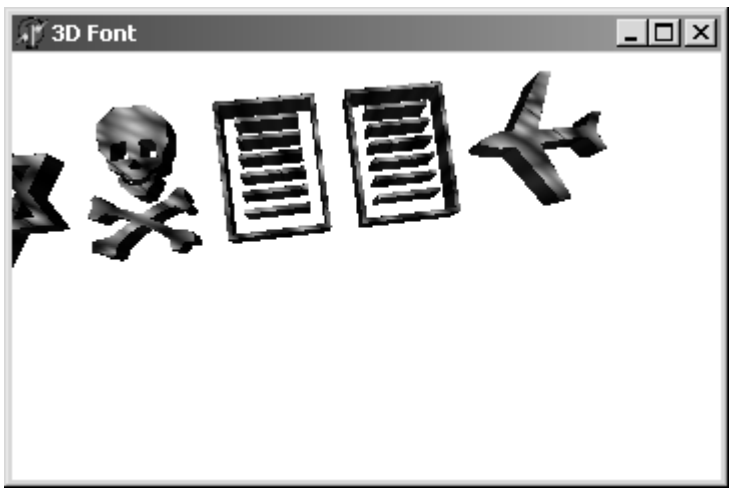
فصل پانزدهم

نمایش متن توسط قلم های سه بعدی



مقدمه :

مطالب این فصل در ادامه و مکمل مطالب فصل پیشین می باشند . در این فصل نحوه استفاده از قلم های OutLine را خواهید آموخت . این قلم ها مقیاس پذیر هستند و می توانند ضخامت نیز داشته باشند . توسط این روش می توان تمام قلم های نصب شده برروی کامپیوتر را به قلم های سه بعدی قابل استفاده در OpenGL تبدیل کرد . در این حالت بردارهای نرمال بصورت خودکار محاسبه می شوند و به راحتی می توان از تکنیک نورپردازی نیز بهره جست .

الگوریتم نمایش متن ۳ بعدی در OpenGL :

۱- ایجاد یک لیست نمایشی
۲- ایجاد و انتخاب یک فونت و سپس مرتبط ساختن آن با DC جاری
۳- فراخوانی تابع wglUseFontOutlines برای ساخت اطلاعات هندسی فونت سه بعدی
۴- فراخوانی glListBase و glCallLists برای نمایش دادن حروف با فونت سه بعدی

شاید بعضی از Active-X های نوشته شده به کمک OpenGL را دیده باشید که کارشان فقط نمایش متن سه بعدی است ، آن هم به قیمت صد دلار !

مروری بر توابع :

تابع به فرمت زبان C

```
typedef struct _POINTFLOAT { // ptf
   FLOAT     x;
   FLOAT     y;
} POINTFLOAT;

typedef struct _GLYPHMETRICSFLOAT { // gmf
   FLOAT     gmfBlackBoxX;
   FLOAT     gmfBlackBoxY;
   POINTFLOAT gmfptGlyphOrigin;
   FLOAT     gmfCellIncX;
   FLOAT     gmfCellIncY;
} GLYPHMETRICSFLOAT;

BOOL wglUseFontOutlines(
  HDC hdc, DWORD first, DWORD count,
  DWORD listBase, FLOAT deviation, FLOAT extrusion,
  int format, LPGLYPHMETRICSFLOAT lpgmf
);
```

توضیح :

این تابع به ازای هر علامت (Glyph) فونت OutLine انتخاب شده از DC جاری برای استفاده در RC جاری ، مجموعه ای از لیست های نمایشی ایجاد می کند . این لیست های نمایشی برای ترسیم کاراکترهای ۳ بعدی قلم های TrueType بکار می روند . هر لیست نمایشی بیانگر یک کاراکتر در مختصات اعشاری است .

آرگومان hDC : بیانگر DC فونت OutLine انتخاب شده است .

first : ابتدای مجموعه علامت ها را که لیست های نمایشی قلم OutLine را تشکیل می دهند ، معین می کند .

Count : تعداد علامت ها رادر مجموعه علامت ها که برای تشکیل لیست های نمایشی قلم OutLine بکار می روند ، مشخص می کند . این تابع به تعداد Count ، لیست نمایشی برای هر علامت در مجموعه علامت ها خلق می کند .

listBase : لیست نمایشی آغاز گر را معین می کند .

deviation : حداکثر انحراف قوسی را از OutLine اصلی معین می کند . هنگامیکه مساوی صفر است ، انحراف قوسی معادل واحد طراحی فونت اصلی می باشد . مقدار این آرگومان باید معادل یا بزرگتر از صفر باشد .

extrusion : ضخامت فونت را در جهت منفی Z معین می کند . مقدار آن باید معادل یا بزرگتر از صفر باشد .

format : یکی از ثوابت WGL_FONT_LINES و یا WGL_FONT_POLYGONS می تواند باشد . در حالت WGL_FONT_LINES لیست های نمایشی فاقد بردار نرمال هستند .

lpgmf : اشاره گری است به آرایه ای به تعداد Count از رکورد یا ساختار GLYPHMETRICSFLOAT که محل و جهت قرارگیری علامت ها را دریافت می کند .

هنگامیکه تابع کارش را با موفقیت به پایان می رساند ، True بر می گرداند .

رکورد GLYPHMETRICSFLOAT :

حاوی اطلاعاتی درباره جهت هر علامت در سلول کاراکتر می باشد .

gmfBlackBoxX ، gmfBlackBoxY : طول و عرض کوچکترین مستطیلی را مشخص می کنند که کاملا علامت را در بر می گیرد .

gmfptGlyphOrigin : مختصات x,y گوشه بالا ، سمت چپ کوچکترین مستطیلی را که کاملا علامت را در بر می گیرد ، مشخص می کند .

gmfCellIncX ، gmfCellIncY : فواصل افقی و عمودی از مبدا سلول کارکتر جاری تا سلول کارکتر بعدی می باشند .

رکورد POINTFLOAT :

حاوی مختصات x,y یک نقطه است .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
void glTexGeni(GLenum coord, GLenum pname, GLint param); void glTexGeniv(GLenum coord, GLenum pname, const GLint *params);	Procedure glTexGeni(coord: GLenum; pname: GLenum; param: GLint); stdcall; external 'OPENGL32.DLL'; Procedure glTexGeniv(coord: GLenum; pname: GLenum; params: PGLint); stdcall; external 'OPENGL32.DLL';

توضیح :

این تابع تولید مختصات بافتی را کنترل کرده و بصورت خودکار آنرا تولید می کند . در این فصل برای اعمال خودکار نگاشت بافتی بر روی متنی ۳ بعدی ، از آن استفاده خواهیم کرد .

در حالت glTexGeni :

آرگومان coord : مختصات بافتی است که می تواند GL_S, GL_T, GL_R, or GL_Q باشد . این ثوابت بیانگر مختص های s ، r ، t و یا q هستند .

pname : نام سمبولیک تولید مختصات بافتی است که باید معادل GL_TEXTURE_GEN_MODE قرار گیرد .

param : پارامتر تولید بافت بوده و یکی از مقادیر زیر را می پذیرد .

GL_OBJECT_LINEAR, GL_EYE_LINEAR, or GL_SPHERE_MAP

در حالت glTexGeniv :

آرگومان coord همانند قبل می باشد .

pname : نام سمبولیک تولید مختصات بافتی است که باید معادل یکی از ثوابت زیر قرار گیرد :

GL_TEXTURE_GEN_MODE, GL_OBJECT_PLANE, or GL_EYE_PLANE

params : اشاره گری است به آرایه پارامترهای تولید بافت . اگر pname معادل GL_TEXTURE_GEN_MODE می باشد ، این آرایه باید حاوی یکی از ثوابت زیر باشد :

GL_OBJECT_LINEAR, GL_EYE_LINEAR, or GL_SPHERE_MAP

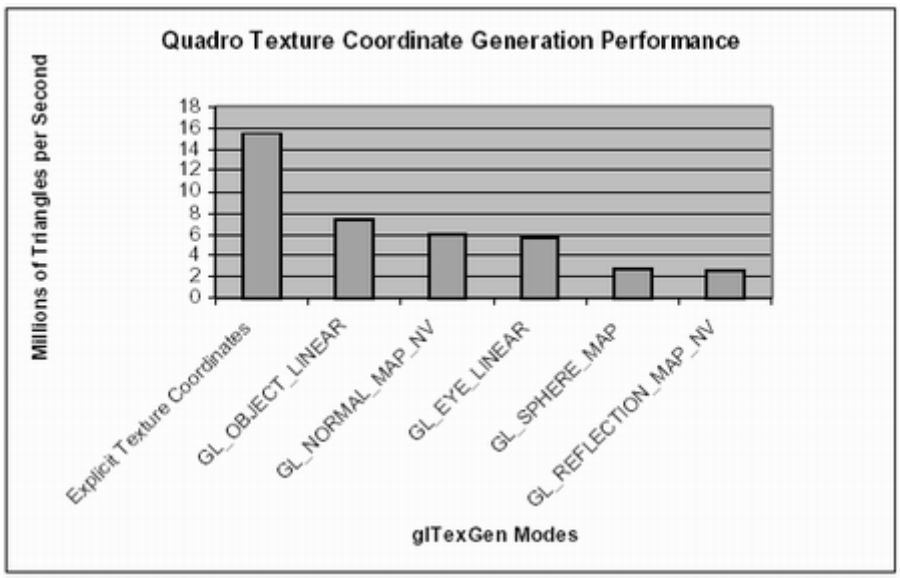
در غیر اینصورت این آرگومان حاوی ضرایب تابع تولید مختصات بافتی معین شده توسط pname می باشد .

ملاحظات :

اگر تصویر بافتی نسبت به شیء متحرک در صحنه ثابت باقی می ماند، از GL_OBJECT_LINEAR استفاده کنید . از GL_EYE_LINEAR هنگامی استفاده می شود که بخواهیم خطوط کانتور پویایی بر

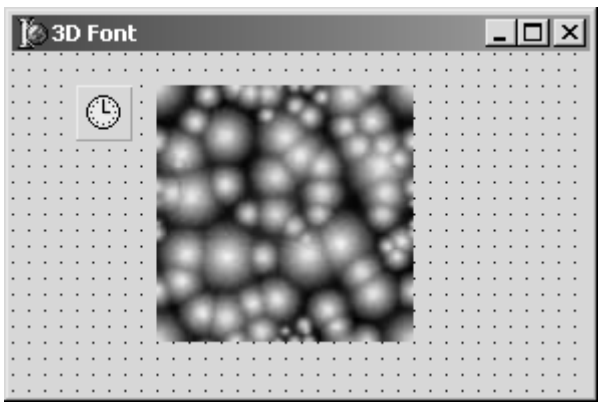
روی اشیاء در حال حرکت ترسیم کنیم . GL_SPHERE_MAP برای نگاشت محیطی بکار می رود . در این حالت شیء منعکس کننده محیط خود خواهد شد .

در نمودار زیر کارآیی پارامترهای مختلف تابع glTexGen را هنگامیکه توسط سخت افزار پشتیبانی می شوند را می بینید .



برنامه فصل :

برای اجرای برنامه این فصل به یک کنترل تایمر و یک کنترل Image که تصویر بیت مپ ۱۲۸×۱۲۸ در ۲۵۶ رنگ درآن بارگذاری شده است ، نیاز می باشد .




```
unit Ch15;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes,
  Graphics, Controls, Forms,
  Dialogs, ExtCtrls , OpenGL, SPF ;
```

```
type
  TForm1 = class(TForm)
    Timer1: TTimer;
    Image1: TImage;
    procedure FormResize(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormPaint(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;

  f_Hdc : LongInt;

implementation
{$R *.dfm}
var
  h_DC : HDC;       // Private GDI Device Context
  base : GLuint;  // Base Display List For The Font Set
  rot : GLfloat;  // Used To Rotate The Text

  texture: array [0..0] of GLuint; // Storage for 1 textures

const glu32 = 'glu32.dll';
const opengl32 = 'opengl32.dll';

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

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

procedure BuildFont;              // Build Our Bitmap Font
var font: HFONT;                 // Windows Font ID
    // Address Buffer For Font Storage
    gmf : array [0..255] of GLYPHMETRICSFLOAT;
begin
  h_DC :=  GetDC(form1.Handle );  // Form1.Canvas.Handle ;
  base := glGenLists(256);      // Storage For 96 Characters
  font := CreateFont(-12,       // Height Of Font
                     0,         // Width Of Font
                     0,         // Angle Of Escapement
                     0,         // Orientation Angle
                     FW_BOLD,  // Font Weight
                     0,         // Italic
                     0,         // Underline
```

```
                    0,          // Strikeout
                    SYMBOL_CHARSET,       // Character Set Identifier
                    OUT_TT_PRECIS,        // Output Precision
                    CLIP_DEFAULT_PRECIS,  // Clipping Precision
                    ANTIALIASED_QUALITY,  // Output Quality
                    FF_DONTCARE or DEFAULT_PITCH, // Family And Pitch
                    'Wingdings');                 // Font Name

  SelectObject(h_DC, font);             // Selects The Font We Want

  wglUseFontOutlines(   h_DC,         // Select The Current DC
                    0,              // Starting Character
                    255,            // Number Of Display Lists To Build
                    base,           // Starting Display Lists
                    0.1,            // Deviation From The True Outlines
                    0.2,            // Font Thickness In The Z Direction
                    WGL_FONT_POLYGONS, // Use Polygons, Not Lines
                    @gmf);                // Address Of Buffer To Recieve Data

end;

procedure KillFont;             // Delete The Font
begin
  glDeleteLists(base, 256);             // Delete All 96 Characters
end;

procedure glPrint(text : pchar);     // Custom GL "Print" Routine
begin
  if (text = '') then             // If There's No Text
        Exit;                           // Do Nothing

  glPushAttrib(GL_LIST_BIT);          // Pushes The Display List Bits
  glListBase(base);                     // Sets The Base Character
  glCallLists(length(text), GL_UNSIGNED_BYTE, text);
  glPopAttrib();                    // Pops The Display List Bits
end;

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

// Load Bitmaps And Convert To Textures
procedure LoadGLTexture( pctPicToLoad : TImage) ;

var
  x , y : LongInt;
  c ,bitmapWidth, bitmapHeight : LongInt ;
  Red , Green , Blue : Integer ;
  //storage for 128*128 bitmap image
  TextureImage: array[0..128,0..128,0..2] of  GLubyte;
begin
   // create the Texture
    pctPicToLoad.AutoSize := True ;
    bitmapHeight := pctPicToLoad.Height ;  // Set the array
```

```
   bitmapWidth := pctPicToLoad.Width ; // size.

 For x := 0 To  bitmapWidth-1  do
    For y := 0 To bitmapHeight-1  do
      begin
        c := ColorToRGB(pctPicToLoad.Canvas.Pixels[x,y]);
        getRGB(c,Red,Green,Blue);
        TextureImage[ x, bitmapHeight - y - 1,0] := red ;
        TextureImage[ x, bitmapHeight - y - 1,1] := green ;
        TextureImage[ x, bitmapHeight - y - 1,2] := blue ;
      end;
      pctPicToLoad.Visible:=false;
  if not Assigned(@TextureImage) then Halt(1);
  // Create MipMapped Texture
  glBindTexture(GL_TEXTURE_2D, texture[0]);
  glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
  glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,
                GL_LINEAR_MIPMAP_NEAREST);
  gluBuild2DMipmaps(GL_TEXTURE_2D, 3, bitmapWidth, bitmapHeight,
                  GL_RGB, GL_UNSIGNED_BYTE, @TextureImage);

  glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR);
  glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR);
  glEnable(GL_TEXTURE_GEN_S);
  glEnable(GL_TEXTURE_GEN_T);
end;

// This Will Be Called Right After The GL Window Is Created
procedure InitGL(Width: GLsizei; Height: GLsizei);
var fWidth, fHeight  : GLfloat;
begin
  BuildFont();                          // Build The Font
  LoadGLTexture(form1.Image1);              // Load Our Texture

  glClearColor(1.0, 1.0, 1.0, 0.0); // Clear The Background Color To White
  glClearDepth(1.0);                // Enables Clearing Of The Depth Buffer
  glDepthFunc(GL_LESS);                     // The Type Of Depth Test To Do
  glEnable(GL_DEPTH_TEST);         // Enables Depth Testing
  glShadeModel(GL_SMOOTH);          // Enables Smooth Color Shading

  glMatrixMode(GL_PROJECTION);                // Select The Projection Matrix
  glLoadIdentity();                          // Reset The Projection Matrix

  fWidth := Width;
  fHeight := Height;
  // Calculate The Aspect Ratio Of The Window
  gluPerspective(45.0, fWidth/fHeight,0.1,100.0);
  glMatrixMode(GL_MODELVIEW);  // Select The Modelview Matrix
  glEnable(GL_LIGHT0);         // Enable Default Light (Quick And Dirty)
  glEnable(GL_LIGHTING);       // Enable Lighting

  glEnable(GL_TEXTURE_2D);
  glBindTexture(GL_TEXTURE_2D, texture[0]);
end;

procedure DrawGLScene();
begin
  // Clear The Screen And The Depth Buffer
```

```
  glClear(GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT);
  glLoadIdentity();                              // Reset The View
  glTranslatef(1.1*cos(rot/16.0),0.8*sin(rot/20.0),-3.0);
  glRotatef(rot,1.0,0.0,0.0);                    // Rotate On The X Axis
  glRotatef(rot*1.2,0.0,1.0,0.0);                // Rotate On The Y Axis
  glRotatef(rot*1.4,0.0,0.0,1.0);                // Rotate On The Z Axis
  glTranslatef(-0.35,-0.35,0.1);                 // Center On X, Y, Z Axis
  glPrint('Q33NY');                              // Draw A Skull And Crossbones Symbol
  rot := rot + 2.6;     // Increase The Rotation Variable
  swapBuffers(f_Hdc);
end;

procedure TForm1.FormResize(Sender: TObject);
var
  fWidth, fHeight: GLfloat;
begin
  wglMakeCurrent(f_Hdc,hrc); //activate the RC
  // Prevent A Divide By Zero If The Window Is Too Small
  if (Height=0) then Height:=1;
  // Reset The Current Viewport And Perspective Transformation
  glViewport(0, 0, Width, Height);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  fWidth := Width;
  fHeight := Height;
  gluPerspective(45.0, fWidth/fHeight, 0.1, 100.0);
  glMatrixMode(GL_MODELVIEW);
  InvalidateRect(Handle, nil, False);// DrawGLScene;  Draw the scene.
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
 KillFont;
 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(width,height);
end;

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

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Application.ProcessMessages; //DoEvents
  InvalidateRect(Handle, nil, False);//   DrawGLScene();
end;

end.
```