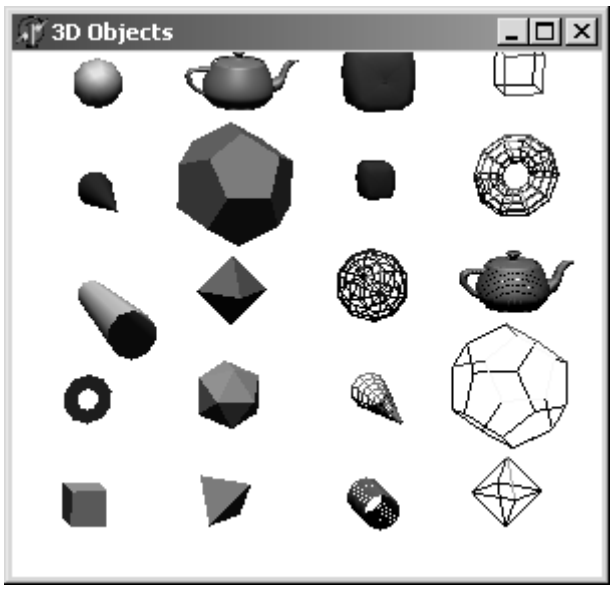
فصل یازدهم

ایجاد کتابخانه اشیاء سه بعدی



مقدمه :

در این فصل قصد داریم یک کتابخانه با پیوند پویا (DLL) برای ایجاد اشیاء سه بعدی مرسوم بوجود آوریم . سپس در برنامه ای طریقه استفاده از آن ذکر خواهد گردید . این کتابخانه از ترجمه کتابخانه های GLAUX و GLUT ، به زبان دلفی ، که در اصل به زبان C نوشته شده اند ، بوجود آمده است . کتابخانه GLAUX را می توانید در قسمت مثالهای OpenGL مربوط به VC++ ، بر روی سی دی ویژوال استودیو مایکروسافت ، بیابید . آدرس اینترنتی کتابخانه GLUT در قسمت مراجع کتاب آورده شده است .

ایجاد کتابخانه GLO :

در محیط دلفی از منوی فایل ، گزینه New را انتخاب کنید . سپس نوع پروژه را DLL انتخاب نمائید . کل محتویات ظاهر شده را پاک نموده ، برنامه را به نام GLO ذخیره کنید . سپس کد ارائه شده زیر را درون آن تایپ کنید :

```
library GLO;
// ************************************************************************
//
// Unit Name : glo
// Purpose   : OpenGL Object rendering library. Draws a number of common
//           shapes such as spheres, cylinder's, cones, disks,
//           torus's, the five platonic solids (tetrahedron, cube,
//           octahedron, isosahedron, and the dodecahedron) and
//           the teapot.
//
// Use       : Include glo.pas in your project, put the glo.dll in the same
//           directory as your project, or in your windows dir.
// ************************************************************************

uses
  OpenGL,
  Math;

type
  TVector = Array [0..2] of GLfloat;
  TIntArray = Array [0..2] of GLint;

  TSuperToroid = record
    list : GLuint;
    r0, r1 : Single;
    n1, n2 : Single;
    drawType : GLenum;
  end;

  TSuperEllipsoid = record
    list : GLuint;
    width : Single;
    n1, n2 : Single;
    drawType : GLenum;
  end;

var
  shapes : GLUquadricObj;
  dodec : Array [0..19] of TVector;
  inited : Boolean = False;
  toroid : TSuperToroid;
  ellipsoid : TSuperEllipsoid;

{$R *.RES}

procedure InitQuadObj();
```

```
begin
  if not (Assigned(shapes)) then
    shapes := gluNewQuadric();
end;

procedure gloWireSphere(radius : GLdouble; slices, stacks : GLint); stdcall;
begin
  InitQuadObj();
  gluQuadricDrawStyle(shapes, GLU_LINE);
  gluQuadricNormals(shapes, GLU_SMOOTH);
  gluQuadricTexture(shapes, GL_TRUE);
  gluSphere(shapes, radius, slices, stacks);
end;

procedure gloSolidSphere(radius : GLdouble; slices, stacks : GLint); stdcall;
begin
  InitQuadObj();
  gluQuadricDrawStyle(shapes, GLU_FILL);
  gluQuadricNormals(shapes, GLU_SMOOTH);
  gluQuadricTexture(shapes, GL_TRUE);
  gluSphere(shapes, radius, slices, stacks);
end;

procedure gloWireCone(base, height : GLdouble;
                  slices, stacks : GLint); stdcall;
begin
  InitQuadObj();
  gluQuadricDrawStyle(shapes, GLU_LINE);
  gluQuadricNormals(shapes, GLU_SMOOTH);
  gluQuadricTexture(shapes, GL_TRUE);
  gluCylinder(shapes, base, 0.0, height, slices, stacks);
end;

procedure gloSolidCone(base, height : GLdouble;
                    slices, stacks : GLint); stdcall;
begin
  InitQuadObj();
  gluQuadricDrawStyle(shapes, GLU_FILL);
  gluQuadricNormals(shapes, GLU_SMOOTH);
  gluQuadricTexture(shapes, GL_TRUE);
  gluCylinder(shapes, base, 0.0, height, slices, stacks);
end;

procedure gloWireCylinder(baseRadius, topRadius, height : GLdouble;
                      slices, stacks : GLint); stdcall;
begin
  InitQuadObj();
  gluQuadricDrawStyle(shapes, GLU_LINE);
  gluQuadricNormals(shapes, GLU_SMOOTH);
  gluQuadricTexture(shapes, GL_TRUE);
  gluCylinder(shapes, baseRadius, topRadius, height, slices, stacks);
end;

procedure gloSolidCylinder(baseRadius, topRadius, height : GLdouble;
                       slices, stacks : GLint); stdcall;
begin
  InitQuadObj();
  gluQuadricDrawStyle(shapes, GLU_FILL);
```

```
  gluQuadricNormals(shapes, GLU_SMOOTH);
  gluQuadricTexture(shapes, GL_TRUE);
  gluCylinder(shapes, baseRadius, topRadius, height, slices, stacks);
end;

procedure gloWireDisk(innerRadius, outerRadius : GLdouble;
                      slices, stacks : GLint); stdcall;
begin
  InitQuadObj();
  gluQuadricDrawStyle(shapes, GLU_LINE);
  gluQuadricNormals(shapes, GLU_SMOOTH);
  gluQuadricTexture(shapes, GL_TRUE);
  gluDisk(shapes, innerRadius, outerRadius, slices, stacks);
end;

procedure gloSolidDisk(innerRadius, outerRadius : GLdouble;
                       slices, stacks : GLint); stdcall;
begin
  InitQuadObj();
  gluQuadricDrawStyle(shapes, GLU_FILL);
  gluQuadricNormals(shapes, GLU_SMOOTH);
  gluQuadricTexture(shapes, GL_TRUE);
  gluDisk(shapes, innerRadius, outerRadius, slices, stacks);
end;

procedure drawCube(size : GLfloat; cType : GLenum);
const
  n : Array[0..5, 0..2] of GLfloat = ((-1.0, 0.0, 0.0),
                                      ( 0.0, 1.0, 0.0),
                                      ( 1.0, 0.0, 0.0),
                                      ( 0.0,-1.0, 0.0),
                                      ( 0.0, 0.0, 1.0),
                                      ( 0.0, 0.0,-1.0));
  faces : Array[0..5, 0..3] of GLint = ((0, 1, 2, 3),
                                        (3, 2, 6, 7),
                                        (7, 6, 5, 4),
                                        (4, 5, 1, 0),
                                        (5, 6, 2, 1),
                                        (7, 4, 0, 3));
var
  v : Array[0..7, 0..2] of GLfloat;
  width : Single;
  I : Integer;
begin
  width := size/2;

  v[0, 0] := -width; v[4, 0] := width;
  v[1, 0] := -width; v[5, 0] := width;
  v[2, 0] := -width; v[6, 0] := width;
  v[3, 0] := -width; v[7, 0] := width;

  v[0, 1] := -width; v[2, 1] := width;
  v[1, 1] := -width; v[3, 1] := width;
  v[4, 1] := -width; v[6, 1] := width;
  v[5, 1] := -width; v[7, 1] := width;

  v[0, 2] := -width; v[1, 2] := width;
  v[3, 2] := -width; v[2, 2] := width;
```

```
  v[4, 2] := -width; v[5, 2] := width;
  v[7, 2] := -width; v[6, 2] := width;

  for I := 5 downto 0 do begin
    glBegin(cType);
      glNormal3fv(@n[I, 0]);
      glTexCoord2f(0.0, 0.0); glVertex3fv(@v[faces[I, 0], 0]);
      glTexCoord2f(1.0, 0.0); glVertex3fv(@v[faces[I, 1], 0]);
      glTexCoord2f(1.0, 1.0); glVertex3fv(@v[faces[I, 2], 0]);
      glTexCoord2f(0.0, 1.0); glVertex3fv(@v[faces[I, 3], 0]);
    glEnd();
  end;
end;

procedure gloWireCube(size : GLdouble); stdcall;
begin
  drawCube(size, GL_LINE_LOOP);
end;

procedure gloSolidCube(size : GLdouble); stdcall;
begin
  drawCube(size, GL_QUADS);
end;

procedure doughnut(innerRadius, outerRadius : GLfloat;
                           nsides, rings : GLint);
var
  i, j : Integer;
  theta, phi, theta1 : GLfloat;
  cosTheta, sinTheta : GLfloat;
  cosTheta1, sinTheta1 : GLfloat;
  ringDelta, sideDelta : GLfloat;

  cosPhi, sinPhi, dist : GLfloat;
begin
  sideDelta := 2.0 * Pi / nsides;
  ringDelta := 2.0 * Pi / rings;

  theta := 0.0;
  cosTheta := 1.0;
  sinTheta := 0.0;

  for i := rings - 1 downto 0 do begin
    theta1 := theta + ringDelta;
    cosTheta1 := cos(theta1);
    sinTheta1 := sin(theta1);
    glBegin(GL_QUAD_STRIP);
      phi := 0.0;
      for j := nsides downto 0 do begin
        phi := phi + sideDelta;
        cosPhi := cos(phi);
        sinPhi := sin(phi);
        dist := outerRadius + (innerRadius * cosPhi);

        glNormal3f(cosTheta1 * cosPhi, -sinTheta1 * cosPhi, sinPhi);
        glVertex3f(cosTheta1 * dist, -sinTheta1 * dist, innerRadius * sinPhi);

        glNormal3f(cosTheta * cosPhi, -sinTheta * cosPhi, sinPhi);
```

```
      glVertex3f(cosTheta * dist, -sinTheta * dist, innerRadius * sinPhi);
    end;
   glEnd();
   theta := theta1;
   cosTheta := cosTheta1;
   sinTheta := sinTheta1;
  end;
end;

procedure gloWireTorus(innerRadius, outerRadius : GLfloat;
                       nsides, rings : GLint); stdcall;
begin
  glPushAttrib(GL_POLYGON_BIT);
   glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
   doughnut(innerRadius, outerRadius, nsides, rings);
  glPopAttrib();
end;

procedure gloSolidTorus(innerRadius, outerRadius : GLfloat;
                        nsides, rings : GLint); stdcall;
begin
  glPushAttrib(GL_POLYGON_BIT);
   glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
   doughnut(innerRadius, outerRadius, nsides, rings);
  glPopAttrib();
end;

procedure drawTeapot(grid : GLint; scale : GLdouble; tType : GLenum);
const
  patchdata : Array[0..9, 0..15] of Integer =
   // Rim
   ((102, 103, 104, 105,4,5,6,7,8,9,10,11,12,13,14,15 ),
   // Body
   ( 12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27 ),
   ( 24, 25, 26, 27, 29, 30, 31, 32,
        33, 34, 35, 36, 37, 38, 39, 40),
   // Lid
   ( 96, 96, 96, 96, 97, 98, 99, 100, 101,
             101, 101, 101,  0,  1,  2,  3 ),
   (  0,  1,  2,  3, 106, 107, 108, 109, 110,
             111, 112, 113, 114, 115, 116, 117 ),
   // Bottom
   ( 118, 118, 118, 118, 124, 122, 119, 121, 123, 126,
                  125, 120, 40, 39, 38, 37 ),
   // Handle
   ( 41, 42, 43, 44, 45, 46, 47, 48, 49,
             50, 51, 52, 53, 54, 55, 56 ),
   ( 53, 54, 55, 56, 57, 58, 59, 60, 61,
             62, 63, 64, 28, 65, 66, 67 ),
   // Spout
   ( 68, 69, 70, 71, 72, 73, 74, 75, 76,
             77, 78, 79, 80, 81, 82, 83 ),
   ( 80, 81, 82, 83, 84, 85, 86, 87,
        88, 89, 90, 91, 92, 93, 94, 95 ));

  cpdata : Array[0..126, 0..2] of Single =
  (( 0.2000,  0.0000, 2.70000 ), ( 0.2000, -0.1120, 2.70000 ),
   ( 0.1120, -0.2000, 2.70000 ),
```

```
( 0.0000, -0.2000, 2.70000 ), ( 1.3375, 0.0000, 2.53125 ),
( 1.3375, -0.7490, 2.53125 ),
( 0.7490, -1.3375, 2.53125 ), ( 0.0000, -1.3375, 2.53125 ),
( 1.4375, 0.0000, 2.53125 ),
( 1.4375, -0.8050, 2.53125 ), ( 0.8050, -1.4375, 2.53125 ),
( 0.0000, -1.4375, 2.53125 ),
( 1.5000, 0.0000, 2.40000 ), ( 1.5000, -0.8400, 2.40000 ),
( 0.8400, -1.5000, 2.40000 ),
( 0.0000, -1.5000, 2.40000 ), ( 1.7500, 0.0000, 1.87500 ),
( 1.7500, -0.9800, 1.87500 ),
( 0.9800, -1.7500, 1.87500 ), ( 0.0000, -1.7500, 1.87500 ),
( 2.0000, 0.0000, 1.35000 ),
( 2.0000, -1.1200, 1.35000 ), ( 1.1200, -2.0000, 1.35000 ),
( 0.0000, -2.0000, 1.35000 ),
( 2.0000, 0.0000, 0.90000 ), ( 2.0000, -1.1200, 0.90000 ),
( 1.1200, -2.0000, 0.90000 ),
( 0.0000, -2.0000, 0.90000 ), ( -2.0000, 0.0000, 0.90000 ),
( 2.0000, 0.0000, 0.45000 ),
( 2.0000, -1.1200, 0.45000 ), ( 1.1200, -2.0000, 0.45000 ),
( 0.0000, -2.0000, 0.45000 ),
( 1.5000, 0.0000, 0.22500 ), ( 1.5000, -0.8400, 0.22500 ),
( 0.8400, -1.5000, 0.22500 ),
( 0.0000, -1.5000, 0.22500 ), ( 1.5000, 0.0000, 0.15000 ),
( 1.5000, -0.8400, 0.15000 ),
( 0.8400, -1.5000, 0.15000 ), ( 0.0000, -1.5000, 0.15000 ),
( -1.6000, 0.0000, 2.02500 ),
( -1.6000, -0.3000, 2.02500 ), ( -1.5000, -0.3000, 2.25000 ),
( -1.5000, 0.0000, 2.25000 ),
( -2.3000, 0.0000, 2.02500 ), ( -2.3000, -0.3000, 2.02500 ),
( -2.5000, -0.3000, 2.25000 ),
( -2.5000, 0.0000, 2.25000 ), ( -2.7000, 0.0000, 2.02500 ),
( -2.7000, -0.3000, 2.02500 ),
( -3.0000, -0.3000, 2.25000 ), ( -3.0000, 0.0000, 2.25000 ),
( -2.7000, 0.0000, 1.80000 ),
( -2.7000, -0.3000, 1.80000 ), ( -3.0000, -0.3000, 1.80000 ),
( -3.0000, 0.0000, 1.80000 ),
( -2.7000, 0.0000, 1.57500 ), ( -2.7000, -0.3000, 1.57500 ),
( -3.0000, -0.3000, 1.35000 ),
( -3.0000, 0.0000, 1.35000 ), ( -2.5000, 0.0000, 1.12500 ),
( -2.5000, -0.3000, 1.12500 ),
( -2.6500, -0.3000, 0.93750 ), ( -2.6500, 0.0000, 0.93750 ),
( -2.0000, -0.3000, 0.90000 ),
( -1.9000, -0.3000, 0.60000 ), ( -1.9000, 0.0000, 0.60000 ),
( 1.7000, 0.0000, 1.42500 ),
( 1.7000, -0.6600, 1.42500 ), ( 1.7000, -0.6600, 0.60000 ),
( 1.7000, 0.0000, 0.60000 ),
( 2.6000, 0.0000, 1.42500 ), ( 2.6000, -0.6600, 1.42500 ),
( 3.1000, -0.6600, 0.82500 ),
( 3.1000, 0.0000, 0.82500 ), ( 2.3000, 0.0000, 2.10000 ),
( 2.3000, -0.2500, 2.10000 ),
( 2.4000, -0.2500, 2.02500 ), ( 2.4000, 0.0000, 2.02500 ),
( 2.7000, 0.0000, 2.40000 ),
( 2.7000, -0.2500, 2.40000 ), ( 3.3000, -0.2500, 2.40000 ),
( 3.3000, 0.0000, 2.40000 ),
( 2.8000, 0.0000, 2.47500 ), ( 2.8000, -0.2500, 2.47500 ),
( 3.5250, -0.2500, 2.49375 ),
( 3.5250, 0.0000, 2.49375 ), ( 2.9000, 0.0000, 2.47500 ),
( 2.9000, -0.1500, 2.47500 ),
```

```
    (  3.4500, -0.1500, 2.51250 ), (  3.4500,  0.0000, 2.51250 ),
     (  2.8000,  0.0000, 2.40000 ),
    (  2.8000, -0.1500, 2.40000 ), (  3.2000, -0.1500, 2.40000 ),
     (  3.2000,  0.0000, 2.40000 ),
    (  0.0000,  0.0000, 3.15000 ), (  0.8000,  0.0000, 3.15000 ),
     (  0.8000, -0.4500, 3.15000 ),
    (  0.4500, -0.8000, 3.15000 ), (  0.0000, -0.8000, 3.15000 ),
     (  0.0000,  0.0000, 2.85000 ),
    (  1.4000,  0.0000, 2.40000 ), (  1.4000, -0.7840, 2.40000 ),
     (  0.7840, -1.4000, 2.40000 ),
    (  0.0000, -1.4000, 2.40000 ), (  0.4000,  0.0000, 2.55000 ),
     (  0.4000, -0.2240, 2.55000 ),
    (  0.2240, -0.4000, 2.55000 ), (  0.0000, -0.4000, 2.55000 ),
     (  1.3000,  0.0000, 2.55000 ),
    (  1.3000, -0.7280, 2.55000 ), (  0.7280, -1.3000, 2.55000 ),
     (  0.0000, -1.3000, 2.55000 ),
    (  1.3000,  0.0000, 2.40000 ), (  1.3000, -0.7280, 2.40000 ),
     (  0.7280, -1.3000, 2.40000 ),
    (  0.0000, -1.3000, 2.40000 ), (  0.0000,  0.0000, 0.00000 ),
     (  1.4250, -0.7980, 0.00000 ),
    (  1.5000,  0.0000, 0.07500 ), (  1.4250,  0.0000, 0.00000 ),
     (  0.7980, -1.4250, 0.00000 ),
    (  0.0000, -1.5000, 0.07500 ), (  0.0000, -1.4250, 0.00000 ),
     (  1.5000, -0.8400, 0.07500 ),
    (  0.8400, -1.5000, 0.07500 ));

  tex : Array [0..1, 0..1, 0..1] of Single = (((0, 0),
                                               (1, 0)),
                                              ((0, 1),
                                               (1, 1)));

var
  p : Array [0..3, 0..3, 0..2] of Single;
  q : Array [0..3, 0..3, 0..2] of Single;
  r : Array [0..3, 0..3, 0..2] of Single;
  s : Array [0..3, 0..3, 0..2] of Single;

  i, j, k, l : Cardinal;

begin
  glPushAttrib(GL_ENABLE_BIT or GL_EVAL_BIT or GL_POLYGON_BIT);
    glEnable(GL_CULL_FACE);
    glCullFace(GL_FRONT);
    glEnable(GL_AUTO_NORMAL);
    glEnable(GL_NORMALIZE);
    glEnable(GL_MAP2_VERTEX_3);
    glEnable(GL_MAP2_TEXTURE_COORD_2);
    glPushMatrix();
      glRotatef(270.0, 1.0, 0.0, 0.0);
      glScalef(0.5 * scale, 0.5 * scale, 0.5 * scale);
      glTranslatef(0.0, 0.0, -1.5);
      for i := 0 to 9 do begin
        for j := 0 to 3 do begin
          for k := 0 to 3 do begin
            for l := 0 to 2 do begin
              p[j][k][l] := cpdata[patchdata[i][j * 4 + k]][l];
              q[j][k][l] := cpdata[patchdata[i][j * 4 + (3 - k)]][l];
              if (l = 1) then
```

```
            q[j][k][l] := q[j][k][l] * -1.0;
          if (i < 6) then begin
            r[j][k][l] := cpdata[patchdata[i][j * 4 + (3 - k)]][l];
            if (l = 0) then
              r[j][k][l] := r[j][k][l] * -1.0;
            s[j][k][l] := cpdata[patchdata[i][j * 4 + k]][l];
            if (l = 0) then
              s[j][k][l] := s[j][k][l] * -1.0;
            if (l = 1) then
              s[j][k][l] := s[j][k][l] * -1.0;
        end;
       end;
      end;
    end;
   glMap2f(GL_MAP2_TEXTURE_COORD_2, 0, 1, 2, 2, 0, 1, 4, 2, @tex[0][0][0]);
   glMap2f(GL_MAP2_VERTEX_3, 0, 1, 3, 4, 0, 1, 12, 4, @p[0][0][0]);
   glMapGrid2f(grid, 0.0, 1.0, grid, 0.0, 1.0);
   glEvalMesh2(tType, 0, grid, 0, grid);
   glMap2f(GL_MAP2_VERTEX_3, 0, 1, 3, 4, 0, 1, 12, 4, @q[0][0][0]);
   glEvalMesh2(tType, 0, grid, 0, grid);
   if (i < 6) then begin
     glMap2f(GL_MAP2_VERTEX_3, 0, 1, 3, 4, 0, 1, 12, 4, @r[0][0][0]);
     glEvalMesh2(tType, 0, grid, 0, grid);
     glMap2f(GL_MAP2_VERTEX_3, 0, 1, 3, 4, 0, 1, 12, 4, @s[0][0][0]);
     glEvalMesh2(tType, 0, grid, 0, grid);
   end;
  end;
  glPopMatrix();
  glPopAttrib();
end;

procedure gloWireTeapot(scale : GLdouble); stdcall;
begin
  drawTeapot(10, scale, GL_LINE);
end;

procedure gloSolidTeapot(scale : GLdouble); stdcall;
begin
  drawTeapot(14, scale, GL_FILL);
end;

procedure initDodecahedron();
var
  alpha, beta : GLfloat;
begin
  alpha := sqrt(2.0 / (3.0 + sqrt(5.0)));
  beta := 1.0 + sqrt(6.0 / (3.0 + sqrt(5.0)) - 2.0 + 2.0 * sqrt(2.0 / (3.0 + sqrt(5.0))));
  dodec[0][0] := -alpha;  dodec[0][1] := 0;      dodec[0][2] := beta;
  dodec[1][0] := alpha;   dodec[1][1] := 0;      dodec[1][2] := beta;
  dodec[2][0] := -1;      dodec[2][1] := -1;     dodec[2][2] := -1;
  dodec[3][0] := -1;      dodec[3][1] := -1;     dodec[3][2] := 1;
  dodec[4][0] := -1;      dodec[4][1] := 1;      dodec[4][2] := -1;
  dodec[5][0] := -1;      dodec[5][1] := 1;      dodec[5][2] := 1;
  dodec[6][0] := 1;       dodec[6][1] := -1;     dodec[6][2] := -1;
  dodec[7][0] := 1;       dodec[7][1] := -1;     dodec[7][2] := 1;
  dodec[8][0] := 1;       dodec[8][1] := 1;      dodec[8][2] := -1;
  dodec[9][0] := 1;       dodec[9][1] := 1;      dodec[9][2] := 1;
  dodec[10][0] := beta;   dodec[10][1] := alpha;  dodec[10][2] := 0;
```

```
  dodec[11][0] := beta;   dodec[11][1] := -alpha; dodec[11][2] := 0;
  dodec[12][0] := -beta;  dodec[12][1] := alpha;  dodec[12][2] := 0;
  dodec[13][0] := -beta;  dodec[13][1] := -alpha; dodec[13][2] := 0;
  dodec[14][0] := -alpha; dodec[14][1] := 0;      dodec[14][2] := -beta;
  dodec[15][0] := alpha;  dodec[15][1] := 0;      dodec[15][2] := -beta;
  dodec[16][0] := 0;      dodec[16][1] := beta;   dodec[16][2] := alpha;
  dodec[17][0] := 0;      dodec[17][1] := beta;   dodec[17][2] := -alpha;
  dodec[18][0] := 0;      dodec[18][1] := -beta;  dodec[18][2] := alpha;
  dodec[19][0] := 0;      dodec[19][1] := -beta;  dodec[19][2] := -alpha;
end;

procedure Diff3(a, b : TVector; var c : TVector);
begin
  c[0] := a[0] - b[0];
  c[1] := a[1] - b[1];
  c[2] := a[2] - b[2];
end;

procedure CrossProd(v1, v2 : TVector; var prod : TVector);
begin
  prod[0] := (v1[1] * v2[2])-(v2[1] * v1[2]);
  prod[1] := (v1[2] * v2[0])-(v2[2] * v1[0]);
  prod[2] := (v1[0] * v2[1])-(v2[0] * v1[1]);
end;

procedure Normalize (var v : TVector);
var
  len : Single;
begin
  len := sqrt(v[0] * v[0] + v[1] * v[1] + v[2] * v[2]);
  if len = 0.0 then
    Exit;

  len := 1.0/len;

  v[0] := v[0]*len;
  v[1] := v[1]*len;
  v[2] := v[2]*len;
end;

procedure pentagon(a, b, c, d, e : Integer; shadeType : GLenum);
var
  n0, d1, d2 : TVector;
begin
  Diff3(dodec[a], dodec[b], d1);
  Diff3(dodec[b], dodec[c], d2);
  CrossProd(d1, d2, n0);
  Normalize(n0);

  glBegin(shadeType);
    glNormal3fv(@n0);
    glVertex3fv(@dodec[a][0]);
    glVertex3fv(@dodec[b][0]);
    glVertex3fv(@dodec[c][0]);
    glVertex3fv(@dodec[d][0]);
    glVertex3fv(@dodec[e][0]);
  glEnd();
end;
```

```
procedure dodecahedron(tType : GLenum);
begin
  if not (inited) then begin
    inited := True;
    initDodecahedron();
  end;

  pentagon(0, 1, 9, 16, 5, tType);
  pentagon(1, 0, 3, 18, 7, tType);
  pentagon(1, 7, 11, 10, 9, tType);
  pentagon(11, 7, 18, 19, 6, tType);
  pentagon(8, 17, 16, 9, 10, tType);
  pentagon(2, 14, 15, 6, 19, tType);
  pentagon(2, 13, 12, 4, 14, tType);
  pentagon(2, 19, 18, 3, 13, tType);
  pentagon(3, 0, 5, 12, 13, tType);
  pentagon(6, 15, 8, 10, 11, tType);
  pentagon(4, 17, 8, 15, 14, tType);
  pentagon(4, 12, 5, 16, 17, tType);
end;

procedure gloWireDodecahedron(); stdcall;
begin
  dodecahedron(GL_LINE_LOOP);
end;

procedure gloSolidDodecahedron(); stdcall;
begin
  dodecahedron(GL_TRIANGLE_FAN);
end;

procedure recordItem(n1, n2, n3 : TVector; shadeType : GLenum);
var
  q0, q1 : TVector;
begin
  Diff3(n1, n2, q0);
  Diff3(n2, n3, q1);
  CrossProd(q0, q1, q1);
  Normalize(q1);

  glBegin(shadeType);
    glNormal3fv(@q1);
    glVertex3fv(@n1);
    glVertex3fv(@n2);
    glVertex3fv(@n3);
  glEnd();
end;

procedure subdivide (v0, v1, v2 : TVector; shadeType : GLenum);
var
  depth : Integer;
  w0, w1, w2 : TVector;
  l : GLfloat;
  i, j, k, n : Integer;
begin
  depth := 1;
  for i := 0 to depth-1 do begin
```

```
    j := 0;
    while (i + j < depth) do begin
      k := depth - i - j;
      for n := 0 to 2 do begin
        w0[n] := (i * v0[n] + j * v1[n] + k * v2[n]) / depth;
        w1[n] := ((i + 1) * v0[n] + j * v1[n] + (k - 1) * v2[n]) / depth;
        w2[n] := (i * v0[n] + (j + 1) * v1[n] + (k - 1) * v2[n]) / depth;
      end;
      l := sqrt(w0[0] * w0[0] + w0[1] * w0[1] + w0[2] * w0[2]);
      w0[0] := w0[0] / l;
      w0[1] := w0[1] / l;
      w0[2] := w0[2] / l;
      l := sqrt(w1[0] * w1[0] + w1[1] * w1[1] + w1[2] * w1[2]);
      w1[0] := w1[0] / l;
      w1[1] := w1[1] / l;
      w1[2] := w1[2] / l;
      l := sqrt(w2[0] * w2[0] + w2[1] * w2[1] + w2[2] * w2[2]);
      w2[0] := w2[0] / l;
      w2[1] := w2[1] / l;
      w2[2] := w2[2] / l;
      recorditem(w1, w0, w2, shadeType);
      Inc(j);
    end;
  end;
end;

procedure drawTriangle(i : Integer; data : Array of TVector;
                ndx : Array of TIntArray; shadeType : Glenum);
var
  x0, x1, x2 : TVector;
begin
  x0 := data[ndx[i][0]];
  x1 := data[ndx[i][1]];
  x2 := data[ndx[i][2]];
  subdivide(x0, x1, x2, shadetype);
end;

procedure octahedron(shadeType : GLenum);
const
  odata : Array [0..5] of TVector = (( 1.0, 0.0, 0.0),
                                     (-1.0, 0.0, 0.0),
                                     ( 0.0, 1.0, 0.0),
                                     ( 0.0,-1.0, 0.0),
                                     ( 0.0, 0.0, 1.0),
                                     ( 0.0, 0.0,-1.0));
  ondex : Array [0..7] of TIntArray = ((0, 4, 2),
                                       (1, 2, 4),
                                       (0, 3, 4),
                                       (1, 4, 3),
                                       (0, 2, 5),
                                       (1, 5, 2),
                                       (0, 5, 3),
                                       (1, 3, 5));
var
  I : Integer;
begin
  for I := 7 downto 0 do begin
    drawTriangle(I, odata, ondex, shadeType);
```

```
  end;
end;

procedure gloWireOctahedron(); stdcall;
begin
  octahedron(GL_LINE_LOOP);
end;

procedure gloSolidOctahedron(); stdcall;
begin
  octahedron(GL_TRIANGLES);
end;

procedure icosahedron(shadeType : GLenum);
const
  X = 0.525731112119133606;
  Z = 0.850650808352039932;

  idata : Array [0..11] of TVector = ((-X, 0, Z),
                                      ( X, 0, Z),
                                      (-X, 0,-Z),
                                      ( X, 0,-Z),
                                      ( 0, Z, X),
                                      ( 0, Z,-X),
                                      ( 0,-Z, X),
                                      ( 0,-Z,-X),
                                      ( Z, X, 0),
                                      (-Z, X, 0),
                                      ( Z,-X, 0),
                                      (-Z,-X, 0));
  index : Array [0..19] of TIntArray = ((0, 4, 1),
                                        (0, 9, 4),
                                        (9, 5, 4),
                                        (4, 5, 8),
                                        (4, 8, 1),
                                        (8, 10,1),
                                        (8, 3,10),
                                        (5, 3, 8),
                                        (5, 2, 3),
                                        (2, 7, 3),
                                        (7, 10,3),
                                        (7, 6,10),
                                        (7, 11,6),
                                        (11, 0,6),
                                        (0, 1, 6),
                                        (6, 1,10),
                                        (9, 0,11),
                                        (9, 11,2),
                                        (9, 2, 5),
                                        (7, 2,11));

var
  I : Integer;
begin
  for I := 19 downto 0 do begin
    drawTriangle(I, idata, index, shadeType);
  end;
end;
```

```
procedure gloWireIsosahedron(); stdcall;
begin
  icosahedron(GL_LINE_LOOP);
end;

procedure gloSolidIsosahedron(); stdcall;
begin
  icosahedron(GL_TRIANGLES);
end;

procedure tetrahedron(shadeType : GLenum);
const
  T = 1.73205080756887729;
  tdata : Array[0..3] of TVector = (( T, T, T),
                                    ( T,-T,-T),
                                    (-T, T,-T),
                                    (-T,-T, T));
  tndex : Array[0..3] of TIntArray = ((0, 1, 3),
                                      (2, 1, 0),
                                      (3, 2, 0),
                                      (1, 2, 3));
var
  I : Integer;
begin
  for I := 3 downto 0 do begin
    drawTriangle(I, tdata, tndex, shadeType);
  end;
end;

procedure gloWireTetrahedron(); stdcall;
begin
  tetrahedron(GL_LINE_LOOP);
end;

procedure gloSolidTetrahedron(); stdcall;
begin
  tetrahedron(GL_TRIANGLES);
end;

//   inner_radius - radius of hole at center
//   outer_radius - radius at center of teeth
//   width        - width or over all length of gear
//   teeth        - number of teeth
//   tooth_depth  - depth or over all length of tooth
procedure gear(inner_radius, outer_radius, width : double;
                  teeth : integer; tooth_depth : double);
var
    i : Integer;
    r0, r1, r2 : Double;
    angle, da : Double;
    u, v, len : Double;
begin
  r0 := inner_radius;
  r1 := outer_radius - tooth_depth/2.0;
  r2 := outer_radius + tooth_depth/2.0;
  da := 2.0*Pi/teeth/4.0;
```

```
glShadeModel(GL_SMOOTH);

glNormal3f(0.0, 0.0, 1.0);

// front face
glBegin(GL_QUAD_STRIP);
  for i := 0 to teeth do begin
    angle := i * 2.0*Pi/ teeth;
    glVertex3f(r0*cos(angle), r0*sin(angle), width*0.5);
    glVertex3f(r1*cos(angle), r1*sin(angle), width*0.5);
    glVertex3f(r0*cos(angle), r0*sin(angle), width*0.5);
    glVertex3f(r1*cos(angle+3*da), r1*sin(angle+3*da), width*0.5);
  end;
glEnd();

// front sides of teeth
glBegin(GL_QUADS);
  da := 2.0*Pi/teeth/4.0;
  for i := 0 to teeth - 1 do begin
    angle := i * 2.0*Pi / teeth;
    glVertex3f(r1*cos(angle), r1*sin(angle), width*0.5);
    glVertex3f(r2*cos(angle+da), r2*sin(angle+da), width*0.5);
    glVertex3f(r2*cos(angle+2*da), r2*sin(angle+2*da), width*0.5);
    glVertex3f(r1*cos(angle+3*da), r1*sin(angle+3*da), width*0.5);
  end;
glEnd();

glNormal3f(0.0, 0.0, -1.0);

// back face
glBegin(GL_QUAD_STRIP);
  for i := 0 to teeth do begin
    angle := i * 2.0*Pi / teeth;
    glVertex3f(r1*cos(angle), r1*sin(angle), -width*0.5);
    glVertex3f(r0*cos(angle), r0*sin(angle), -width*0.5);
    glVertex3f(r1*cos(angle+3*da), r1*sin(angle+3*da), -width*0.5);
    glVertex3f(r0*cos(angle), r0*sin(angle), -width*0.5);
  end;
glEnd();

// back sides of teeth
glBegin(GL_QUADS);
  da := 2.0*Pi / teeth / 4.0;
  for i := 0 to teeth - 1 do begin
    angle := i * 2.0*Pi / teeth;

    glVertex3f(r1*cos(angle+3*da), r1*sin(angle+3*da), -width*0.5);
    glVertex3f(r2*cos(angle+2*da), r2*sin(angle+2*da), -width*0.5);
    glVertex3f(r2*cos(angle+da), r2*sin(angle+da), -width*0.5);
    glVertex3f(r1*cos(angle), r1*sin(angle), -width*0.5);
  end;
glEnd();

// outward faces of teeth
glBegin(GL_QUAD_STRIP);
  for i := 0 to teeth - 1 do begin
    angle := i * 2.0*Pi / teeth;
```

```
      glVertex3f(r1*cos(angle), r1*sin(angle), width*0.5);
      glVertex3f(r1*cos(angle), r1*sin(angle), -width*0.5);
      u   := r2*cos(angle+da) - r1*cos(angle);
      v   := r2*sin(angle+da) - r1*sin(angle);
      len := sqrt(u*u + v*v);
      u   := u / len;
      v   := v / len;

      glNormal3f(v, -u, 0.0 );
      glVertex3f(r2*cos(angle+da), r2*sin(angle+da), width*0.5);
      glVertex3f(r2*cos(angle+da), r2*sin(angle+da), -width*0.5);
      glNormal3f(cos(angle), sin(angle), 0.0);
      glVertex3f(r2*cos(angle+2*da), r2*sin(angle+2*da), width*0.5);
      glVertex3f(r2*cos(angle+2*da), r2*sin(angle+2*da), -width*0.5);
      u := r1*cos(angle+3*da) - r2*cos(angle+2*da);
      v := r1*sin(angle+3*da) - r2*sin(angle+2*da);

      glNormal3f(v, -u, 0.0);
      glVertex3f(r1*cos(angle+3*da), r1*sin(angle+3*da), width*0.5);
      glVertex3f(r1*cos(angle+3*da), r1*sin(angle+3*da), -width*0.5);
      glNormal3f(cos(angle), sin(angle), 0.0);
    end;

    glVertex3f(r1*cos(0), r1*sin(0), width*0.5);
    glVertex3f(r1*cos(0), r1*sin(0), -width*0.5);
  glEnd();

  // inside radius cylinder
  glBegin(GL_QUAD_STRIP);
    for i := 0 to teeth do begin
      angle := i * 2.0*Pi / teeth;
      glNormal3f(-cos(angle), -sin(angle), 0.0);
      glVertex3f(r0*cos(angle), r0*sin(angle), -width*0.5);
      glVertex3f(r0*cos(angle), r0*sin(angle), width*0.5);
    end;
  glEnd();
end;

procedure gloWireGear(inner_radius, outer_radius, width: double;
                      teeth: integer; tooth_depth: double); stdcall;
begin
  glPushAttrib(GL_POLYGON_BIT);
    glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
    gear(inner_radius, outer_radius, width, teeth, tooth_depth);
  glPopAttrib();
end;

procedure gloSolidGear(inner_radius, outer_radius, width: double;
                       teeth: integer; tooth_depth: double); stdcall;
begin
  gear(inner_radius, outer_radius, width, teeth, tooth_depth);
end;

function pow(f, p : Double) : Double;
var
  sign : Integer;
  absf : Double;
begin
```

```
  if f < 0 then
    sign := -1
  else
    sign := 1;

  if f < 0 then
    absf := -f
  else
    absf := f;

  if (absf < 0.00001) then
    Result := 0.0
  else
    Result := sign * Power(absf, p);
end;

procedure drawSuperToroid(r0, r1, n1, n2 : Single;
                          slices, stacks : Integer; drawType : GLenum);
const
  ST = 0.01745329252;
var
  u, v, du, dv : Double;
  theta, phi : Single;
  quads : Array [0..3, 0..2] of GLfloat;
  normals : Array [0..3, 0..2] of GLfloat;
begin
  du := 360/slices;
  dv := 360/stacks;
  u := 0;
  v := 0;

  if n1 < 0 then n1 := 0.0;
  if n2 < 0 then n2 := 0.0;
  if (toroid.r0 <> r0) or (toroid.r1 <> r1) or
     (toroid.n1 <> n1) or (toroid.n2 <> n2) or
     (toroid.drawType <> drawType) then begin
    if toroid.list <> 0 then
      glDeleteLists(toroid.list, 1);

    toroid.list := glGenLists(1);
    glNewList(toroid.list, GL_COMPILE);

    toroid.r0 := r0;
    toroid.r1 := r1;
    toroid.n1 := n1;
    toroid.n2 := n2;
    toroid.drawType := drawType;

    glBegin(drawType);
      while u < 360 do begin
        while v < 360 do begin
          theta := u * ST;
          phi := v * ST;

          quads[0, 0] := (r0 + r1 * pow(cos(phi), n1))*pow(cos(theta), n2);
          quads[0, 1] := (r0 + r1 * pow(cos(phi), n1))*pow(sin(theta), n2);
          quads[0, 2] := r1 * pow(sin(phi), n1);
```

```
      normals[0, 0] := (pow(cos(phi), 2-(n1))) * pow(cos(theta), 2-(n2));
      normals[0, 1] := (pow(cos(phi), 2-(n1))) * pow(sin(theta), 2-(n2));
      normals[0, 2] := pow(sin(phi), 2-(n1));

      theta := (u+du) * ST;
      phi := v * ST;

      quads[1, 0] := (r0 + r1 * pow(cos(phi), n1))*pow(cos(theta), n2);
      quads[1, 1] := (r0 + r1 * pow(cos(phi), n1))*pow(sin(theta), n2);
      quads[1, 2] := r1 * pow(sin(phi), n1);

      normals[1, 0] := (pow(cos(phi), 2-(n1))) * pow(cos(theta), 2-(n2));
      normals[1, 1] := (pow(cos(phi), 2-(n1))) * pow(sin(theta), 2-(n2));
      normals[1, 2] := pow(sin(phi), 2-(n1));

      theta := (u+du) * ST;
      phi := (v+dv) * ST;

      quads[2, 0] := (r0 + r1 * pow(cos(phi), n1))*pow(cos(theta), n2);
      quads[2, 1] := (r0 + r1 * pow(cos(phi), n1))*pow(sin(theta), n2);
      quads[2, 2] := r1 * pow(sin(phi), n1);

      normals[2, 0] := (pow(cos(phi), 2-(n1))) * pow(cos(theta), 2-(n2));
      normals[2, 1] := (pow(cos(phi), 2-(n1))) * pow(sin(theta), 2-(n2));
      normals[2, 2] := pow(sin(phi), 2-(n1));

      theta := u * ST;
      phi := (v+dv) * ST;

      quads[3, 0] := (r0 + r1 * pow(cos(phi), n1))*pow(cos(theta), n2);
      quads[3, 1] := (r0 + r1 * pow(cos(phi), n1))*pow(sin(theta), n2);
      quads[3, 2] := r1 * pow(sin(phi), n1);

      normals[3, 0] := (pow(cos(phi), 2-(n1))) * pow(cos(theta), 2-(n2));
      normals[3, 1] := (pow(cos(phi), 2-(n1))) * pow(sin(theta), 2-(n2));
      normals[3, 2] := pow(sin(phi), 2-(n1));

      glNormal3fv(@normals[0]);
      glVertex3fv(@quads[0]);

      glNormal3fv(@normals[1]);
      glVertex3fv(@quads[1]);

      glNormal3fv(@normals[2]);
      glVertex3fv(@quads[2]);

      glNormal3fv(@normals[3]);
      glVertex3fv(@quads[3]);
      v := v + dv;
    end;
    v := 0;
    u := u + du;
  end;
 glEnd();
 glEndList();
end;

glCallList(toroid.list);
```

```
end;

procedure gloWireSuperToroid(inner_radius, outer_radius,
            n1, n2 : Single; slices, stacks : Integer); stdcall;
begin
  drawSuperToroid(inner_radius, outer_radius, n1, n2,
                    slices, stacks, GL_LINE_LOOP);
end;

procedure gloSolidSuperToroid(inner_radius, outer_radius,
            n1, n2 : Single; slices, stacks : Integer); stdcall;
begin
  drawSuperToroid(inner_radius, outer_radius, n1, n2,
                      slices, stacks, GL_QUADS);
end;

procedure drawSuperEllipsoid(width, n1, n2 : Single;
                  slices, stacks : Integer; drawType : GLenum);
const
  ST = 0.01745329252;
var
  u, v, du, dv : Single;
  theta, phi : Single;
  quads : Array [0..3, 0..2] of GLfloat;
  normals : Array [0..3, 0..2] of GLfloat;
begin
  du := 360/slices;
  dv := 360/stacks;
  u := 0;
  v := 0;

  if n1 < 0 then n1 := 0.0;
  if n2 < 0 then n2 := 0.0;

  if (ellipsoid.width <> width) or (ellipsoid.n1 <> n1) or
     (ellipsoid.n2 <> n2) or (ellipsoid.drawType <> drawType) then begin
    if ellipsoid.list <> 0 then
      glDeleteLists(ellipsoid.list, 1);

    ellipsoid.list := glGenLists(1);
    glNewList(ellipsoid.list, GL_COMPILE);

    ellipsoid.width := width;
    ellipsoid.n1 := n1;
    ellipsoid.n2 := n2;
    ellipsoid.drawType := drawType;

    while u < 360 do begin
      while v < 360 do begin
        theta := u * ST;
        phi := v * ST;

        glBegin(drawType);
          quads[0, 0] := (width * pow(cos(phi), n1))*pow(cos(theta), n2);
          quads[0, 1] := (width * pow(cos(phi), n1))*pow(sin(theta), n2);
          quads[0, 2] := width * pow(sin(phi), n1);

          normals[0, 0] := (pow(cos(phi), 2-(n1))) * pow(cos(theta), 2-(n2));
```

```
        normals[0, 1] := (pow(cos(phi), 2-(n1))) * pow(sin(theta), 2-(n2));
        normals[0, 2] := pow(sin(phi), 2-(n1));

        theta := (u+du) * ST;
        phi := v * ST;

        quads[1, 0] := (width * pow(cos(phi), n1))*pow(cos(theta), n2);
        quads[1, 1] := (width * pow(cos(phi), n1))*pow(sin(theta), n2);
        quads[1, 2] := width * pow(sin(phi), n1);

        normals[1, 0] := (pow(cos(phi), 2-(n1))) * pow(cos(theta), 2-(n2));
        normals[1, 1] := (pow(cos(phi), 2-(n1))) * pow(sin(theta), 2-(n2));
        normals[1, 2] := pow(sin(phi), 2-(n1));

        theta := (u+du) * ST;
        phi := (v+dv) * ST;

        quads[2, 0] := (width * pow(cos(phi), n1))*pow(cos(theta), n2);
        quads[2, 1] := (width * pow(cos(phi), n1))*pow(sin(theta), n2);
        quads[2, 2] := width * pow(sin(phi), n1);

        normals[2, 0] := (pow(cos(phi), 2-(n1))) * pow(cos(theta), 2-(n2));
        normals[2, 1] := (pow(cos(phi), 2-(n1))) * pow(sin(theta), 2-(n2));
        normals[2, 2] := pow(sin(phi), 2-(n1));

        theta := u * ST;
        phi := (v+dv) * ST;

        quads[3, 0] := (width * pow(cos(phi), n1))*pow(cos(theta), n2);
        quads[3, 1] := (width * pow(cos(phi), n1))*pow(sin(theta), n2);
        quads[3, 2] := width * pow(sin(phi), n1);

        normals[3, 0] := (pow(cos(phi), 2-(n1))) * pow(cos(theta), 2-(n2));
        normals[3, 1] := (pow(cos(phi), 2-(n1))) * pow(sin(theta), 2-(n2));
        normals[3, 2] := pow(sin(phi), 2-(n1));

        glNormal3fv(@normals[0]);
        glVertex3fv(@quads[0]);

        glNormal3fv(@normals[1]);
        glVertex3fv(@quads[1]);

        glNormal3fv(@normals[2]);
        glVertex3fv(@quads[2]);

        glNormal3fv(@normals[3]);
        glVertex3fv(@quads[3]);
      glEnd();
      v := v + dv;
    end;
    v := 0;
    u := u + du;
  end;
  glEndList();
 end;

 glCallList(ellipsoid.list);
end;
```

```
procedure gloWireSuperEllipsoid(width, n1, n2 : Single;
                        slices, stacks : Integer); stdcall;
begin
  drawSuperEllipsoid(width, n1, n2, slices, stacks, GL_LINE_LOOP);
end;

procedure gloSolidSuperEllipsoid(width, n1, n2 : Single;
                        slices, stacks : Integer); stdcall;
begin
  drawSuperEllipsoid(width, n1, n2, slices, stacks, GL_QUADS);
end;

exports
  gloWireSphere           name 'gloWireSphere',
  gloSolidSphere          name 'gloSolidSphere',
  gloWireCone             name 'gloWireCone',
  gloSolidCone            name 'gloSolidCone',
  gloSolidCylinder        name 'gloSolidCylinder',
  gloWireCylinder         name 'gloWireCylinder',
  gloWireDisk             name 'gloWireDisk',
  gloSolidDisk            name 'gloSolidDisk',
  gloWireCube             name 'gloWireCube',
  gloSolidCube            name 'gloSolidCube',
  gloWireTorus            name 'gloWireTorus',
  gloSolidTorus           name 'gloSolidTorus',
  gloWireTeapot           name 'gloWireTeapot',
  gloSolidTeapot          name 'gloSolidTeapot',
  gloWireDodecahedron     name 'gloWireDodecahedron',
  gloSolidDodecahedron    name 'gloSolidDodecahedron',
  gloWireOctahedron       name 'gloWireOctahedron',
  gloSolidOctahedron      name 'gloSolidOctahedron',
  gloWireIsosahedron      name 'gloWireIsosahedron',
  gloSolidIsosahedron     name 'gloSolidIsosahedron',
  gloWireTetrahedron      name 'gloWireTetrahedron',
  gloSolidTetrahedron     name 'gloSolidTetrahedron',
  gloWireGear             name 'gloWireGear',
  gloSolidGear            name 'gloSolidGear',
  gloWireSuperToroid      name 'gloWireSuperToroid',
  gloSolidSuperToroid     name 'gloSolidSuperToroid',
  gloWireSuperEllipsoid   name 'gloWireSuperEllipsoid',
  gloSolidSuperEllipsoid  name 'gloSolidSuperEllipsoid';
end.
```

اکنون برنامه را کامپایل کنید تا فایل DLL ساخته شود . فراموش نکنید که فایل DLL باید در یکی از سه مکان زیر قرار گیرد :

۱- دایرکتوری ویندوز

۲- دایرکتوری سیستم ویندوز

۳- دایرکتوری جاری (مسیری که فایل اجرایی برنامه در آن قرار دارد .)

برنامه فصل :

در برنامه زیر طرز استفاده از کتابخانه فوق (GLO.DLL) توضیح داده شده است :

```
unit Ch11;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics,
  Controls, Forms, Dialogs, OpenGL,SPF;

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

var
  Form1: TForm1;

  f_Hdc : LongInt;

implementation
{$R *.DFM}

procedure gloWireSphere(radius : GLdouble; slices, stacks : GLint);
stdcall;external 'glo.dll';
procedure gloWireCone(base, height : GLdouble; slices, stacks : GLint);
stdcall;external 'glo.dll';
procedure gloWireCylinder(baseRadius, topRadius, height : GLdouble; slices,
 stacks : GLint); stdcall;external 'glo.dll';
procedure gloWireDisk(innerRadius, outerRadius : GLdouble;
 slices, stacks : GLint);stdcall;external 'glo.dll';
procedure gloWireCube(size : GLdouble); stdcall;external 'glo.dll';
procedure gloWireTorus(innerRadius, outerRadius : GLfloat;
 nsides, rings : GLint);stdcall;external 'glo.dll';
procedure gloWireTeapot(scale : GLdouble); stdcall;external 'glo.dll';
procedure gloWireDodecahedron(); stdcall;external 'glo.dll';
procedure gloWireOctahedron(); stdcall;external 'glo.dll';
procedure gloWireIsosahedron(); stdcall;external 'glo.dll';
procedure gloWireTetrahedron(); stdcall;external 'glo.dll';
procedure gloWireGear(inner_radius, outer_radius, width: double;
 teeth: integer; tooth_depth: double);stdcall;external 'glo.dll';
procedure gloWireSuperToroid(inner_radius, outer_radius,
 n1, n2 : Single; slices, stacks : Integer);stdcall;external 'glo.dll';
procedure gloWireSuperEllipsoid(width, n1, n2 : Single;
 slices, stacks : Integer);stdcall;external 'glo.dll';

procedure gloSolidSphere(radius : GLdouble; slices, stacks : GLint);
stdcall;external 'glo.dll';
procedure gloSolidCone(base, height : GLdouble; slices, stacks : GLint);
```

```
stdcall;external 'glo.dll';
procedure gloSolidCylinder(baseRadius, topRadius, height : GLdouble;
 slices, stacks : GLint);stdcall;external 'glo.dll';
procedure gloSolidDisk(innerRadius, outerRadius : GLdouble;
 slices, stacks : GLint);stdcall;external 'glo.dll';
procedure gloSolidCube(size : GLdouble); stdcall;external 'glo.dll';
procedure gloSolidTorus(innerRadius, outerRadius : GLfloat;
nsides, rings : GLint);stdcall;external 'glo.dll';
procedure gloSolidTeapot(scale : GLdouble); stdcall;external 'glo.dll';
procedure gloSolidDodecahedron(); stdcall;external 'glo.dll';
procedure gloSolidOctahedron(); stdcall;external 'glo.dll';
procedure gloSolidIsosahedron(); stdcall;external 'glo.dll';
procedure gloSolidTetrahedron(); stdcall;external 'glo.dll';
procedure gloSolidGear(inner_radius, outer_radius, width: double;
 teeth: integer; tooth_depth: double);stdcall;external 'glo.dll';
procedure gloSolidSuperToroid(inner_radius, outer_radius,
 n1, n2 : Single; slices, stacks : Integer);stdcall;external 'glo.dll';
procedure gloSolidSuperEllipsoid(width, n1, n2 : Single;
 slices, stacks : Integer);stdcall;external 'glo.dll';

var
   ambient : array[0..3] of  GLfloat =( 0, 0, 0, 1 );
   diffuse : array[0..3] of  GLfloat =( 1, 1, 1, 1 );
   specular : array[0..3] of  GLfloat =( 1, 1, 1, 1 );
   position : array[0..3] of  GLfloat =( 0, 3, 3, 0 );
   lmodel_ambient : array[0..3] of  GLfloat =( 0.2, 0.2, 0.2, 1 );
   local_view : array[0..0] of  GLfloat =( 0 );

   Obj1  : GLuint;              // Storage For The Display List
   Obj2  : GLuint;              // Storage For The Second Display List
   Obj3  : GLuint;   Obj4  : GLuint;
   Obj5  : GLuint;   Obj6  : GLuint;
   Obj7  : GLuint;   Obj8  : GLuint;
   Obj9  : GLuint;   Obj10  : GLuint;
   Obj11  : GLuint;   Obj12  : GLuint;
   Obj13  : GLuint;   Obj14  : GLuint;
   Obj15  : GLuint;   Obj16  : GLuint;
   Obj17  : GLuint;   Obj18  : GLuint;
   Obj19  : GLuint;   Obj20  : GLuint;
   Obj21  : GLuint;   Obj22  : GLuint;
   Obj23  : GLuint;   Obj24  : GLuint;

Procedure initGL();
begin
   glClearColor(1,1,1,0);
   glLightfv (GL_LIGHT0, GL_AMBIENT, @ambient);
   glLightfv (GL_LIGHT0, GL_DIFFUSE, @diffuse);
   glLightfv (GL_LIGHT0, GL_POSITION, @position);
   glLightModelfv (GL_LIGHT_MODEL_Ambient, @lmodel_ambient);
   glLightModelfv (GL_LIGHT_MODEL_LOCAL_VIEWER, @local_view);

   glEnable (GL_LIGHTING);
   glEnable (GL_LIGHT0);
   glEnable (GL_DEPTH_TEST);
//  be efficient--make Object display list
   Obj1 := glGenLists(20);
   glNewList (Obj1, GL_COMPILE);
     gloSolidTetrahedron();
```

```
glEndList;

Obj2 := Obj1 + 1 ;
glNewList (Obj2, GL_COMPILE);
  gloSolidSphere(0.7,10,10);
glEndList;

Obj3 := Obj2 + 1 ;
glNewList (Obj3, GL_COMPILE);
  gloSolidCone(0.5,2.70,10,10);
glEndList;

Obj4 := Obj3 + 1 ;
glNewList (Obj4, GL_COMPILE);
  gloSolidCylinder(0.4,0.7,4.7,10,10);
glEndList;

Obj5 := Obj4 + 1 ;
glNewList (Obj5, GL_COMPILE);
  gloSolidDisk(0.3,0.7,10,10);
glEndList;

Obj6 := Obj5 + 1 ;
glNewList (Obj6, GL_COMPILE);
  gloSolidCube(1);
glEndList;

Obj7 := Obj6 + 1 ;
glNewList (Obj7, GL_COMPILE);
  gloSolidTorus(0.1,0.4,10,10);
glEndList;

Obj8 := Obj7 + 1 ;
glNewList (Obj8, GL_COMPILE);
  gloSolidTeapot(1);
glEndList;

Obj9 := Obj8 + 1 ;
glNewList (Obj9, GL_COMPILE);
  gloSolidDodecahedron();
glEndList;

Obj10 := Obj9 + 1 ;
glNewList (Obj10, GL_COMPILE);
  gloSolidOctahedron();
glEndList;

Obj11 := Obj10 + 1 ;
glNewList (Obj11, GL_COMPILE);
  gloSolidIsosahedron();
glEndList;

Obj12 := Obj11 + 1 ;
glNewList (Obj12, GL_COMPILE);
  gloSolidTetrahedron();
glEndList;

Obj13 := Obj12 + 1 ;
```

```
glNewList (Obj13, GL_COMPILE);
  gloSolidGear(0.2,0.7,0.3,4,1);
glEndList;

Obj14 := Obj13 + 1 ;
glNewList (Obj14, GL_COMPILE);
  gloSolidSuperToroid(0.3,0.7,0.5,0.5,10,10);
glEndList;

Obj15 := Obj14 + 1 ;
glNewList (Obj15, GL_COMPILE);
  gloSolidSuperEllipsoid(0.5,0.4,0.4,10,10);
glEndList;

Obj16 := Obj15 + 1 ;
glNewList (Obj16, GL_COMPILE);
  gloWireSphere(1.0,10,10);
glEndList;

Obj17 := Obj16 + 1 ;
glNewList (Obj17, GL_COMPILE);
  gloWireCone(0.5,3.8,10,10);
glEndList;

Obj18 := Obj17 + 1 ;
glNewList (Obj18, GL_COMPILE);
  gloWireCylinder(0.5,0.5,1.8,10,10);
glEndList;

Obj19 := Obj18 + 1 ;
glNewList (Obj19, GL_COMPILE);
  gloWireDisk(0.3,0.8,10,10);
glEndList;

Obj20 := Obj19 + 1 ;
glNewList (Obj20, GL_COMPILE);
  gloWireCube(1.1);
glEndList;

Obj21 := Obj20 + 1 ;
glNewList (Obj21, GL_COMPILE);
  gloWireTorus(0.4,0.8,10,10);
glEndList;

Obj22 := Obj21 + 1 ;
glNewList (Obj22, GL_COMPILE);
  gloWireTeapot(1);
glEndList;

Obj23 := Obj22 + 1 ;
glNewList (Obj23, GL_COMPILE);
  gloWireDodecahedron();
glEndList;

Obj24 := Obj23 + 1 ;
glNewList (Obj24, GL_COMPILE);
  gloWireOctahedron();
glEndList;
```

```
End;

//  Move object into position.  Use 3rd through 12th
//  parameters to specify the material property.  Draw an Object.
procedure renderObject(x, y,
                       ambr, ambg, ambb,
                       difr, difg, difb ,
                       specr, specg, specb,
                       shine : GLfloat;
                       ObjIndex : GLuint);
var
   mat : array[0..3] of  GLfloat ;
begin
  glPushMatrix;
  glRotatef(20,1,1,0);
  glTranslatef (x, y, 0);
  mat[0] := ambr; mat[1] := ambg; mat[2] := ambb; mat[3] := 1;
  glMaterialfv (GL_FRONT, GL_AMBIENT, @mat);
  mat[0] := difr; mat[1] := difg; mat[2] := difb;
  glMaterialfv (GL_FRONT, GL_DIFFUSE, @mat);
  mat[0] := specr; mat[1] := specg; mat[2] := specb;
  glMaterialfv (GL_FRONT, GL_SPECULAR, @mat);
  glMaterialf (GL_FRONT, GL_SHININESS, shine * 128);
  glCallList (ObjIndex);
  glPopMatrix;
End;

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

  renderObject (2, 17, 0.0215, 0.1745, 0.0215,
    0.07568, 0.61424, 0.07568, 0.633, 0.727811, 0.633, 0.6,Obj1);
  renderObject (2, 14, 0.135, 0.2225, 0.1575,
    0.54, 0.89, 0.63, 0.316228, 0.316228, 0.316228, 0.1,Obj2);
  renderObject (2, 11, 0.05375, 0.05, 0.06625,
    0.18275, 0.17, 0.22525, 0.332741, 0.328634, 0.346435, 0.3,Obj3);
  renderObject (2, 8, 0.25, 0.20725, 0.20725,
    1, 0.829, 0.829, 0.296648, 0.296648, 0.296648, 0.088,Obj4);
  renderObject (2, 5, 0.1745, 0.01175, 0.01175,
    0.61424, 0.04136, 0.04136, 0.727811, 0.626959, 0.626959, 0.6,Obj5);
  renderObject (2, 2, 0.1, 0.18725, 0.1745,
    0.396, 0.74151, 0.69102, 0.297254, 0.30829, 0.306678, 0.1,Obj6);
  renderObject (6, 17, 0.329412, 0.223529, 0.027451,
    0.780392, 0.568627, 0.113725, 0.992157, 0.941176, 0.807843,
    0.21794872,Obj7);
  renderObject (6, 14, 0.2125, 0.1275, 0.054,
    0.714, 0.4284, 0.18144, 0.393548, 0.271906, 0.166721, 0.2,Obj8);
  renderObject (6, 11, 0.25, 0.25, 0.25,
    0.4, 0.4, 0.4, 0.774597, 0.774597, 0.774597, 0.6,Obj9);
  renderObject (6, 8, 0.19125, 0.0735, 0.0225,
     0.7038, 0.27048, 0.0828, 0.256777, 0.137622, 0.086014, 0.1,Obj10);
  renderObject (6, 5, 0.24725, 0.1995, 0.0745,
    0.75164, 0.60648, 0.22648, 0.628281, 0.555802, 0.366065, 0.4,Obj11);
  renderObject (6, 2, 0.19225, 0.19225, 0.19225,
     0.50754, 0.50754, 0.50754, 0.508273, 0.508273, 0.508273, 0.4,Obj12);
  renderObject (10, 17, 0, 0, 0, 0.01, 0.01, 0.01,
```

```
    0.5, 0.5, 0.5, 0.25 ,Obj13);
  renderObject (10, 14, 0, 0.1, 0.06, 0, 0.50980392, 0.50980392,
    0.50196078, 0.50196078, 0.50196078, 0.25,Obj14);
  renderObject (10, 11, 0, 0, 0,
    0.1, 0.35, 0.1, 0.45, 0.55, 0.45, 0.25,Obj15);
  renderObject (10, 8, 0, 0, 0, 0.5, 0, 0,
    0.7, 0.6, 0.6, 0.25,Obj16);
  renderObject (10, 5, 0, 0, 0, 0.55, 0.55, 0.55,
    0.7, 0.7, 0.7, 0.25,Obj17);
  renderObject (10, 2, 0, 0, 0, 0.5, 0.5, 0,
    0.6, 0.6, 0.5, 0.25,Obj18);
  renderObject (14, 17, 0.02, 0.02, 0.02, 0.01, 0.01, 0.01,
    0.4, 0.4, 0.4, 0.078125,Obj19);
  renderObject (14, 14, 0, 0.05, 0.05, 0.4, 0.5, 0.5,
    0.04, 0.7, 0.7, 0.078125,Obj20);
  renderObject (14, 11, 0, 0.05, 0, 0.4, 0.5, 0.4,
    0.04, 0.7, 0.04, 0.078125,Obj21);
  renderObject (14, 8, 0.05, 0, 0, 0.5, 0.4, 0.4,
    0.7, 0.04, 0.04, 0.078125,Obj22);
  renderObject (14, 5, 0.05, 0.05, 0.05, 0.5, 0.5, 0.5,
    0.7, 0.7, 0.7, 0.078125,Obj23);
  renderObject (14, 2, 0.05, 0.05, 0, 0.5, 0.5, 0.4,
    0.7, 0.7, 0.04, 0.078125,Obj24);
  SwapBuffers(f_Hdc);
End;
procedure TForm1.FormResize(Sender: TObject);
begin
  wglMakeCurrent(f_Hdc,hrc); //activate the RC
  glViewport (0, 0, width, height);
  glMatrixMode (GL_PROJECTION);
  glLoadIdentity;
  If (width <= height) Then
    glOrtho (0, 16, 0, 16 * height / width, -10, 10)
  Else
    glOrtho (0, 16 * width / height, 0, 16, -10, 10);

  glMatrixMode (GL_MODELVIEW);
 InvalidateRect(Handle, nil, False);// DrawGLScene;  Draw the scene.
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.FormPaint(Sender: TObject);
begin
 DrawGLScene();
 wglMakeCurrent(f_Hdc,hrc); //activate the RC
end;

end.
```