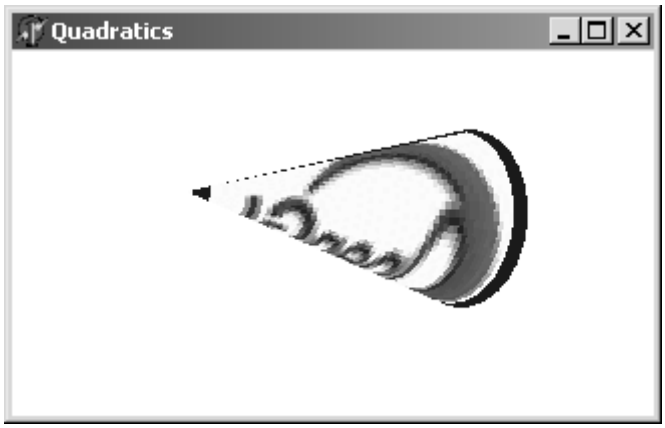
فصل دهم

سطوح و اشکال درجه دوم (Quadratics)



مقدمه :

کتابخانه OpenGL تنها ترسیم نقاط ، خطوط و چند ضلعی ها را میسر می سازد . کتابخانه کمکی OpenGL یا GLU حاوی چند گروه تابع است که مکمل توابع OpenGL می باشند . کتابخانه GLU برای ایجاد اشکال ۲و۳ بعدی مرسوم (مانند : کره ، استوانه و غیره) از OpenGL مشتق شده است . این کتابخانه کمکی از توابع OpenGL استفاده می کند ؛ بنابراین هر نوع نگارشی از OpenGL این توابع را پشتیبانی خواهند کرد . همانطور که پیشتر نیز ذکر شد، توابع و ثوابت این مجموعه (glu32.dll) با glu شروع می شوند . یکی از این مجموعه ها ، Quadratics نام دارد .

اشیاء درجه دوم :

Quadratics روش ترسیم اشیاء پیچیده است توسط تعدادی حلقه for و مقداری آشنایی با مثلثات . تابع gluNewQuadric اشاره گری را به محلی که آن در حافظه ذخیره می شود ، تولید می نماید . تابع gluQuadricNormals بردارهای نرمال لازم برای عملیات نورپردازی را ایجاد می کند وسبب gluQuadricTexture فعال شدن نگاشت بافت ها برروی اشیاء درجه دوم می گردد.

در ادامه مروری خواهیم داشت بر توابع گروه اشیاء درجه دوم کتابخانه کمکی OpenGL .

مروری بر توابع :

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
GLUquadricObj* gluNewQuadric (void);	type TGLUquadricObj = record end; PGLUquadricObj = ^TGLUquadricObj; Function gluNewQuadric: PGLUquadricObj; stdcall; external 'GLU32.DLL';

توضیح :

یک شیء درجه دوم خلق می کند و اشاره گری را به شیء ایجاد شده ، باز می گرداند . اگر خروجی این تابع صفر باشد ، بدین معنا است که حافظه کافی برای انجام این کار وجود ندارد.

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
void gluDeleteQuadric(GLUquadricObj *state);	Procedure gluDeleteQuadric(state: PGLUquadricObj); stdcall; external 'GLU32.DLL';

توضیح :

شیء درجه دوم را تخریب کرده و حافظه تخصیص یافته شده توسط آنرا آزاد می نماید.

آرگومان state : شیء درجه دومی است که باید تخریب شود (توسط تابع gluNewQuadric ایجاد شده است) .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
void gluQuadricCallback(GLUquadricObj *qobj, GLenum which, void (* fn)());	type TGLQuadricErrorProc = Procedure(errcode: GLenum); stdcall; Procedure gluQuadricCallback(qobj: PGLUquadricObj; which: GLenum; callback: TGLQuadricErrorProc); stdcall; external 'GLU32.DLL';

توضیح :

یک CallBack برای شیء درجه دوم تعریف می کند .

آرگومان qobj : شیء درجه دومی است که توسط تابعgluNewQuadric ایجاد گردیده است .

آرگومان which : مقدار مجاز آن GLU_ERROR می باشد . این تابع هنگامی که خطایی رخ دهد فراخوانی می شود .

fn : تابعی که باید فراخوانی گردد.

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
void gluQuadricNormals(GLUquadricObj *qobj, GLenum normals);	Procedure gluQuadricNormals(quadObject: PGLUquadricObj; normals: GLenum); stdcall; external 'GLU32.DLL';

توضیح :

معین می کند که چه نوع بردار نرمالی باید برای شیء درجه دوم بکار رود .

آرگومان qobj : شیء درجه دومی است که توسط تابعgluNewQuadric ایجاد گردیده است .

normals : نوع بردار نرمال مورد نیاز است . مقادیر زیر مجاز هستند :

GLU_NONE : بردار نرمالی تولید نمی شود .

GLU_FLAT : یک بردار نرمال به ازای هر وجه شیء درجه دوم تعریف می شود .

GLU_SMOOTH : یک بردار نرمال به ازای هر راس یک شیء درجه دوم تولید می گردد (و مقدار پیش فرض می باشد) .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
void gluQuadricTexture(GLUquadricObj *quadObject, GLboolean textureCoords);	Procedure gluQuadricTexture(quadObject: PGLUquadricObj; textureCoords: GLboolean); stdcall; external 'GLU32.DLL';

توضیح :

معین می سازد که باید روی کدام شیء درجه دوم ، نگاشت بافتی صورت گیرد .

آرگومان quadObject : شیء درجه دومی است که توسط تابع gluNewQuadric ایجاد گردیده است .

textureCoords : پرچمی است که ایجاد و یا عدم ایجاد مختصات بافتی را معین نموده و مقادیر زیر را می پذیرد :

GLU_TRUE : مختصات بافتی تولید می شود .

GLU_FALSE : مختصات بافتی تولید نمی شود .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
void gluQuadricOrientation(GLUquadricObj *quadObject, GLenum orientation);	Procedure gluQuadricOrientation(quadObject: PGLUquadricObj; orientation: GLenum); stdcall; external 'GLU32.DLL';

توضیح :

جهت رو به درون و یا رو به خارج بودن بردارهای نرمال شیء درجه دوم را معین می کند.

آرگومان quadObject : شیء درجه دومی است که توسط تابع gluNewQuadric ایجاد گردیده است .

orientation : جهت مورد نیاز . مقادیر زیر برای آن مجاز هستند .

GLU_OUTSIDE : شیء درجه دومی را ترسیم می کند که بردارهای نرمال آن رو به خارج آن اشاره می نمایند (و مقدار پیش فرض می باشد) .

GLU_INSIDE : شیء درجه دومی را ترسیم می کند که بردارهای نرمال آن رو به درون آن اشاره می نمایند .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
void gluQuadricDrawStyle(GLUquadricObj *qobj, GLenum drawStyle);	Procedure gluQuadricDrawStyle(quadObject: PGLUquadricObj; drawStyle: GLenum); stdcall; external 'GLU32.DLL';

توضیح :

شیوه ترسیم شیء درجه دوم را معین می کند .

آرگومان qobj : شیء درجه دومی است که توسط تابعgluNewQuadric ایجاد گردیده است .

drawStyle : شیوه ترسیم مورد نیاز بوده و یکی از مقادیر زیر می تواند باشد :

GLU_FILL : اشیاء درجه دوم توسط چند ضلعی ها ترسیم خواهند شد . چند ضلعی ها در خلاف عقربه های ساعت ، متناظر با بردار نرمال مربوطه اشان ترسیم خواهند شد .

GLU_LINE : اشیاء درجه دوم توسط خطوط ترسیم خواهند شد .

GLU_SILHOUETTE : اشیاء درجه دوم توسط مجموعه ای از خطوط ترسیم خواهند شد ، بجز لبه های دو صفحه متقاطع .

GLU_POINT : اشیاء درجه دوم توسط مجموعه ای از نقاط ترسیم خواهند شد .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
void gluSphere(GLUquadricObj *qobj, GLdouble radius, GLint slices, GLint stacks);	Procedure gluSphere(qobj: PGLUquadricObj; radius: GLdouble; slices: GLint; stacks: GLint); stdcall; external 'GLU32.DLL';

توضیح :

یک کره را به مرکز مبدا حول Z ترسیم می نماید .

آرگومان qobj : شیء درجه دومی است که توسط تابعgluNewQuadric ایجاد گردیده است .

radius : شعاع کره .

slices : تعداد تقسیم های فرعی در طول محور Z . (مشابه خطوط طولی)

stacks : تعداد تقسیمات فرعی حول محور Z . (مشابه خطوط عرضی)

هر چه تعداد تقسیمات فرعی بیشتر باشد ، شیء با جزئیات بیشتری ترسیم خواهد شد ، اما سرعت نیز کاهش می یابد .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
void gluCylinder(GLUquadricObj *qobj, GLdouble baseRadius, GLdouble topRadius, GLdouble height, GLint slices, GLint stacks);	Procedure gluCylinder(qobj: PGLUquadricObj; baseRadius: GLdouble; topRadius: GLdouble; height: GLdouble; slices: GLint; stacks: GLint); stdcall; external 'GLU32.DLL';

توضیح :

یک استوانه را در امتداد محور Z ترسیم مـی کنـد . قاعـده آن در Z=0 قـرار دارد و راس آن در Z=Height .

آرگومان qobj : شیء درجه دومی است که توسط تابعgluNewQuadric ایجاد گردیده است .

baseRadius : شعاع استوانه در صفحه Z=0 .

topRadius : شعاع استوانه در Z=Height .

height : ارتفاع استوانه .

slices : تعداد تقسیم های فرعی در طول محور Z . (مشابه خطوط طولی)

stacks : تعداد تقسیمات فرعی حول محور Z . (مشابه خطوط عرضی)

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
void gluDisk(GLUquadricObj *qobj, GLdouble innerRadius, GLdouble outerRadius, GLint slices, GLint loops);	Procedure gluDisk(qobj: PGLUquadricObj; innerRadius: GLdouble; outerRadius: GLdouble; slices: GLint; loops: GLint); stdcall; external 'GLU32.DLL';

توضیح :

یک قرص را در صفحه Z=0 ترسیم می کند .

آرگومان qobj : شیء درجه دومی است که توسط تابعgluNewQuadric ایجاد گردیده است .

innerRadius : شعاع داخلی قرص .

outerRadius : شعاع خارجی قرص .

slices : تعداد تقسیم های فرعی در طول محور Z . (مشابه خطوط طولی)

loops : تعداد حلقه های هم مرکز ، حول مرکز .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
void gluPartialDisk(GLUquadricObj *qobj, GLdouble innerRadius, GLdouble outerRadius, GLint slices, GLint loops, GLdouble startAngle, GLdouble sweepAngle);	Procedure gluPartialDisk(qobj: PGLUquadricObj; innerRadius: GLdouble; outerRadius: GLdouble; slices: GLint; loops: GLint; startAngle: GLdouble; sweepAngle: GLdouble); stdcall; external 'GLU32.DLL';

توضیح :

قوسی از یک قرص را روی صفحه Z=0 ترسیم می نماید .

آرگومان qobj : شیء درجه دومی است که توسط تابعgluNewQuadric ایجاد گردیده است .

innerRadius : شعاع داخلی قرص .

outerRadius : شعاع خارجی قرص .

slices : تعداد تقسیم های فرعی در طول محور Z . (مشابه خطوط طولی)

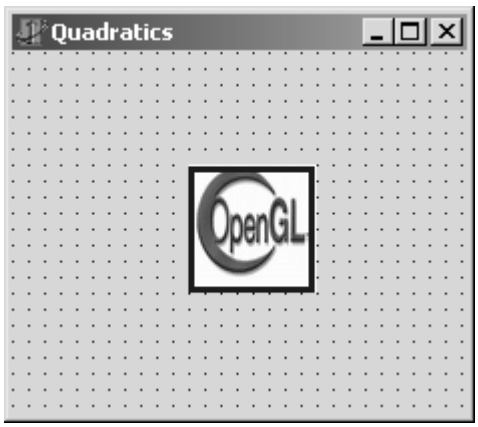
loops : تعداد حلقه های هم مرکز ، حول مرکز .

startAngle : زاویه آغازین ، قوس به درجه .

sweepAngle : زاویه روبنده قرص به درجه .

برنامه فصل :

برای اجرای برنامه زیر به یک کنترل Image که در آن تصویری به ابعاد ۶۴×۶۴ بارگذاری شده است نیاز می باشد (شکل زیر) . با فشردن کلید Space اشکال متفاوتی را خواهید دید .




```
unit Ch10;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
   Forms, Dialogs, OpenGL, SPF, ExtCtrls ;

type
  TForm1 = class(TForm)
    Image1: TImage;
    procedure FormResize(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormPaint(Sender: TObject);
    procedure FormKeyDown(Sender: TObject; var Key: Word;
      Shift: TShiftState);
  private
    { Private declarations }
```

```
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

  f_Hdc : LongInt;

implementation

{$R *.DFM}

procedure glGenTextures(n: GLsizei; var textures: GLuint);
stdcall; external 'opengl32.dll';
procedure glBindTexture(target: GLenum; texture: GLuint);
stdcall; external 'opengl32.dll';

// The gluBuild2DMipmaps declaration in the
//Delphi 4 version of the OpenGL unit
// was improperly declared. I've redeclared it here:=
function gluBuild2DMipmaps(target: GLenum; components,
 width, height: GLint; format, atype: GLenum; Data: Pointer): GLint;
 stdcall; external glu32;

var

  light : bool;                         // Lighting ON/OFF

  part1 : GLuint;                       // Start Of Disc
  part2 : GLuint;                       // End Of Disc
  p1 : GLuint = 0;                      // Increase 1
  p2 : GLuint = 1;                      // Increase 2

  xrot : GLfloat;               // X Rotation
  yrot : GLfloat;               // Y Rotation
  xspeed : GLfloat;                     // X Rotation Speed
  yspeed : GLfloat;                     // Y Rotation Speed
  z : GLfloat = -5.0;                   // Depth Into The Screen

  quadratic : GLUquadricObj;          // Storage For Our Quadratic Objects

  LightAmbient : array [0..3] of GLfloat = (0.5, 0.5, 0.5, 1.0);
  LightDiffuse : array [0..3] of GLfloat = (1.0, 1.0, 1.0, 1.0);
  LightPosition : array [0..3] of GLfloat = (0.0, 0.0, 2.0, 1.0);

  filter : GLuint;                      // Which Filter To Use
  texture : array [0..2] of GLuint;             // Storage For 3 Textures
  selectedobject : GLuint = 0;                  // Which Object To Draw


procedure getRGB(num : LongInt; var r , g , b : Integer);
begin
//  extract R,G,B from a long format RBG
      b := Trunc((num And 16711680)/ 65536) ;
      g := Trunc((num And 65280)/ 256 );
      r :=  num  And 255 ;
End;
```

```
// Load Bitmaps And Convert To Textures
function LoadGLTextures (pctPicToLoad : TImage): boolean;
var
  Status : boolean;                // Status Indicator
  x , y : LongInt;
  c ,bitmapWidth, bitmapHeight : LongInt ;
  Red , Green , Blue : Integer ;
  TextureImage: array[0..63,0..63,0..2] of  GLubyte;
begin
  Status := FALSE;
   // create the Texture
    pctPicToLoad.AutoSize := True ;
    bitmapHeight := pctPicToLoad.Height ;  // Set the array
    bitmapWidth := pctPicToLoad.Width ; // size.

 For x := 0 To  bitmapWidth-1  do
     For y := 0 To bitmapHeight-1  do
       begin
         c := ColorToRGB(pctPicToLoad.Canvas.Pixels[x,y]);
         getRGB(c,Red,Green,Blue);
         TextureImage[ x, bitmapHeight - y - 1,0] := red ;
         TextureImage[ x, bitmapHeight - y - 1,1] := green ;
         TextureImage[ x, bitmapHeight - y - 1,2] := blue ;
       end;
       pctPicToLoad.Visible:=false;


  if (@TextureImage <> nil) then
  begin
    Status := TRUE;                    // Set The Status To TRUE
    glGenTextures(3, texture[0]);          // Create Two Textures

  // Create Nearest Filtered Texture
  glGenTextures(3, texture[0]);
  glBindTexture(GL_TEXTURE_2D, texture[0]);
  glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_NEAREST);
  glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_NEAREST);
  glTexImage2D(GL_TEXTURE_2D, 0, 3, bitmapWidth, bitmapHeight,
  0, GL_RGB, GL_UNSIGNED_BYTE, @TextureImage);

  // Create Linear Filtered Texture
  glBindTexture(GL_TEXTURE_2D, texture[1]);
  glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
  glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR);
  glTexImage2D(GL_TEXTURE_2D, 0, 3, bitmapWidth, bitmapHeight,
  0, GL_RGB, GL_UNSIGNED_BYTE, @TextureImage);

  // Create MipMapped Texture
  glBindTexture(GL_TEXTURE_2D, texture[2]);
  glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
  glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,
                GL_LINEAR_MIPMAP_NEAREST);
  gluBuild2DMipmaps(GL_TEXTURE_2D, 3, bitmapWidth, bitmapHeight,
   GL_RGB, GL_UNSIGNED_BYTE, @TextureImage);
  end;

  result := Status;                  // Return The Status
```

```
end;

function InitGL : boolean;  // All Setup For OpenGL Goes Here
begin
  // Jump To Texture Loading Routine
  if (LoadGLTextures(form1.image1) = false) then
  begin
    result := FALSE;        // If Texture Didn't Load Return FALSE
    exit;
  end;

  glEnable(GL_TEXTURE_2D);              // Enable Texture Mapping
  glShadeModel(GL_SMOOTH);          // Enable Smooth Shading
  glClearColor(1.0, 1.0, 1.0, 0.5);          // White Background
  glClearDepth(1.0);                         // Depth Buffer Setup
  glEnable(GL_DEPTH_TEST);              // Enables Depth Testing
  glDepthFunc(GL_LEQUAL);               // The Type Of Depth Testing To Do
  // Really Nice Perspective Calculations
  glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);

  glLightfv(GL_LIGHT1, GL_AMBIENT, @LightAmbient);  // Setup The Ambient Light
  glLightfv(GL_LIGHT1, GL_DIFFUSE, @LightDiffuse);  // Setup The Diffuse Light
  glLightfv(GL_LIGHT1, GL_POSITION,@LightPosition); // Position The Light
  glEnable(GL_LIGHT1);                                  // Enable Light One
  // Create A Pointer To The Quadric Object (Return 0 If No Memory)
  quadratic := gluNewQuadric();
  gluQuadricNormals(quadratic, GLU_SMOOTH);   // Create Smooth Normals
  gluQuadricTexture(quadratic, GL_TRUE);     // Create Texture Coords

  result := TRUE;                                       // Initialization Went OK
end;

procedure glDrawCube;
begin
  glBegin(GL_QUADS);
    // Front Face
    glNormal3f( 0.0, 0.0, 1.0);
    glTexCoord2f(0.0, 0.0); glVertex3f(-1.0, -1.0,  1.0);
    glTexCoord2f(1.0, 0.0); glVertex3f( 1.0, -1.0,  1.0);
    glTexCoord2f(1.0, 1.0); glVertex3f( 1.0,  1.0,  1.0);
    glTexCoord2f(0.0, 1.0); glVertex3f(-1.0,  1.0,  1.0);
    // Back Face
    glNormal3f( 0.0, 0.0,-1.0);
    glTexCoord2f(1.0, 0.0); glVertex3f(-1.0, -1.0, -1.0);
    glTexCoord2f(1.0, 1.0); glVertex3f(-1.0,  1.0, -1.0);
    glTexCoord2f(0.0, 1.0); glVertex3f( 1.0,  1.0, -1.0);
    glTexCoord2f(0.0, 0.0); glVertex3f( 1.0, -1.0, -1.0);
    // Top Face
    glNormal3f( 0.0, 1.0, 0.0);
    glTexCoord2f(0.0, 1.0); glVertex3f(-1.0,  1.0, -1.0);
    glTexCoord2f(0.0, 0.0); glVertex3f(-1.0,  1.0,  1.0);
    glTexCoord2f(1.0, 0.0); glVertex3f( 1.0,  1.0,  1.0);
    glTexCoord2f(1.0, 1.0); glVertex3f( 1.0,  1.0, -1.0);
    // Bottom Face
    glNormal3f( 0.0,-1.0, 0.0);
    glTexCoord2f(1.0, 1.0); glVertex3f(-1.0, -1.0, -1.0);
    glTexCoord2f(0.0, 1.0); glVertex3f( 1.0, -1.0, -1.0);
    glTexCoord2f(0.0, 0.0); glVertex3f( 1.0, -1.0,  1.0);
```

```
    glTexCoord2f(1.0, 0.0); glVertex3f(-1.0, -1.0,  1.0);
    // Right Face
    glNormal3f( 1.0, 0.0, 0.0);
    glTexCoord2f(1.0, 0.0); glVertex3f( 1.0, -1.0, -1.0);
    glTexCoord2f(1.0, 1.0); glVertex3f( 1.0,  1.0, -1.0);
    glTexCoord2f(0.0, 1.0); glVertex3f( 1.0,  1.0,  1.0);
    glTexCoord2f(0.0, 0.0); glVertex3f( 1.0, -1.0,  1.0);
    // Left Face
    glNormal3f(-1.0, 0.0, 0.0);
    glTexCoord2f(0.0, 0.0); glVertex3f(-1.0, -1.0, -1.0);
    glTexCoord2f(1.0, 0.0); glVertex3f(-1.0, -1.0,  1.0);
    glTexCoord2f(1.0, 1.0); glVertex3f(-1.0,  1.0,  1.0);
    glTexCoord2f(0.0, 1.0); glVertex3f(-1.0,  1.0, -1.0);
  glEnd();
end;

function DrawGLScene : boolean;          // Here's Where We Do All The Drawing
begin
  // Clear The Screen And The Depth Buffer
  glClear(GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT);
  glLoadIdentity();                  // Reset The View
  glTranslatef(0.0,0.0,z);

  glRotatef(xrot,1.0,0.0,0.0);
  glRotatef(yrot,0.0,1.0,0.0);

  glBindTexture(GL_TEXTURE_2D, texture[filter]);

  case selectedobject of
    0:   glDrawCube();
    1:   begin
           glTranslatef(0.0,0.0,-1.5);              // Center The Cylinder
           // A Cylinder With A Radius Of 0.5 And A Height Of 2
           gluCylinder(quadratic,1.0,1.0,3.0,32,32);
         end;
         // Draw A Disc (CD Shape) With An Inner Radius Of 0.5,
         // And An Outer Radius Of 2.  Plus A Lot Of Segments ;)
    2:   gluDisk(quadratic,0.5,1.5,32,32);
         // Draw A Sphere With A Radius Of 1 And 16 Longitude
         // And 16 Latitude Segments
    3:   gluSphere(quadratic,1.3,32,32);
    4:   begin
           glTranslatef(0.0,0.0,-1.5);              // Center The Cone
           // A Cone With A Bottom Radius Of .5 And A Height Of 2
           gluCylinder(quadratic,1.0,0.0,3.0,32,32);
         end;
    5:   begin
           part1 := part1 + p1;
           part2 := part2 + p2;

           if (part1>359) then                              // 360 Degrees
           begin
             p1:=0;
             part1:=0;
             p2:=1;
             part2:=0;
           end;
           if (part2>359) then              // 360 Degrees
```

```
          begin
            p1:=1;
            p2:=0;
          end;
          // A Disk Like The One Before
          gluPartialDisk(quadratic,0.5,1.5,32,32,part1,part2-part1);
        end;
  end;

  xrot := xrot + xspeed;
  yrot := yrot + yspeed;        // Increase The Second Counter

  swapBuffers(f_Hdc);

  result := TRUE;                                  // Everything Went OK
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);             // Reset The Current Viewport
  glMatrixMode(GL_PROJECTION);                    // Select The Projection Matrix
  glLoadIdentity();                         // Reset The Projection Matrix
  // Calculate Window Aspect Ratio
  gluPerspective(45.0,Width/Height,0.1,100.0);
  glMatrixMode(GL_MODELVIEW);           // Select The Modelview Matrix
  glLoadIdentity();                   // Reset The Modelview Matrix
  InvalidateRect(Handle, nil, False);// DrawGLScene;  Draw the scene.
end;

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

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

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

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin

    if key=ord('L')  then
    begin
```

```
    light := not(light);
    if light then
      glEnable(GL_LIGHTING)
    else
      glDisable(GL_LIGHTING);
  end;

  if key=ord('F')  then
  begin
    filter := filter + 1;
    if (filter>2) then filter := 0;
  end;

  if key=32  then
  begin
    selectedobject := selectedobject + 1;
    if (selectedobject>5) then selectedobject := 0;
  end;

  if (key=39) then xspeed := xspeed - 0.1;
  if (key=37) then xspeed := xspeed + 0.1;
  if (key=38) then yspeed := yspeed + 0.1;
  if (key=40) then yspeed := yspeed - 0.1;

 Application.ProcessMessages; //DoEvents
 InvalidateRect(Handle, nil, False);//   DrawGLScene();
end;

end.
```