فصل نوزدهم

نگاشت محیط

مقدمه :

در این فصل قصد داریم نگاشتی از تصویر محیط را روی اشیاء بوجود آوریم . بدین طریق شیء انعکاس دهنده اطراف خود خواهد شد . برای مثال اگر شما به یک فلز کاملا پولیش شده نگاه کنید ، می توانید تصویر محیط اطراف را مانند یک آینه بر روی آن ببینید . این کار در OpenGL توسط تابع glTexGeni و پارامتر GL_SPHERE_MAP بصورت خودکار انجام می شود .

واحد Textures :

در ادامه واحد جدیدی بنام Textures معرفی می گردد که توانایی بارگذاری تصاویر از فایل و تبدیل آنها به texture مناسب OpenGL را دارد .

```
//----------------------------------------------------------------------------
// Description : A unit that used with OpenGL projects to load BMP, JPG and TGA
//            files from the disk or a resource file.
// Usage       : LoadTexture(Filename, TextureName, LoadFromResource);
//
//            eg : LoadTexture('logo.jpg', LogoTex, TRUE);
//                 will load a JPG texture from the resource included
//                 with the EXE. File has to be loaded into the Resource
//                 using this format  "logo JPEG logo.jpg"
//
//----------------------------------------------------------------------------
unit Textures;
interface
uses
  Windows, OpenGL, Graphics, Classes, JPEG, SysUtils;

function LoadTexture(Filename: String; var Texture: GLuint;
        LoadFromRes : Boolean): Boolean;

implementation

function gluBuild2DMipmaps(Target: GLenum;
        Components, Width, Height: GLint; Format, atype: GLenum;
        Data: Pointer): GLint; stdcall; external glu32;
procedure glGenTextures(n: GLsizei; var textures: GLuint);
        stdcall; external opengl32;
procedure glBindTexture(target: GLenum; texture: GLuint);
        stdcall; external opengl32;

{-------------------------------------------------------------------}
{  Create the Texture                                               }
{-------------------------------------------------------------------}
function CreateTexture(Width, Height, Format : Word;
        pData : Pointer) : Integer;
var
  Texture : GLuint;
begin
  glGenTextures(1, Texture);
  glBindTexture(GL_TEXTURE_2D, Texture);
  {Texture blends with object background}
  glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
  {Texture does NOT blend with object background}
//  glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);

  { Select a filtering type. BiLinear filtering produces
    very good results with little performance impact
    GL_NEAREST              - Basic texture (grainy looking texture)
    GL_LINEAR               - BiLinear filtering
    GL_LINEAR_MIPMAP_NEAREST - Basic mipmapped texture
    GL_LINEAR_MIPMAP_LINEAR  - BiLinear Mipmapped texture
  }
 { only first two can be used }
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
  { all of the above can be used }
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
```

```
  if Format = GL_RGBA then
    gluBuild2DMipmaps(GL_TEXTURE_2D, GL_RGBA, Width, Height,
                      GL_RGBA, GL_UNSIGNED_BYTE, pData)
  else
    gluBuild2DMipmaps(GL_TEXTURE_2D, 3, Width, Height,
                      GL_RGB, GL_UNSIGNED_BYTE, pData);
//  glTexImage2D(GL_TEXTURE_2D, 0, 3, Width, Height,
// 0, GL_RGB, GL_UNSIGNED_BYTE, pData);
// Use when not wanting mipmaps to be built by openGL

  result :=Texture;
end;

{------------------------------------------------------------------}
{  Load BMP textures                                               }
{------------------------------------------------------------------}
function LoadBMPTexture(Filename: String;
        var Texture : GLuint; LoadFromResource : Boolean) : Boolean;
var
  FileHeader: BITMAPFILEHEADER;
  InfoHeader: BITMAPINFOHEADER;
  Palette: array of RGBQUAD;
  BitmapFile: THandle;
  BitmapLength: LongWord;
  PaletteLength: LongWord;
  ReadBytes: LongWord;
  Front: ^Byte;
  Back: ^Byte;
  Temp: Byte;
  I : Integer;
  Width, Height : Integer;
  pData : Pointer;

  // used for loading from resource
  ResStream : TResourceStream;
begin
  result :=FALSE;

  if LoadFromResource then // Load from resource
  begin
    try
      ResStream := TResourceStream.Create(hInstance,
        PChar(copy(Filename, 1, Pos('.', Filename)-1)), 'BMP');
      ResStream.ReadBuffer(FileHeader, SizeOf(FileHeader));  // FileHeader
      ResStream.ReadBuffer(InfoHeader, SizeOf(InfoHeader));  // InfoHeader
      PaletteLength := InfoHeader.biClrUsed;
      SetLength(Palette, PaletteLength);
      ResStream.ReadBuffer(Palette, PaletteLength);          // Palette

      Width := InfoHeader.biWidth;
      Height := InfoHeader.biHeight;

      BitmapLength := InfoHeader.biSizeImage;
      if BitmapLength = 0 then
        BitmapLength := Width * Height * InfoHeader.biBitCount Div 8;

      GetMem(pData, BitmapLength);
      ResStream.ReadBuffer(pData^, BitmapLength);      // Bitmap Data
```

```
    ResStream.Free;
  except on
    EResNotFound do
    begin
      MessageBox(0, PChar('File not found in resource - ' + Filename),
              PChar('BMP Texture'), MB_OK);
      Exit;
    end
    else
    begin
      MessageBox(0, PChar('Unable to read from resource - ' + Filename),
       PChar('BMP Unit'), MB_OK);
      Exit;
    end;
  end;
end
else
begin   // Load image from file
  BitmapFile := CreateFile(PChar(Filename), GENERIC_READ, FILE_SHARE_READ,
   nil, OPEN_EXISTING, 0, 0);
  if (BitmapFile = INVALID_HANDLE_VALUE) then begin
    MessageBox(0, PChar('Error opening ' + Filename),
     PChar('BMP Unit'), MB_OK);
    Exit;
  end;

  // Get header information
  ReadFile(BitmapFile, FileHeader, SizeOf(FileHeader), ReadBytes, nil);
  ReadFile(BitmapFile, InfoHeader, SizeOf(InfoHeader), ReadBytes, nil);

  // Get palette
  PaletteLength := InfoHeader.biClrUsed;
  SetLength(Palette, PaletteLength);
  ReadFile(BitmapFile, Palette, PaletteLength, ReadBytes, nil);
  if (ReadBytes <> PaletteLength) then begin
    MessageBox(0, PChar('Error reading palette'),
     PChar('BMP Unit'), MB_OK);
    Exit;
  end;

  Width  := InfoHeader.biWidth;
  Height := InfoHeader.biHeight;
  BitmapLength := InfoHeader.biSizeImage;
  if BitmapLength = 0 then
    BitmapLength := Width * Height * InfoHeader.biBitCount Div 8;

  // Get the actual pixel data
  GetMem(pData, BitmapLength);
  ReadFile(BitmapFile, pData^, BitmapLength, ReadBytes, nil);
  if (ReadBytes <> BitmapLength) then begin
    MessageBox(0, PChar('Error reading bitmap data'),
     PChar('BMP Unit'), MB_OK);
    Exit;
  end;
  CloseHandle(BitmapFile);
end;

// Bitmaps are stored BGR and not RGB, so swap the R and B bytes.
```

```
  for I :=0 to Width * Height - 1 do
  begin
    Front := Pointer(Integer(pData) + I*3);
    Back := Pointer(Integer(pData) + I*3 + 2);
    Temp := Front^;
    Front^ := Back^;
    Back^ := Temp;
  end;

  Texture :=CreateTexture(Width, Height, GL_RGB, pData);
  FreeMem(pData);
  result :=TRUE;
end;

{------------------------------------------------------------------}
{  Load JPEG textures                                              }
{------------------------------------------------------------------}
function LoadJPGTexture(Filename: String;
        var Texture: GLuint; LoadFromResource : Boolean): Boolean;
var
  Data : Array of LongWord;
  W, Width : Integer;
  H, Height : Integer;
  BMP : TBitmap;
  JPG : TJPEGImage;
  C : LongWord;
  Line : ^LongWord;
  ResStream : TResourceStream;      // used for loading from resource
begin
  result :=FALSE;
  JPG:=TJPEGImage.Create;

  if LoadFromResource then // Load from resource
  begin
    try
      ResStream := TResourceStream.Create(hInstance,
        PChar(copy(Filename, 1, Pos('.', Filename)-1)), 'JPEG');
      JPG.LoadFromStream(ResStream);
      ResStream.Free;
    except on
      EResNotFound do
      begin
        MessageBox(0, PChar('File not found in resource - ' + Filename),
          PChar('JPG Texture'), MB_OK);
        Exit;
      end
      else
      begin
        MessageBox(0, PChar('Couldn''t load JPG Resource - "'+ Filename +'"'),
          PChar('BMP Unit'), MB_OK);
        Exit;
      end;
    end;
  end
  else
  begin
    try
      JPG.LoadFromFile(Filename);
```

```
    except
      MessageBox(0, PChar('Couldn''t load JPG - "'+ Filename +'"'),
        PChar('BMP Unit'), MB_OK);
      Exit;
    end;
  end;

  // Create Bitmap
  BMP:=TBitmap.Create;
  BMP.pixelformat:=pf32bit;
  BMP.width:=JPG.width;

  BMP.height:=JPG.height;
  BMP.canvas.draw(0,0,JPG);        // Copy the JPEG onto the Bitmap

  //  BMP.SaveToFile('D:\test.bmp');
  Width :=BMP.Width;
  Height :=BMP.Height;
  SetLength(Data, Width*Height);

  For H:=0 to Height-1 do
  Begin
    Line :=BMP.scanline[Height-H-1];   // flip JPEG
    For W:=0 to Width-1 do
    Begin
      c:=Line^ and $FFFFFF; // Need to do a color swap
 Data[W+(H*Width)] :=
 (((c and $FF) shl 16)+(c shr 16)+(c and $FF00)) or $FF000000;  // 4 channel.
      inc(Line);
    End;
  End;

  BMP.free;
  JPG.free;

  Texture :=CreateTexture(Width, Height, GL_RGBA, addr(Data[0]));
  result :=TRUE;
end;

{------------------------------------------------------------------}
{  Loads 24 and 32bpp (alpha channel) TGA textures                 }
{------------------------------------------------------------------}
function LoadTGATexture(Filename: String;
        var Texture: GLuint; LoadFromResource : Boolean): Boolean;
var
  TGAHeader : packed record   // Header type for TGA images
    FileType     : Byte;
    ColorMapType : Byte;
    ImageType    : Byte;
    ColorMapSpec : Array[0..4] of Byte;
    OrigX  : Array [0..1] of Byte;
    OrigY  : Array [0..1] of Byte;
    Width  : Array [0..1] of Byte;
    Height : Array [0..1] of Byte;
    BPP    : Byte;
    ImageInfo : Byte;
  end;
  TGAFile   : File;
```

```
  bytesRead : Integer;
  image     : Pointer;    {or PRGBTRIPLE}
  Width, Height : Integer;
  ColorDepth    : Integer;
  ImageSize     : Integer;
  I : Integer;
  Front: ^Byte;
  Back: ^Byte;
  Temp: Byte;

  ResStream : TResourceStream;      // used for loading from resource
begin
  result :=FALSE;
  GetMem(Image, 0);
  if LoadFromResource then // Load from resource
  begin
    try
      ResStream := TResourceStream.Create(hInstance,
        PChar(copy(Filename, 1, Pos('.', Filename)-1)), 'TGA');
      ResStream.ReadBuffer(TGAHeader, SizeOf(TGAHeader));  // FileHeader
      result :=TRUE;
    except on
      EResNotFound do
      begin
        MessageBox(0, PChar('File not found in resource - ' + Filename),
         PChar('TGA Texture'), MB_OK);
        Exit;
      end
      else
      begin
        MessageBox(0, PChar('Unable to read from resource - ' + Filename),
         PChar('BMP Unit'), MB_OK);
        Exit;
      end;
    end;
  end
  else
  begin
    if FileExists(Filename) then
    begin
      AssignFile(TGAFile, Filename);
      Reset(TGAFile, 1);

      // Read in the bitmap file header
      BlockRead(TGAFile, TGAHeader, SizeOf(TGAHeader));
      result :=TRUE;
    end
    else
    begin
      MessageBox(0, PChar('File not found  - ' + Filename),
       PChar('TGA Texture'), MB_OK);
      Exit;
    end;
  end;

  if Result = TRUE then
  begin
    Result :=FALSE;
```

```
// Only support uncompressed images
if (TGAHeader.ImageType <> 2) then  { TGA_RGB }
begin
  Result := False;
  CloseFile(tgaFile);
  MessageBox(0, PChar('Couldn''t load "'+ Filename +
    '". Compressed TGA files not supported.'),
     PChar('TGA File Error'), MB_OK);
  Exit;
end;

// Don't support colormapped files
if TGAHeader.ColorMapType <> 0 then
begin
  Result := False;
  CloseFile(TGAFile);
  MessageBox(0, PChar('Couldn''t load "'+ Filename +
   '". Colormapped TGA files not supported.'),
   PChar('TGA File Error'), MB_OK);
  Exit;
end;

// Get the width, height, and color depth
Width  := TGAHeader.Width[0]  + TGAHeader.Width[1]  * 256;
Height := TGAHeader.Height[0] + TGAHeader.Height[1] * 256;
ColorDepth := TGAHeader.BPP;
ImageSize  := Width*Height*(ColorDepth div 8);

if ColorDepth <> 24 then
begin
  Result := False;
  CloseFile(TGAFile);
  MessageBox(0, PChar('Couldn''t load "'+ Filename +
  '". Only 24 bit TGA files supported.'),
   PChar('TGA File Error'), MB_OK);
  Exit;
end;

GetMem(Image, ImageSize);

if LoadFromResource then // Load from resource
begin
  try
    ResStream.ReadBuffer(Image^, ImageSize);
    ResStream.Free;
  except
    MessageBox(0, PChar('Unable to read from resource - ' + Filename),
     PChar('BMP Unit'), MB_OK);
    Exit;
  end;
end
else        // Read in the image from file
begin
  BlockRead(TGAFile, image^, ImageSize, bytesRead);
  if bytesRead <> ImageSize then
  begin
    Result := False;
```

```
      CloseFile(TGAFile);
      MessageBox(0, PChar('Couldn''t read file "'+ Filename +'".'),
       PChar('TGA File Error'), MB_OK);
      Exit;
    end;
  end;
end;

// TGAs are stored BGR and not RGB, so swap the R and B bytes.
for I :=0 to Width * Height - 1 do
begin
  Front := Pointer(Integer(Image) + I*3);
  Back := Pointer(Integer(Image) + I*3 + 2);
  Temp := Front^;
  Front^ := Back^;
  Back^ := Temp;
end;

Texture :=CreateTexture(Width, Height, GL_RGB, Image);
Result :=TRUE;
FreeMem(Image);
end;

{------------------------------------------------------------------}
{  Determines file type and sends to correct function            }
{------------------------------------------------------------------}
function LoadTexture(Filename: String;
       var Texture : GLuint; LoadFromRes : Boolean) : Boolean;
begin
  if copy(filename, length(filename)-3, 4) = '.bmp' then
    LoadBMPTexture(Filename, Texture, LoadFromRes);
  if copy(filename, length(filename)-3, 4) = '.jpg' then
    LoadJPGTexture(Filename, Texture, LoadFromRes);
  if copy(filename, length(filename)-3, 4) = '.tga' then
    LoadTGATexture(Filename, Texture, LoadFromRes);
end;


end.
```

برنامه فصل :

برای اجرای این برنامه به یک فایل تصویری و یک کنترل تایمر بـا Interval مسـاوی ۶۰ نیـاز می باشد .

```
unit ch19;

interface

uses
```

```
  Windows, Messages, SysUtils, Variants, Classes,
  Graphics, Controls, Forms,
  Dialogs , OpenGL , Textures , SPF, ExtCtrls ;

type
  TForm1 = class(TForm)
    Timer1: TTimer;
    procedure FormResize(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormPaint(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
    procedure FormKeyPress(Sender: TObject; var Key: Char);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

  f_Hdc : LongInt;

implementation

{$R *.dfm}

var
  xrot , yrot : GLfloat;
  xspeed : GLfloat;        // X Rotation Speed
  yspeed : GLfloat;        // Y Rotation Speed
  z : GLfloat = -10.0;  // 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);
  selectedobject : GLuint = 1;      // Which Object To Draw
  // Textures
  MyTextureTex : glUint;

  procedure glBindTexture(target: GLenum; texture: GLuint);
  stdcall; external opengl32;

function InitGL:boolean; // All Setup For OpenGL Goes Here

begin

  glEnable(GL_TEXTURE_2D);        // Enable Texture Mapping
  glShadeModel(GL_SMOOTH);      // Enable Smooth Shading
  glClearColor(0.0, 0.0, 0.0, 0.5);// Black 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);

  LoadTexture('texture.jpg', MyTextureTex, FALSE);    // Load the Texture
```

```
  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
  // Set The Texture Generation Mode For S To Sphere Mapping
  glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);
  // Set The Texture Generation Mode For T To Sphere Mapping
  glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);

  initGL:=true;   // Everything went fine
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 RenderScene():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);
  glEnable(GL_TEXTURE_GEN_S); // Enable Texture Coord Generation For S
  glEnable(GL_TEXTURE_GEN_T); // Enable Texture Coord Generation For T
  // Bind the Texture to the object
  glBindTexture(GL_TEXTURE_2D, MyTextureTex);
  glPushMatrix();
  glRotatef(xrot,1.0,0.0,0.0);
  glRotatef(yrot,0.0,1.0,0.0);

  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 Sphere With A Radius Of 1 And 16
         //Longitude And 16 Latitude Segments
    2:   gluSphere(quadratic,1.3,32,32);
    3:   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;
  end;

  glPopMatrix();
  glDisable(GL_TEXTURE_GEN_S);
  glDisable(GL_TEXTURE_GEN_T);
  // Bind the Texture to the object
  glBindTexture(GL_TEXTURE_2D, MyTextureTex);

  glPushMatrix();
        glTranslatef(0.0, 0.0, -24.0);
        glBegin(GL_QUADS);
           glNormal3f( 0.0, 0.0, 1.0);
           glTexCoord2f(0.0, 0.0); glVertex3f(-13.3, -10.0,  10.0);
           glTexCoord2f(1.0, 0.0); glVertex3f( 13.3, -10.0,  10.0);
           glTexCoord2f(1.0, 1.0); glVertex3f( 13.3,  10.0,  10.0);
           glTexCoord2f(0.0, 1.0); glVertex3f(-13.3,  10.0,  10.0);
        glEnd();
  glPopMatrix();

  xrot := xrot + xspeed;  //increase the first counter
  yrot := yrot + yspeed;  //increase the second counter

  SwapBuffers(f_Hdc);

  RenderScene:=true;      // Everything Went OK
```

```
end;


procedure TForm1.FormResize(Sender: TObject);
begin
   wglMakeCurrent(f_Hdc,hrc); //activate the RC
  if (Height=0) then // Prevent A Divide By Zero If The Window Is Too Small
     Height:=1;      // By Making The Height One
  // Reset The Current Viewport And Perspective Transformation
  glViewport(0, 0, Width, Height);
  glMatrixMode(GL_PROJECTION);       // Select The Projection Matrix
  glLoadIdentity();                  // Reset The Projection Matrix
  // Calculate The Aspect Ratio Of The Window
  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; Dr
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
 wglMakeCurrent(f_Hdc,hrc); //activate the RC
 RenderScene;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
   xspeed := xspeed + 0.5;
   yspeed := yspeed - 0.4;
   z := z + 0.02;
   xrot := xrot + 0.1;
   yrot := yrot + 0.3;
   InvalidateRect(Handle, nil, False);// DrawGLScene;

end;

procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
      selectedobject := selectedobject + 1;
      if (selectedobject>3) then selectedobject := 0;
      InvalidateRect(Handle, nil, False);// DrawGLScene;
end;

end.
```