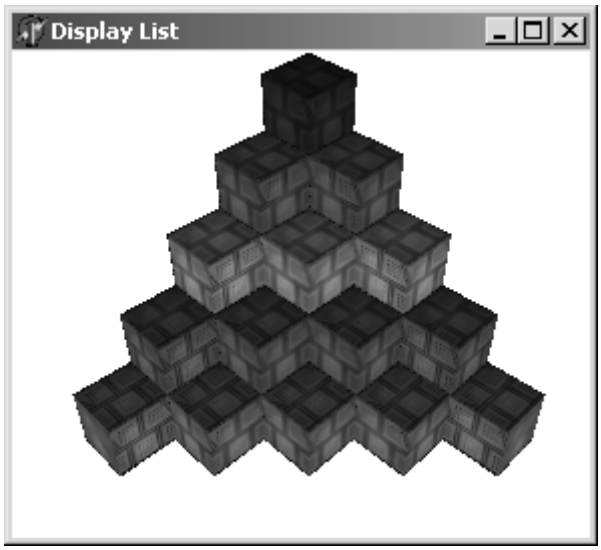
فصل نهم

لیست های نمایشی



مقدمه :

لیست نمایشی مجموعه ای از دستورات OpenGL است که برای اجرا در قسمت های بعدی برنامه ذخیره می شود . هنگامیکه لیست نمایشی اجرا می گردد ، دستورات درون آن به ترتیبی که در حافظه ذخیره شده اند ، اجرا خواهند گردید . مثالی در این زمینه ، ایجاد لیست نمایشی برای شیءایی است که باید بیش از یک بار ترسیم شود .

ایجاد ، اجرا و فلسفه لیست های نمایشی :

دستورات glNewList و glEndList برای آغاز و پایان تعریف لیست نمایشی بکار می روند . بعد از ایجاد لیست نمایشی ، آنرا با فراخوانی تابع glCallList می توان اجرا نمود . فراموش نکنید ، اگر توابعی بجز توابع OpenGL درون لیست نمایشی قرارگیرند ، فقط یکبار ، آن هم در هنگام ایجاد لیست نمایشی ، اجرا خواهند شد و آنها درون لیست نمایشی ذخیره نمی گردند . البته از توابع glu می توان استفاده کرد زیرا آنها از توابع OpenGL مشتق شده اند .

لیست های نمایشی نه تنها سرعت برنامه شما را افزایش می دهند (زیرا در حافظه ذخیره می شوند) ، بلکه تعداد خطوط کد برنامه شما را نیز کاهش خواهند داد . فرض کنید شیءایی را که می خواهید ترسیم کنید ، تولید رئوس آن احتیاج به محاسبات زیادی دارد (مانند برنامه فصل نورپردازی) . با قرار دادن کد تولید مربوط به شیء درون لیست نمایشی ، خلق آن شیء و محاسبات مربوطه تنها یکبار صورت خواهد گرفت و در فراخوانی های بعدی ، شیء از حافظه بارگذاری می گردد .

فرض کنید می خواهیم یک سه چرخه را ترسیم کنیم که سه چرخ آن مانند هم هستند . روش بهینه ترسیم سه چرخه ، ترسیم یک چرخ و قرار دادن آن درون لیست نمایشی و اجرای آن پس از انتقالات لازم برای چرخ های دیگر می باشد .

با استفاده از لیست نمایشی ، شما یک شیء را تنها یکبار خلق می کنید . هنگامی که لیست نمایشی ساخته می شود ، دیگر **نمی توان** آنرا تغییر داد . با توجه به این موضوع اگر می خواهید شیء درون لیست نمایشی را رنگ آمیزی کنید ، فراخوانی تابع مربوطه باید قبل از فراخوانی لیست نمایشی صورت گیرد و اگر فراخوانی دستور رنگ آمیزی درون لیست نمایشی انجام شود دیگرنمی توان رنگ شیء را تغییر داد، زیرا لیست نمایشی خلق شده است وتغییر ناپذیرمی باشد .

برای بهینه سازی سرعت برنامه می توان موارد زیر را درون لیست نمایشی قرار داد :
عملیات ماتریسی ، نورپردازی و عملیات مربوط به مواد و بافت ها .

توابع مورد استفاده درتولید لیست نمایشی :

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
GLuint glGenLists(GLsizei range);	Function glGenLists(range: GLsizei): GLuint; stdcall; external 'OPENGL32.DLL';

توضیح :

هر لیست نمایشی با یک اندیس صحیح معین می شود . تابع glGenLists مجموعه ای از اندیس های لیست های نمایشی خالی پیوسته را ایجاد می کند .

آرگومان Range : تعداد اندیس لیست های نمایشی خالی پیوسته برای ایجاد می باشد .

این تابع در صورت عدم موفقیت صفر بر می گرداند و در صورت موفقیت ، اندیس ابتدای مجموعه اندیس های لیست های نمایشی را ارائه خواهد کرد .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
void glNewList(GLuint list, GLenum mode);	Procedure glNewList(list: GLuint; mode: GLenum); stdcall; external 'OPENGL32.DLL';
void glEndList(void);	Procedure glEndList; stdcall; external 'OPENGL32.DLL';

توضیح :

این توابع یک لیست نمایشی را جایگزین یا خلق می کنند .

آرگومانList : نام لیست نمایشی است (خروجی تابع glGenLists). عددی صحیح ومثبت می باشد .

آرگومان Mode : حالت کامپایل بوده و مقادیر زیر را می پذیرد :

GL_COMPILE : در این حالت دستورات درهمان لحظه ای که درون لیست قرار می گیرند ، اجرا نخواهند شد .

GL_COMPILE_AND_EXECUTE : در این حالت دستورات هنگامیکه درون لیست نمایشی کامپایل می شوند ، اجرا نیز می گردند .

glNewList ابتدای لیست نمایشی را معین می کند و glEndList انتهای آنرا .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
void glCallList(GLuint list);	Procedure glCallList(list: GLuint); stdcall; external 'OPENGL32.DLL';
void glCallLists(GLsizei n, GLenum type, const GLvoid *lists);	Procedure glCallLists(n: GLsizei; atype: GLenum; const lists); stdcall; external 'OPENGL32.DLL';

توضیح :

glCallList لیست نمایشی را اجرا می کند .

آرگومان List : نام عددی لیست نمایشی است که باید اجرا گردد .

glCallLists لیستی از لیست های نمایشی را اجرا می کند .

آرگومان n : تعداد لیست های نمایشی است که باید اجرا شوند .

آرگومان type : نوع داده ای آرگومان lists بوده و می تواند یکی از ثوابت زیر باشد :

GL_BYTE ، GL_UNSIGNED_BYTE، GL_SHORT، GL_UNSIGNED_SHORT، GL_INT، GL_UNSIGNED_INT، GL_FLOAT، GL_2_BYTES، GL_3_BYTES، GL_4_BYTES .

آرگومان lists : آدرس آرایه حاوی نامهای لیست های نمایشی .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
void glListBase(GLuint base);	Procedure glListBase(base: GLuint); stdcall; external 'OPENGL32.DLL';

توضیح :

لیست نمایشی مبنا را برای تابع glCallLists تنظیم می کند .

آرگومان base : یک افست صحیح است که به افست glCallLists اضافه می شود تا نامهای لیست های نمایشی را تولید کند . مقدار آغازین آن صفر است . این تابع آرایه ای از افست ها را ایجاد می کند . نامهای لیست های نمایشی بوسیله اضافه کردن base به هر افست ایجاد می شود .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
void glDeleteLists(GLuint list, Glsizei range);	Procedure glDeleteLists(list: GLuint; range: GLsizei); stdcall; external 'OPENGL32.DLL';

توضیح :

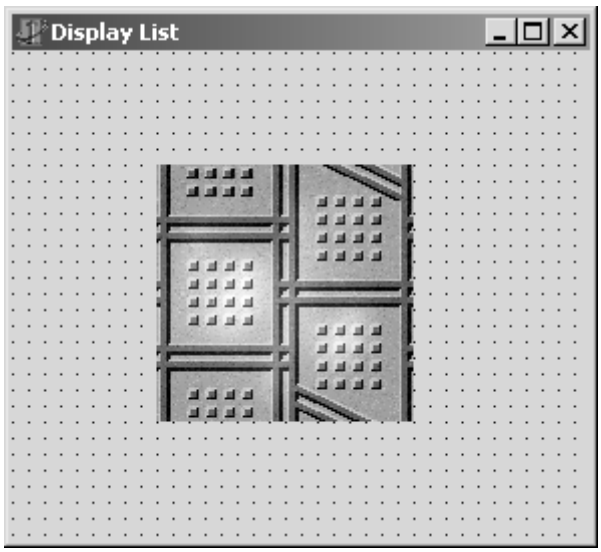
گروه پیوسته لیست های نمایشی را حذف می کند .

List : نام صحیح اولین لیست نمایشی برای حذف می باشد .

Range : تعداد لیست های نمایشی برای حذف کردن است .

برنامه فصل :

برای اجرای این برنامه به یک کنترل Image که تصویری ۱۲۸×۱۲۸ در آن بارگذاری شده است یا خواهد شد! نیاز می باشد (شکل زیر) . در ضمن فشردن کلیدهای left arrow و غیره را هم فراموش نکنید !




```
unit Ch09;

interface

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

type
  TForm1 = class(TForm)
    Image1: TImage;
    procedure FormPaint(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure FormResize(Sender: TObject);
    procedure FormKeyDown(Sender: TObject; var Key: Word;
      Shift: TShiftState);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

  f_Hdc : LongInt;

implementation
```

```
{$R *.DFM}
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;
var
  cube  : GLuint;              // Storage For The Display List
  top   : GLuint;              // Storage For The Second Display List

  xrot : GLfloat;              // Rotates Cube On The X Axis
  yrot : GLfloat;              // Rotates Cube On The Y Axis

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

const boxcol : array [0..4, 0..2] of GLfloat = (
    (1.0,0.0,0.0),
    (1.0,0.5,0.0),
    (1.0,1.0,0.0),
    (0.0,1.0,0.0),
    (0.0,1.0,1.0));
const topcol : array [0..4, 0..2] of GLfloat = (
    (0.5,0.0,0.0),
    (0.5,0.25,0.0),
    (0.5,0.5,0.0),
    (0.0,0.5,0.0),
    (0.0,0.5,0.5));

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;

// Build Cube Display List
procedure BuildList;
begin
  cube := glGenLists(2);
  glNewList(cube,GL_COMPILE);
  glBegin(GL_QUADS);
    // 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);
```

```
    // 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);
    // 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();
  glEndList();

  top := cube + 1;
  glNewList(top,GL_COMPILE);
  glBegin(GL_QUADS);
    // 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);
  glEnd();
  glEndList();
end;

// Load Bitmap And Convert To a Texture
procedure LoadGLTexture (pctPicToLoad : TImage );
var
  x , y : LongInt;
   c , bitmapWidth, bitmapHeight : LongInt ;
   Red , Green , Blue : Integer ;
   texture1: array[0..127,0..127,0..2] of  GLubyte;
begin
   // Load 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);
```

```
        texture1[ x, bitmapHeight - y - 1,0] := Red ;   //GetRed
        texture1[ x, bitmapHeight - y - 1,1] := Green ; //GetGreen
        texture1[ x, bitmapHeight - y - 1,2] := Blue ;  //GetBlue
      end;

      pctPicToLoad.Visible:=false;

  if not Assigned(@texture1) then Halt(1);

  // Create Nearest Filtered Texture
  glGenTextures(1, texture[0]);
  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, @texture1);
end;

// This Will Be Called Right After The GL Window Is Created
procedure InitGL(Width: GLsizei; Height: GLsizei);
var fWidth, fHeight  : GLfloat;
begin
  LoadGLTexture(form1.image1);                  // Load The Texture
  BuildList;                                    // Build The Display List
  glEnable(GL_TEXTURE_2D);                      // Enable Texture Mapping
  // This Will Clear The Background Color To white
  glClearColor(1.0, 1.0, 1.0, 0.0);
  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);
  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);
  glEnable(GL_LIGHT0);
  glEnable(GL_LIGHTING);
  glEnable(GL_COLOR_MATERIAL);
end;

procedure DrawGLScene();
var
  xloop : GLuint;                 // Loop For X Axis
  yloop : GLuint;                 // Loop For Y Axis
  xfloat, yfloat : GLfloat;

  begin
  // Clear The Screen And The Depth Buffer
  glClear(GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT);

  glBindTexture(GL_TEXTURE_2D, texture[0]);
```

```
  for yloop := 1 to 5 do
  begin
    for xloop := 0 to yloop-1 do
    begin
      glLoadIdentity();                                    // Reset The View
      xfloat := xloop;
      yfloat := yloop;
      glTranslatef(1.4+(xfloat*2.8)-(yfloat*1.4),
                ((6.0-yfloat)*2.4)-7.0,-20.0);
      glRotatef(45.0-(2.0*yloop)+xrot,1.0,0.0,0.0);
      glRotatef(45.0+yrot,0.0,1.0,0.0);
      glColor3f(boxcol[yloop-1, 0], boxcol[yloop-1, 1],
              boxcol[yloop-1, 2]);
      glCallList(cube);
      glColor3f(topcol[yloop-1, 0], topcol[yloop-1, 1],
              topcol[yloop-1, 2]);
      glCallList(top);
    end;
  end;
   swapBuffers(f_Hdc);
end;

procedure TForm1.FormPaint(Sender: TObject);
begin
 DrawGLScene();
 wglMakeCurrent(f_Hdc,hrc); //activate the RC
 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.FormDestroy(Sender: TObject);
begin
 CleanUp(f_Hdc);// Clean up and terminate.
end;

procedure TForm1.FormResize(Sender: TObject);
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();
  gluPerspective(45.0, Width/Height, 0.1, 100.0);
  glMatrixMode(GL_MODELVIEW);
  InvalidateRect(Handle, nil, False);// DrawGLScene;  Draw the scene.
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
```

```
    if (key=39) then yrot := yrot - 5;
    if (key=37) then yrot := yrot + 5;
    if (key=38) then xrot := xrot - 5;
    if (key=40) then xrot := xrot + 5;

  Application.ProcessMessages; //DoEvents
  InvalidateRect(Handle, nil, False);//    DrawGLScene();
end;

end.
```