

## BAB I

### PENDAHULUAN

#### 1.1 Sekilas tentang Grafika Komputer

##### 1.1.1 Tujuan Pengajaran :

1. Mahasiswa dapat membuat gambar menggunakan OpenGL dan bahasa C++
2. Mahasiswa dapat membuat gambar 2D dan gambar 3D
3. Mahasiswa dapat menerapkan prinsip-prinsip transformasi 2D dan 3D

##### 1.1.2 Materi Grafika Komputer

1. Pengenalan mengenai konsep-konsep dasar dari grafika computer beserta peralatan yang digunakan untuk keperluan grafika computer.
2. Pengenalan mengenai teknik menggambar dengan menggunakan cara primitif yang merupakan dasar dari setiap gambar, seperti titik, garis, dll

##### 1.1.3 Perangkat yang digunakan

1. Komputer
2. Sistem Operasi Windows
3. Microsoft Visual C++
4. OpenGL sebagai library untuk grafika computer

#### 1.2 Apa itu grafika Komputer

Grafika computer adalah gambar atau grafik yang dihasilkan oleh komputer. Teknik-teknik yang dipelajari dalam grafika computer adalah teknik-teknik bagaimana membuat

atau menciptakan gambar dengan menggunakan komputer. Ada perbedaan antara photo dan gambar, dimana pada photo semua detail objek terlihat sedangkan gambar tidak dapat memperlihatkan semua detail yang ada tetapi hanya detail yang dianggap penting dalam menunjukkan pola suatu gambar.

Adapun beberapa program sederhana sampai program yang sangat kompleks guna membuat gambar computer seperti Paint, Microsoft Photo Editor, Adobe Photoshop, Maya, Autocad, 3D Studio max, dan lain-lain.

Untuk membuat program-program diatas dapat digunakan berbagai macam bahasa pemrograman dengan library grafis yang sesuai. Dalam modul ini digunakan bahasa pemrograman **C++**. Bila menggunakan system operasi windows maka menggunakan bahasa pemrograman **C++** dan bila menggunakan system operasi Linux maka menggunakan **gcc** atau **g++**. Sedangkan library grafik yang digunakan dalam modul ini adalah **OpenGL**.

### 1.3 Bidang Yang berhubungan dengan Grafika Komputer

Bidang –bidang yang berhubungan dengan grafika computer diperlukan untuk :

1. Seni, entertainment, dan publishing, seperti produksi film, animasi, special effect, game computer, web browsing, dll
2. Image processing atau pengolahan citra digital, yang dalam hal ini grafika computer dapat digunakan didalam coding untuk pemindahan data citra menjadi data vector yang banyak digunakan dalam keperluan GIS.
3. Menampilkan simulasi, yang dihasilkan berupa grafik data atau grafik visualisasi proses, seperti dalam visualisasi proses pengaturan lampu lalu lintas.
4. CAD ( Computer Aided Design ). Satu program CAD yang paling banyak digunakan adalah AUTOCAD, yang digunakan untuk desain pola atau desain layout.

### 1.4 Elemen Dasar Grafika Komputer

Beberapa elemen dasar dari grafika computer adalah ;

**1. Polylines**

Adalah deretan garis lurus yang berhubungan. Polyline ini merupakan dasar dari pembuatan grafik.

**2. Text**

Menunjukkan pola-pola huruf pada computer yang menyebabkan layout dari hasil editing dapat menghasilkan banyak variasi tulisan.

**3. Filled Region**

Adalah bagaimana member warna atau pattern pada sebuah luasan.

**4. Raster Image**

Adalah penyajian gambar menggunakan matriks dari setiap sel gambar, dimana sebuah gambar didefinisikan sebagai array dari besar-besaran numeric. Model data ini banyak digunakan untuk keperluan pengolahan citra.

## BAB II

### PRIMITIVE DRAWING

#### 2.1 Materi

Program Dasar dengan OpenGL

- Menggambar Titik ( GL\_POINTS )
- Menggambar Garis ( GL\_LINES )
- Menggambar Polyline ( GL\_LINE\_STRIP )
- Menggambar Polygon ( GL\_LINE\_LOOP )
- Pewarnaan ( glColor )

#### 2.2 Struktur Dasar Program Grafik dengan OpenGL

Pada pemrograman Open GL dibutuhkan sebuah kerangka utama yang dibutuhkan untuk membuat tampilan grafis yang ingin kita buat yang biasa disebut dengan program utama.

Program utama yang diperlukan bisa anda lihat seperti dibawah ini :

```
#include <glut.h>
void userdraw()
{
    static int tick=0;
    /*program grafik ditulis disini*/
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    userdraw();
    glutSwapBuffers();
}
int main(int argc, char **argv)
{
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB);
    glutInitWindowPosition(100,100);
    glutInitWindowSize(640,480);
    glutCreateWindow("Program Grafikku");
    glClearColor(1.0,1.0,1.0,0.0);
```

```

gluOrtho2D(0.,640.,-240.,500.);
glutIdleFunc(display);
glutDisplayFunc(display);
glutMainLoop();
return 0;
}

```

Pada program utama diatas dapat dijelaskan fungsi dari perintah – perintah yang ada dengan penjelasan sebagai berikut :

```

glutInitWindowPosition(100,100);
glutInitWindowSize(640,480);

```

Membuat windows dengan ukuran (640,480) dengan titik kiri atas jendela diletakkan pada posisi (100,100) di layar komputer. Penentuan ukuran layar tidak harus sama dengan di atas tetapi bisa disesuaikan dengan ukuran resolusi monitor anda masing–masing.

```

glClearColor(1.0,1.0,1.0,0.0);

```

Mendefinisikan warna dari windows yang dibuat dengan warna (1,1,1) yaitu warna putih. ( Penkodean warna dapat anda lihat pada halaman tentang warna)

```

gluOrtho2D(0.,640.,-240.,240.);

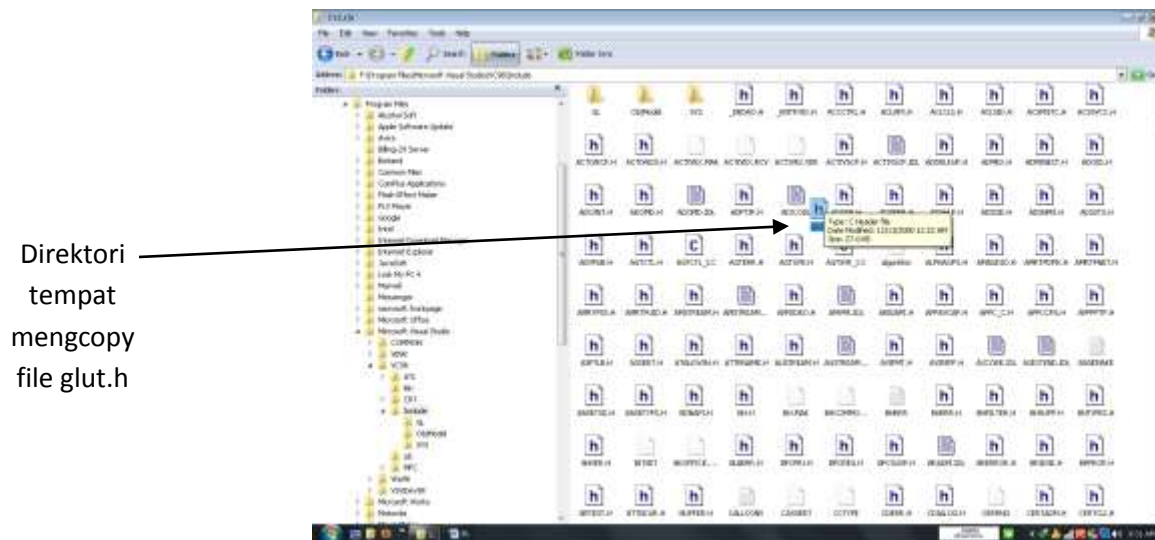
```

Mendefinisikan besarnya sistem koordinat dengan range sumbu x adalah [0,640] dan range untuk sumbu y adalah [-240,240]. Pada perintah diatas adalah untuk ukuran layar untuk memposisikan objek di layar yang telah kita buat.

### 2.2.1 Memulai Program Open GL

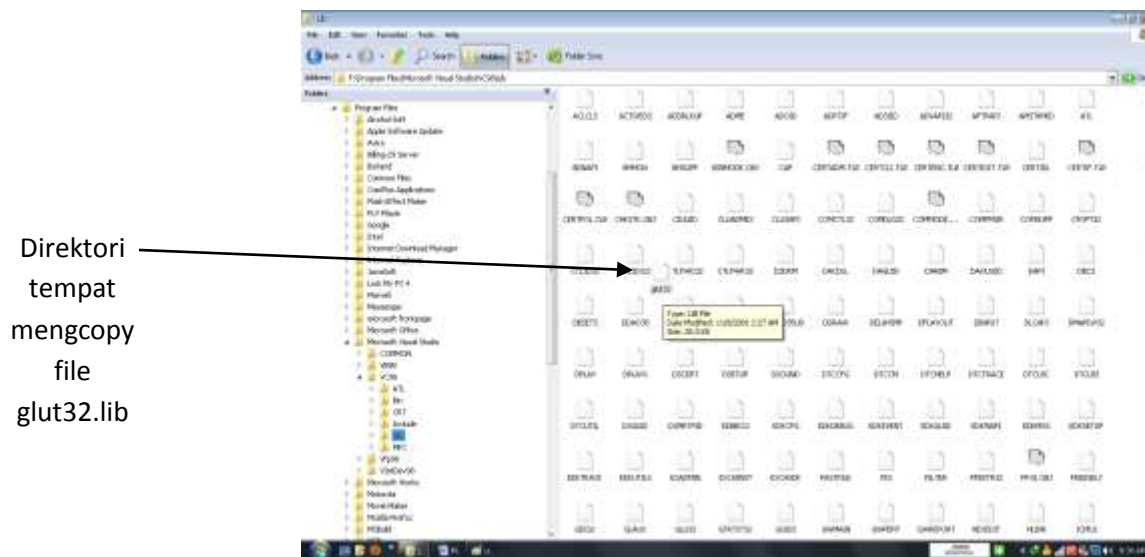
Berikut Langkah –langkah yang harus dilakukan sebelum memulai program Open GL :

1. Software yang digunakan adalah Visual Studio 6.0 dengan pemrograman Visual C++ dan boleh menggunakan program dengan versi terbaru.
2. Pastikan file –file yang dibutuhkan oleh openGL seperti **glut.h** , **glut32.lib**, dan **glut32.dll** telah ditambahkan pada directori yang telah ditentukan, berikut penjelasannya :
  - a. Copykan file **glut.h** ke direktori **C:\Program Files\Microsoft Visual Studio\VC98\Include**



**GAMBAR 2.1** Direktori tempat mengcopykan file glut.h

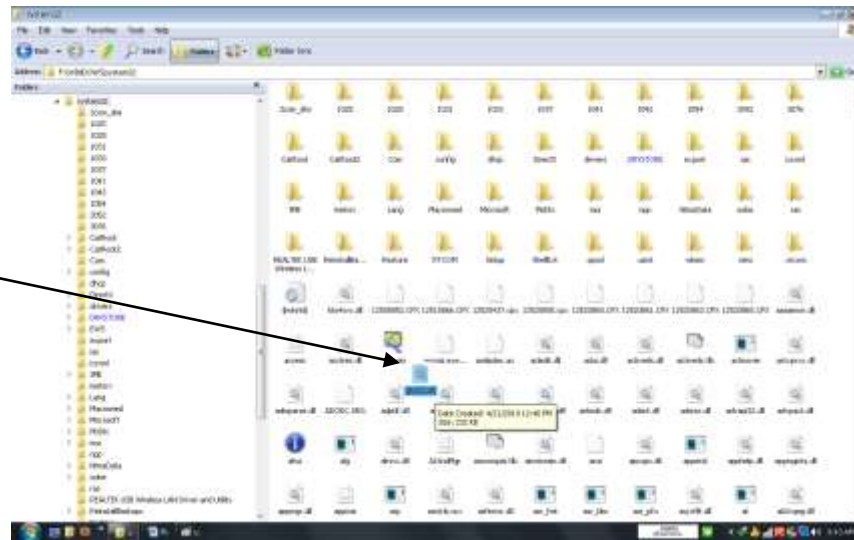
- b. Copykan file **glut32.lib** ke direktori **C:\Program Files\Microsoft Visual Studio\VC98\lib**



**GAMBAR 2.2** Direktori tempat mengcopykan file glut32.lib

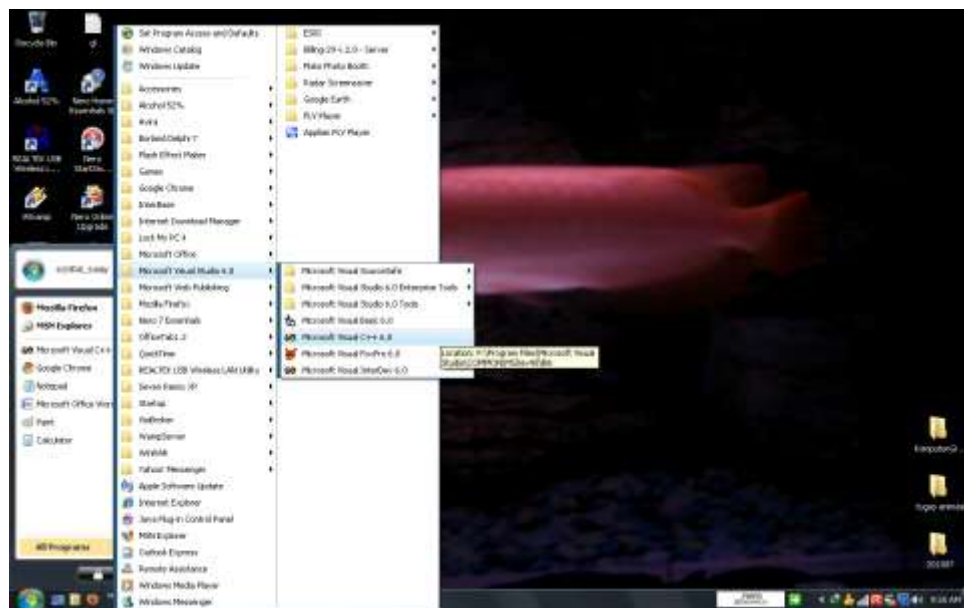
- c. Copykan file **glut32.dll** ke direktori **C:\WINDOWS\system32**

Direktori  
tempat  
mengcopy  
file  
glut32.dll



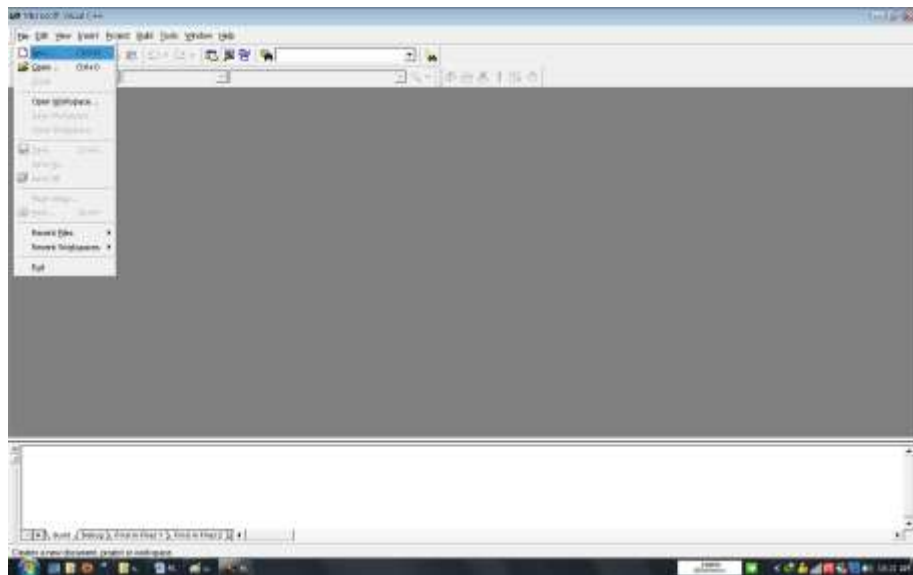
**GAMBAR 2.3** Direktori tempat mengcopykan file glut32.dll

3. Setelah ketiga file di atas dicopykan maka silahkan membuka program Visual C++



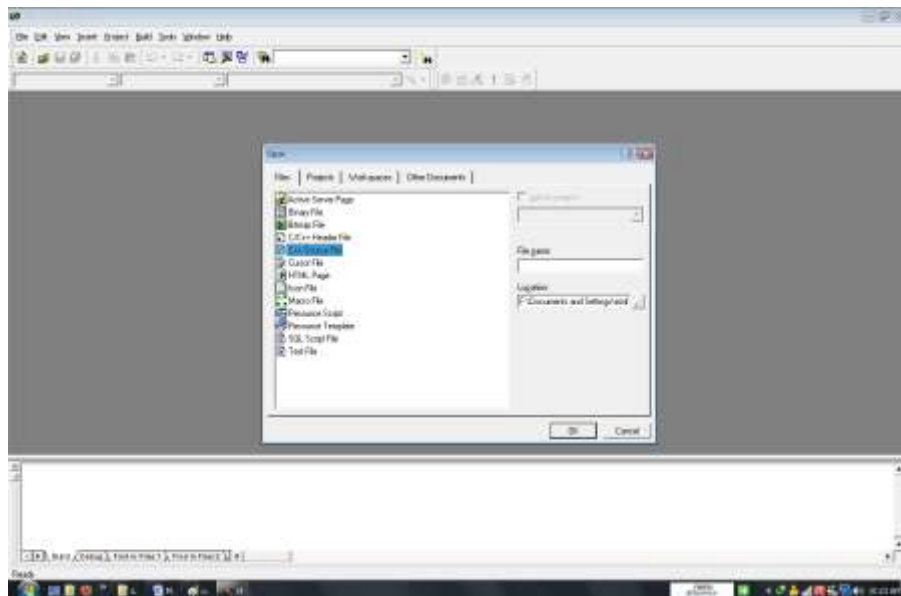
**GAMBAR 2.4** Memulai Program C++

4. Setelah terbuka pilih Menu file -> New



**GAMBAR 2.5** Pilih File Menu New

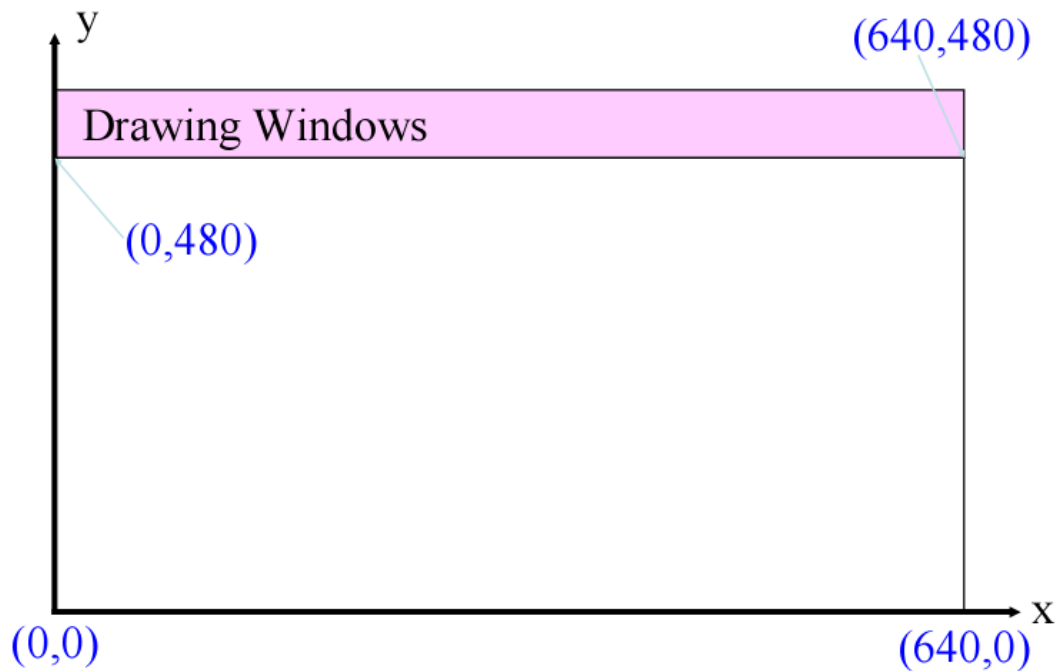
5. Pilih tab file --> pilih C++ Source File



**GAMBAR 2.6** Pilih Tab File dan Pilih C++ Source File



### 2.3 Sistem Koordinat



### 2.4 Menggambar Titik

Prosedur untuk membuat titik, namun pada pembuatan objek primitive prosedur ini tidak perlu diikuti.

**`glVertex2i(x,y)`**

Untuk menggambar titik diposisi (x,y) dimana x dan y didefinisikan sebagai bilangan bulat (integer).

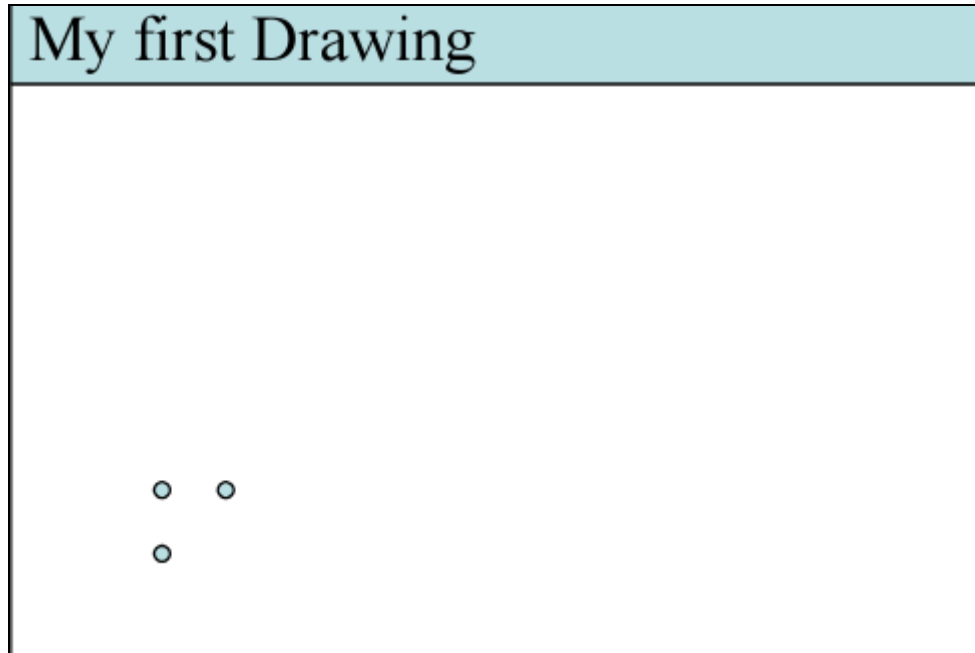
**`glVertex2f(x,y)`**

**`glVertex2d(x,y)`**

Untuk menggambar titik diposisi (x,y) dimana x dan y didefinisikan sebagai bilangan pecahan (float/double).

Berikut program untuk menggambar titik :

```
glBegin(GL_POINTS);
    glVertex2i(100,50);
    glVertex2i(100,130);
    glVertex2i(150,130);
glEnd();
```



Fungsi untuk membuat titik :

```
void drawDot(int x, int y)
{
    glBegin(GL_POINTS);
    glVertex2i(x,y);
    glEnd();
}
```

Fungsi ini digunakan bila x dan y didefinisikan sebagai integer

```
void drawDot(float x, float y)
{
    glBegin(GL_POINTS);
    glVertex2f(x,y);
    glEnd();
}
```

Fungsi ini digunakan bila x dan y didefinisikan sebagai float

Untuk mengubah ukuran titik dapat menggunakan perintah **GLPointSize(UKURANTITIK);** Bila ditulis **glPointSize(4)** maka besar titiknya adalah **4X4 PIXEL**. Bila tidak digunakan maka ukuran titiknya adalah **1 PIXEL**.

### Contoh titik dengan ukuran dan warna yang berbeda :

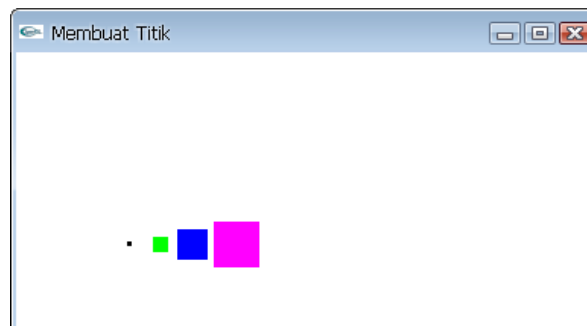
```
void userdraw()
{
    static int tick=0;
    glColor3f(0.,0.,0.);
    glPointSize(3);
    glBegin(GL_POINTS);
        glVertex2f(125,150);
    glEnd();

    glColor3f(0.,1.,0.);
    glPointSize(10);
    glBegin(GL_POINTS);
        glVertex2f(160,150);
    glEnd();

    glColor3f(0.,0.,1.);
    glPointSize(20);
    glBegin(GL_POINTS);
        glVertex2f(195,150);
    glEnd();

    glColor3f(1.,0.,1.);
    glPointSize(30);
    glBegin(GL_POINTS);
        glVertex2f(245,150);
    glEnd();
}
```

Hasilnya :

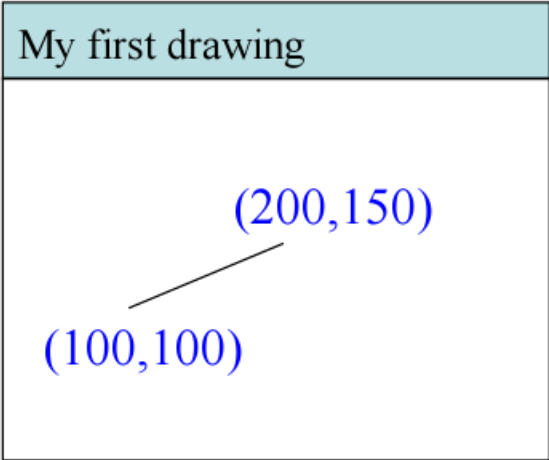


## 2.5 Menggambar Garis

Untuk membuat garis diperlukan library `GL_LINES` dengan menyatukan titik awal dan titik akhir dari garis. Untuk mengubah ukuran garis dapat menggunakan perintah **`glLineWidth(TEBALGARIS)`**; Bila ditulis **`glLineWidth(4)`** maka tebalnya garis adalah **4X4 PIXEL**. Bila tidak digunakan maka ukuran titiknya adalah **1 PIXEL**.

```
glBegin(GL_LINES);
    glVertex2i(100,100);
    glVertex2i(200,150);
glEnd();
```

My first drawing



Fungsi untuk menggambar garis :

```
void drawLine(int x1,int y1,int x2,int y2)
{
    glBegin(GL_LINES);
        glVertex2i(x1,y1);
        glVertex2i(x2,y2);
    glEnd();
}
```

```
void drawLine(float x1,float y1,float x2,float y2)
{
    glBegin(GL_LINES);
    glVertex2f(x1,y1);
    glVertex2f(x2,y2);
    glEnd();
}
```

Menggambar efek hujan, sama seperti menggambar bintang dengan titik acak, tetapi yang obyeknya berupa garis diagonal. Program untuk menggambar **efek hujan** ini adalah:

```
void userdraw(void)
{
    float xp,yp;
    for(int i=0;i<300;i++)
    {
        xp=640*(float)rand()/RAND_MAX;
        yp=480*(float)rand()/RAND_MAX;
        glColor3f(1,1,1);
        glBegin(GL_LINES);
        glVertex2f(xp,yp);
        glVertex2f(xp+8,yp-8);
        glEnd();
    }
}
```

Atau bisa menggunakan Program lain seperti di bawah ini :

```
static point2D_f hujan[300];
static int tick=0, toc=0;
int i;

if(tick==0){
    for(i=0;i<300;i++){
        hujan[i].x=rand()%140-0;//pengaruhi daerah yg hujan
        hujan[i].y=rand()%100-0;
    }
}

if(toc==0){
    for(i=0;i<700;i++){
        hujan[i].x-=1;
        hujan[i].y-=1;

        if (((hujan[i].x<640) || (hujan[i].y<300)))
        {
            hujan[i].x=rand()%335-0;//pengaruhi t4 keluar air hujan
        }
    }
}
```

```

                hujan[i].y=rand()%276-0;
            }
        }
    }

    glColor3f(1,1,1);

    for(i=0;i<200;i++){
        gambarGaris(hujan[i].x,hujan[i].y,hujan[i].x-3,hujan[i].y-3);
    }
    tick++;
    toc=tick%100;

```

Derasnya hujan dapat diatur dengan memperbanyak jumlah garis yang digambar atau membuat garisnya lebih panjang. Sedangkan arah gerakan air hujan dapat diatur dengan operasi penjumlahan atau pengurangan dari masing-masing nilai x dan nilai y pada setiap garis. Permainan sudut akan menjadi lebih baik karena bisa mengubah arah hujan menjadi lebih realistis dengan memberikan afek angin.

Contoh membuat garis dengan berbagai ukuran dan warna :

```

void userdraw()
{
    static int tick=0;
    glColor3f(0.,0.,0.);
    glLineWidth(1);
    glBegin(GL_LINES);
        glVertex2f(125,150);
        glVertex2f(125,450);
    glEnd();

    glColor3f(0.,1.,0.);
    glLineWidth(3);
    glBegin(GL_LINES);
        glVertex2f(160,150);
        glVertex2f(160,450);
    glEnd();

    glColor3f(0.,0.,1.);
    glLineWidth(8);
    glBegin(GL_LINES);
        glVertex2f(195,150);
        glVertex2f(195,450);
    glEnd();

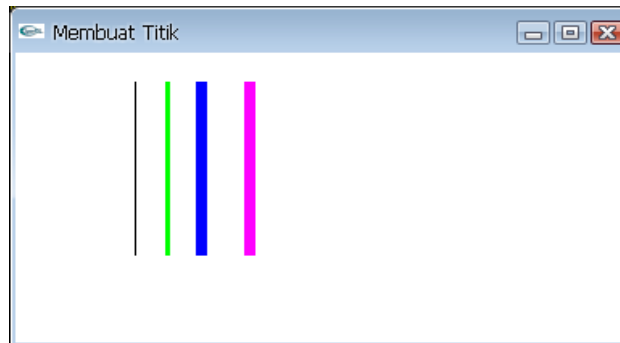
    glColor3f(1.,0.,1.);
    glLineWidth(100);
    glBegin(GL_LINES);
        glVertex2f(245,150);

```

```

        glVertex2f(245,450);
    glEnd();
}

```



## 2.6 Membuat Polyline

Polyline adalah sekumpulan garis yang terhubung satu dengan yang lainnya hingga membentuk sebuah obyek gambar.

```

glBegin(GL_LINE_STRIP);
    glVertex2i(x1,y1);
    glVertex2i(x2,y2);
    glVertex2i(x3,y3);
    .....
    glVertex2i(xn,yn);
glEnd();

```

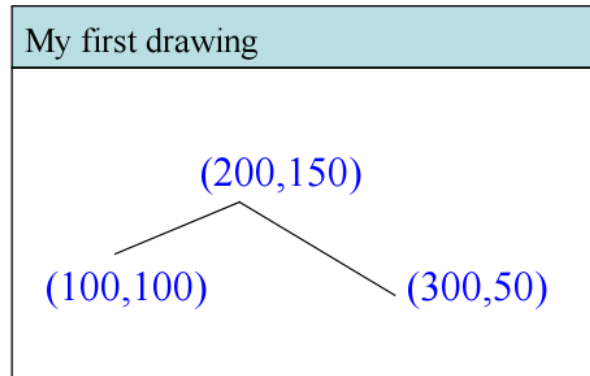
Contoh input :

```

glBegin(GL_LINE_STRIP);
    glVertex2i(100,100);
    glVertex2i(200,150);
    glVertex2i(300,50);
glEnd();

```

Output Program :



## 2.7 Membuat Polygon

Polygon adalah sekumpulan garis yang terhubung satu dengan yang lainnya dan berbentuk kurva tertutup hingga membentuk sebuah obyek gambar.

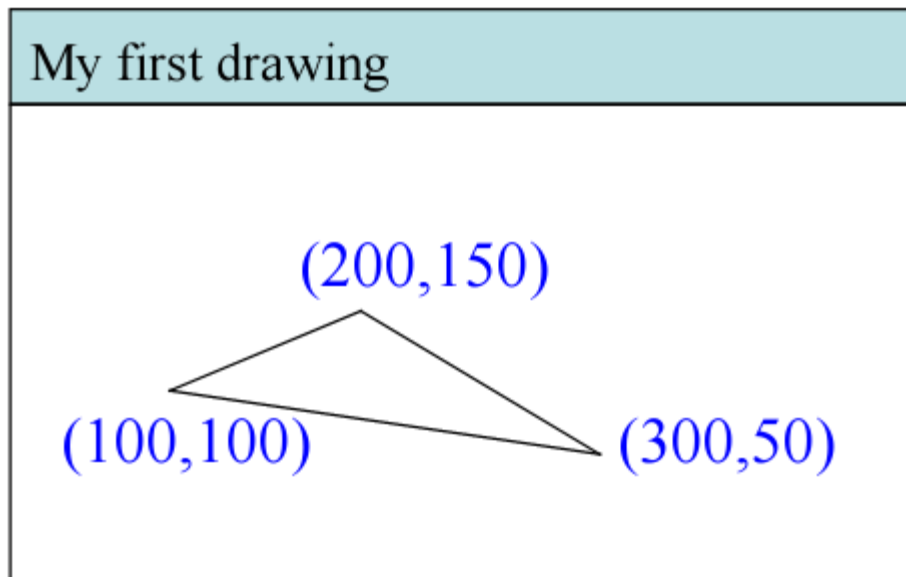
```
glBegin(GL_LINE_LOOP);
    glVertex2i(x1,y1);
    glVertex2i(x2,y2);
    glVertex2i(x3,y3);
    .....
    glVertex2i(xn,yn);
glEnd();
```

Contoh input :

```
glBegin(GL_LINE_LOOP);
    glVertex2i(100,100);
    glVertex2i(200,150);
    glVertex2i(300,50);
glEnd();
```

Output program :





Pemberian warna :

```
glColor3f (red, green, blue) ;
```

Red, green, blue bervariasi diantara 0. S/d 1.

```
glColor3f (0., 0., 0.) ; //black  
glColor3f (0., 0., 1.) ; //blue  
glColor3f (0., 1., 0.) ; //green  
glColor3f (0., 1., 1.) ; //cyan  
glColor3f (1., 0., 0.) ; //red  
glColor3f (1., 0., 1.) ; //magenta  
glColor3f (1., 1., 0.) ; //yellow  
glColor3f (1., 1., 1.) ; //white
```

**Contoh program : ( Membuat Grid )**

```
void userdraw() {
    glColor3f(0,0,1);
    for (float x=-100;x<=100;x+=10)
    {
        glBegin(GL_LINES);
        glVertex2f(x,-100);
        glVertex2f(x,100);
        glEnd();
        glBegin(GL_LINES);
        glVertex2f(-100,x);
        glVertex2f(100,x);
        glEnd();
    }

    glColor3f(0,1,1);
    glBegin(GL_LINES);
    glVertex2f(-100,0);
    glVertex2f(100,0);
    glEnd();
    glBegin(GL_LINES);
    glVertex2f(0,-100);
    glVertex2f(0,100);
    glEnd();
}
```

## BAB III

### MEMBUAT GRAFIK 2 DIMENSI

#### 3.1 Materi

- a. Definisi Obyek Grafik 2-D
- b. Polyline
- c. Mewarnai Area ( *Fill Polygon* )
- d. Membangun obyek Grafik 2-D
- e. Animasi 2-D

#### 3.2 Definisi Obyek Grafik 2-D

Adalah sekumpulan titik-titik 2-D yang dihubungkan dengan garis lurus baik berupa polyline, polygon, atau kurva, yang secara komputasi dinyatakan sebagai array 1-D atau linked-list.

##### 3.2.1 Mendefinisikan Titik 2-D

1. Mendefinisikan struktur dari titik 2-D ( *Point2D\_t* )
2. Mendefinisikan struktur warna ( *Color\_t* )
3. Mendefinisikan struktur dari obyek grafik 2-D sebagai array dari titik 2-D ( *Object2D\_t* ).

```
typedef struct {
    float x;
    float y;
} point2D_t;
```

Definisi ini digunakan bila titik didefinisikan dalam sistem koordinat yang menggunakan bilangan pecahan (*float*)

```
typedef struct {
    int x;
    int y;
} point2D_t;
```

Definisi ini digunakan bila titik didefinisikan dalam sistem koordinat yang menggunakan bilangan bulat (*integer*)

### 3.2.2 Mendefinisikan Warna

```
typedef struct {
    float r;
    float g;
    float b;
} color_t;
```

Warna terdiri dari 3 elemen warna yaitu red (r), green (g) dan blue (b) yang nilainya antara 0 dan 1

Fungsi untuk memberi warna pada obyek grafik:

```
void setColor(color_t col)
{
    glColor3f(col.r, col.g, col.b);
}
```

### 3.2.3 Mendefinisikan Obyek Grafik 2-D

Definisi obyek ini dapat dituliskan pada *function* **userdraw** secara langsung dengan menyatakannya sebagai array dari titik 2-D. Sebagai contoh untuk menyatakan obyek shape dapat dituliskan:

```
Point2D_t shape[1000]
```

Untuk menyatakan obyek bunga dapat dituliskan:

```
Point2D_t bunga[360]
```

### 3.2.4 Polyline

Polyline adalah suatu fungsi yang digunakan untuk menggambarkan obyek 2-D yang sudah didefinisikan di depan.

```
void drawPolyline(point2D_t pnt[],int n)
{
    int i;
    glBegin(GL_LINE_STRIP);
        for (i=0;i<n;i++) {
            glVertex2f(pnt[i].x, pnt[i].y);
        }
    glEnd();
}
```

### 3.2.5 Polygon

Polygon adalah suatu fungsi yang mirip dengan polyline hanya saja hasilnya adalah kurva tertutup, sedangkan polyline hasilnya kurva terbuka

```
void drawPolygon(point2D_t pnt[],int n)
{
    int i;
    glBegin(GL_LINE_LOOP);
        for (i=0;i<n;i++) {
            glVertex2f(pnt[i].x, pnt[i].y);
        }
    glEnd();
}
```

### 3.2.6 Fill Polygon

Fungsi ini digunakan untuk mewarnai sebuah polygon dengan warna tertentu

```
void fillPolygon(point2D_t pnt[],int n,
color_t color)
{
    int i;
    setColor(color);
    glBegin(GL_POLYGON);
        for (i=0;i<n;i++) {
            glVertex2f(pnt[i].x, pnt[i].y);
        }
    glEnd();
}
```

### 3.2.7 Gradata Polygon

Fungsi ini digunakan untuk mewarnai sebuah polygon dengan warna-warna yang bergradasi dari suatu warna ke warna lainnya

```
void GradataPolygon(point2D_t pnt[],int
n, color_t color)
{
    int i;
    glBegin(GL_POLYGON);
        for (i=0;i<n;i++) {
            setColor(color);
            glVertex2f(pnt[i].x, pnt[i].y);
        }
    glEnd();
}
```

### 3.3 Membangun Objek Grafik 2-D

#### 3.3.1 Membangun Objek grafik 2-D secara langsung

Membuat obyek grafik 2-D secara langsung bisa dilakukan pada function `userdraw()` dengan menyatakan secara langsung koordinat titik-titiknya

```
void userdraw()  
{  
    Point2D_t kotak[4]={ {100,100}, {300,100},  
                          {300,200}, {100,200} };  
    Polygon(kotak,4);  
}
```

Program ini digunakan untuk membuat kotak

```
void userdraw()  
{  
    point2D_t kotak[4]={ {100,100}, {300,100},  
                          {300,200}, {100,200} };  
    setColor(1,1,0);  
    drawPolygon(kotak,4);  
}
```

Program ini menghasilkan kotak yang garisnya berwarna kuning.

```
void userdraw()
{
    point2D_t kotak[4]={{100,100},{300,100},
        {300,200},{100,200}};
    color_t magenta={0,1,1};
    fillPolygon(kotak,4,magenta);
}
```

Program ini menghasilkan kotak yang isinya berwarna magenta.

```
void userdraw()
{
    point2D_t kotak[4]={{100,100},{300,100},
        {300,200},{100,200}};
    color_t magenta={0,1,1};
    fillPolygon(kotak,4,magenta);
    setColor(1,1,0);
    drawPolygon(kotak,4);
}
```

Program ini menghasilkan kotak yang isinya berwarna magenta dan garis tepinya berwarna kuning.

```
void userdraw()
{
    Point2D_t bintang[10]={{80,146},{99,90},
        {157,90},{110,55},{128,1},
        {80,34},{32,1},{54,55},
        {3,90},{63,90}};
    Polygon(bintang,10);
}
```

Hasilnya adalah:





**Berikut prosedur untuk lingkaran :**

```
void drawCircle(double r, int pos_x, int pos_y, color col)
{
    point2D_t circle[360];
    double srاد;

    for (int i = 0; i < 360; i++)
    {
        srاد = i * 3.14 / 180;
        circle[i].x = (float)(r * cos (srاد)) + pos_x;
        circle[i].y = (float)(r * sin (srاد)) + pos_y;
    }
    fillPolygon(circle, 360, col);
}
```

**Dan dibawah ini untuk membuat lingkarannya :**

```
void userdraw()
{
    point2D_t lingkaran[360]; // diperlukan 360 titik
    double srاد, r;
    color warnaLink = {1, 1, 0};
    for (int i = 0; i < 360; i++)
    {
        srاد = i * 3.14 / 180;
        r = sin(srاد) * 120; // untuk menentukan ukuran lingkaran
        lingkaran[i].x = (float)(r * cos (srاد) * 1.3) + 713; // koordinat x
        lingkaran[i].y = (float)(r * sin (srاد)) + 400; // koordinat y
    }
    fillPolygon(lingkaran, 360, warnaLink);
    setColor(1, 0, 0);
    drawPolygon(lingkaran, 360);
}
```

### 3.3.2 Membangun Objek 2-D Dengan Persamaan Matematik

Dengan persamaan matematik  $y=f(x)$  dapat digambarkan kurva dengan variasi bentuk yang menarik seperti sinus, cosinus, exponential dan logaritma, atau fungsi gabungannya. Bentuk persamaan matematik yang menarik untuk dibuat adalah persamaan matematik dengan menggunakan sistem koordinat polar.

$$\begin{aligned} r &= f(\theta) \\ x &= r \cdot \cos(\theta) \\ y &= r \cdot \sin(\theta) \end{aligned}$$

$\theta$  adalah sudut yang berjalan dari 0 s/d 360 yang dinyatakan dalam radian (0 s/d  $2\pi$ ). Macam-macam  $r=f(\theta)$  dapat menghasilkan gambar yang bervariasi.

Contoh Fungsi Polar :

$r=\sin(\theta)$	Lingkaran
$r=\sin(2\theta)$	Rose 4 daun
$r=\sin(3\theta)$	Rose 3 daun
$r=\sin(n\theta)$	Rose n daun bila n bilangan prima
$r=\theta$	Spiral

Catatan : Masih banyak variasi fungsi yang lain yang dapat dibangun dengan menggunakan koordinat polar ini.

Berikut kode program untuk membangun obyek grafik 2-D dengan menggunakan koordinat polar.

```
void userdraw()
{
    Point2D_t shape[360];
    double srاد,r;
    for(int s=0;s<360;s++)
    {
        srاد=s*3.14/180;
        r=sin(5*srاد);
        shape[s].x=(float)(r*cos(srاد));
        shape[s].y=(float)(r*sin(srاد));
    }
    Polygon(shape,360);
}
```

Fungsi sin(5θ) yang menghasilkan rose 5 daun.

### Contoh Obyek 2 Dimensi (DrawCircle[])

Secara definisi, lingkaran adalah segi banyak. Jadi persoalannya disini adalah bagaimana membuat sebuah bangun segi banyak, dimana setiap segi mempunyai sudut yang sama besar. Konversi sistem koordinat sudut menjadi sistem koordinat Cartesian adalah berikut :

$$\begin{aligned}x &= r \cdot \cos(a) \\ y &= r \cdot \sin(a)\end{aligned}$$

```
void drawCircle(float r, int n){
    point2D_t p[360];
    float a=6.28/n;
    for(int i=0;i<n;i++){
        p[i].x=r*(float)cos((float)i*a);
        p[i].y=r*(float)sin((float)i*a);
    }
    drawPolygon(p,n);
}
```

Pada userdraw() :  
drawCircle(30,40);



### Contoh Obyek 2 Dimensi (DrawEllipse[])

Menggambar ellipse sama dengan menggambar lingkaran hanya jari-jari untuk sb x dan sb y berbeda sbb :

```
void drawEllipse(point2D_t p[],float r1,float r2, int n){
float a=6.28/n;
for(int i=0;i<n;i++){
    p[i].x=r1*cos(i*a);
    p[i].y=r2*sin(i*a);
}
}
```

Override drawEllipse() dengan titik pusat tidak berada di sumbu koordinat sbb :

```
void drawEllipse(point2D_t p[], point2D_t p0, float r1,float r2,int n){
float a=6.28/n;
for(int i=0;i<n;i++){
    p[i].x=p0.x+r1*cos(i*a);
    p[i].y=p0.y+r2*sin(i*a);
}
}
```

### Contoh Obyek 2 Dimensi (CenterPolygon[])


Pewarnaan obyek yang memiliki titik pusat, misalnya obyek lingkaran atau obyek polar lainnya (rose, spiral, dll) dapat dilakukan dengan pewarnaan pergaris ( dari titik pusat : warna putih, ketitik ke obyek lingkaran/polar : warna sembarang ) hasilnya akan didapatkan degradasi warna putih dan warna sembarang yang cantik, prosedurnya sebagai berikut :

```

void centerPolygon(point2D_t p[],point2D_t pc,color_t
col,color_t colp,int n){
for(int i=0;i<n;i++) {
    glBegin(GL_LINES);
    setColor(colp);
    glVertex2f(pc.x,pc.y);
    setColor(col);
    glVertex2f(p[i].x,p[i].y);
    glEnd();
}
}
            
```

```

void userdraw(){
point2D_t p[360];
point2D_t pusat={0.,0.};
color_t putih={1,1,1};
color_t biru={0,0,1};
createCircle(p,50,360);
centerPolygon(p,pusat,biru,putih,360);
}
            
```



### 3.4 Animasi 2 Dimensi

Yang dilakukan dalam animasi 2 dimensi :

1. Membuat objek grafik 2-D menjadi bergerak
2. Animasi yang dilakukan adalah memindahkan posisi gambar
3. Pada sistem koordinat kartesian animasi akan berefek gerakan linier (translasi), pada sistem koordinat polar akan berefek gerakan berputar (rotasi).

Pembuatan animasi 2 dimensi memiliki langkah sebagai berikut :

1. Pada main() ditambahkan fungsi glutIdleFunc(display) sebelum fungsi glutDisplayFunc(display)
2. Pada awal fungsi userdraw() ditambahkan perintah untuk menambah nilai tick secara terus menerus dengan perintah tick++
3. Tambahkan nilai tick ini pada nilai variabel dasar pembuatan grafik.

Berikut code program Animasi 2-D menggunakan koordinat polar :

```
void userdraw()
{
    static int tick=0;
    Point2D_t shape[360];
    double srاد,r;
    for(int s=0;s<360;s++)
    {
        srاد=(s+tick)*3.14/180;
        r=sin(5*srاد);
        shape[s].x=(float) (r*cos(srاد));
        shape[s].y=(float) (r*sin(srاد));
    }
    Polygon(shape,360);
    tick++;
}
```

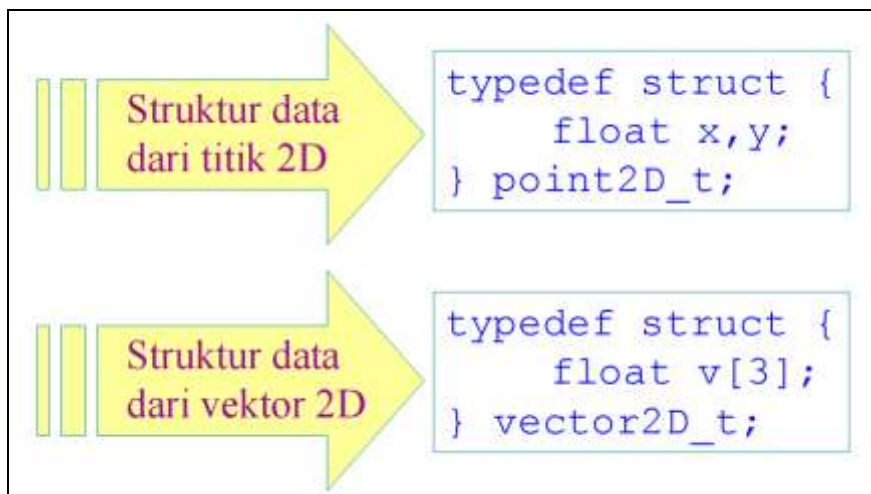
## BAB IV

### TRANSFORMASI 2 DIMENSI

#### 4.1 Materi

1. Struktur titik dan vektor
2. Perubahan struktur titik ke vektor
3. Perubahan struktur vektor ke titik
4. Tranformasi 2-D
5. Perkalian Matriks
6. Komposisi Transformasi

#### 4.2 Struktur Titik dan Vektor



### 4.3 Perubahan struktur titik ke vektor

Fungsi ini digunakan untuk memindahkan tipe data titik menjadi tipe data vektor. Hal ini sangat berguna untuk operasional matrik yang digunakan dalam melakukan transformasi dan pengolahan matrik pada grafika komputer.

```
vector2D_t point2vector(point2D_t pnt)
{
    vector2D_t vec;
    vec.v[1]=pnt.x;
    vec.v[2]=pnt.y;
    vec.v[3]=1.;
}
```

### 4.4 Perubahan struktur vektor ke titik

Fungsi ini digunakan untuk memindahkan tipe data vektor menjadi tipe data titik. Hal ini sangat berguna untuk penyajian grafis setelah proses pengolahan matrik yang dikenakan pada obyek 2-D.

```
point2D_t vector2point(vector2D_t vec)
{
    point2D_t pnt;
    pnt.x=vec.v[1];
    pnt.y=vec.v[2];
}
```

### 4.5 Transformasi 2-D

Matrik transformasi 2-D adalah matrik yang membuat sebuah obyek mengalami perubahan baik berupa perubahan posisi, maupun perubahan ukuran. Matrik 2-D dinyatakan dalam ukuran 3x3, dimana kolom ke-3 digunakan untuk menyediakan tempat untuk proses translasi.

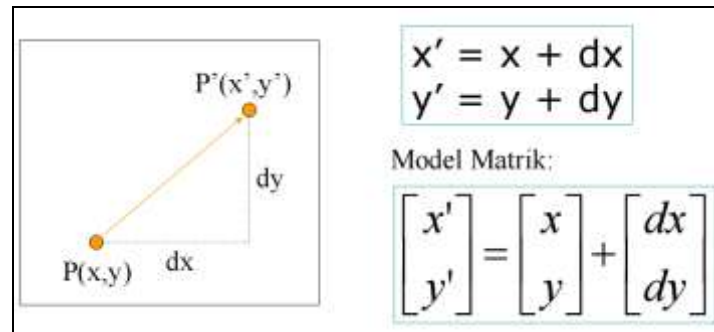


Berikut bentuk matrik 3 x 3 :

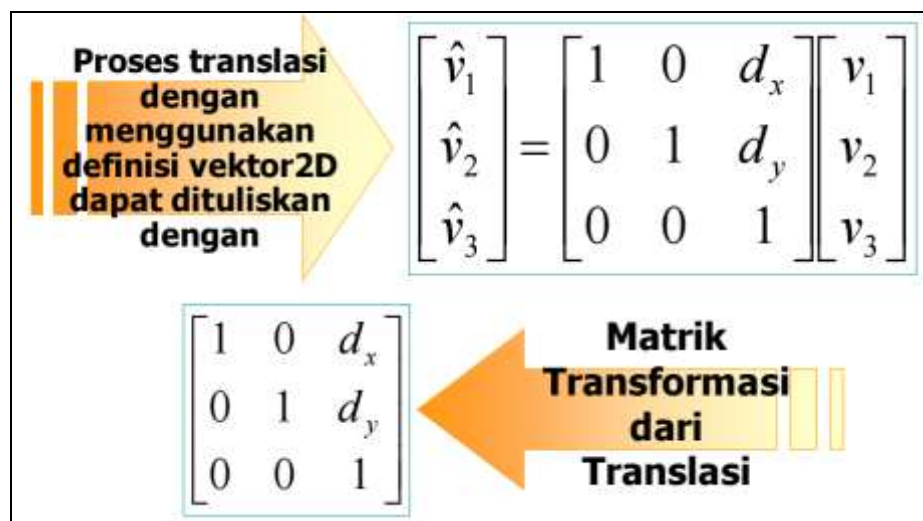
$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

#### 4.5.1 Translasi

Translasi adalah perpindahan obyek dari titik P ke titik P' secara linier.



Proses translasi dengan menggunakan definisi vektor 2-D



Implementasi Matrik Transformasi untuk translasi



```
matrix2D_t translationMTX(float dx,float dy)
{
    matrix2D_t trans=createIdentity();
    trans.m[0][2]=dx;
    trans.m[1][2]=dy;
    return trans;
}
```

Fungsi untuk membuat matrik identitas

Matrik Identitas

Adalah matrik yang nilai diagonal utamanya adalah 1 dan yang lain bernilai nol.

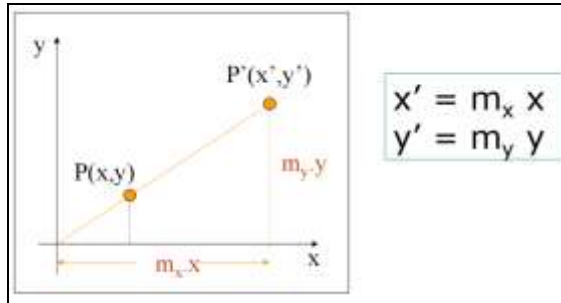
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Code programnya adalah sebagai berikut :

```
matrix2D_t createIdentity(void) {
    matrix2D_t u;
    int i,j;
    for (i=0;i<3;i++) {
        for(j=0;j<3;j++) u.m[i][j]=0.;
        u.m[i][i]=1.;
    }
    return u;
}
```

### 4.5.2 Scalling

Scaling  $m$  adalah perpindahan obyek dari titik  $P$  ke titik  $P'$ , dimana jarak titik  $P'$  adalah  $m$  kali titik  $p$ .



Proses scalling dengan menggunakan definisi vektor 2-D

**Proses scaling dengan menggunakan definisi vektor2D dapat dituliskan dengan**

$$\begin{bmatrix} \hat{v}_1 \\ \hat{v}_2 \\ \hat{v}_3 \end{bmatrix} = \begin{bmatrix} m_x & 0 & 0 \\ 0 & m_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

**Matrik Transformasi dari Scaling**

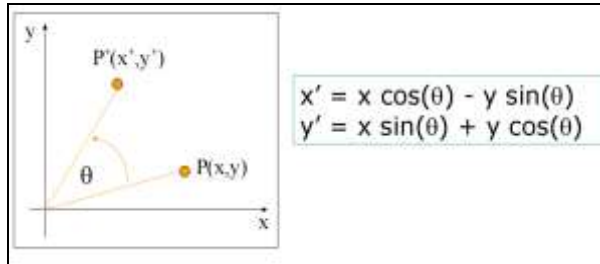
$$\begin{bmatrix} m_x & 0 & 0 \\ 0 & m_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Code programnya adalah sebagai berikut :

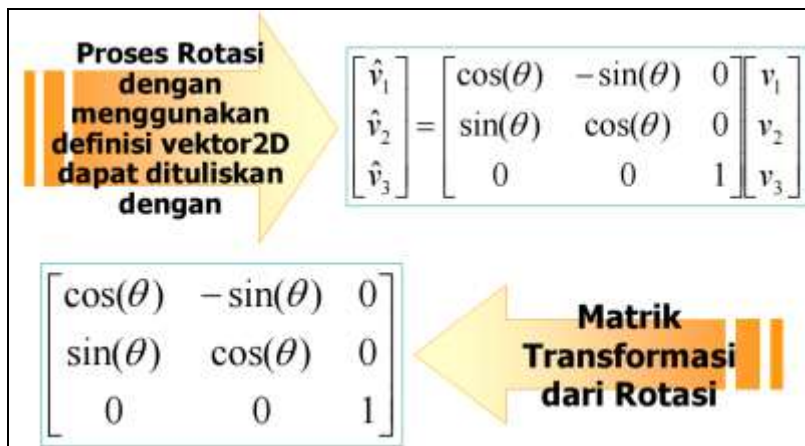
```
matrix2D_t scalingMTX(float mx,float my)
{
    matrix2D_t scale=createIdentity();
    scale.m[0][0]=mx;
    scale.m[1][1]=my;
    return scale;
}
```

### 4.5.3 Rotasi

Adalah perpindahan obyek dari titik P ke P' yang berupa pemindahan berputar sebesar sudut  $\theta$ .



Proses rotasi dengan menggunakan definisi vektor 2-D



Code programnya adalah sebagai berikut :

```
matrix2D_t rotationMTX(float theta)
{
    matrix2D_t rotate=createIdentity();
    float cs=cos(theta);
    float sn=sin(theta);
    rotate.m[0][0]=cs; rotate.m[0][1]=-sn;
    rotate.m[1][0]=sn; rotate.m[1][1]=cs;
    return rotate;
}
```

## 4.6 Perkalian Matrik

Catatan :

1. Perkalian matrik dengan matrik menghasilkan matrik
2. Perkalian matrik dengan vektor menghasilkan vektor
3. Perkalian matrik ini digunakan untuk operasional transformasi dari obyek 2D dan untuk komposisi (menggabungkan) transformasi.

Perumusan perkalian matrik dengan matrik, dimana untuk nilai i dan j bernilai 0 s/d 2

$$c_{ij} = \sum_{k=0}^2 a_{ik} b_{kj}$$

Code program perkalian matrik

```
matrix2D_t operator * (matrix2D_t a, matrix2D_t b)
{
    matrix2D_t c;//c=a*b
    int i,j,k;
    for (i=0;i<3;i++) for (j=0;j<3;j++) {
        c.m[i][j]=0;
        for (k=0;k<3;k++)
            c.m[i][j]+=a.m[i][k]*b.m[k][j];
    }
    return c;
}
```

Perumusan perkalian matrik dengan vektor, dimana untuk nilai i dan j bernilai 0 s/d 2

$$c_i = \sum_{k=0}^2 a_{ik} b_k$$

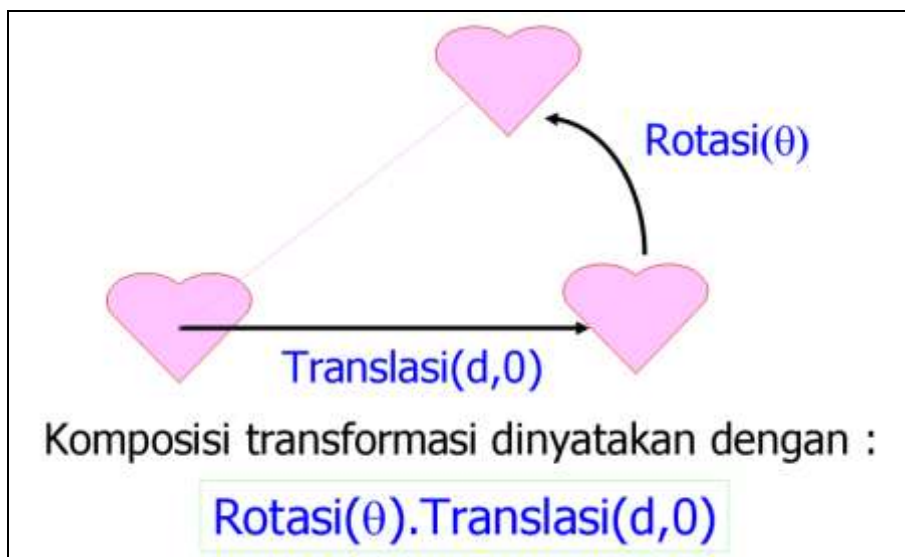
Code program perkalian matrik dengan vektor

```
vector2D_t operator * (matrix2D_t a, vector2D_t b)
{
    vector2D_t c;//c=a*b
    int i,j;
    for (i=0;i<3;i++) {
        c.v[i]=0;
        for (j=0;j<3;j++)
            c.v[i]+=a.m[i][j]*b.v[j];
    }
    return c;
}
```

#### 4.7 Komposisi Transformasi

Adalah penggabungan beberapa transformasi sehingga didapat hasil transformasi yang lebih kompleks. Komposisi transformasi dapat dilakukan dengan mengalikan matrik-matrik transformasi.

Contoh komposisi transformasi :



Rotasi( $\theta$ )



Translasi( $d,0$ )

Komposisi transformasi dinyatakan dengan :

$\text{Translasi}(d,0). \text{Rotasi}(\theta)$

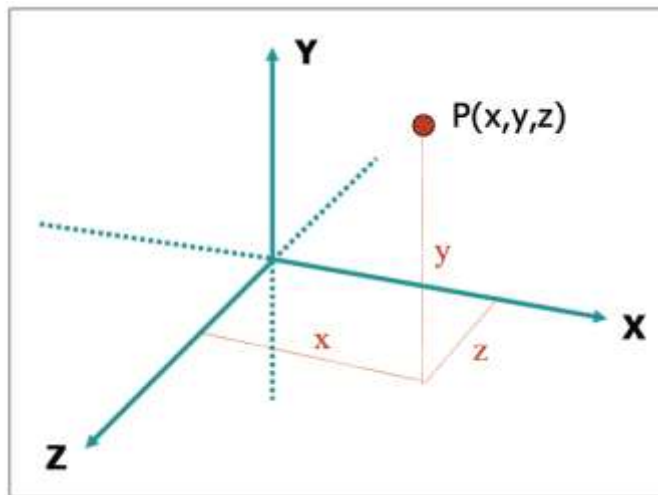
## BAB V

### GRAFIK 3 DIMENSI

#### 5.1 Materi

1. Sistem Koordinat 3 Dimensi
2. Definisi obyek 3 Dimensi
3. Cara menggambar Obyek 3 Dimensi
4. Konversi vektor 3D menjadi titik 2D
5. Konversi titik 2D menjadi vektor 3D
6. Visible dan Invisible

Sistem Koordinat 3 Dimensi

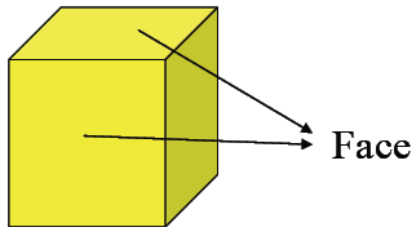


Titik 3 Dimensi dinyatakan dengan  $P(x, y, z)$  dan dituliskan dalam bentuk :

```
typedef struct {  
    float x,y,z;  
} point3D_t
```

## 5.2 Definisi Obyek 3 Dimensi

Adalah sekumpulan titik 3-D (x,y,z) yang membentuk luasan – luasan (face) yang digabungkan menjadi satu kesatuan. Face adalah gabungan titik – titik yang membentuk luasan tertentu atau sering dinamakan dengan sisi.



Obyek kubus mempunyai  
8 titik dan 6 face

Implementasi definisi dari struktur face :

```
typedef struct {
    int NumberofVertices;
    short int pnt[32];
} face_t;
```

**NumberofVertices** menyatakan jumlah titik pada sebuah face.

**Pnt[32]** menyatakan nomor – nomor titik yang digunakan untuk membentuk face, dengan maksimum 32 titik.

Implementasi definisi dari Struktur Obyek 3-D :

```
typedef struct {
    int NumberofVertices;
    point3D_t pnt[100];
    int NumberofFaces;
    face_t fc[32];
} object3D_t;
```



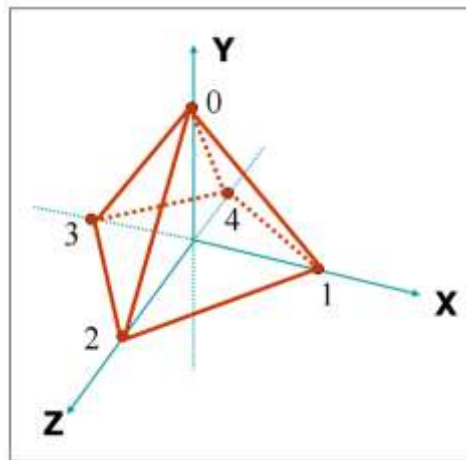
**NumberOfVertices** menyatakan jumlah titik yang membentuk obyek.

**Pnt[100]** menyatakan titik-titik yang membentuk face, dengan maksimum 100 titik.

**NumberOfFaces** menyatakan jumlah face yang membentuk obyek.

**Fc[32]** menyatakan face-face yang membentuk obyek.

Contoh pernyataan obyek limas segi empat :



Titik-titik yang membentuk obyek:  
 Titik 0 → (0,150,0)  
 Titik 1 → (100,0,0)  
 Titik 2 → (0,0,100)  
 Titik 3 → (-100,0,0)  
 Titik 4 → (0,0,-100)

Face yang membentuk obyek :  
 Face 0 → 0,2,1  
 Face 1 → 0,3,2  
 Face 2 → 0,4,3  
 Face 3 → 0,1,4  
 Face 4 → 1,2,3,4

Implementasi Pernyataan Obyek 3 Dimensi

```
object3D_t prisma={5,
  {{0,100,0},{100,0,0},{0,0,100},
  {-100,0,0},{0,0,-100}},
  5,
  {{3,{0,1,2}},{3,{0,2,3}},
  {3,{0,3,4}},{3,{0,4,1}},
  {4,{1,4,3,2}}}};
```

Ditulis pada userdraw sebagai nilai dari obyek 3-D yang akan digambarkan.

### 5.3 Cara Menggambar Obyek 3 Dimensi

1. Obyek 3D terdiri dari titik – titik dan face – face
2. Penggambaran dilakukan pada setiap face menggunakan polygon
3. Polygon terdiri dari titik – titik yang terdapat pada sebuah face
4. Titik – titik dinyatakan dalam struktur 3D, sedangkan layar komputer dalam struktur 2D. Sehingga diperlukan konversi dari titik 3D ke 2D.

### 5.4 Koversi Vektor 3D menjadi 2D

1. Untuk menggambar obyek 3D, untuk setiap face perlu dilakukan pengubahan titik 3D menjadi vektor 3D.
2. Setelah proses pengolahan vektor, maka bentuk vektor 3D menjadi 2D.
3. Sumbu Z adalah sumbu yang searah dengan garis mata, sehingga perlu transformasi untuk menampilkan sumbu ini. Untuk hal ini perlu dilakukan rotasi sumbu.
4. Dalam konversi arah Z tidak diambil.

Vektor 3 D :

$$vec = \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

```
typedef struct {
    float v[4];
} vector3D_t;
```

Implementasi Konversi vektor 3D menjadi titik 2D

```
point2D_t Vector2Point2D(vector3D_t vec)
{
    point2D_t pnt;
    pnt.x=vec.v[0];
    pnt.y=vec.v[1];
    return pnt;
}
```

Implementasi konversi titik 3D menjadi vektor 3D

```
vector3D_t Point2Vector(point3D_t pnt)
{
    vector3D_t vec;
    vec.v[0]=pnt.x;
    vec.v[1]=pnt.y;
    vec.v[2]=pnt.z;
    vec.v[3]=1.;
    return vec;
}
```

Kode Program Menggambar Obyek 3D

```
mat=tilting;
for(i=0;i<prisma.NumberofVertices;i++)
{
    vec[i]=Point2Vector(prisma.pnt[i]);
    vec[i]=mat*vec[i];
}
for(i=0;i<prisma.NumberofFaces;i++)
{
    for(j=0;j<prisma.fc[i].NumberOfVertices;j++)
        vecbuff[j]=vec[prisma.fc[i].pnt[j]];
    for(j=0;j<prisma.fc[i].NumberOfVertices;j++)
        titik2D[j]=Vector2Point2D(vec[prisma.fc[i].pnt[j]]);
    drawPolygon(titik2D,prisma.fc[i].NumberOfVertices);
}
```

```

mat=tilting;
for(i=0;i<prisma.NumberofVertices;i++)
{
    vec[i]=Point2Vector(prisma.pnt[i]);
    vec[i]=mat*vec[i];
}

```

Deklarasi **mat** sebagai matrik tilting menyatakan bahwa obyek yang digambar mengikuti pergerakan sumbu koordinat ( *tilting* ).

Setiap titik diubah menjadi vektor dengan memperhatikan matrik transformasi yang dinyatakan dalam **mat**.

### Implementasi Tilting

Tilting adalah matrik rotasi dari sumbu koordinat dan semua obyek yang digambar didalamnya.

```

float theta=0.5;
matrix3D_t tilting=rotationXMTX(theta)*rotationYMTX(-theta);

```

Dalam deklarasi ini, matrik tilting adalah rotasi terhadap sumbu y sebesar -0.5 rad dan rotasi terhadap sumbu x sebesar 0.5 rad.

```

for(i=0;i<prisma.NumberofFaces;i++)
{
    for(j=0;j<prisma.fc[i].NumberOfVertices;j++)
        vecbuff[j]=vec[prisma.fc[i].pnt[j]];
    for(j=0;j<prisma.fc[i].NumberOfVertices;j++)
        titik2D[j]=Vector2Point2D(vec[prisma.fc[i].pnt[j]]);
    drawPolygon(titik2D,prisma.fc[i].NumberOfVertices);
}

```

Untuk setiap face pada obyek 3D :

1. Ambil vektor dari setiap titik pada face tersebut
2. Konversikan setiap vektor 3D menjadi titik 2D
3. Dari hasil konversi digambarkan polygon

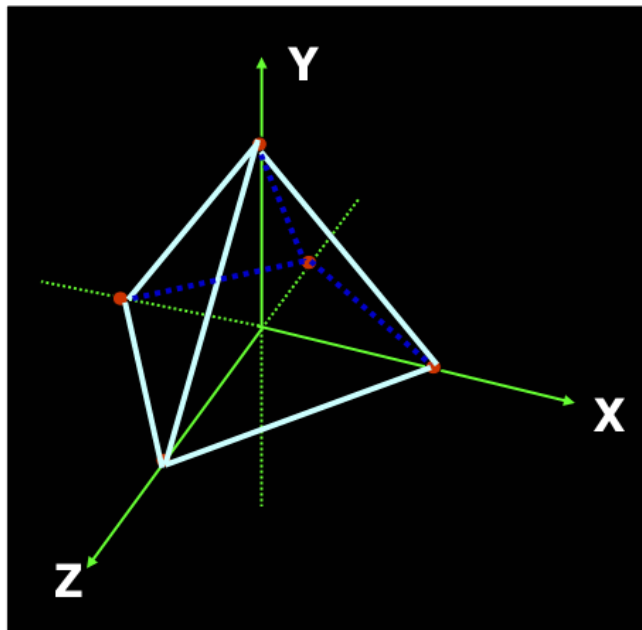
## 5.5 Visible dan Invisible

1. Visible dan Invisible menyatakan apakah suatu face terlihat (visible) atau tidak terlihat (invisible).
2. Pada obyek 3D tidak semua face terlihat, karena terdapat face-face yang berada di bagian belakang dan terhalang oleh face yang lain.
3. Untuk menyatakan face visible dan invisible digunakan vektor normal pada face tersebut.
4. Suatu face visible bila arah z pada vektor normal positif, dan invisible bila arah z pada vektor normalnya negatif.

### Implementasi Visible dan Invisible

1. Untuk mengimplementasikan face visible dan invisible maka dilakukan penggambaran dua kali.
2. Pertama digambar dulu face-face yang invisible ( $\text{NormalVektor.v}[2] < 0$ )
3. Kedua digambar face-face yang visible ( $\text{NormalVektor.v}[2] > 0$ )

### Contoh visible dan Invisible



```

setColor(0,0,1);
for(i=0;i<prisma.NumberofFaces;i++)
{
    for(j=0;j<prisma.fc[i].NumberOfVertices;j++)
        vecbuff[j]=vec[prisma.fc[i].pnt[j]];
    NormalVector=(vecbuff[1]-vecbuff[0])^(vecbuff[2]-vecbuff[0]);
    normalzi=NormalVector.v[2];
    if(normalzi<0.)
    {
        for(j=0;j<prisma.fc[i].NumberOfVertices;j++)
            titik2D[j]=Vector2Point2D(vec[prisma.fc[i].pnt[j]]);
        drawPolygon(titik2D,prisma.fc[i].NumberOfVertices);
    }
}

```

Menghitung vektor normal dari setiap face (*NormalVector*)

Menghitung arah z dari vektor normal (*normalzi*)

Menentukan apakah face invisible (*normalize<0*)

Bagian invisible diberi warna biru (0,0,1)

```

setColor(0,1,1);
for(i=0;i<prisma.NumberofFaces;i++)
{
    for(j=0;j<prisma.fc[i].NumberOfVertices;j++)
        vecbuff[j]=vec[prisma.fc[i].pnt[j]];
    NormalVector=(vecbuff[1]-vecbuff[0])^(vecbuff[2]-vecbuff[0]);
    normalzi=NormalVector.v[2];
    if(normalzi>0.)
    {
        for(j=0;j<prisma.fc[i].NumberOfVertices;j++)
            titik2D[j]=Vector2Point2D(vec[prisma.fc[i].pnt[j]]);
        drawPolygon(titik2D,prisma.fc[i].NumberOfVertices);
    }
}

```

Menghitung vektor normal dari setiap face (*NormalVector*)

Menghitung arah z dari vektor normal (*normalzi*)

Menentukan apakah face visible (*normalize>0*)

Bagian visible diberi warna cyan (0,1,1)

## 5.6 Contoh contoh grafik

### MENGGAMBAR PRISMA

```
#include <glut.h>
#include <math.h>

struct point
{
    float x,y,z;
};

struct face
{
    int jumtitikons;
    int indextitik[40];
};

struct o3d
{
    int jumtitik;
    point titik[100];
    int jumsisi;
    face sisi[100];
};

void DrawLineLoop(o3d obj)
{
    int i,j;

    for(i=0;i<obj.jumsisi;i++)
    {
        glBegin(GL_LINE_LOOP);
        glColor3f(1.0f,0.0f,0.0f);
        for(j=0;j<obj.sisi[i].jumtitikons;j++)
        {
            glVertex3f(obj.titik[obj.sisi[i].indextitik[j]].x,
                        obj.titik[obj.sisi[i].indextitik[j]].y,
                        obj.titik[obj.sisi[i].indextitik[j]].z);
        }
        glEnd();
    }
}

void Prisma()
{
    o3d
    prisma6={12,{5,10,30},{15,20,30},{25,20,30},{35,10,30},{25,0,30},{15,0,30},{5,10,10},{15,20,10},{25,20,10},{35,10,10},{25,0,10},{15,0,10}},8,{6,{0,1,2,3,4,5}},
```

```

        {6,{6,7,8,9,10,11}}, {4,{0,6,7,1}}, {4,{1,7,8,2}}, {4,{2,8,9,3}}, {4,{3,9,10,4}}, {4,{4,10,11,5}}, {4,{5,11,6,0}}
    };

    DrawLineLoop(prisma6);
}

void UserDraw()
{
    glClearColor(1.0f,1.0f,1.0f,0.0f);
    glClear(GL_COLOR_BUFFER_BIT);
    glRotatef(30.1f,1.0f,1.0f,1.0f);
    Prisma();
    glutSwapBuffers();
}

void main(int argc,char **argv)
{
    glutInit(&argc,argv);
    glutInitWindowPosition(20,20);
    glutInitWindowSize(640,640);
    glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB);
    glutCreateWindow("Prisma Segi 6 Tiga Dimensi");
    glOrtho(-30.0f,50.0f,-30.0f,50.0f,-30.0f,50.0f);
    //glutIdleFunc(UserDraw);
    glutDisplayFunc(UserDraw);
    glutMainLoop();
}

```

## PROGRAM BUAT LIMAS

```

#include <glut.h>
#include <math.h>

struct point
{
    float x,y,z;
};
struct face
{
    int jumtitikons;
    int indextitik[40];
};

struct o3d
{
    int jumtitik;
    point titik[100];
}

```



```

        int jumsisi;
        face sisi[100];
};

void DrawLineLoop(o3d obj)
{
    int i,j;
    for(i=0;i<obj.jumsisi;i++)
    {
        glBegin(GL_LINE_LOOP);
        glColor3f(1.0f,0.0f,0.0f);
        for(j=0;j<obj.sisi[i].jumtitikons;j++)
        {
            glVertex3f(obj.titik[obj.sisi[i].indexitik[j]].x,
                obj.titik[obj.sisi[i].indexitik[j]].y,
                obj.titik[obj.sisi[i].indexitik[j]].z);
        }
        glEnd();
    }
}

void Limas4()
{
    o3d limas4a={5,{{10,15,20},{20,15,20},{20,15,10},{10,10,25},{15,25,15}},5,{{3,{0,1,4}},
        {3,{3,2,4}}, {3,{0,4,3}}, {3,{1,2,4}}, {4,{0,1,2,3}}}};
    DrawLineLoop(limas4a);
}

void UserDraw()
{
    glClearColor(1.0f,1.0f,1.0f,0.0f);
    glClear(GL_COLOR_BUFFER_BIT);
    Limas4();
    glutSwapBuffers();
}

void main(int argc,char **argv)
{
    glutInit(&argc,argv);
    glutInitWindowPosition(20,20);
    glutInitWindowSize(640,640);
    glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB);
    glutCreateWindow("3 Dimensi - Klp 3");
    glOrtho(-30.0f,50.0f,-30.0f,50.0f,-30.0f,50.0f);
    glutIdleFunc(UserDraw);
    glutDisplayFunc(UserDraw);
    glutMainLoop();
}

```

## MENGGAMBAR OBJEK KOTAK DENGAN WARNA GRADASI

```
#include <glut.h>

typedef struct {
    float x,y;
} point2D_t;

typedef struct {
    float r,g,b;
} color_t;

typedef struct
{
    float r, g, b;
}color;

void setColor(color col)
{
    glColor3f(col.r, col.g, col.b);
}

void setColor(float r, float g, float b)
{
    glColor3f(r, g, b);
}

void drawPolygon(point2D_t pnt[], int n)
{
    int i;
    glBegin(GL_LINE_LOOP);
    for (i=0; i<n; i++)
        glVertex2f(pnt[i].x,pnt[i].y);
    glEnd();
}

void fillPolygon(point2D_t pnt[], int n, color_t color)
{
    int i;
    //setColor(color);
    glBegin(GL_POLYGON);
    for (i=0; i<n; i++)
    {
        glVertex2f(pnt[i].x, pnt[i].y);
    }
}
```

```

    }
    glEnd();
}

void gradatePolygon(point2D_t line[], int n, color col[])
{
    glBegin(GL_POLYGON);
    for (int i=0; i<n;i++)
    {
        setColor(col[i]);
        glVertex2f(line[i].x,line[i].y);
    }
    glEnd();
}

void userdraw()
{
    static int tick=0;
    point2D_t kotak1[4]={{100,100},{300,100},{300,200},{100,200}}; // input penggambaran objek
    color AREA[2]={{0.5,0,1},{0.5,0.5,0.2}}; //Input 2 warna untuk pewarnaan gradasi
    gradatePolygon(kotak1,4,AREA); //Pemanggilan objek yang akan ditampilkan
    setColor(1,1,1); //Pemberian warna garis tepi
    drawPolygon(kotak1,4); //Pemanggilan objek untuk garis tepi
}

void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    userdraw();
    glutSwapBuffers();
}

int main(int argc, char **argv)
{
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB);
    glutInitWindowPosition(100,100); //meletakkan pada posisi (100,100) di layar komputer
    glutInitWindowSize(640,480); //membuat ukuran window (640,480)
    glutCreateWindow("Tampilan Windows Open GL"); //pemberian judul pada window
    glClearColor(0.0,0.0,0.0,0.0); //pemberian warna pada window dengan format RGB
    gluOrtho2D(-50.,840.,-140.,840.);
    glutIdleFunc(display);
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}

```

## MEMBUAT EFEK BINTANG

```
#include <math.h>
#include <glut.h>

typedef struct
{
    float x, y;
}point2D_t;

typedef struct
{
    float r, g, b;
}color;

typedef struct
{
    float r, g, b;
}color_t;

void setColor(color col)
{
    glColor3f(col.r, col.g, col.b);
}

void setColor(float r, float g, float b)
{
    glColor3f(r, g, b);
}

void fillPolygon(point2D_t line[], int n, color col)
{
    glBegin(GL_POLYGON);
    setColor(col);
    for (int i=0; i<n; i++)
        glVertex2f(line[i].x, line[i].y);
    glEnd();
}

void gradatePolygon(point2D_t line[], int n, color col[])
{
    glBegin(GL_POLYGON);
    for (int i=0; i<n; i++)
    {
        setColor(col[i]);
        glVertex2f(line[i].x, line[i].y);
    }
}
```

```

        }
        glEnd();
    }

void drawPolygon(point2D_t pnt[], int n) {
    int i;
    glBegin(GL_POLYGON);
    for (i=0; i<n; i++)
        glVertex2f(pnt[i].x,pnt[i].y);
    glEnd();
}

static void createStar(point2D_t p[],float a,point2D_t p1)
{
    float teta=36,sudut,r=10;

    for(int i=0;i<10;i++)
    {
        sudut=(float)i*teta/57.3;
        if (r==2)r=10;else r=2;
        p[i].x=p1.x+r*cos(sudut+a);
        p[i].y=p1.y+r*sin(sudut+a);
    }
}

void userdraw()
{
    point2D_t batas[4] = {{36,20},{988,20},{988,583},{36,583}};
    color wlangit[4]={{0.3,0.0,0.3},{0.3,0.0,0.3},{0.9,1,0.5},{0.9,0.8,0.5}};
    gradatePolygon(batas,4,wlangit);
    static float a=0;
    color warna={1,1,0};
    point2D_t c10[10],p10={530,121};
    createStar(c10,a,p10);
    point2D_t c1[10],p1={500,321};
    createStar(c1,a,p1);
    point2D_t c2[10],p2={433,247};
    createStar(c2,a,p2);
    point2D_t c3[10],p3={286,125};
    createStar(c3,a,p3);
    point2D_t c4[10],p4={553,150};
    createStar(c4,a,p4);
    point2D_t c5[10],p5={232,350};
    createStar(c5,a,p5);
    point2D_t c6[10],p6={643,123};
    createStar(c6,a,p6);
    point2D_t c7[10],p7={200,250};

```

```
        createStar(c7,a,p7);
        point2D_t c8[10],p8={433,247};
        createStar(c8,a,p8);
        point2D_t c9[10],p9={334,350};
        createStar(c9,a,p9);

        fillPolygon(c1,10,warna);
        fillPolygon(c2,10,warna);
        fillPolygon(c3,10,warna);
        fillPolygon(c4,10,warna);
        fillPolygon(c5,10,warna);
        fillPolygon(c6,10,warna);
        fillPolygon(c7,10,warna);
        fillPolygon(c8,10,warna);
        fillPolygon(c9,10,warna);
        fillPolygon(c10,10,warna);
        a+=0.005; // Mengatur efek bintang berputar
    }

void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    userdraw();
    glutSwapBuffers();
}

int main (int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowPosition (0,0);
    glutInitWindowSize(1020,760);
    glutCreateWindow("Fighting");
    glClearColor(1,1,1,0);
    gluOrtho2D(0,1024,768,0);
    glutIdleFunc(display);
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```