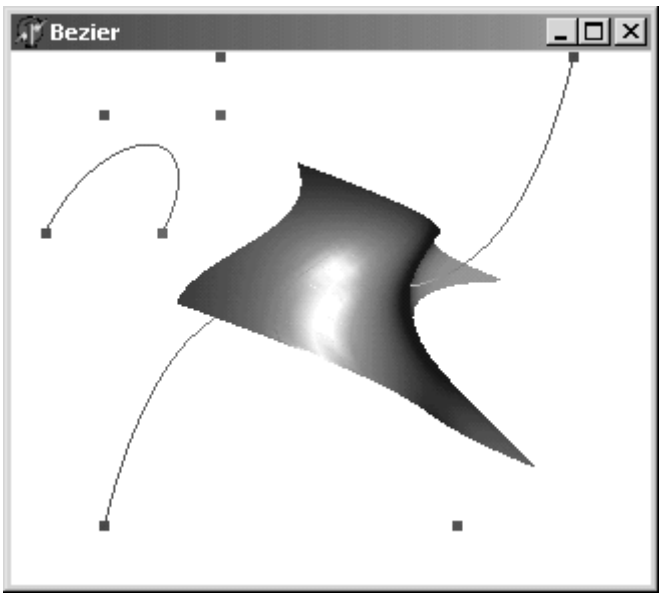
فصل دوازدهم

منحنی ها و رویه های بزییر (Bezier)



مقدمه :

در OpenGL با استفاده از ارزیاب ها (Evaluators) می توان نقـاط روی منحنـی یـا رویـه را بـا استفاده از نقاط کنترلی معین کرد و برای این کار از منحنی و سطوح بزییـر اسـتفاده مـی شـود . باید خاطر نشان کرد که OpenGL هیچ دستوری برای ترسیم منحنی ندارد . آن فقط راهی را فراهم می آورد که منحنی را به چهار گوش ها (quadrilaterals) و یا مثلـث هـا ترجمـه کنیـد . ارزیـاب هـا طریقه شکستن منحنی ها را به موارد یاد شده ، میسر می سازند .

منحنی و رویه های بزییر :

مبنای ریاضی منحنی بزییر یک تابع ترکیبی چند جمله ای است که بین نقاط ابتدا و انتها بوسیله درون یابی تولید شده باشد . نقاط منحنی بزییر از رابطه زیر بدست می آیند :

$$p(t)=\sum_{i=0}^{n} p_i\, j_{n,i}(t)$$

که در آن P_i شامل مؤلفه های برداری رئوس مختلف می باشد و t در بازه صفر و یک تغییر می کند . $j_{n,i}$ از رابطه زیر بدست می آید :

$$j_{n,i}(t)=\binom{n}{i}\times t^{i}(1-t)^{n-i}$$

که در آن :

$$\binom{n}{i}=\frac{n!}{i!(n-i)!}$$

در این معادلات n درجه چند جمله ای و i راس مورد نظر است (از صفر تا n) . عموما یک چند جمله ای درجه n با n+1 راس مشخص می شود .

مثال :

برای منحنی های درجه سوم بزییر داریم :

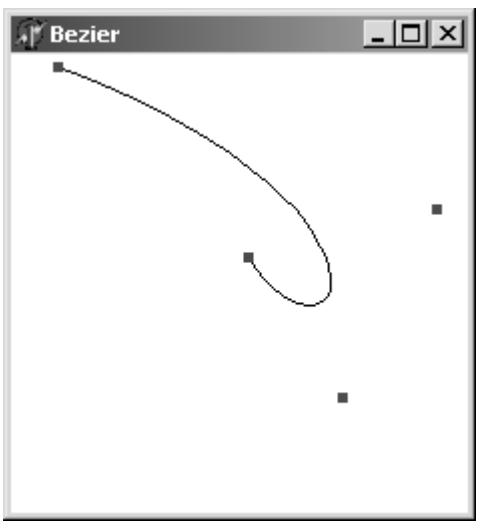
$$J_{3,1}(t) = 3t(1-t^2)$$
$$J_{3,2}(t) = 3t^2(1-t)$$
$$J_{3,0}(t) = (1-t)^3$$

نتیجه می شود که :

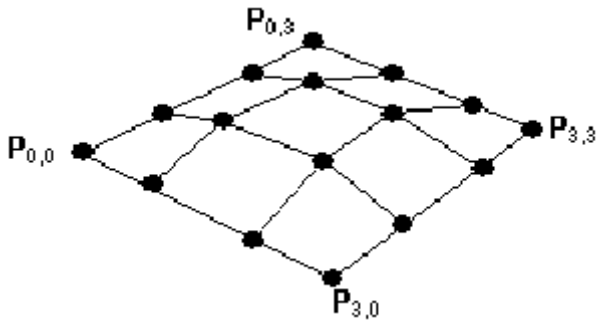
$$P(t) = p_0\, j_{3,0} + p_1\, j_{3,1} + p_2\, j_{3,2} + p_3\, j_{3,3}$$

بدین ترتیب برای ساخت یک قطعه منحنی درجه ۳ بزییر، کافی است تنها ۴ راس یک چند ضلعی را مشخص کنیم . سپس نقاط واقع در طول منحنی به ازای مقادیر t که در بازه صفر و یک قرار دارد ، محاسبه و ارزیابی می شوند .

در شکل زیر یک منحنی درجه سوم بزییر با استفاده از ۴ نقطه کنترلی ترسیم شده است :



منحنی بزییر در ۲ بعد بوسیله ۴ نقطه کنترل تعریف می شود . با استفاده از این ۴ نقطه تمام نقاط منحنی با درون یابی بدست می آیند . بطور مشابهی ، رویه ها و سطوح ۳ بعدی بزییر بوسیله شبکه ای از ۱۶ نقطه کنترلی قابل تعریف است . در این حالت ، رویه توسط ۴ ردیف که هر ردیف یک منحنی ۲ بعدی بزییر است ، تشکیل می شود (شکل زیر) .



تذکر:

۱- به هیچ عنوان نگران معادلات ریاضی فوق نباشید (!) ، زیرا OpenGL بصورت خودکار آنها را انجام می دهد .

۲- منحنی درجه سوم بزییر عمومی ترین حالت این منحنی ها می باشد .

مروری بر توابع :

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
void glMap1f(GLenum target, GLfloat u1, GLfloat u2, GLint stride, GLint order, const GLfloat *points);	Procedure glMap1f(target: GLenum; u1: GLfloat; u2: GLfloat; stride: GLint; order: GLint; points: PGLfloat); stdcall; external 'OPENGL32.DLL';

توضیح :

یک ارزیاب یک بعدی را تعریف می کند .

آرگومان target : نوع مقادیری است که توسط ارزیاب تولید می شود . توسط این مقادیر سمبولیک که در جدول زیر ارائه شده اند ، نوع و اجزاء نقاط کنترلی که در آرگومان *points* قرار می گیرند و همچنین نوع خروجی تولید شده ، تعیین خواهند شد .

برای مثال در حالت GL_MAP1_VERTEX_3 هر نقطه کنترلی از ۳ جزء x ، y و z تشکیل می شود . در این حالت دستورات glvertex3 ، توسط تابع ، بصورت درونی ، محاسبه و تولید خواهند شد.

Parameter	Meaning
GL_MAP1_VERTEX_3	*x*, *y*, *z* vertex coordinates
GL_MAP1_VERTEX_4	*x*, *y*, *z*, *w* vertex coordinates
GL_MAP1_INDEX	color index
GL_MAP1_COLOR_4	R, G, B, A
GL_MAP1_NORMAL	normal coordinates
GL_MAP1_TEXTURE_COORD_1	*s* texture coordinates
GL_MAP1_TEXTURE_COORD_2	*s*, *t* texture coordinates
GL_MAP1_TEXTURE_COORD_3	*s*, *t*, *r* texture coordinates
GL_MAP1_TEXTURE_COORD_4	*s*, *t*, *r*, *q* texture coordinates

برای فعال شدن هر کدام از نه مقدار فوق ، دستور glEnable بعلاوه مقادیر ذکر شده بالا ، باید بکار گرفته شود .

u1 و u2 : مقادیر حد پایین و بالای u هستند .

stride : مقداری که به یک نقطه کنترلی اضافه می شود تا نقطه بعدی بدست آید و یا تعداد اعداد بین ابتدای یک نقطه کنترلی و ابتدای نقطه بعدی در ساختار داده ای *points* .

order : تعداد نقاط کنترلی است که باید مثبت باشد . (=degree+1)

points : اشاره گری به آرایه نقاط کنترلی .

ملاحظات :

همانطور که ذکر شد ارزیاب ها (Evaluators) روشی هستند برای استفاده از چند جمله ای ها برای تولید رئوس ، بردارهای نرمال ، مختصات بافتی و رنگ های مربوطه . مقادیر تولید شده توسط ارزیاب ها به دستورات بعدی فرستاده خواهند شد و روی مقادیر جاری بردارهای نرمال ، مختصات بافتی و یا رنگ تاثیری ندارند .

تمام چند جمله ای ها و چند جمله ای های منطق با هر درجه ای ، می توانند با استفاده از ارزیاب ها ، تعریف شوند . ارزیاب ها ، منحنی ها را بر مبنای چند جمله ای های برنشتاین تعریف می کنند :

$$\mathbf{p}(\hat{u}) = \sum_{i=0}^{n} B_i^n(\hat{u})\mathbf{R}_i$$

R_i نقطه کنترلی است و $\hat{u}$ ، i امین چند جمله ای برنشتاین از درجه n (order=n+1) است و

$$B_i^n(\hat{u}) = \binom{n}{i} \hat{u}^i (1-\hat{u})^{n-i}$$

بطوریکه

$$0^0 \equiv 1 \text{ and } \binom{n}{0} \equiv 1$$

تابع glEvalCoord1 نگاشت های یک بعدی فعال شده توسط دستورات فوق را محاسبه می کند.

در این حالت :

$$\hat{u} = \frac{u - u1}{u2 - u1}$$

تذکر :

تغییر نقاط کنترلی پس از فراخوانی glMap1 تاثیری نخواهد داشت .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
void glEvalCoord1f(GLfloat u); void glEvalCoord1fv(const GLfloat * u);	Procedure glEvalCoord1f(u: GLfloat); stdcall; external 'OPENGL32.DLL'; Procedure glEvalCoord1fv(u: PGLfloat); stdcall; external 'OPENGL32.DLL';

توضیح :

این توابع نگاشت های یک بعدی فعال شده توسط glMap1 را محاسبه می کنند .

آرگومان u : تعیین کننده مختصات قلمرو (و یا اشاره گری به آرایه ای حاوی آن) ، تعریف شده در glMap1 است .

لازم به تذکر است که مقادیر محاسبه شده مانند رئوس ، رنگ و ... جدید ، بر روی مقادیر متناظر جاری تاثیری نخواهند گذاشت .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
void glMap2f(GLenum target, GLfloat u1, GLfloat u2, GLint ustride, GLint uorder, GLfloat v1, GLfloat v2, GLint vstride, GLint vorder, const GLfloat *points);	Procedure glMap2f(target: GLenum; u1: GLfloat; u2: GLfloat; ustride: GLint; uorder: GLint; v1: GLfloat; v2: GLfloat; vstride: GLint; vorder: GLint; points: PGLfloat); stdcall; external 'OPENGL32.DLL';

توضیح :

یک ارزیاب و محاسب ۲ بعدی را تعریف می کند .

آرگومان target : همان مقادیر ذکر شده برای تابع glMap1 می باشد ، با این تفاوت که در اینجا بجای Map1 ، Map2 قرار خواهد گرفت .

برای فعال شدن هر کدام از نه مقدار فوق ، دستور glEnable بعلاوه مقادیر ذکر آنها ، باید بکار گرفته شود .

تفاوت حالت ۲ بعدی با حالت ذکر شده یک بعدی در این است که : در اینجا تمام دستورات از u و v استفاده می کنند و نقاط ، رنگ ها ، بردارهای نرمال و مختصات بافتی ، بجای یک منحنی ، باید بر روی یک سطح و رویه تعریف شوند .

دستورالعمل استفاده از محاسبه گرهای ۲ بعدی به صورت زیر است :

۱- تعریف محاسبه گر توسط توابع glMap2*() .

۲- فعال ساختن آنها توسط glEnable و آرگومان مناسب آن .

۳- اجرای آنها توسط فراخوانی glEvalCoord2*() بین glBegin و glEnd و سپس اعمال یک شبکه توسط glMapGrid2 و glEvalMesh2 .

آرگومان های u1 و u2 : حداقل و حداکثر مقدار u هستند .

ustride : معین می کند که از نقطه R_{ij} چند مکان حافظه به جلو باید رفت تا به نقطه $R_{(i+1)j}$ رسید .

uorder : ابعاد آرایه کنترلی در محور u که باید مثبت باشد .

v1 و v2 : حداقل و حداکثر مقدار v .

vstride : معین می کند که از نقطه R_{ij} چند مکان حافظه به جلو باید رفت تا به نقطه $R_{i(j+1)}$ رسید .

vorder : ابعاد آرایه کنترلی در محور v که باید مثبت باشد .

points : اشاره گری به آرایه نقاط کنترلی .

ملاحظات :

سطوح و رویه ها بر مبنای چند جمله ای های برنشتاین بصورت زیر محاسبه خواهند شد :

$$\mathbf{p}(\hat{u},\hat{v}) = \sum_{i=0}^{n}\sum_{j=0}^{m} B_i^n(\hat{u})\, B_j^m(\hat{v})\, \mathbf{R}_{ij}$$

R_{ij} نقاط کنترلی بوده و $\hat{u}$ ، i امین چند جمله ای برنشتاین از درجه n است :

$$B_i^n(\hat{u}) = \binom{n}{i} \hat{u}^i (1-\hat{u})^{n-i}$$

$$B_j^m(\hat{v}) = \binom{m}{j} \hat{v}^j (1-\hat{v})^{m-j}$$

بطوریکه

$$0^0 \equiv 1 \text{ and } \binom{n}{0} \equiv 1$$

$$\hat{u} = \frac{u - u1}{u2 - u1}$$

$$\hat{v} = \frac{v - v1}{v2 - v1}$$

glMap2 معین می کند که چه نوع مقادیری باید تولید شوند .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
void glEvalCoord2f(GLfloat u, GLfloat v); void glEvalCoord2fv(const GLfloat * u);	Procedure glEvalCoord2f(u: GLfloat; v: GLfloat); stdcall; external 'OPENGL32.DLL'; Procedure glEvalCoord2fv(u: PGLfloat); stdcall; external 'OPENGL32.DLL';

توضیح :

سبب محاسبه بافت ۲ بعدی تعریف شده می شوند .

آرگومانهای u و v مقادیر (یا اشاره گری به مقادیر) مختصات قلمرو هستند .

توسط دستور glEnable(GL_AUTO_NORMALS); بردارهای نرمال سطح ، برای نور پردازی ، بصورت خودکار محاسبه خواهند شد (فقط در این مورد) .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
void glMapGrid1f(GLint un, GLfloat u1, GLfloat u2); void glMapGrid2f(GLint un, GLfloat u1, GLfloat u2, GLint vn, GLfloat v1, GLfloat v2);	Procedure glMapGrid1f(un: GLint; u1: GLfloat; u2: GLfloat); stdcall; external 'OPENGL32.DLL'; Procedure glMapGrid2f(un: GLint; u1: GLfloat; u2: GLfloat; vn: GLint; v1: GLfloat; v2: GLfloat); stdcall; external 'OPENGL32.DLL';

توضیح :

glMapGrid1f شبکه ای یک بعدی را تعریف می کند .

un : تعداد قطعات اجزاء شبکه در بازه u1 و u2 می باشد که باید مثبت باشد .

u1 و u2 : قلمرو نگاشت شبکه از i=0 تا i=un .

از این تابع برای تعریف بهینه تولید نگاشت قلمرو با فواصل مساوی ، استفاده می شود . تابع glEvalMesh سپس عملیات محاسبه و تولید را انجام خواهد داد .

glMapGrid2f : همانند نگارش یک بعدی آن است با این تفاوت که اطلاعات u و v باید به آن اضافه شود .

آرگومان vn : تعداد قطعات شبکه در بازه v1 و v2 است .

آرگومانهای v1 و v2 : قلمرو نگاشت شبکه از j=0 تا j=vn است .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
void glEvalMesh1(GLenum mode, GLint i1, GLint i2); void glEvalMesh2(GLenum mode, GLint i1, GLint i2, GLint j1, GLint j2);	Procedure glEvalMesh1(mode: GLenum; i1: GLint; i2: GLint); stdcall; external 'OPENGL32.DLL'; Procedure glEvalMesh2(mode: GLenum; i1: GLint; i2: GLint; j1: GLint; j2: GLint); stdcall; external 'OPENGL32.DLL';

توضیح :

glEvalMesh1 شبکه ای از نقاط یا خطوط یک بعدی تعریف شده را محاسبه می کند .

آرگومان Mode : می تواند GL_POINT یا GL_LINE باشد .

i1 و i2 : مقادیر ابتدا و انتهای قلمرو متغیر ها هستند .

این تابع با دستورات زیر معادل است :

به فرمت زبان C	به فرمت زبان دلفی

glBegin(GL_POINTS);
/* OR glBegin(GL_LINE_STRIP); */
for (i = i1; i <= i2; i++)
glEvalCoord1(u1 + i*(u2-u1)/n);
glEnd();
	glBegin(GL_POINTS);
	// OR glBegin(GL_LINE_STRIP);
	for i := i1 to i2 do
	glEvalCoord1(u1 + i*(u2-u1)/n);
	glEnd();

glEvalMesh2 نگارش ۲ بعدی تابع glEvalMesh1 است و آرگومان mode آن : GL_FILL نیز می تواند باشد .

j1 و j2 : مقادیر ابتدا و انتهای قلمرو متغیر j هستند .

این تابع با کدهای زیر نیز معادل است :

به فرمت زبان C	به فرمت زبان دلفی

glBegin(GL_POINTS);
/* mode == GL_POINT */
for (i = nu1; i <= nu2; i++)
for (j = nv1; j <= nv2; j++)
glEvalCoord2(
u1 + i*(u2-u1)/nu, v1+j*(v2-v1)/nv);
glEnd();
or
for (i = nu1; i <= nu2; i++) {
/* mode == GL_LINE */
glBegin(GL_LINES);
for (j = nv1; j <= nv2; j++)
glEvalCoord2(
u1 + i*(u2-u1)/nu, v1+j*(v2-v1)/nv);
glEnd();
}
for (j = nv1; j <= nv2; j++) {
glBegin(GL_LINES);
for (i = nu1; i <= nu2; i++)
glEvalCoord2(
u1 + i*(u2-u1)/nu, v1+j*(v2-v1)/nv);
glEnd();
}
or
for (i = nu1; i < nu2; i++) {
/* mode == GL_FILL */
glBegin(GL_QUAD_STRIP);
for (j = nv1; j <= nv2; j++) {
glEvalCoord2(
u1 + i*(u2-u1)/nu, v1+j*(v2-v1)/nv);
glEvalCoord2(
u1 + (i+1)*(u2-u1)/nu, v1+j*(v2-v1)/nv);
glEnd();
}
}
	glBegin(GL_POINTS);
	// mode == GL_POINT
	for i := nu1 to nu2 do
	for j := nv1 to nv2 do
	glEvalCoord2(
	u1 + i*(u2-u1)/nu, v1+j*(v2-v1)/nv);
	glEnd();
	or
	for i := nu1 to nu2 do
	begin
	// mode == GL_LINE
	glBegin(GL_LINES);
	for j := nv1 to nv2 do
	glEvalCoord2(
	u1 + i*(u2-u1)/nu, v1+j*(v2-v1)/nv);
	glEnd();
	End;
	for j := nv1 to nv2 do
	begin
	glBegin(GL_LINES);
	for i := nu1 to nu2 do
	glEvalCoord2(
	u1 + i*(u2-u1)/nu, v1+j*(v2-v1)/nv);
	glEnd();
	End;
	or
	for i := nu1 to nu2 do
	begin
	// mode == GL_FILL
	glBegin(GL_QUAD_STRIP);
	for j := nv1 to nv2 do
	begin
	glEvalCoord2(
	u1 + i*(u2-u1)/nu, v1+j*(v2-v1)/nv);
	glEvalCoord2(
	u1 + (i+1)*(u2-u1)/nu, v1+j*(v2-v1)/nv);
	glEnd();
	End;
	End;

برنامه فصل :

در برنامه زیر طرز استفاده از توابع فوق را فرا خواهید گرفت :

```
unit Ch12;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics,
   Controls, Forms, Dialogs, OpenGL, SPF;

type
  TForm1 = class(TForm)
    procedure FormPaint(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure FormResize(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;

  f_Hdc : LongInt;

implementation
{$R *.DFM}

type TPoints= array[0..3,0..2] of GLfloat;
var
  ctrlpoints1 : TPoints = (
        ( -4.0, -4.0, 0.0), ( -2.0, 4.0, 0.0),
        (2.0, -4.0, 0.0), (4.0, 4.0, 0.0));

  ctrlpoints2 : TPoints = (
        ( 1.0, 1.0, 0.0), ( 2.0, 3.0, 0.0),
        (4.0, 3.0, 0.0), (3.0, 1.0, 0.0));

  ctrlpoints3 : TPoints = (
        ( -3.0, 2.0, 0.0), ( 4.0, 1.0, 0.0),
        (2.0, -3.0, 0.0), (0.0, 0.0, 0.0));

type TPoints3D= array[0..3,0..3,0..2] of GLfloat;
var
   ctrlpoints3D : TPoints3D=  (
   ((-1.5, -1.5, 4.0), (-0.5, -1.5, 2.0),
    (0.5, -1.5, -1.0), (1.5, -1.5, 2.0)),
   ((-1.5, -0.5, 1.0), (-0.5, -0.5, 3.0),
    (0.5, -0.5, 0.0), (1.5, -0.5, -1.0)),
   ((-1.5, 0.5, 4.0), (-0.5, 0.5, 0.0),
    (0.5, 0.5, 3.0), (1.5, 0.5, 4.0)),
   ((-1.5, 1.5, -2.0), (-0.5, 1.5, -2.0),
    (0.5, 1.5, 0.0), (1.5, 1.5, -1.0))
);
```

```
  ambient : array[0..3] of GLfloat =(0.2, 0.2, 0.2, 1.0);
  position : array[0..3] of GLfloat =(0.0, 0.0, 2.0, 1.0);
  mat_diffuse :array[0..3] of GLfloat = (0.6, 0.6, 0.6, 1.0);
  mat_specular : array[0..3] of GLfloat = (1.0, 1.0, 1.0, 1.0);
  mat_shininess : array[0..0] of GLfloat =( 50.0 );

procedure initLights();
begin
  glEnable(GL_LIGHTING);
  glEnable(GL_LIGHT0);

  glLightfv(GL_LIGHT0, GL_AMBIENT, @ambient);
  glLightfv(GL_LIGHT0, GL_POSITION, @position);

  glMaterialfv(GL_FRONT, GL_DIFFUSE, @mat_diffuse);
  glMaterialfv(GL_FRONT, GL_SPECULAR, @mat_specular);
  glMaterialfv(GL_FRONT, GL_SHININESS, @mat_shininess);
end;

procedure initGL();
begin
  glClearColor(1.0, 1.0, 1.0, 0.0);
  glEnable(GL_DEPTH_TEST);
  glEnable(GL_MAP1_VERTEX_3);
  glEnable(GL_MAP2_VERTEX_3);
  glEnable(GL_AUTO_NORMAL); // for lighting
  glMapGrid2f(20, 0.0, 1.0, 20, 0.0, 1.0);
  initlights();
end;

procedure DrawBezierCurve(xTrans,yTrans : single;
                    ctrlpoints : TPoints );
var
  i : Integer;
begin
  glColor3f(0.0,0.0, 0.0);
  glPushMatrix();
    glTranslatef(xTrans,yTrans,0);
    glMap1f(GL_MAP1_VERTEX_3, 0.0, 1.0, 3, 4, @ctrlpoints);
    glBegin(GL_LINE_STRIP);
     for i := 0 to 30  do
       glEvalCoord1f(i/30.0);
    glEnd();
    //    The following code displays the control points as dots.
    glPointSize(5.0);
    glColor3f(Random, Random,0);
    glBegin(GL_POINTS);
     for i := 0 to  4-1 do
       glVertex3fv(@ctrlpoints[i,0]);
     glEnd();
  glPopMatrix();
  glFlush();
end;

procedure DrawBezierSurface(xTrans,yTrans,zTrans : single;
                     ctrlpoints : TPoints3D );
var
  i , j : Integer;
```

```
begin

glColor3f(0.0, 0.0, 0.0);

   glMap2f(GL_MAP2_VERTEX_3, 0, 1, 3, 4,
          0, 1, 12, 4, @ctrlpoints);

   glPushMatrix ();
   glTranslatef(xTrans,yTrans,zTrans);
   glRotatef(85.0, 1.0, 1.0, 1.0);
   glEvalMesh2(GL_FILL, 0, 20, 0, 20); // for Ligthing
    for j :=0 to 8 do
     begin
      glBegin(GL_LINE_STRIP);
        for i := 0 to  30 do
          glEvalCoord2f(i/30.0, j/8.0);
      glEnd();
      glBegin(GL_LINE_STRIP);
        for i := 0 to  30 do
          glEvalCoord2f(j/8.0, i/30.0);
      glEnd();
   end;
   glPopMatrix ();
   glFlush();

end;

procedure DrawGLScene();
begin
  glClear(GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT);

  DrawBezierCurve(0,0,ctrlpoints1);
  DrawBezierCurve(-6,0,ctrlpoints2);
//  DrawBezierCurve(3,-3,ctrlpoints3);

  DrawBezierSurface(0,0,0,ctrlpoints3D);

  swapBuffers(f_Hdc);
end;

procedure TForm1.FormPaint(Sender: TObject);
begin
 wglMakeCurrent(f_Hdc,hrc); //activate the RC
 DrawGLScene();
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
 f_Hdc := GetDC(handle);
 SetDCPixelFormat(f_Hdc,16,16);// Create a rendering context.
 InitGL();
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
 CleanUp(f_Hdc);// Clean up and terminate.
end;
```

```
procedure TForm1.FormResize(Sender: TObject);
begin
  wglMakeCurrent(f_Hdc,hrc); //activate the RC
  // Prevent A Divide By Zero If The Window Is Too Small
  if (Height=0)
    then Height:=1;
   glViewport(0, 0, width, Height );
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   if (width <= Height)  then
      glOrtho(-5.0, 5.0, -5.0*Height/width,
              5.0*Height/width, -5.0, 5.0)
   else
      glOrtho(-5.0*width/Height,
              5.0*width/Height, -5.0, 5.0, -5.0, 5.0);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
  InvalidateRect(Handle, nil, False);// DrawGLScene;  Draw the scene.
end;

end.
```