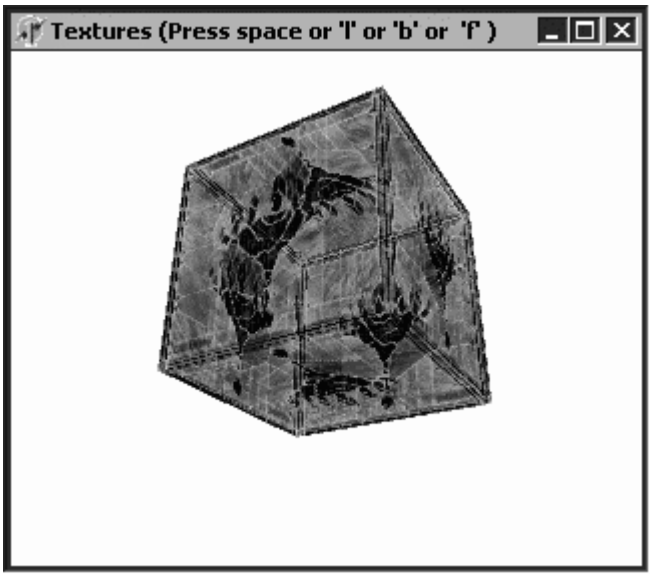
فصل بیست و ششم

موارد تکمیلی نگاشت بافت ها و شفافیت



مقدمه :

اغلب جلوه های ویژه در OpenGL برمبنای اختلاط (Blending) می باشند . برای ترکیب رنگ نقطه ای معین که در حال ترسیم روی صفحه می باشد با نقطه ای موجود ، از اختلاط استفاده می شود . چگونگی ترکیب رنگ ها بر مبنای تنظیم جزء آلفای رنگ و/یا توابع اختلاط صورت می گیرد . آلفا ، چهارمین جزء رنگ است که معمولا در توابع OpenGL در انتهای آرگومانها ذکر می گردد . از ثابت GL_RGB برای تعیین سه جزء رنگ استفاده می شود و با بکارگیری GL_RGBA می توان از آلفا نیز استفاده کرد . بعلاوه از تابع glColor4f() می توان بجای glColor3f() کمک گرفت . به بیانی ساده تر اگر مقدار آلفا صفر باشد ، بدین معنا است که ماده کاملا شفاف است و اگر مساوی یک باشد ، یعنی ماده کاملا مات و کدر خواهد بود .

اختلاط و شفافیت در OpenGL :

فعال سازی اختلاط مانند فعال سازی سایر موارد در OpenGL می باشد . سپس تابع مربوطه را تنظیم کرده و عمق بافر را از کار می اندازیم ، زیرا در هنگام ترسیم اشیاء شفاف مایل هستیم که اجسام پشت آنها نیز مشاهده گردند (ترسیم شوند) . هر چند این روش اختلاط صحیحی را بدست نمی دهد ، اما اغلب اوقات خوب کار می کند !

روش صحیح بدین صورت است که تمام چند ضلعی های شفاف (با مقدار آلفای کمتر از یک) ، پس از ترسیم تمام صحنه ، به صورتیکه دورترین آنها ابتدا رسم شود ، ترسیم گردند . این بدان علت است که اختلاط دو چند ضلعی (۱و۲) به ترتیبی متفاوت ، نتایج مختلفی را هم ارائه خواهند داد . برای مثال فرض کنید که چند ضلعی ۱ به بیننده نزدیک تر است . روش صحیح اختلاط ، رسم چند ضلعی ۲ در ابتدا و سپس ۱ می باشد . در واقعیت تمام نورهایی که از پشت این دو چند ضلعی به چشم بیننده می رسند (که شفاف نیز هستند) ابتدا چند ضلعی ۲ را قطع کرده و سپس چند ضلعی ۱ را . بنابراین باید چند ضلعی های شفاف در عمق مرتب شده ، پس از ترسیم کل صحنه ، با فعال سازی عمق بافر ، ترسیم گردند ، در غیر اینصورت نتایجی نادرست بدست خواهد آمد .

برنامه فصل :

```
unit ch26;
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 FormKeyDown(Sender: TObject; var Key: Word;
      Shift: TShiftState);
  private
    { Private declarations }
   procedure Idle(Sender: TObject; var Done: Boolean);
  public
    { Public declarations }
  end;
```

```
var
  Form1: TForm1;
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;

function gluBuild2DMipmaps(target: GLenum; components,
        width, height: GLint; format, atype: GLenum;
        Data: Pointer): GLint; stdcall; external glu32;

var
  light: boolean;               // Lighting ON/OFF
  lp: boolean;                  // L Pressed?
  fp: boolean;                  // F Pressed?
  blend: boolean;                // Blending OFF/ON?
  bp: boolean;                        // B Pressed?

  xrot: GLfloat;    // X Rotation
  yrot: GLfloat;    // Y Rotation
  zrot : GLfloat;   // Z Rotation
  xspeed: GLfloat;  // X Rotation Speed
  yspeed: GLfloat;  // Y Rotation Speed

  z: GLfloat = -5.0;  // Depth Into The Screen

  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 );

  filter: GLuint = 0;  // Which Filter To Use
  texture: array [0..2] of GLuint;    // Storage for 3 textures

  f_Hdc : longInt;

  // The array for the points on the grid of our "wave"
  points : array [0..44, 0..44, 0..2] of GLfloat;
  wiggle_count : integer = 0;

  s2 : boolean = false; // another tutorial !

{------------------------------------------------------------------}
// Load Bitmaps And Convert To Textures
{------------------------------------------------------------------}
procedure LoadGLTextures(Width, Height : Word; pData : Pointer) ;
begin
        if not Assigned(pData) then   Halt(1);

        // Create Nearest Filtered Texture
        glGenTextures(3, texture[0]);
        glBindTexture(GL_TEXTURE_2D, texture[0]);
        glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_NEAREST);
        glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_NEAREST);
        glTexImage2D(GL_TEXTURE_2D, 0, 3, Width, Height,
```

```
                0, GL_RGB, GL_UNSIGNED_BYTE, pData);

        // Create Linear Filtered Texture
        glBindTexture(GL_TEXTURE_2D, texture[1]);
        glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR);
        glTexImage2D(GL_TEXTURE_2D, 0, 3, Width, Height,
                0, GL_RGB, GL_UNSIGNED_BYTE, pData);

        // Create MipMapped Texture
        glBindTexture(GL_TEXTURE_2D, texture[2]);
        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, Width, Height,
                   GL_RGB, GL_UNSIGNED_BYTE, pData);
end;

{------------------------------------------------------------------}
{  Load 24-bits-BMP images                                         }
{------------------------------------------------------------------}
function LoadBMPImage(Filename: String;
                  var Width, Height : Integer) : pointer;
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;

  pData : Pointer;

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 '), MB_OK);
      LoadBMPImage := nil;
     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 '), MB_OK);
      LoadBMPImage := nil;
      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 '), MB_OK);
       LoadBMPImage := nil;
      Exit;
    end;
    CloseHandle(BitmapFile);

  // 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;
  result := pData;
end;

// This Will Be Called Right After The GL Window Is Created
procedure InitGL(Width: GLsizei; Height: GLsizei);
var fWidth, fHeight: GLfloat;
var
    h,w : integer;
    pData : pointer;
    float_x, float_y : GLint; // Delphi needs integers for loops
begin
        pData := LoadBMPImage ('GLASS.bmp', w , h );
        LoadGLTextures(w,h,pData);  // Load The Texture(s)

           glEnable(GL_TEXTURE_2D);    // Enable Texture Mapping
        // This Will Clear The Background Color To black
           glClearColor(0.0, 0.0, 0.0, 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);

          glLightfv(GL_LIGHT1, GL_AMBIENT, @LightAmbient);
          glLightfv(GL_LIGHT1, GL_DIFFUSE, @LightDiffuse);
          glLightfv(GL_LIGHT1, GL_POSITION,@LightPosition);
          glEnable(GL_LIGHT1);



  glMatrixMode(GL_MODELVIEW);

  for float_x := 0 to 44 do
  begin
    for float_y := 0 to 44 do
    begin
      points[float_x, float_y, 0] := float_x/5 - 4.4;
      points[float_x, float_y, 1] := float_y/5 - 4.4;
      points[float_x, float_y, 2] := sin(((float_x * 8)/360) * PI * 2);
    end;
  end;

end;

procedure ReSizeGLScene(Width: GLsizei; Height: GLsizei);
var
  fWidth, fHeight: GLfloat;
begin
  if (Height=0) // Prevent A Divide By Zero If The Window Is Too Small
    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);
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(0.0,0.0,z);

          glRotatef(xrot,1.0,0.0,0.0);
          glRotatef(yrot,0.0,1.0,0.0);

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

```
      glPolygonMode( GL_BACK, GL_FILL );
      glPolygonMode( GL_FRONT, GL_FILL );

        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();

      SwapBuffers(f_Hdc);
end;

procedure DrawGLScene2();
var x, y : GLint;
   float_x, float_y, float_xb, float_yb : GLfloat;
begin
  glClear(GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT);
  glLoadIdentity();
  glTranslatef(0.0,0.0,-12.0);

  glRotatef(xrot,1.0,0.0,0.0);
  glRotatef(yrot,0.0,1.0,0.0);
  glRotatef(zrot,0.0,0.0,1.0);
```

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

  glPolygonMode( GL_BACK, GL_FILL );
  glPolygonMode( GL_FRONT, GL_LINE );

  glBegin( GL_QUADS );
{
 The next section should be done for x := 0 to 44 and y := 0 to 44,
 but that produces spectacular, unwanted results. Try it, and you
 will see that there's something going wrong. It appears to me that
 this is cause by a difference in the sin() function between Delphi
 and Visual C++.

 The problem is created in the final part of the InitGL() function
 above.
}
  for x := 0 to 42 do
  begin
    for y := 0 to 42 do
    begin
      float_x := x/44;
      float_y := y/44;
      float_xb := (x+1)/44;
      float_yb := (y+1)/44;

      glTexCoord2f( float_x, float_y);
      glVertex3f( points[x,y,0], points[x,y,1], points[x,y,2]);

      glTexCoord2f( float_x, float_yb );
      glVertex3f( points[x,y+1,0],
                points[x,y+1,1], points[x,y+1,2]);

      glTexCoord2f( float_xb, float_yb );
      glVertex3f( points[x+1,y+1,0],
                points[x+1,y+1,1], points[x+1,y+1,2] );

      glTexCoord2f( float_xb, float_y );
      glVertex3f( points[x+1,y,0],
                points[x+1,y,1], points[x+1,y,2] );
    end;
  end;
  glEnd();

  if (wiggle_count = 2) then
  begin
    for y := 0 to 44 do
    begin
      points[44,y,2] := points[0,y,2];
    end;
    for x := 0 to 44 do
    begin
      for y := 0 to 44 do
      begin
        points[x,y,2] := points[x+1,y,2];
      end;
    end;
    wiggle_count := 0;
  end;
```

```
  wiggle_count := wiggle_count + 1;
  zrot := zrot + 0.4;
  SwapBuffers(f_Hdc);
end;

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

procedure TForm1.FormResize(Sender: TObject);
begin
   wglMakeCurrent(f_Hdc,hrc); //activate the RC
    ReSizeGLScene(width,height);
    InvalidateRect(Handle, nil, False);// 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
 if s2 = false then
   DrawGLScene() // Draw the scene.
 else
   DrawGLScene2();// Draw the scene.
end;

procedure TForm1.Idle(Sender: TObject; var Done: Boolean);
begin
  z := z - 0.02;
  if z <-7 then z := -5;

  xspeed  :=  0.5;
  yspeed := 0.5;

  xrot := xrot + xspeed;
  yrot := yrot + yspeed;

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

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
  application.ProcessMessages ;
  if (key = VK_ESCAPE) then SendMessage(handle,WM_CLOSE,0,0);
  if key = VK_SPACE then  s2 := not(s2) ;
  if (key =ord('L') ) then
  begin
    lp := True;
```

```
    light := not light;
    if (not light)
      then glDisable(GL_LIGHTING)
    else glEnable(GL_LIGHTING);
  end;
  if (key = ord('F')) then
  begin
    fp := True;
    Filter := Filter + 1;
    if (Filter > 2) then Filter := 0;
  end;

   if (key = ord('B') ) then
   begin
     bp := TRUE;
     blend := not(blend);
     if (blend) then
     begin
       glEnable(GL_BLEND);       // Turn Blending On
       // Set The Blending Function For Translucency
       glBlendFunc(GL_SRC_ALPHA,GL_ONE);
       glDisable(GL_DEPTH_TEST);
     end
     else
     begin
       glDisable(GL_BLEND);          // Turn Blending Off
       glEnable(GL_DEPTH_TEST);             // Turn Depth Buffer Writes On
     end;
   end;

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

end;

end.
```