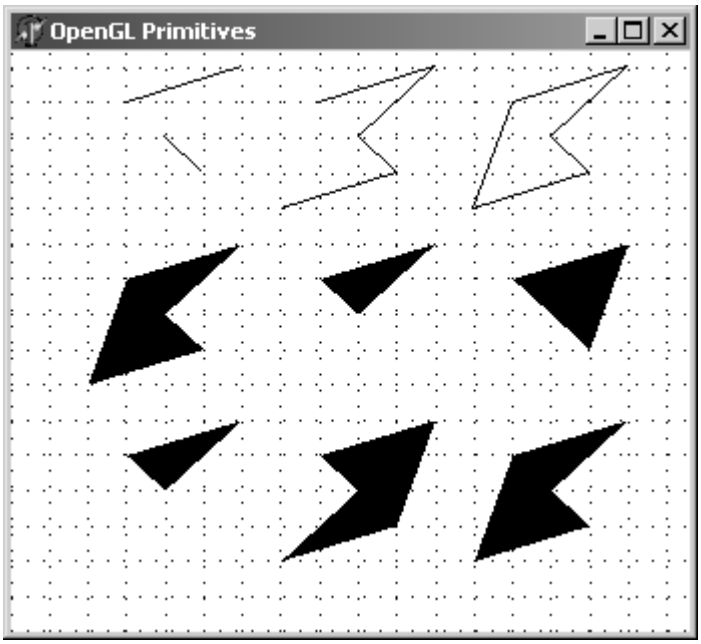
فصل چهارم

ترسیم اشکال اولیه دوبعدی :
نقاط ، خطوط و چند ضلعی ها



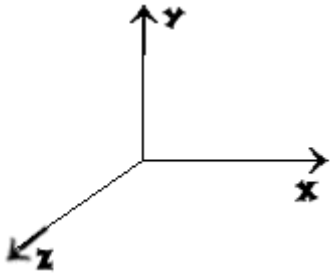
مقدمه :

قبل از اینکه ترسیم اشیای سه بعدی را آغاز کنیم ، لازم است تا با تعدادی از اصول اولیه ترسیمات دو بعدی آشنا شویم. OpenGL از سیستم مختصات سه بعدی کارتزین استفاده می کند . با استفاده از این سیستم ، می توان نقطه ای را در فضای سه بعدی ارائه داد که به آن راس (Vertex) گوئیم و با سه مختص x ، y و z تعریف می شود . همانطور که در فصول قبلی به آن اشاره شد،با استفاده از تابع glVertex می توان رئوس را معرفی نمود (البته بین دو تابع glBegin و glEnd) . هر شیء هندسی با این رئوس و نوع شکلی که باید ترسیم گردد ، تعریف می شود .

OpenGL ثوابت متعددی را برای ترسیم چند ضلعی ها و اشکال اولیه ارائه داده است ، مانند GL_POLYGON و غیره که اشاره ای به آنها گردید . مطلبی را که باید بخاطر داشت این است که اغلب شتاب دهنده های گرافیکی برای ترسیم مثلث ها بهینه گشته اند ، بنابراین بدیهی است که از GL_TRIANGLES بیش از همه استفاده شود و فراموش نکنید که پس از دستورات ترسیمی باید از تابع glFlush برای اطمینان حاصل نمودن از اجرای آنها ، استفاده نمود .

سیستم مختصات در OpenGL :

سیستم کارتزینی که در OpenGL مورد استفاده قرار می گیرد در شکل زیر نشان داده شده است :



تصویری از نحوه قرار گیری محورها در سیستم مختصاتی OpenGL .

برای حرکت و انتقال در این دستگاه مختصات از تابع glTranslatef استفاده می شود .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
void glTranslatef(GLfloat x, GLfloat y, GLfloat z);	Procedure glTranslatef(x: GLfloat; y: GLfloat; z: GLfloat); stdcall; external 'OPENGL32.DLL';

تابع glTranslatef ماتریس جاری را در ماتریس انتقال ضرب می کند . x ، y و z مختص های بردار انتقال هستند .

ماتریس انتقال	$\begin{pmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{pmatrix}$

برای مثال تابع glTranslatef(-1,0,-6); سبب حرکت به اندازه ۱ واحد به سمت چپ ، ۶ واحد به سمت داخل صفحه و هیچ حرکتی روی محور y می شود. هنگامیکه از این تابع استفاده می شود ، حرکت از مرکز صفحه صورت نمی گیرد بلکه ، حرکت از محلی است که هم اکنون شی ایی در آنجا ترسیم گشته است (یا سیستم مختصات محلی را به همان مقدار حرکت می دهد) .
با استفاده از تابع glLoadIdentity می توان هر چیزی را به مکان اولیه اش باز گرداند .

تبدیلات در OpenGL :

OpenGL از ماتریس های زیادی منجمله 'Projection' برای انتقال یک نقطه ، قبل از نمایش آن روی صفحه ، استفاده می کند . ماتریس Projection چگونگی گرفتن یک نقطه از دنیای سه بعدی و قرار دادن آن روی یک صفحه دو بعدی را بیان می کند (تصویر کردن) . برای انجام این کار ابتدا باید به OpenGL گفته شود که ماتریس Projection را می خواهیم تغییر دهیم . برای این منظور از دستور glMatrixMode(GL_PROJECTION); استفاده می شود . سپس تصویر کردن را می توان با استفاده از توابع glOrtho ویا gluOrtho2D انجام داد ، که شرح آنها در فصول قبلی رفت . Projection برای برپایی درگاه دید و وجوه مکعب مستطیلی که اشیای ما در آن واقع می شوند (ایجاد دستگاه مختصات) بکار می رود . ماتریس دیگری که در OpenGL برای انتقال ، دوران و تغییر ابعاد اشیاء بکار گرفته می شود ModelView نام دارد . ماتریس ModelView در حقیقت از دو ماتریس به شکل یک ماتریس تشکیل شده است . ماتریس اول ، ماتریس دید می باشد و مکان بیننده را در دنیای سه بعدی مشخص می کند . ماتریس دوم ماتریس مدل بوده و نقاط اصلی شیء را دریافت کرده و آنرا در دنیای سه بعدی نمایش می دهد . برای فراخوانی آن از دستور glMartixMode(GL_MODELVIEW); کمک گرفته می شود .

توابع دیگر ترسیم اشکال اولیه در OpenGL :

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
void glRectf(GLfloat x1, GLfloat y1, GLfloat x2, GLfloat y2); void glRectfv(const GLfloat *v1, const GLfloat *v2);	type PGLfloat = ^GLfloat; Procedure glRectf(x1: GLfloat; y1: GLfloat; x2: GLfloat; y2: GLfloat); stdcall; external 'OPENGL32.DLL'; Procedure glRectfv(v1: PGLfloat; v2: PGLfloat); stdcall; external 'OPENGL32.DLL';

توضیح :

glRect برای ترسیم یک مستطیل بکار می رود. x1,y1 و x2,y2 رئوس در جهت مخالف هم هستند . V1 و V2 اشاره گر هایی هستند به رئوس مخالف مستطیل . تابع glRect(x1, y1, x2, y2) با دستورات زیر معادل است :

```
glBegin(GL_POLYGON);
  glVertex2(x1, y1);
  glVertex2(x2, y1);
  glVertex2(x2, y2);
  glVertex2(x1, y2);
glEnd( );
```

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
void glPointSize(GLfloat size);	Procedure glPointSize(size: GLfloat); stdcall; external 'OPENGL32.DLL';

توضیح :

قطر یک نقطه را معین می کند . مقدار پیش فرض یک می باشد .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
void glLineWidth(GLfloat width);	Procedure glLineWidth(width: GLfloat); stdcall; external 'OPENGL32.DLL';

توضیح :

ضخامت خط را معین می سازد . مقدار پیش فرض آن یک می باشد .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
void glLineStipple(GLint factor, GLushort pattern);	type UNSIGNED_SHORT = Word; type INT = Integer; type GLint = INT; type GLushort = UNSIGNED_SHORT; Procedure glLineStipple(factor: GLint; pattern: GLushort); stdcall; external 'OPENGL32.DLL';

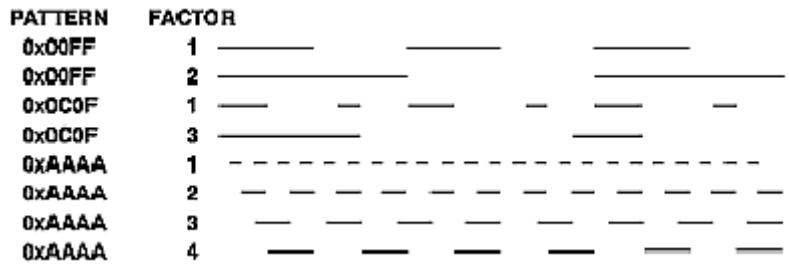
توضیح :

الگوی سایه زدن خط را معین می کند . factor در هر بیت الگوی سایه زدن ضرب می شود . برای مثال اگرfactor مساوی ۳ است،هر بیت در الگو، سه مرتبه قبل از بیت بعدی در الگو بکار می رود . این عدد در بازه یک و ۲۵۶ قرار دارد و مقدار پیش فرض آن یک می باشد .

Pattern : عدد ۱۶ بیتی صحیح ، بطوریکه الگوی بیتی آن (سری صفر و یک ها) معین می سازد که کدام اجزاء از خط رسم خواهند شد (عدد یک به معنای ترسیم و صفر به معنای عدم ترسیم می باشد) . بیت صفر در ابتدا بکار می رود و تمام الگوی پیش فرض از یک تشکیل شده است.

قبل از بکار بردن این تابع باید از دستور ;glEnable(GL_LINE_STIPPLE) برای فعال سازی این حالت استفاده کرد . برای غیر فعال سازی آن ، همانطور که ذکر گردید از تابع glDisable با آرگومان یاد شده ، استفاده می شود .

برای مثال با الگوی $3F07 که در مبنای دو معادل 0011111100000111 می باشد ، خطی رسم خواهد شد با سه نقطه روشن ، ۵ نقطه خاموش ، ۶ نقطه روشن و ۲ نقطه خاموش . اگر factor مساوی ۲ باشد ، الگو کشیده می شود : ۶ نقطه روشن ، ۱۰ نقطه خاموش ، ۱۲ نقطه روشن و ۴ نقطه خاموش . در تصویر زیر مثالهای بیشتری ارائه شده اند . باید خاطر نشان کرد که اعداد مبنای ۱۶ در زبان C با 0X آغاز می شوند و در زبان دلفی با $.



تصویری از اجرای تابع glLineStipple با آرگومانهای متفاوت .

اولین برنامه فصل :

در این برنامه مرور تقریبا کاملی بر ترسیمات دوبعدی خواهیم داشت . (حتما به یاد دارید که واحد SPF را در فصول قبلی ایجاد نمودیم .)

```
unit ch04;

interface

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

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

var
  Form1: TForm1;

implementation
{$R *.DFM}

var
 f_Hdc : LongInt;

procedure InitGL;          // All Setup For OpenGL Goes Here
begin
// select clearing color
   glClearColor( 1.0, 1.0, 1.0, 0);
   glShadeModel (GL_FLAT);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
end;

procedure drawOneLine(x1,y1,x2,y2 : single);
begin
  glBegin(GL_LINES);
    glVertex2f(x1,y1);  glVertex2f(x2,y2);
  glEnd();
end;

procedure DrawGLScene();
 // Here's Where We Do All The Drawing!!!
var i: integer;
begin
   glClear (GL_COLOR_BUFFER_BIT);
{/* select black for all lines  */}
   glColor3f (0.0, 0.0, 0.0);

   glLoadIdentity();
glEnable (GL_LINE_STIPPLE);
glLineStipple (1, $0101);  //  dotted
   for i:=-9 to 9 do
     begin
      drawoneline(i,-9,i,9);
      drawoneline(-9,i,9,i);
     end;
glDisable (GL_LINE_STIPPLE);

        glLoadIdentity();
       // Translate the object
       glTranslatef (-5, 5, 0);
```

```
//Draw the verticies connected according to the GL_LINES primitive
glBegin(GL_LINES);
      glVertex2f( -1, 1);  // Vertex 1
      glVertex2f( 2, 2);   // Vertex 2
      glVertex2f( 0, 0);   // Vertex 3
      glVertex2f( 1, -1);  // Vertex 4
      glVertex2f( -2, -2); // Vertex 5
glEnd();

glLoadIdentity();  //reset or going back to origin.
glTranslatef (0, 5, 0);  // going to new origin.

glBegin (GL_LINE_STRIP);
      glVertex2f (-1, 1); glVertex2f (2, 2);
      glVertex2f (0, 0);  glVertex2f (1, -1);
      glVertex2f (-2, -2);
glEnd();

glLoadIdentity();
glTranslatef (5, 5, 0);

glBegin (GL_LINE_LOOP);
      glVertex2f (-1, 1); glVertex2f (2, 2);
      glVertex2f (0, 0);  glVertex2f (1, -1);
      glVertex2f (-2, -2);
glEnd();

glLoadIdentity();
glTranslatef (-5, 0, 0);

glBegin (GL_POLYGON);
      glVertex2f (-1, 1); glVertex2f (2, 2);
      glVertex2f (0, 0);  glVertex2f (1, -1);
      glVertex2f (-2, -2);
glEnd();

glLoadIdentity();
glTranslatef (0, 0, 0);

glBegin (GL_QUADS);
      glVertex2f (-1, 1); glVertex2f (2, 2);
      glVertex2f (0, 0);  glVertex2f (1, -1);
      glVertex2f (-2, -2);
glEnd();

glLoadIdentity();
glTranslatef (5, 0, 0);

glBegin (GL_QUAD_STRIP);
      glVertex2f (-1, 1); glVertex2f (2, 2);
      glVertex2f (0, 0);  glVertex2f (1, -1);
      glVertex2f (-2, -2);
glEnd();

glLoadIdentity();
glTranslatef (-5, -5, 0);

glBegin (GL_TRIANGLES);
```

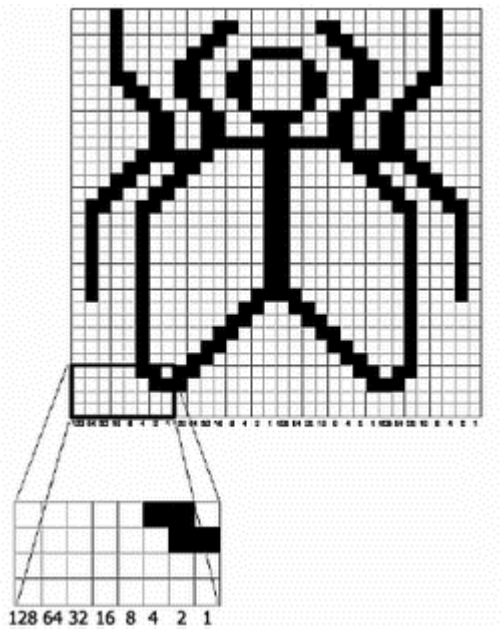
```
            glVertex2f (-1, 1); glVertex2f (2, 2);
            glVertex2f (0, 0);  glVertex2f (1, -1);
            glVertex2f (-2, -2);
        glEnd();

        glLoadIdentity();
        glTranslatef (0, -5, 0);

        glBegin (GL_TRIANGLE_STRIP);
            glVertex2f (-1, 1); glVertex2f (2, 2);
            glVertex2f (0, 0);  glVertex2f (1, -1);
            glVertex2f (-2, -2);
        glEnd();

        glLoadIdentity();
        glTranslatef (5, -5, 0);

        glBegin (GL_TRIANGLE_FAN);
            glVertex2f (-1, 1); glVertex2f (2, 2);
            glVertex2f (0, 0);  glVertex2f (1, -1);
            glVertex2f (-2, -2);
        glEnd ();

    glFlush ();
end;

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

procedure TForm1.FormResize(Sender: TObject);
 begin
  // Redefine the viewing volume and viewport
  //when the window size changes.
    wglMakeCurrent(f_Hdc, hrc);
    // Now, set up the viewing area-select the full client area
    glViewport( 0, 0, Width , Height);
    // Choose the projection matrix to be the matrix
    // manipulated by the following calls
        glMatrixMode (GL_PROJECTION);
  // Set the projection matrix to be the identity matrix
        glLoadIdentity();
  // Define the dimensions of the Orthographic Viewing Volume
```

```
        glOrtho (-13, 13, -13, 13, -13, 13);
  //      gluOrtho2D(-9, 9, -9, 9);
  // Choose the modelview matrix to be the matrix
  // manipulated by further calls
       glMatrixMode (GL_MODELVIEW); // this is Necessary for glTranslatef !
   DrawGLScene();
 end;
end.
```

تابع به فرمت زبان دلفی	تابع به فرمت زبان C
Procedure glPolygonStipple(mask: PGLubyte); stdcall; external 'OPENGL32.DLL';	void glPolygonStipple(const GLubyte *mask);

توضیح :

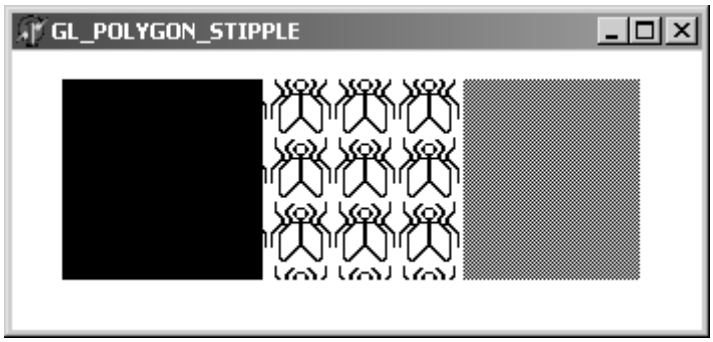
الگوی سایه زدن یک چند ضلعی را معین می سازد . mask اشاره گری است به یک الگوی 32×32 بیتی . قبل از بکار گیری تابع باید این حالت را بوسیله دستور glEnable(GL_POLYGON_STIPPLE); فعال نمود . بصورت پیش فرض تمام چند ضلعی ها با یک الگوی توپر ترسیم می شوند . برای توضیحات بیشتر به شکل زیر و مثال دوم این فصل مراجعه کنید .



تصویری از چگونگی ایجاد یک mask بیتی .

دومین برنامه فصل :

در این برنامه قصد داریم ، الگوی بیتی فوق را ایجاد نماییم .




```
unit alakiGL;

interface

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

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

var
  Form1: TForm1;
 // hrc: HGLRC;  // Permanent Rendering Context
  fly : array[0..127] of GLubyte =(
      $00, $00, $00, $00, $00, $00, $00, $00,
      $03, $80, $01, $C0, $06, $C0, $03, $60,
      $04, $60, $06, $20, $04, $30, $0C, $20,
      $04, $18, $18, $20, $04, $0C, $30, $20,
      $04, $06, $60, $20, $44, $03, $C0, $22,
      $44, $01, $80, $22, $44, $01, $80, $22,
      $44, $01, $80, $22, $44, $01, $80, $22,
      $44, $01, $80, $22, $44, $01, $80, $22,
      $66, $01, $80, $66, $33, $01, $80, $CC,
      $19, $81, $81, $98, $0C, $C1, $83, $30,
      $07, $e1, $87, $e0, $03, $3f, $fc, $c0,
      $03, $31, $8c, $c0, $03, $33, $cc, $c0,
      $06, $64, $26, $60, $0c, $cc, $33, $30,
      $18, $cc, $33, $18, $10, $c4, $23, $08,
      $10, $63, $C6, $08, $10, $30, $0c, $08,
      $10, $18, $18, $08, $10, $00, $00, $08);
```

```
  halftone : array[0..127] of GLubyte =(
      $AA, $AA, $AA, $AA, $55, $55, $55, $55,
      $AA, $AA, $AA, $AA, $55, $55, $55, $55,
      $AA, $AA, $AA, $AA, $55, $55, $55, $55,
      $AA, $AA, $AA, $AA, $55, $55, $55, $55,
      $AA, $AA, $AA, $AA, $55, $55, $55, $55,
      $AA, $AA, $AA, $AA, $55, $55, $55, $55,
      $AA, $AA, $AA, $AA, $55, $55, $55, $55,
      $AA, $AA, $AA, $AA, $55, $55, $55, $55,
      $AA, $AA, $AA, $AA, $55, $55, $55, $55,
      $AA, $AA, $AA, $AA, $55, $55, $55, $55,
      $AA, $AA, $AA, $AA, $55, $55, $55, $55,
      $AA, $AA, $AA, $AA, $55, $55, $55, $55,
      $AA, $AA, $AA, $AA, $55, $55, $55, $55,
      $AA, $AA, $AA, $AA, $55, $55, $55, $55,
      $AA, $AA, $AA, $AA, $55, $55, $55, $55,
      $AA, $AA, $AA, $AA, $55, $55, $55, $55);

implementation
{$R *.DFM}

var
   f_Hdc : LongInt;

procedure InitGL;           // All Setup For OpenGL Goes Here
begin
// select clearing color
   glClearColor( 1.0, 1.0, 1.0, 0);
   glShadeModel (GL_FLAT);
end;

procedure drawOneLine(x1,y1,x2,y2 : single);
begin
  glBegin(GL_LINES);
    glVertex2f(x1,y1);  glVertex2f(x2,y2);
  glEnd();
end;

procedure DrawGLScene();
 // Here's Where We Do All The Drawing!!!
begin
   glClear (GL_COLOR_BUFFER_BIT);
{/* select black for all lines  */}
   glColor3f (0.0, 0.0, 0.0);

{/*  draw one solid, unstippled rectangle,       */
/*  then two stippled rectangles                 */}
   glRectf (25.0, 25.0, 125.0, 125.0);
   glEnable (GL_POLYGON_STIPPLE);
   glPolygonStipple(@fly);
   glRectf (125.0, 25.0, 225.0, 125.0);
   glPolygonStipple (@halftone);
   glRectf (225.0, 25.0, 325.0, 125.0);
   glDisable (GL_POLYGON_STIPPLE);
 glFlush ();
end;
```

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

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

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

procedure TForm1.FormResize(Sender: TObject);
 begin
  // Redefine the viewing volume and viewport
  //when the window size changes.
   wglMakeCurrent(f_Hdc, hrc);
   // Now, set up the viewing area-select the full client area
     glViewport( 0, 0, Width , Height);
   // Choose the projection matrix to be the matrix
   // manipulated by the following calls
   glMatrixMode (GL_PROJECTION);
   glLoadIdentity ();
   gluOrtho2D (0.0, width, 0.0, height);
   DrawGLScene();
 end;
end.
```

نحوه بررسی خطاها در OpenGL :

پس از فراخوانی دستورات OpenGL با استفاده از تابع glGetError می توان از ایجـاد و یـا عـدم ایجاد خطا اطمینان حاصل کرد . کد زیر نحوه انجام این کار را نشان می دهد :

```
{error checking}
errorCode:=glGetError;
if errorCode <> GL_NO_ERROR then
 raise Exception.Create('Error...'#13+gluErrorString(errorCode));
```