学习小结：前面学习了 Bezier 曲线，B 样条基函数和 B 样条曲线的一些基础知识。掌握关键问题是一条 B 样条曲线间的多段曲线的光滑连接。因为现在是用多段 Bezier 曲线来描绘一条 B 样条曲线，所以问题变为两段 Bezier 曲线间光滑连接。两段 Bezier 曲线段（3 次）B1 和 B2 光滑连接的条件：

（1）.要求 B1 和 B2 有共同的连接点，即 G^0 连续。

（2）.要求 B1 和 B2 在连接点处有成比例的一阶导数，即 G^1 连续。由端点处的一阶导数 $B'1(1)=3(P_3-P_2), B'2(0)=3(Q_1-Q_0)$，为实现 G^1 连续，则有：

$$B'2(0)=B'1(1) \quad \text{即：} \quad Q_1-Q_0=P_3-P_2$$

这也表明，$P_2, P_3(Q_0), Q_1$ 三点共线。如下图表示了一条 3 次 B 样条曲线的所有控制多边形：

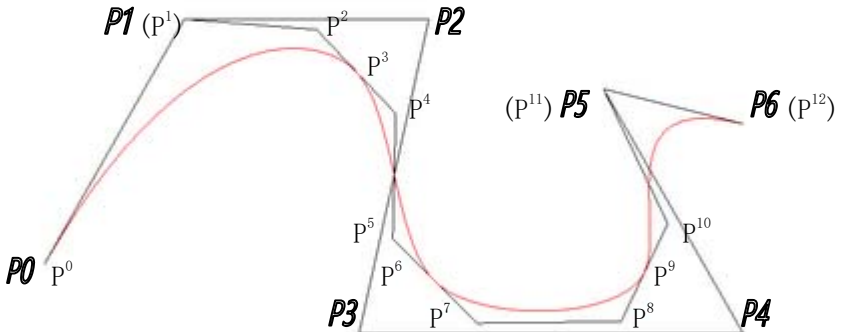

图 5. 3 次 B 样条曲线和所有控制多边形

图 5 中，P0 至 P6 为原始 3 次 B 样条曲线控制多边形顶点，P^0 至 P^{12} 是计算后最终形成 B 样条曲线控制多边形顶点。

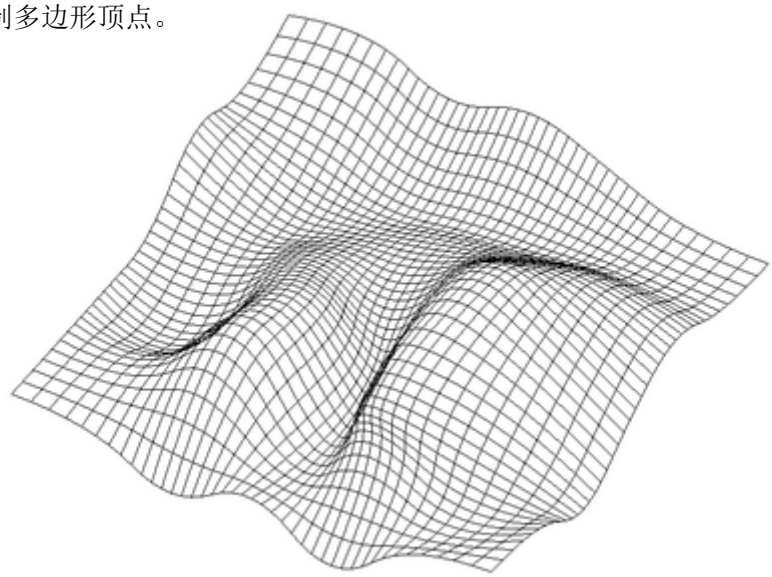

图 6. 双二次（2x2）B 样条曲面

6.B 样条曲线曲面和 NURBS 曲线曲面的 C 语言实现算法源程序

```
#ifndef _mynurbs_h
#ifndef _MYNURBS_H

#include "gl\gl.h"
#include "math.h"
//*-*-*-*-*-*-*-*-*-*-*-*-*-*-* B 样条基函数计算部分 *-*-*-*-*-*-*-*-*-*-*-*-*-*-*
//确定参数 u 所在的节点区间下标
//n=m-p-1
//m 为节点矢量 U[]的最大下标
//p 为 B 样条函数次数
int FindSource(int n,int p,float u,float U[])
{
    int low,high,mid;

    if(u==U[n+1])//特殊情况
        return n;
    //进行二分搜索
    low=p;
    high=n+1;
    mid=(int)(low+high)/2;
    while(u<U[mid]||u>U[mid])
    {
```

```
        if(u<U[mid])
            high=mid;
        else
            low=mid;
        mid=(int)(low+high)/2;
        if(u>=U[mid]&&u<U[mid+1])//找到 u 所在的节点区间的下标
            break;     //退出二分搜索
    }
    return mid;       //返回区间下标
}
//计算所有非零 B 样条基函数并返回其值
//i 为参数 u 所在的节点区间下标
void BasisFunction(int i,int p,float u,float U[],float N[])
{
     int j,di,dp,k;
    float tul,tur,left,right;
    float tmpN[50][50];

    for(k=0;k<=p;k++)
    {
    dp=0;
    for(di=i+p-k;di>=i-k;di--)
```

```
{
    if(u>=U[di]&&u<U[di+1])
        tmpN[di][0]=1;
    else
        tmpN[di][0]=0;
    dp+=1;
    for(j=1;j<dp;j++)
    {
        tul=U[di+j]-U[di];
        tur=U[di+j+1]-U[di+1];
        if(tul!=0)
            left=(u-U[di])/tul;
        else
            left=0;
        if(tur!=0)
            right=(U[di+j+1]-u)/tur;
        else
            right=0;
        tmpN[di][j]=left*tmpN[di][j-1]+right*tmpN[di+1][j-1];
    }
}
N[i-k]=tmpN[i-k][p];
```

```
        }
}
//---------------------------------------------------------------------------
//计算基函数的 1 阶导数并保存在 NP[]中
//i 为参数 u 所在的节点区间下标
//p 为 B 样条函数次数 P>2
void DerBasisFunc(int i,int p,float u,float U[],float NP[])
{
        int j,di,dp,k;
        float tul,tur,left,right,saved,dl,dr;
        float tmpN[50][50];

        for(k=0;k<=p;k++)
        {
        dp=0;
        for(di=i+p-k;di>=i-k;di--)
        {
            if(u>=U[di]&&u<U[di+1])
                tmpN[di][0]=1;
            else
                tmpN[di][0]=0;
            dp+=1;
```

```
            for(j=1;j<dp;j++)
             {
                    tul=U[di+j]-U[di];
                    tur=U[di+j+1]-U[di+1];
                    if(tul!=0)
                          left=(u-U[di])/tul,dl=1/tul;
                    else
                          left=0,dl=0;
                    if(tur!=0)
                          right=(U[di+j+1]-u)/tur,dr=1/tur;
                    else
                          right=0,dr=0;
                    tmpN[di][j]=(left*tmpN[di][j-1]+right*tmpN[di+1][j-1]);
                    saved=p*(dl*tmpN[di][j-1]-dr*tmpN[di+1][j-1])/(p+p-1);
             }
       }
       NP[i-k]=saved;
       }
}
//*-*-*-*-*-*-*-*-*-*-*-*-*-* Bezier 曲线曲面部分  *-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
//计算参数 u 的 p 次基函数值并存在 BC[]中
void BernsteinFunc(int p,double t,float BC[])
```

```
{
    for(int i=0;i<=p;i++)
    {
        if(i==0)
            BC[0]=(float)pow(1-t,p);
        if(i==p)
            BC[p]=(float)pow(t,p);
        if(i>0&&i<p)
            BC[i]=p*(float)pow(t,i)*(float)pow(1-t,p-i);
    }
}
//获取 p 次 Bezier 曲线上的 lines 个点的值
void BezierPoint(int p,float px[],float py[],float pz[],int lines,float tmp[][3])
{
    float BC[20];
    int i,j;

    for(j=0;j<=lines;j++)
    {
       double t=j/(float)lines;
       BernsteinFunc(p,t,BC);
       tmp[j][0]=tmp[j][1]=tmp[j][2]=0;
```

```
            for(i=0;i<p+1;i++)
            {
              tmp[j][0]+=BC[i]*px[i];
              tmp[j][1]+=BC[i]*py[i];
              tmp[j][2]+=BC[i]*pz[i];
            }
        }
}
//获取 p 次有理 Bezier 曲线上的 lines 个点的值
void NBezierPoint(int p,float px[],float py[],float pz[],float pw[],int lines,float tmp[][4])
{
        float x,y,z,w,BC[20];
        int i,j;

        for(j=0;j<=lines;j++)
        {
            double t=j/(float)lines;
            BernsteinFunc(p,t,BC);
            x=y=z=w=0;
            for(i=0;i<p+1;i++)
            {
              x+=BC[i]*px[i]*pw[i];
```

```
            y+=BC[i]*py[i]*pw[i];
            z+=BC[i]*pz[i]*pw[i];
            w+=BC[i]*pw[i];
          }
          tmp[j][0]=x/w;
          tmp[j][1]=y/w;
          tmp[j][2]=z/w;
          tmp[j][3]=w;
      }
}
//---------------------------------------------------------------------------
//绘制 p 次的 Bezier 曲线
void Bezier(int p,float px[],float py[],float pz[],int lines)
{
      float pt[100][3];
      int j;

      BezierPoint(p,px,py,pz,lines,pt);
      for(j=1;j<=lines;j++)
      {
         glBegin(GL_LINES);
         glVertex3f(pt[j-1][0],pt[j-1][1],pt[j-1][2]);
```

```
            glVertex3f(pt[j][0],pt[j][1],pt[j][2]);
            glEnd();
        }
}
//-------------------------------------------------------------------------
//绘制 p 次的有理 Bezier 曲线
void NBezier(int p,float px[],float py[],float pz[],float w[],int lines)
{
        float pt[100][4];
        int j;

        NBezierPoint(p,px,py,pz,w,lines,pt);
        for(j=1;j<=lines;j++)
        {
            glBegin(GL_LINES);
            glVertex3f(pt[j-1][0],pt[j-1][1],pt[j-1][2]);
            glVertex3f(pt[j][0],pt[j][1],pt[j][2]);
            glEnd();
        }
}
//-------------------------------------------------------------------------
//计算双 p 次 Bezier 曲面上所有的点并保存在 Pt[][][]中
```

```
//u 和 v 分别为曲面(u,v)方向上的网格数
void BezierFacePoint(int p,int u,int v,float px[][4],float py[][4],float pz[][4],float
pt[161][161][3])
{
    float urx[11][161],ury[11][161],urz[11][161];
    float tx[11],ty[11],tz[11],tmp[161][3];
    int i,j,k;

    for(j=0;j<p+1;j++)
    {
     for(i=0;i<p+1;i++)
     {
          tx[i]=px[i][j];
          ty[i]=py[i][j];
          tz[i]=pz[i][j];
     }
     BezierPoint(p,tx,ty,tz,v,tmp);
     for(k=0;k<=v;k++)
     {
          urx[j][k]=tmp[k][0];
          ury[j][k]=tmp[k][1];
          urz[j][k]=tmp[k][2];
```

```
        }
      }
      for(i=0;i<=v;i++)
      {
        for(k=0;k<p+1;k++)
        {
            tx[k]=urx[k][i];
            ty[k]=ury[k][i];
            tz[k]=urz[k][i];
        }
        BezierPoint(p,tx,ty,tz,u,tmp);
        for(j=0;j<=u;j++)
        {
            pt[i][j][0]=tmp[j][0];
            pt[i][j][1]=tmp[j][1];
            pt[i][j][2]=tmp[j][2];
        }
      }
}
//-------------------------------------------------------------------------
//计算双 p 次有理 Bezier 曲面上所有的点并保存在 Pt[][][]中
//u 和 v 分别为曲面(u,v)方向上的网格数
```

```
void NuBezierFacePoint(int p,int u,int v,float px[][4],float py[][4],float pz[][4],float w[][4],float
pt[161][161][3])
{
	float urx[11][161],ury[11][161],urz[11][161],urw[11][161];
	float tx[11],ty[11],tz[11],tw[11],tmp[161][4];
	int i,j,k;

	for(j=0;j<p+1;j++)
	{
	 for(i=0;i<p+1;i++)
	 {
		tx[i]=px[i][j];
		ty[i]=py[i][j];
		tz[i]=pz[i][j];
		tw[i]=w[i][j];
	 }
	 NBezierPoint(p,tx,ty,tz,tw,v,tmp);
	 for(k=0;k<=v;k++)
	 {
		urx[j][k]=tmp[k][0];
		ury[j][k]=tmp[k][1];
		urz[j][k]=tmp[k][2];
```

```
            urw[j][k]=tmp[k][3];
          }
        }
        for(i=0;i<=v;i++)
        {
          for(k=0;k<p+1;k++)
          {
            tx[k]=urx[k][i];
            ty[k]=ury[k][i];
            tz[k]=urz[k][i];
            tw[k]=urw[k][i];
          }
          NBezierPoint(p,tx,ty,tz,tw,u,tmp);
          for(j=0;j<=u;j++)
          {
            pt[i][j][0]=tmp[j][0];
            pt[i][j][1]=tmp[j][1];
            pt[i][j][2]=tmp[j][2];
          }
        }
}
//-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-* B 样条曲线曲面部分 -*-*-*-*-*-*-*-*-*-*-*-*-*-
```

```
//计算样条曲线的 1 阶导矢（u 所对应的所有点）保存在 Der[]中
//n=m-p-1
//p 为曲线的次数
void BSplineDer(int n,int p,float U[],float P[],float Der[])
{
    float N[100],tmp;
    int i,j;

    for(i=p+1;i<=n;i++)
    {
        DerBasisFunc(i,p,U[i],U,N);
        tmp=0;
        for(j=i;j>=i-p;j--)
            tmp+=N[j]*P[j];
        Der[i-p]=tmp;
    }
}
//计算曲线上的点（u 所对应的所有点）保存在 Poi[]中
//n=m-p-1
//p 为曲线的次数
void BSplinePoint(int n,int p,float U[],float P[],float Poi[])
{
```

```
    float N[100],tmp;
    int i,j;

    for(i=p+1;i<=n;i++)
    {
        BasisFunction(i,p,U[i],U,N);
        tmp=0;
        for(j=i;j>=i-p;j--)
            tmp+=N[j]*P[j];
        Poi[i-p]=tmp;
    }
}
//计算 3 次样条曲线上的所有控制多边形保存在 CP[]中
//m 为节点矢量 U[]的最大下标
void B3SplineControlPoint(int m,float U[],float P[],float CP[])
{
    int n,k,i,cp,p;
    float Poi[100],Der[100],add;

    p=3;
    n=m-p-1;
    BSplinePoint(n,p,U,P,Poi);
```

```
        BSplineDer(n,p,U,P,Der);
        cp=(n-p)*3+p;
        for(i=0;i<2;i++)
        {
            CP[i]=P[i];
            CP[cp-i]=P[n-i];
        }
        for(i=3;i<cp-1;i+=3)
        {
            k=(int)i/3;
            add=Der[k]/p;
            CP[i]=Poi[k];
            CP[i-1]=CP[i]-add;
            CP[i+1]=CP[i]+add;
        }
}
//计算 2 次样条曲线上的所有控制多边形保存在 CP[]中
//m 为节点矢量 U[]的最大下标
void B2SplineControlPoint(int m,float U[],float P[],float CP[])
{
        int n,k,tm,i,cp,p;
        float Poi[100];
```

```
        p=2;
        n=m-p-1;
        BSplinePoint(n,p,U,P,Poi);
        cp=(n-p)*2+p;
        for(i=0;i<2;i++)
            CP[i]=P[i];
        CP[cp]=P[n];
        tm=2;
        for(i=2;i<cp-1;i+=2)
        {
            k=(int)i/2;
            CP[i]=Poi[k];
            CP[i+1]=P[tm];
            tm++;
        }
}
//绘制 3 次 B 样条曲线
//m 为节点矢量 U[]的最大下标
void BSpline3L(int m,float U[],float px[],float py[],float pz[])
{
        float pcx[100],pcy[100],pcz[100],drx[4],dry[4],drz[4];
```

```
int i,j,tmcp;


B3SplineControlPoint(m,U,px,pcx);
B3SplineControlPoint(m,U,py,pcy);
B3SplineControlPoint(m,U,pz,pcz);
/*
glColor3f(0.0f,0.0f,0.0f);
for(i=1;i<3*m-17;i++)
{
	glBegin(GL_LINES);
	glVertex3f(pcx[i-1],pcy[i-1],pcz[i-1]);
	glVertex3f(pcx[i],pcy[i],pcz[i]);
	glEnd();
}
glColor3f(1.0f,0.0f,0.0f);*/
tmcp=m-7;
for(i=0;i<=tmcp;i++)
{
	for(j=i*3;j<i*3+4;j++)
	{
		drx[j-i*3]=pcx[j];
		dry[j-i*3]=pcy[j];
```

```
                drz[j-i*3]=pcz[j];
            }
            Bezier(3,drx,dry,drz,20);
        }
}
//绘制 2 次 B 样条曲线
//m 为节点矢量 U[]的最大下标
void BSpline2L(int m,float U[],float px[],float py[],float pz[])
{
    float pcx[100],pcy[100],pcz[100],drx[3],dry[3],drz[3];
    int i,j,tmcp;

    B2SplineControlPoint(m,U,px,pcx);
    B2SplineControlPoint(m,U,py,pcy);
    B2SplineControlPoint(m,U,pz,pcz);
    tmcp=m-5;
    for(i=0;i<=tmcp;i++)
    {
        for(j=i*2;j<i*2+3;j++)
        {
            drx[j-i*2]=pcx[j];
            dry[j-i*2]=pcy[j];
```

```
                drz[j-i*2]=pcz[j];
            }
            Bezier(2,drx,dry,drz,20);
        }
}
//计算双三次（3x3）B 样条曲面所有控制多边形顶点，并保存在 pt[][][]中
//mu,mv 分别为节点矢量 U[],V[]的最大下标值
void BS3FaceControlPoint(int mu,float U[],int mv,float V[],float px[],float py[],float pz[],float
pt[100][100][3])
{
    int i,j,k,dp;
    float tmx[50],tmy[50],tmz[50];
    float tmpx[50][100],tmpy[50][100],tmpz[50][100];
    float uvx[100][100],uvy[100][100],uvz[100][100];

    for(i=0;i<mv-3;i++)
    {
        dp=i*(mu-3);
        for(j=dp;j<mu-3+dp;j++)
        {
            tmx[j-dp]=px[j];
            tmy[j-dp]=py[j];
```

```
            tmz[j-dp]=pz[j];
        }
        B3SplineControlPoint(mu,U,tmx,tmpx[i]);
        B3SplineControlPoint(mu,U,tmy,tmpy[i]);
        B3SplineControlPoint(mu,U,tmz,tmpz[i]);
    }
    for(i=0;i<3*mu-17;i++)
    {
        for(j=0;j<mv-3;j++)
        {
            tmx[j]=tmpx[j][i];
            tmy[j]=tmpy[j][i];
            tmz[j]=tmpz[j][i];
        }
        B3SplineControlPoint(mv,V,tmx,uvx[i]);
        B3SplineControlPoint(mv,V,tmy,uvy[i]);
        B3SplineControlPoint(mv,V,tmz,uvz[i]);
        for(k=0;k<3*mv-17;k++)
        {
            pt[i][k][0]=uvx[i][k];
            pt[i][k][1]=uvy[i][k];
            pt[i][k][2]=uvz[i][k];
```

```
            }
        }
}
//计算双二次（2x2）B 样条曲面所有控制多边形顶点，并保存在 pt[][][]中
//mu,mv 分别为节点矢量 U[],V[]的最大下标值
void BS2FaceControlPoint(int mu,float U[],int mv,float V[],float px[],float py[],float pz[],float
pt[100][100][3])
{
    int i,j,k,dp;
    float tmx[50],tmy[50],tmz[50];
    float tmpx[50][100],tmpy[50][100],tmpz[50][100];
    float uvx[100][100],uvy[100][100],uvz[100][100];

    for(i=0;i<mv-2;i++)
    {
        dp=i*(mu-2);
        for(j=dp;j<mu-2+dp;j++)
        {
            tmx[j-dp]=px[j];
            tmy[j-dp]=py[j];
            tmz[j-dp]=pz[j];
        }
```

```
        B2SplineControlPoint(mu,U,tmx,tmpx[i]);
        B2SplineControlPoint(mu,U,tmy,tmpy[i]);
        B2SplineControlPoint(mu,U,tmz,tmpz[i]);
    }
    for(i=0;i<2*mu-7;i++)
    {
        for(j=0;j<mv-2;j++)
        {
            tmx[j]=tmpx[j][i];
            tmy[j]=tmpy[j][i];
            tmz[j]=tmpz[j][i];
        }
        B2SplineControlPoint(mv,V,tmx,uvx[i]);
        B2SplineControlPoint(mv,V,tmy,uvy[i]);
        B2SplineControlPoint(mv,V,tmz,uvz[i]);
        for(k=0;k<2*mv-7;k++)
        {
            pt[i][k][0]=uvx[i][k];
            pt[i][k][1]=uvy[i][k];
            pt[i][k][2]=uvz[i][k];
        }
    }
```

```
}
//------------------------------------------------------------------------------
//设置网格数
void SetGridCount(int dt,int tu,int tmk[])
{
    int i,tm;

    tm=tu%dt;
    for(i=0;i<dt-1;i++)
        tmk[i]=(tu-tm)/dt;
    tmk[dt-1]=tmk[0]+tm;
}
//------------------------------------------------------------------------------
//计算双三次(3x3 次)或双二次(2x2 次)B 样条曲面上所有的点并保存在 bs[][][]中
//nu,mv 分别为节点矢量 U[],V[]的最大下标
//uk,vk 分别为 B 样条曲面(u,v)方向上的网格数
//p 为曲面的次数
void BSplineFace(int p,int nu,float U[],int uk,int mv,float V[],int vk,
                                float px[],float py[],float pz[],float bs[161][161][3])
{
    int udk[20],vdk[20],i,j,k,l,hu,sv,du,dv;
    float tp[100][100][3],td[161][161][3];
```

```
float tmx[4][4],tmy[4][4],tmz[4][4];


du=nu-2*p;
dv=mv-2*p;
SetGridCount(du,uk,udk);
SetGridCount(dv,vk,vdk);
if(p==3)
      BS3FaceControlPoint(nu,U,mv,V,px,py,pz,tp);
if(p==2)
      BS2FaceControlPoint(nu,U,mv,V,px,py,pz,tp);
for(i=0;i<dv;i++)
{
 for(k=0;k<du;k++)
   {
   for(j=i*p;j<p+1+i*p;j++)
     for(l=k*p;l<p+1+k*p;l++)
     {
           tmx[j-i*p][l-k*p]=tp[l][j][0];
           tmy[j-i*p][l-k*p]=tp[l][j][1];
           tmz[j-i*p][l-k*p]=tp[l][j][2];
     }
   BezierFacePoint(p,udk[k],vdk[i],tmx,tmy,tmz,td);
```

```
            for(sv=i*vdk[0];sv<=vdk[i]+i*vdk[0];sv++)
              for(hu=k*udk[0];hu<=udk[k]+k*udk[0];hu++)
              {
                    bs[sv][hu][0]=td[sv-i*vdk[0]][hu-k*udk[0]][0];
                    bs[sv][hu][1]=td[sv-i*vdk[0]][hu-k*udk[0]][1];
                    bs[sv][hu][2]=td[sv-i*vdk[0]][hu-k*udk[0]][2];
              }
            }
      }
}
//-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-* Nurbs 样条曲线曲面部分 -*-*-*-*-*-*-*-*-*-*-*-*-*-*-
//计算 Nurbs 曲线上的点（u 所对应的所有点）保存在 Poi[]中
//n=m-p-1
//p 为曲线的次数
void NurbsPoint(int n,int p,float U[],float P[],float W[],float Poi[])
{
      float N[100],tmp,tmw;
      int i,j;

      for(i=p+1;i<=n;i++)
      {
            BasisFunction(i,p,U[i],U,N);
```

```
            tmp=0;tmw=0;
            for(j=i;j>=i-p;j--)
            {
                tmp+=N[j]*P[j]*W[j];
                tmw+=N[j]*W[j];
            }
            Poi[i-p]=tmp/tmw;
        }
}
//计算 Nurbs 曲线的 1 阶导矢（u 所对应的所有点）保存在 Der[]中
//n=m-p-1
//p 为曲线的次数
void NurbsDer(int n,int p,float U[],float P[],float W[],float Der[])
{
        float N[100],CP[100],NW[100],tmp,tw;
        int i,j;

        NurbsPoint(n,p,U,P,W,CP);
        BSplinePoint(n,p,U,W,NW);
        for(i=p+1;i<=n;i++)
        {
            DerBasisFunc(i,p,U[i],U,N);
```

```
            tmp=0;tw=0;
            for(j=i;j>=i-p;j--)
            {
                tmp+=N[j]*P[j]*W[j];
                tw+=N[j]*W[j];
            }
            Der[i-p]=(tmp-tw*CP[i-p])/NW[i-p];
        }
}
//计算 3 次 Nurbs 曲线上的所有控制多边形保存在 CP[]中
//m 为节点矢量 U[]的最大下标
void Nurbs3ControlPoint(int m,float U[],float P[],float W[],float CP[])
{
        int n,k,i,cp,p;
        float Poi[100],Der[100],add;

        p=3;
        n=m-p-1;
        NurbsPoint(n,p,U,P,W,Poi);
        NurbsDer(n,p,U,P,W,Der);
        cp=(n-p)*3+p;
        for(i=0;i<2;i++)
```

```
    {
        CP[i]=P[i];
        CP[cp-i]=P[n-i];
    }
    for(i=3;i<cp-1;i+=3)
    {
        k=(int)i/3;
        add=Der[k]/p;
        CP[i]=Poi[k];
        CP[i-1]=CP[i]-add;
        CP[i+1]=CP[i]+add;
    }
}
//计算 2 次 Nurbs 曲线上的所有控制多边形保存在 CP[]中
//m 为节点矢量 U[]的最大下标
void Nurbs2ControlPoint(int m,float U[],float P[],float W[],float CP[])
{
    int n,k,tm,i,cp,p;
    float Poi[100];

    p=2;
    n=m-p-1;
```

```
        NurbsPoint(n,p,U,P,W,Poi);
        cp=(n-p)*2+p;
        for(i=0;i<2;i++)
                CP[i]=P[i];
        CP[cp]=P[n];
        tm=2;
        for(i=2;i<cp-1;i+=2)
        {
                k=(int)i/2;
                CP[i]=Poi[k];
                CP[i+1]=P[tm];
                tm++;
        }
}
//绘制 3 次 Nurbs 样条曲线
//m 为节点矢量 U[]的最大下标
void Nurbs3L(int m,float U[],float px[],float py[],float pz[],float W[])
{
        float pcx[100],pcy[100],pcz[100],drx[4],dry[4],drz[4];
        float pcw[100],drw[4];
        int i,j,tmcp;
```

```
        Nurbs3ControlPoint(m,U,px,W,pcx);
        Nurbs3ControlPoint(m,U,py,W,pcy);
        Nurbs3ControlPoint(m,U,pz,W,pcz);
        B3SplineControlPoint(m,U,W,pcw);
        tmcp=m-7;
        for(i=0;i<=tmcp;i++)
        {
            for(j=i*3;j<i*3+4;j++)
            {
                drx[j-i*3]=pcx[j];
                dry[j-i*3]=pcy[j];
                drz[j-i*3]=pcz[j];
                drw[j-i*3]=pcw[j];
            }
            NBezier(3,drx,dry,drz,drw,20);
        }
}
//绘制 2 次 Nurbs 样条曲线
//m 为节点矢量 U[]的最大下标
void Nurbs2L(int m,float U[],float px[],float py[],float pz[],float W[])
{
        float pcx[100],pcy[100],pcz[100],drx[3],dry[3],drz[3];
```

```
    float pcw[100],drw[3];
    int i,j,tmcp;

    Nurbs2ControlPoint(m,U,px,W,pcx);
    Nurbs2ControlPoint(m,U,py,W,pcy);
    Nurbs2ControlPoint(m,U,pz,W,pcz);
    B2SplineControlPoint(m,U,W,pcw);
    tmcp=m-5;
    for(i=0;i<=tmcp;i++)
    {
        for(j=i*2;j<i*2+3;j++)
        {
            drx[j-i*2]=pcx[j];
            dry[j-i*2]=pcy[j];
            drz[j-i*2]=pcz[j];
            drw[j-i*2]=pcw[j];
        }
        NBezier(2,drx,dry,drz,drw,20);
    }
}
//计算双三次（3x3）Nurbs 样条曲面所有控制多边形顶点，并保存在 pt[][][]中
//mu,mv 分别为节点矢量 U[],V[]的最大下标值
```

```
void Nurbs3FControlPoint(int mu,float U[],int mv,float V[],float px[],float py[],float pz[],float
W[],float pt[100][100][4])
{
    int i,j,k,dp;
    float tmx[50],tmy[50],tmz[50],tmw[50];
    float tmpx[50][100],tmpy[50][100],tmpz[50][100],tmpw[50][100];
    float uvx[100][100],uvy[100][100],uvz[100][100],uvw[100][100];

    for(i=0;i<mv-3;i++)
    {
        dp=i*(mu-3);
        for(j=dp;j<mu-3+dp;j++)
        {
            tmx[j-dp]=px[j];
            tmy[j-dp]=py[j];
            tmz[j-dp]=pz[j];
            tmw[j-dp]=W[j];
        }
        Nurbs3ControlPoint(mu,U,tmx,tmw,tmpx[i]);
        Nurbs3ControlPoint(mu,U,tmy,tmw,tmpy[i]);
        Nurbs3ControlPoint(mu,U,tmz,tmw,tmpz[i]);
        B3SplineControlPoint(mu,U,tmw,tmpw[i]);
```

```
}
for(i=0;i<3*mu-17;i++)
{
	for(j=0;j<mv-3;j++)
	{
		tmx[j]=tmpx[j][i];
		tmy[j]=tmpy[j][i];
		tmz[j]=tmpz[j][i];
		tmw[j]=tmpw[j][i];
	}
	Nurbs3ControlPoint(mv,V,tmx,tmw,uvx[i]);
	Nurbs3ControlPoint(mv,V,tmy,tmw,uvy[i]);
	Nurbs3ControlPoint(mv,V,tmz,tmw,uvz[i]);
	B3SplineControlPoint(mv,V,tmw,uvw[i]);
	for(k=0;k<3*mv-17;k++)
	{
		pt[i][k][0]=uvx[i][k];
		pt[i][k][1]=uvy[i][k];
		pt[i][k][2]=uvz[i][k];
		pt[i][k][3]=uvw[i][k];
	}
}
```

```
}
//计算双二次（2x2）Nurbs 样条曲面所有控制多边形顶点，并保存在 pt[][][]中
//mu,mv 分别为节点矢量 U[],V[]的最大下标值
void Nurbs2FControlPoint(int mu,float U[],int mv,float V[],float px[],float py[],float pz[],float
W[],float pt[100][100][4])
{
    int i,j,k,dp;
    float tmx[50],tmy[50],tmz[50],tmw[50];
    float tmpx[50][100],tmpy[50][100],tmpz[50][100],tmpw[50][100];
    float uvx[100][100],uvy[100][100],uvz[100][100],uvw[100][100];

    for(i=0;i<mv-2;i++)
    {
        dp=i*(mu-2);
        for(j=dp;j<mu-2+dp;j++)
        {
            tmx[j-dp]=px[j];
            tmy[j-dp]=py[j];
            tmz[j-dp]=pz[j];
            tmw[j-dp]=W[j];
        }
        Nurbs2ControlPoint(mu,U,tmx,tmw,tmpx[i]);
```

```
        Nurbs2ControlPoint(mu,U,tmy,tmw,tmpy[i]);
        Nurbs2ControlPoint(mu,U,tmz,tmw,tmpz[i]);
        B2SplineControlPoint(mu,U,tmw,tmpw[i]);
    }
    for(i=0;i<2*mu-7;i++)
    {
        for(j=0;j<mv-2;j++)
        {
            tmx[j]=tmpx[j][i];
            tmy[j]=tmpy[j][i];
            tmz[j]=tmpz[j][i];
            tmw[j]=tmpw[j][i];
        }
        Nurbs2ControlPoint(mv,V,tmx,tmw,uvx[i]);
        Nurbs2ControlPoint(mv,V,tmy,tmw,uvy[i]);
        Nurbs2ControlPoint(mv,V,tmz,tmw,uvz[i]);
        B2SplineControlPoint(mv,V,tmw,uvw[i]);
        for(k=0;k<2*mv-7;k++)
        {
            pt[i][k][0]=uvx[i][k];
            pt[i][k][1]=uvy[i][k];
            pt[i][k][2]=uvz[i][k];
```

```
                pt[i][k][3]=uvw[i][k];
            }
        }
}
//-------------------------------------------------------------------------
//计算双三次(3x3 次)或双二次(2x2 次)Nurbs 样条曲面上所有的点并保存在 bs[][][]中
//nu,mv 分别为节点矢量 U[],V[]的最大下标
//uk,vk 分别为 B 样条曲面(u,v)方向上的网格数
//p 为曲面的次数
void NurbsFace(int p,int nu,float U[],int uk,int mv,float V[],int vk,
                        float px[],float py[],float pz[],float w[],float bs[161][161][3])
{
    int udk[20],vdk[20],i,j,k,l,hu,sv,du,dv;
    float tp[100][100][4],td[161][161][3];
    float tmx[4][4],tmy[4][4],tmz[4][4],tmw[4][4];

    du=nu-2*p;
    dv=mv-2*p;
    SetGridCount(du,uk,udk);
    SetGridCount(dv,vk,vdk);
    if(p==3)
        Nurbs3FControlPoint(nu,U,mv,V,px,py,pz,w,tp);
```

```
if(p==2)
       Nurbs2FControlPoint(nu,U,mv,V,px,py,pz,w,tp);
for(i=0;i<dv;i++)
{
  for(k=0;k<du;k++)
    {
    for(j=i*p;j<p+1+i*p;j++)
      for(l=k*p;l<p+1+k*p;l++)
      {
            tmx[j-i*p][l-k*p]=tp[l][j][0];
            tmy[j-i*p][l-k*p]=tp[l][j][1];
            tmz[j-i*p][l-k*p]=tp[l][j][2];
            tmw[j-i*p][l-k*p]=tp[l][j][3];
      }
    NuBezierFacePoint(p,udk[k],vdk[i],tmx,tmy,tmz,tmw,td);
    for(sv=i*vdk[0];sv<=vdk[i]+i*vdk[0];sv++)
      for(hu=k*udk[0];hu<=udk[k]+k*udk[0];hu++)
      {
            bs[sv][hu][0]=td[sv-i*vdk[0]][hu-k*udk[0]][0];
            bs[sv][hu][1]=td[sv-i*vdk[0]][hu-k*udk[0]][1];
            bs[sv][hu][2]=td[sv-i*vdk[0]][hu-k*udk[0]][2];
      }
```

```
        }
    }
}
//*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-* 绘制曲面部分 *-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
//计算多边形的外法线返回值 tmN[]
void getN(float x[3],float y[3],float z[3],float tmN[3])
{
    float p1,p2,p3,q1,q2,q3;
    float nx,ny,nz;

    p1=x[1]-x[0];
    p2=y[1]-y[0];
    p3=z[1]-z[0];
    q1=x[2]-x[1];
    q2=y[2]-y[1];
    q3=z[2]-z[1];
    nx=p2*q3-q2*p3;
    ny=q1*p3-p1*q3;
    nz=p1*q2-p2*q1;
    tmN[0]=nx;
    tmN[1]=ny;
    tmN[2]=nz;
```

```
}

//-------------------------------------------------------------------------------

//显示 B 样条曲面

//fill 取值为 0 或 1

void ShowSurface(int u,int v,float bs[161][161][3],int fill)

{

    int i,j;

    float x[3],y[3],z[3],tmn[3];



    for(i=0;i<=v;i++)

      for(j=0;j<=u;j++)

      {

          if(fill!=0)

          {

          x[0]=bs[i][j][0];

          x[1]=bs[i+1][j][0];

          x[2]=bs[i+1][j+1][0];

          y[0]=bs[i][j][1];

          y[1]=bs[i+1][j][1];

          y[2]=bs[i+1][j+1][1];

          z[0]=bs[i][j][2];

          z[1]=bs[i+1][j][2];
```

```
    z[2]=bs[i+1][j+1][2];
    getN(x,y,z,tmn);
    glEnable(GL_NORMALIZE);
    glBegin(GL_QUADS);
    glNormal3f(tmn[0],tmn[1],tmn[2]);
    if(j<u)
    {
        glVertex3f(bs[i][j][0],bs[i][j][1],bs[i][j][2]);
        glVertex3f(bs[i][j+1][0],bs[i][j+1][1],bs[i][j+1][2]);
    }
    if(i<v)
    {
        glVertex3f(bs[i+1][j+1][0],bs[i+1][j+1][1],bs[i+1][j+1][2]);
        glVertex3f(bs[i+1][j][0],bs[i+1][j][1],bs[i+1][j][2]);
    }
    glEnd();
    glDisable(GL_NORMALIZE);
    }
else{
    glBegin(GL_LINES);
    if(j<u)
    {
```

```
            glVertex3f(bs[i][j][0],bs[i][j][1],bs[i][j][2]);
            glVertex3f(bs[i][j+1][0],bs[i][j+1][1],bs[i][j+1][2]);
        }
        if(i<v)
        {
            glVertex3f(bs[i][j][0],bs[i][j][1],bs[i][j][2]);
            glVertex3f(bs[i+1][j][0],bs[i+1][j][1],bs[i+1][j][2]);
        }
        glEnd();
    }
    }
}
#endif
#endif
```

建立库文件“myNurbs.h”保存在 include 目录下，在文件开始处写上#include "myNurbs.h"即可用各函数的功能。

例如：

```
#include "windows.h"
#include "glut.h"
#include "myNurbs.h"
```

```
float x[]={17,17,-17,-17,-17,17,17,
                    10,10,-10,-10,-10,10,10,
                    15,15,-15,-15,-15,15,15,
                    30,30,-30,-30,-30,30,30,
                    20,20,-20,-20,-20,20,20,
                    10,10,-10,-10,-10,10,10,
                    0,0,0,0,0,0,0};
float y[]={100,100,100,100,100,100,100,
                    90,90,90,90,90,90,90,
                    60,60,60,60,60,60,60,
                    30,30,30,30,30,30,30,
                    10,10,10,10,10,10,10,
                    0,0,0,0,0,0,0,
                    0,0,0,0,0,0,0};
float z[]={60,77,77,60,43,43,60,
                    60,70,70,60,50,50,60,
                    60,75,75,60,45,45,60,
                    60,90,90,60,30,30,60,
                    60,80,80,60,40,40,60,
                    60,70,70,60,50,50,60,
                    60,60,60,60,60,60,60};
float w[]={1,0.5,0.5,1,0.5,0.5,1,
```

```
                1,0.5,0.5,1,0.5,0.5,1,

                1,0.5,0.5,1,0.5,0.5,1,

                1,0.5,0.5,1,0.5,0.5,1,

                1,0.5,0.5,1,0.5,0.5,1,

                1,0.5,0.5,1,0.5,0.5,1,

                1,0.5,0.5,1,0.5,0.5,1};
float u[]={0,0,0,0.25,0.5,0.5,0.75,1,1,1};
float v[]={0,0,0,0.1,0.4,0.6,0.95,1,1,1};
float suv[161][161][3];
float r=0.0f;
float dx=1.0f;
float mx=0.0f;
void lightm()
{
    GLfloat lamb[4]={0.35f,0.35f,0.35f,1.0f};
    GLfloat ldif[4]={0.35f,0.35f,0.35f,1.0f};
    GLfloat lspe[4]={0.55f,0.55f,0.55f,1.0f};
    GLfloat lpos[4]={200.0f,200.0f,60.0f,1.0f};

    GLfloat mamb[4]={0.5f,0.5f,0.5f,1.0f};
    GLfloat mdif[4]={0.5f,0.5f,0.5f,1.0f};
    GLfloat mspe[4]={0.7f,0.7f,0.7f,1.0f};
```

```
    GLfloat memi[4]={0.0f,0.0f,0.0f,1.0f};
    GLfloat mshininess=128.0f;

    glLightfv(GL_LIGHT1,GL_AMBIENT,lamb);
    glLightfv(GL_LIGHT1,GL_DIFFUSE,ldif);
    glLightfv(GL_LIGHT1,GL_SPECULAR,lspe);
    glLightfv(GL_LIGHT1,GL_POSITION,lpos);

    glMaterialfv(GL_FRONT_AND_BACK,GL_AMBIENT,mamb);
    glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,mdif);
    glMaterialfv(GL_FRONT_AND_BACK,GL_SPECULAR,mspe);
    glMaterialfv(GL_FRONT_AND_BACK,GL_EMISSION,memi);
    glMaterialf(GL_FRONT_AND_BACK,GL_SHININESS,mshininess);
}
void TimerFunction(int value)
{
    if(r<360.f)
        r+=1.0f;
    else
        r=0.0f;
    mx+=dx;
    if(mx>50)
```

```
            dx=-dx;
        if(mx<-100)
            dx=-dx;
        glutPostRedisplay();
        glutTimerFunc(33,TimerFunction,1);
}
void RenderScene()
{
        glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
        glPushMatrix();
        glColor3f(1.0f,1.0f,1.0f);
        glTranslatef(0,-60,0);
        glRotatef(-30,1.0,0.0,0.0);
        glRotatef(-2*r,0.0,1.0,0.0);
        glRotatef(-r,1.0,1.0,1.0);
        //NurbsFace(2,9,u,40,9,v,40,x,y,z,w,suv);
        //ShowSurface(40,40,suv,0);
        NurbsFace(2,9,u,160,9,v,160,x,y,z,w,suv);
        ShowSurface(160,160,suv,1);
        glPopMatrix();
        glutSwapBuffers();
}
```

```
void SetupRC()
{
        glClearColor(0.0f,0.0f,0.0f,0.0f);
        lightm();
        glEnable(GL_DEPTH_TEST);
        glEnable(GL_LIGHTING);
        glLightModelf(GL_LIGHT_MODEL_LOCAL_VIEWER,1.0);
        glEnable(GL_LIGHT1);
        glEnable(GL_COLOR_MATERIAL);
}
void WindowSize(GLsizei w,GLsizei h)
{
      GLfloat aspectRatio;
      GLfloat tmb=110.0;

      if(h==0)
            h=1;
      glViewport(0,0,w,h);
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      aspectRatio=(GLfloat)w/(GLfloat)h;
      if(w<=h)
```

```
            glOrtho(-tmb,tmb,-tmb/aspectRatio,tmb/aspectRatio,10*tmb,-10*tmb);
        else
            glOrtho(-tmb*aspectRatio,tmb*aspectRatio,-tmb,tmb,10*tmb,-10*tmb);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
}
int main(int argc,char *argv[])
{
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGBA);
        glutInitWindowSize(1024,768);
        glutCreateWindow("B-NURBS");
        glutDisplayFunc(RenderScene);
        glutReshapeFunc(WindowSize);
        glutTimerFunc(33,TimerFunction,1);

        SetupRC();
        glutMainLoop();
        return 0;
}
```

运行程序结果如下图所示（的瓶子光照动画画面）。