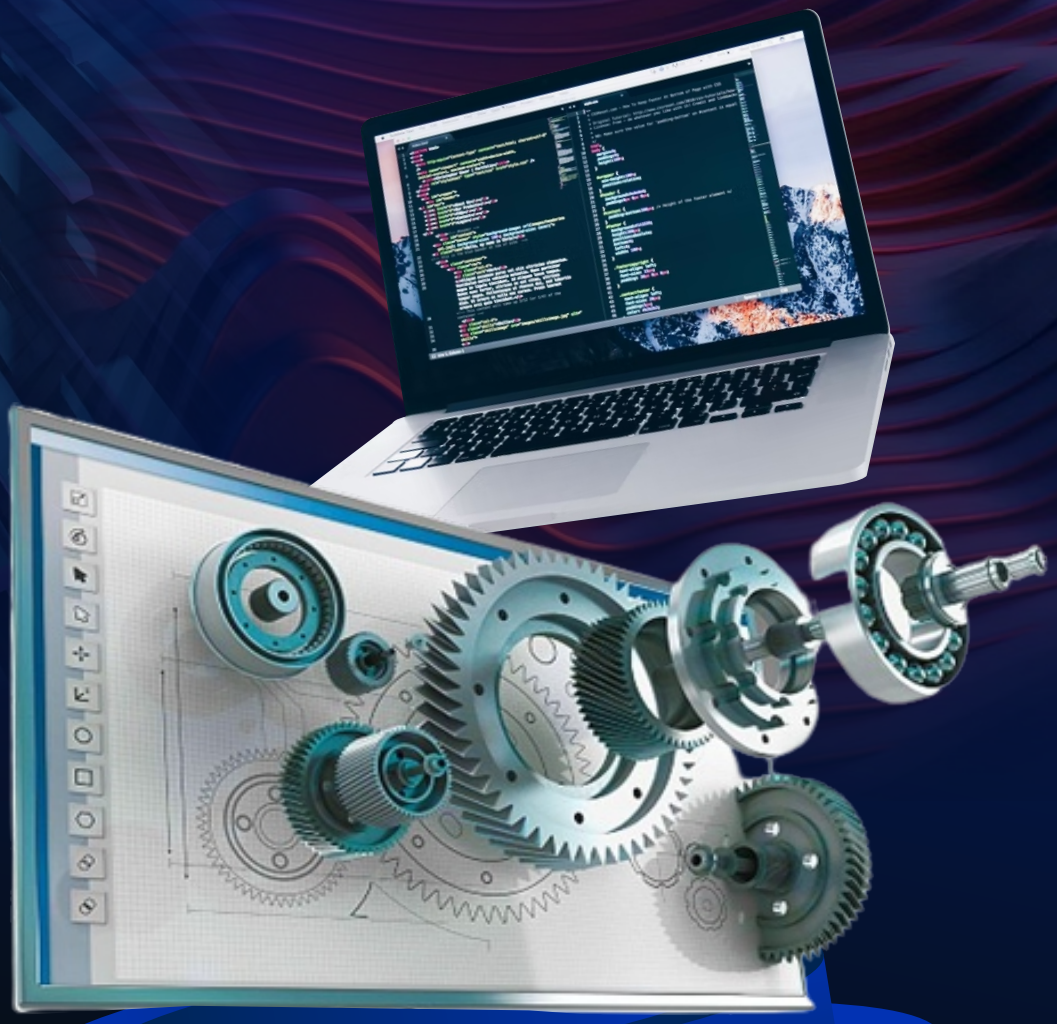


GRAFIKA KOMPUTER 3D

Dr.Silvia Ratna, S.kom, M.Kom

Dr. Silvia Ratna, S.Kom., M.Kom.

GRAFIKA KOMPUTER 3D

Fakultas Ekonomi dan Bisnis Islam

IAIN Lhokseumawe

GRAFIKA KOMPUTER 3D

Penulis:
Dr. Silvia Ratna, S.Kom., M.Kom.

ISBN:
978-623-88237-0

Desain Cover:
Ali Muhajir

Layout
Okva Riza

Cetakan Pertama:
Agustus 2022

1.hal.266. :16,5x23 cm
I. Judul

PENERBIT:
FAKULTAS EKONOMI DAN BISNIS ISLAM – IAIN LHOKSEUMAWE
Anggota Afiliasi Penerbit Perguruan Tinggi Indonesia (APPTI)
Nomor: 005.152.1.3.2022
Jl. Medan-Banda Aceh Km. 275, No. 1, Bukit Rata, Alue Awe
Lhokseumawe 24352, Aceh, Indonesia
Email: penerbitfebi@iainlhokseumawe.ac.id
https://febi.iainlhokseumawe.ac.id/penerbit

KATA PENGANTAR

Grafika komputer 3D adalah representasi data geometrik 3D sebagai hasil pemrosesan dan pemberian efek cahaya terhadap grafika komputer 2D. Hasil ini kadang kala ditampilkan dalam waktu nyata (real time) untuk keperluan simulasi. Secara umum prinsip yang dipakai mirip dengan grafika komputer 2D, dalam hal: penggunaan algoritma, grafika vektor, model rangka (wire frame model), dan grafika rasternya.

Grafika komputer 3D sering disebut sebagai model 3D. Namun, model 3D lebih menekankan pada representasi matematis untuk objek 3D. Data matematis ini belum bisa dikatakan sebagai gambar grafis sampai ditampilkan secara visual pada layar komputer atau printer. Proses penampilan suatu model matematis ke bentuk citra 2D biasanya dikenal dengan proses perenderan 3D.

Proses pembuatan grafik komputer 3D dapat dibagi ke dalam tiga fase, yaitu 3D modeling yang mendeskripsikan bentuk dari sebuah objek, layout dan animasi yang mendeskripsikan gerakan dan tata letak sebuah objek, dan 3D rendering yang memproduksi image dari objek tersebut.

Adapun materi yang dibahas meliputi : Garis, Solar System 3D, Font, dan Fish dan Aqua.

Penulis

DAFTAR ISI

Bab 1

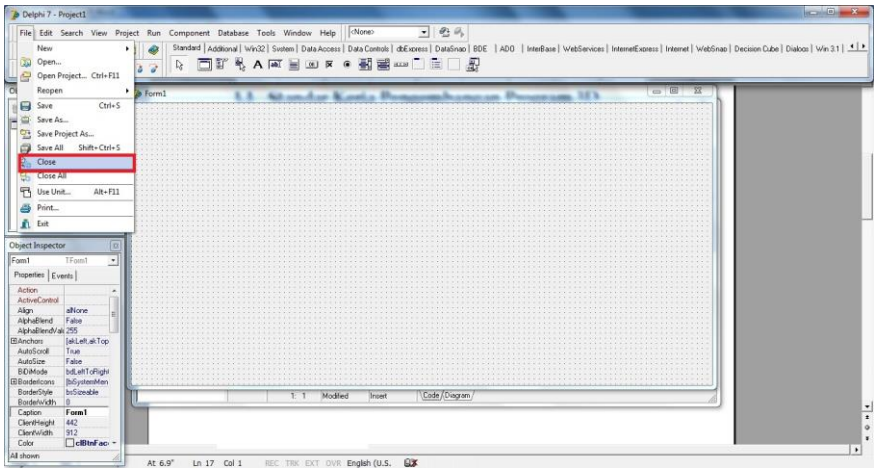
PENDAHULUAN

1.1 Standar Kerja Pengembangan Program 3D

Standar kerja pengembangan program 3D di dalam buku ini meliputi dua teknik pendekatan : (1) pembangunan source file, dan (2) pembangunan proyek program utama.

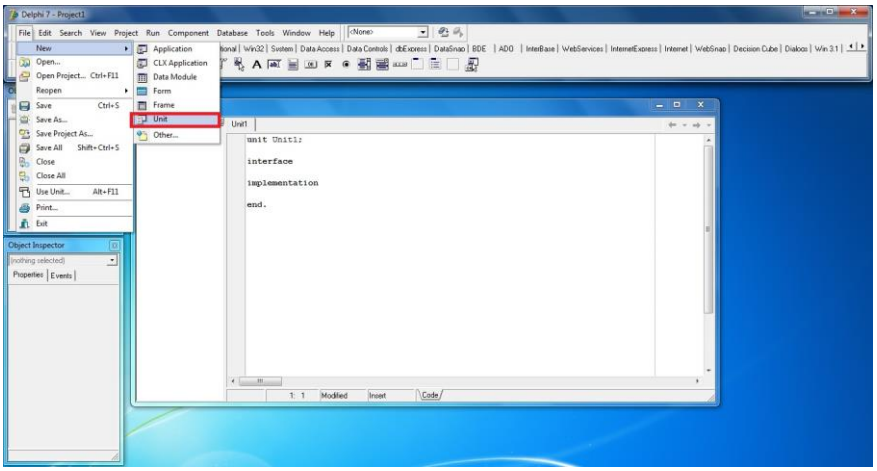
Sebelum masuk kepada penjelasan akan pendekatan tersebut, maka setelah muncul IDE utama Delphi, lakukan langkah berikut :

File dan Close



1.2 Source File

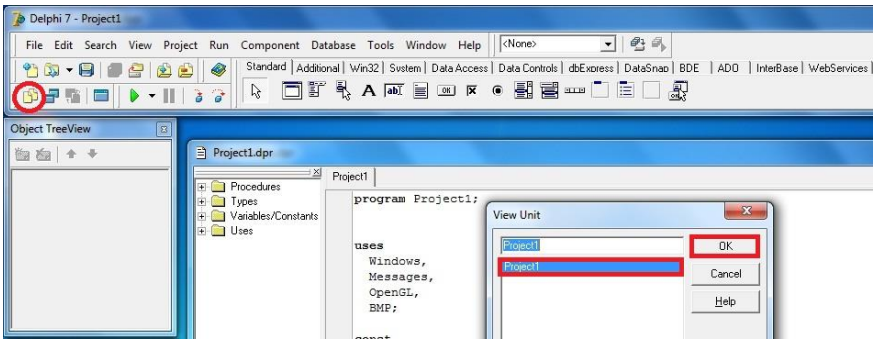
Untuk membangun Delphi Source File, maka sorot ke menu File, lalu sorot New dan klik Unit.



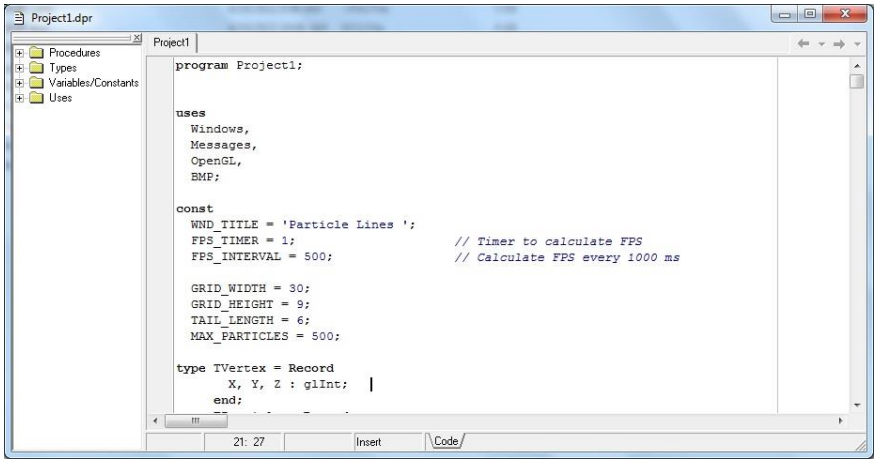
Setelah menuliskan koding, lalu Save.

1.3 Proyek Utama

Pada pembangunan proyek utama, lakukan File dan Close, lalu klik icon View Unit  pada pojok kiri panel komponen, dan pilih project1, lalu klik tombol ok.



Setelah muncul halaman koding, selanjutnya tuliskan listing program, dan klik Save All pada menu utama File.



Bab 2

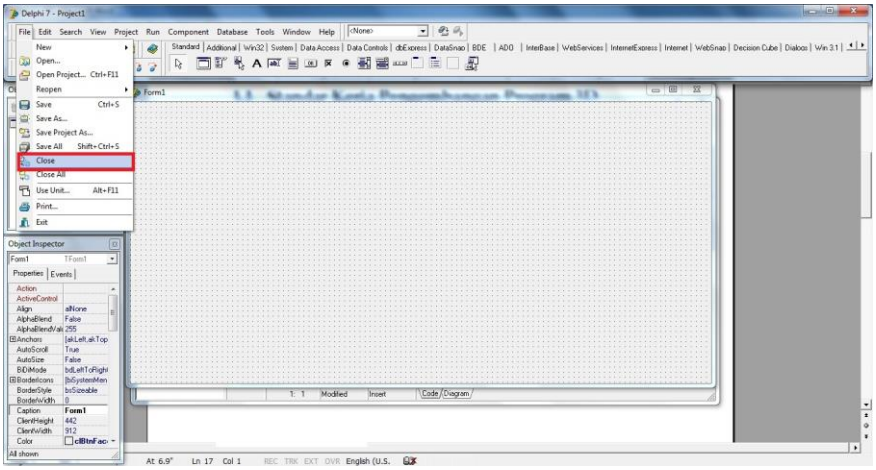
GARIS 3D VISUAL

2.1 Persiapan Awal

Untuk pembangunan program garis dengan visual 3D, terlebih dahulu disiapkan dua buah file template gambar berformat bmp. Kegunaan dari gambar-gambar tersebut hanya untuk memberi kesan atau efek visual yang lebih nyata dalam lingkungan 3D.

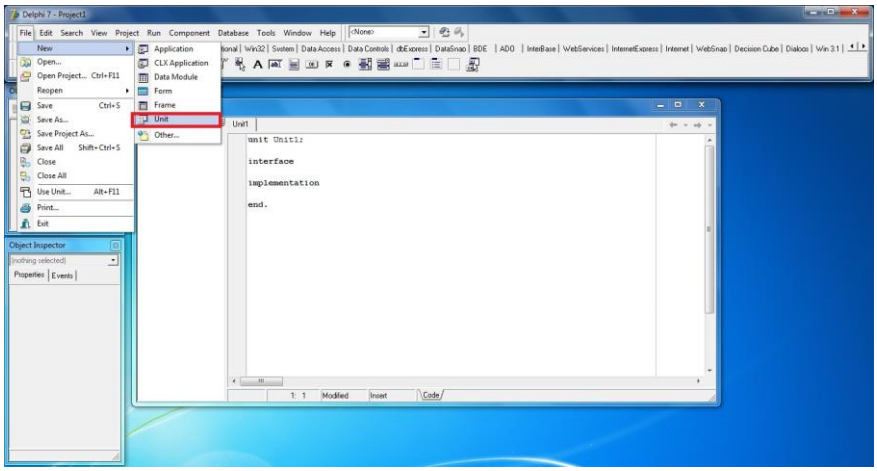
Sebelum masuk kepada penulisan koding program source file, maka setelah muncul IDE utama Delphi, lakukan langkah berikut :

File dan Close



2.2 Source File

Untuk membangun Delphi Source File, maka sorot ke menu File, lalu sorot New, dan klik Unit.



Setelah muncul halaman koding, lalu tuliskan koding berikut :

```
unit BMP;

interface

uses
  Windows, OpenGL;

function LoadTexture(Filename: String; var Texture: GLuint): 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;
```

```
{------------------------------------------------------------------}
{  Swap bitmap format from BGR to RGB }
{------------------------------------------------------------------}
procedure SwapRGB(data : Pointer; Size : Integer);
asm
  mov ebx, eax
  mov ecx, size

@@loop :
  mov al,[ebx+0]
  mov ah,[ebx+2]
  mov [ebx+2],al
  mov [ebx+0],ah
  add ebx,3
  dec ecx
  jnz @@loop
end;

{------------------------------------------------------------------}
{  Load BMP file }
{------------------------------------------------------------------}
procedure LoadBitmap(Filename: String; out Width: Cardinal; out Height: Cardinal;
out pData: Pointer);
var
  FileHeader: TBITMAPFILEHEADER;
  InfoHeader: TBITMAPINFOHEADER;
  Palette: array of RGBQUAD;
  BitmapFile: THandle;
  BitmapLength: LongWord;
  PaletteLength: LongWord;
  ReadBytes: LongWord;
begin
  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);

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

```
  SwapRGB(pData, Width*Height);
end;

{------------------------------------------------------------------}
{  Load BMP textures }
{------------------------------------------------------------------}
function LoadTexture(Filename: String; var Texture: GLuint): Boolean;
var
  pData: Pointer;
  Width: LongWord;
  Height: LongWord;
begin
  pData :=nil;
  LoadBitmap(Filename, Width, Height, pData);

  if (Assigned(pData)) then
    Result := True
  else
  begin
    Result := False;
    MessageBox(0, PChar('Unable to load ' + filename), 'Loading Textures', MB_OK);
    Halt(1);
  end;

  glGenTextures(1, Texture);
  glBindTexture(GL_TEXTURE_2D, Texture);
  glTexEnvi(GL_TEXTURE_ENV,       GL_TEXTURE_ENV_MODE,       GL_MODULATE);
{Texture blends with object background}
//   glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);  {Texture
does NOT blend with object background}

  { Select a filtering type. BiLinear filtering produces very good results with little
performance impact
    GL_NEAREST            - Basic texture (grainy looking texture)
```

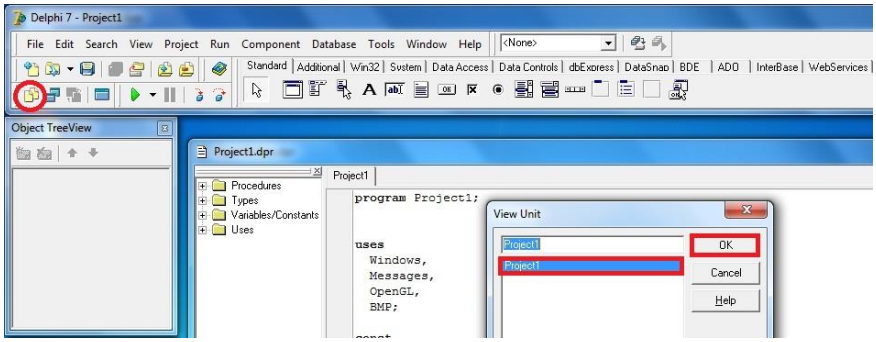
GL_LINEAR - BiLinear filtering
GL_LINEAR_MIPMAP_NEAREST - Basic mipmapped texture
GL_LINEAR_MIPMAP_LINEAR - BiLinear Mipmapped texture }
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR); { only first two can be used }
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR); { all of the above can be used }

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
end;

end.
Lalu Save dengan nama file BMP.

2.3 Proyek Utama

Pada pembangunan proyek utama, lakukan File dan Close, lalu klik icon View Unit pada pojok kiri panel komponen, dan pilih project1, lalu klik tombol ok.



Setelah muncul halaman koding, selanjutnya tuliskan koding berikut :
program Project1;

```
uses
  Windows,
  Messages,
  OpenGL,
  BMP;

const
  WND_TITLE = 'Particle Lines ';
  FPS_TIMER = 1;                  // Timer to calculate FPS
  FPS_INTERVAL = 500;             // Calculate FPS every 1000 ms

  GRID_WIDTH = 30;
  GRID_HEIGHT = 9;
  TAIL_LENGTH = 6;
  MAX_PARTICLES = 500;

type TVertex = Record
      X, Y, Z : glInt;
    end;
    TParticle = Record
      NextVertex : Integer;       // vertex its traveling to
      DestValue  : Integer;       // speed*elapsedtime value when it gets there.
      Visited : Array[1..TAIL_LENGTH] of Integer;  // last 5 visited vertices
      Speed   : Single;           // speed of the particle
      Step    : glFloat;
    end;

var
  h_Wnd  : HWND;                  // Global window handle
  h_DC   : HDC;                   // Global device context
  h_RC   : HGLRC;                 // OpenGL rendering context
  keys : Array[0..255] of Boolean;   // Holds keystrokes
  FPSCount : Integer = 0;         // Counter for FPS
  ElapsedTime : Integer;          // Elapsed time between frames
```

```
  // Textures
  particleTex : glUint;
  backgroundTex : glUint;

  // User variables
  Particles : Integer = 150;
  Grid      : Array[0..GRID_WIDTH*GRID_WIDTH*GRID_HEIGHT] of TVertex;  // grid
of 100x100x7
  Particle : Array[1..MAX_PARTICLES] of TParticle;
  ShowBackground : Boolean = TRUE;

{$R *.RES}

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

{------------------------------------------------------------------}
{  Function to convert int to string.}
{------------------------------------------------------------------}
function IntToStr(Num : Integer) : String;  // using SysUtils increase file size by
100K
begin
  Str(Num, result);
end;

//--- Check if the particle has reached its destination. If it has, select a new
destination
procedure UpdateParticles;
var I, P : Integer;
begin
  for P :=1 to PARTICLES do
  with Particle[P] do
  begin
```

```
  if ElapsedTime >= DestValue then        // Check if the particle has reached its destination
  begin
   for I :=TAIL_LENGTH downto 2 do     // store current location in visited list and thift the other up
    Visited[I] :=Visited[I-1];
   Visited[1] :=NextVertex;

   // select a new random direction
   repeat
    I :=Random(6);
    case I of
     0 : if Visited[1] MOD GRID_WIDTH > 0 then NextVertex := Visited[1] - 1; // left
     1 : if Visited[1] MOD GRID_WIDTH < GRID_WIDTH-1 then NextVertex := Visited[1] + 1;     // right
     2 : if Visited[1] MOD (GRID_WIDTH*GRID_WIDTH) > GRID_WIDTH then NextVertex := Visited[1] - GRID_WIDTH;     // back
     3 : if Visited[1] MOD (GRID_WIDTH*GRID_WIDTH) < GRID_WIDTH*GRID_WIDTH-GRID_WIDTH then NextVertex := Visited[1] + GRID_WIDTH;  // forward
     4 : if Visited[1] DIV (GRID_WIDTH*GRID_WIDTH) > 0 then NextVertex := Visited[1] - GRID_WIDTH*GRID_WIDTH;          // down
     5 : if Visited[1] DIV (GRID_WIDTH*GRID_WIDTH) < GRID_HEIGHT-1 then NextVertex := Visited[1] + GRID_WIDTH*GRID_WIDTH;  // up
    end;
    if NextVertex <> Visited[1] then    // not equal current vertex
     if NextVertex <> Visited[2] then  // not equal previous vertex
      I :=-1;
   until I = -1;

   // calculate when the next endpoint will be reached
   DestValue :=Round(((ElapsedTime DIV (Round(10/Speed)))*10+10)/Speed);
  end;
```

```
  end;
end;

//--- Draws the tail of the particle. Color fades away. Fast particles are red
procedure drawParticleTail;
var I, P : Integer;
   C : glFloat;
begin
 for P :=1 to PARTICLES do
 with Particle[P] do
 begin

   // Draw the path of the particle and fade the color
   Step := 1-(DestValue - ElapsedTime)/10*Speed;
   glBegin(GL_LINE_STRIP);
    glColor3f(1, 1, 1);
    glVertex3f((Grid[NextVertex].X-Grid[Visited[1]].X)*Step + Grid[Visited[1]].X,
            (Grid[NextVertex].Y-Grid[Visited[1]].Y)*Step + Grid[Visited[1]].Y,
            (Grid[NextVertex].Z-Grid[Visited[1]].Z)*Step + Grid[Visited[1]].Z);
    for I :=1 to TAIL_LENGTH-1 do
    begin
     C :=(TAIL_LENGTH-I)/TAIL_LENGTH - Step/TAIL_LENGTH;
     if Speed < 0.05 then
      glColor3f(C, C, 0.5+C)
     else
      glColor3f(0.5+C, C, C);    // faster particles are red
     glVertex3iv(@Grid[Visited[I]]);
    end;
    glColor3f(0, 0, 0.1);
    glVertex3f((Grid[Visited[TAIL_LENGTH-1]].X-
Grid[Visited[TAIL_LENGTH]].X)*Step + Grid[Visited[TAIL_LENGTH]].X,
            (Grid[Visited[TAIL_LENGTH-1]].Y-Grid[Visited[TAIL_LENGTH]].Y)*Step    +
Grid[Visited[TAIL_LENGTH]].Y,
```

```
            (Grid[Visited[TAIL_LENGTH-1]].Z-Grid[Visited[TAIL_LENGTH]].Z)*Step    +
Grid[Visited[TAIL_LENGTH]].Z);
   glEnd();
  end;
end;

//--- Draws the head of the particle. Fast particles are red
procedure drawParticleHead();
var P : Integer;
   X, Y, Z : glFloat;
begin
 glBindTexture(GL_TEXTURE_2D, particleTex);
 glEnable(GL_TEXTURE_2D);
 glEnable(GL_BLEND);
 for P :=1 to Particles do
 with Particle[P] do
 begin
   X :=(Grid[NextVertex].X-Grid[Visited[1]].X)*Step + Grid[Visited[1]].X;
   Y :=(Grid[NextVertex].Y-Grid[Visited[1]].Y)*Step + Grid[Visited[1]].Y;
   Z :=(Grid[NextVertex].Z-Grid[Visited[1]].Z)*Step + Grid[Visited[1]].Z;
   if Particle[P].Speed < 0.05 then
    glColor3f(1, 1, 1)
   else
    glColor3f(1, 0.7, 0.7);     // faster particles are red
   glBegin(GL_QUADS);
    glTexCoord2f(0.0, 0.0); glVertex3f(X-0.5, Y-0.5, Z);
    glTexCoord2f(1.0, 0.0); glVertex3f(X+0.5, Y-0.5, Z);
    glTexCoord2f(1.0, 1.0); glVertex3f(X+0.5, Y+0.5, Z);
    glTexCoord2f(0.0, 1.0); glVertex3f(X-0.5, Y+0.5, Z);
   glEnd();
 end;
 glDisable(GL_BLEND);
 glDisable(GL_TEXTURE_2D);
end;
```

```
{------------------------------------------------------------------}
{  Function to draw the actual scene }
{------------------------------------------------------------------}
procedure glDraw();
begin
 glClear(GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT);       // Clear The
Screen And The Depth Buffer
 glLoadIdentity();                          // Reset The View

 if ShowBackground then
 begin
  glPushMatrix();
   glEnable(GL_TEXTURE_2D);
   glBindTexture(GL_TEXTURE_2D, backgroundTex);
   glTranslatef(0.0,0.0,-2.8);
   glRotate(ElapsedTime/500, 0, 0, 1);
   glColor3f(1, 1, 1);
   glBegin(GL_QUADS);
    glTexCoord2f(0.0, 0.0); glVertex3f(-1.0, -0.9, 0);
    glTexCoord2f(1.0, 0.0); glVertex3f(+1.0, -0.9, 0);
    glTexCoord2f(1.0, 1.0); glVertex3f(+1.0, +0.9, 0);
    glTexCoord2f(0.0, 1.0); glVertex3f(-1.0, +0.9, 0);
   glEnd();
   glDisable(GL_TEXTURE_2D);
  glPopMatrix;
 end;

 glTranslatef(0.0,0.0,-31);

 glRotatef(20, 1, 0, 0);
 glRotatef(ElapsedTime/300, 0, 1, 0);

 // Check if the particle has reached its destination. If so, select a new destination
 updateParticles;
```

```
  // Draw the particle tail
  drawParticleTail;

  // Draw the actual particles
  drawParticleHead;
end;

{------------------------------------------------------------------}
{  Initialise OpenGL  }
{------------------------------------------------------------------}
procedure glInit();
var I, J : Integer;
begin
  glClearColor(0.1, 0.1, 0.4, 0.0);               // Black Background
  glShadeModel(GL_SMOOTH);               // Enables Smooth Color Shading
  glClearDepth(1.0);                    // Depth Buffer Setup
//  glEnable(GL_DEPTH_TEST);               // Enable Depth Buffer
  glDepthFunc(GL_LESS);                           // The Type Of Depth Test To Do
  glBlendFunc(GL_ONE, GL_ONE);

  glHint(GL_PERSPECTIVE_CORRECTION_HINT,    GL_NICEST);          //Realy    Nice
perspective calculations
  glEnable(GL_TEXTURE_2D);                  // Enable Texture Mapping
  LoadTexture('particle.bmp', particleTex);    // Load the Texture
  LoadTexture('background.bmp', backgroundTex);    // Load the Texture
  glBindTexture(GL_TEXTURE_2D, particleTex);  // Bind the Texture to the object

  Randomize;
  // Initialise all particles
  for I :=1 to MAX_PARTICLES do
  begin
    Particle[I].Visited[1] :=random(GRID_WIDTH*GRID_WIDTH*GRID_HEIGHT);
    for J :=2 to TAIL_LENGTH do
      Particle[I].Visited[J] :=Particle[I].Visited[1];
```

```
   Particle[I].NextVertex :=Particle[I].Visited[1];
   Particle[I].Speed :=0.01 + Round(Random(5))/100;
   Particle[I].DestValue :=0;
  end;

  // initialise grid
  for I :=0 to GRID_WIDTH * GRID_WIDTH * GRID_HEIGHT-1 do
  begin
   Grid[I].X :=  I MOD GRID_WIDTH - (GRID_WIDTH DIV 2);
   Grid[I].Z := (I DIV GRID_WIDTH) MOD GRID_WIDTH - (GRID_WIDTH DIV 2);
   Grid[I].Y := (I DIV (GRID_WIDTH*GRID_WIDTH)) MOD GRID_HEIGHT -
(GRID_HEIGHT DIV 2);
  end;
end;

{------------------------------------------------------------------}
{  Handle window resize }
{------------------------------------------------------------------}
procedure glResizeWnd(Width, Height : Integer);
begin
  if (Height = 0) then              // prevent divide by zero exception
   Height := 1;
  glViewport(0, 0, Width, Height);    // Set the viewport for the OpenGL window
  glMatrixMode(GL_PROJECTION);       // Change Matrix Mode to Projection
  glLoadIdentity();                 // Reset View
  gluPerspective(45.0, Width/Height, 1.0, 100.0);      // Do the perspective
calculations. Last value = max clipping depth

  glMatrixMode(GL_MODELVIEW);        // Return to the modelview matrix
  glLoadIdentity();                 // Reset View
end;

{------------------------------------------------------------------}
{  Processes all the keystrokes  }
```

```
{------------------------------------------------------------------}
procedure ProcessKeys;
begin
 if Keys[VK_ADD] then
 begin
  Inc(Particles);
  if Particles > MAX_PARTICLES then
   Particles :=MAX_PARTICLES;
 end;

 if Keys[VK_SUBTRACT] then
 begin
  Dec(Particles);
  if Particles = 0then
   Particles :=1;
 end;

 if Keys[Ord('B')] then
 begin
  ShowBackground :=NOT(ShowBackground);
  Keys[Ord('B')] :=FALSE;
 end;
end;

{------------------------------------------------------------------}
{  Determines the application's response to the messages received  }
{------------------------------------------------------------------}
function WndProc(hWnd: HWND; Msg: UINT;  wParam: WPARAM;  lParam:
LPARAM): LRESULT; stdcall;
begin
 case (Msg) of
  WM_CREATE:
   begin
    // Insert stuff you want executed when the program starts
```

```
    end;
  WM_CLOSE:
    begin
      PostQuitMessage(0);
      Result := 0
    end;
  WM_KEYDOWN:        // Set the pressed key (wparam) to equal true so we can
check if its pressed
    begin
      keys[wParam] := True;
      Result := 0;
    end;
  WM_KEYUP:        // Set the released key (wparam) to equal false so we can check
if its pressed
    begin
      keys[wParam] := False;
      Result := 0;
    end;
  WM_SIZE:        // Resize the window with the new width and height
    begin
      glResizeWnd(LOWORD(lParam),HIWORD(lParam));
      Result := 0;
    end;
  WM_TIMER :                  // Add code here for all timers to be used.
    begin
      if wParam = FPS_TIMER then
      begin
        FPSCount :=Round(FPSCount * 1000/FPS_INTERVAL);   // calculate to get per
Second incase intercal is less or greater than 1 second
        SetWindowText(h_Wnd, PChar(WND_TITLE + '    [' + intToStr(FPSCount) + '
FPS]    Particles : ' + intToStr(Particles)));
        FPSCount := 0;
        Result := 0;
      end;
```

```
    end;
   else
    Result := DefWindowProc(hWnd, Msg, wParam, lParam);     // Default result if
nothing happens
  end;
end;

{-------------------------------------------------------------------}
{  Properly destroys the window created at startup (no memory leaks)  }
{-------------------------------------------------------------------}
procedure glKillWnd(Fullscreen : Boolean);
begin
 if Fullscreen then           // Change back to non fullscreen
 begin
  ChangeDisplaySettings(devmode(nil^), 0);
  ShowCursor(True);
 end;

 // Makes current rendering context not current, and releases the device
 // context that is used by the rendering context.
 if (not wglMakeCurrent(h_DC, 0)) then
  MessageBox(0, 'Release of DC and RC failed!', 'Error', MB_OK or MB_ICONERROR);

 // Attempts to delete the rendering context
 if (not wglDeleteContext(h_RC)) then
 begin
  MessageBox(0, 'Release of rendering context failed!', 'Error', MB_OK or
MB_ICONERROR);
  h_RC := 0;
 end;

 // Attemps to release the device context
 if ((h_DC > 0) and (ReleaseDC(h_Wnd, h_DC) = 0)) then
 begin
```

```
  MessageBox(0, 'Release of device context failed!', 'Error', MB_OK or
MB_ICONERROR);
  h_DC := 0;
 end;

 // Attempts to destroy the window
 if ((h_Wnd <> 0) and (not DestroyWindow(h_Wnd))) then
 begin
  MessageBox(0, 'Unable to destroy window!', 'Error', MB_OK or MB_ICONERROR);
  h_Wnd := 0;
 end;

 // Attempts to unregister the window class
 if (not UnRegisterClass('OpenGL', hInstance)) then
 begin
  MessageBox(0, 'Unable to unregister window class!', 'Error', MB_OK or
MB_ICONERROR);
  hInstance := 0;
 end;
end;

{-------------------------------------------------------------------}
{  Creates the window and attaches a OpenGL rendering context to it  }
{-------------------------------------------------------------------}
function glCreateWnd(Width, Height : Integer; Fullscreen : Boolean; PixelDepth :
Integer) : Boolean;
var
 wndClass : TWndClass;        // Window class
 dwStyle : DWORD;             // Window styles
 dwExStyle : DWORD;           // Extended window styles
 dmScreenSettings : DEVMODE;   // Screen settings (fullscreen, etc...)
 PixelFormat : GLuint;        // Settings for the OpenGL rendering
 h_Instance : HINST;          // Current instance
 pfd : TPIXELFORMATDESCRIPTOR;  // Settings for the OpenGL window
```

```
begin
 h_Instance := GetModuleHandle(nil);      //Grab An Instance For Our Window
 ZeroMemory(@wndClass, SizeOf(wndClass));  // Clear the window class structure

 with wndClass do                  // Set up the window class
 begin
  style       := CS_HREDRAW or    // Redraws entire window if length changes
                 CS_VREDRAW or    // Redraws entire window if height changes
                 CS_OWNDC;       // Unique device context for the window
  lpfnWndProc    := @WndProc;           // Set the window procedure to our func
WndProc
  hInstance     := h_Instance;
  hCursor      := LoadCursor(0, IDC_ARROW);
  lpszClassName := 'OpenGL';
 end;

 if (RegisterClass(wndClass) = 0) then  // Attemp to register the window class
 begin
  MessageBox(0, 'Failed to register the window class!', 'Error', MB_OK or
MB_ICONERROR);
  Result := False;
  Exit
 end;

 // Change to fullscreen if so desired
 if Fullscreen then
 begin
  ZeroMemory(@dmScreenSettings, SizeOf(dmScreenSettings));
  with dmScreenSettings do begin            // Set parameters for the screen setting
   dmSize      := SizeOf(dmScreenSettings);
   dmPelsWidth  := Width;                // Window width
   dmPelsHeight := Height;               // Window height
   dmBitsPerPel := PixelDepth;             // Window color depth
   dmFields     := DM_PELSWIDTH or DM_PELSHEIGHT or DM_BITSPERPEL;
```

```
  end;

  // Try to change screen mode to fullscreen
  if (ChangeDisplaySettings(dmScreenSettings, CDS_FULLSCREEN) =
DISP_CHANGE_FAILED) then
  begin
   MessageBox(0, 'Unable to switch to fullscreen!', 'Error', MB_OK or
MB_ICONERROR);
   Fullscreen := False;
  end;
 end;

 // If we are still in fullscreen then
 if (Fullscreen) then
 begin
  dwStyle := WS_POPUP or          // Creates a popup window
        WS_CLIPCHILDREN       // Doesn't draw within child windows
        or WS_CLIPSIBLINGS;   // Doesn't draw within sibling windows
  dwExStyle := WS_EX_APPWINDOW;    // Top level window
  ShowCursor(False);            // Turn of the cursor (gets in the way)
 end
 else
 begin
  dwStyle := WS_OVERLAPPEDWINDOW or  // Creates an overlapping window
        WS_CLIPCHILDREN or     // Doesn't draw within child windows
        WS_CLIPSIBLINGS;       // Doesn't draw within sibling windows
  dwExStyle := WS_EX_APPWINDOW or   // Top level window
         WS_EX_WINDOWEDGE;    // Border with a raised edge
 end;

 // Attempt to create the actual window
 h_Wnd := CreateWindowEx(dwExStyle,  // Extended window styles
              'OpenGL',   // Class name
              WND_TITLE,  // Window title (caption)
```

```
                dwStyle,      // Window styles
                0, 0,         // Window position
                Width, Height,  // Size of window
                0,            // No parent window
                0,            // No menu
                h_Instance,    // Instance
                nil);         // Pass nothing to WM_CREATE
 if h_Wnd = 0 then
 begin
  glKillWnd(Fullscreen);              // Undo all the settings we've changed
  MessageBox(0, 'Unable to create window!', 'Error', MB_OK or MB_ICONERROR);
  Result := False;
  Exit;
 end;

 // Try to get a device context
 h_DC := GetDC(h_Wnd);
 if (h_DC = 0) then
 begin
  glKillWnd(Fullscreen);
  MessageBox(0, 'Unable to get a device context!', 'Error', MB_OK or
MB_ICONERROR);
  Result := False;
  Exit;
 end;

 // Settings for the OpenGL window
 with pfd do
 begin
  nSize          := SizeOf(TPIXELFORMATDESCRIPTOR); // Size Of This Pixel Format
Descriptor
  nVersion       := 1;                // The version of this data structure
  dwFlags            := PFD_DRAW_TO_WINDOW      // Buffer supports drawing to
window
```

```
            or PFD_SUPPORT_OPENGL // Buffer supports OpenGL drawing
            or PFD_DOUBLEBUFFER;  // Supports double buffering
  iPixelType     := PFD_TYPE_RGBA;       // RGBA color format
  cColorBits     := PixelDepth;          // OpenGL color depth
  cRedBits       := 0;                   // Number of red bitplanes
  cRedShift      := 0;                   // Shift count for red bitplanes
  cGreenBits     := 0;                   // Number of green bitplanes
  cGreenShift    := 0;                   // Shift count for green bitplanes
  cBlueBits      := 0;                   // Number of blue bitplanes
  cBlueShift     := 0;                   // Shift count for blue bitplanes
  cAlphaBits     := 0;                   // Not supported
  cAlphaShift    := 0;                   // Not supported
  cAccumBits     := 0;                   // No accumulation buffer
  cAccumRedBits  := 0;                   // Number of red bits in a-buffer
  cAccumGreenBits := 0;                  // Number of green bits in a-buffer
  cAccumBlueBits  := 0;                  // Number of blue bits in a-buffer
  cAccumAlphaBits := 0;                  // Number of alpha bits in a-buffer
  cDepthBits     := 16;                  // Specifies the depth of the depth buffer
  cStencilBits   := 0;                   // Turn off stencil buffer
  cAuxBuffers    := 0;                   // Not supported
  iLayerType     := PFD_MAIN_PLANE;      // Ignored
  bReserved      := 0;                   // Number of overlay and underlay planes
  dwLayerMask    := 0;                   // Ignored
  dwVisibleMask  := 0;                   // Transparent color of underlay plane
  dwDamageMask   := 0;                   // Ignored
 end;

 // Attempts to find the pixel format supported by a device context that is the best
match to a given pixel format specification.
 PixelFormat := ChoosePixelFormat(h_DC, @pfd);
 if (PixelFormat = 0) then
 begin
  glKillWnd(Fullscreen);
```

```
  MessageBox(0, 'Unable to find a suitable pixel format', 'Error', MB_OK or
MB_ICONERROR);
  Result := False;
  Exit;
 end;

 // Sets the specified device context's pixel format to the format specified by the
PixelFormat.
 if (not SetPixelFormat(h_DC, PixelFormat, @pfd)) then
 begin
  glKillWnd(Fullscreen);
  MessageBox(0, 'Unable to set the pixel format', 'Error', MB_OK or
MB_ICONERROR);
  Result := False;
  Exit;
 end;

 // Create a OpenGL rendering context
 h_RC := wglCreateContext(h_DC);
 if (h_RC = 0) then
 begin
  glKillWnd(Fullscreen);
  MessageBox(0, 'Unable to create an OpenGL rendering context', 'Error', MB_OK or
MB_ICONERROR);
  Result := False;
  Exit;
 end;

 // Makes the specified OpenGL rendering context the calling thread's current
rendering context
 if (not wglMakeCurrent(h_DC, h_RC)) then
 begin
  glKillWnd(Fullscreen);
```

```
   MessageBox(0, 'Unable to activate OpenGL rendering context', 'Error', MB_OK or
MB_ICONERROR);
   Result := False;
   Exit;
  end;

  // Initializes the timer used to calculate the FPS
  SetTimer(h_Wnd, FPS_TIMER, FPS_INTERVAL, nil);

  // Settings to ensure that the window is the topmost window
  ShowWindow(h_Wnd, SW_SHOW);
  SetForegroundWindow(h_Wnd);
  SetFocus(h_Wnd);

  // Ensure the OpenGL window is resized properly
  glResizeWnd(Width, Height);
  glInit();
  Result := True;
end;

{------------------------------------------------------------------}
{  Main message loop for the application }
{------------------------------------------------------------------}
function WinMain(hInstance : HINST; hPrevInstance : HINST;
            lpCmdLine : PChar; nCmdShow : Integer) : Integer; stdcall;
var
  msg : TMsg;
  finished : Boolean;
  DemoStart, LastTime : DWord;
begin
  finished := False;

  // Perform application initialization:
  if not glCreateWnd(800, 600, FALSE, 32) then
```

```
 begin
  Result := 0;
  Exit;
 end;
 DemoStart := GetTickCount();          // Get Time when demo started

 // Main message loop:
 while not finished do
 begin
  if (PeekMessage(msg, 0, 0, 0, PM_REMOVE)) then // Check if there is a message
for this window
  begin
   if (msg.message = WM_QUIT) then     // If WM_QUIT message received then we
are done
    finished := True
   else
   begin                         // Else translate and dispatch the message to this window
        TranslateMessage(msg);
    DispatchMessage(msg);
   end;
  end
  else
  begin
   Inc(FPSCount);                  // Increment FPS Counter

   LastTime :=ElapsedTime;
   ElapsedTime :=GetTickCount() - DemoStart;    // Calculate Elapsed Time
   ElapsedTime :=(LastTime + ElapsedTime) DIV 2; // Average it out for smoother
movement

   glDraw();                       // Draw the scene
   SwapBuffers(h_DC);              // Display the scene

   if (keys[VK_ESCAPE]) then         // If user pressed ESC then set finised TRUE
```

```
      finished := True
    else
      ProcessKeys;                    // Check for any other key Pressed
   end;
  end;
  glKillWnd(FALSE);
  Result := msg.wParam;
end;

begin
  WinMain( hInstance, hPrevInst, CmdLine, CmdShow );
end.
```

dan klik Save All pada menu utama File.

Bab 3

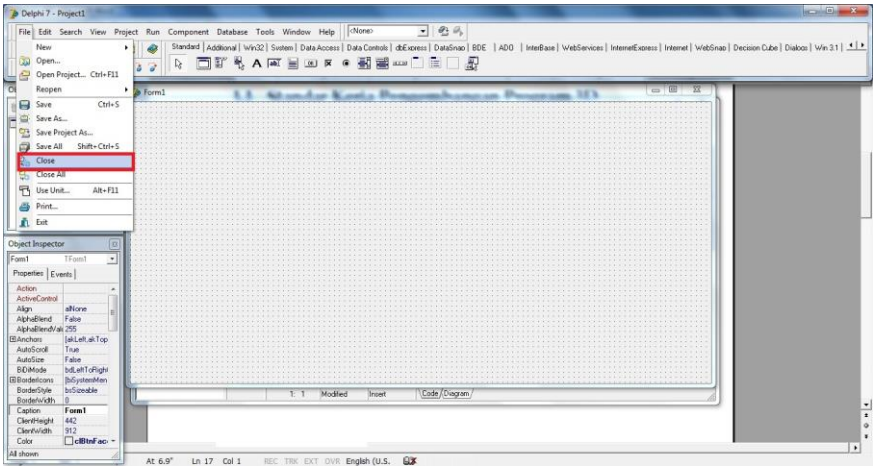
SOLAR SYSTEM 3D

3.1 Persiapan Awal

Untuk pembangunan program Solar System dengan visual 3D, terlebih dahulu disiapkan beberapa buah file template gambar berformat bmp. Kegunaan dari gambar-gambar tersebut hanya untuk memberi efek visual yang lebih nyata pada objek-objek seperti matahari, dan planet-planet lainnya.

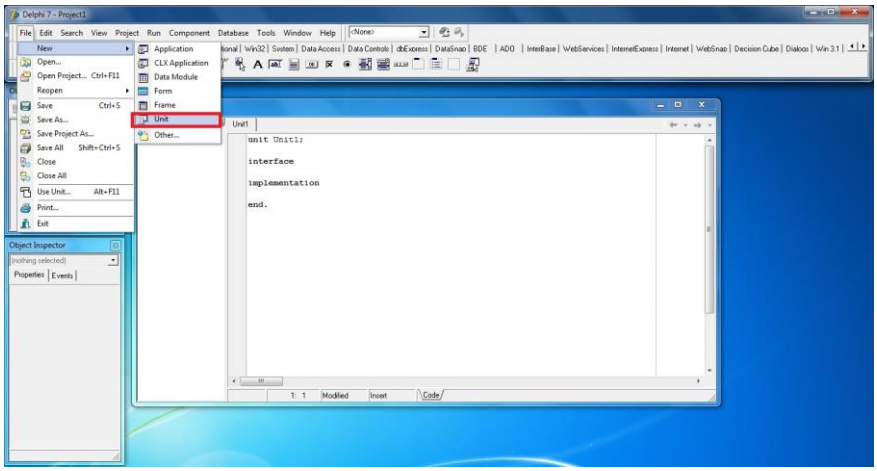
Sebelum masuk kepada penulisan koding program source file, maka setelah muncul IDE utama Delphi, lakukan langkah berikut :

File dan Close



3.2 Source File

Untuk membangun Delphi Source File, maka sorot ke menu File, lalu sorot New, dan klik Unit.



Setelah muncul halaman koding, lalu tuliskan koding berikut :

```
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);
  glTexEnvi(GL_TEXTURE_ENV,  GL_TEXTURE_ENV_MODE,  GL_MODULATE);
{Texture blends with object background}
//  glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);  {Texture
does NOT blend with object background}

  { 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 }

  glTexParameteri(GL_TEXTURE_2D,  GL_TEXTURE_MAG_FILTER,  GL_LINEAR);  {
only first two can be used }
  glTexParameteri(GL_TEXTURE_2D,  GL_TEXTURE_MIN_FILTER,  GL_LINEAR);  { all
of the above can be used }

  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;
```

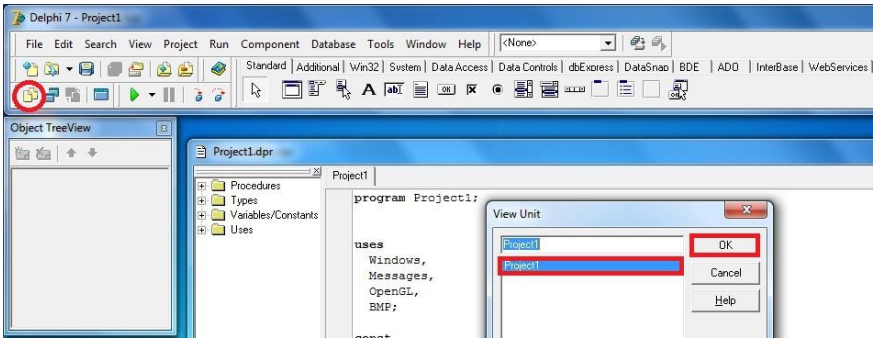
```
  ImageSize  := Width*Height*(ColorDepth div 8);

  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.
```

Lalu Save dengan nama file Textures.

3.3 Proyek Utama

Pada pembangunan proyek utama, lakukan File dan Close, lalu klik icon View Unit pada pojok kiri panel komponen, dan pilih project1, lalu klik tombol ok.



Setelah muncul halaman koding, selanjutnya tuliskan koding berikut :

```
program Project1;

uses
  Windows,
  Messages,
  OpenGL,
  Textures;

const
  WND_TITLE = 'OpenGL Solar System';
  FPS_TIMER = 1;                // Timer to calculate FPS
  FPS_INTERVAL = 1000;          // Calculate FPS every 1000 ms

  SUN   = 100;                                          // This is the
object ID for the SUN
  EARTH = 101;                                          // This is the
object ID for the EARTH
  PLUTO         = 102;                                          //
This is the object ID for PLUTO
  MARS  = 103;            // PLANET BARU
```

```
var
  h_Wnd  : HWND;                // Global window handle
  h_DC   : HDC;                 // Global device context
  h_RC   : HGLRC;               // OpenGL rendering context
  keys : Array[0..255] of Boolean;   // Holds keystrokes
  FPSCount : Integer = 0;          // Counter for FPS
  ElapsedTime : cardinal;          // Elapsed time between frames

  // Textures
  SunTex :  glUInt;
  EarthTex: glUInt;
  PlutoTex: glUInt;
  //tambahkan planet baru
  MarsTex: glUInt;         //PLANET BARU

  // User variables
  pObj: GLUQuadricObj;
  SunRotation   : single = 90;
          // This holds our sun's current rotation
  EarthRotation : single = 90;
          // This holds our earth's current rotation
  PlutoRotation : single = 90;
          // This holds pluto's current rotation

  MarsRotation  : single = 90;              //PLANET BARU

  g_Fullscreen: boolean;

{$R *.RES}

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

{------------------------------------------------------------------}
```

```
{  Function to convert int to string. (No sysutils = smaller EXE)  }
{------------------------------------------------------------------}
function IntToStr(Num : Integer) : String;  // using SysUtils increase file size by 100K
begin
  Str(Num, result);
end;

{------------------------------------------------------------------}
{  Function to draw the actual scene }
{------------------------------------------------------------------}
procedure glDraw();
begin
  glClear(GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT);          // Clear The Screen And The Depth Buffer
  glLoadIdentity();
            // Reset The matrix

  // We make our position a bit high and back to view the whole scene

  //        Position      View          Up Vector
  gluLookAt(0, 3, 6,    0, 0, 0,    0, 1, 0);           // This determines where the camera's position and view is

  glInitNames();
                    // This clears the name stack so we always start with 0 names.

  // This next line is important.  If you don't push on at least ONE name,
  // The selection won't work.  Instead of glLoadName()/glEnd() you can use
  // glPushName(TheID) and glPopName();    Then you don't need to glPushName(0);

  glPushName(0);
                    // This starts off the first object in the stack
```

```
 pObj := gluNewQuadric();                    // Get a new Quadric off the
stack

 gluQuadricTexture(pObj, true);                                    //
This turns on texture coordinates for our Quadrics

 // Bind the sun texture to the sun quadratic
 glBindTexture(GL_TEXTURE_2D, SunTex);              // Bind the Sun
texture to the sun

 // Below we call glLoadName().  We need to pass in an ID that we can check later
that will
 // be associated with the polygons drawn next.  Here is how it works.  We call
glLoadName()
 // and pass an ID.  Then we draw any primitives or shapes, then we call glEnd()
which
 // stops assigning polys to that object name.  We now have a group of polygons
that are
 // given an ID.  Our ID SUN now refers to the sun Quadric we draw below.

 glLoadName(SUN);
        // Push on our SUN label (IMPORTANT)

 // If we use glLoadName(), then we need to end it with glEnd().  There is a
problem
 // though with some video cards that MUST have a glBegin()/glEnd() between
 // the calls to glLoadName()/glEnd().  This is strange but some cards grind
 // to a 2 FPS speed (VoodooCards).  Since we are using Quadrics there is
 // no glBegin()/glEnd() being called explicitly, so we need to fake it.
 // *Remember, you only have to do this if you are using Quadrics.*  So, to fix
 // this we just put a empty glBegin()/glEnd() statement between each object ID
passed in.
 glBegin(GL_LINES);
 glEnd();
```

```
// Here we push on a new matrix so we don't affect any other quadrics.
// We first translate the quadric to the origin (0, 0, 0), Then we rotate it
// about the Y axis.  This gives it the spinning effect.  Then we draw the
// largest of the spheres.  This represents the sun with its texture map.

glPushMatrix();
                    // Push on a new matrix scope
  glTranslatef(0, 0, 0);                                        // Translate
this sphere to the left
  glRotatef(SunRotation, 0, 1.0, 0);                    // Rotate the sphere
around the Y axis to make it spin
  gluSphere(pObj, 0.5, 20, 20);                         // Draw the sunwith a
radius of 0.5
glPopMatrix();
                    // End the current scope of this matrix

// Now that we drew the sun, we want to end our Object name.  We call
glPopName()
// to do that.  Now nothing else will be associated with the SUN ID.

glEnd();
                    // Stop assigning polygons to the SUN label (IMPORTANT)

// Next, we want to bind the Earth texture to our Earth sphere
glBindTexture(GL_TEXTURE_2D, EarthTex);

// Once again, we want to create a object ID for our earth, so we push on the
EARTH ID.
// Now, when we draw the next sphere, it will be associated with the EARTH ID.

glLoadName(EARTH);
                    // Push on our EARTH label (IMPORTANT)

// Pass in our empty glBegin()/glEnd() statement because we are using Quadrics.
```

```
 // If we don't do this when using glLoadName(), it will grind to a hault on some cards.
 glBegin(GL_LINES);
 glEnd();

 // Once again, we want to pop on a new matrix as not to affect any other spheres.
 // We rotate the sphere by its current rotation value FIRST before we translate it.
 // This makes it rotate around the origin, which is where the sun is.
 // Then we rotate it again about the Y-axis to make it spin around itself.

 glPushMatrix();
                // Push on a new matrix scope
        glRotatef(EarthRotation / 3, 0, 1.0, 0);        // Rotate the sphere around the origin (the sun)
        glTranslatef(-2, 0, 0);                                                 // Translate this sphere to the left
        glRotatef(EarthRotation, 0, 1.0, 0);                    // Rotate the sphere to make it spin
        gluSphere(pObj, 0.2, 20, 20);                                   // Draw the sphere with a radius of 0.2 so it's smaller than the sun
 glPopMatrix();
                // End the current scope of this matrix

 // We are done assigning the EARTH object, so we need
 // to stop assigning polygons to the current ID.

 glEnd();
                // Stop assigning polygons to the EARTH label (IMPORTANT)

 // Bind the pluto texture to the last sphere
 glBindTexture(GL_TEXTURE_2D, PlutoTex);

 // Finally, we want to be able to click on Pluto, so we need a pluto ID.
```

```
 glLoadName(PLUTO);
                    // Push on our PLUTO label (IMPORTANT)

 // Like we did with the earth, we rotate Pluto around the sun first,
 // then we translate it farther away from the sun.  Next, we rotate Pluto
 // around the Y axis to give it some spin.

 // Pass in our empty glBegin()/glEnd() statement because we are using Quadrics.
 // If we don't do this when using glLoadName(), it will grind to a hault on some cards.
 glBegin(GL_LINES);
 glEnd();

 glPushMatrix();
                    // Push on a new matrix scope
 glRotatef(PlutoRotation / 2, 0, 1.0, 0);          // Rotate the sphere around the sun
 glTranslatef(3, 0, 0);                                    // Translate this sphere farther away from the sun than the earth
        glRotatef(PlutoRotation, 0, 1.0, 0);          // Rotate the sphere around itself to produce the spin
        gluSphere(pObj, 0.1, 20, 20);                    // Draw the sphere with a radius of 0.1 (smallest planet)
 glPopMatrix();
                    // End the current scope of this matrix

        // We are finished with the PLUTO object ID, so we need to pop it off the name stack.
 glEnd();
                    // Stop assigning polygons to our PLUTO label (IMPORTANT)

 //SwapBuffers(h_DC);
                    // Swap the backbuffers to the foreground
```

```
 //tambahkan     TAMBAH OBJEK BARU

 glBindTexture(GL_TEXTURE_2D, MarsTex);

 glLoadName(MARS);
        // Push on our OBJEK label (IMPORTANT)
 glBegin(GL_LINES);
 glEnd();

 glPushMatrix();
                // Push on a new matrix scope
 glRotatef(MarsRotation / 4, 0, 1.5, 0);          // Rotate the sphere around the sun
 glTranslatef(2.5, 0, 0);                                                    //
JARAK DARI PUSAT SOLAR SYSTEM
        glRotatef(MarsRotation, 0, 1.0, 0);                    // Rotate the sphere
around itself to produce the spin
        gluSphere(pObj, 0.1, 15, 15);                                  // Draw the
sphere with a radius of 0.1 (smallest planet)
 glPopMatrix();
                // End the current scope of this matrix

 glEnd();

 //akhir tambah     batas BATAS TAMBAHAN OBJEK

 gluDeleteQuadric(pObj);
        // Free the Quadric

 glPopName();
        // We pushed a name on the stack, we must pop it back off

 // Below we increase the rotations for each sphere.
```

```
  SunRotation   := SunRotation   + 0.2;
                    // Rotate the sun slowly
  EarthRotation := EarthRotation + 0.5;
                               // Increase the rotation for the each
  PlutoRotation := PlutoRotation + 0.6;
                    // Make pluto go the fastest
  MarsRotation  := MarsRotation  + 0.7;            //SETTING PERUBAHAN KECEPTAN
end;

function RetrieveObjectID(x, y: integer): integer;
var
  objectsFound: integer;
  viewportCoords: array [0..3] of integer;
  selectBuffer: array[0..31] of cardinal;
  lowestDepth: cardinal;
  selectedObject: cardinal;
  i: integer;
begin
  objectsFound := 0;                                                              //
This will hold the amount of objects clicked
  ZeroMemory(@viewportCoords, sizeof(viewportCoords));
                               // We need an array to hold our view port coordinates
  ZeroMemory(@selectBuffer, sizeof(selectBuffer));
        // This will hold the ID's of the objects we click on.
        // We make it an arbitrary number of 32 because openGL also stores other
information
        // that we don't care about.  There is about 4 slots of info for every object ID
taken up.
        // glSelectBuffer is what we register our selection buffer with.  The first
parameter
        // is the size of our array.  The next parameter is the buffer to store the
information found.
        // More information on the information that will be stored in selectBuffer is
further below.
```

```
	glSelectBuffer(32, @selectBuffer);                                    //
Setup our selection buffer to accept object ID's

	// This function returns information about many things in OpenGL.  We
pass in GL_VIEWPORT
	// to get the view port coordinates.  It saves it like a RECT with {top, left,
bottom, right}

	glGetIntegerv(GL_VIEWPORT, @viewportCoords);                    // Get
the current view port coordinates

	// Now we want to get out of our GL_MODELVIEW matrix and start
effecting our
	// GL_PROJECTION matrix.  This allows us to check our X and Y coords
against 3D space.

	glMatrixMode(GL_PROJECTION);
	// We want to now effect our projection matrix

	glPushMatrix();
			// We push on a new matrix so we don't effect our 3D
projection

		// This makes it so it doesn't change the frame buffer if we render
into it, instead,
		// a record of the names of primitives that would have been drawn
if the render mode was
		// GL_RENDER are now stored in the selection array (selectBuffer).

		glRenderMode(GL_SELECT);
	// Allows us to render the objects, but not change the frame buffer

		glLoadIdentity();
		// Reset our projection matrix
```

```
// gluPickMatrix allows us to create a projection matrix that is around our
// cursor.  This basically only allows rendering in the region that we specify.
// If an object is rendered into that region, then it saves that objects ID for us (The magic).
// The first 2 parameters are the X and Y position to start from, then the next 2
// are the width and height of the region from the starting point. The last parameter is
// of course our view port coordinates.  You will notice we subtract "y" from the
// BOTTOM view port coordinate.  We do this to flip the Y coordinates around.  The 0 y
// coordinate starts from the bottom, which is opposite to window's coordinates.
// We also give a 2 by 2 region to look for an object in.  This can be changed to preference.

gluPickMatrix(x, viewportCoords[3]-y-Integer(not g_Fullscreen)*(GetSystemMetrics(SM_CYCAPTION)+
GetSystemMetrics(SM_CYSIZEFRAME) shl 1), 2, 2, @viewportCoords);

// Next, we just call our normal gluPerspective() function, exactly as we did on startup.
// This is to multiply the perspective matrix by the pick matrix we created up above.

gluPerspective(45.0, viewportCoords[2]/viewportCoords[3], 0.5, 150.0);

glMatrixMode(GL_MODELVIEW);
// Go back into our model view matrix
```

```
            glDraw();
            // Now we render into our selective mode to pinpoint clicked
objects

            // If we return to our normal render mode from select mode,
glRenderMode returns
            // the number of objects that were found in our specified region
(specified in gluPickMatrix())

            objectsFound := glRenderMode(GL_RENDER);                    //
Return to render mode and get the number of objects found

            glMatrixMode(GL_PROJECTION);
        // Put our projection matrix back to normal.
        glPopMatrix();
            // Stop effecting our projection matrix

        glMatrixMode(GL_MODELVIEW);
            // Go back to our normal model view matrix

        // PHEW!  That was some stuff confusing stuff.  Now we are out of the clear
and should have
        // an ID of the object we clicked on.  objectsFound should be at least 1 if we
found an object.

        if (objectsFound > 0) then
        begin
            // If we found more than one object, we need to check the depth
values
            // of all the objects found.  The object with the LEAST depth value is
            // the closest object that we clicked on.  Depending on what you are
doing,
            // you might want ALL the objects that you clicked on (if some
objects were
```

```
		// behind the closest one), but for this tutorial we just care about the one
		// in front.  So, how do we get the depth value?  Well, The selectionBuffer
		// holds it.  For every object there is 4 values.  The first value is
		// "the number of names in the name stack at the time of the event, followed
		// by the minimum and maximum depth values of all vertices that hit since the
		// previous event, then followed by the name stack contents, bottom name first." - MSDN
		// The only ones we care about are the minimum depth value (the second value) and
		// the object ID that was passed into glLoadName() (This is the fourth value).
		// So, [0 - 3] is the first object's data, [4 - 7] is the second object's data, etc...
		// Be carefull though, because if you are displaying 2D text in front, it will
		// always find that as the lowest object.  So make sure you disable text when
		// rendering the screen for the object test.  I use a flag for RenderScene().
		// So, lets get the object with the lowest depth!

		// Set the lowest depth to the first object to start it off.
		// 1 is the first object's minimum Z value.
		// We use an unsigned int so we don't get a warning with selectBuffer below.
		lowestDepth := selectBuffer[1];
		// Set the selected object to the first object to start it off.
		// 3 is the first object's object ID we passed into glLoadName().
		selectedObject := selectBuffer[3];
		// Go through all of the objects found, but start at the second one
```

```
                for i := 1 to objectsFound - 1 do
                begin
                        // Check if the current objects depth is lower than the
current lowest
                        // Notice we times i by 4 (4 values for each object) and add
1 for the depth.
                        if (selectBuffer[(i * 4) + 1] < lowestDepth) then
                        begin
                                // Set the current lowest depth
                                lowestDepth := selectBuffer[(i * 4) + 1];

                                // Set the current object ID
                                selectedObject := selectBuffer[(i * 4) + 3];
                        end
                end;

                // Return the selected object
                result := selectedObject;
         exit;
        end;

        // We didn't click on any objects so return 0
        result := 0;
end;

{------------------------------------------------------------------}
{  Initialise OpenGL  }
{------------------------------------------------------------------}
procedure glInit();
begin
 glClearColor(0.0, 0.0, 0.0, 0.0);              // Black Background
 glShadeModel(GL_SMOOTH);              // Enables Smooth Color Shading
 glClearDepth(1.0);                  // Depth Buffer Setup
 glEnable(GL_DEPTH_TEST);              // Enable Depth Buffer
```

```
 glDepthFunc(GL_LESS);                          // The Type Of Depth Test To Do
 glHint(GL_PERSPECTIVE_CORRECTION_HINT,   GL_NICEST);        //Realy   Nice
perspective calculations
 glEnable(GL_TEXTURE_2D);                // Enable Texture Mapping

 LoadTexture('Sun.bmp', SunTex, false);                                  //          Load
"Sun.bmp" into openGL as a texture for the Sun
 LoadTexture('Earth.bmp', EarthTex, false);                    //  Load  "Earth.bmp"
into openGL as a texture for the Earth
 LoadTexture('Pluto.bmp', PlutoTex, false);                    //  Load  "Pluto.bmp"
into openGL as a texture for Pluto

 LoadTexture('Mars.bmp', MarsTex, false);                      //     gambar   bitmap
citra planet
end;
{-------------------------------------------------------------------}
{  Handle window resize }
{-------------------------------------------------------------------}
procedure glResizeWnd(Width, Height : Integer);
begin
 if (Height = 0) then            // prevent divide by zero exception
   Height := 1;
 glViewport(0, 0, Width, Height);    // Set the viewport for the OpenGL window
 glMatrixMode(GL_PROJECTION);       // Change Matrix Mode to Projection
 glLoadIdentity();               // Reset View

 gluPerspective(45.0, Width/Height, 0.5, 150.0);

 glMatrixMode(GL_MODELVIEW);        // Return to the modelview matrix
 glLoadIdentity();               // Reset View
end;
{-------------------------------------------------------------------}
{  Processes all the keystrokes }
{-------------------------------------------------------------------}
```

```
procedure ProcessKeys;
begin
end;

{------------------------------------------------------------------}
{  Determines the application's response to the messages received  }
{------------------------------------------------------------------}
function WndProc(hWnd: HWND; Msg: UINT;  wParam: WPARAM;  lParam:
LPARAM): LRESULT; stdcall;
var
  objectID: integer;
begin
 case (Msg) of
   WM_CREATE:
    begin
     // Insert stuff you want executed when the program starts
    end;
   WM_CLOSE:
    begin
     PostQuitMessage(0);
     Result := 0
    end;
   WM_KEYDOWN:        // Set the pressed key (wparam) to equal true so we can
check if its pressed
    begin
     keys[wParam] := True;
     Result := 0;
    end;
   WM_KEYUP:        // Set the released key (wparam) to equal false so we can check
if its pressed
    begin
     keys[wParam] := False;
     Result := 0;
    end;
```

```
  WM_SIZE:        // Resize the window with the new width and height
   begin
    glResizeWnd(LOWORD(lParam),HIWORD(lParam));
    Result := 0;
   end;
  WM_TIMER :                // Add code here for all timers to be used.
   begin
    if wParam = FPS_TIMER then
    begin
     FPSCount :=Round(FPSCount * 1000/FPS_INTERVAL);   // calculate to get per
Second incase intercal is less or greater than 1 second
     SetWindowText(h_Wnd, PChar(WND_TITLE + '    [' + intToStr(FPSCount) + '
FPS]'));
     FPSCount := 0;
     Result := 0;
    end;
   end;
  WM_LBUTTONDOWN:
   begin                                                    // If the left
mouse button was clicked
        // Here we pass in the cursors X and Y coordinates to test for an object
under the mouse.
        objectID := RetrieveObjectID(LOWORD(lParam), HIWORD(lParam));

        // Now we just do a switch on our object ID's to see if we hit one.

        case (objectID) of // Check the objectID passed back
         SUN:    MessageBox(0, 'The Sun!', 'Click', MB_OK);
         EARTH: MessageBox(0, 'The Earth!', 'Click', MB_OK);
         PLUTO: MessageBox(0, 'Pluto!', 'Click', MB_OK);
    end;
   end;
  else
```

```
    Result := DefWindowProc(hWnd, Msg, wParam, lParam);     // Default result if
nothing happens
  end;
end;

{------------------------------------------------------------------}
{  Properly destroys the window created at startup (no memory leaks)  }
{------------------------------------------------------------------}
procedure glKillWnd(Fullscreen : Boolean);
begin
  if Fullscreen then           // Change back to non fullscreen
  begin
    ChangeDisplaySettings(devmode(nil^), 0);
    ShowCursor(True);
  end;

  // Makes current rendering context not current, and releases the device
  // context that is used by the rendering context.
  if (not wglMakeCurrent(h_DC, 0)) then
    MessageBox(0, 'Release of DC and RC failed!', 'Error', MB_OK or MB_ICONERROR);

  // Attempts to delete the rendering context
  if (not wglDeleteContext(h_RC)) then
  begin
    MessageBox(0, 'Release of rendering context failed!', 'Error', MB_OK or
MB_ICONERROR);
    h_RC := 0;
  end;

  // Attemps to release the device context
  if ((h_DC > 0) and (ReleaseDC(h_Wnd, h_DC) = 0)) then
  begin
    MessageBox(0, 'Release of device context failed!', 'Error', MB_OK or
MB_ICONERROR);
```

```
  h_DC := 0;
 end;

 // Attempts to destroy the window
 if ((h_Wnd <> 0) and (not DestroyWindow(h_Wnd))) then
 begin
  MessageBox(0, 'Unable to destroy window!', 'Error', MB_OK or MB_ICONERROR);
  h_Wnd := 0;
 end;

 // Attempts to unregister the window class
 if (not UnRegisterClass('OpenGL', hInstance)) then
 begin
  MessageBox(0, 'Unable to unregister window class!', 'Error', MB_OK or
MB_ICONERROR);
  hInstance := 0;
 end;
end;

{-------------------------------------------------------------------}
{  Creates the window and attaches a OpenGL rendering context to it  }
{-------------------------------------------------------------------}
function glCreateWnd(Width, Height : Integer; Fullscreen : Boolean; PixelDepth :
Integer) : Boolean;
var
 wndClass : TWndClass;        // Window class
 dwStyle : DWORD;             // Window styles
 dwExStyle : DWORD;           // Extended window styles
 dmScreenSettings : DEVMODE;  // Screen settings (fullscreen, etc...)
 PixelFormat : GLuint;        // Settings for the OpenGL rendering
 h_Instance : HINST;          // Current instance
 pfd : TPIXELFORMATDESCRIPTOR;  // Settings for the OpenGL window
begin
 h_Instance := GetModuleHandle(nil);      //Grab An Instance For Our Window
```

```
 ZeroMemory(@wndClass, SizeOf(wndClass));  // Clear the window class structure

 with wndClass do                  // Set up the window class
 begin
  style       := CS_HREDRAW or    // Redraws entire window if length changes
                 CS_VREDRAW or    // Redraws entire window if height changes
                 CS_OWNDC;       // Unique device context for the window
  lpfnWndProc    := @WndProc;           // Set the window procedure to our func
WndProc
  hInstance     := h_Instance;
  hCursor      := LoadCursor(0, IDC_ARROW);
  lpszClassName := 'OpenGL';
 end;

 if (RegisterClass(wndClass) = 0) then  // Attemp to register the window class
 begin
  MessageBox(0, 'Failed to register the window class!', 'Error', MB_OK or
MB_ICONERROR);
  Result := False;
  Exit
 end;

 // Change to fullscreen if so desired
 if Fullscreen then
 begin
  ZeroMemory(@dmScreenSettings, SizeOf(dmScreenSettings));
  with dmScreenSettings do begin            // Set parameters for the screen setting
   dmSize      := SizeOf(dmScreenSettings);
   dmPelsWidth  := Width;                 // Window width
   dmPelsHeight := Height;                // Window height
   dmBitsPerPel := PixelDepth;             // Window color depth
   dmFields     := DM_PELSWIDTH or DM_PELSHEIGHT or DM_BITSPERPEL;
  end;
```

```
  // Try to change screen mode to fullscreen
  if (ChangeDisplaySettings(dmScreenSettings, CDS_FULLSCREEN) =
DISP_CHANGE_FAILED) then
  begin
   MessageBox(0, 'Unable to switch to fullscreen!', 'Error', MB_OK or
MB_ICONERROR);
    Fullscreen := False;
  end;
 end;

 // If we are still in fullscreen then
 if (Fullscreen) then
 begin
  dwStyle := WS_POPUP or            // Creates a popup window
        WS_CLIPCHILDREN          // Doesn't draw within child windows
        or WS_CLIPSIBLINGS;      // Doesn't draw within sibling windows
  dwExStyle := WS_EX_APPWINDOW;       // Top level window
  //ShowCursor(False);                // Turn of the cursor (gets in the way)
 end
 else
 begin
  dwStyle := WS_OVERLAPPEDWINDOW or    // Creates an overlapping window
        WS_CLIPCHILDREN or       // Doesn't draw within child windows
        WS_CLIPSIBLINGS;         // Doesn't draw within sibling windows
  dwExStyle := WS_EX_APPWINDOW or      // Top level window
         WS_EX_WINDOWEDGE;       // Border with a raised edge
 end;

 // Attempt to create the actual window
 h_Wnd := CreateWindowEx(dwExStyle,     // Extended window styles
                'OpenGL',      // Class name
                WND_TITLE,     // Window title (caption)
                dwStyle,       // Window styles
                0, 0,          // Window position
```

```
                Width, Height,  // Size of window
                0,          // No parent window
                0,          // No menu
                h_Instance,    // Instance
                nil);         // Pass nothing to WM_CREATE
 if h_Wnd = 0 then
 begin
  glKillWnd(Fullscreen);              // Undo all the settings we've changed
  MessageBox(0, 'Unable to create window!', 'Error', MB_OK or MB_ICONERROR);
  Result := False;
  Exit;
 end;

 // Try to get a device context
 h_DC := GetDC(h_Wnd);
 if (h_DC = 0) then
 begin
  glKillWnd(Fullscreen);
  MessageBox(0, 'Unable to get a device context!', 'Error', MB_OK or
MB_ICONERROR);
  Result := False;
  Exit;
 end;

 // Settings for the OpenGL window
 with pfd do
 begin
  nSize          := SizeOf(TPIXELFORMATDESCRIPTOR); // Size Of This Pixel Format
Descriptor
  nVersion       := 1;                // The version of this data structure
  dwFlags             := PFD_DRAW_TO_WINDOW      // Buffer supports drawing to
window
              or PFD_SUPPORT_OPENGL // Buffer supports OpenGL drawing
              or PFD_DOUBLEBUFFER;  // Supports double buffering
```

```
  iPixelType     := PFD_TYPE_RGBA;       // RGBA color format
  cColorBits     := PixelDepth;          // OpenGL color depth
  cRedBits       := 0;                   // Number of red bitplanes
  cRedShift      := 0;                   // Shift count for red bitplanes
  cGreenBits     := 0;                   // Number of green bitplanes
  cGreenShift    := 0;                   // Shift count for green bitplanes
  cBlueBits      := 0;                   // Number of blue bitplanes
  cBlueShift     := 0;                   // Shift count for blue bitplanes
  cAlphaBits     := 0;                   // Not supported
  cAlphaShift    := 0;                   // Not supported
  cAccumBits     := 0;                   // No accumulation buffer
  cAccumRedBits  := 0;                   // Number of red bits in a-buffer
  cAccumGreenBits := 0;                  // Number of green bits in a-buffer
  cAccumBlueBits  := 0;                  // Number of blue bits in a-buffer
  cAccumAlphaBits := 0;                  // Number of alpha bits in a-buffer
  cDepthBits     := 16;                  // Specifies the depth of the depth buffer
  cStencilBits   := 0;                   // Turn off stencil buffer
  cAuxBuffers    := 0;                   // Not supported
  iLayerType     := PFD_MAIN_PLANE;      // Ignored
  bReserved      := 0;                   // Number of overlay and underlay planes
  dwLayerMask    := 0;                   // Ignored
  dwVisibleMask  := 0;                   // Transparent color of underlay plane
  dwDamageMask   := 0;                   // Ignored
 end;

 // Attempts to find the pixel format supported by a device context that is the best
match to a given pixel format specification.
 PixelFormat := ChoosePixelFormat(h_DC, @pfd);
 if (PixelFormat = 0) then
 begin
  glKillWnd(Fullscreen);
  MessageBox(0, 'Unable to find a suitable pixel format', 'Error', MB_OK or
MB_ICONERROR);
  Result := False;
```

```
  Exit;
 end;

 // Sets the specified device context's pixel format to the format specified by the
PixelFormat.
 if (not SetPixelFormat(h_DC, PixelFormat, @pfd)) then
 begin
  glKillWnd(Fullscreen);
  MessageBox(0, 'Unable to set the pixel format', 'Error', MB_OK or
MB_ICONERROR);
  Result := False;
  Exit;
 end;

 // Create a OpenGL rendering context
 h_RC := wglCreateContext(h_DC);
 if (h_RC = 0) then
 begin
  glKillWnd(Fullscreen);
  MessageBox(0, 'Unable to create an OpenGL rendering context', 'Error', MB_OK or
MB_ICONERROR);
  Result := False;
  Exit;
 end;

 // Makes the specified OpenGL rendering context the calling thread's current
rendering context
 if (not wglMakeCurrent(h_DC, h_RC)) then
 begin
  glKillWnd(Fullscreen);
  MessageBox(0, 'Unable to activate OpenGL rendering context', 'Error', MB_OK or
MB_ICONERROR);
  Result := False;
  Exit;
```

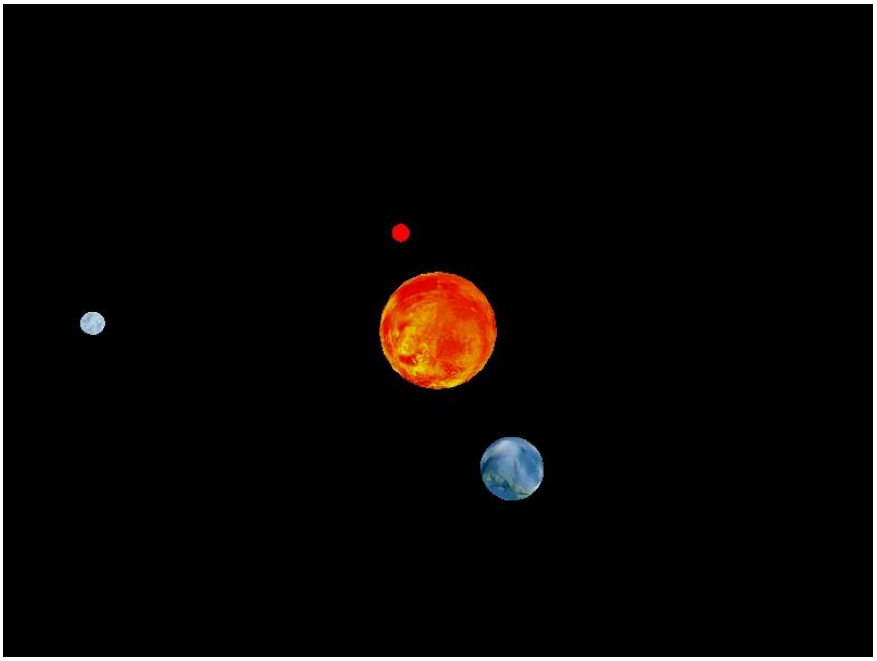
```
  end;

  // Initializes the timer used to calculate the FPS
  SetTimer(h_Wnd, FPS_TIMER, FPS_INTERVAL, nil);

  // Settings to ensure that the window is the topmost window
  ShowWindow(h_Wnd, SW_SHOW);
  SetForegroundWindow(h_Wnd);
  SetFocus(h_Wnd);

  // Ensure the OpenGL window is resized properly
  glResizeWnd(Width, Height);
  glInit();

  Result := True;
end;
{------------------------------------------------------------------}
{  Main message loop for the application }
{------------------------------------------------------------------}
function WinMain(hInstance : HINST; hPrevInstance : HINST;
            lpCmdLine : PChar; nCmdShow : Integer) : Integer; stdcall;
var
  msg : TMsg;
  finished : Boolean;
  DemoStart, LastTime : DWord;
begin
  finished := False;
  g_Fullscreen := true;

  // Perform application initialization:
  if not glCreateWnd(800, 600, g_Fullscreen, 32) then
  begin
    Result := 0;
    Exit;
```

```
 end;

 DemoStart := GetTickCount();          // Get Time when demo started

 // Main message loop:
 while not finished do
 begin
  if (PeekMessage(msg, 0, 0, 0, PM_REMOVE)) then // Check if there is a message
for this window
  begin
   if (msg.message = WM_QUIT) then      // If WM_QUIT message received then we
are done
    finished := True
   else
   begin                         // Else translate and dispatch the message to this window
        TranslateMessage(msg);
    DispatchMessage(msg);
   end;
  end
  else
  begin
   Inc(FPSCount);                  // Increment FPS Counter

   LastTime :=ElapsedTime;
   ElapsedTime :=GetTickCount() - DemoStart;    // Calculate Elapsed Time
   ElapsedTime :=(LastTime + ElapsedTime) DIV 2; // Average it out for smoother
movement

   glDraw();                       // Draw the scene
   SwapBuffers(h_DC);              // Display the scene

   if (keys[VK_ESCAPE]) then         // If user pressed ESC then set finised TRUE
    finished := True
   else
```

```
      ProcessKeys;                    // Check for any other key Pressed
    end;
  end;
  glKillWnd(Boolean(g_Fullscreen));
  Result := msg.wParam;
end;

begin
  {$WARNINGS OFF}
  WinMain( hInstance, 0, CmdLine, CmdShow );
  {$WARNINGS ON}
end.
```

dan klik Save All pada menu utama File.

Tampilan eksekusi program akan berupa :



Bab 4

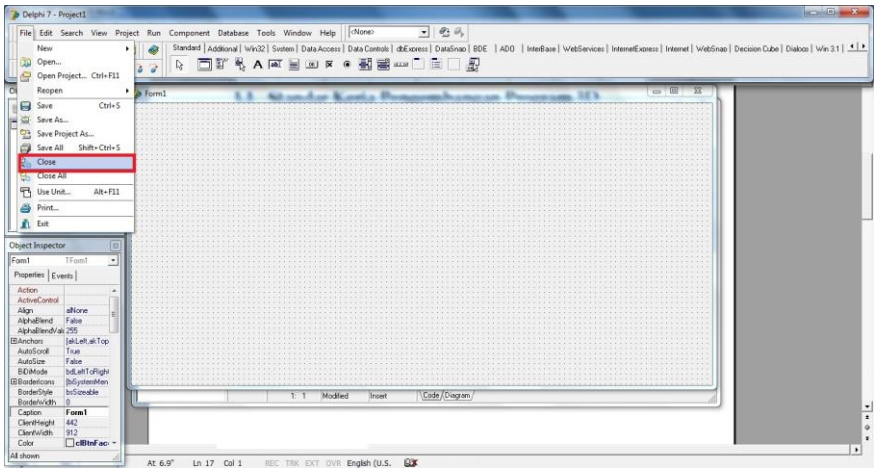
FONT

4.1 Persiapan Awal

Untuk pembangunan program Font dengan visual 3D, tidak dibutuhkan source file.

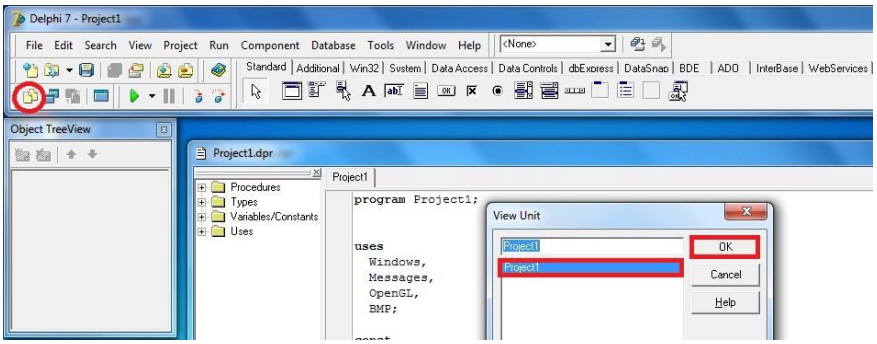
Sebelum masuk kepada penulisan koding program source file, maka setelah muncul IDE utama Delphi, lakukan langkah berikut :

File dan Close



4.2 Proyek Utama

Pada pembangunan proyek utama, lakukan File dan Close, lalu klik icon View Unit pada pojok kiri panel komponen, dan pilih project1, lalu klik tombol ok.



Setelah muncul halaman koding, selanjutnya tuliskan koding berikut :

```
program Project1;
uses
 Windows,
 Messages,
 OpenGL,
 Graphics;

const
 WND_TITLE     = '3D Font Informatika Universitas Malikussaleh';
 FPS_TIMER     = 1;                    // Timer to calculate FPS
 FPS_INTERVAL  = 1000;                 // Calculate FPS every 1000 ms

 gGlobalAmbient  : array[0..3] of glFloat = (1.0, 1.0, 1.0, 1.0); // Set Ambient Lighting

 gLight0Pos    : array[0..3] of glFloat = (0.0, 5.0, 0.0, 1.0); // Position for Light0
 gLight0Ambient : array[0..3] of glFloat = (0.2, 0.2, 0.2, 1.0); // Ambient setting for light0
 gLight0Diffuse  : array[0..3] of glFloat = (1.0, 1.0, 1.0, 1.0); // Diffuse setting for Light0
```

```
 gLight0Specular : array[0..3] of glFloat = (0.8, 0.8, 0.8, 1.0); // Specular lighting for
Light0
 gLModelAmbient  : array[0..3] of glFloat = (1.0, 1.0, 1.0, 1.0); // And More Ambient
Light
type
 TFontSize      = record
                   fBoxX, fBoxY : single;
                  end;

 TRGB          = array[0..2] of byte;

var
 h_Wnd          : HWND;                    // Global window handle
 h_DC           : HDC;                    // Global device context
 h_RC           : HGLRC;                   // OpenGL rendering context

 gKeys          : array[0..255] of boolean;      // Holds keystrokes
 gFPSCount      : integer = 0;                // Counter for FPS
 gElapsedTime   : cardinal;                   // Elapsed time between frames

 gFontSizes          : array[0..1023] of TFontSize;       // Buffer for character size
information
 gFontList      : integer;                  // Handle to character display lists
 gTexture       : array[0..127, 0..127] of TRGB;  // Texture data
 gTextureHandle  : integer;                   // Handle to OpenGL texture

{$R *.RES}

{------------------------------------------------------------------------------}
{  Function to convert int to string. (No sysutils = smaller EXE) }
{------------------------------------------------------------------------------}
function IntToStr(pNum: integer): string; // Using SysUtils increase file size by
100K
begin
```

```
  Str(pNum, Result);
end; { IntToStr }

{------------------------------------------------------------------------------}
{ Function to make a texture from the pixel data }
{------------------------------------------------------------------------------}
procedure glBindTexture(target: GLenum; texture: GLuint); stdcall; external opengl32;
procedure glGenTextures(n: GLsizei; var textures: GLuint); stdcall; external opengl32;

function CreateTexture(pWidth, pHeight: integer; pData: pointer): glUint;
begin
  glGenTextures(1, Result);
  glBindTexture(GL_TEXTURE_2D, Result);
  glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
  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, pWidth, pHeight, 0, GL_RGB, GL_UNSIGNED_BYTE, pData);
end; { CreateTexture }

{------------------------------------------------------------------------------}
{ Function to draw the 3d font }
{------------------------------------------------------------------------------}
procedure glWrite3D(pText: string; pX, pY, pZ: single; pCenter: boolean);
var
  i  : integer;
  lX : single;
begin
  // Check empty string...
  if (pText = '') then Exit;

  lX:= pX;
```

```
if pCenter then
begin
  // Calculate the half width of the text...
  for i:= 1 to Length(pText) do lX:= lX - gFontSizes[Ord(pText[i])].fBoxX;
end;

glPushMatrix();

  // Center the text...
  glTranslatef(lX, pY, pZ);

  glEnable(GL_TEXTURE_2D);
  glBindTexture(GL_TEXTURE_2D, gTextureHandle);

  // Enable texture genarate...
  glEnable(GL_TEXTURE_GEN_S);
  glEnable(GL_TEXTURE_GEN_T);

    // Set the texture genarate mode, creates the texture coordinates...
    glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR);
    glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR);

    // Set the font list...
    glListBase(gFontList);

    glCallLists(Length(pText),     // Length of the string
            GL_UNSIGNED_BYTE,   // Chars, one byte each
            @pText[1]);        // Pointer to the first char of the string

  glDisable(GL_TEXTURE_GEN_S);
  glDisable(GL_TEXTURE_GEN_T);

glPopMatrix();
```

```
end; { glWrite3D }

{------------------------------------------------------------------------------}
{ Function to draw the actual scene }
{------------------------------------------------------------------------------}
procedure glDraw();
begin
 glClear(GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT);            // Clear The
Screen And The Depth Buffer
 glLoadIdentity();                                  // Reset The View

 glLightModelfv(GL_LIGHT_MODEL_AMBIENT, @gGlobalAmbient);        // Set The
Global Ambient Light Model

 glLightfv(GL_LIGHT0, GL_POSITION, @gLight0Pos);                  // Set The Lights
Position
 glLightfv(GL_LIGHT0, GL_AMBIENT,  @gLight0Ambient);           // Set The Ambient
Light
 glLightfv(GL_LIGHT0, GL_DIFFUSE,   @gLight0Diffuse);             // Set The Diffuse
Light
 glLightfv(GL_LIGHT0, GL_SPECULAR, @gLight0Specular);           // Set Up Specular
Lighting

 glEnable(GL_LIGHTING);                              // Enable Lighting
 glEnable(GL_LIGHT0);                               // Enable Light0

 glLightModelfv(GL_LIGHT_MODEL_AMBIENT, @gLModelAmbient);        // Set The
Ambient Light Model

 glTranslatef(0.0, -0.3, -4.0);

 glRotatef(Sin(GetTickCount() / 1000) * 60.0, 0.0, 1.0, 0.0);  // Rotate arount Y

 glWrite3D('Grafik',  0.7, -0.5, 0.0, true);
```

```
  glWrite3D('Komputer', -0.7,  0.5, 0.0, true);
end; { glDraw }

{-------------------------------------------------------------------------------}
{  Initialise OpenGL }
{-------------------------------------------------------------------------------}
procedure glInit();
const
  lColor : array[0..1] of TRGB = ((255, 255, 255), (255, 76, 100));
var
  i, j : integer;
begin
  glClearColor(0.0, 0.0, 0.0, 0.0);                    // Black Background
  glShadeModel(GL_SMOOTH);                    // Enables Smooth Color Shading
  glClearDepth(1.0);                          // Depth Buffer Setup
  glEnable(GL_DEPTH_TEST);                    // Enable Depth Buffer
  glDepthFunc(GL_LESS);                                // The Type Of Depth Test To
Do
  glEnable(GL_TEXTURE_2D);

  glHint(GL_PERSPECTIVE_CORRECTION_HINT,   GL_NICEST);          //   Realy   Nice
perspective calculations

  // Create a texture with 16 red and white rectangles...
  for i:= 0 to 127 do for j:= 0 to 127 do gTexture[i, j]:= lColor[(i shr 5 + j shr 5) and
1];

  gTextureHandle:= CreateTexture(128, 128, @gTexture);
end; { glInit }

{-------------------------------------------------------------------------------}
{  Handle window resize }
{-------------------------------------------------------------------------------}
procedure glResizeWnd(pWidth, pHeight: integer);
```

```
begin
  if (pHeight = 0) then pHeight:= 1;              // Prevent divide by zero exception
  glViewport(0, 0, pWidth, pHeight);              // Set the viewport for the OpenGL window
  glMatrixMode(GL_PROJECTION);                    // Change Matrix Mode to Projection
  glLoadIdentity();                               // Reset View
  gluPerspective(45.0, pWidth / pHeight, 1.0, 100.0);  // Do the perspective calculations. Last value = max clipping depth

  glMatrixMode(GL_MODELVIEW);                     // Return to the modelview matrix
  glLoadIdentity();                               // Reset View
end; { glResizeWnd }

{-------------------------------------------------------------------------------}
{ Processes all the keystrokes }
{-------------------------------------------------------------------------------}
procedure ProcessKeys();
begin
  if gKeys[32] then
  begin
    gKeys[32]:= false;
  end;
end; { ProcessKeys }

{-------------------------------------------------------------------------------}
{  Determines the application's response to the messages received}
{-------------------------------------------------------------------------------}
function WndProc(hWnd: HWND; Msg: UINT; wParam: WPARAM; lParam: LPARAM): LRESULT; stdcall;
begin
  Result:= 0;

  case (Msg) of
    WM_CREATE:
```

```
  begin
            // Insert stuff you want executed when the program starts
  end;
  WM_CLOSE:
  begin
   PostQuitMessage(0);
  end;
  WM_KEYDOWN:        // Set the pressed key (wparam) to equal true so we can
check if its pressed
  begin
   gKeys[wParam]:= true;
  end;
  WM_KEYUP:        // Set the released key (wparam) to equal false so we can check
if its pressed
  begin
   gKeys[wParam]:= false;
  end;
  WM_SIZE:        // Resize the window with the new width and height
  begin
   glResizeWnd(LOWORD(lParam),HIWORD(lParam));
  end;
  WM_MOUSEMOVE:
  begin
  end;
  WM_TIMER:        // Add code here for all timers to be used.
  begin
   if (wParam = FPS_TIMER) then
   begin
     // Calculate to get per Second incase intercal is less or greater than 1 second
    gFPSCount:= Round(gFPSCount * 1000 / FPS_INTERVAL);
    SetWindowText(h_Wnd, PChar(WND_TITLE + '    [' + intToStr(gFPSCount) + '
FPS]'));
    gFPSCount:= 0;
   end;
```

```
   end else          // Default result if nothing happens
   begin
     Result:= DefWindowProc(hWnd, Msg, wParam, lParam);
   end;
  end;
end; { WndProc }

{-------------------------------------------------------------------------------}
{ Properly destroys the window created at startup (no memory leaks) }
{-------------------------------------------------------------------------------}
procedure glKillWnd(Fullscreen: boolean);
begin
 if Fullscreen then            // Change back to non fullscreen
 begin
   ChangeDisplaySettings(devmode(nil^), 0);
   ShowCursor(True);
 end;

 // Makes current rendering context not current, and releases the device
 // context that is used by the rendering context.
 if not wglMakeCurrent(h_DC, 0) then
 begin
   MessageBox(0, 'Release of DC and RC failed!', 'Error', MB_OK or MB_ICONERROR);
 end;

 // Attempts to delete the rendering context
 if not wglDeleteContext(h_RC) then
 begin
   MessageBox(0, 'Release of rendering context failed!', 'Error', MB_OK or
MB_ICONERROR);
   h_RC:= 0;
 end;

 // Attemps to release the device context
```

```
 if ((h_DC = 1) and (ReleaseDC(h_Wnd, h_DC) <> 0)) then
 begin
  MessageBox(0, 'Release of device context failed!', 'Error', MB_OK or
MB_ICONERROR);
  h_DC:= 0;
 end;

 // Attempts to destroy the window
 if ((h_Wnd <> 0) and (not DestroyWindow(h_Wnd))) then
 begin
  MessageBox(0, 'Unable to destroy window!', 'Error', MB_OK or MB_ICONERROR);
  h_Wnd:= 0;
 end;

 // Attempts to unregister the window class
 if not UnRegisterClass('OpenGL', hInstance) then
 begin
  MessageBox(0, 'Unable to unregister window class!', 'Error', MB_OK or
MB_ICONERROR);
  hInstance:= 0;
 end;
end; { glKillWnd }

procedure glCreateFont();
var
 lFont : TFont;
begin
 lFont:= TFont.Create();

 lFont.Name:= 'Arial';              // Set the font name for the windows font

 SelectObject(h_DC, lFont.Handle);     // Select font device context

 // Create display lists for each character in the font:
```

```
 gFontList:= glGenLists(256);
 wglUseFontOutlines(h_DC,            // Device context of font -source-
              0,              // First character
              256,             // Number of characters
              gFontList,        // Handle of font display lists
              0.0,              // This is the sampling tolerance. Higher values create less
detailed outlines.
              0.2,             // This is the extrusion depth.
              WGL_FONT_POLYGONS,  // What kind of output? You can also use
WGL_FONT_LINES
              @gFontSizes);      // Array to store info about character sizes

 lFont.Free();
end; { glCreateFont }

{--------------------------------------------------------------------------------}
{  Creates the window and attaches a OpenGL rendering context to it }
{--------------------------------------------------------------------------------}
function glCreateWnd(Width, Height: integer; Fullscreen: boolean; PixelDepth:
integer): boolean;
var
 wndClass       : TWndClass;            // Window class
 dwStyle        : DWORD;               // Window styles
 dwExStyle      : DWORD;                // Extended window styles
 dmScreenSettings : DEVMODE;              // Screen settings (fullscreen, etc...)
 PixelFormat    : GLuint;               // Settings for the OpenGL rendering
 h_Instance     : HINST;                // Current instance
 pfd            : TPIXELFORMATDESCRIPTOR;  // Settings for the OpenGL window
begin
 h_Instance:= GetModuleHandle(nil);        // Grab An Instance For Our Window
 ZeroMemory(@wndClass, SizeOf(wndClass));        // Clear the window class
structure

 with wndClass do                        // Set up the window class
```

```
 begin
  style       := CS_HREDRAW or          // Redraws entire window if length changes
                 CS_VREDRAW or          // Redraws entire window if height changes
                 CS_OWNDC;              // Unique device context for the window
  lpfnWndProc   := @WndProc;                // Set the window procedure to our func
WndProc
  hInstance     := h_Instance;
  hCursor       := LoadCursor(0, IDC_ARROW);
  lpszClassName := 'OpenGL';
 end;

 if (RegisterClass(wndClass) = 0) then  // Attemp to register the window class
 begin
  MessageBox(0, 'Failed to register the window class!', 'Error', MB_OK or
MB_ICONERROR);
  Result:= false;
  Exit
 end;

 // Change to fullscreen if so desired
 if Fullscreen then
 begin
  ZeroMemory(@dmScreenSettings, SizeOf(dmScreenSettings));
  with dmScreenSettings do
  begin                             // Set parameters for the screen setting
   dmSize     := SizeOf(dmScreenSettings);
   dmPelsWidth := Width;                // Window width
   dmPelsHeight:= Height;               // Window height
   dmBitsPerPel:= PixelDepth;             // Window color depth
   dmFields    := DM_PELSWIDTH or DM_PELSHEIGHT or DM_BITSPERPEL;
  end;

  // Try to change screen mode to fullscreen
```

```
  if (ChangeDisplaySettings(dmScreenSettings, CDS_FULLSCREEN) =
DISP_CHANGE_FAILED) then
  begin
   MessageBox(0, 'Unable to switch to fullscreen!', 'Error', MB_OK or
MB_ICONERROR);
    Fullscreen:= false;
  end;
 end;

 // If we are still in fullscreen then
 if (Fullscreen) then
 begin
  dwStyle  := WS_POPUP       or       // Creates a popup window
           WS_CLIPCHILDREN or       // Doesn't draw within child windows
           WS_CLIPSIBLINGS;         // Doesn't draw within sibling windows
  dwExStyle:= WS_EX_APPWINDOW;         // Top level window
  ShowCursor(false);                  // Turn of the cursor (gets in the way)
 end else
 begin
  dwStyle  := WS_OVERLAPPEDWINDOW or    // Creates an overlapping window
           WS_CLIPCHILDREN    or    // Doesn't draw within child windows
           WS_CLIPSIBLINGS;         // Doesn't draw within sibling windows
  dwExStyle:= WS_EX_APPWINDOW    or    // Top level window
           WS_EX_WINDOWEDGE;       // Border with a raised edge
 end;

 // Attempt to create the actual window
 h_Wnd:= CreateWindowEx(dwExStyle,      // Extended window styles
                  'OpenGL',       // Class name
                  WND_TITLE,      // Window title (caption)
                  dwStyle,        // Window styles
                  0, 0,           // Window position
                  Width, Height,  // Size of window
                  0,              // No parent window
```

```
                0,              // No menu
                h_Instance,     // Instance
                nil);           // Pass nothing to WM_CREATE
 if (h_Wnd = 0) then
 begin
  glKillWnd(Fullscreen);              // Undo all the settings we've changed
  MessageBox(0, 'Unable to create window!', 'Error', MB_OK or MB_ICONERROR);
  Result:= false;
  Exit;
 end;

 // Try to get a device context
 h_DC:= GetDC(h_Wnd);
 if (h_DC = 0) then
 begin
  glKillWnd(Fullscreen);
  MessageBox(0, 'Unable to get a device context!', 'Error', MB_OK or
MB_ICONERROR);
  Result:= false;
  Exit;
 end;

 // Settings for the OpenGL window
 with pfd do
 begin
  nSize          := SizeOf(TPIXELFORMATDESCRIPTOR); // Size Of This Pixel Format
Descriptor
  nVersion       := 1;                    // The version of this data structure
  dwFlags        := PFD_DRAW_TO_WINDOW or          // Buffer supports drawing to
window
                PFD_SUPPORT_OPENGL or         // Buffer supports OpenGL drawing
                PFD_DOUBLEBUFFER;           // Supports double buffering
  iPixelType     := PFD_TYPE_RGBA;              // RGBA color format
  cColorBits     := PixelDepth;                // OpenGL color depth
```

```
  cRedBits       := 0;                    // Number of red bitplanes
  cRedShift      := 0;                    // Shift count for red bitplanes
  cGreenBits     := 0;                     // Number of green bitplanes
  cGreenShift    := 0;                     // Shift count for green bitplanes
  cBlueBits      := 0;                    // Number of blue bitplanes
  cBlueShift     := 0;                    // Shift count for blue bitplanes
  cAlphaBits     := 0;                     // Not supported
  cAlphaShift    := 0;                     // Not supported
  cAccumBits     := 0;                      // No accumulation buffer
  cAccumRedBits  := 0;                        // Number of red bits in a-buffer
  cAccumGreenBits := 0;                        // Number of green bits in a-buffer
  cAccumBlueBits  := 0;                       // Number of blue bits in a-buffer
  cAccumAlphaBits := 0;                        // Number of alpha bits in a-buffer
  cDepthBits     := 16;                     // Specifies the depth of the depth buffer
  cStencilBits   := 0;                     // Turn off stencil buffer
  cAuxBuffers    := 0;                      // Not supported
  iLayerType     := PFD_MAIN_PLANE;              // Ignored
  bReserved      := 0;                    // Number of overlay and underlay planes
  dwLayerMask    := 0;                       // Ignored
  dwVisibleMask  := 0;                       // Transparent color of underlay plane
  dwDamageMask   := 0;                         // Ignored
 end;

 // Attempts to find the pixel format supported by a device context that is the best
match to a given pixel format specification.
 PixelFormat:= ChoosePixelFormat(h_DC, @pfd);
 if (PixelFormat = 0) then
 begin
  glKillWnd(Fullscreen);
  MessageBox(0, 'Unable to find a suitable pixel format', 'Error', MB_OK or
MB_ICONERROR);
  Result:= false;
  Exit;
 end;
```

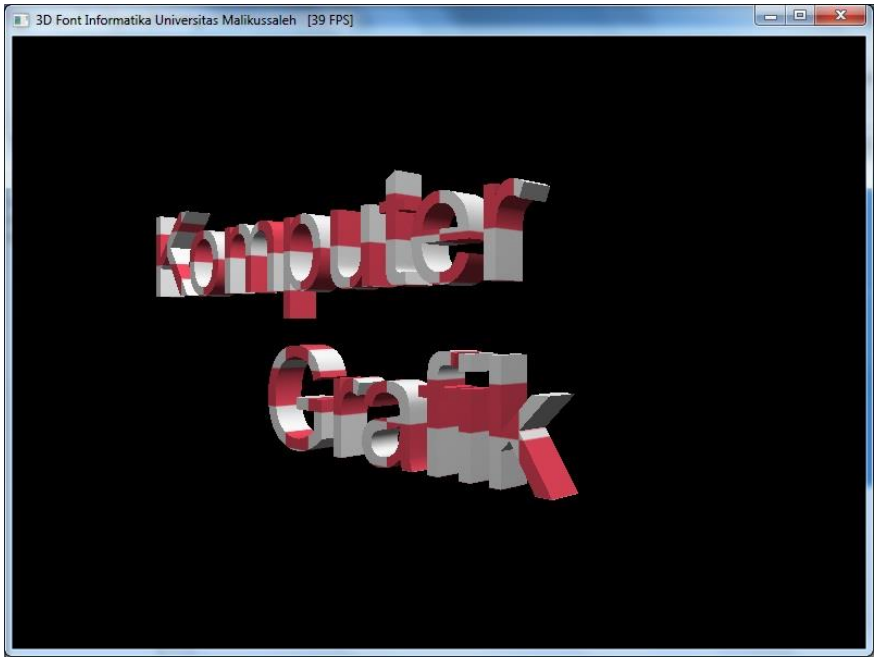
```
 // Sets the specified device context's pixel format to the format specified by the
PixelFormat.
 if (not SetPixelFormat(h_DC, PixelFormat, @pfd)) then
 begin
  glKillWnd(Fullscreen);
  MessageBox(0, 'Unable to set the pixel format', 'Error', MB_OK or
MB_ICONERROR);
  Result:= false;
  Exit;
 end;

 // Create a OpenGL rendering context
 h_RC:= wglCreateContext(h_DC);
 if (h_RC = 0) then
 begin
  glKillWnd(Fullscreen);
  MessageBox(0, 'Unable to create an OpenGL rendering context', 'Error', MB_OK or
MB_ICONERROR);
  Result:= false;
  Exit;
 end;

 // Makes the specified OpenGL rendering context the calling thread's current
rendering context
 if not wglMakeCurrent(h_DC, h_RC) then
 begin
  glKillWnd(Fullscreen);
  MessageBox(0, 'Unable to activate OpenGL rendering context', 'Error', MB_OK or
MB_ICONERROR);
  Result:= false;
  Exit;
 end;
```

```
  // Initializes the timer used to calculate the FPS
  SetTimer(h_Wnd, FPS_TIMER, FPS_INTERVAL, nil);

  // Settings to ensure that the window is the topmost window
  ShowWindow(h_Wnd, SW_SHOW);
  SetForegroundWindow(h_Wnd);
  SetFocus(h_Wnd);

  // Ensure the OpenGL window is resized properly
  glResizeWnd(Width, Height);
  glInit();

  // Create the 3d font
  glCreateFont();

  Result:= true;
end; { glCreateWnd }

{------------------------------------------------------------------------------}
{ Main message loop for the application}
{------------------------------------------------------------------------------}
function WinMain(hInstance: HINST; hPrevInstance: HINST;
                 lpCmdLine: PChar; nCmdShow: integer): integer; stdcall;
var
  lMsg                : TMsg;
  lFinished           : boolean;
  lDemoStart, lLastTime : DWord;
begin
  lFinished:= false;

  if not glCreateWnd(800, 600, FALSE, 32) then          // Perform application
initialization:
  begin
    Result:= 0;
```

```
  Exit;
 end;

 lDemoStart:= GetTickCount();                    // Get Time when demo started

 // Main message loop:
 while not lFinished do
 begin
  if PeekMessage(lMsg, 0, 0, 0, PM_REMOVE) then      // Check if there is a message
for this window
  begin
   if (lMsg.message = WM_QUIT) then                  // If WM_QUIT message received
then we are done
    begin
     lFinished:= true
    end else
    begin                                  // Else translate and dispatch the message to this
window
     TranslateMessage(lMsg);
     DispatchMessage(lMsg);
    end;
  end else
  begin
   Inc(gFPSCount);                             // Increment FPS Counter

   lLastTime  := gElapsedTime;
   gElapsedTime:= GetTickCount() - lDemoStart;     // Calculate Elapsed Time
   gElapsedTime:= (lLastTime + gElapsedTime) div 2; // Average it out for
smoother movement

   glDraw();                                  // Draw the scene
   SwapBuffers(h_DC);                             // Display the scene
```

```
    if gKeys[VK_ESCAPE] then                    // If user pressed ESC then set finised
TRUE
    begin
     lFinished:= true
    end else
    begin
     ProcessKeys();                    // Check for any other key Pressed
    end;
   end;
  end;
  glKillWnd(false);
  Result:= lMsg.wParam;
end; { WinMain }

begin
  WinMain(hInstance, hPrevInst, CmdLine, CmdShow);
end.
```

dan klik Save All pada menu utama File.
Tampilan eksekusi program akan berupa :



Bab 5

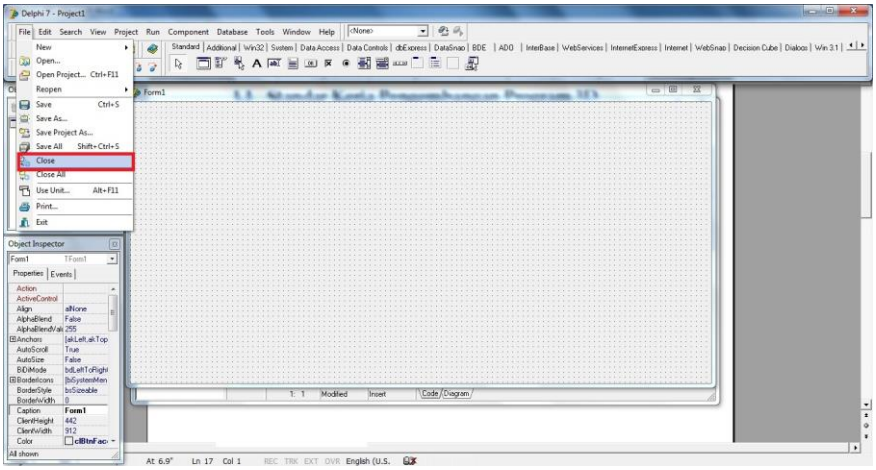
FISH DAN VISUAL AQUA

5.1 Persiapan Awal

Untuk pembangunan program animasi Fish dan air secara visual 3D, terlebih dahulu disiapkan beberapa buah file template gambar berformat bmp. Program yang dibangun akan menggunakan banyak source file.

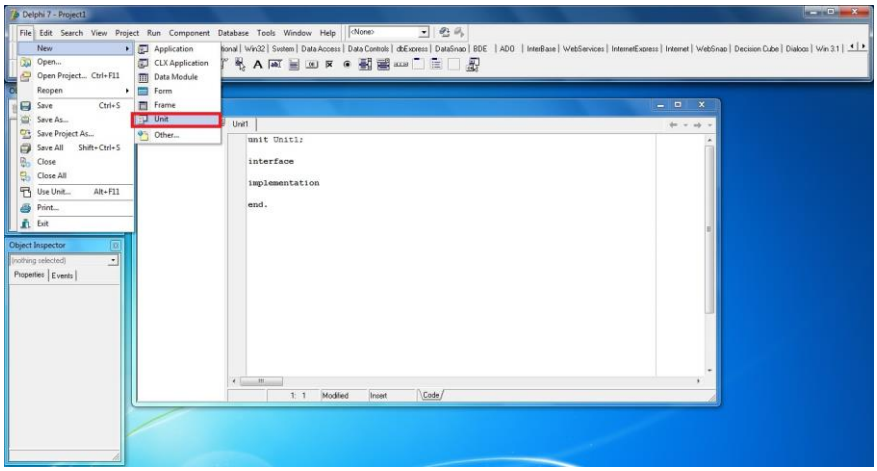
Sebelum masuk kepada penulisan koding program source file, maka setelah muncul IDE utama Delphi, lakukan langkah berikut :

File dan Close



5.2 Source File Animate

Untuk membangun Delphi Source File, maka sorot ke menu File, lalu sorot New, dan klik Unit.



Setelah muncul halaman koding, lalu tuliskan koding berikut :

```
unit Animate;

interface

uses
  OpenGL,
  Model,
  Dialogs,
  timer;

type
  TAnimate = class
  public
    AnimationFile : string;
    FramesPerSecond : double;
    Model : TModel;
    angle_speed : double;
    dist_speed : double;
```

```
    show_bones : boolean;
    mouse_x, mouse_y , mouse_b : integer;
    t : TPerfTimer;
            time : single;
            elapsedTime : single;
    canMultiTexture , isMultiTexture : Boolean;

    { Condtructor }
    constructor Create(iAnimationFile : string; iFramesPerSecond : double;
icanMultiTexture , iisMultiTexture : Boolean);
    { Destructor }
    destructor Destroy; override;

    { Render Te Model }
    procedure display(model : TModel);

    { Render Model and Calculate FPS }
    procedure render();

    { Set OpenGL Projection }
    procedure set_projection();
  end;
implementation

uses
  SysUtils;

{ TAnimate }

{--------------------------------------------------------------------}
{  Condtructor }
{--------------------------------------------------------------------}
constructor TAnimate.Create(iAnimationFile: string; iFramesPerSecond: double;
icanMultiTexture, iisMultiTexture : Boolean);
```

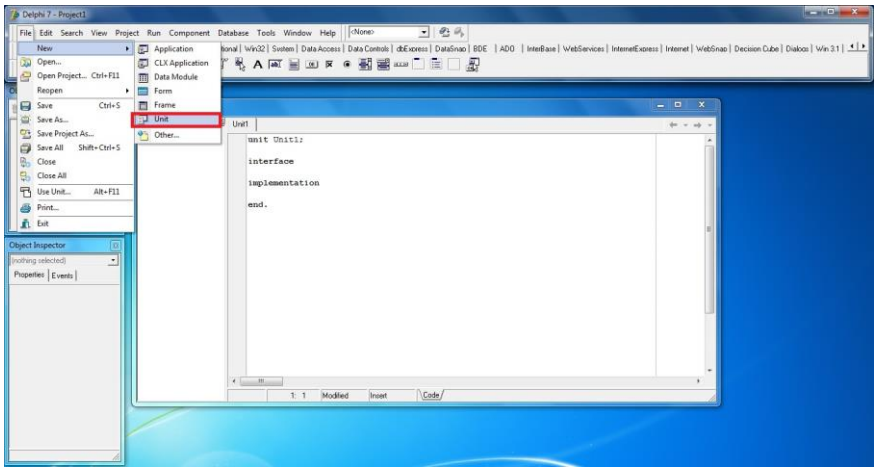
```
begin
  angle_speed := 0.5;
  dist_speed := 0.5;
  show_bones := true;
  AnimationFile := iAnimationFile;
  FramesPerSecond := iFramesPerSecond;

  canMultiTexture := icanMultiTexture;
  isMultiTexture := iisMultiTexture;
  model := TModel.create(canMultiTexture,isMultiTexture);

            if( not model.loadFromMs3dAsciiFile( AnimationFile ) ) then
                        ShowMessage( 'Couldn''t load the model' );

  t := TPerfTimer.create;
end;

{------------------------------------------------------------------}
{  Destructor }
{------------------------------------------------------------------}
destructor TAnimate.Destroy;
begin
  FreeAndNil(model);
  FreeAndNil(t);

  inherited Destroy;
end;

{------------------------------------------------------------------}
{  Set OpenGL Projection }
{------------------------------------------------------------------}
procedure TAnimate.set_projection;
begin
            glMatrixMode (GL_PROJECTION);
```

```
        glLoadIdentity ();
        glFrustum (-1.0, 1.0, -1.0, 1.0, 1.0, 1500.0);
end;

{------------------------------------------------------------------}
{  Render Te Model  }
{------------------------------------------------------------------}
procedure TAnimate.display(model: TModel);
begin
        model.render();                                  // render model
end;

{------------------------------------------------------------------}
{  Render Model and Calculate FPS }
{------------------------------------------------------------------}
procedure TAnimate.render();
begin
 glPushMatrix;
 glRotate(90,1.0,0.0,0.0);
 display(model);
 glPopMatrix;
 // advance the animation in accordance to
        // elapsed time since last call
        elapsedTime := t.getTime() - time;
        time := time + elapsedTime;
        model.advanceAnimation( elapsedTime * FramesPerSecond);
end;
end.
```

Lalu Save dengan nama file Animate.

5.3 Source File Boid

Untuk membangun Delphi Source File, maka sorot ke menu File, lalu sorot New, dan klik Unit.



Setelah muncul halaman koding, lalu tuliskan koding berikut :

```
unit Boid;
interface

uses
  Graphics,
  Math,
  Classes,
  OpenGL,
  Animate,
  SysUtils,
  Contnrs;

type
  TBoid = class;
  PBoid = ^TBoid;
  TBoidGene = class;
  T2DCoord = class
  public
```

```
  X, Y : single;
  constructor Create( iX, iY : single);
 end;
 TColor = class
  public
  r, g, b : single;
  constructor Create(red, green, blue : single);
 end;
 TFlock = class
  private
  { FIELDS }

    numberOfMembers : integer;                    // Number of critters on each
team.
    numberOfTeams : integer;                      // Number of teams.
    members : TObjectList;                        // The critters.
    meanHeading, meanSpeed, meanX, meanY : double;       // Flock averages.
    height, width : double;                       // The world.  (0,0) is in the upper left
corner.

    procedure checkBounds( boid : TBoid);
    procedure addToMomentum( b : TBoid);

    function FindAngle( FSourceX, FSourceY, FDestX, FDestY  : single) : single;
  public
    Skins : array of TAnimate;

    { CONSTRUCTOR }
    constructor Create(size, teams : integer; h, w : double; skin : string;
icanMultitexture, iisMultiTexture : boolean);

    { DESTRUCTOR }
    destructor Destroy; override;
```

```
{ METHODS }
procedure moveMembers;
{ UTILITY METHODS }

function calculateMeanX : double;
function calculateMeanY : double;

//procedure initializeFlock;
procedure initializeDuel( gene, challenger : TBoidGene);
procedure add( b : TBoid);
procedure addBoid(direction,
                  magnitude,
                  xPos,
                  yPos,
                  maxSpeed,
                  minSpeed,
                  angle,
                  accel,
                  sensor,
                  death,
                  deathC,
                  collision,
                  attackC,
                  retreatC: double;
                  iteamIndex: integer;
                  iTeamColorR,
                  iTeamColorG,
                  iTeamColorB : single);

procedure remove( b : TBoid);
function elements : TList;
procedure Render(ShowLines, ShowEnemyLines : boolean);
{ GET AND SET METHODS }
```

```
    function getNumberOfMembers : integer;
    procedure setNumberOfMembers( n : integer);

    function getNumberOfTeams : integer;
    procedure setNumberOfTeams( teams : integer);

    function getMeanHeading : double;
    procedure setMeanHeading( heading : double); // Priv?

    function getMeanSpeed : double;
    procedure setMeanSpeed( speed : double); // Priv?

    function getMeanX : double;
    procedure setMeanX( x : double); // Priv?

    function getMeanY : double;
    procedure setMeanY( y : double); // Priv?

    function getHeight : double;
    procedure setHeight( h : double);

    function getWidth : double;
    procedure setWidth( w : double);

end;
PFlock = ^TFlock;

TBoidGene = class
private

  { FIELDS }
  numberOfTrials : double;
  trialsWon :double;
```

```
  heading,
  speed,
  x,
  y,
  maximumSpeed,
  minimumSpeed,
  turnAngle,
  acceleration,
  sensorRange,
  collisionRange,
  deathRange,
  deathConstant,
  attackConstant,
  retreatConstant : double;
  color : TColor;
  flock : TFlock;
public

  teamIndex : integer;

  { CONSTRUCTORS }
  constructor Create( direction,
            magnitude,
            xPos,
            yPos,
            maxSpeed,
            minSpeed,
            angle,
            accel,
            sensor,
            death,
            deathC,
            collision,
            attackC,
```

```
        retreatC : double;
        c : TColor;
        gaggle : TFlock;
        iteamIndex : integer);

procedure setHeading( direction : double);
procedure setSpeed( magnitude : double);
procedure setX( xPos : double);
procedure setY( yPos : double);
procedure setMaximumSpeed( s : double);
procedure setMinimumSpeed( s : double);
procedure setColor( c : TColor);
procedure setTurnAngle( a : double);
procedure setAcceleration( a : double);
procedure setSensorRange( sensor : double);
procedure setDeathRange( death : double);
procedure setDeathConstant( c : double);
procedure setAttackConstant( c : double);
procedure setRetreatConstant( c : double);
procedure setCollisionRange( collision : double);
procedure setFlock( gaggle : TFlock);
procedure setNumberOfTrials( n : double);
procedure setTrialsWon( t : double);

{ FUNCTIONS }

function reproduceWith( o : TBoidGene) : TBoidGene;
function getNumberOfTrials : double;
function getTrialsWon : double;
function getFitness : double;
function getAcceleration: double;
function getAttackConstant: double;
function getCollisionRange: double;
function getColor: TColor;
```

```
  function getDeathConstant: double;
  function getDeathRange: double;
  function getFlock: TFlock;
  function getHeading: double;
  function getMaximumSpeed: double;
  function getMinimumSpeed: double;
  function getRetreatConstant: double;
  function getSensorRange: double;
  function getSpeed: double;
  function getTurnAngle: double;
  function getX: double;
  function getY: double;
end;

TBoid = class
private
 { FIELDS }

  heading,
  speed,
  x, y,
  maximumSpeed,
  minimumSpeed,
  turnAngle,
  acceleration,
  sensorRange,
  collisionRange,
  deathRange,
  deathRangeNumber,
  deathConstant,
  attackConstant,
  retreatConstant : double;
  friendsHeading, friendsSpeed, friendsX, friendsY, friendsNumber : double;
  enemiesHeading, enemiesSpeed, enemiesX, enemiesY, enemiesNumber : double;
```

```
  color : TColor;
  flock : TFlock;
  nearestFriend : TBoid; // Set only if within the collision range.
  nearestEnemy : TBoid;
public
  teamIndex : integer;
  { CONSTRUCTORS }
  constructor Create (direction, magnitude, xPos, yPos, maxSpeed, minSpeed,
                      angle, accel, sensor, death, deathC, attackC, retreatC,
                      collision : double;
                      Color : TColor ;
                      gaggle : TFlock); overload;
  constructor Create (b : TBoidGene); overload;

  { DESTRUCTOR }
  destructor Destroy; override;

  { PROCEDURES }
  procedure move;
  procedure slowDown;
  procedure speedUp;
  procedure turnLeft;
  procedure turnRight;
  procedure senseFlock(f : TFlock);
  procedure decide;
  procedure avoidCollision;
  procedure attackEnemies;
  procedure hangOutWithFriends;
  procedure retreat;
  procedure turnAway(xPos, yPos: double);
  procedure turnToward(xPos, yPos: double);
  procedure setHeading( direction : double);
  procedure setSpeed( magnitude : double);
  procedure setX( xPos : double);
```

```
    procedure setY( yPos : double);
    procedure setMaximumSpeed( s : double);
    procedure setMinimumSpeed( s : double);
    procedure setColor( c : TColor);
    procedure setTurnAngle( a : double);
    procedure setAcceleration( a : double);
    procedure setSensorRange( sensor : double);
    procedure setDeathRange( death : double);
    procedure setDeathConstant( c : double);
    procedure setAttackConstant( c : double);
    procedure setRetreatConstant( c : double);
    procedure setDeathRangeNumber( number : double);
    procedure setCollisionRange( collision : double);
    procedure setFlock( gaggle : TFlock);
    procedure setFriendsHeading( direction : double);
    procedure setFriendsSpeed( speed : double);
    procedure setFriendsX( xPos : double);
    procedure setFriendsY( yPos : double);
    procedure setFriendsNumber( number : double);
    procedure setEnemiesHeading( direction : double);
    procedure setEnemiesSpeed( speed : double);
    procedure setEnemiesX( xPos : double);
    procedure setEnemiesY(yPos : double );
    procedure setEnemiesNumber(number : double );
    procedure setNearestFriend(b : TBoid );
    procedure setNearestEnemy(b : TBoid );

  { FUNCTIONS }
    function isInRange(b : TBoid; range : double) : boolean;
    function isDead: boolean;
    function isFriend(b: TBoid): boolean;
    function relativeAngle(xPos, yPos : double) : double;
    function getAcceleration: double;
    function getAttackConstant: double;
```

```
    function getCollisionRange: double;
    function getColor: TColor;
    function getDeathConstant: double;
    function getDeathRange: double;
    function getDeathRangeNumber: double;
    function getEnemiesHeading: double;
    function getEnemiesNumber: double;
    function getEnemiesSpeed: double;
    function getEnemiesX: double;
    function getEnemiesY: double;
    function getFlock: TFlock;
    function getFriendsHeading: double;
    function getFriendsNumber: double;
    function getFriendsSpeed: double;
    function getFriendsX: double;
    function getFriendsY: double;
    function getHeading: double;
    function getMaximumSpeed: double;
    function getMinimumSpeed: double;
    function getNearestEnemy: TBoid;
    function getNearestFriend: TBoid;
    function getRetreatConstant: double;
    function getSensorRange: double;
    function getSpeed: double;
    function getTurnAngle: double;
    function getX: double;
    function getY: double;

    function reportStatistics : string;
end;
implementation

{ METHODS }
{ TBoid }
```

```
{-------------------------------------------------------------------}
{  Create a Boid }
{-------------------------------------------------------------------}

constructor TBoid.Create(direction, magnitude, xPos, yPos, maxSpeed,
  minSpeed, angle, accel, sensor, death, deathC, attackC, retreatC,
  collision: double; Color: TColor; gaggle: TFlock);
begin
   setMaximumSpeed(maxSpeed);
   setMinimumSpeed(minSpeed);
   setHeading(direction);
   setSpeed(magnitude);
   setX(xPos);
   setY(yPos);
   setColor(Color);
   setTurnAngle(angle);
   setAcceleration(accel);
   setSensorRange(sensor);
   setDeathRange(death);
   setDeathConstant(deathC);
   setAttackConstant(attackC);
   setRetreatConstant(retreatC);
   setCollisionRange(collision);
   setFlock(gaggle);
   setNearestFriend(nil);
   setNearestEnemy(nil);

   teamIndex := 0;
end;

{-------------------------------------------------------------------}
{  Create a Boid as a sibling of another Boid  }
{-------------------------------------------------------------------}
constructor TBoid.Create( b : TBoidGene );begin
```

```
 setMaximumSpeed(b.getMaximumSpeed());
 setMinimumSpeed(b.getMinimumSpeed());
 setHeading(b.getHeading());
 setSpeed(b.getSpeed());
 setX(b.getX());
 setY(b.getY());
 setColor(b.getColor());
 setTurnAngle(b.getTurnAngle());
 setAcceleration(b.getAcceleration());
 setSensorRange(b.getSensorRange());
 setDeathRange(b.getDeathRange());
 setDeathConstant(b.getDeathConstant());
 setAttackConstant(b.getAttackConstant());
 setRetreatConstant(b.getRetreatConstant());
 setCollisionRange(b.getCollisionRange());
 setFlock(b.getFlock());
 setNearestFriend(nil);
 setNearestEnemy(nil);
 teamIndex := b.teamIndex;
end;

{------------------------------------------------------------------}
{  Destroy the Boid }
{------------------------------------------------------------------}
destructor TBoid.Destroy;
begin
 // This is NOT the best place to release the colour, but at this point
 // the other location instances of the color no longer exist, and this
 // is the only remaining reference to the object, so if it is still here
 // we need to clean it up.
 if Assigned(Color) then
   Color.Free;
 inherited;
end;
```

```
{-------------------------------------------------------------------}
{  Move the Boid  }
{-------------------------------------------------------------------}
procedure TBoid.move;
begin
  setX(getX() + getSpeed()*Cos(getHeading()));
  setY(getY() + getSpeed()*Sin(getHeading()));
end;

{-------------------------------------------------------------------}
{ Slow Down (Never less than min speed) }
{-------------------------------------------------------------------}
procedure TBoid.slowDown;
begin
   setSpeed(getSpeed() - getAcceleration());
end;

{-------------------------------------------------------------------}
{  Speed Up (Never more than Max Speed) }
{-------------------------------------------------------------------}
procedure TBoid.speedUp;
begin
   setSpeed(getSpeed() + getAcceleration());
end;

{-------------------------------------------------------------------}
{  Turn Left  }
{-------------------------------------------------------------------}
procedure TBoid.turnLeft;
begin
   setHeading(getHeading() + getTurnAngle());
end;

{-------------------------------------------------------------------}
```

```
{  Turn Right }
{-------------------------------------------------------------------}
procedure TBoid.turnRight;
begin
   setHeading(getHeading() - getTurnAngle());
end;

{-------------------------------------------------------------------}
{  Is this Boid in range of a target Boid?  }
{-------------------------------------------------------------------}
function TBoid.isInRange(b : TBoid; range : double) : boolean;
begin
   Result  :=  (sqrt((b.getX()-getX())*(b.getX()-getX())  +  (b.getY()-getY())*(b.getY()-
getY())) < range);
end;

{-------------------------------------------------------------------}
{  Check the whole flock for friends and enemies close by }
{-------------------------------------------------------------------}
procedure TBoid.senseFlock(f : TFlock);
var b : TBoid;
   friends,
   totalFriendsX,
   totalFriendsY,
   totalFriendsSpeed,
   totalFriendsHeading,
   enemies,
   totalEnemiesX,
   totalEnemiesY,
   totalEnemiesSpeed,
   totalEnemiesHeading,
   distanceToFriend,
   distanceToEnemy : double;
   i : integer;
```

```
    e : TList;
  begin
    e := f.elements();
    friends := 0.0;
    totalFriendsX := 0.0;
    totalFriendsY := 0.0;
    totalFriendsSpeed := 0.0;
    totalFriendsHeading := 0.0;
    enemies := 0.0;
    totalEnemiesX := 0.0;
    totalEnemiesY := 0.0;
    totalEnemiesSpeed := 0.0;
    totalEnemiesHeading := 0.0;
    setNearestFriend(nil);
    setNearestEnemy(nil);
    setDeathRangeNumber(0.0);
    distanceToFriend := getCollisionRange();
    distanceToEnemy := getSensorRange();

    for i := 0 to e.Count - 1 do
    begin
      b := TBoid(e.Items[i]);
      if (b <> self) then
      begin
        if (isFriend(b)) then
        begin
          if (isInRange(b, distanceToFriend)) then
          begin
            setNearestFriend(b);
            distanceToFriend :=
              sqrt((b.getX()-getX())*(b.getX()-getX()) +
                   (b.getY()-getY())*(b.getY()-getY()));
          end;
          {if(distanceToFriend < getCollisionRange) then // Not correct
```

```
            begin
              turnAway(b.getX(),b.getY());
            end;}
            if (isInRange(b, getSensorRange())) then
            begin
              friends := friends + 1.0;
              totalFriendsX := totalFriendsX + b.getX();
              totalFriendsY := totalFriendsY + b.getY();
              totalFriendsSpeed := totalFriendsSpeed + b.getSpeed();
              totalFriendsHeading := totalFriendsHeading + b.getHeading();
            end;
          end
          else
          begin // Is enemy
            if (isInRange(b, distanceToEnemy)) then
            begin
              setNearestEnemy(b);
              distanceToEnemy :=
                sqrt((b.getX()-getX())*(b.getX()-getX()) +
                      (b.getY()-getY())*(b.getY()-getY()));
            end;
            {if(b.getCollisionRange() < distanceToEnemy) then // Kinda defeats the
point of hunting, but I cant think of another way to stop collisions
            begin
             slowDown();
            end;}
            if (isInRange(b, getDeathRange())) then
            begin
              setDeathRangeNumber(getDeathRangeNumber() + 1.0);
            end;
            if (isInRange(b, getSensorRange())) then
            begin
              enemies := enemies + 1.0;
              totalEnemiesX := totalEnemiesX + b.getX();
```

```
        totalEnemiesY := totalEnemiesY + b.getY();
        totalEnemiesSpeed := totalEnemiesSpeed + b.getSpeed();
        totalEnemiesHeading := totalEnemiesHeading + b.getHeading();
      end;
    end;
  end;
if (friends = 0.0) then
begin
  setFriendsNumber(0.0);
  setFriendsHeading(getHeading());
  setFriendsSpeed(getSpeed());
  setFriendsX(getX());
  setFriendsY(getY());
end
else
begin
  setFriendsNumber(friends);
  setFriendsHeading(totalFriendsHeading/friends);
  setFriendsSpeed(totalFriendsSpeed/friends);
  setFriendsX(totalFriendsX/friends);
  setFriendsY(totalFriendsY/friends);
end;

if (enemies = 0.0) then
begin
  setEnemiesNumber(0.0);
  setEnemiesHeading(getHeading());
  setEnemiesSpeed(getSpeed());
  setEnemiesX(getX());
  setEnemiesY(getY());
end
else
begin
  setEnemiesNumber(enemies);
```

```
        setEnemiesHeading(totalEnemiesHeading/enemies);
        setEnemiesSpeed(totalEnemiesSpeed/enemies);
        setEnemiesX(totalEnemiesX/enemies);
        setEnemiesY(totalEnemiesY/enemies);
      end;
   end;
end;

{---------------------------------------------------------------------}
{This is where the boid decides what to do based on what it senses. }
{---------------------------------------------------------------------}

procedure TBoid.decide;
begin
   if (getEnemiesNumber() <> 0.0) then
   begin
    if ((getFriendsNumber()+1)/getEnemiesNumber() > getAttackConstant()) then
    begin
       avoidCollision();
       attackEnemies();

    end
    else if ((getFriendsNumber()+1)/getEnemiesNumber() < getRetreatConstant())
then
    begin
       retreat();
    end;
   end
   else if (getNearestFriend() <> nil) then
   begin
     avoidCollision();
   end
   else
   begin
```

```
      hangOutWithFriends();
   end;
end;

{-----------------------------------------------------------------}
{  Try not to collide with others }
{-----------------------------------------------------------------}
procedure TBoid.avoidCollision;
var n : TBoid;
 angle : double;
begin
 n := getNearestEnemy();
 if (n <> nil) then
 begin
    turnAway(n.getX(),n.getY());
    angle := relativeAngle(n.getX(),n.getY());
    if (Cos(n.getHeading() - self.getHeading()) > 0) then  // going the same direction
    begin
       if (cos(angle - getHeading()) > 0) then          // this boid is behind the other
       begin
          // Added to avoid collision Not working Properly Yet!!!!
          if(sqrt((n.getX()-getX())*(n.getX()-getX())     +     (n.getY()-getY())*(n.getY()-
getY())) > n.collisionRange) then
          begin
            n.speedUp;
            slowDown;
            turnAway(n.getX(),n.getY());
          end;
          slowDown();
       end
       else
       begin                                    // this boid is in front of the other
          speedUp();
       end;
```

```
    end
    else
    begin                                   // going opposite directions
      if (Cos(angle - getHeading()) > 0) then          // head-on collision course
      begin
        slowDown();
      end;
    end;
  end;

  n := getNearestFriend();
  if (n <> nil) then
  begin
    turnAway(n.getX(),n.getY());
    angle := relativeAngle(n.getX(),n.getY());
    if (Cos(n.getHeading() - self.getHeading()) > 0) then  // going the same direction
    begin
      if (cos(angle - getHeading()) > 0) then          // this boid is behind the other
      begin
        // Added to avoid collision Not working Properly yet!!!!
        if(sqrt((n.getX()-getX())*(n.getX()-getX())     +     (n.getY()-getY())*(n.getY()-
getY())) > n.collisionRange) then
        begin
         //n.speedUp;
         slowDown;
         //turnAway(n.getX(),n.getY());
        end;
        slowDown();
      end
      else
      begin                                 // this boid is in front of the other
        speedUp();
      end;
    end
```

```
    else
    begin                                   // going opposite directions
      if (cos(angle - getHeading()) > 0) then          // head-on collision course
      begin
        slowDown();
      end;
    end;
  end;
end;

{-----------------------------------------------------------------}
{  Stay with Friendly Boids }
{-----------------------------------------------------------------}
procedure TBoid.hangOutWithFriends;
begin
  if (getFriendsNumber() > 0.0) then
  begin
    turnToward(getFriendsX(),getFriendsY());
    if (getFriendsSpeed() > getSpeed()) then
    begin
      speedUp();
    end
    else if (getFriendsSpeed() < getSpeed()) then
    begin
      slowDown();
    end;
  end
  else
  begin
    slowDown();
    if (random < 0.5) then
    begin
      turnLeft();
    end
```

```
      else
      begin
        turnRight();
      end;
    end;
end;

{-----------------------------------------------------------------}
{  Atack enemy boids to drive them away from the flock/school  }
{-----------------------------------------------------------------}
procedure TBoid.attackEnemies;
begin
   turnToward(getEnemiesX(),getEnemiesY());
   if (getEnemiesSpeed() > getSpeed()) then
   begin
      speedUp();
   end
   else if (getEnemiesSpeed() < getSpeed()) then
   begin
      slowDown();
   end;
end;

{-----------------------------------------------------------------}
{  Retreat from a fight                                  }
{-----------------------------------------------------------------}
procedure TBoid.retreat;
begin
   turnAway(getEnemiesX(),getEnemiesY());
   speedUp();
end;

{-----------------------------------------------------------------}
{  Get the relative angle from a Line                        }
```

```
{-----------------------------------------------------------------}
function TBoid.relativeAngle(xPos, yPos : double) : double;
var dx, dy, r, angle : double;
begin
   dx := xPos - getX();
   dy := yPos - getY();
   r := sqrt(dx*dx + dy*dy);
   angle := arccos(dx/r);
   if (dy < 0) then
   begin
      angle := 2.0*PI - angle;
   end;
   Result := angle;
end;

{-----------------------------------------------------------------}
{  Tunt away from a Specified Position                           }
{-----------------------------------------------------------------}
procedure TBoid.turnAway(xPos, yPos : double);
var angle : double;
begin
   if ((xPos = getX()) and (yPos = getY())) then
   begin
      exit;
   end;
   angle := relativeAngle(xPos,yPos);
   if (abs(angle-getHeading()) < PI)  then
   begin
      if (angle - getHeading() > 0) then
      begin
         turnRight();
      end
      else
      begin
```

```
        turnLeft();
      end;
    end
    else
    begin
      if (angle - getHeading() > 0) then
      begin
        turnLeft();
      end
      else
      begin
        turnRight();
      end;
    end;
end;

{-----------------------------------------------------------------------}
{  Turn Towards a Specified Position }
{-----------------------------------------------------------------------}
procedure TBoid.turnToward(xPos, yPos : double);
var angle : double;
begin
   //Do seperation calculation here.
   if ((xPos = getX()) and (yPos = getY())) then
   begin
      turnAway(xPos,yPos);
      exit;
   end;
   angle := relativeAngle(xPos,yPos);
   if (abs(angle - getHeading()) < PI) then
   begin
      if (angle - getHeading() > 0) then
      begin
         turnLeft();
```

```
      end
      else
      begin
        turnRight();
      end;
    end
    else
    begin
      if (angle - getHeading() > 0) then
      begin
        turnRight();
      end
      else
      begin
        turnLeft();
      end;
    end;
end;

{-----------------------------------------------------------------}
{  Examine boid colour to see if it is a friend  }
{-----------------------------------------------------------------}
function TBoid.isFriend(b : TBoid ) : boolean;
var PotentialFriendColor : TColor;
   SelfColor : TColor;
begin
   PotentialFriendColor := b.getColor;
   SelfColor := getColor;
   if (PotentialFriendColor.r = SelfColor.r) and (PotentialFriendColor.g = SelfColor.g)
and (PotentialFriendColor.b = SelfColor.b) then
     Result := true
   else
     Result := false;
end;
```

```
{------------------------------------------------------------------}
{  Is the current boid dead (Not yet fully Functional) }
{------------------------------------------------------------------}
function TBoid.isDead : boolean;
begin
   Result := (random < (1.0 - exp(-getDeathConstant()*getDeathRangeNumber())));
   {*
    * If death constant is 0.231049, then there is a 50% chance
    * of dying when confronted with three enemies.
    *}
end;

{  GET AND SET METHODS }

function TBoid.getHeading : double;
begin
  Result := heading;
end;
procedure TBoid.setHeading( direction : double);
begin
   heading := direction;
 if (heading > 2.0*PI) then
 begin
   heading := heading - 2.0*PI;
 end
 else
 begin
   if (heading < 0) then
   begin
      heading := 2.0*PI + heading;
   end;
 end;
end;
```

```
function TBoid.getSpeed : double;
begin
  Result := speed;
end;
procedure TBoid.setSpeed( magnitude : double);
begin
   speed := abs(magnitude);
   if (speed > getMaximumSpeed()) then
   begin
      speed := getMaximumSpeed();
   end;
   if (speed < getMinimumSpeed()) then
   begin
      speed := getMinimumSpeed();
   end;
end;

function TBoid.getX : double;
begin
  Result := x;
end;
procedure TBoid.setX( xPos : double);
begin
  x := xPos;
end;

function TBoid.getY : double;
begin
  Result := y;
end;
procedure TBoid.setY( yPos : double);
begin
  y := yPos;
end;
```

```
function TBoid.getMaximumSpeed : double;
begin
  Result := maximumSpeed;
end;
procedure TBoid.setMaximumSpeed( s : double);
begin
  maximumSpeed := abs(s);
end;

function TBoid.getMinimumSpeed : double;
begin
  Result := minimumSpeed;
end;
procedure TBoid.setMinimumSpeed( s : double);
begin
  minimumSpeed := abs(s);
end;

function TBoid.getColor : TColor;
begin
  Result := color;
end;
procedure TBoid.setColor( c : TColor);
begin
  color := c;
end;

function TBoid.getTurnAngle : double;
begin
  Result := turnAngle;
end;
procedure TBoid.setTurnAngle( a : double);
begin
    while (a>2.0*PI) do
```

```
   begin
      a := a - 2.0*PI;
   end;
   while (a<0) do
   begin
      a := 2.0*PI + a;
   end;
   turnAngle := a;
end;

function TBoid.getAcceleration : double;
begin
  Result := acceleration;
end;
procedure TBoid.setAcceleration( a : double);
begin
  acceleration := abs(a);
end;

function TBoid.getSensorRange : double;
begin
  Result := sensorRange;
end;
procedure TBoid.setSensorRange( sensor : double);
begin
  sensorRange := abs(sensor);
end;

function TBoid.getDeathRange : double;
begin
  Result := deathRange;
end;
procedure TBoid.setDeathRange( death : double);
begin
```

```
  deathRange := abs(death);
end;

function TBoid.getDeathConstant : double;
begin
  Result := deathConstant;
end;
procedure TBoid.setDeathConstant( c : double);
begin
  deathConstant := abs(c);
end;

function TBoid.getAttackConstant : double;
begin
  Result := attackConstant;
end;
procedure TBoid.setAttackConstant( c : double);
begin
  attackConstant := abs(c);
end;

function TBoid.getRetreatConstant : double;
begin
  Result := retreatConstant;
end;
procedure TBoid.setRetreatConstant( c : double);
begin
  retreatConstant := abs(c);
end;

function TBoid.getDeathRangeNumber : double;
begin
  Result := deathRangeNumber;
end;
```

```
procedure TBoid.setDeathRangeNumber( number : double);
begin
  deathRangeNumber := number;
end;

function TBoid.getCollisionRange : double;
begin
  Result := collisionRange;
end;
procedure TBoid.setCollisionRange( collision : double);
begin
  collisionRange := abs(collision);
end;

function TBoid.getFlock : TFlock;
begin
  Result := flock;
end;
procedure TBoid.setFlock( gaggle : TFlock);
begin
  flock := gaggle;
end;

function TBoid.getFriendsHeading : double;
begin
  Result := friendsHeading;
end;
procedure TBoid.setFriendsHeading( direction : double);
begin
  friendsHeading := direction;
end;

function TBoid.getFriendsSpeed : double;
begin
```

```
  Result := friendsSpeed;
end;
procedure TBoid.setFriendsSpeed( speed : double);
begin
  friendsSpeed := speed;
end;

function TBoid.getFriendsX : double;
begin
  Result := friendsX;
end;
procedure TBoid.setFriendsX( xPos : double);
begin
  friendsX := xPos;
end;

function TBoid.getFriendsY : double;
begin
  Result := friendsY;
end;
procedure TBoid.setFriendsY( yPos : double);
begin
  friendsY := yPos;
end;

function TBoid.getFriendsNumber : double;
begin
  Result := friendsNumber;
end;
procedure TBoid.setFriendsNumber( number : double);
begin
  friendsNumber := number;
end;
```

```
function TBoid.getEnemiesHeading : double;
begin
  Result := enemiesHeading;
end;
procedure TBoid.setEnemiesHeading( direction : double);
begin
  enemiesHeading := direction;
end;

function TBoid.getEnemiesSpeed : double;
begin
  Result := enemiesSpeed;
end;
procedure TBoid.setEnemiesSpeed( speed : double);
begin
  enemiesSpeed := speed;
end;

function TBoid.getEnemiesX : double;
begin
  Result := enemiesX;
end;
procedure TBoid.setEnemiesX( xPos : double);
begin
  enemiesX := xPos;
end;

function TBoid.getEnemiesY : double;
begin
  Result := enemiesY;
end;
procedure TBoid.setEnemiesY(yPos : double );
begin
  enemiesY := yPos;
```

```
end;

function TBoid.getEnemiesNumber : double;
begin
  Result := enemiesNumber;
end;
procedure TBoid.setEnemiesNumber(number : double );
begin
  enemiesNumber := number;
end;

function TBoid.getNearestFriend : TBoid;
begin
  Result := nearestFriend;
end;
procedure TBoid.setNearestFriend(b : TBoid );
begin
  nearestFriend := b;
end;

function TBoid.getNearestEnemy : TBoid;
begin
  Result := nearestEnemy;
end;
procedure TBoid.setNearestEnemy(b : TBoid );
begin
  nearestEnemy := b;
end;

{-----------------------------------------------------------------}
{  Function to report all the curent attributes of this boid }
{-----------------------------------------------------------------}
function TBoid.reportStatistics: string;
var stats : string;
```

```
begin
 stats := stats + 'Position : x - ' + FloatToStr(Round(x)) + ' y - '  +
FloatToStr(Round(y));
 stats := stats + ' Heading: ' + FloatToStr(getHeading);
 Result := stats;
end;

{ TBoidGene }
{--------------------------------------------------------------------}
{  Basic data class (Gene Memory) of a boid, }
{  Will pass this on to any Children of the parent boid  }
{--------------------------------------------------------------------}
constructor TBoidGene.Create(direction, magnitude, xPos, yPos, maxSpeed,
 minSpeed, angle, accel, sensor, death, deathC, collision, attackC,
 retreatC: double; c: TColor; gaggle: TFlock; iteamIndex : integer);
begin
 setMaximumSpeed(maxSpeed);
    setMinimumSpeed(minSpeed);
    setHeading(direction);
    setSpeed(magnitude);
    setX(xPos);
    setY(yPos);
    setColor(c);
    setTurnAngle(angle);
    setAcceleration(accel);
    setSensorRange(sensor);
    setDeathRange(death);
    setDeathConstant(deathC);
    setAttackConstant(attackC);
    setRetreatConstant(attackC);
    setCollisionRange(collision);
    setFlock(gaggle);
    setNumberOfTrials(0.0);
    setTrialsWon(0.0);
```

```
    self.teamIndex := iteamIndex
end;

function TBoidGene.getAcceleration: double;
begin
  Result := acceleration;
end;

function TBoidGene.getAttackConstant: double;
begin
  Result := attackConstant;
end;

function TBoidGene.getCollisionRange: double;
begin
  Result := collisionRange;
end;
function TBoidGene.getColor: TColor;
begin
  Result := color;
end;

function TBoidGene.getDeathConstant: double;
begin
  Result := deathConstant;
end;

function TBoidGene.getDeathRange: double;
begin
  Result := deathRange;
end;

function TBoidGene.getFitness: double;
begin
```

```
  Result := getTrialsWon()/getNumberOfTrials();
end;

function TBoidGene.getFlock: TFlock;
begin
  Result := flock;
end;

function TBoidGene.getHeading: double;
begin
  Result := heading;
end;

function TBoidGene.getMaximumSpeed: double;
begin
  Result := maximumSpeed;
end;
function TBoidGene.getMinimumSpeed: double;
begin
  Result := minimumSpeed;
end;

function TBoidGene.getNumberOfTrials: double;
begin
  Result := numberOfTrials;
end;

function TBoidGene.getRetreatConstant: double;
begin
  Result := retreatConstant;
end;

function TBoidGene.getSensorRange: double;
begin
```

```
  Result := sensorRange;
end;

function TBoidGene.getSpeed: double;
begin
  Result := speed;
end;

function TBoidGene.getTrialsWon: double;
begin
  Result := trialsWon;
end;

function TBoidGene.getTurnAngle: double;
begin
  Result := turnAngle;
end;
function TBoidGene.getX: double;
begin
  Result := x;
end;

function TBoidGene.getY: double;
begin
  Result := y;
end;

{-----------------------------------------------------------------}
{  Reproduction takes place }
{-----------------------------------------------------------------}
function TBoidGene.reproduceWith(o : TBoidGene): TBoidGene;
var direction, magnitude, xPos, yPos, maxSpeed, minSpeed, turnRate,
   accel, sRange, cRange, attackC, retreatC : double;
begin
```

```
if (random > 0.5) then
begin
   direction := o.getHeading();
end
else
begin
   direction := getHeading();
end;
if (random > 0.5) then
begin
   magnitude := o.getSpeed();
end
else
begin
   magnitude := getSpeed();
end;
if (random > 0.5) then
begin
   xPos := o.getX();
end
else
begin
   xPos := getX();
end;
if (random > 0.5) then
begin
   yPos := o.getY();
end
else
begin
   yPos := getY();
end;
if (random > 0.5) then
begin
```

```
   maxSpeed := o.getMaximumSpeed();
end
else
begin
   maxSpeed := getMaximumSpeed();
end;
if (random > 0.5) then
begin
   minSpeed := o.getMinimumSpeed();
end
else
begin
   minSpeed := getMinimumSpeed();
end;
if (minSpeed > maxSpeed) then
begin
   maxSpeed := minSpeed;
end;
if (random > 0.5) then
begin
   turnRate := o.getTurnAngle();
end
else
begin
   turnRate := getTurnAngle();
end;
if (random > 0.5) then
begin
   accel := o.getAcceleration();
end
else
begin
   accel := getAcceleration();
end;
```

```
if (random > 0.5) then
begin
   sRange := o.getSensorRange();
end
else
begin
   sRange := getSensorRange();
end;
if (random > 0.5) then
begin
   cRange := o.getCollisionRange();
end
else
begin
   cRange := getCollisionRange();
end;
if (random > 0.5) then
begin
   attackC := o.getAttackConstant();
end
else
begin
   attackC := getAttackConstant();
end;
if (random > 0.5) then
begin
   retreatC := o.getRetreatConstant();
end
else
begin
   retreatC := getRetreatConstant();
end;
Result := TBoidGene.Create(
            direction,
```

```
                magnitude,
                xPos,
                yPos,
                maxSpeed,
                minSpeed,
                turnRate,
                accel,
                sRange,
                getDeathRange(),
                getDeathConstant(),
                cRange,
                attackC,
                retreatC,
                TColor.Create(0.0,0.0,1.0),
                getFlock(),
                teamIndex);
end;

procedure TBoidGene.setAcceleration(a: double);
begin
 acceleration := a;
end;

procedure TBoidGene.setAttackConstant(c: double);
begin
 attackConstant := c;
end;

procedure TBoidGene.setCollisionRange(collision: double);
begin
 collisionRange := collision;
end;

procedure TBoidGene.setColor(c: TColor);
```

```
begin
  color := c;
end;

procedure TBoidGene.setDeathConstant(c: double);
begin
  deathConstant := c;
end;

procedure TBoidGene.setDeathRange(death: double);
begin
  deathRange := death;
end;

procedure TBoidGene.setFlock(gaggle: TFlock);
begin
  flock := gaggle;
end;

procedure TBoidGene.setHeading(direction: double);
begin
  heading := direction;
end;

procedure TBoidGene.setMaximumSpeed(s: double);
begin
  maximumSpeed := s;
end;

procedure TBoidGene.setMinimumSpeed(s: double);
begin
  minimumSpeed := s;
end;
```

```
procedure TBoidGene.setNumberOfTrials(n: double);
begin
 numberOfTrials := n;
end;

procedure TBoidGene.setRetreatConstant(c: double);
begin
 retreatConstant := c;
end;

procedure TBoidGene.setSensorRange(sensor: double);
begin
 sensorRange := sensor;
end;

procedure TBoidGene.setSpeed(magnitude: double);
begin
 speed := magnitude;
end;

procedure TBoidGene.setTrialsWon(t: double);
begin
 trialsWon := t;
end;

procedure TBoidGene.setTurnAngle(a: double);
begin
 turnAngle := a;
end;

procedure TBoidGene.setX(xPos: double);
begin
 x := xPos;
end;
```

```
procedure TBoidGene.setY(yPos: double);
begin
  y := yPos;
end;

{ TFlock }
{-------------------------------------------------------------------}
{  Holding Flock for all boids in our aquarium  }
{-------------------------------------------------------------------}
procedure TFlock.add(b: TBoid);
begin
  members.Add(b);
end;
procedure TFlock.addToMomentum(b: TBoid);
var x, y, r, angle : double;
begin
  x := getMeanSpeed()*cos(getMeanHeading()) + b.getSpeed()*cos(b.getHeading());
  y := getMeanSpeed()*sin(getMeanHeading()) + b.getSpeed()*sin(b.getHeading());
  r := sqrt(x*x + y*y);
  angle := arccos(x/r);
  if (y < 0) then
  begin
     angle := 2.0*PI - angle;
  end;
  setMeanSpeed(r);
  setMeanHeading(angle);
end;

function TFlock.calculateMeanX: double;
var e : TList;
   sum : double;
   i : integer;
begin
  e := elements();
```

```
 sum := 0;
 for i := 0 to e.Count - 1 do
 begin
  sum := sum + (TBoid(e.Items[i])).getX();
 end;
 Result := sum/getNumberOfMembers;
end;

function TFlock.calculateMeanY: double;
var e : TList;
   sum : double;
   i : integer;
begin
 e := elements();
 sum := 0;
 for i := 0 to e.Count - 1 do
 begin
  sum := sum + (TBoid(e.Items[i])).getY();
 end;
 Result := sum/getNumberOfMembers;
end;

{-----------------------------------------------------------------}
{  Wrap around when Boid goes out of bounds }
{-----------------------------------------------------------------}
procedure TFlock.checkBounds(boid: TBoid);
begin
 if (boid.getX() > width) then
 begin
    boid.setX(boid.getX() - width);
 end;
 if (boid.getX() < 0) then
 begin
    boid.setX(boid.getX() + width);
```

```
  end;
  if (boid.getY() > height) then
  begin
     boid.setY(boid.getY() - height);
  end;
  if (boid.getY() < 0) then
  begin
     boid.setY(boid.getY() + height);
  end;
end;

{--------------------------------------------------------------------}
{  Create the flock  }
{--------------------------------------------------------------------}
constructor TFlock.Create(size, teams: integer; h, w: double; skin : string;
icanMultitexture, iisMultiTexture : boolean);
var i : integer;
begin
  randomize;

  members := TObjectList.Create;
  members.OwnsObjects := True;

  setNumberOfMembers(size);
  setNumberOfTeams(teams);
  setHeight(h);
  setWidth(w);

  SetLength(Skins,teams);
  for i := 0 to teams-1 do
    Skins[i] := TAnimate.create(skin + IntToStr(i) + '.txt',24,icanMultitexture,
iisMultiTexture);
end;
```

```
function TFlock.elements: TList;
begin
  Result := members;
end;

function TFlock.getHeight: double;
begin
  Result := height;
end;

function TFlock.getMeanHeading: double;
begin
  Result := meanHeading;
end;

function TFlock.getMeanSpeed: double;
begin
  Result := meanSpeed;
end;

function TFlock.getMeanX: double;
begin
  Result := meanX;
end;

function TFlock.getMeanY: double;
begin
  Result := meanY;
end;

function TFlock.getNumberOfMembers: integer;
begin
   Result := members.Count -1;
end;
```

```
function TFlock.getNumberOfTeams: integer;
begin
  Result := numberOfTeams;
end;

function TFlock.getWidth: double;
begin
  Result := width;
end;

procedure TFlock.initializeDuel(gene, challenger: TBoidGene);
var i : integer;
begin
  members := TObjectList.Create;
  for i := 0 to getNumberOfMembers -1 do
  begin
     add(TBoid.Create(gene));
  end;
  for i := 0 to getNumberOfMembers -1 do
  begin
     add(TBoid.Create(challenger));
  end;
  setMeanX(calculateMeanX());
  setMeanY(calculateMeanY());
end;

{-----------------------------------------------------------------}
{  Calculate next movement for all boids }
{-----------------------------------------------------------------}
procedure TFlock.moveMembers;
var e : TList;
    boid : TBoid;
    colorSeen : TColor;
    i : integer;
```

```
begin
  e := elements();
  colorSeen := nil;
  setMeanHeading(0.0);
  setMeanSpeed(0.0);
  setMeanX(0.0);
  setMeanY(0.0);
  for i := 0 to e.Count -1 do
  begin
     boid := TBoid(e.Items[i]);
     if (getNumberOfTeams() > 1) then
     begin
        if (colorSeen = nil) then
        begin
           colorSeen := boid.getColor;
        end;
     end;
     boid.senseFlock(self);
     boid.decide();
     boid.move();
     checkBounds(boid);
     addToMomentum(boid);
     setMeanX(getMeanX() + boid.getX());
     setMeanY(getMeanY() + boid.getY());
  end;
  setMeanX(getMeanX()/getNumberOfMembers());
  setMeanY(getMeanY()/getNumberOfMembers());
end;

function TFlock.FindAngle( FSourceX, FSourceY, FDestX, FDestY  : single) : single;
var
  XDiff, YDiff: Single;
  fpAngle: Single;
  FSourcePoint , FDestPoint : T2DCoord;
```

```
begin
  FSourcePoint := T2DCoord.Create(FSourceX, FSourceY);
  FDestPoint   := T2DCoord.Create(FDestX, FDestY);
  fpAngle := 0;
  { If we have two points here, then we can work out the angle }
  if (FSourcePoint.X <> -1) AND (FSourcePoint.Y <> -1) AND (FDestPoint.X <> -1)
AND (FDestPoint.Y <> -1) then
  begin
    { Basic Trig:
              B
             /|       Angle ?  =  ArcTan( Y / X )
            / |
           /  |       Given any two points we can use the rule of
          /   |Y       opposite-over-adjacent to try to determine the
         /`   |       unknown angle.
        / ?`  _|
       /_____`__|_|       However, all of this maths is based on
      A     X            right-angled triangles, which obviously can only
                        account for a 90 degree segment at any point in
                        time, so we need to try to calculate which
                        quadrant of the full circle the angle represents.

     _____180_____
    |     |     |    So basing the quadrants around the co-ordinate
    | -x  | +x  |     system I have chosen in this particular case we
    | -y  | -y  |     find ourselves with the following grid.
  270|_____|_____|90
    |     |     |    As such we need to react to the locations the user
    | -x  | +x  |     has clicked in to be the basis of our maths
    | +y  | +y  |     calculations.  Working against this information,
    |_____|_____|     we can then calculate a correct angle.
          0                           }

    { All angles derived are is based from the X axis, and as such we need to
```

```
    do the maths to determine the actual angle }

  if (FDestPoint.X > FSourcePoint.X) AND (FDestPoint.Y > FSourcePoint.Y) then
  begin
    { 0 - 90 degree quadrant }
    XDiff := FDestPoint.X - FSourcePoint.X;
    YDiff := FDestPoint.Y - FSourcePoint.Y;

    { In this case the actual angle is:

      90 degrees minus the angle we calculate }
    fpAngle := 90 - RadToDeg(ArcTan(YDiff / XDiff));
  end else if (FDestPoint.X > FSourcePoint.X) AND (FDestPoint.Y <
FSourcePoint.Y) then
  begin
    { 90 - 180 degree quadrant }
    XDiff := FDestPoint.X - FSourcePoint.X;
    YDiff := FSourcePoint.Y - FDestPoint.Y;

    { In this case the actual angle is:

      90 degrees plus the angle we calculate }
    fpAngle := 90 + RadToDeg(ArcTan(YDiff / XDiff));
  end else if (FDestPoint.X < FSourcePoint.X) AND (FDestPoint.Y <
FSourcePoint.Y) then
  begin
    { 180 - 270 degree quadrant }
    XDiff := FSourcePoint.X - FDestPoint.X;
    YDiff := FSourcePoint.Y - FDestPoint.Y;

    { In this case the actual angle is:

      270 degrees minus the angle we calculate }
    fpAngle := 270 - RadToDeg(ArcTan(YDiff / XDiff));
```

```
    end else
    begin
     { 270 - 360/0 degree quadrant }
     XDiff := FSourcePoint.X - FDestPoint.X;
     YDiff := FDestPoint.Y - FSourcePoint.Y;

     { In this case the actual angle is:

       270 degrees plus the angle we calculate }
     fpAngle := 270 + RadToDeg(ArcTan(YDiff / XDiff));
    end;

    FreeAndNil(FSourcePoint);
    FreeAndNil(FDestPoint);

  end;
  Result := fpAngle;
end;

{-----------------------------------------------------------------}
{  Render the flock }
{-----------------------------------------------------------------}
procedure TFlock.Render(ShowLines, ShowEnemyLines : boolean);
var i : integer;
   e :TList;
   tempBoid : TBoid;
   xPoints : array [0..2] of single;
   yPoints : array [0..2] of single;
   rotangle : glfloat;
   LocalX, LocalY,PositionColor : single;
begin
  e := elements();
  for i := 0 to e.Count - 1 do
  begin
```

```
  tempBoid := TBoid(e.Items[i]);
  glColor3f(tempBoid.color.r,tempBoid.color.g,tempBoid.color.b);

  xPoints[0] := (tempBoid.getX()+20.0*cos(tempBoid.getHeading()))/100;
  yPoints[0] := (tempBoid.getY()+20.0*sin(tempBoid.getHeading()))/100;
  xPoints[1] := (tempBoid.getX()+5.0*cos(tempBoid.getHeading() +
(2.0/3.0)*PI))/100;
  yPoints[1] := (tempBoid.getY()+5.0*sin(tempBoid.getHeading() +
(2.0/3.0)*PI))/100;
  xPoints[2] := (tempBoid.getX()+5.0*cos(tempBoid.getHeading() -
(2.0/3.0)*PI))/100;
  yPoints[2] := (tempBoid.getY()+5.0*sin(tempBoid.getHeading() -
(2.0/3.0)*PI))/100;

  LocalX := tempBoid.getX()/100;
  LocalY := tempBoid.getY()/100;

  glPushMatrix;
   glTranslatef(LocalX,LocalY,0.0);
   glScalef(0.005,0.005,0.005);

   rotangle := -FindAngle(LocalX, LocalY, xPoints[0], yPoints[0]);

   glRotatef(rotangle ,0.0,0.0,1.0) ;
   PositionColor := 1.0-LocalY/10;
   glColor3f(PositionColor,PositionColor,PositionColor);

   Skins[tempBoid.teamIndex].render;
  glPopMatrix;

  if ShowLines then
  begin
   glBegin(GL_LINES);
    glDisable(GL_TEXTURE_2D);
```

```
      glColor3f(1.0,1.0,1.0);
      glVertex3f(tempBoid.x/100,tempBoid.y/100,0.0);
      glVertex3f(tempBoid.friendsX/100,tempBoid.friendsY/100,0.0);
     glEnd;
    end;
    if ShowEnemyLines then
    begin
     glBegin(GL_LINES);
      glColor3f(0.0,0.0,0.0);
      glVertex3f(tempBoid.x/100,tempBoid.y/100,0.0);
      glVertex3f(tempBoid.enemiesX/100,tempBoid.enemiesY/100,0.0);
     glEnd;
    end;
  end;
end;

procedure TFlock.remove(b: TBoid);
begin
  members.Remove(b)
end;

procedure TFlock.setHeight(h: double);
begin
  height := abs(h);
end;

procedure TFlock.setMeanHeading(heading: double);
begin
  meanHeading := heading;
end;

procedure TFlock.setMeanSpeed(speed: double);
begin
  meanSpeed := speed;
```

```
end;

procedure TFlock.setMeanX(x: double);
begin
 meanX := x;
end;

procedure TFlock.setMeanY(y: double);
begin
 meanY := y;
end;

procedure TFlock.setNumberOfMembers(n: integer);
begin
 numberOfMembers := abs(n);
end;

procedure TFlock.setNumberOfTeams(teams: integer);
begin
 numberOfTeams := abs(teams);
end;

procedure TFlock.setWidth(w: double);
begin
 width := abs(w);
end;

{------------------------------------------------------------------}
{  Destructor  }
{------------------------------------------------------------------}
destructor TFlock.Destroy;
var i : integer;
 ani: TAnimate;
begin
```

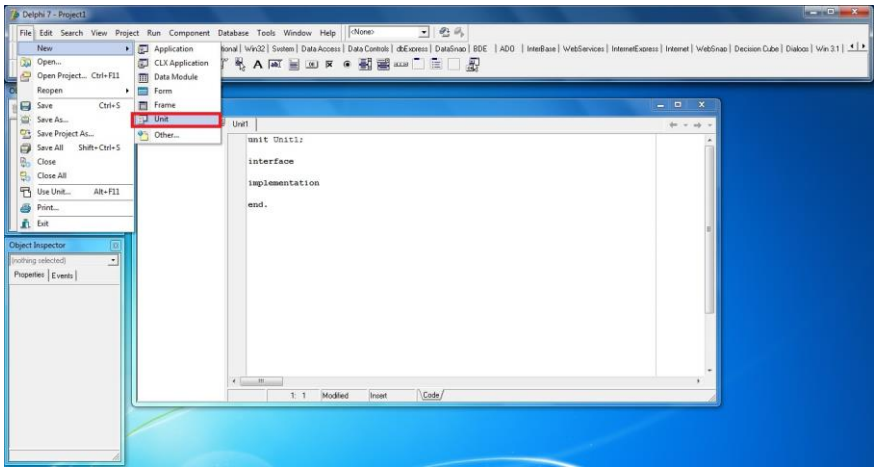
```
 FreeAndNil(members);

 for i := numberOfTeams - 1 downto 0 do
 begin
  ani := Skins[i];
  ani.Free;
  Skins[i] := Nil;
 end;

 inherited Destroy;
end;

procedure TFlock.addBoid(direction, magnitude, xPos, yPos, maxSpeed,
 minSpeed, angle, accel, sensor, death, deathC, collision, attackC,
 retreatC: double; iteamIndex: integer; iTeamColorR, iTeamColorG, iTeamColorB :
single);
var teamColor : TColor;
  gene : TBoidGene;
begin
 teamColor := TColor.Create(iTeamColorR, iTeamColorG, iTeamColorB);

  gene := TBoidGene.Create(
            2*random*PI,          // heading
            random(1),            // speed
            random*width,         // x position
            random*height,        // y position
            maxSpeed,             // maximum speed
            minSpeed,             // minimum speed
            angle,                // turning rate
            accel,                // acceleration
            sensor,               // sensor range
            death,                // death range
            deathC,               // death constant
            collision,            // collision range
```

```
                attackC,                // attack constant
                retreatC,               // retreat constant
                teamColor,              // color
                self,                   // This Boid
                iteamIndex);            // flock

  add(TBoid.Create(gene));

  FreeAndNil(gene);
end;
{ TColor }

constructor TColor.Create(red, green, blue: single);
begin
 r := red;
 g := green;
 b := blue;
end;

{ T2DCoord }

constructor T2DCoord.Create(iX, iY: single);
begin
 X := iX;
 Y := iY;
end;
end.
```

Lalu Save dengan nama file Boid.

5.4 Source File ExtOpenGL

Untuk membangun Delphi Source File, maka sorot ke menu File, lalu sorot New, dan klik Unit.



Setelah muncul halaman koding, lalu tuliskan koding berikut :

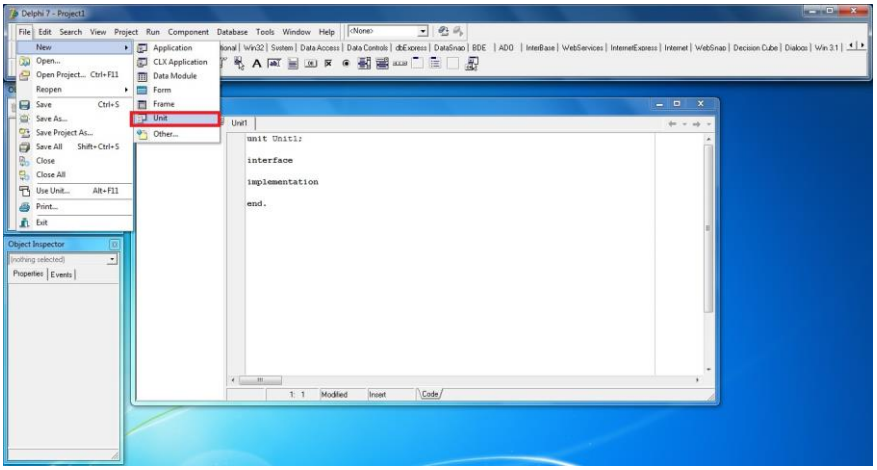
```
unit ExtOpenGL;
interface
uses OpenGL;

{------------------------------------------------------------------}
{  Extentions for standard Delphi OPENGL.PAS }
{------------------------------------------------------------------}

var
  glMultiTexCoord2fARB: procedure(target: GLenum; s, t: GLfloat); stdcall;
  glActiveTextureARB: procedure(target: GLenum); stdcall;

CONST
  GL_TEXTURE0_ARB  =  $84C0;
  GL_TEXTURE1_ARB  =  $84C1;
  GL_COMBINE_ARB      =  $8570;
  GL_RGB_SCALE_ARB =          $8573;
```

```
 GL_CLAMP_TO_EDGE =  $812F;

implementation
end.
```

Lalu Save dengan nama file ExtOpenGL.

5.5 Source File Matrix

Untuk membangun Delphi Source File, maka sorot ke menu File, lalu sorot New, dan klik Unit.



Setelah muncul halaman koding, lalu tuliskan koding berikut :

```
unit Matrix;

interface
type pdouble = ^double;
type TMatrix = class
        public
                { Constructor. }
                constructor create();

                { Set to identity. }
                procedure loadIdentity();
```

```
        { Set the values of the matrix. }
        procedure setMatrixValues( matrix : array of single);   {procedure
setMatrixValues( const float *matrix );}

        { Post-multiply by another matrix. }
        procedure postMultiply( var matrix : TMatrix );   {procedure
postMultiply( const Matrix& matrix );}

        { Set the translation of the current matrix. Will erase any previous
values. }
        procedure setTranslation( translation : array of single ); {procedure
setTranslation( const float *translation );}

        { Set the inverse translation of the current matrix. Will erase any
previous values. }
        procedure setInverseTranslation( translation : array of single);

        { Make a rotation matrix from Euler angles. The 4th row and column
are unmodified. }
        procedure setRotationRadians( angles : array of single );

        { Make a rotation matrix from Euler angles. The 4th row and column
are unmodified. }
        procedure setRotationDegrees( angles : array of single );

        { Make an inverted rotation matrix from Euler angles. The 4th row
and column are unmodified. }
        procedure setInverseRotationRadians( angles : array of single );

        { Make an inverted rotation matrix from Euler angles. The 4th row
and column are unmodified. }
        procedure setInverseRotationDegrees( angles : array of single );
```

```
                { Get the matrix data. }
                procedure getMatrix(var matrix : array of single); { return
m_matrix;}

  { Translate Vector }
                procedure translateVect( var pVect : array of single );

                { Rotate a vector by the inverse of the rotation part of this matrix. }
                procedure rotateVect( var pVect : array of single );

                { Translate a vector by the inverse of the translation part of this
matrix. }
                procedure inverseTranslateVect( var pVect : array of single );

                { Rotate a vector by the inverse of the rotation part of this matrix. }
                procedure inverseRotateVect( var pVect: array of single );

        private
                //      Matrix data, stored in column-major order
                m_matrix : array [0..15] of single;
end;

implementation

{ TMatrix }

{------------------------------------------------------------------}
{  Constructor.}
{------------------------------------------------------------------}
constructor TMatrix.create;
begin
 loadIdentity();
end;
```

```
{------------------------------------------------------------------}
{  Get the matrix data. }
{------------------------------------------------------------------}
procedure TMatrix.getMatrix(var matrix : array of single);
var i : integer;
begin
  {TODO : this might be dodge, maybe use m_matrix as public}
  for i := 0 to 15 do
   matrix[i] := m_matrix[i];
end;

{------------------------------------------------------------------}
{  Rotate a vector by the inverse of the rotation part of this matrix.}
{------------------------------------------------------------------}
procedure TMatrix.inverseRotateVect(var pVect: array of single);
var vec : array [0..2] of single;
begin
        vec[0]                                                          :=
pVect[0]*m_matrix[0]+pVect[1]*m_matrix[1]+pVect[2]*m_matrix[2];
        vec[1]                                                          :=
pVect[0]*m_matrix[4]+pVect[1]*m_matrix[5]+pVect[2]*m_matrix[6];
        vec[2]                                                          :=
pVect[0]*m_matrix[8]+pVect[1]*m_matrix[9]+pVect[2]*m_matrix[10];

  pVect[0] := vec[0];
  pVect[1] := vec[1];
  pVect[2] := vec[2];
end;

{------------------------------------------------------------------}
{  Set to identity. }
{------------------------------------------------------------------}
procedure TMatrix.loadIdentity;
begin
```

```
 m_matrix[0]  := 1;
 m_matrix[1]  := 0;
 m_matrix[2]  := 0;
 m_matrix[3]  := 0;
 m_matrix[4]  := 0;
 m_matrix[5]  := 1;
 m_matrix[6]  := 0;
 m_matrix[7]  := 0;
 m_matrix[8]  := 0;
 m_matrix[9]  := 0;
 m_matrix[10] := 1;
 m_matrix[11] := 0;
 m_matrix[12] := 0;
 m_matrix[13] := 0;
 m_matrix[14] := 0;
 m_matrix[15] := 1;
end;
{--------------------------------------------------------------------}
{  Post-multiply by another matrix. }
{--------------------------------------------------------------------}
procedure TMatrix.postMultiply(var matrix: TMatrix);
var newMatrix : array [0..15] of single;
begin

        newMatrix[0]        :=        m_matrix[0]*matrix.m_matrix[0]        +
m_matrix[4]*matrix.m_matrix[1] + m_matrix[8]*matrix.m_matrix[2];
        newMatrix[1]        :=        m_matrix[1]*matrix.m_matrix[0]        +
m_matrix[5]*matrix.m_matrix[1] + m_matrix[9]*matrix.m_matrix[2];
        newMatrix[2]        :=        m_matrix[2]*matrix.m_matrix[0]        +
m_matrix[6]*matrix.m_matrix[1] + m_matrix[10]*matrix.m_matrix[2];
        newMatrix[3] := 0;

        newMatrix[4]        :=        m_matrix[0]*matrix.m_matrix[4]        +
m_matrix[4]*matrix.m_matrix[5] + m_matrix[8]*matrix.m_matrix[6];
```

```
        newMatrix[5]  :=  m_matrix[1]*matrix.m_matrix[4]  +
m_matrix[5]*matrix.m_matrix[5] + m_matrix[9]*matrix.m_matrix[6];
        newMatrix[6]  :=  m_matrix[2]*matrix.m_matrix[4]  +
m_matrix[6]*matrix.m_matrix[5] + m_matrix[10]*matrix.m_matrix[6];
        newMatrix[7] := 0;

        newMatrix[8]  :=  m_matrix[0]*matrix.m_matrix[8]  +
m_matrix[4]*matrix.m_matrix[9] + m_matrix[8]*matrix.m_matrix[10];
        newMatrix[9]  :=  m_matrix[1]*matrix.m_matrix[8]  +
m_matrix[5]*matrix.m_matrix[9] + m_matrix[9]*matrix.m_matrix[10];
        newMatrix[10]  :=  m_matrix[2]*matrix.m_matrix[8]  +
m_matrix[6]*matrix.m_matrix[9] + m_matrix[10]*matrix.m_matrix[10];
        newMatrix[11] := 0;

        newMatrix[12]  :=  m_matrix[0]*matrix.m_matrix[12]  +
m_matrix[4]*matrix.m_matrix[13]  +  m_matrix[8]*matrix.m_matrix[14]  +
m_matrix[12];
        newMatrix[13]  :=  m_matrix[1]*matrix.m_matrix[12]  +
m_matrix[5]*matrix.m_matrix[13]  +  m_matrix[9]*matrix.m_matrix[14]  +
m_matrix[13];
        newMatrix[14]  :=  m_matrix[2]*matrix.m_matrix[12]  +
m_matrix[6]*matrix.m_matrix[13]  +  m_matrix[10]*matrix.m_matrix[14]  +
m_matrix[14];
        newMatrix[15] := 1;

        setMatrixValues( newMatrix );
end;

{-----------------------------------------------------------------}
{  Rotate a vector by the inverse of the rotation part of this matrix.}
{-----------------------------------------------------------------}
procedure TMatrix.rotateVect(var pVect : array of single);
var vec : array [0..2] of single;
begin
```

```
 vec[0] := pVect[0]*m_matrix[0]+pVect[1]*m_matrix[4]+pVect[2]*m_matrix[8];
        vec[1]                                                              :=
pVect[0]*m_matrix[1]+pVect[1]*m_matrix[5]+pVect[2]*m_matrix[9];
        vec[2]                                                              :=
pVect[0]*m_matrix[2]+pVect[1]*m_matrix[6]+pVect[2]*m_matrix[10];

 pVect[0] := vec[0];
 pVect[1] := vec[1];
 pVect[2] := vec[2];

end;
{-----------------------------------------------------------------}
{  Make an inverted rotation matrix from Euler angles.  }
{  The 4th row and column are unmodified. }
{-----------------------------------------------------------------}
procedure TMatrix.setInverseRotationDegrees(angles : array of single);
var vec : array [0..2] of single;
begin

        vec[0] := angles[0]*180.0/PI ;
        vec[1] := angles[1]*180.0/PI ;
        vec[2] := angles[2]*180.0/PI ;
        setInverseRotationRadians( vec );
end;

{-----------------------------------------------------------------}
{  Make an inverted rotation matrix from Euler angles. }
{  The 4th row and column are unmodified. }
{-----------------------------------------------------------------}
procedure TMatrix.setInverseRotationRadians(angles : array of single);
var cr , sr , cp , sp , cy , sy , srsp , crsp : single;
begin

 cr := cos( angles[0] );
```

```
        sr := sin( angles[0] );
        cp := cos( angles[1] );
        sp := sin( angles[1] );
        cy := cos( angles[2] );
        sy := sin( angles[2] );

        m_matrix[0] := cp*cy ;
        m_matrix[4] := cp*sy ;
        m_matrix[8] := -sp ;

        srsp := sr*sp;
        crsp := cr*sp;

        m_matrix[1] := srsp*cy-cr*sy ;
        m_matrix[5] := srsp*sy+cr*cy ;
        m_matrix[9] := sr*cp ;
        m_matrix[2] := crsp*cy+sr*sy ;
        m_matrix[6] := crsp*sy-sr*cy ;
        m_matrix[10] := cr*cp ;
end;

{-----------------------------------------------------------------------}
{  Set the inverse translation of the current matrix. }
{  Will erase any previous values. }
{-----------------------------------------------------------------------}
procedure TMatrix.setInverseTranslation(translation : array of single);
begin
  m_matrix[12] := -translation[0];
        m_matrix[13] := -translation[1];
        m_matrix[14] := -translation[2];
end;

{-----------------------------------------------------------------------}
{  Set the values of the matrix. }
```

```
{-------------------------------------------------------------------}
procedure TMatrix.setMatrixValues(matrix : array of single);
var i : integer;
begin
 for i := 0 to 15 do
   m_matrix[i] := matrix[i];

end;
{-------------------------------------------------------------------}
{  Make a rotation matrix from Euler angles. }
{  The 4th row and column are unmodified. }
{-------------------------------------------------------------------}
procedure TMatrix.setRotationDegrees(angles : array of single);
var vec : array [0..2] of single;
begin
          vec[0] := angles[0]*180.0/PI ;
          vec[1] := angles[1]*180.0/PI ;
          vec[2] := angles[2]*180.0/PI ;
          setRotationRadians( vec );
end;

{-------------------------------------------------------------------}
{  Make a rotation matrix from Euler angles. }
{  The 4th row and column are unmodified.}
{-------------------------------------------------------------------}
procedure TMatrix.setRotationRadians(angles : array of single);
var cr , sr , cp , sp , cy , sy , srsp , crsp : single;
begin

 cr := cos( angles[0] );
          sr := sin( angles[0] );
          cp := cos( angles[1] );
          sp := sin( angles[1] );
          cy := cos( angles[2] );
```

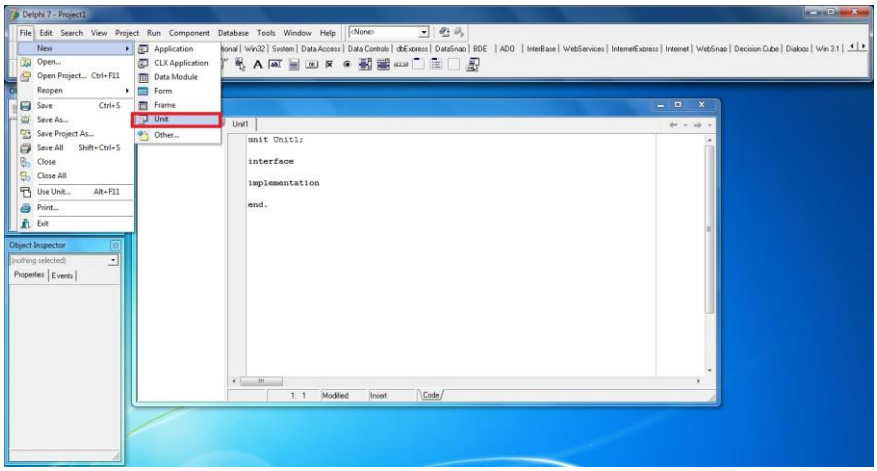
```
            sy := sin( angles[2] );

            m_matrix[0] := cp*cy ;
            m_matrix[1] := cp*sy ;
            m_matrix[2] := -sp ;

  if m_matrix[2] = -0 then
    m_matrix[2] := 0;

            srsp := sr*sp;
            crsp := cr*sp;

            m_matrix[4] := srsp*cy-cr*sy ;
            m_matrix[5] := srsp*sy+cr*cy ;
            m_matrix[6] := sr*cp ;
            m_matrix[8] := crsp*cy+sr*sy ;
            m_matrix[9] := crsp*sy-sr*cy ;
            m_matrix[10] := cr*cp ;
end;

{--------------------------------------------------------------------}
{  Set the translation of the current matrix. }
{  Will erase any previous values. }
{--------------------------------------------------------------------}
procedure TMatrix.setTranslation(translation : array of single);
begin

  m_matrix[12] := translation[0];
            m_matrix[13] := translation[1];
            m_matrix[14] := translation[2];
end;

{--------------------------------------------------------------------}
{  Translate Vector }
```

```
{-------------------------------------------------------------------}
procedure TMatrix.translateVect(var pVect : array of single);
begin

  pVect[0] := pVect[0]+m_matrix[12];
            pVect[1] := pVect[1]+m_matrix[13];
            pVect[2] := pVect[2]+m_matrix[14];
end;

{-------------------------------------------------------------------}
{  Translate a vector by the inverse }
{  of the translation part of this matrix. }
{-------------------------------------------------------------------}
procedure TMatrix.inverseTranslateVect(var pVect : array of single);
begin

  pVect[0] := pVect[0]-m_matrix[12];
            pVect[1] := pVect[1]-m_matrix[13];
            pVect[2] := pVect[2]-m_matrix[14];
end;

end.
```

Lalu Save dengan nama file Matrix.

5.6 Source File Model

Untuk membangun Delphi Source File, maka sorot ke menu File, lalu sorot New, dan klik Unit.



Setelah muncul halaman koding, lalu tuliskan koding berikut :

```
unit Model;

interface
uses
  OpenGL,
  ExtOpenGL,
  Matrix,
  Windows,
  Textures,
  SysUtils;

const
  MS_MAX_NAME = 32;
  MS_MAX_PATH = 256;

type
  TVec = class
  public
```

```
   x,y,z : single;
   w     : single;  // 4th homogenous coordinate
   u,v   : single;  // texture position
   bone : integer;          // index into Model bones array

   { Transform All 4 coorditates }
            procedure      transform( m : TMatrix );

   { Transform first 3 Coordinates }
   procedure     transform3( m : TMatrix );
 end;
type PVec = ^TVec;

type
   TTri = class
   public
    v : array [0..2] of integer;     // indices into vertices[]
              n : array [0..2] of integer;      // indices into normals[], normal per vertex
   end;
type PTri = ^TTri;

type
 TNormal = class
 public
          x,y,z : single;
 end;
type PNormal = ^TNormal;

type
 TShape = class
          public
   num_vertices : integer;
                    vertices : array of TVec;
```

```
                num_triangles : integer;
                triangles : array of TTri;

                num_normals : integer;
                normals : array of TNormal;

                {       Constructor. }
                constructor create();

  { Destructor }
  destructor Destroy; override;

  { Load an Milkshape 3D Ascii File Section }
                function loadFromMs3dAsciiSegment(var ifile : TextFile) : boolean;

  { Render Shape }
                procedure render();
 end;
type PShape = ^TShape;

type
 TMaterial = class
 public
  canMultiTexture, isMultiTexture : boolean;

  { Constructor. }
        constructor create(icanMultiTexture, iisMultiTexture : Boolean);

  { Load an Milkshape 3D Ascii File Section }
        function loadFromMs3dAsciiSegment(var ifile : TextFile ) : boolean;

  { Activate Materials And Textures}
        procedure activate();
```

```
  { Reload Textures}
          procedure reloadTexture();

 private
           Name : string;
          Ambient : array [0..3] of single;
          Diffuse : array [0..3] of single;
          Specular : array [0..3] of single;
          Emissive : array [0..3] of single;
          Shininess : single;
          Transparency : single;
          DiffuseTexture : string;
          AlphaTexture : string;
          textexture : GLuint;
   MultiTexture : array [0..31] of GLUint;
   CurrentTextureIndex : integer;
 end;
type PMaterial = ^TMaterial;
type
 TKeyFrame = class
          public
                   Time : single;
                   Value : array [0..2] of single;
 end;
type PKeyFrame = ^TKeyFrame;

type
 TBone = class
          public
   Name : string;
                   ParentName : string;
                   Parent : ^TBone;                                        //
pointer to parent bone (or NULL)
                   startPosition : array [0..2] of single;
```

```
   startRotation : array [0..2] of single;
                  m_relative : TMatrix;                    //          fixed
transformation matrix relative to parent
                  m_final : TMatrix;                       //    absolute    in
accordance to animation
                  NumPositionKeys : integer;
                  PositionKeyFrames : array of TKeyFrame;
                  NumRotationKeys : integer;
                  RotationKeyFrames : array of TKeyFrame;

   { Constructor }
   constructor Create();

   { Destructor }
   destructor Destroy; override;

   { Load an Milkshape 3D Ascii File Section }
                  function loadFromMs3dAsciiSegment(var ifile : TextFile ) : boolean;

   { Render Bones }
                  procedure render();

   { Not Implemented }
                  procedure fixup();

   { Advance Animation to Current time }
                  procedure advanceTo( CurrentTime : single );

   { Initialize Bone System }
                  procedure initialize();
 end;
type PBone = ^TBone;

type
```

```
TModel = class
        public

 MaxTime : single;                  // length of animation (= Frames)
                    CurrentTime : single;    // advancement within animation

                    num_shapes : integer;
                    shapes : array of TShape;
                    material_indices : array of integer;

                    num_materials : integer;
                    materials : array of TMaterial;

                    num_bones : integer;
                    bones :  array of TBone;

 canMultiTexture, isMultiTexture, isSphereMap : Boolean;
 { Constructor. }
                    constructor Create(icanMultiTexture, iisMultiTexture : Boolean);

 { Destructor }
 destructor Destroy; override;

 { Load a Milkshape 3D Ascii File }
                    function loadFromMs3dAsciiFile( filename : string ) : boolean;

 { Reload all Textures }
                    procedure       reloadTextures();

 { Render Model}
                    procedure       render();

 { Render Bones }
                    procedure       renderBones();
```

```
  { Link Child And Parent Bones }
                function linkBones() : boolean;
  { Initialize All The Bones }
                procedure       initializeBones();

  { Advance Animation To DeltaTime }
                procedure       advanceAnimation( deltaTime : single );

  { Attach The Mesh To The Bones }
                procedure       attachSkin();
 end;

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

{ TVec }

{-------------------------------------------------------------------}
{  Transform All 4 coorditates                          }
{-------------------------------------------------------------------}
procedure TVec.transform(m: TMatrix);
var matrix : array [0..15] of single;
   vector : array [0..3] of single;
begin

 m.getMatrix(matrix);

        vector[0] := x*matrix[0]+y*matrix[4]+z*matrix[8]+matrix[12];
        vector[1] := x*matrix[1]+y*matrix[5]+z*matrix[9]+matrix[13];
        vector[2] := x*matrix[2]+y*matrix[6]+z*matrix[10]+matrix[14];
        vector[3] := x*matrix[3]+y*matrix[7]+z*matrix[11]+matrix[15];

 x := vector[0];
```

```
        y := vector[1];
        z := vector[2];
        w := vector[3];

end;

{------------------------------------------------------------------}
{  Transform first 3 Coordinates   }
{------------------------------------------------------------------}
procedure TVec.transform3(m: TMatrix);
var matrix : array [0..15] of single;
   vec : array [0..2] of single;
begin
        m.getMatrix(matrix);

        vec[0] := x*matrix[0]+y*matrix[4]+z*matrix[8];
        vec[1] := x*matrix[1]+y*matrix[5]+z*matrix[9];
        vec[2] := x*matrix[2]+y*matrix[6]+z*matrix[10];

 x := vec[0];
 y := vec[1];
 z := vec[2];

end;

{ TShape }

{------------------------------------------------------------------}
{  Constructor.  }
{------------------------------------------------------------------}
constructor TShape.create;
begin
 num_vertices := 0;
        vertices := Nil;
```

```
            num_triangles := 0;
            triangles := Nil;
            num_normals := 0;
            normals := Nil;
end;

{---------------------------------------------------------------------}
{  Destructor  }
{---------------------------------------------------------------------}
destructor TShape.Destroy;
var
  I: Integer;
begin
  for I := (num_vertices - 1) downto 0 do
    if Assigned(vertices[I]) then
      vertices[I].Free;

  for I := (num_triangles - 1) downto 0 do
    if Assigned(triangles[I]) then
      triangles[I].Free;

  for I := (num_normals - 1) downto 0 do
    if Assigned(normals[I]) then
      normals[I].Free;

  inherited Destroy;
end;

{---------------------------------------------------------------------}
{  Render Shape  }
{---------------------------------------------------------------------}
procedure TShape.render;
var i,j : integer;
              tri : TTri;
```

```
            vec : TVec;
            N   : TNormal;
begin
//          glEnable(GL_LIGHTING);
            glBegin(GL_TRIANGLES);
  for i := 0 to num_triangles - 1           do// for each triangle
  begin
   tri := triangles[i];                                 // get next triangle to triangle

   for j := 0 to 2 do       // 3 vertices of the triangle
   begin
    N := normals[tri.n[j]];
    glNormal3f(N.x, N.y, N.z);  // set normal vector  (object space)

    vec := vertices[tri.v[j]];                  // pointer to vertex
    // Give OpenGL the current terrain texture coordinate for our height map
            glMultiTexCoord2fARB(GL_TEXTURE0_ARB, vec.u, vec.v);

            // Give OpenGL the current detail texture coordinate for our height map
    glMultiTexCoord2fARB(GL_TEXTURE1_ARB, vec.u, vec.v);
    //glTexCoord2f (vec.u, vec.v);                          // texture coordinate

    glVertex3f( vec.x, vec.y, vec.z );      // 3d coordinate (object space)
   end;
  end;
          glEnd();
end;

{-------------------------------------------------------------------}
{  Load an Milkshape 3D Ascii File Section  }
{-------------------------------------------------------------------}
function TShape.loadFromMs3dAsciiSegment(var ifile: TextFile): boolean;
var nFlags, nIndex, j : integer;
begin
```

```
  //
  // vertices
  //
  Readln(ifile,num_vertices);
  if eof(ifile) then
  begin
    result := false;
    exit;
  end;
  SetLength(vertices,num_vertices);
  if (num_vertices > 0)then
  begin

    for j := 0 to num_vertices - 1 do
    begin
      //init
      vertices[j] := TVec.Create;

      Readln(ifile,nFlags,vertices[j].x,          vertices[j].y,          vertices[j].z,vertices[j].u,
vertices[j].v,vertices[j].bone);

      if eof(ifile) then
      begin
        result := false;
        exit;
      end;

      // adjust the y direction of the texture coordinate
      vertices[j].v := 1.0 - vertices[j].v;
    end;
  end;

  //
  // normals
```

```
//
Readln(ifile,num_normals);
if eof(ifile) then
begin
 result := false;
 exit;
end;

SetLength(normals,num_normals);
if num_normals > 0 then
begin
 for j := 0 to num_normals - 1 do
 begin

  //init
  normals[j] := TNormal.Create;

  Readln(ifile,normals[j].x, normals[j].y, normals[j].z);
  if eof(ifile) then
  begin
   result := false;
   exit;
  end;

 end;
end;

//
// triangles
//
Readln(ifile,num_triangles);

if eof(ifile) then
begin
```

```
    result := false;
    exit;
  end;

  SetLength (triangles,num_triangles);

  for j := 0 to num_triangles - 1 do
  begin

    triangles[j] := TTri.Create;

    ReadLn(ifile,nFlags,triangles[j].v[0],                    triangles[j].v[1],
triangles[j].v[2],triangles[j].n[0], triangles[j].n[1], triangles[j].n[2],nIndex);
    if eof(ifile) then
    begin
      result := false;
      exit;
    end;

    assert(triangles[j].v[0] >= 0);
    assert(triangles[j].v[0] < num_vertices);
    assert(triangles[j].v[1] >= 0);
    assert(triangles[j].v[1] < num_vertices);
    assert(triangles[j].v[2] >= 0);
    assert(triangles[j].v[2] < num_vertices);
  end;

  result := true;
end;

{ TMaterial }

{-------------------------------------------------------------------}
{  Activate Materials And Textures }
```

```
{-----------------------------------------------------------------------}
procedure TMaterial.activate;
begin
 glMaterialfv( GL_FRONT, GL_AMBIENT, @Ambient );
          glMaterialfv( GL_FRONT, GL_DIFFUSE, @Diffuse );
          glMaterialfv( GL_FRONT, GL_SPECULAR, @Specular );
          glMaterialfv( GL_FRONT, GL_EMISSION, @Emissive );
          glMaterialf( GL_FRONT, GL_SHININESS, Shininess );

          if ( textexture > 0 ) then
          begin
  if(canMultiTexture) then
  begin
   glActiveTextureARB(GL_TEXTURE0_ARB);
   glBindTexture( GL_TEXTURE_2D, textexture );
   glEnable( GL_TEXTURE_2D );
   if(isMultiTexture) then
   begin
    glActiveTextureARB(GL_TEXTURE1_ARB);
    glBindTexture( GL_TEXTURE_2D,MultiTexture[CurrentTextureIndex]);
    glEnable( GL_TEXTURE_2D );
    CurrentTextureIndex := CurrentTextureIndex + 1;
    if(CurrentTextureIndex > 31) then
     CurrentTextureIndex := 0;
   end;
  end
  else
  begin
   glBindTexture( GL_TEXTURE_2D, textexture );
                    glEnable( GL_TEXTURE_2D );
  end;
          end
          else
                  glDisable( GL_TEXTURE_2D );
```

```
end;

{-----------------------------------------------------------------------}
{  Constructor.  }
{-----------------------------------------------------------------------}
constructor TMaterial.Create(icanMultiTexture, iisMultiTexture : Boolean);
begin
 // Only load these if we want to multitexture
 canMultiTexture := icanMultiTexture;
 isMultiTexture := iisMultiTexture;

 if(canMultiTexture and isMultiTexture) then
 begin
  LoadTexture('Caust00.jpg',MultiTexture[0],false);
  LoadTexture('Caust01.jpg',MultiTexture[1],false);
  LoadTexture('Caust02.jpg',MultiTexture[2],false);
  LoadTexture('Caust03.jpg',MultiTexture[3],false);
  LoadTexture('Caust04.jpg',MultiTexture[4],false);
  LoadTexture('Caust05.jpg',MultiTexture[5],false);
  LoadTexture('Caust06.jpg',MultiTexture[6],false);
  LoadTexture('Caust07.jpg',MultiTexture[7],false);
  LoadTexture('Caust08.jpg',MultiTexture[8],false);
  LoadTexture('Caust09.jpg',MultiTexture[9],false);
  LoadTexture('Caust10.jpg',MultiTexture[10],false);
  LoadTexture('Caust11.jpg',MultiTexture[11],false);
  LoadTexture('Caust12.jpg',MultiTexture[12],false);
  LoadTexture('Caust13.jpg',MultiTexture[13],false);
  LoadTexture('Caust14.jpg',MultiTexture[14],false);
  LoadTexture('Caust15.jpg',MultiTexture[15],false);
  LoadTexture('Caust16.jpg',MultiTexture[16],false);
  LoadTexture('Caust17.jpg',MultiTexture[17],false);
  LoadTexture('Caust18.jpg',MultiTexture[18],false);
  LoadTexture('Caust19.jpg',MultiTexture[19],false);
  LoadTexture('Caust20.jpg',MultiTexture[20],false);
```

```
   LoadTexture('Caust21.jpg',MultiTexture[21],false);
   LoadTexture('Caust22.jpg',MultiTexture[22],false);
   LoadTexture('Caust23.jpg',MultiTexture[23],false);
   LoadTexture('Caust24.jpg',MultiTexture[24],false);
   LoadTexture('Caust25.jpg',MultiTexture[25],false);
   LoadTexture('Caust26.jpg',MultiTexture[26],false);
   LoadTexture('Caust27.jpg',MultiTexture[27],false);
   LoadTexture('Caust28.jpg',MultiTexture[28],false);
   LoadTexture('Caust29.jpg',MultiTexture[29],false);
   LoadTexture('Caust30.jpg',MultiTexture[30],false);
   LoadTexture('Caust31.jpg',MultiTexture[31],false);
   CurrentTextureIndex := 0;
  end;
end;

{------------------------------------------------------------------------}
{  Load an Milkshape 3D Ascii File Section }
{------------------------------------------------------------------------}
function TMaterial.loadFromMs3dAsciiSegment(var ifile: TextFile): boolean;
var szLine : string;
begin
   // name
   if (eof(ifile)) then
   begin
                        result := false;
     exit;
   end;

   Readln(ifile,szLine);
   Name := StringReplace(szLine,'"','',[rfReplaceAll]);

   // ambient
   if (eof(ifile)) then
   begin
```

```
                result := false;
 exit;
end;

Readln(ifile,Ambient[0], Ambient[1], Ambient[2], Ambient[3]);

// diffuse
if (eof(ifile)) then
begin
                result := false;
 exit;
end;

Readln(ifile,Diffuse[0], Diffuse[1], Diffuse[2], Diffuse[3]);

// specular
if (eof(ifile)) then
begin
                result := false;
 exit;
end;

Readln(ifile,Specular[0], Specular[1], Specular[2], Specular[3]);

// emissive
if (eof(ifile)) then
begin
                result := false;
 exit;
end;

Readln(ifile,Emissive[0], Emissive[1], Emissive[2], Emissive[3]);
```

```
// shininess
if (eof(ifile)) then
begin
                    result := false;
 exit;
end;

Readln(ifile,Shininess);

// transparency
if (eof(ifile)) then
begin
                    result := false;
 exit;
end;

Readln(ifile,Transparency);

// diffuse texture
if (eof(ifile)) then
begin
                    result := false;
 exit;
end;

        DiffuseTexture := '';
Readln(ifile,DiffuseTexture);

DiffuseTexture := StringReplace(DiffuseTexture,'"','',[rfReplaceAll]);

// alpha texture
if (eof(ifile)) then
begin
                    result := false;
```

```
    exit;
   end;

   AlphaTexture := '';
   Readln(ifile,AlphaTexture);

   AlphaTexture := StringReplace(AlphaTexture,'"','',[rfReplaceAll]);

            reloadTexture();

            result := true;
end;

{-----------------------------------------------------------------------}
{  Reload Textures   }
{-----------------------------------------------------------------------}
procedure TMaterial.reloadTexture;
begin
 if( length(DiffuseTexture) > 0 ) then
   if( FileExists(DiffuseTexture)) then
    LoadTexture(DiffuseTexture, textexture,false)
   else
    LoadTexture('data\' + DiffuseTexture, textexture,false)
          else
                  textexture := 0;
end;

{ TBone }

{-----------------------------------------------------------------------}
{  Constructor   }
{-----------------------------------------------------------------------}
constructor TBone.create;
begin
```

```
 m_relative := TMatrix.create;                    // fixed transformation
matrix relative to parent
 m_final := TMatrix.create;                       // absolute in accordance to
animation
end;

{-------------------------------------------------------------------}
{  Destructor  }
{-------------------------------------------------------------------}
destructor TBone.Destroy;
var
 i: Integer;
begin
 for i := (NumRotationKeys -1) downto 0 do
  if Assigned(RotationKeyFrames[i]) then
   RotationKeyFrames[i].Free;

 for i := (NumPositionKeys - 1) downto 0 do
  if Assigned(PositionKeyFrames[i]) then
   PositionKeyFrames[i].Free;

 if Assigned(m_relative) then
  FreeAndNil(m_relative);
 if Assigned(m_final) then
  FreeAndNil(m_final);

 inherited Destroy;
end;

{-------------------------------------------------------------------}
{  Advance Animation to Current time  }
{-------------------------------------------------------------------}
procedure TBone.advanceTo(CurrentTime: single);
var
```

```
 i : integer;
 deltaTime : single;
 fraction : single;
 Position : array [0..2] of single;
 Rotation : array [0..2] of single;
 m_rel , m_frame : TMatrix;
 tempm : array [0..15] of single;
begin
 //
 // Position
 //

 // Find appropriate position key frame
 i := 0;
 while ( (i < NumPositionKeys-1) and (PositionKeyFrames[i].Time < CurrentTime)
) do
   i := i + 1;

 assert(i < NumPositionKeys);

 if( i > 0 ) then
 begin
  // Interpolate between 2 key frames

  // time between the 2 key frames
  deltaTime := PositionKeyFrames[i].Time - PositionKeyFrames[i-1].Time;

  assert( deltaTime > 0 );

  // relative position of interpolation point to the keyframes [0..1]
  fraction := (CurrentTime - PositionKeyFrames[i-1].Time) / deltaTime;

  assert( fraction > 0 );
  //assert( fraction < 1.0 );
```

```
  if(fraction > 1.0) then
    fraction := 1.0;
  Position[0] := PositionKeyFrames[i-1].Value[0] + fraction *
(PositionKeyFrames[i].Value[0] - PositionKeyFrames[i-1].Value[0]);
  Position[1] := PositionKeyFrames[i-1].Value[1] + fraction *
(PositionKeyFrames[i].Value[1] - PositionKeyFrames[i-1].Value[1]);
  Position[2] := PositionKeyFrames[i-1].Value[2] + fraction *
(PositionKeyFrames[i].Value[2] - PositionKeyFrames[i-1].Value[2]);
 end
 else
 begin
  Position[0] := PositionKeyFrames[i].Value[0];
  Position[1] := PositionKeyFrames[i].Value[1];
  Position[2] := PositionKeyFrames[i].Value[2];
 end;

 //
 // Rotation
 //

 // Find appropriate rotation key frame
 i := 0;
 while( (i < NumRotationKeys-1) and (RotationKeyFrames[i].Time < CurrentTime)
) do
      i := i + 1;

 assert(i < NumRotationKeys);

 if( i > 0 ) then
 begin
      // Interpolate between 2 key frames

      // time between the 2 key frames
      deltaTime := RotationKeyFrames[i].Time - RotationKeyFrames[i-1].Time;
```

```
      assert( deltaTime > 0 );

      // relative position of interpolation point to the keyframes [0..1]
      fraction := (CurrentTime - RotationKeyFrames[i-1].Time) / deltaTime;
      assert( fraction > 0 );
      //assert( fraction < 1.0 );
      if( fraction > 1.0) then
        fraction := 1.0;
      Rotation[0] := RotationKeyFrames[i-1].Value[0] + fraction *
(RotationKeyFrames[i].Value[0] - RotationKeyFrames[i-1].Value[0]);
      Rotation[1] := RotationKeyFrames[i-1].Value[1] + fraction *
(RotationKeyFrames[i].Value[1] - RotationKeyFrames[i-1].Value[1]);
      Rotation[2] := RotationKeyFrames[i-1].Value[2] + fraction *
(RotationKeyFrames[i].Value[2] - RotationKeyFrames[i-1].Value[2]);
    end
    else
    begin
      Rotation[0] := RotationKeyFrames[i].Value[0];
      Rotation[1] := RotationKeyFrames[i].Value[1];
      Rotation[2] := RotationKeyFrames[i].Value[2];
    end;

    // Now we know the position and rotation for this animation frame.
    // Let's calculate the transformation matrix (m_final) for this bone...

    m_rel := TMatrix.create;
    m_frame := TMatrix.create;

    // Create a transformation matrix from the position and rotation of this
    // joint in the rest position
    m_rel.setRotationRadians( startRotation );
    m_rel.setTranslation( startPosition );

    // Create a transformation matrix from the position and rotation
```

```
 // m_frame: additional transformation for this frame of the animation
 m_frame.setRotationRadians( Rotation );
 m_frame.setTranslation( Position );

 // Add the animation state to the rest position
 m_rel.postMultiply( m_frame );

 //
 if ( Parent = nil ) then                                    // this is the root node
 begin
  m_rel.getMatrix(tempm);
  m_final.setMatrixValues(tempm);     // m_final := m_rel
 end
 else                                                                  // not the root
node
 begin
  // m_final := parent's m_final * m_rel (matrix concatenation)
  Parent.m_final.getMatrix(tempm);
  m_final.setMatrixValues(tempm);
  m_final.postMultiply( m_rel );
 end;

 { >> Scotts Added Stuff }
 FreeAndNil(m_frame);
 FreeAndNil(m_rel);
end;

{-------------------------------------------------------------------}
{  Not Implemented  }
{-------------------------------------------------------------------}
procedure TBone.fixup;
begin
 {TODO : Look for the code.Is this inherent in c++?}
end;
```

```
{-------------------------------------------------------------------}
{  Initialize Bone System }
{-------------------------------------------------------------------}
procedure TBone.initialize;
var m_rel : TMatrix;
   tempm : array [0..15] of single;
begin
 m_rel := TMatrix.create;
 // Create a transformation matrix from the position and rotation

 m_rel.setRotationRadians( startRotation );
 m_rel.setTranslation( startPosition );

 // Each bone's final matrix is its relative matrix concatenated onto its
 // parent's final matrix (which in turn is ....)
 //
 if ( Parent = nil ) then
 begin
  m_rel.getMatrix(tempm);
  m_final.setMatrixValues(tempm);
 end
 else
 begin
  Parent.m_final.getMatrix(tempm);
  m_final.setMatrixValues(tempm);
  m_final.postMultiply( m_rel );
 end;

 { >> Scotts Added Stuff }
 FreeAndNil(m_rel);
end;

{-------------------------------------------------------------------}
{  Render Bones   }
```

```
{-----------------------------------------------------------------------}
procedure TBone.render;
var vector , parentvector : TVec;
begin
 vector := TVec.Create;
 parentvector := TVec.Create;

 vector.x := 0;
 vector.y := 0;
 vector.z := 0;
 vector.w := 1;
 vector.transform( m_final );

 if( Parent <> nil ) then
 begin
  parentvector.x := 0;
  parentvector.y := 0;
  parentvector.z := 0;
  parentvector.w := 1;
  parentvector.transform( Parent.m_final );
 end;

 glDisable( GL_TEXTURE_2D );

 // render bone as a line
 glLineWidth(1.0);
 glColor3f(1.0, 0, 0);
 glBegin(GL_LINES);
 glVertex3f( vector.x, vector.y, vector.z );
 if( Parent <> nil ) then
  glVertex3f( parentvector.x, parentvector.y, parentvector.z )
 else
  glVertex3f( vector.x, vector.y, vector.z );
```

```
 glEnd();

 // render bone-ends as fat points
 glPointSize(2.0);
 glColor3f(1.0, 0, 1.0);
 glBegin(GL_POINTS);
 glVertex3f( vector.x, vector.y, vector.z );
 if( Parent <> nil ) then
  glVertex3f( parentvector.x, parentvector.y, parentvector.z );

 glEnd();
 glColor3f(1.0, 1.0, 1.0);

 { >> Scotts Added Stuff }
 FreeAndNil(vector);
 FreeAndNil(parentvector);
end;

{--------------------------------------------------------------------}
{  Load an Milkshape 3D Ascii File Section  }
{--------------------------------------------------------------------}
function TBone.loadFromMs3dAsciiSegment(var ifile: TextFile): boolean;
var szLine : string;
   j ,nFlags : integer;
begin
   // name
   if (eof(ifile)) then
   begin
                        result := false;
    exit;
   end;

   Readln(ifile,Name);
```

```
Name := StringReplace(Name,'"','',[rfReplaceAll]) ;

// parent name
 if (eof(ifile)) then
begin
                    result := false;
  exit;
end;

ParentName := '';
Readln(ifile,ParentName);

ParentName := StringReplace(ParentName,'"','',[rfReplaceAll]) ;

// Start Position
          if (eof(ifile)) then
begin
                    result := false;
  exit;
end;
ReadLn(ifile,nFlags,
          startPosition[0], startPosition[1], startPosition[2],
          startRotation[0], startRotation[1], startRotation[2]);


// position key count
if (eof(ifile)) then
begin
                    result := false;
  exit;
end;
Readln(ifile,NumPositionKeys);

          SetLength(PositionKeyFrames,NumPositionKeys);
```

```
for j := 0 to NumPositionKeys - 1 do
begin

 PositionKeyFrames[j] := TKeyFrame.Create;

                    if (eof(ifile)) then
 begin
                     result := false;
  exit;
 end;

 Readln(ifile , PositionKeyFrames[j].Time,
                          PositionKeyFrames[j].Value[0],
                          PositionKeyFrames[j].Value[1],
                          PositionKeyFrames[j].Value[2] );

end;

// rotation key count
if (eof(ifile)) then
begin
                    result := false;
 exit;
end;

Readln(ifile,NumRotationKeys);

        SetLength(RotationKeyFrames,NumRotationKeys);

for j := 0 to NumRotationKeys -1 do
begin

                    RotationKeyFrames[j] :=  TKeyFrame.Create;
```

```
    if (eof(ifile)) then
    begin
                        result := false;
     exit;
    end;

     Readln(ifile , RotationKeyFrames[j].Time,
     RotationKeyFrames[j].Value[0],
                                RotationKeyFrames[j].Value[1],
                                RotationKeyFrames[j].Value[2] );
  end;
          result := true;
end;

{ TModel }

{--------------------------------------------------------------------}
{  Advance Animation To DeltaTime }
{--------------------------------------------------------------------}
procedure TModel.advanceAnimation(deltaTime: single);
var i : integer;
begin
          CurrentTime := CurrentTime + deltaTime;
          if( CurrentTime > MaxTime ) then       // this is a looped animation!
                    CurrentTime := 1.0;              // N.B. time starts at 1, not at 0!

 for i := 0 to num_bones -1 do
           bones[i].advanceTo( CurrentTime );

end;

{--------------------------------------------------------------------}
{  Attach The Mesh To The Bones }
```

```
{-------------------------------------------------------------------}
procedure TModel.attachSkin;
var i, j : integer;
        bone : integer;
        matrix : TMatrix;
        v : array [0..2] of single;
begin
        // Make vertex vector relative to the bone it's attached to

        for i := 0 to num_shapes - 1 do  // for all shapes
        begin
                for j := 0 to shapes[i].num_vertices - 1 do         // for all vertices
                begin
                        bone := shapes[i].vertices[j].bone;

                        if( bone <> -1 ) then      // if vertex attached to a bone
                        begin
                                matrix := bones[bone].m_final;

                                // make relative to bone position and orientation
                                // modifies vertex x,y,z

                                v[0] := shapes[i].vertices[j].x;
                                v[1] := shapes[i].vertices[j].y;
                                v[2] := shapes[i].vertices[j].z;
                                matrix.inverseTranslateVect( v );
                                matrix.inverseRotateVect( v );
                                shapes[i].vertices[j].x := v[0];
                                shapes[i].vertices[j].y := v[1];
                                shapes[i].vertices[j].z := v[2];
                        end;

                end;
        end;
```

```
end;

{-----------------------------------------------------------------------}
{  Constructor. }
{-----------------------------------------------------------------------}
constructor TModel.Create(icanMultiTexture , iisMultiTexture : Boolean);
begin
 num_shapes := 0;
 shapes := nil;
 num_materials := 0;
 materials := nil;

 canMultiTexture := icanMultiTexture;
 isMultiTexture := iisMultiTexture;

 if (icanMultiTexture and isMultiTexture) then
 begin
  glMultiTexCoord2fARB := wglGetProcAddress('glMultiTexCoord2fARB');
  glActiveTextureARB := wglGetProcAddress('glActiveTextureARB');
 end;

end;
{-----------------------------------------------------------------------}
{  Destructor   }
{-----------------------------------------------------------------------}
destructor TModel.Destroy;
var
 i: Integer;
begin
 for i := (num_bones -1) downto 0 do
  if Assigned(bones[i]) then
   bones[i].Free;

 for i := (num_materials - 1) downto 0 do
```

```
   if Assigned(materials[i]) then
    materials[i].Free;

  for i := (num_shapes - 1) downto 0 do
   if Assigned(shapes[i]) then
    shapes[i].Free;

  inherited Destroy;
end;

{--------------------------------------------------------------------}
{  Initialize All The Bones   }
{--------------------------------------------------------------------}

procedure TModel.initializeBones;
var i : integer;
begin
  for i := 0 to num_bones-1 do
                                bones[i].initialize;
end;

{--------------------------------------------------------------------}
{  Link Child And Parent Bones  }
{--------------------------------------------------------------------}
function TModel.linkBones: boolean;
var i, j : integer;
begin
  // The relationship between child and parent bones are defined
          // by each child bone providing the name of its parent bone.
          // This function builds corresponding pointers for faster lookup.

          // Link children to parent
  for i := 0 to num_bones-1 do
          begin
```

```
                bones[i].Parent := nil;

                if( length(bones[i].ParentName) >  0 ) then    // does bone have
parent?
                begin
                        for j := 0 to num_bones -1 do // search for parent
                        begin
                                if(  bones[j].Name  =  bones[i].ParentName)  then
        // j is parent of i
                                begin
                                        bones[i].Parent := @bones[j];
                                        break;
                                end;
                        end;
                        if ( bones[i].Parent = nil) then // Unable to find parent bone
    begin
     result := false;
     exit;
    end;
   end;
 end;
        result := true;
end;

{--------------------------------------------------------------------}
{  Load a Milkshape 3D Ascii File }
{--------------------------------------------------------------------}
function TModel.loadFromMs3dAsciiFile(filename: string): boolean;
var StartTime : single;
   bError : bool;
   szLine , szName , strTemp : string;
   nFrame, nFlags, nIndex, i : integer;
   fModel :  TextFile;
```

```
begin
 bError := false;

 AssignFile(fModel,filename);
 Reset(fModel);

          CurrentTime := 0;

 while ((not eof(fModel)) and (not bError)) do
 begin
  Readln(fmodel,szLine);
  if ( copy(szLine,0,2) = '//') then
    continue;

  if szLine = '' then
     continue;

  if (pos('Frames:',szLine) = 1) then
  begin
   nFrame := StrToInt(copy(szLine,8,length(szLine)));
                                MaxTime := 1.0 * nFrame;                    // length of animation
   continue;
  end;

  if (pos('Frame:',szLine) = 1) then
  begin
   nFrame := StrToInt(copy(szLine,7,length(szLine)));
                     StartTime := 1.0 * nFrame;
   continue;
  end;

  if (pos('Meshes: ',szLine) = 1) then
  begin
   num_shapes := StrToInt(StringReplace(szLine,'Meshes: ','',[rfReplaceAll]));
```

```
    SetLength(shapes,num_shapes);
    SetLength(material_indices, num_shapes);
    for i := 0 to num_shapes - 1 do
    begin
     shapes[i] := TShape.create;
     material_indices[num_shapes-1] := 0;
    end;

    for i := 0 to num_shapes - 1 do
    begin

      // mesh: name, flags, material index
      Readln(fmodel,szName);
      if (eof(fmodel)) then
      begin
        bError := true;
        break;
      end;

      //remove "
      szName := StringReplace(szName,'"','',[rfReplaceAll]);

      //get nFlags
      strTemp := szName;
      strTemp := copy(strTemp,pos(' ',strTemp)+1,length(strTemp));
      nFlags := StrToInt(copy(strTemp,0,pos(' ',strTemp)-1));
      //get nIndex
      nIndex := StrToInt(copy(strTemp,pos(' ',strTemp)+1,length(strTemp)));

                                material_indices[i] := nIndex;
                                if(                                    not
shapes[i].loadFromMs3dAsciiSegment(fModel) ) then
      begin
        bError := true;
```

```
          break;
        end;
                              end;
    continue;
                    end;

  //
  // materials
  //
  if (pos('Materials: ',szLine) = 1) then
  begin

    num_materials := StrToInt(StringReplace(szLine,'Materials: ','',[rfReplaceAll]));

    SetLength(materials,num_materials);

    // Init materials
    for i := 0 to num_materials -1 do
      materials[i] := TMaterial.create(canMultiTexture,isMultiTexture);

    for i := 0 to num_materials -1 do
    begin
                                        if(                                                    not
materials[i].loadFromMs3dAsciiSegment(fModel) ) then
      begin
        bError := true;
        break;
      end;
                              end;
    continue;
  end;

                    //
  // bones
```

```
 //
 if (pos('Bones: ',szLine) = 1) then
 begin

    num_bones := StrToInt(StringReplace(szLine,'Bones: ','',[rfReplaceAll]));

    SetLength(bones,num_bones);

    // Initialize Bones
    for i := 0 to num_bones -1 do
     bones[i] := TBone.create;

    for i := 0 to num_bones -1 do
    begin

       if( not bones[i].loadFromMs3dAsciiSegment(fModel) ) then
       begin
          bError := true;
          break;
       end;
    end;
    continue;
 end;
end;

          CloseFile(fModel);

          if( not linkBones() ) then
begin
 result := false;
 exit;
end;
          initializeBones();
          attachSkin();
```

```
        advanceAnimation( StartTime );
        result := true;
end;

{-------------------------------------------------------------------}
{  Reload all Textures  }
{-------------------------------------------------------------------}
procedure TModel.reloadTextures;
var i : integer;
begin
        for i := 0 to num_materials - 1 do          // for each shape
                materials[i].reloadTexture();

end;

{-------------------------------------------------------------------}
{  Render Model   }
{-------------------------------------------------------------------}
procedure TModel.render;
var k,i,j,materialIndex : integer;
   tri : TTri;
   N : TNormal;
   vec : TVec;
   bone : TBone;
   v : array [0..2] of single;
   matrix : TMatrix;
begin
 for k := 0 to num_shapes - 1 do           // for each shape
 begin
   materialIndex := material_indices[k];
   if ( materialIndex >= 0 ) then
     materials[materialIndex].activate()
   else
   begin
```

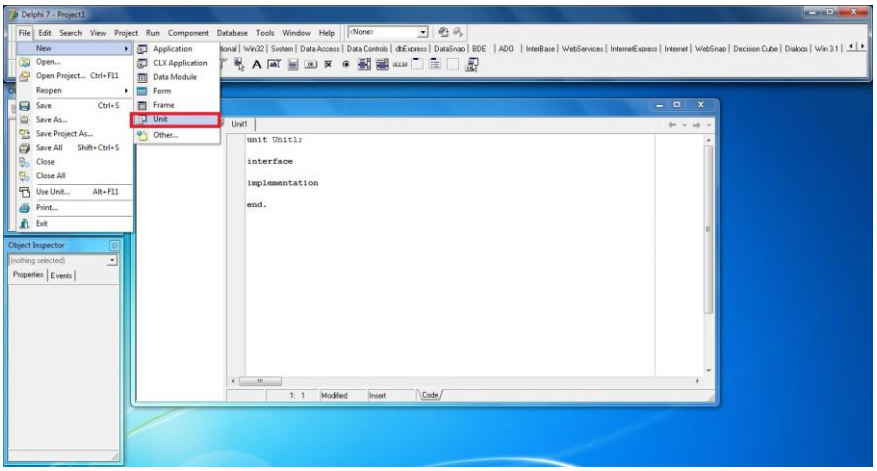
```
 // Material properties?
 glDisable( GL_TEXTURE_2D );
end;

glBegin(GL_TRIANGLES);
for i := 0 to shapes[k].num_triangles - 1 do   // for each triangle
begin
 tri := shapes[k].triangles[i];                          // pointer to triangle

 for j := 0 to 2 do       // 3 vertices of the triangle
 begin
  N := shapes[k].normals[tri.n[j]];
  glNormal3f(N.x, N.y, N.z);  // set normal vector  (object space)

  vec := shapes[k].vertices[tri.v[j]];              // pointer to vertex
  if (canMultiTexture) then
  begin
   // Give OpenGL the current terrain texture coordinate
            glMultiTexCoord2fARB(GL_TEXTURE0_ARB, vec.u, vec.v);

   if(isMultiTexture) then
   begin
    // Give OpenGL the current detail texture coordinate
    glMultiTexCoord2fARB(GL_TEXTURE1_ARB, vec.u, vec.v);
    //glTexCoord2f (vec.u, vec.v);                        // texture coordinate
   end;
  end
  else
  begin
   // Just do the Boring Old Texture Mapping.
   glTexCoord2f (vec.u, vec.v);                        // texture coordinate
   // Give OpenGL the current terrain texture coordinate
            //glMultiTexCoord2fARB(GL_TEXTURE0_ARB, vec.u, vec.v);
  end;
```

```
      if( vec.bone = -1 ) then
      begin
        glVertex3f( vec.x, vec.y, vec.z );
      end
      else
      begin
        bone := bones[vec.bone];
        matrix := bone.m_final;

        v[0] := vec.x;
        v[1] := vec.y;
        v[2] := vec.z;
        matrix.rotateVect( v );
        matrix.translateVect( v );
        glVertex3fv( @v );
      end;
    end;
  end;
  glEnd();
 end;
end;

{------------------------------------------------------------------}
{  Render Bones }
{------------------------------------------------------------------}
procedure TModel.renderBones;
var i : integer;
begin
          for i := 0 to num_bones - 1 do   // for each bone
            bones[i].render();
end;

end.
```

Lalu Save dengan nama file Model.

5.7 Source File Setup

Untuk membangun Delphi Source File, maka sorot ke menu File, lalu sorot New, dan klik Unit.



Setelah muncul halaman koding, lalu tuliskan koding berikut :

```
unit setup;
interface

uses
  Windows,
  Messages;

Const SETUP_WIDTH  = 320;
     SETUP_HEIGHT = 225;
     NULL = nil;
var Width  : Integer = 800;
   Height : Integer = 600;
   PixelDepth : Integer = 32;
   FullScreen : Boolean = False;
   MultiTexture : Boolean = True;
   HighPoly : Boolean = True;

function SetupWin(hInstance : HINST; hPrevInstance : HINST) : Boolean; stdcall;
```

```
implementation

var
  h_Wnd  : HWND;                  // Global window handle
  h_DC   : HDC;                   // Global device context
  keys   : Array[0..255] of Boolean; // Holds keystrokes
  Done   : Boolean = FALSE;
  bitmapDC, CheckBoxDC : HDC;

procedure PaintWindow;
begin
  BitBlt(h_DC, 0, 0, SETUP_WIDTH, SETUP_HEIGHT, bitmapDC, 0, 0, SRCCOPY);

  if MultiTexture = True then BitBlt(h_DC, 15, 160, 13, 12, checkboxDC, 0, 0,
SRCCOPY);
  if HighPoly = True then BitBlt(h_DC, 285, 160, 13, 12, checkboxDC, 0, 0, SRCCOPY);
  if Width = 640 then BitBlt(h_DC, 15, 99, 13, 12, checkboxDC, 0, 0, SRCCOPY);
  if Width = 800 then BitBlt(h_DC, 15, 119, 13, 12, checkboxDC, 0, 0, SRCCOPY);
  if Width =1024 then BitBlt(h_DC, 15, 140, 13, 12, checkboxDC, 0, 0, SRCCOPY);
  if Fullscreen = True then BitBlt(h_DC, 285, 140, 13, 12, checkboxDC, 0, 0,
SRCCOPY);
  if PixelDepth = 32 then BitBlt(h_DC, 285, 119, 13, 12, checkboxDC, 0, 0, SRCCOPY);
  if PixelDepth = 16 then BitBlt(h_DC, 285, 99, 13, 12, checkboxDC, 0, 0, SRCCOPY);
end;

procedure LoadSetupBMP;
var hBMP : HBitmap;
begin
  hBMP := LoadImage(hInstance, 'data\setup.bmp', IMAGE_BITMAP, 0, 0,
LR_DEFAULTSIZE OR LR_LOADFROMFILE);
  bitmapDC := CreateCompatibleDC( h_dc );
  SelectObject(bitmapDC, hBMP);
  DeleteObject(hBMP);
```

```
 hBMP := LoadImage(hInstance, 'data\setup_check.bmp', IMAGE_BITMAP, 0, 0,
LR_DEFAULTSIZE OR LR_LOADFROMFILE);
 checkBoxDC := CreateCompatibleDC( h_dc );
 SelectObject(checkBoxDC, hBMP);
 DeleteObject(hBMP);

 PaintWindow;
end;

{------------------------------------------------------------------}
{  Determines the application's response to the messages received  }
{------------------------------------------------------------------}
function WndProc(hWnd: HWND; Msg: UINT;  wParam: WPARAM;  lParam:
LPARAM): LRESULT; stdcall;
begin
 case (Msg) of
  WM_CREATE:
   begin
    // Insert stuff you want executed when the program starts
   end;
  WM_CLOSE:
   begin
    PostQuitMessage(0);
    Result := 0
   end;
  WM_MOVE:
   begin
    PaintWindow;  // Repaint the screen
   end;
  WM_ACTIVATE:
   begin
    PaintWindow;  // Repaint the screen
   end;
```

```
  WM_KEYDOWN:         // Set the pressed key (wparam) to equal true so we can
check if its pressed
    begin
     keys[wParam] := True;
     Result := 0;
    end;
  WM_KEYUP:       // Set the released key (wparam) to equal false so we can check
if its pressed
    begin
     keys[wParam] := False;
     Result := 0;
    end;
  WM_LBUTTONDOWN:  // Left mouse button down
    begin

     // test for checkboxes on lowest Y Range
     if (HiWORD(lParam) >=160) AND (HiWORD(lParam) <=173) then  { Y range }
     begin
      { X Ranges}
      if (LOWORD(lParam) >=15 ) AND (LOWORD(lParam) <=28) then
MultiTexture := Not MultiTexture;
      if (LOWORD(lParam) >=286 ) AND (LOWORD(lParam) <=298) then HighPoly
:= Not HighPoly;
     end;

     // test for checkboxes on higher Y Range
     if (HiWORD(lParam) >=140) AND (HiWORD(lParam) <=152) then  { Y range }
     begin
      { X Ranges}
      if (LOWORD(lParam) >=15 ) AND (LOWORD(lParam) <=28) then Width :=
1024;
      if (LOWORD(lParam) >=286 ) AND (LOWORD(lParam) <=298) then
FullScreen := Not FullScreen;
     end;
```

```
    // test for checkboxes on higher Y Range
    if (HiWORD(lParam) >=119) AND (HiWORD(lParam) <=131) then  { Y range }
    begin
     { X Ranges}
     if (LOWORD(lParam) >=15 ) AND (LOWORD(lParam) <=28) then Width :=
800;
     if (LOWORD(lParam) >=286 ) AND (LOWORD(lParam) <=298) then
PixelDepth := 32;
    end;

    // test for checkboxes on higher Y Range
    if (HiWORD(lParam) >=99) AND (HiWORD(lParam) <=112) then  { Y range }
    begin
     { X Ranges}
     if (LOWORD(lParam) >=15 ) AND (LOWORD(lParam) <=28) then Width :=
640;
     if (LOWORD(lParam) >=286 ) AND (LOWORD(lParam) <=298) then
PixelDepth := 16;
    end;

    // Test for OK button
    if (LOWORD(lParam) >=88 ) AND (LOWORD(lParam) <=148) AND  { X range }
      (HiWORD(lParam) >=195) AND (HiWORD(lParam) <=215) then { Y range }
    begin
     Done :=TRUE;
     keys[VK_ESCAPE] := TRUE;
    end;

    // Test for Cancel button
    if (LOWORD(lParam) >=156 ) AND (LOWORD(lParam) <=216) AND  { X range }
      (HiWORD(lParam) >=195) AND (HiWORD(lParam) <=215) then { Y range }
        keys[VK_ESCAPE] := TRUE;

    Result := 0;
```

```
      PaintWindow;  // Repaint the screen
     end;
    else
    begin
     Result := DefWindowProc(hWnd, Msg, wParam, lParam);
     Exit;
    end;
  end;
end;

{-------------------------------------------------------------------}
{  Properly destroys the window created at startup (no memory leaks)  }
{-------------------------------------------------------------------}
procedure KillWnd(Fullscreen : Boolean);
begin
 // Try to free the utilised DC's
 if (bitmapDC <> 0) and (DeleteDC(bitmapDC) = False) then
 begin
  MessageBox(0, 'Release of the bitmap device context failed!', 'Error', MB_OK or
MB_ICONERROR);
  bitmapDC := 0;
 end;

 if (CheckBoxDC <> 0) and (DeleteDC(CheckBoxDC) = False) then
 begin
  MessageBox(0, 'Release of the bitmap device context failed!', 'Error', MB_OK or
MB_ICONERROR);
  CheckBoxDC := 0;
 end;

 // Attemps to release the device context
 if ((h_DC <> 0) and (ReleaseDC(h_Wnd, h_DC) = 0)) then
 begin
```

```
  MessageBox(0, 'Release of device context failed!', 'Error', MB_OK or
MB_ICONERROR);
  h_DC := 0;
 end;

 // Attempts to destroy the window
 if ((h_Wnd <> 0) and (not DestroyWindow(h_Wnd))) then
 begin
  MessageBox(0, 'Unable to destroy window!', 'Error', MB_OK or MB_ICONERROR);
  h_Wnd := 0;
 end;

 // Attempts to unregister the window class
 if (not UnRegisterClass('SetupGL', hInstance)) then
 begin
  MessageBox(0, 'Unable to unregister window class!', 'Error', MB_OK or
MB_ICONERROR);
  hInstance := cardinal(NULL);
 end;
end;

{------------------------------------------------------------------}
{  Creates the window and attaches a OpenGL rendering context to it  }
{------------------------------------------------------------------}
function CreateWnd(Width, Height : Integer; Fullscreen : Boolean) : Boolean;
Const   pfd : PIXELFORMATDESCRIPTOR = (
  nSize:            SizeOf(PIXELFORMATDESCRIPTOR); // Size Of This Pixel Format
Descriptor
  nVersion:      1;              // The version of this data structure
  dwFlags:       PFD_DRAW_TO_WINDOW   // Buffer supports drawing to window
              or PFD_SUPPORT_OPENGL // Buffer supports OpenGL drawing
              or PFD_DOUBLEBUFFER;  // Supports double buffering
  iPixelType:    PFD_TYPE_RGBA;      // RGBA color format
  cColorBits:    16;             // OpenGL color depth
```

```
    cRedBits:        0;              // Number of red bitplanes
    cRedShift:       0;              // Shift count for red bitplanes
    cGreenBits:      0;              // Number of green bitplanes
    cGreenShift:     0;              // Shift count for green bitplanes
    cBlueBits:       0;              // Number of blue bitplanes
    cBlueShift:      0;              // Shift count for blue bitplanes
    cAlphaBits:      0;              // Not supported
    cAlphaShift:     0;              // Not supported
    cAccumBits:      0;              // No accumulation buffer
    cAccumRedBits:   0;              // Number of red bits in a-buffer
    cAccumGreenBits: 0;              // Number of green bits in a-buffer
    cAccumBlueBits:  0;              // Number of blue bits in a-buffer
    cAccumAlphaBits: 0;              // Number of alpha bits in a-buffer
    cDepthBits:      16;             // Specifies the depth of the depth buffer
    cStencilBits:    0;              // Turn off stencil buffer
    cAuxBuffers:     0;              // Not supported
    iLayerType:      PFD_MAIN_PLANE;      // Ignored
    bReserved:       0;              // Number of overlay and underlay planes
    dwLayerMask:     0;              // Ignored
    dwVisibleMask:   0;              // Transparent color of underlay plane
    dwDamageMask:    0               // Ignored
  );
var
  wndClass : TWndClass;      // Window class
  dwStyle : DWORD;           // Window styles
  dwExStyle : DWORD;         // Extended window styles
  dmScreenSettings : DEVMODE; // Screen settings (fullscreen, etc...)
  PixelFormat : Integer;      // Settings for the OpenGL rendering
  h_Instance : HINST;        // Current instance
  hPos, vPos : Integer;
  hDcZero: HDC;
begin
  h_Instance := GetModuleHandle(nil);      //Grab An Instance For Our Window
  ZeroMemory(@wndClass, SizeOf(wndClass));  // Clear the window class structure
```

```
 with wndClass do                // Set up the window class
 begin
  style       := CS_HREDRAW or    // Redraws entire window if length changes
                 CS_VREDRAW or    // Redraws entire window if height changes
                 CS_OWNDC;        // Unique device context for the window
  lpfnWndProc    := @WndProc;           // Set the window procedure to our func
WndProc
  hInstance     := h_Instance;
  hCursor       := LoadCursor(0, IDC_ARROW);
  lpszClassName := 'SetupGL';
 end;

 if (RegisterClass(wndClass) = 0) then  // Attemp to register the window class
 begin
  MessageBox(0, 'Failed to register the window class!', 'Error', MB_OK or
MB_ICONERROR);
  Result := False;
  Exit
 end;

 // Change to fullscreen if so desired
 if Fullscreen then
 begin
  ZeroMemory(@dmScreenSettings, SizeOf(dmScreenSettings));
  with dmScreenSettings do begin            // Set parameters for the screen setting
   dmSize       := SizeOf(dmScreenSettings);
   dmPelsWidth  := Width;                   // Window width
   dmPelsHeight := Height;                  // Window height
   dmBitsPerPel := 16;                    // 16 bit color
   dmFields     := DM_PELSWIDTH or DM_PELSHEIGHT or DM_BITSPERPEL;
  end;

  // Try to change screen mode to fullscreen
```

```
  if (ChangeDisplaySettings(dmScreenSettings, CDS_FULLSCREEN) =
DISP_CHANGE_FAILED) then
  begin
   MessageBox(0, 'Unable to switch to fullscreen!', 'Error', MB_OK or
MB_ICONERROR);
    Fullscreen := False;
  end;
 end;

 dwStyle := WS_CLIPCHILDREN or    // Doesn't draw within child windows
        WS_CLIPSIBLINGS;     // Doesn't draw within sibling windows
 dwExStyle := WS_EX_APPWINDOW or  // Top level window
         WS_EX_WINDOWEDGE or WS_EX_DLGMODALFRAME;     // Border with a
raised edge

 // Get screen center and coordinates to position window in the center
 hDcZero := GetDC(0);
 hPos := GetDeviceCaps(hDcZero, HORZRES);  // Screen width
 vPos := GetDeviceCaps(hDcZero, VERTRES);  // Screen Height
 hPos := (hPos - Width) DIV 2;
 vPos := (vPos - Height) DIV 2;

 // Attempt to create the actual window
 h_Wnd := CreateWindowEx(dwExStyle,    // Extended window styles
                 'SetupGL',    // Class name
                 'Schooling',  // Window title (caption)
                 dwStyle,      // Window styles
                 hPos, vPos,   // Window Position
                 Width, Height,  // Size of window
                 0,            // No parent window
                 0,            // No menu
                 h_Instance,   // Instance
                 nil);         // Pass nothing to WM_CREATE
 if h_Wnd = 0 then
```

```
 begin
  KillWnd(Fullscreen);              // Undo all the settings we've changed
  MessageBox(0, 'Unable to create window!', 'Error', MB_OK or MB_ICONERROR);
  Result := False;
  Exit;
 end;

 // Try to get a device context
 h_DC := GetDC(h_Wnd);
 if (h_DC = 0) then
 begin
  KillWnd(Fullscreen);
  MessageBox(0, 'Unable to get a device context!', 'Error', MB_OK or
MB_ICONERROR);
  Result := False;
  Exit;
 end;

 // Attempts to find the pixel format supported by a device context that is the
 // best match to a given pixel format specification.
 PixelFormat := ChoosePixelFormat(h_DC, @pfd);
 if (PixelFormat = 0) then
 begin
  KillWnd(Fullscreen);
  MessageBox(0, 'Unable to find a suitable pixel format', 'Error', MB_OK or
MB_ICONERROR);
  Result := False;
  Exit;
 end;

 // Sets the specified device context's pixel format to the format specified by
 // the PixelFormat.
 if (not SetPixelFormat(h_DC, PixelFormat, @pfd)) then
 begin
```

```
   KillWnd(Fullscreen);
   MessageBox(0, 'Unable to set the pixel format', 'Error', MB_OK or
MB_ICONERROR);
   Result := False;
   Exit;
  end;

  // Settings to ensure that the window is the topmost window
  ShowWindow(h_Wnd, SW_SHOW);
  SetForegroundWindow(h_Wnd);
  SetFocus(h_Wnd);

  LoadSetupBMP;
  Result := True;
  ReleaseDC(0, hDcZero);
end;

{------------------------------------------------------------------}
{  Main message loop for the application  }
{------------------------------------------------------------------}
function SetupWin(hInstance : HINST; hPrevInstance : HINST) : Boolean; stdcall;
var
  msg : TMsg;
  finished : Boolean;
begin
  finished := False;
  Done   :=FALSE;
  Result :=FALSE;

  // Perform application initialization:
  if not CreateWnd(SETUP_WIDTH+6, SETUP_HEIGHT+25, FALSE) then
   Exit;

  // Main message loop:
```

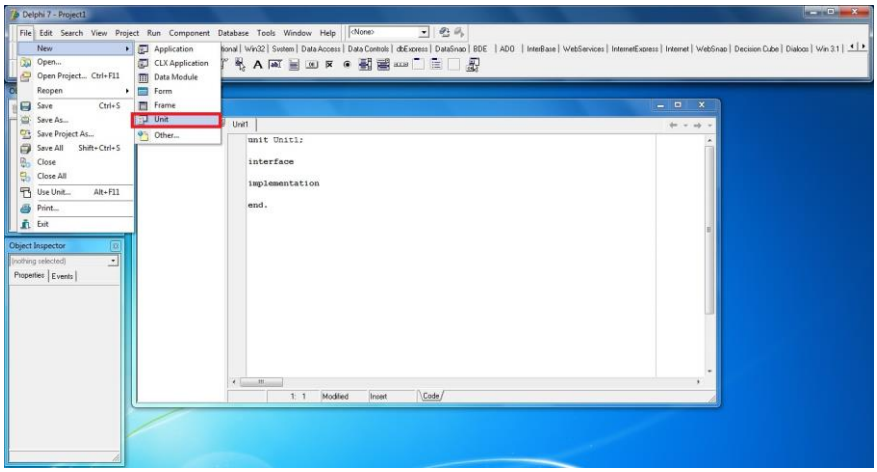
```
 while not(finished) do
 begin
  if (PeekMessage(msg, 0, 0, 0, PM_REMOVE)) then // Check if there is a message
for this window
  begin
   if (msg.message = WM_QUIT) then      // If WM_QUIT message received then we
are done
    finished := True
   else
   begin                          // Else translate and dispatch the message to this window
        TranslateMessage(msg);
    DispatchMessage(msg);
   end;
  end
  else
  begin
   if (keys[VK_ESCAPE]) then
    finished := True;

   if (keys[VK_RETURN]) then
   begin
    finished := True;
    Done :=True;
   end;

  end;
 end;

 case width of
   640 : Height :=480;
   800 : Height :=600;
  1024 : Height :=768;
 end;
 KillWnd(FALSE);
```

```
  result  :=Done;
end;

end.
```

Lalu Save dengan nama file Setup.

5.8 Source File Textures

Untuk membangun Delphi Source File, maka sorot ke menu File, lalu sorot New, dan klik Unit.



Setelah muncul halaman koding, lalu tuliskan koding berikut :

```
unit Textures;

interfac
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;

{------------------------------------------------------------------}
{  Swap bitmap format from BGR to RGB }
{------------------------------------------------------------------}
procedure SwapRGB(data : Pointer; Size : Integer);
asm
  mov ebx, eax
  mov ecx, size

@@loop :
  mov al,[ebx+0]
  mov ah,[ebx+2]
  mov [ebx+2],al
  mov [ebx+0],ah
  add ebx,3
  dec ecx
  jnz @@loop
end;

{------------------------------------------------------------------}
{  Create the Texture  }
{------------------------------------------------------------------}
function CreateTexture(Width, Height, Format : Word; pData : Pointer) : Integer;
var
  Texture : GLuint;
begin
  glGenTextures(1, Texture);
  glBindTexture(GL_TEXTURE_2D, Texture);
```

```
 glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE); {Texture blends with object background}
// glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL); {Texture does NOT blend with object background}

 { 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}

 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR); { only first two can be used }
 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR); { all of the above can be used }

 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;
 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.
 SwapRGB(pData, Width*Height);

 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('JPG Unit'), MB_OK);
        Exit;
      end;
    end;
```

```
 end
 else
 begin
  try
   if(FileExists(Filename)) then
    JPG.LoadFromFile(Filename)
   else
    JPG.LoadFromFile('data\' + Filename);
  except
   MessageBox(0, PChar('Couldn''t load JPG - "'+ Filename +'"'), PChar('JPG 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}
```

```
  CompImage : Pointer;
  Width, Height : Integer;
  ColorDepth    : Integer;
  ImageSize     : Integer;
  BufferIndex : Integer;
  currentByte : Integer;
  CurrentPixel : Integer;
  I : Integer;
  Front: ^Byte;
  Back: ^Byte;
  Temp: Byte;

  ResStream : TResourceStream;      // used for loading from resource
  // Copy a pixel from source to dest and Swap the RGB color values
  procedure CopySwapPixel(const Source, Destination : Pointer);
  asm
    push ebx
    mov bl,[eax+0]
    mov bh,[eax+1]
    mov [edx+2],bl
    mov [edx+1],bh
    mov bl,[eax+2]
    mov bh,[eax+3]
    mov [edx+0],bl
    mov [edx+3],bh
    pop ebx
  end;

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('TGA 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 24, 32 bit images
   if (TGAHeader.ImageType <> 2) AND    { TGA_RGB }
     (TGAHeader.ImageType <> 10) then  { Compressed RGB }
   begin
    Result := False;
    CloseFile(tgaFile);
    MessageBox(0, PChar('Couldn''t load "'+ Filename +'". Only 24 and 32bit TGA
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 and 32 bit TGA
files supported.'), PChar('TGA File Error'), MB_OK);
    Exit;
  end;

  GetMem(Image, ImageSize);

  if TGAHeader.ImageType = 2 then   // Standard 24, 32 bit TGA file
  begin
    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('TGA 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;

  // TGAs are stored BGR and not RGB, so swap the R and B bytes.
  // 32 bit TGA files have alpha channel and gets loaded differently
  if TGAHeader.BPP = 24 then
  begin
   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);
  end
  else
  begin
   for I :=0 to Width * Height - 1 do
   begin
    Front := Pointer(Integer(Image) + I*4);
    Back := Pointer(Integer(Image) + I*4 + 2);
    Temp := Front^;
    Front^ := Back^;
    Back^ := Temp;
   end;
   Texture :=CreateTexture(Width, Height, GL_RGBA, Image);
  end;
 end;

 // Compressed 24, 32 bit TGA files
 if TGAHeader.ImageType = 10 then
```

```
  begin
    ColorDepth :=ColorDepth DIV 8;
    CurrentByte :=0;
    CurrentPixel :=0;
    BufferIndex :=0;

    if LoadFromResource then // Load from resource
    begin
     try
      GetMem(CompImage, ResStream.Size-sizeOf(TGAHeader));
      ResStream.ReadBuffer(CompImage^, ResStream.Size-sizeOf(TGAHeader));  //
load compressed date into memory
      ResStream.Free;
     except
      MessageBox(0, PChar('Unable to read from resource - ' + Filename),
PChar('TGA Unit'), MB_OK);
      Exit;
     end;
    end
    else
    begin
     GetMem(CompImage, FileSize(TGAFile)-sizeOf(TGAHeader));
     BlockRead(TGAFile, CompImage^, FileSize(TGAFile)-sizeOf(TGAHeader),
BytesRead);  // load compressed data into memory
     if bytesRead <> FileSize(TGAFile)-sizeOf(TGAHeader) then
     begin
      Result := False;
      CloseFile(TGAFile);
      MessageBox(0, PChar('Couldn''t read file "'+ Filename +'".'), PChar('TGA File
Error'), MB_OK);
      Exit;
     end
    end;
```

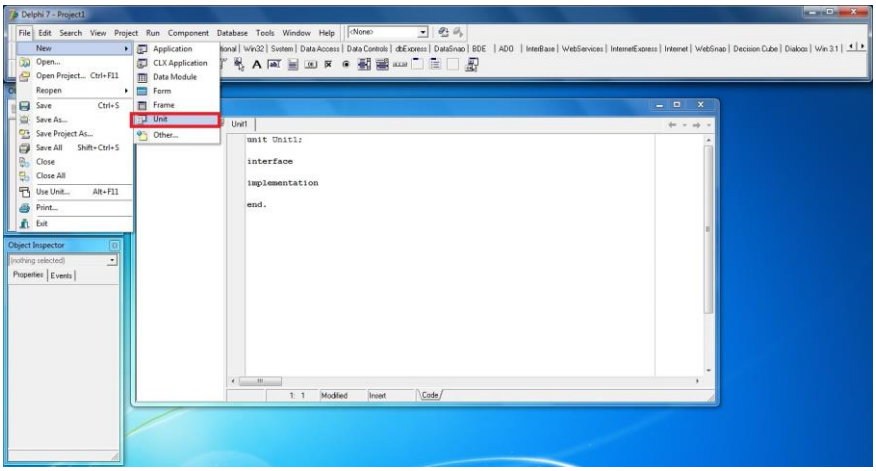
```
    // Extract pixel information from compressed data
    repeat
     Front := Pointer(Integer(CompImage) + BufferIndex);
     Inc(BufferIndex);
     if Front^ < 128 then
     begin
      For I := 0 to Front^ do
      begin
       CopySwapPixel(Pointer(Integer(CompImage)+BufferIndex+I*ColorDepth),
Pointer(Integer(image)+CurrentByte));
       CurrentByte := CurrentByte + ColorDepth;
       inc(CurrentPixel);
      end;
      BufferIndex :=BufferIndex + (Front^+1)*ColorDepth
     end
     else
     begin
      For I := 0 to Front^ -128 do
      begin
       CopySwapPixel(Pointer(Integer(CompImage)+BufferIndex),
Pointer(Integer(image)+CurrentByte));
       CurrentByte := CurrentByte + ColorDepth;
       inc(CurrentPixel);
      end;
      BufferIndex :=BufferIndex + ColorDepth
     end;
    until CurrentPixel >= Width*Height;

    if ColorDepth = 3 then
     Texture :=CreateTexture(Width, Height, GL_RGB, Image)
    else
     Texture :=CreateTexture(Width, Height, GL_RGBA, Image);
   end;
```

```
   Result :=TRUE;
   FreeMem(Image);
  end;
end;

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

end.
```

Lalu Save dengan nama file Textures.

5.9 Source File Timer

Untuk membangun Delphi Source File, maka sorot ke menu File, lalu sorot New, dan klik Unit.



Setelah muncul halaman koding, lalu tuliskan koding berikut :

```
unit Timer;

interface
uses
 windows,
 mmsystem;

type pLARGE_INTEGER = ^LARGE_INTEGER;
type
 ttimer = record
            // Create A Structure For The Timer Information
  frequency : int64;                                    //      Timer
Frequency
  resolution : single;                                  //      Timer
Resolution
  mm_timer_start : Longword;                  // Multimedia Timer Start
Value
```

```
   mm_timer_elapsed : Longword        ;                          // Multimedia Timer
Elapsed Time
   performance_timer : boolean;                          // Using The Performance
Timer?
   performance_timer_start : int64;      // Performance Timer Start Value
   performance_timer_elapsed : int64;// Performance Timer Elapsed Time
  end;

type
  TPerfTimer = class
  public
   timer : ttimer;
            constructor create();
   function  getTime() : single;  // in seconds since Timer creation
  end;
implementation

{ clsTimer }

constructor TPerfTimer.create;
begin
          // Check To See If A Performance Counter Is Available
          // If One Is Available The Timer Frequency Will Be Updated
          if ( not QueryPerformanceFrequency(timer.frequency)) then
          begin
                    // No Performace Counter Available
                    timer.performance_timer        := FALSE;
          // Set Performance Timer To FALSE
                    timer.mm_timer_start  := timeGetTime();                    // Use
timeGetTime() To Get Current Time
                    timer.resolution          := 1.0/1000.0;                   // Set
Our Timer Resolution To .001f
                    timer.frequency                := 100;
          // Set Our Timer Frequency To 1000
```

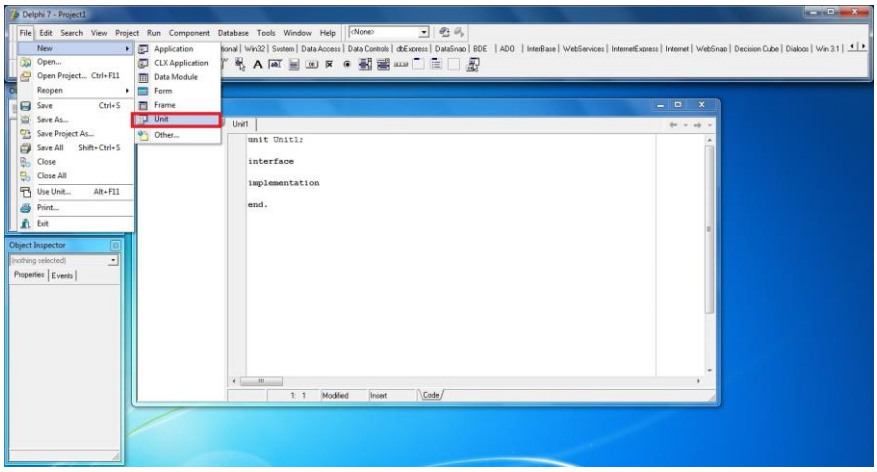
```
                timer.mm_timer_elapsed  := timer.mm_timer_start;
                // Set The Elapsed Time To The Current Time
        end
        else
        begin
                // Performance Counter Is Available, Use It Instead Of The
Multimedia Timer
                // Get The Current Time And Store It In performance_timer_start
                QueryPerformanceCounter(timer.performance_timer_start);
                timer.performance_timer         := TRUE;
                // Set Performance Timer To TRUE
                // Calculate The Timer Resolution Using The Timer Frequency
                timer.resolution        := ((1.0)/(timer.frequency));
                // Set The Elapsed Time To The Current Time
                timer.performance_timer_elapsed  :=
timer.performance_timer_start;
        end;
end;

function TPerfTimer.getTime: single;
var time : int64;                                                       //
time Will Hold A 64 Bit Integer
begin
        if (timer.performance_timer) then
        // Are We Using The Performance Timer?
        begin
                QueryPerformanceCounter(time);          // Grab The Current
Performance Time
                // Return The Current Time Minus The Start Time Multiplied By The
Resolution
                result := ( ( time - timer.performance_timer_start) *
timer.resolution);
        end
        else
```

```
        begin
                // Return The Current Time Minus The Start Time Multiplied By The
Resolution
                result := (( timeGetTime() - timer.mm_timer_start) *
timer.resolution);
        end;
end;

end.
```

Lalu Save dengan nama file Timer.

5.10 Source File Vector

Untuk membangun Delphi Source File, maka sorot ke menu File, lalu sorot New, dan klik Unit.



Setelah muncul halaman koding, lalu tuliskan koding berikut :

```
unit Vector;

interface
uses
  Matrix;

type psingle = ^single;
```

```
type
 clsVector = class
        public

  //    Vector data
              m_vector : array [0..3] of single;

              {       Constructor. (0, 0, 0, 1) }
              constructor create();overload;

              {       Constructor. 3 float values. }
              constructor create( vector : array of single );overload;

              {       Retrieve vector. }
              procedure getVector(var vector : array of single);

              {       Transform the vector by a matrix. }
              procedure transform( m : clsMatrix );

              {       Transform the vector by a matrix, not including the scaling
or transformation elements (use only top-left 3x3 of matrix). }
              procedure transform3( m : clsMatrix );

              {       Set the values of the vector. Takes 3 float values. }
              procedure setVector( vector : array of single );

              {       Translate by another vector. }
              procedure add( v : clsVector );

              {       Reset to (0, 0, 0, 1). }
              procedure reset();

              {       Get the length of the vector. }
              function length() : double;
```

```
                {       Normalize (make it a unit vector). }
                procedure normalize();

end;
implementation

{ clsVector }

procedure clsVector.add(v: clsVector);
begin
  m_vector[0] := m_vector[0] + v.m_vector[0];
        m_vector[1] := m_vector[1] + v.m_vector[1];
        m_vector[2] := m_vector[2] + v.m_vector[2];
        m_vector[3] := m_vector[3] + v.m_vector[3];
end;

constructor clsVector.create;
begin
  reset();
end;

constructor clsVector.create(vector: array of single);
begin
  setVector( vector );
        m_vector[3] := 1;
end;

procedure clsVector.getVector(var vector : array of single);
var i : integer;
begin
  for i := 0 to 3 do
    vector[i] := m_vector[i];
end;
```

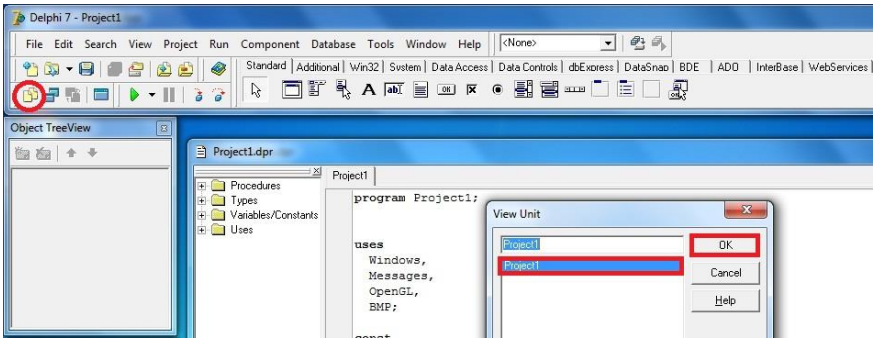
```
function clsVector.length: double;
begin
 result := sqrt(
m_vector[0]*m_vector[0]+m_vector[1]*m_vector[1]+m_vector[2]*m_vector[2] );
end;

procedure clsVector.normalize;
var len : double;
begin
 len := length();

        m_vector[0] := m_vector[0]/len;
        m_vector[1] := m_vector[1]/len;
        m_vector[2] := m_vector[2]/len;
end;

procedure clsVector.reset;
begin
 m_vector[0] := 0;
 m_vector[1] := 0;
 m_vector[2] := 0;
        m_vector[3] := 1;
end;

procedure clsVector.setVector(vector: array of single);
begin
 m_vector[0] := vector[0];
        m_vector[1] := vector[1];
        m_vector[2] := vector[2];
end;

procedure clsVector.transform(m: clsMatrix);
var matrix : array [0..15] of single;
begin
```

```
        m.getMatrix(matrix);
        m_vector[0]                                                    :=
m_vector[0]*matrix[0]+m_vector[1]*matrix[4]+m_vector[2]*matrix[8]+matrix[12];
        m_vector[1]                                                    :=
m_vector[0]*matrix[1]+m_vector[1]*matrix[5]+m_vector[2]*matrix[9]+matrix[13];
        m_vector[2]                                                    :=
m_vector[0]*matrix[2]+m_vector[1]*matrix[6]+m_vector[2]*matrix[10]+matrix[14
];
        m_vector[3]                                                    :=
m_vector[0]*matrix[3]+m_vector[1]*matrix[7]+m_vector[2]*matrix[11]+matrix[15
];
end;

procedure clsVector.transform3(m: clsMatrix);
var matrix : array [0..15] of single;
begin
        m.getMatrix(matrix);
        m_vector[0]                                                    :=
m_vector[0]*matrix[0]+m_vector[1]*matrix[4]+m_vector[2]*matrix[8];
        m_vector[1]                                                    :=
m_vector[0]*matrix[1]+m_vector[1]*matrix[5]+m_vector[2]*matrix[9];
        m_vector[2]                                                    :=
m_vector[0]*matrix[2]+m_vector[1]*matrix[6]+m_vector[2]*matrix[10];
        m_vector[3] := 1;
end;

end.
```

Lalu Save dengan nama file Vector.

5.11 Proyek Utama

Pada pembangunan proyek utama, lakukan File dan Close, lalu klik icon View Unit pada pojok kiri panel komponen, dan pilih project1, lalu klik tombol ok.



Setelah muncul halaman koding, selanjutnya tuliskan koding berikut :

```
program Project1;

uses
  Windows,
  Messages,
  OpenGL,
  Textures in 'Textures.pas',
  Boid in 'Boid.pas',
  Animate in 'Animate.pas',
  ExtOpenGL in 'ExtOpenGL.pas',
  Setup in 'Setup.pas';

const
  WND_TITLE = '2D Boids (Flocking) in OpenGL By Universitas Malikussaleh';
  FPS_TIMER = 1;                // Timer to calculate FPS
  FPS_INTERVAL = 1000;          // Calculate FPS every 1000 ms

var
  h_Wnd  : HWND;                // Global window handle
  h_DC   : HDC;                 // Global device context
```

```
  h_RC   : HGLRC;                // OpenGL rendering context
  keys : Array[0..255] of Boolean;   // Holds keystrokes
  FPSCount : Integer = 0;          // Counter for FPS
  ElapsedTime : Integer;           // Elapsed time between frames

  // User variables
  HighlightIndex : integer;
  LPressed : boolean;
  KPressed : boolean;
  isSetupMultiTexture : boolean = True;
  isHighPolyCount : boolean = true;
  TexBG : GLuint;

  // Flocking variables
  Flock : TFlock;

  //Fog Variables
  fogColor : array [0..3] of GLfloat;

  //Terrain
  Terrain : TAnimate;
{$R *.RES}

procedure glBindTexture(target: GLenum; texture: GLuint); stdcall; external
opengl32;
{------------------------------------------------------------------}
{  Function to convert int to string. (No sysutils = smaller EXE)  }
{------------------------------------------------------------------}
function IntToStr(Num : Integer) : String;  // using SysUtils increase file size by
100K
begin
  Str(Num, result);
end;
{------------------------------------------------------------------}
```

```
{  Function to draw the actual scene }
{--------------------------------------------------------------------}
procedure glDraw();
begin
 glClear(GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT);       // Clear The
Screen And The Depth Buffer
 glClearColor(0.3,0.3,1.0,0.0);
 glLoadIdentity();                          // Reset The View

 glTranslatef(-4.5,-2.0,-5);

  // Render Flock
 glPushMatrix;
  glRotate(-45,1.0,0.0,0.0);
  Flock.moveMembers;
  Flock.Render(LPressed,KPressed);
 glPopMatrix;

 // Draw Background
 if(MultiTexture) then
 begin
  glActiveTextureARB(GL_TEXTURE0_ARB);
 end;

 glPushMatrix;
  glTranslatef(4.5,4.0,-8.0);
  glBindTexture(GL_TEXTURE_2D,TexBG);
  glColor3f(1.0,1.0,1.0);
  glBegin(GL_QUADS);
    glTexCoord2f(0.0, 0.0);  glVertex3f( 8.0,-1.0, 0.0);
    glTexCoord2f(0.0, 1.0);  glVertex3f( 8.0, 4.0, 0.0);
    glTexCoord2f(1.0, 1.0);  glVertex3f(-8.0, 4.0, 0.0);
    glTexCoord2f(1.0, 0.0);  glVertex3f(-8.0,-1.0, 0.0);
  glEnd;
```

```
  glPopMatrix;
  glDisable(GL_TEXTURE_2D);

  glPushMatrix;
   glTranslatef(6.5,-2.0,-1.5);
   glRotatef(-70,1.0,0.0,0.0);
   glScalef(0.05,0.05,0.05);
   glColor3f(1.0,1.0,1.0);
   // Render Terrain
   Terrain.render;
  glPopMatrix;

  glDisable(GL_TEXTURE_2D);
  glDisable(GL_TEXTURE_2D);
end;
{------------------------------------------------------------------}
{  Initialise OpenGL  }
{------------------------------------------------------------------}
procedure glInit();
var ColRed, ColGreen, ColBlue : single;
   i : integer;
   XPos, YPos : double;
   SkinString : string;
begin
 glClearColor(0.3,0.3,1.0,0.0);         // Blue Background
 glShadeModel(GL_SMOOTH);               // Enables Smooth Color Shading
 glClearDepth(1.0);                    // Depth Buffer Setup
 glEnable(GL_DEPTH_TEST);              // Enable Depth Buffer
 glDepthFunc(GL_LESS);                                  // The Type Of Depth Test To Do

 glEnable(GL_ALPHA_TEST);
 glAlphaFunc(GL_GREATER,0.4);

 // Add some fog
```

```
 glEnable(GL_FOG);

 fogColor[0] := 0.0;
 fogColor[1] := 0.0;
 fogColor[2] := 0.8;
 fogColor[3] := 0.0;

 glFogi (GL_FOG_MODE, GL_EXP2);
 glFogfv (GL_FOG_COLOR, @fogColor);
 glFogf (GL_FOG_DENSITY, 0.04);
 glHint (GL_FOG_HINT,GL_NICEST );

 glHint(GL_PERSPECTIVE_CORRECTION_HINT,    GL_NICEST);          //Realy    Nice
perspective calculations

 LoadTexture('data\reefbackgr.jpg', TexBG, FALSE);    // Load the Texture

 // Use High Polygon Models?
 if(isHighPolyCount) then
  SkinString := 'data\fish'
 else
  SkinString := 'data\lpfish';

 Flock := TFlock.Create(8 ,4 ,1000 ,1000,SkinString,isSetupMultiTexture,false);

 // Add Sharks
 for i := 0 to 2 do
 begin
  Flock.addBoid(2*random*PI,                // heading
             random(1),                 // speed
             random*1000,                 // x position
             random*1000,                 // y position
             13,                       // maximum speed
             7.5,                       // minimum speed
```

```
            0.01,                   // turning rate
            0.05,                   // acceleration
            60.0,                   // sensor range
            10.0,                   // death range
            0.23109,                    // death constant
            50.0,                   // collision range
            1.0,                    // attack constant
            1.0,                    // retreat constant
            0,                      // Team Index
            0.99,                   // Red
            0.99,                   // Green
            0.99);                   // Blue
end;

// Add Small Fish To Flock
ColRed := ((random-0.1)/2)+0.5;
ColGreen := ((random-0.1)/2)+0.5;
ColBlue := ((random-0.1)/2)+0.5;
XPos := random*1000;
YPos := random*1000;
for i := 0 to 1 do       //tambah 100
begin
  Flock.addBoid(2*random*PI,                // heading
            random(1),                  // speed
            XPos,                   // x position
            YPos,                   // y position
            9.5,                    // maximum speed
            7.6,                    // minimum speed
            0.02,                   // turning rate
            0.05,                   // acceleration
            60.0,                   // sensor range
            10.0,                   // death range
            0.23109,                    // death constant
            25.0,                   // collision range
```

```
            1.0,                    // attack constant
            1.0,                    // retreat constant
            1,                      // Team Index
            ColRed,                 // Red
            ColGreen,               // Green
            ColBlue);               // Blue
end;

// Add Middle Sized Fish To Flock
ColRed := ((random-0.1)/2)+0.5;
ColGreen := ((random-0.1)/2)+0.5;
ColBlue := ((random-0.1)/2)+0.5;

for i := 0 to 10 do
begin
 Flock.addBoid(2*random*PI,             // heading
            random(1),                  // speed
            random*1000,                // x position
            random*1000,                // y position
            1.0,                    // maximum speed
            0.5,                    // minimum speed
            0.02,                   // turning rate
            0.05,                   // acceleration
            60.0,                   // sensor range
            10.0,                   // death range
            0.23109,                // death constant
            40.0,                   // collision range
            1.0,                    // attack constant
            1.0,                    // retreat constant
            2,                      // Team Index
            ColRed,                 // Red
            ColGreen,               // Green
            ColBlue);               // Blue
end;
```

```
// Add Big Fish To Flock
ColRed := ((random-0.1)/2)+0.5;
ColGreen := ((random-0.1)/2)+0.5;
ColBlue := ((random-0.1)/2)+0.5;

for i := 0 to 10 do
begin
  Flock.addBoid(2*random*PI,              // heading
          random(1),              // speed
          random*1000,            // x position
          random*1000,            // y position
          0.8,                    // maximum speed
          0.3,                    // minimum speed
          0.01,                   // turning rate
          0.05,                   // acceleration
          60.0,                   // sensor range
          10.0,                   // death range
          0.23109,                // death constant
          35.0,                   // collision range
          1.0,                    // attack constant
          1.0,                    // retreat constant
          3,                      // Team Index
          ColRed,                 // Red
          ColGreen,               // Green
          ColBlue);               // Blue
end;

Flock.setMeanX(Flock.calculateMeanX());
Flock.setMeanY(Flock.calculateMeanY());

// Load Terrain
HighlightIndex := 0;
Terrain                                                                    :=
TAnimate.Create('data\terrain.txt',24,isSetupMultiTexture,isSetupMultiTexture);
```

```
end;
{------------------------------------------------------------------}
{  Handle window resize }
{------------------------------------------------------------------}
procedure glResizeWnd(Width, Height : Integer);
begin
 if (Height = 0) then              // prevent divide by zero exception
   Height := 1;
 glViewport(0, 0, Width, Height);    // Set the viewport for the OpenGL window
 glMatrixMode(GL_PROJECTION);       // Change Matrix Mode to Projection
 glLoadIdentity();                 // Reset View
 gluPerspective(45.0, Width/Height, 1.0, 100.0);      // Do the perspective
calculations. Last value = max clipping depth

 glMatrixMode(GL_MODELVIEW);        // Return to the modelview matrix
 glLoadIdentity();                 // Reset View
end;
{------------------------------------------------------------------}
{  Processes all the keystrokes }
{------------------------------------------------------------------}
procedure ProcessKeys;
begin
 if (keys[76]) then // L
 begin
   // Toggle on or off
   LPressed := not LPressed;
   keys[76] := not keys[76];
 end;

 if (keys[75]) then // K
 begin
   // Toggle on or off
   KPressed := not KPressed;
   keys[75] := not keys[75];
```

```
  end;
end;
{------------------------------------------------------------------}
{  Determines the application's response to the messages received  }
{------------------------------------------------------------------}
function WndProc(hWnd: HWND; Msg: UINT;  wParam: WPARAM;  lParam:
LPARAM): LRESULT; stdcall;
begin
  case (Msg) of
    WM_CREATE:
      begin
        // Insert stuff you want executed when the program starts
      end;
    WM_CLOSE:
      begin
        PostQuitMessage(0);
        Result := 0
      end;
    WM_KEYDOWN:        // Set the pressed key (wparam) to equal true so we can
check if its pressed
      begin
        keys[wParam] := True;
        Result := 0;
      end;
    WM_KEYUP:        // Set the released key (wparam) to equal false so we can check
if its pressed
      begin
        keys[wParam] := False;
        Result := 0;
      end;
    WM_SIZE:        // Resize the window with the new width and height
      begin
        glResizeWnd(LOWORD(lParam),HIWORD(lParam));
        Result := 0;
```

```
    end;
   WM_TIMER :                  // Add code here for all timers to be used.
    begin
     if wParam = FPS_TIMER then
     begin
      FPSCount :=Round(FPSCount * 1000/FPS_INTERVAL);   // calculate to get per
Second incase intercal is less or greater than 1 second
      SetWindowText(h_Wnd, PChar(WND_TITLE + '    [' + intToStr(FPSCount) + '
FPS]'));
      FPSCount := 0;
      Result := 0;
     end;
    end;
   else
    Result := DefWindowProc(hWnd, Msg, wParam, lParam);     // Default result if
nothing happens
  end;
end;

{-------------------------------------------------------------------}
{  Properly destroys the window created at startup (no memory leaks)  }
{-------------------------------------------------------------------}
procedure glKillWnd(Fullscreen : Boolean);
begin
 if Fullscreen then           // Change back to non fullscreen
 begin
  ChangeDisplaySettings(devmode(nil^), 0);
  ShowCursor(True);
 end;

 // Makes current rendering context not current, and releases the device
 // context that is used by the rendering context.
 if (not wglMakeCurrent(h_DC, 0)) then
  MessageBox(0, 'Release of DC and RC failed!', 'Error', MB_OK or MB_ICONERROR);
```

```
  // Attempts to delete the rendering context
  if (not wglDeleteContext(h_RC)) then
  begin
    MessageBox(0, 'Release of rendering context failed!', 'Error', MB_OK or
MB_ICONERROR);
    h_RC := 0;
  end;

  // Attemps to release the device context
  if ((h_DC > 0) and (ReleaseDC(h_Wnd, h_DC) = 0)) then
  begin
    MessageBox(0, 'Release of device context failed!', 'Error', MB_OK or
MB_ICONERROR);
    h_DC := 0;
  end;

  // Attempts to destroy the window
  if ((h_Wnd <> 0) and (not DestroyWindow(h_Wnd))) then
  begin
    MessageBox(0, 'Unable to destroy window!', 'Error', MB_OK or MB_ICONERROR);
    h_Wnd := 0;
  end;

  // Attempts to unregister the window class
  if (not UnRegisterClass('OpenGL', hInstance)) then
  begin
    MessageBox(0, 'Unable to unregister window class!', 'Error', MB_OK or
MB_ICONERROR);
    hInstance := 0;
  end;
end;
{-------------------------------------------------------------------}
{  Creates the window and attaches a OpenGL rendering context to it  }
{-------------------------------------------------------------------}
```

```
function glCreateWnd(Width, Height : Integer; Fullscreen : Boolean; PixelDepth : Integer) : Boolean;
var
 wndClass : TWndClass;        // Window class
 dwStyle : DWORD;             // Window styles
 dwExStyle : DWORD;           // Extended window styles
 dmScreenSettings : DEVMODE;  // Screen settings (fullscreen, etc...)
 PixelFormat : GLuint;        // Settings for the OpenGL rendering
 h_Instance : HINST;          // Current instance
 pfd : TPIXELFORMATDESCRIPTOR;  // Settings for the OpenGL window
begin
 h_Instance := GetModuleHandle(nil);      //Grab An Instance For Our Window
 ZeroMemory(@wndClass, SizeOf(wndClass));  // Clear the window class structure

 with wndClass do                  // Set up the window class
 begin
  style       := CS_HREDRAW or    // Redraws entire window if length changes
                 CS_VREDRAW or    // Redraws entire window if height changes
                 CS_OWNDC;        // Unique device context for the window
  lpfnWndProc    := @WndProc;           // Set the window procedure to our func WndProc
  hInstance     := h_Instance;
  hCursor       := LoadCursor(0, IDC_ARROW);
  lpszClassName := 'OpenGL';
 end;

 if (RegisterClass(wndClass) = 0) then  // Attemp to register the window class
 begin
  MessageBox(0, 'Failed to register the window class!', 'Error', MB_OK or MB_ICONERROR);
  Result := False;
  Exit
 end;
```

```
 // Change to fullscreen if so desired
 if Fullscreen then
 begin
  ZeroMemory(@dmScreenSettings, SizeOf(dmScreenSettings));
  with dmScreenSettings do begin            // Set parameters for the screen setting
   dmSize      := SizeOf(dmScreenSettings);
   dmPelsWidth  := Width;                   // Window width
   dmPelsHeight := Height;                  // Window height
   dmBitsPerPel := PixelDepth;              // Window color depth
   dmFields     := DM_PELSWIDTH or DM_PELSHEIGHT or DM_BITSPERPEL;
  end;

  // Try to change screen mode to fullscreen
  if        (ChangeDisplaySettings(dmScreenSettings,        CDS_FULLSCREEN)        =
DISP_CHANGE_FAILED) then
  begin
   MessageBox(0,  'Unable  to  switch  to  fullscreen!',  'Error',  MB_OK  or
MB_ICONERROR);
   Fullscreen := False;
  end;
 end;

 // If we are still in fullscreen then
 if (Fullscreen) then
 begin
  dwStyle := WS_POPUP or              // Creates a popup window
          WS_CLIPCHILDREN           // Doesn't draw within child windows
          or WS_CLIPSIBLINGS;       // Doesn't draw within sibling windows
  dwExStyle := WS_EX_APPWINDOW;        // Top level window
  ShowCursor(False);                  // Turn of the cursor (gets in the way)
 end
 else
 begin
  dwStyle := WS_OVERLAPPEDWINDOW or     // Creates an overlapping window
```

```
        WS_CLIPCHILDREN or        // Doesn't draw within child windows
        WS_CLIPSIBLINGS;          // Doesn't draw within sibling windows
  dwExStyle := WS_EX_APPWINDOW or      // Top level window
         WS_EX_WINDOWEDGE;       // Border with a raised edge
 end;

 // Attempt to create the actual window
 h_Wnd := CreateWindowEx(dwExStyle,     // Extended window styles
                 'OpenGL',      // Class name
                 WND_TITLE,     // Window title (caption)
                 dwStyle,       // Window styles
                 0, 0,          // Window position
                 Width, Height,  // Size of window
                 0,             // No parent window
                 0,             // No menu
                 h_Instance,    // Instance
                 nil);          // Pass nothing to WM_CREATE
 if h_Wnd = 0 then
 begin
  glKillWnd(Fullscreen);              // Undo all the settings we've changed
  MessageBox(0, 'Unable to create window!', 'Error', MB_OK or MB_ICONERROR);
  Result := False;
  Exit;
 end;

 // Try to get a device context
 h_DC := GetDC(h_Wnd);
 if (h_DC = 0) then
 begin
  glKillWnd(Fullscreen);
  MessageBox(0, 'Unable to get a device context!', 'Error', MB_OK or
MB_ICONERROR);
  Result := False;
  Exit;
```

```
 end;

 // Settings for the OpenGL window
 with pfd do
 begin
  nSize          := SizeOf(TPIXELFORMATDESCRIPTOR); // Size Of This Pixel Format
Descriptor
  nVersion       := 1;                // The version of this data structure
  dwFlags            := PFD_DRAW_TO_WINDOW      // Buffer supports drawing to
window
              or PFD_SUPPORT_OPENGL // Buffer supports OpenGL drawing
              or PFD_DOUBLEBUFFER;  // Supports double buffering
  iPixelType     := PFD_TYPE_RGBA;       // RGBA color format
  cColorBits     := PixelDepth;          // OpenGL color depth
  cRedBits       := 0;                // Number of red bitplanes
  cRedShift      := 0;                // Shift count for red bitplanes
  cGreenBits     := 0;                // Number of green bitplanes
  cGreenShift    := 0;                // Shift count for green bitplanes
  cBlueBits      := 0;                // Number of blue bitplanes
  cBlueShift     := 0;                // Shift count for blue bitplanes
  cAlphaBits     := 0;                // Not supported
  cAlphaShift    := 0;                // Not supported
  cAccumBits     := 0;                // No accumulation buffer
  cAccumRedBits  := 0;                // Number of red bits in a-buffer
  cAccumGreenBits := 0;               // Number of green bits in a-buffer
  cAccumBlueBits  := 0;               // Number of blue bits in a-buffer
  cAccumAlphaBits := 0;               // Number of alpha bits in a-buffer
  cDepthBits     := 16;               // Specifies the depth of the depth buffer
  cStencilBits   := 0;                // Turn off stencil buffer
  cAuxBuffers    := 0;                // Not supported
  iLayerType     := PFD_MAIN_PLANE;      // Ignored
  bReserved      := 0;                // Number of overlay and underlay planes
  dwLayerMask    := 0;                // Ignored
  dwVisibleMask  := 0;                // Transparent color of underlay plane
```

```
  dwDamageMask    := 0;                 // Ignored
 end;

 // Attempts to find the pixel format supported by a device context that is the best match to a given pixel format specification.
 PixelFormat := ChoosePixelFormat(h_DC, @pfd);
 if (PixelFormat = 0) then
 begin
  glKillWnd(Fullscreen);
  MessageBox(0, 'Unable to find a suitable pixel format', 'Error', MB_OK or MB_ICONERROR);
  Result := False;
  Exit;
 end;

 // Sets the specified device context's pixel format to the format specified by the PixelFormat.
 if (not SetPixelFormat(h_DC, PixelFormat, @pfd)) then
 begin
  glKillWnd(Fullscreen);
  MessageBox(0, 'Unable to set the pixel format', 'Error', MB_OK or MB_ICONERROR);
  Result := False;
  Exit;
 end;

 // Create a OpenGL rendering context
 h_RC := wglCreateContext(h_DC);
 if (h_RC = 0) then
 begin
  glKillWnd(Fullscreen);
  MessageBox(0, 'Unable to create an OpenGL rendering context', 'Error', MB_OK or MB_ICONERROR);
  Result := False;
```

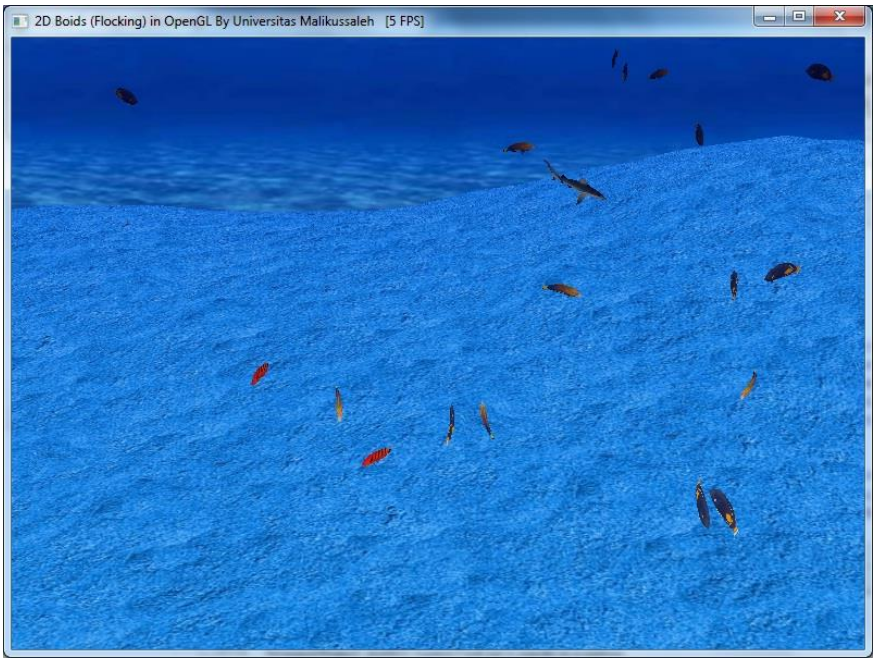
```
  Exit;
 end;

 // Makes the specified OpenGL rendering context the calling thread's current
rendering context
 if (not wglMakeCurrent(h_DC, h_RC)) then
 begin
  glKillWnd(Fullscreen);
  MessageBox(0, 'Unable to activate OpenGL rendering context', 'Error', MB_OK or
MB_ICONERROR);
  Result := False;
  Exit;
 end;

 // Initializes the timer used to calculate the FPS
 SetTimer(h_Wnd, FPS_TIMER, FPS_INTERVAL, nil);

 // Settings to ensure that the window is the topmost window
 ShowWindow(h_Wnd, SW_SHOW);
 SetForegroundWindow(h_Wnd);
 SetFocus(h_Wnd);

 // Ensure the OpenGL window is resized properly
 glResizeWnd(Width, Height);
 glInit();

 Result := True;
end;
{------------------------------------------------------------------}
{  Main message loop for the application   }
{------------------------------------------------------------------}
function WinMain(hInstance : HINST; hPrevInstance : HINST;
            lpCmdLine : PChar; nCmdShow : Integer) : Integer; stdcall;
var
```

```
  msg : TMsg;
  finished : Boolean;
  DemoStart, LastTime : DWord;
begin
  finished := False;

  // Get the Demo Multitexture And Polygon Count For models.
  if not SetupWin(hInstance, hPrevInstance) then
  begin
    Result := 0;
    Exit;
  end;

  // Set Local Variables For Later Use
  isSetupMultiTexture := Setup.MultiTexture;
  isHighPolyCount := Setup.HighPoly;

  // Perform application initialization:
  if not glCreateWnd(setup.Width, Setup.Height, Setup.FullScreen, Setup.PixelDepth)
then
  begin
    Result := 0;
    Exit;
  end;

  DemoStart := GetTickCount();          // Get Time when demo started

  // Main message loop:
  while not finished do
  begin
    if (PeekMessage(msg, 0, 0, 0, PM_REMOVE)) then // Check if there is a message
for this window
    begin
```

```
    if (msg.message = WM_QUIT) then      // If WM_QUIT message received then we
are done
      finished := True
    else
    begin                          // Else translate and dispatch the message to this window
          TranslateMessage(msg);
      DispatchMessage(msg);
    end;
   end
   else
   begin
    Inc(FPSCount);                  // Increment FPS Counter

    LastTime :=ElapsedTime;
    ElapsedTime :=GetTickCount() - DemoStart;     // Calculate Elapsed Time
    ElapsedTime :=(LastTime + ElapsedTime) DIV 2; // Average it out for smoother
movement

    glDraw();                       // Draw the scene
    SwapBuffers(h_DC);              // Display the scene

    if (keys[VK_ESCAPE]) then          // If user pressed ESC then set finised TRUE
      finished := True
    else
      ProcessKeys;                  // Check for any other key Pressed
   end;
  end;
  glKillWnd(FALSE);

  // Remember to release the objects created
  if Assigned(Flock) then
   Flock.Free;
  if Assigned(Terrain) then
   Terrain.Free;
```

```
  Result := msg.wParam;
end;

begin
  WinMain( hInstance, hPrevInst, CmdLine, CmdShow );
end..
```

dan klik Save All pada menu utama File.
Tampilan eksekusi program akan berupa :



GRAFIKA KOMPUTER 3D

Silvia Ratna

Lahir di Puruk-Cahu Provinsi Kalimantan-Tengah pada tanggal 13 September 1975, Sekolah dasar sampai dengan SMA ditamatkan di Puruk-Cahu. Pada Tahun 1995 melanjutkan kuliah di STMIK AKAKOM Yogyakarta dan memperoleh gelar Sarjana Komputer (S.Kom) pada tahun 2000, Pada tahun 2006 melanjutkan pendidikan S2 di STTS Surabaya dan memperoleh gelar Magister Komputer (M.Kom)

pada tahun 2008, selanjutnya pada tahun 2011 melanjutkan pendidikan S3 Program Doktor Ilmu Administrasi Minat Ilmu Administrasi Bisnis di Fakultas Ilmu Administrasi Universitas Brawijaya Malang dan selesai pada tahun 2016 dengan menyandang gelar Doktor (Dr.), Pengalaman kerja Menjadi Dosen dan Pembantu Ketua 1 pada STMIK Banjarbaru(Kampus Banjarmasin) pada tahun 2003 s/d 2005, tahun 2005 diterima menjadi Dosen PNS Kopertis Wilayah XI dpk pada Universitas Islam Kalimantan Muhammad Arsyad Al-Banjari, tahun 2008 s/d 2011 menjabat sebagai Pembantu Dekan II pada Fakultas Teknik Universitas Islam Kalimantan Muhammad Arsyad Al-Banjari (UNISKA MAAB) dan pada tahun 2011 sd 2013 menjabat sebagai Dekan pada Fakultas Teknik Universitas Islam Kalimantan Muhammad Arsyad Al-Banjari, Tahun 2014 s/d Sekarang menjabat sebagai Dekan pada Fakultas Teknologi Informasi Universitas Islam Kalimantan Muhammad Arsyad Al-Banjari, sebagai Asesor BKD dan sebagai pengurus sekaligus penyantun di Yayasan Graha Education yang menaungi Pendidikan Anak Usia Dini Islam Terpadu "Anak sholeh Mandiri" (PAUD IT ASM), SD IT ASM, SMP IT ASM

Penerbit
FAKULTAS EKONOMI DAN BISNIS ISLAM – IAIN LHOKSEUMAWE
Anggota Afiliasi Penerbit Perguruan Tinggi Indonesia (APPTI)
Nomor: 005.152.1.3.2022
Email: penerbitfebi@iainlhokseumawe.ac.id
https://febi.iainlhokseumawe.ac.id/penerbit

ISBN 978-623-88237-0-3