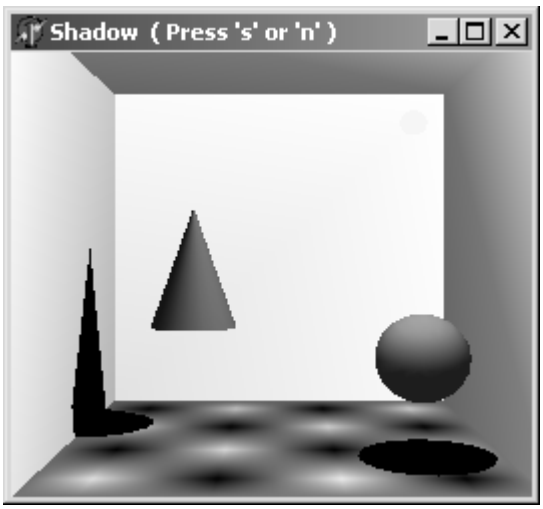
فصل بیست و سوم

خلق سایه

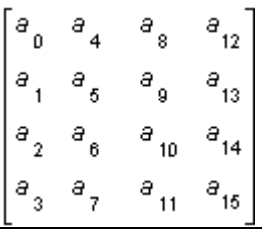


مقدمه :

یکی دیگر از جلوه های طبیعی اشیاء سایه می باشد ، سایه ای که قابلیت امتداد یافتن و افتادن بر روی سایر اشیاء را دارد . فرض آغازین این فصل بر آن است که شما تاکنون اطلاعات قابل توجهی از OpenGL را آموخته اید و نیز به ریاضیات ماتریس ها ، آشنایی کامل دارید !

طریقه ایجاد سایه :

گاهی اوقات لازم است در انجام عملیات ماتریسی OpenGL دخالت کنیم . برای مثال با استفاده از عملیات ماتریسی خاصی ایجاد سایه صورت می گیرد . بدین جهت لازم است در ابتدا مروری داشته باشیم بر نحوه تعریف ماتریس ها در OpenGL . در زیر نحوه قرار گیری یک ماتریس 4*4 در OpenGL و آنچه که در ریاضیات متداول است با هم مقایسه شده اند :

0 1 2 3 4 5 6 7 **THE OPENGL WAY** 8 9 10 11 12 13 14 15	0 4 8 12 1 5 9 13 THE MATHEMATICIANS WAY 2 6 10 14 3 7 11 15 

برای خلق سایه باید ابتدا تعیین شود که کدامیک از سطوح به سمت منبع نور هستند . اینکار با قرار دادن موقعیت منبع نور در تابع findplane که درکد برنامه این فصل توسعه داده شده است ، صورت می گیرد . اگر مقدار حاصل بزرگتر از صفر بود بدین معنا است که جهت آن سطح با جهت نرمال بر صفحه یکی است و به سمت منبع نور می باشد و در غیر اینصورت ، خیر . سپس از تابع glMultMatrixf برای ضرب کردن ماتریس ایجاد سایه که آن نیز در این فصل توسعه داده شده است ، در ماتریس جاری OpenGL استفاده می شود . توسط این عملیات تمام نقاط شیء مورد نظر بر روی صفحات معین شده تصویر خواهند شد (با توجه به موقعیت منبع نور) . بدون استفاده از بافر استنسیل ، محاسبات هر نقطه دوبار صورت می گیرد که بدیهی است بازدهی و زیبایی کار ، کاهش خواهد یافت .

مروری بر توابع :

تابع به فرمت زبان C	تابع به فرمت زبان دلفی

void glMultMatrixf(
  const GLfloat *m
);
	Procedure glMultMatrixf(m: PGLfloat);
	  stdcall; external 'OPENGL32.DLL';

این تابع ماتریس جاری را در یک ماتریس دلخواه ضرب می کند .

آرگومان m : اشاره گری است به ماتریسی 4*4 .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی

void glLoadMatrixf(
  const GLfloat *m
);
	Procedure glLoadMatrixf(m: PGLfloat);
	  stdcall; external 'OPENGL32.DLL';

این تابع ماتریس جاری را با ماتریسی دلخواه جایگزین می نماید .
آرگومان m : اشاره گری است به ماتریسی 4*4 .

برنامه فصل :

```
unit Ch23;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms, Dialogs , OpenGL ,SPF;
type
  TForm1 = class(TForm)
    procedure FormCreate(Sender: TObject);
    procedure FormResize(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure FormPaint(Sender: TObject);
    procedure FormKeyPress(Sender: TObject; var Key: Char);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
implementation
{$R *.dfm}

const
  C_SPHERE = 1;  C_CONE=2;  C_LIGHT=3;
  C_LEFTWALL=4;  C_FLOOR=5;
  X=0;  Y=1;  Z=2;  W=3;
  A=0;  B=1;  C=2;  D=3;
  TEXDIM = 256;
  NONE=0; SHADOW=1;

type
    Mat = array[0..3] of GLfloat;

var
  //* material properties for objects in scene */
  wall_mat :array[0..3] of GLfloat =(1, 1, 1, 1);
  sphere_mat :array[0..3] of GLfloat=(1, 0.5, 0, 1);
  cone_mat :array[0..3] of GLfloat=(0, 0.5, 1, 1);
  leftwallshadow :array[0..3]of Mat;//array[0..3] of GLfloat;
  floorshadow :array[0..3]of Mat;// array[0..3] of GLfloat;
  lightpos :array[0..3] of GLfloat=(50, 50, -320, 1);
  tex :array[0..2] of array[0..3] of array[0..3] of GLfloat;
  rendermode : Integer = NONE;
  f_Hdc : longInt;

// create a matrix that will project the desired shadow */
procedure shadowmatrix(var shadowMat : array of Mat;
```

```
        groundplane, lightpos: array of GLfloat);
var
   dot :GLfloat;
begin

 // find dot product between light position
 //vector and ground plane normal
  dot := groundplane[X] * lightpos[X] +
    groundplane[Y] * lightpos[Y] +
    groundplane[Z] * lightpos[Z] +
    groundplane[W] * lightpos[W];

  shadowMat[0,0] := dot - lightpos[X] * groundplane[X];
  shadowMat[1,0] := 0 - lightpos[X] * groundplane[Y];
  shadowMat[2,0] := 0 - lightpos[X] * groundplane[Z];
  shadowMat[3,0] := 0 - lightpos[X] * groundplane[W];

  shadowMat[X,1] := 0 - lightpos[Y] * groundplane[X];
  shadowMat[1,1] := dot - lightpos[Y] * groundplane[Y];
  shadowMat[2,1] := 0 - lightpos[Y] * groundplane[Z];
  shadowMat[3,1] := 0 - lightpos[Y] * groundplane[W];

  shadowMat[X,2] := 0 - lightpos[Z] * groundplane[X];
  shadowMat[1,2] := 0 - lightpos[Z] * groundplane[Y];
  shadowMat[2,2] := dot - lightpos[Z] * groundplane[Z];
  shadowMat[3,2] := 0 - lightpos[Z] * groundplane[W];
  shadowMat[X,3] := 0 - lightpos[W] * groundplane[X];
  shadowMat[1,3] := 0 - lightpos[W] * groundplane[Y];
  shadowMat[2,3] := 0 - lightpos[W] * groundplane[Z];
  shadowMat[3,3] := dot - lightpos[W] * groundplane[W];
end;

//* find the plane equation given 3 points */
procedure findplane(var plane, v0 ,v1 ,v2 :array of GLfloat);
var
   vec0, vec1 :array[0..2] of GLfloat;
begin
  //* need 2 vectors to find cross product */
  vec0[X] := v1[X] - v0[X];
  vec0[Y] := v1[Y] - v0[Y];
  vec0[Z] := v1[Z] - v0[Z];

  vec1[X] := v2[X] - v0[X];
  vec1[Y] := v2[Y] - v0[Y];
  vec1[Z] := v2[Z] - v0[Z];

  //* find cross product to get A, B, and C of plane equation */
  plane[A] := vec0[Y] * vec1[Z] - vec0[Z] * vec1[Y];
  plane[B] := -(vec0[X] * vec1[Z] - vec0[Z] * vec1[X]);
  plane[C] := vec0[X] * vec1[Y] - vec0[Y] * vec1[X];

  plane[D] := -(plane[A] * v0[X] + plane[B] * v0[Y] + plane[C] * v0[Z]);
end;

procedure make_texture();
begin
    tex[0, 0, 0] := 255; tex[1, 0, 0] := 255;   tex[2, 0, 0] := 255;
    tex[0, 0, 1] := 0;   tex[1, 0, 1] := 0;     tex[2, 0, 1] := 0;
```

```
    tex[0, 0, 2] := 255; tex[1, 0, 2] := 255;   tex[2, 0, 2] := 255;
    tex[0, 0, 3] := 0;   tex[1, 0, 3] := 0;     tex[2, 0, 3] := 0;
    tex[0, 1, 0] := 0;   tex[1, 1, 0] := 0;     tex[2, 1, 0] := 0;
    tex[0, 1, 1] := 255; tex[1, 1, 1] := 255;   tex[2, 1, 1] := 255;
    tex[0, 1, 2] := 0;   tex[1, 1, 2] := 0;     tex[2, 1, 2] := 0;
    tex[0, 1, 3] := 255; tex[1, 1, 3] := 255;   tex[2, 1, 3] := 255;
    tex[0, 2, 0] := 255; tex[1, 2, 0] := 255;   tex[2, 2, 0] := 255;
    tex[0, 2, 1] := 0;   tex[1, 2, 1] := 0;     tex[2, 2, 1] := 0;
    tex[0, 2, 2] := 255; tex[1, 2, 2] := 255;   tex[2, 2, 2] := 255;
    tex[0, 2, 3] := 0;   tex[1, 2, 3] := 0;     tex[2, 2, 3] := 0;
    tex[0, 3, 0] := 0;   tex[1, 3, 0] := 0;     tex[2, 3, 0] := 0;
    tex[0, 3, 1] := 255; tex[1, 3, 1] := 255;   tex[2, 3, 1] := 255;
    tex[0, 3, 2] := 0;   tex[1, 3, 2] := 0;     tex[2, 3, 2] := 0;
    tex[0, 3, 3] := 255; tex[1, 3, 3] := 255;   tex[2, 3, 3] := 255;
end;

Procedure initGL();
var
   m_sphere, m_cone, m_base : GLUquadricObj;
   plane : array[0..3] of GLfloat ;
   v0, v1, v2 : array[0..2] of GLfloat ;
begin
  //* draw a perspective scene */
  glMatrixMode(GL_PROJECTION);
  glFrustum(-100, 100, -100, 100, 320, 640);
  glMatrixMode(GL_MODELVIEW);

  //* turn on features */
  glEnable(GL_DEPTH_TEST);
  glEnable(GL_LIGHTING);
  glEnable(GL_LIGHT0);

  //* make shadow matricies */

  //* 3 points on floor */
  v0[X] := -100;
  v0[Y] := -100;
  v0[Z] := -320;
  v1[X] := 100;
  v1[Y] := -100;
  v1[Z] := -320;
  v2[X] := 100;
  v2[Y] := -100;
  v2[Z] := -520;

  findplane(plane, v0, v1, v2);
  shadowmatrix(floorshadow, plane, lightpos);

  //* 3 points on left wall */
  v0[X] := -100;
  v0[Y] := -100;
  v0[Z] := -320;
  v1[X] := -100;
  v1[Y] := -100;
  v1[Z] := -520;
  v2[X] := -100;
  v2[Y] := 100;
  v2[Z] := -520;
```

```
  findplane(plane, v0, v1, v2);
  shadowmatrix(leftwallshadow, plane, lightpos);

  //* place light 0 in the right place */
  glLightfv(GL_LIGHT0, GL_POSITION, @lightpos);

  //* remove back faces to speed things up */
  glCullFace(GL_BACK);

  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

  //* make display lists for sphere and cone; for efficiency */

  glNewList(C_SPHERE, GL_COMPILE);
    m_sphere := gluNewQuadric();
    gluSphere(m_sphere, 20, 20, 20);
    gluDeleteQuadric(m_sphere);
  glEndList();

  glNewList(C_LIGHT, GL_COMPILE);
    m_sphere := gluNewQuadric();
    gluSphere(m_sphere, 5, 20, 20);
    gluDeleteQuadric(m_sphere);
  glEndList();

  glNewList(C_CONE, GL_COMPILE);
    m_cone := gluNewQuadric();
    m_base := gluNewQuadric();
    glRotatef(-90, 1, 0, 0);
    gluDisk(m_base, 0, 20, 20, 1);
    gluCylinder(m_cone, 20., 0., 60., 20, 20);
    gluDeleteQuadric(m_cone);
    gluDeleteQuadric(m_base);
  glEndList();

  glNewList(C_FLOOR, GL_COMPILE);
  glEndList();

  glNewList(C_LEFTWALL, GL_COMPILE);
  glEndList();

  //* load pattern for current 2d texture */
  make_texture();
  glTexImage2D(GL_TEXTURE_2D, 0, 1, 4, 4,
            0, GL_RED, GL_FLOAT, @tex);
end;

procedure sphere();
begin
  glPushMatrix();
  glTranslatef(60, -50, -360);
  glCallList(C_SPHERE);
  glPopMatrix();
end;

procedure cone();
begin
```

```
  glPushMatrix();
  glTranslatef(-40, -40, -400);
  glCallList(C_CONE);
  glPopMatrix();
end;

procedure redraw();
begin
  glClear( GL_DEPTH_BUFFER_BIT or GL_COLOR_BUFFER_BIT
          or GL_STENCIL_BUFFER_BIT);

  // Note: wall verticies are ordered so they are all
  // front facing this lets me do back face culling to speed things up.
  glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, @wall_mat);
  // floor */
  // make the floor textured */
  glEnable(GL_TEXTURE_2D);

  { Since we want to turn texturing on for floor only,
    we have to make floor
    a separate glBegin()/glEnd() sequence. You can't turn
    texturing on and off between begin and end calls }
  glBegin(GL_QUADS);
  glNormal3f(0, 1, 0);
  glTexCoord2i(0, 0);
  glVertex3f(-100, -100, -320);
  glTexCoord2i(1, 0);
  glVertex3f(100, -100, -320);
  glTexCoord2i(1, 1);
  glVertex3f(100, -100, -520);
  glTexCoord2i(0, 1);
  glVertex3f(-100, -100, -520);
  glEnd();

  glDisable(GL_TEXTURE_2D);
  if (rendermode = SHADOW) then
   begin
    glDisable(GL_DEPTH_TEST);
    glDisable(GL_LIGHTING);
    glColor3f(0, 0, 0);  // shadow color */

    glPushMatrix();
    glMultMatrixf(@floorshadow);
    cone();
    glPopMatrix();

    glPushMatrix();
    glMultMatrixf(@floorshadow);
    sphere();
    glPopMatrix();
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_LIGHTING);
  end;
  // walls */

  if (rendermode = SHADOW) then
  begin
    glEnable(GL_STENCIL_TEST);
```

```
  glStencilFunc(GL_ALWAYS, 1, 0);
  glStencilOp(GL_KEEP, GL_KEEP, GL_REPLACE);
end;
glBegin(GL_QUADS);
// left wall */
glNormal3f(1, 0, 0);
glVertex3f(-100, -100, -320);
glVertex3f(-100, -100, -520);
glVertex3f(-100, 100, -520);
glVertex3f(-100, 100, -320);
glEnd();

if (rendermode = SHADOW) then
begin
  glStencilFunc(GL_EQUAL, 1, 1);
  glDisable(GL_DEPTH_TEST);
  glDisable(GL_LIGHTING);
  glColor3f(0, 0, 0);  //* shadow color */
  glDisable(GL_DEPTH_TEST);
  glPushMatrix();
  glMultMatrixf(@leftwallshadow);
  cone();
  glPopMatrix();
  glEnable(GL_DEPTH_TEST);
  glDisable(GL_STENCIL_TEST);
  glEnable(GL_DEPTH_TEST);
  glEnable(GL_LIGHTING);
end;
glBegin(GL_QUADS);
// right wall */
glNormal3f(-1, 0, 0);
glVertex3f(100, -100, -320);
glVertex3f(100, 100, -320);
glVertex3f(100, 100, -520);
glVertex3f(100, -100, -520);

// ceiling */
glNormal3f(0, -1, 0);
glVertex3f(-100, 100, -320);
glVertex3f(-100, 100, -520);
glVertex3f(100, 100, -520);
glVertex3f(100, 100, -320);

// back wall */
glNormal3f(0, 0, 1);
glVertex3f(-100, -100, -520);
glVertex3f(100, -100, -520);
glVertex3f(100, 100, -520);
glVertex3f(-100, 100, -520);
glEnd();

glPushMatrix();
 glTranslatef(lightpos[X], lightpos[Y], lightpos[Z]);
 glDisable(GL_LIGHTING);
 glColor3f(1, 1, 0.7);
 glCallList(C_LIGHT);
 glEnable(GL_LIGHTING);
glPopMatrix();
```

```
  glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, @cone_mat);
  cone();

  glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, @sphere_mat);
  sphere();

  SwapBuffers(f_Hdc); // high end machines may need this */
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
 f_Hdc := GetDC( form1.handle );
 SetDCPixelFormat(f_Hdc,16,16);// Create a rendering context.
 initGL();
end;

procedure TForm1.FormResize(Sender: TObject);
begin
   wglMakeCurrent(f_Hdc,hrc); //activate the RC
   glViewport( 0, 0, Width, Height);
   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.FormPaint(Sender: TObject);
begin
 wglMakeCurrent(f_Hdc,hrc); //activate the RC
  reDraw();                  // Draw the scene
end;

procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  if key='s' then rendermode:=SHADOW;
  if key='n' then rendermode:=NONE;

  InvalidateRect(Handle, nil, False);//  Draw the scene.
end;

end.
```