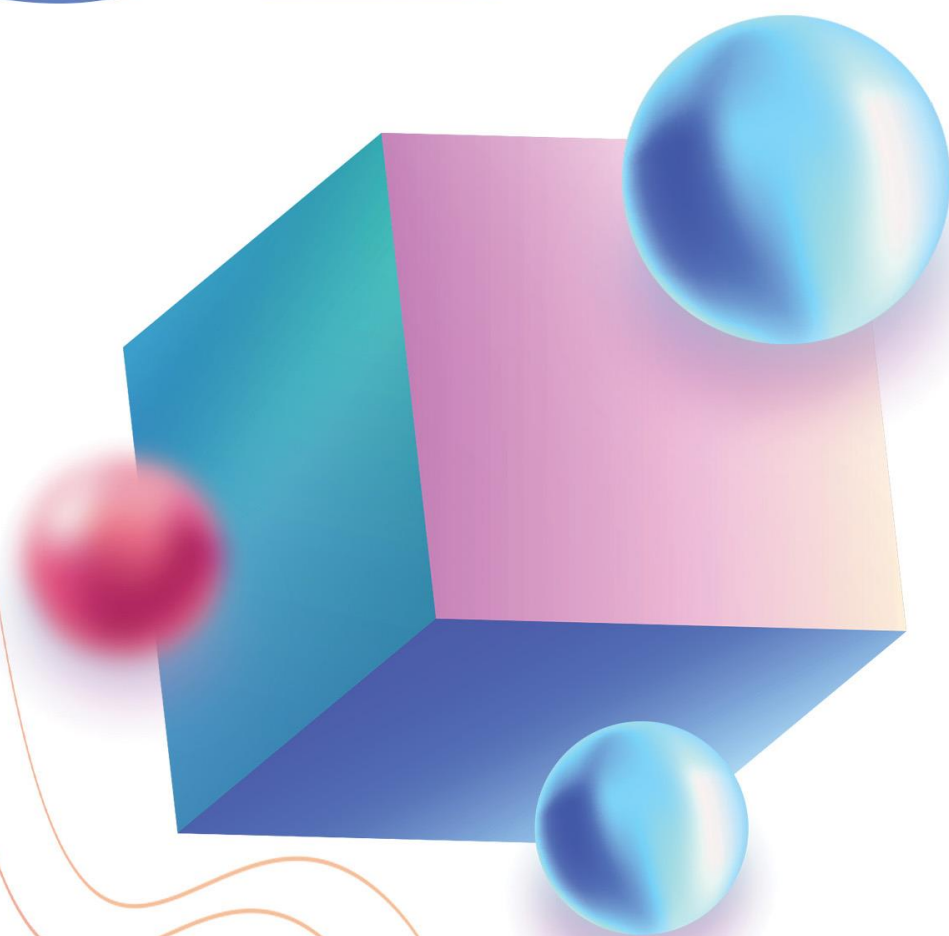


GRAFIKA KOMPUTER 3D



Fadlisyah, S.Si., M.T.
Saïd Fadlan Anshari, S.Kom., M.Kom.

GRAFIKA KOMPUTER 3D

Fadlisyah, S.Si., M.T.

Said Fadlan Anshari, S.Kom., M.Kom.



PENERBIT KBM INDONESIA

adalah penerbit dengan misi memudahkan proses penerbitan buku-buku penulis di tanah air Indonesia, serta menjadi media *sharing* proses penerbitan buku.

GRAFIKA KOMPUTER 3D

Copyright @ 2026 Fadlisyah, S.Si., M.T., dan Said Fadlan Anshari, S.Kom., M.Kom

All right reserved

Penulis

Fadlisyah, S.Si., M.T.
Said Fadlan Anshari, S.Kom., M.Kom.

Desain Sampul

Eca Art

Tata Letak

Ika Puji A

Editor

Dr. Muhamad Husein Maruapey, Drs., M.Sc.
Background isi buku di ambil dari [https://www. freepik.com/](https://www.freepik.com/)

Official

Depok, Sleman-Jogjakarta (Kantor)
Penerbit Karya Bakti Makmur (KBM) Indonesia
Anggota IKAPI/No. IKAPI 279/JTI/2021
081357517526 (Tlpn/WA)

Website

<https://penerbitkbm.com>
[www. penerbitbukumurah.com](http://www.penerbitbukumurah.com)

Email

naskah@penerbitkbm.com

Distributor

[https://penerbitkbm. com/toko-buku/](https://penerbitkbm.com/toko-buku/)

Youtube

Penerbit KBM Sastrabook

Instagram

@penerbit. kbmindonesia
@penerbitbukujogja

ISBN: 978-634-278-322-1

Cetakan ke-1, April 2026
21 x 29,7 cm, 141 halaman

Isi buku diluar tanggungjawab penerbit
Hak cipta merek KBM Indonesia sudah terdaftar di DJKI-Kemenkumham
dan isi buku dilindungi undang-undang.

Dilarang keras menerjemahkan, memfotokopi, atau
memperbanyak sebagian atau seluruh isi buku ini
tanpa seizin penerbit karena beresiko sengketa hukum

Sanksi Pelanggaran Pasal 113

Undang-Undang No. 28 Tahun 2014 Tentang Hak Cipta

- i. Setiap Orang yang dengan tanpa hak melakukan pelanggaran hak ekonomi sebagaimana dimaksud dalam Pasal 9 ayat (1) huruf i untuk Penggunaan Secara Komersial dipidana dengan pidana penjara paling lama 1 (satu) tahun dan/atau pidana denda paling banyak Rp 100.000.000 (seratus juta rupiah).
- ii. Setiap Orang yang dengan tanpa hak dan/atau tanpa izin Pencipta atau pemegang Hak Cipta melakukan pelanggaran hak ekonomi Pencipta sebagaimana dimaksud dalam Pasal 9 ayat (1) huruf c, huruf d, huruf f, dan/atau huruf h untuk Penggunaan Secara Komersial dipidana dengan pidana penjara paling lama 3 (tiga) tahun dan/atau pidana denda paling banyak Rp 500.000.000,00 (lima ratus juta rupiah).
- iii. Setiap Orang yang dengan tanpa hak dan/atau tanpa izin Pencipta atau pemegang Hak Cipta melakukan pelanggaran hak ekonomi Pencipta sebagaimana dimaksud dalam Pasal 9 ayat (1) huruf a, huruf b, huruf e, dan/atau huruf g untuk Penggunaan Secara Komersial dipidana dengan pidana penjara paling lama 4 (empat) tahun dan/atau pidana denda paling banyak Rp 1.000.000.000,00 (satu miliar rupiah).
- iv. Setiap Orang yang memenuhi unsur sebagaimana dimaksud pada ayat (3) yang dilakukan dalam bentuk pembajakan, dipidana dengan pidana penjara paling lama 10 (sepuluh) tahun dan/atau pidana denda paling banyak Rp 4.000.000.000,00 (empat miliar rupiah)

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

Kata Pengantar

Daftar Isi

BAB 1 – PENDAHULUAN

1.1	Standar Pengembangan Program 3D	1
1.2	Source File	1
1.3	Proyek Utama	2

BAB 2 – GARIS 3D VISUAL

2.1	Persiapan Awal	3
2.2	Source File	3
2.3	Proyek Utama	6

BAB 3 – SOLAR SYSTEM 3D

3.1	Persiapan Awal	17
3.2	Source File	17
3.3	Proyek Utama	24

BAB 4 – FONT

4.1	Persiapan Awal	39
4.2	Proyek Utama	39

BAB 5 – FISH DAN VISUAL AQUA

5.1	Persiapan Awal	50
5.2	Source File Animate	50
5.3	Source File Boid	53
5.4	Source File ExtOpenGL	82
5.5	Source File Matrix	83
5.6	Source File Model	88
5.7	Source File Setup	109
5.8	Source File Textures	116
5.9	Source File Timer	125
5.10	Source File Vector	126
5.11	Proyek Utama	129

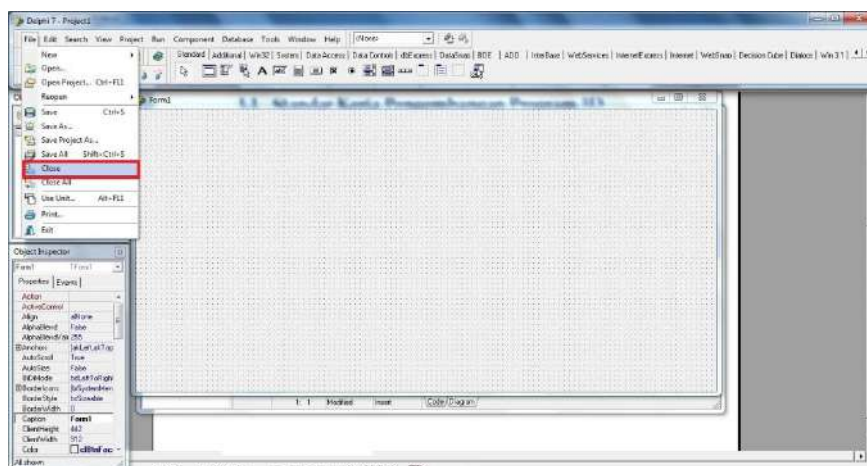
BAB I PENDAHULUAN

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.

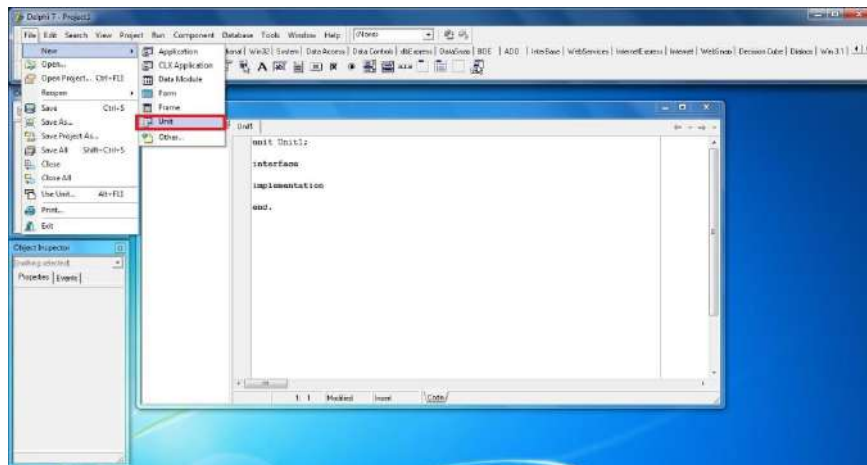
Sebelum masuk kepada penjelasan akan pendekatan tersebut, maka setelah muncul IDE utama Delphi, lakukan langkah berikut :

File dan Close



1.2 Source File

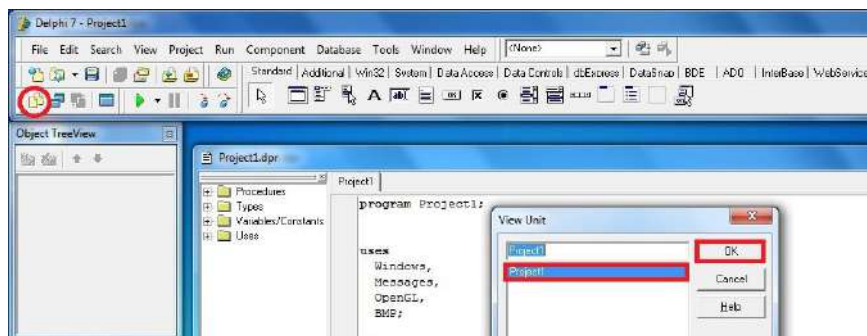
Untuk membangun Delphi Source File, maka sorot ke menu File, lalu sorot New, dan klik Unit.



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

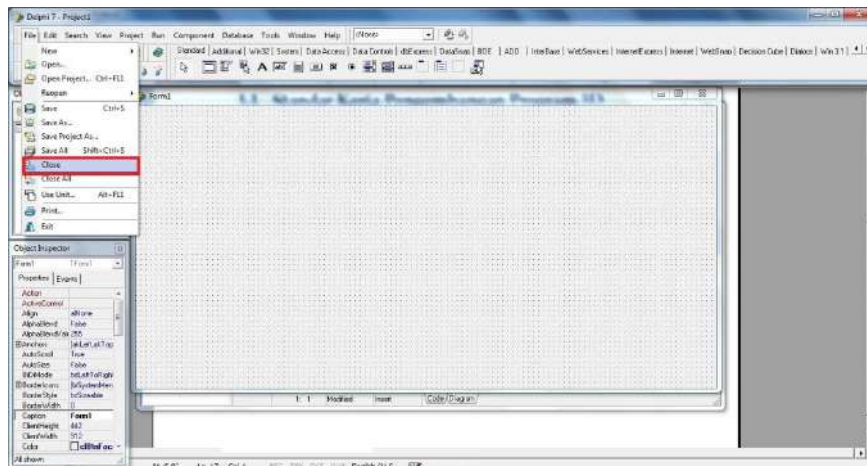
GARIS 3D VISUAL

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.

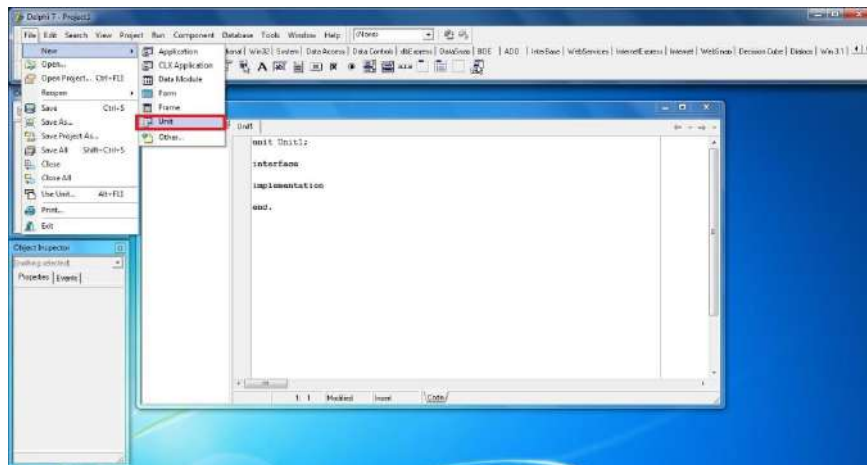
Sebelum masuk kepada penulisan koding program source file, maka setelah muncul IDE utama Delphi, lakukan langkah berikut :

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;
end;

glGenTextures(1, Texture);
glBindTexture(GL_TEXTURE_2D, Texture);
glTexEnvf(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 }

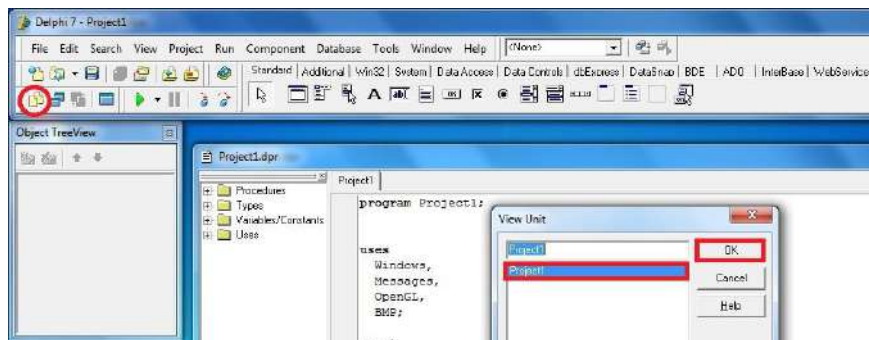
gluBuild2DMipmaps(GL_TEXTURE_2D, 3, Width, Height, GL_RGB, GL_UNSIGNED_BYTE, pData);
// glTexImage2D(GL_TEXTURE_2D, 0, 3, Width, Height, 0, GL_RGB, GL_UNSIGNED_BYTE,
pData); // Use when not wanting mipmaps to be built by OpenGL
end;

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[TAI_LENGTH-1]].X-Grid[Visited[TAI_LENGTH]].X)*Step +
Grid[Visited[TAI_LENGTH]].X,
                        (Grid[Visited[TAI_LENGTH-1]].Y-Grid[Visited[TAI_LENGTH]].Y)*Step +
Grid[Visited[TAI_LENGTH]].Y,
                        (Grid[Visited[TAI_LENGTH-1]].Z-Grid[Visited[TAI_LENGTH]].Z)*Step +
Grid[Visited[TAI_LENGTH]].Z);
                    glEnd();
                end;
            end;
        end;
    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); //Really 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;
    end;
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;
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;

// Attempts 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_OWND;      // 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;
end;

```

```

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
    iPixelFormat   := 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.

DAFTAR PUSTAKA

- Fadlisyah. 2007. *Computer Vision dan Pengolahan Citra*, Penerbit Andi. Yogyakarta. (978-979-29-0013-2)
- Fadlisyah. 2007. *Pengantar Komputer Grafik*, Penerbit Andi. Yogyakarta. (978-979-29-0060-6)
- Fadlisyah. 2008. *Pengolahan Citra*, Penerbit Graha Ilmu. Yogyakarta. (978-979-756-332-5)
- Fadlisyah. 2008. *Bahasa Assembly*, Penerbit Andi. Yogyakarta. (978-979-29-0453-6)
- Fadlisyah. 2008. *Pemrograman Mikroprosesor*, Penerbit Graha Ilmu. Yogyakarta. (978-979-756-352-3)
- Fadlisyah. 2008. *Robotika : Reasoning, Learning, Planning*, Penerbit Graha Ilmu. Yogyakarta. (978-979-756-353-0)
- Fadlisyah. 2009. *Robot Visi*, Penerbit Graha Ilmu. Yogyakarta. (978-979-756-469-8)
- Fadlisyah. 2009. *Matematika Diskrit*, Penerbit Graha Ilmu. Yogyakarta. (978-979-756-474-2)
- Fadlisyah. 2009. *Algoritma Genetik*, Penerbit Graha Ilmu. Yogyakarta. (978-979-756-483-4)
- Fadlisyah. 2010. *Panduan Praktis Komputer Grafik dan Perancangan Game*, Penerbit Graha Ilmu. Yogyakarta. (978-979-756-686-9)
- Fadlisyah. 2010. *Pemrograman Kamera PC Menggunakan Delphi*, Penerbit Graha Ilmu. Yogyakarta. (978-979-7566-88-3)
- Fadlisyah dan Rizal. 2011. *Pemrograman Computer Vision pada Video*, Penerbit Graha Ilmu. Yogyakarta. (978-979-756-737-8)
- Fadlisyah, Bustami, dan Ikhwanus. 2013. *Pengolahan Suara*, Penerbit Graha Ilmu. Yogyakarta. (978-979-756-929-7)
- Bustami, Dahlan Abdullah, dan Fadlisyah. 2014. *Statistika : terapannya di informatika*, Penerbit Graha Ilmu. Yogyakarta. (978-979-756-929-7)
- Fadlisyah. 2020. *Pengolahan Citra dan Suara*, Penerbit Graha Ilmu. Yogyakarta. (978-623-7110-74-3)
- Fadlisyah. 2022. *Pemrograman Visual*, Penerbit Graha Ilmu. Yogyakarta. (978-623-6433-63-8)
- Fadlisyah dan Muhammad Fikry. 2025. *Pemrograman Video*, Penerbit Graha Ilmu. Yogyakarta (978-634-7198-01-3)

RIWAYAT HIDUP PENULIS

Fadlisyah alumnus FMIPA Universitas Padjadjaran dan S2 Teknik Komputer USU yang saat ini berprofesi sebagai pengajar di Jurusan Teknik Informatika dan pernah menjabat sebagai Kepala Pusat Studi Kecerdasan Robot Universitas Malikussaleh. Selain aktif mengajar, beliau juga anggota dari Himpunan Ulama Dayah Aceh (HUDA) wilayah kota Lhokseumawe, anggota Majelis Tasawuf, Tauhid, dan Fiqh (TASTAFI) Lhokseumawe, sekretaris Persatuan Tarbiyah Islamiyah (PERTI). Mengasuh mata kuliah Pengolahan Citra, Pengolahan Suara dan Grafika Komputer.

Said Fadlan Anshari lahir di Kota Langsa, Aceh, pada tahun 1994. Ia menyelesaikan pendidikan Magister Informatika di Universitas Telkom pada tahun 2019 dan saat ini berprofesi sebagai dosen di Universitas Malikussaleh sejak tahun 2022. Beliau mengampu beberapa mata kuliah di Program Studi Teknik Informatika, antara lain Kecerdasan Komputasional, Pengelolaan Data Besar, Visualisasi Data dan Dashboard, serta Basis Data. Selain aktif mengajar, ia juga produktif dalam menulis dan telah menghasilkan sejumlah publikasi nasional dan internasional, baik dalam bentuk jurnal penelitian maupun pengabdian kepada masyarakat. Ia dapat dihubungi melalui email saidfadlan@unimal.ac.id.

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.