIndex[0][1] = 25; S[2].pIndex[0][2] = 26; S[2].pIndex[0][3] = 27;
```

```
S[2].pIndex[1][0] = 28; S[2].pIndex[1][1] = 29; S[2].pIndex[1][2] = 30; S[2].pIndex[1][3] = 31;
S[2].pIndex[2][0] = 32; S[2].pIndex[2][1] = 33; S[2].pIndex[2][2] = 34; S[2].pIndex[2][3] = 35;
S[2].pIndex[3][0] = 36; S[2].pIndex[3][1] = 37; S[2].pIndex[3][2] = 38; S[2].pIndex[3][3] = 39;
    //第 4 卦限面片
S[3].pIndex[0][0] = 36; S[3].pIndex[0][1] = 37; S[3].pIndex[0][2] = 38; S[3].pIndex[0][3] = 39;
S[3].pIndex[1][0] = 40; S[3].pIndex[1][1] = 41; S[3].pIndex[1][2] = 42; S[3].pIndex[1][3] = 43;
S[3].pIndex[2][0] = 44; S[3].pIndex[2][1] = 45; S[3].pIndex[2][2] = 46; S[3].pIndex[2][3] = 47;
S[3].pIndex[3][0] =  0; S[3].pIndex[3][1] =  1; S[3].pIndex[3][2] =  2; S[3].pIndex[3][3] =  3;
}
void CRevolution::DrawRevolutionSurface(CDC* pDC)//绘制回转体曲面
{
    CP3 Point[4][4];//双三次曲面片的 16 个控制点
    for(int nPatch = 0;nPatch < 4;nPatch++)
    {
        for(int i = 0;i < 4;i++)
            for (int j = 0;j < 4;j++)
                Point[i][j] = V[S[nPatch].pIndex[i][j]];
        int nRecursiveDepth = 3;//递归深度
        CBezierPatch patch;//双三次曲面
        patch.ReadControlPoint(Point, nRecursiveDepth);
        patch.DrawCurvedPatch(pDC);
        //patch.DrawControlGrid(pDC);
    }
}
```

（5）初始化曲线段

- ➢ CTestView.cpp

```
CTestView::CTestView() noexcept
{
    // TODO: 在此处添加构造代码
    bPlay = FALSE;
    double R = 200;
    double m = 0.5523;
    CP2 P2[7];//二维控制点
    P2[0] = CP2(0,  -1);//7 个二维点模拟半圆
    P2[1] = CP2(m,  -1);
    P2[2] = CP2(1,  -m);
    P2[3] = CP2(1,   0);
    P2[4] = CP2(1,   m);
    P2[5] = CP2(m,   1);
    P2[6] = CP2(0,   1);
    CP3 DownPoint[4];//下半圆的三维控制点
    DownPoint[0] = CP3(P2[0].x, P2[0].y, 0.0);
    DownPoint[1] = CP3(P2[1].x, P2[1].y, 0.0);
    DownPoint[2] = CP3(P2[2].x, P2[2].y, 0.0);
    DownPoint[3] = CP3(P2[3].x, P2[3].y, 0.0);
    revoDown.ReadCubicBezierControlPoint(DownPoint);
    tranDown.SetMatrix(revoDown.GetVertexArrayName(), 48);
    tranDown.Scale(R, R, R);
    CP3 UpPoint[4];//上半圆的三维控制点
    UpPoint[0] = CP3(P2[3].x, P2[3].y, 0.0);
    UpPoint[1] = CP3(P2[4].x, P2[4].y, 0.0);
    UpPoint[2] = CP3(P2[5].x, P2[5].y, 0.0);
    UpPoint[3] = CP3(P2[6].x, P2[6].y, 0.0);
    revoUp.ReadCubicBezierControlPoint(UpPoint);
    tranUp.SetMatrix(revoUp.GetVertexArrayName(), 48);
    tranUp.Scale(R, R, R);
}
```

（6）绘制线框球

- ➢ CTestView.cpp

```
void CTestView::OnDraw(CDC* pDC)
{
    CTestDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    if (!pDoc)
        return;
    // TODO: 在此处为本机数据添加绘制代码
    DoubleBuffer(pDC); //双缓冲
}
void CTestView::DoubleBuffer(CDC* pDC)//双缓冲
{
    CRect rect;//定义客户区矩形
    GetClientRect(&rect);//获得客户区的大小
    pDC->SetMapMode(MM_ANISOTROPIC);//pDC 自定义坐标系
    pDC->SetWindowExt(rect.Width(), rect.Height());//设置窗口范围
    pDC->SetViewportExt(rect.Width(), -rect.Height());//设置视区范围,x 轴水平向右 y 轴垂直向上
    pDC->SetViewportOrg(rect.Width() / 2, rect.Height() / 2);//客户区中心为原点
    CDC memDC;//内存 DC
    memDC.CreateCompatibleDC(pDC);//创建一个与显示 pDC 兼容的内存 memDC
    CBitmap NewBitmap, *pOldBitmap;//内存中承载的临时位图
    NewBitmap.CreateCompatibleBitmap(pDC, rect.Width(), rect.Height());//创建兼容位图
    pOldBitmap = memDC.SelectObject(&NewBitmap);//将兼容位图选入 memDC
    memDC.FillSolidRect(rect, pDC->GetBkColor());//按原来背景填充客户区，否则是黑色
    memDC.SetMapMode(MM_ANISOTROPIC);//memDC 自定义坐标系
    memDC.SetWindowExt(rect.Width(), rect.Height());
    memDC.SetViewportExt(rect.Width(), -rect.Height());
    memDC.SetViewportOrg(rect.Width() / 2, rect.Height() / 2);
    rect.OffsetRect(-rect.Width() / 2, -rect.Height() / 2);
    DrawObject(&memDC);//向 memDC 绘制图形
    pDC->BitBlt(rect.left, rect.top, rect.Width(), rect.Height(), &memDC, -rect.Width() / 2, -rect.Height() / 2, SRCCOPY);//将内存memDC中的位图拷贝到显示pDC中
    memDC.SelectObject(pOldBitmap);//恢复位图
    NewBitmap.DeleteObject();//删除位图
}
void CTestView::DrawObject(CDC* pDC)//绘制图形
{
    revoUp.DrawRevolutionSurface(pDC); //revoUp为上半球体的回转类对象
    revoDown.DrawRevolutionSurface(pDC);//revoDown为下半球体的回转类对象
}
```