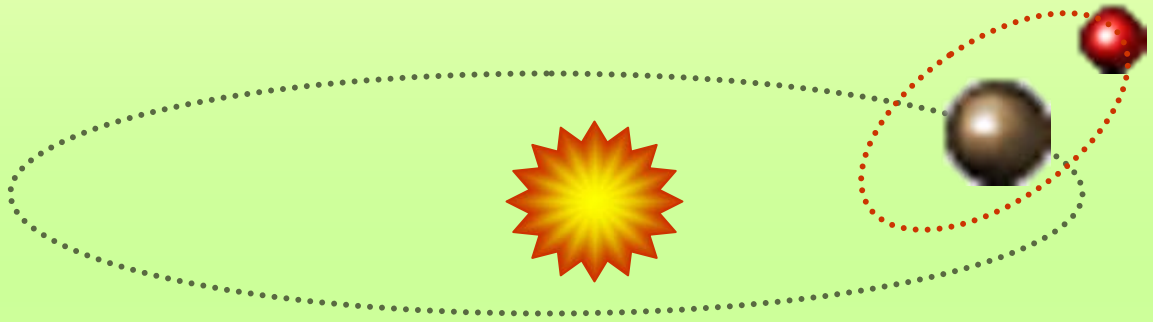
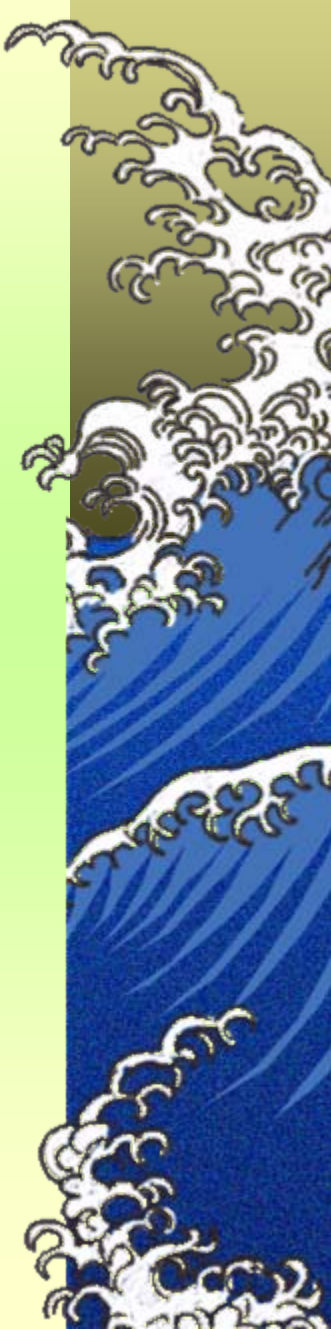
Grafik 3 Dimensi



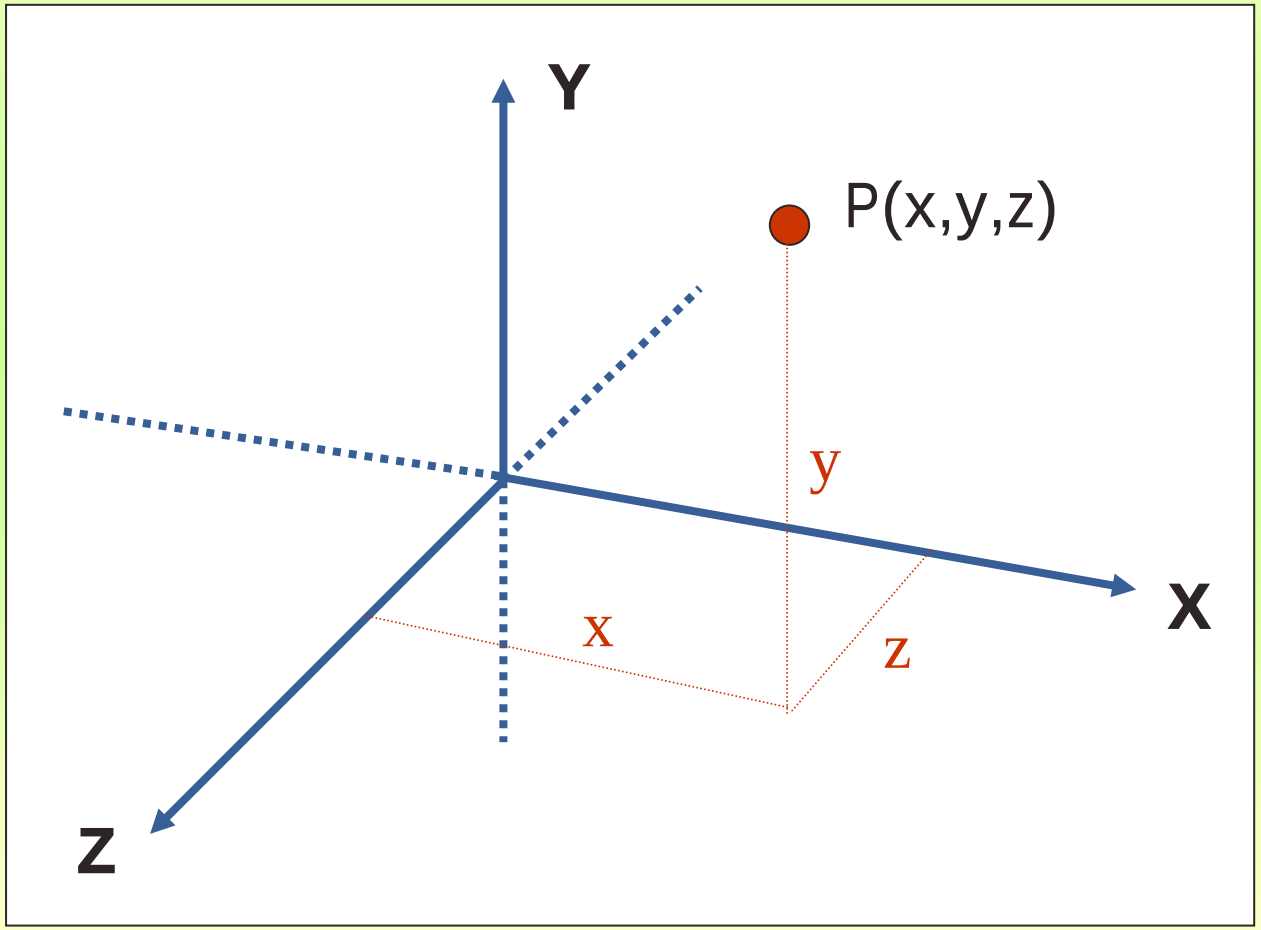
Achmad Basuki – Nana R
Politeknik Elektronika Negeri Surabaya
Surabaya 2009

Materi

- Sistem Koordinat 3D
- Definisi Obyek 3D
- Cara Menggambar Obyek 3D
- Konversi Vektor 3D menjadi Titik 2D
- Konversi Titik 2D menjadi Vektor 3D
- Visible dan Invisible



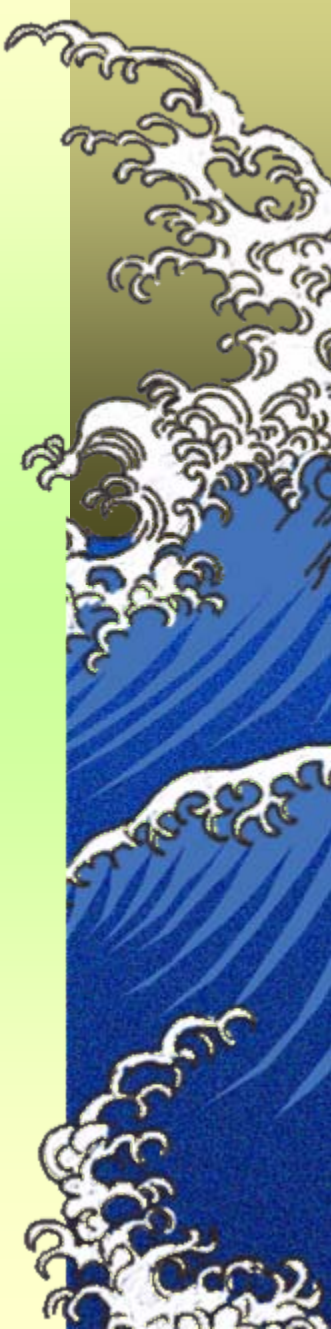
Sistem Koordinat 3 Dimensi



Titik 3D

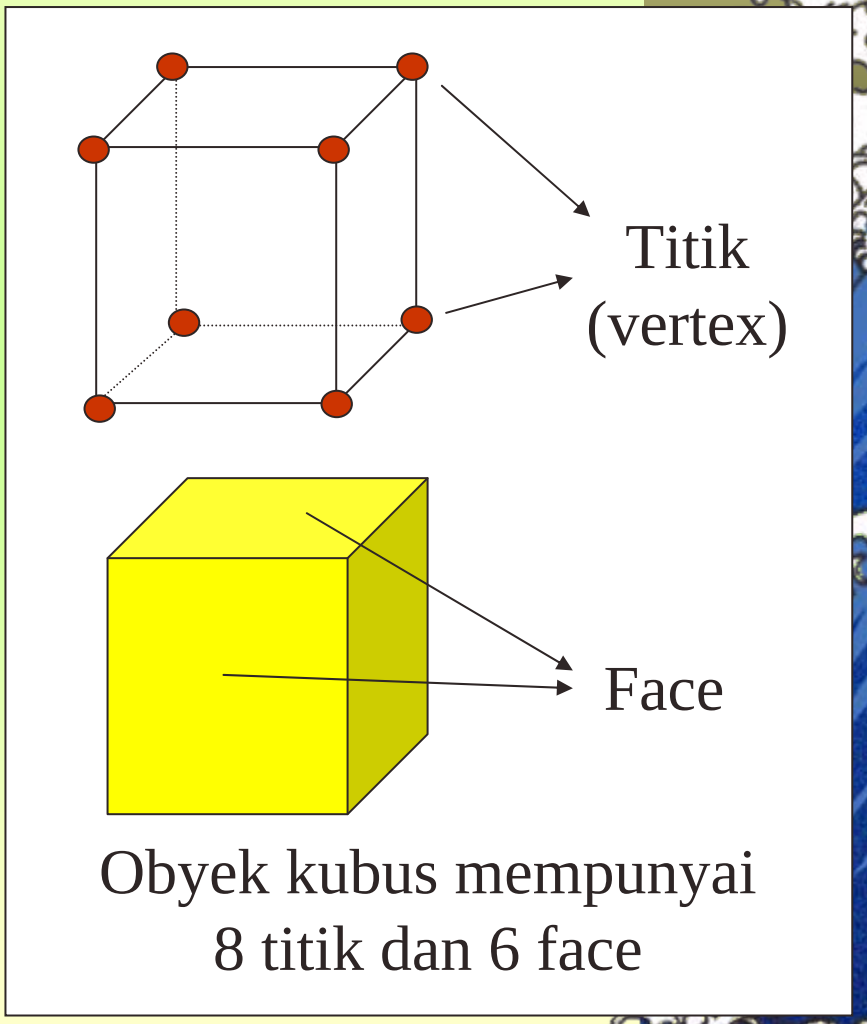
Titik 3D dinyatakan dengan :
P(x,y,z)

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



Definisi Obyek 3 Dimensi

- **Obyek 3-D adalah sekumpulan titik-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.**

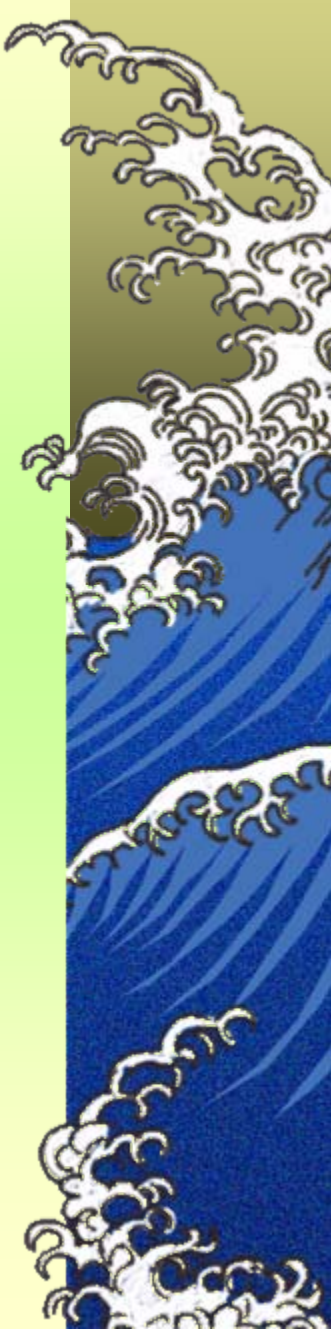


Implementasi Definisi Dari Struktur Faces

```
typedef struct {
//jumlah titik pada face
        int NumberofVertices;
//nomor-nomor titik pada face
        int pnt[100];
} 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 3D

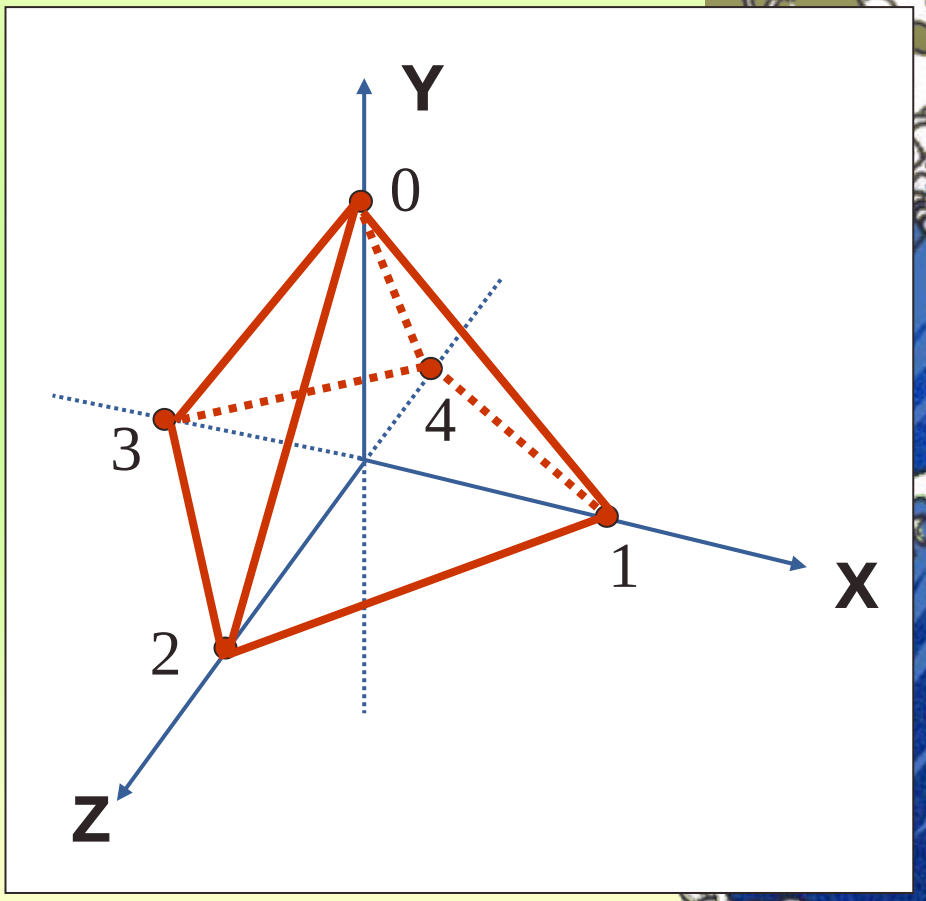
```
typedef struct {
// jumlah titik pada obyek gambar
int NumberofVertices;
// posisi titik pada obyek gambar
point3D_t pnt[640];
// Jumlah face
int NumberofFaces;
// Face-face yang ada pada obyek
face_t fc[480];
} object3D_t;
```

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 SegiEmpat

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)

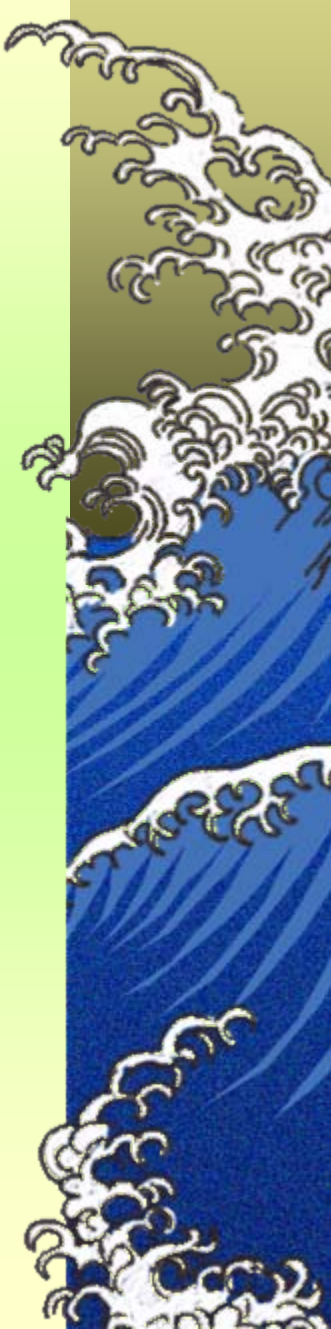
Face yang membentuk obyek :
Face 0 → 0,2,1
Face 1 → 0,3,2
Face 2 → 0,4,3
Face 3 → 0,1,4
Face 4 → 1,2,3,4



Implementasi Pernyataan Obyek 3 Dimensi

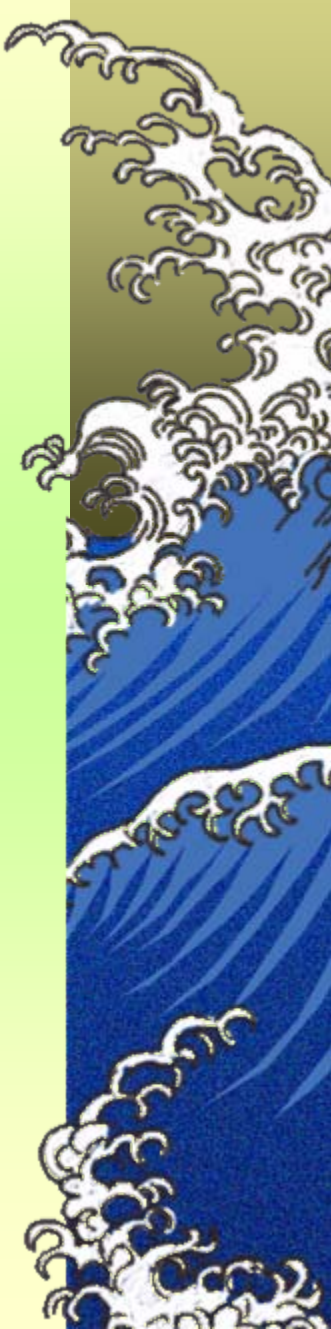
```
object3D_t prisma={5,
     {{0,100,0},{100,0,0},{0,0,100},
     {-100,0,0},{0,0,-100}},
     5,
     {{3,{0,1,2}},{3,{0,2,3}},
     {3,{0,3,4}},{3,{0,4,1}},
     {4,{1,4,3,2}}}};
```

Pernyataan ini ditulis pada fungsi userdraw sebagai nilai dari obyek 3D yang akan digambarkan



Cara Menggambar Obyek 3 Dimensi

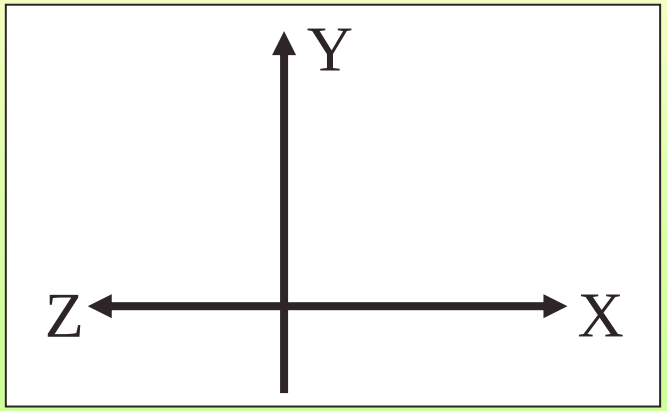
- Obyek 3 Dimensi terdiri dari titik-titik dan face-face.
- Penggambaran dilakukan pada setiap face menggunakan polygon.
- Polygon dibentuk dari titik-titik yang terdapat pada sebuah face.
- Titik-titik dinyatakan dalam struktur 3D, sedangkan layar komputer dalam struktur 2D. Sehingga diperlukan konversi dari 3D menjadi 2D.



Konversi Vektor 3D menjadi 2D

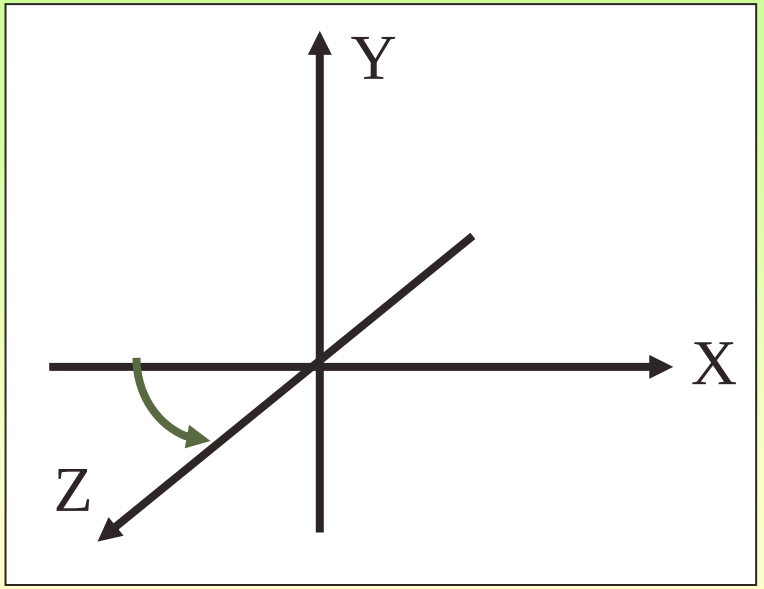
- Untuk menggambar obyek 3D, untuk setiap face perlu dilakukan pengubahan titik 3D menjadi vektor 3D, agar mudah dilakukan transformasi.
- Setelah proses pengolahan vektor, maka bentuk vektor 3D menjadi 2D.
- Sumbu Z adalah sumbu yang searah dengan garis mata, sehingga perlu transformasi untuk menampilkan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

Konversi Vektor 3D menjadi 2D



Konversi untuk menggambar obyek

Transformasi Sumbu (*Tilting*)



Vektor 3D

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

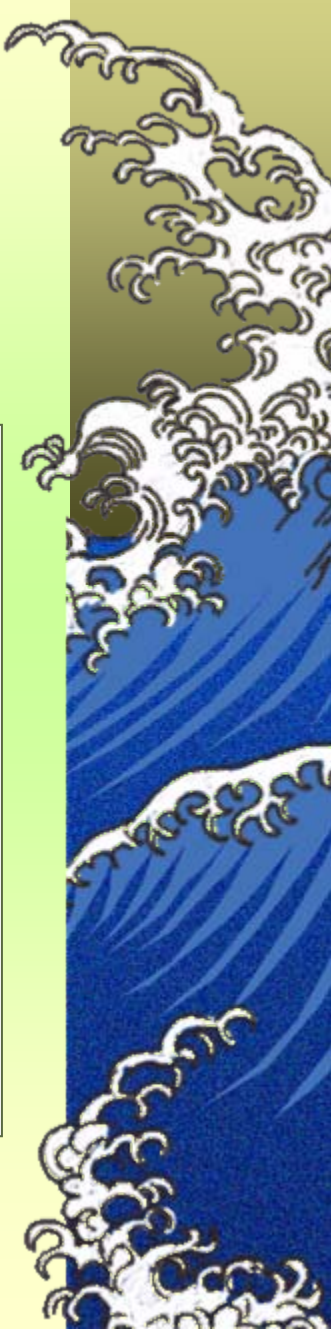
```
typedef struct {
    float v[4];
} vector3D_t;
```

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;
}
```



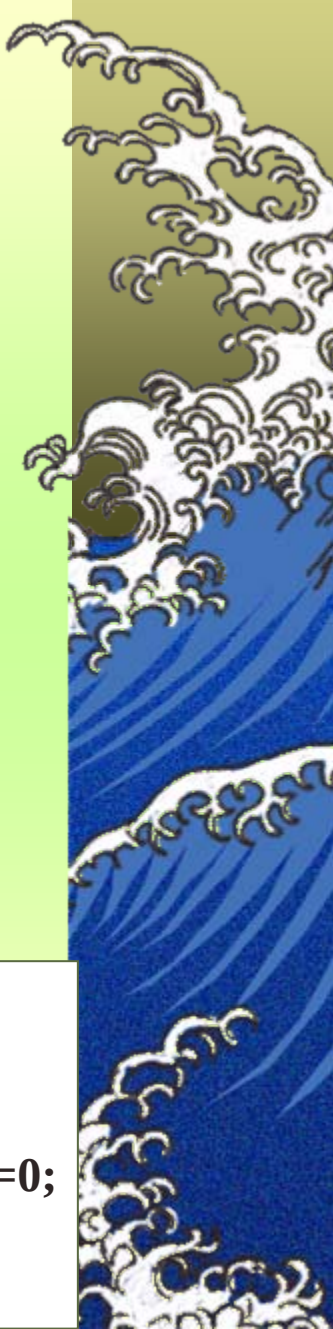
Transformasi 3 Dimensi

Transformasi 3 dimensi merupakan proses pengolahan matrik A=B*C.

Hanya saja karena ukuran vektor 3 dimensi adalah 4, maka ukuran matrik transformasi 3 dimensi adalah 4x4. Definisi matrik transformasi 3 dimensi dan matrik identitas adalah sebagai berikut:

```
typedef struct {
        float m[4][4];
} matrix3D_t;
```

```
matrix3D_t createIdentity(){
        matrix3D_t mat;
        for(int i=0;i<4;i++) {
                for(int j=0;j<4;j++) mat.m[i][j]=0;
                mat.m[i][i]=1;
        }}
```



Translasi

Proses translasi adalah proses untuk memindahkan obyek ke arah sumbu X, sumbu Y dan sumbu Z sebesar (*dx*,*dy*,*dz*) Matrik transformasi dari proses translasi 3 dimensi seperti matrik translasi 2 dimensi dengan menambahkan nilai z, adalah:

$$T = \begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

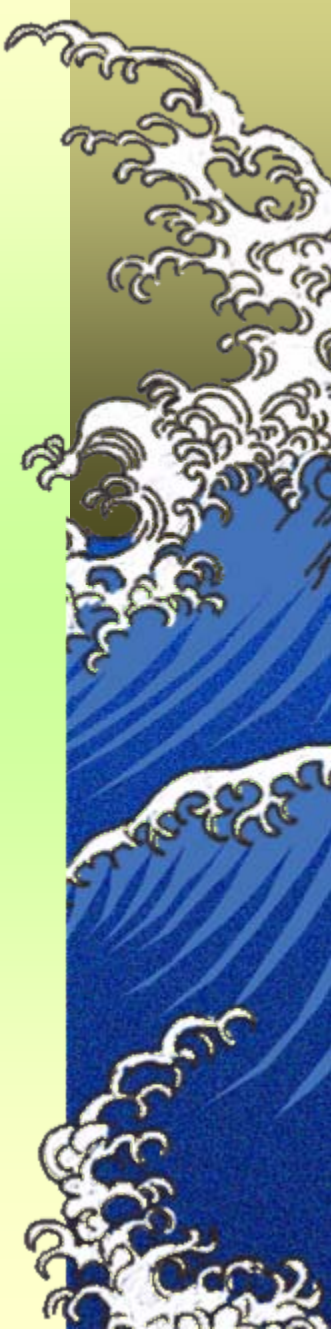
```
matrix3D_t translationMTX(float dx, float dy, float dz){
        matrix3D_t mat=createIdentity();
        mat.m[0][3]=dx;
        mat.m[1][3]=dy;
        mat.m[2][3]=dz;
        return mat;
}
```

Scaling

Seperti matrik transformasi scaling 2 dimensi, matrik transformasi dari proses scaling 3 dimensi adalah:

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

```
matrix3D_t scalingMTX(float mx, float my, float mz){
        matrix3D_t mat=createIdentity();
        mat.m[0][0]=mx;
        mat.m[1][1]=my;
        mat.m[2][2]=mz;
        return mat;
}
```



Rotasi

Untuk proses rotasi 3 dimensi, terdapat tiga macam rotasi, yaitu rotasi terhadap sumbu X (rotationXMTX), rotasi terhadap sumbu Y (rotationYMTX) dan rotasi pada sumbu Z (rotationZMTX). Perbedaan dari masing-masing matriks rotasi ini adalah peletakan nilai cos(*a*) dan sin(*a*).

Rotasi thdp Sb X

$$R_X = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & \cos(a) & -\sin(a) & 0 \\ 0 & \sin(a) & \cos(a) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

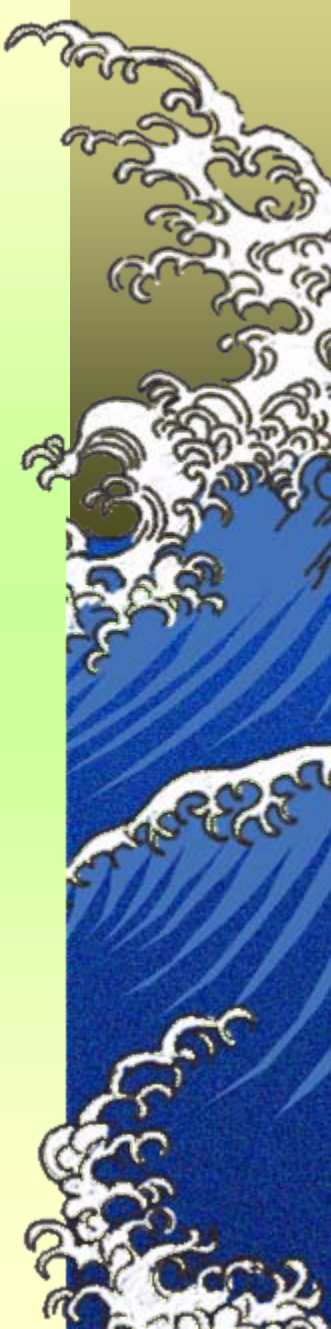
```
matrix3D_t rotationXMTX(float theta)
{
        matrix3D_t rotate=createIdentity();
        float cs=cos(theta);
        float sn=sin(theta);
        rotate.m[1][1]=cs; rotate.m[1][2]=-sn;
        rotate.m[2][1]=sn; rotate.m[2][2]=cs;
        return rotate;
}
```

Rotasi

Rotasi thdp Sb Y

$$R_Y = \begin{bmatrix} \cos(a) & 0 & -\sin(a) & 0 \\ 0 & 0 & 0 & 0 \\ \sin(a) & 0 & \cos(a) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
matrix3D_t rotationYMTX(float theta)
{
        matrix3D_t rotate=createIdentity();
        float cs=cos(theta);
        float sn=sin(theta);
        rotate.m[0][0]=cs; rotate.m[0][2]=sn;
        rotate.m[2][0]=-sn; rotate.m[2][2]=cs;
        return rotate;
}
```

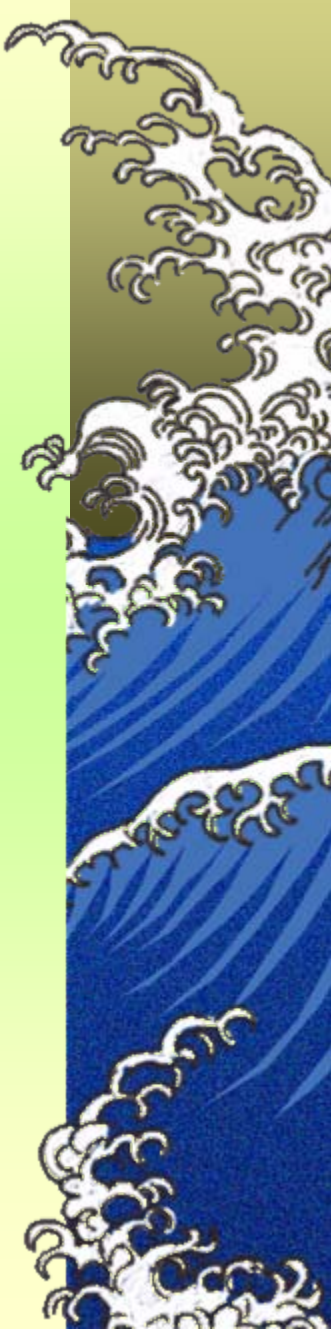


Rotasi

Rotasi thdp Sb Z

$$\begin{bmatrix} \cos(a) & -\sin(a) & 0 & 0 \\ \sin(a) & \cos(a) & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
matrix3D_t rotationZMTX(float theta)
{
        matrix3D_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;
}
```



Operasi Matrik dan Vektor

Penjumlahan Vektor

```
vector3D_t operator + (vector3D_t a, vector3D_t b){
        vector3D_t c;
        for(int i=0;i<4;i++) c.v[i]=a.v[i]+b.v[i];
        return c;
}
```

Pengurangan Vektor

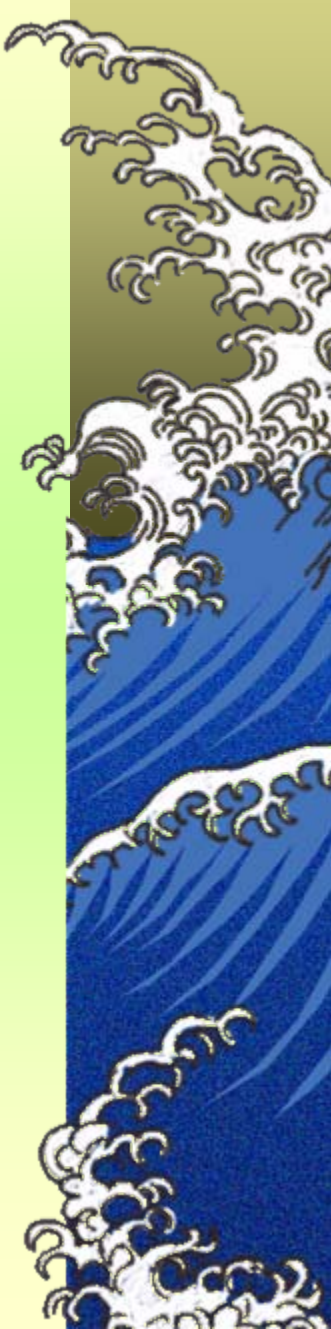
```
vector3D_t operator - (vector3D_t a, vector3D_t b){
        vector3D_t c;
        for(int i=0;i<4;i++) c.v[i]=a.v[i]-b.v[i];
        return c;
}
```

Operasi Matrik dan Vektor

Penjumlahan Vektor

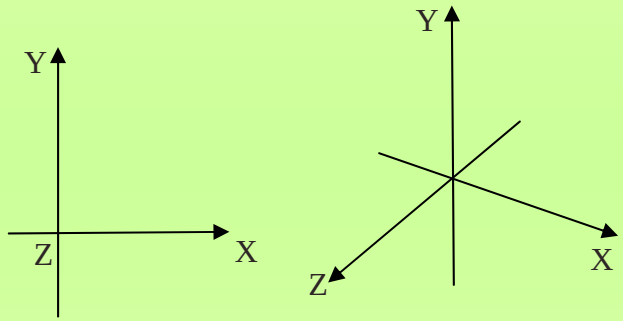
```
vector3D_t operator * (matrix3D_t a, vector3D_t b){
        vector3D_t c;
        for(int i=0;i<4;i++) {
                c.v[i]=0;
                        for(int j=0;j<4;j++)
                c.v[i]+=a.m[i][j]*b.v[j];
        }
        return c;
}
```

```
matrix3D_t operator * (matrix3D_t a, matrix3D_t b){
        matrix3D_t c;
        for(int i=0;i<4;i++)
                for(int j=0;j<4;j++){
                        c.m[i][j]=0;
                        for(int k=0;k<4;k++)
                        c.m[i][j]+=a.m[i][k]*b.m[k][j];
                }
        return c;
}
```



Matrik Tilting

Matriks tilting adalah suatu bentuk matrik transformasi yang digunakan untuk mentransformasikan sumbu koordinat sehingga obyek terlihat dengan detail yang cukup lengkap. Boleh dikatakan bahwa matriks ini menyatakan sudut pandang suatu obyek ruang.



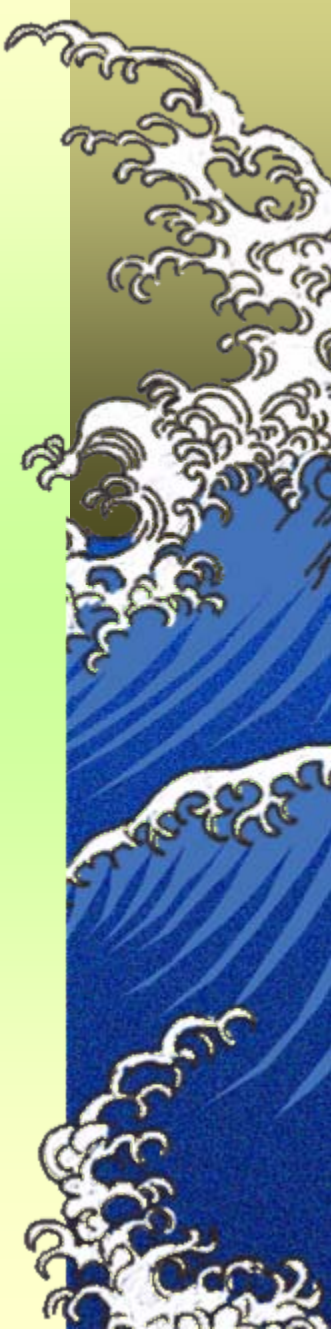
Pembuatan matrik tilting dapat secara langsung ditulis dalam userdraw() dengan menggunakan komposisi transformasi.

```
matrix3D_t tilting=rotationXMTX(0.25)*rotationYMTX(-0.5);
drawAxes(tilting);
```

Matrik Tilting

Fungsi drawAxes(tilting) adalah fungsi untuk menggambar sumbu koordinat dengan matriks transformasi tilting, sebagai berikut.
Proses ini terdiri dari menggambar huruf X, Y dan Z,
dan menggambar posisi garis sumbu koordinat masing-masing.

```
void drawcharX(float x,float y){
drawLine(x,y,x+10,y+12);drawLine(x,y+12,x+10,y);
}
void drawcharY(float x,float y){
drawLine(x+5,y,x+5,y+7);drawLine(x,y+12,x+5,y+7);
drawLine(x+10,y+12,x+5,y+7);
}
void drawcharZ(float x,float y){
drawLine(x,y+12,x+10,y+12);drawLine(x+10,y+12,x,y);
drawLine(x,y,x+10,y);
}
void drawAxes(matrix3D_t view){
#define HALFAXIS  220
#define HALFAXIS1 (HALFAXIS-10)
```



Matrik Tilting

```
point3D_t axes[14]={
                {-HALFAXIS,0,0}, {HALFAXIS,0,0},
                {HALFAXIS1,5,0},{HALFAXIS1,0,0},{0,0,0},
                {0,-HALFAXIS,0}, {0,HALFAXIS,0},
                {0,HALFAXIS1,5},{0,HALFAXIS1,0},{0,0,0},
                {0,0,-HALFAXIS}, {0,0,HALFAXIS},
                {5,0,HALFAXIS1}, {0,0,HALFAXIS1}
                };
vector3D_t vec[14];
point2D_t buff[14];
int i;
for (i=0;i<14;i++) {
                vec[i]=Point2Vector(axes[i]);
                vec[i]=view*vec[i];
                buff[i]=Vector2Point2D(vec[i]);
}
drawPolyline(buff,14);drawcharX(buff[1].x,buff[1].y);
drawcharY(buff[6].x,buff[6].y);
drawcharZ(buff[11].x-14,buff[11].y);}
```

Fungsi untuk menggunakan matrik tilting pada userdraw adalah sebagai berikut:

```
void userdraw(void){
matrix3D_t tilting=rotationXMTX(a)*rotationYMTX(b);
setColor(0,1,0);
drawAxes(tilting);
}
```

Contoh Pemakaian Matrik Tilting

Pemakaian tilting untuk menstransformasikan sistem koordinat 3 dimensi atau mengubah sudut pandangan adalah dengan memutar sumbu koordinat dengan sudut a ke arah sumbu x dan memutar sumbu koordinat dengan sudut b ke arah sumbu Y.

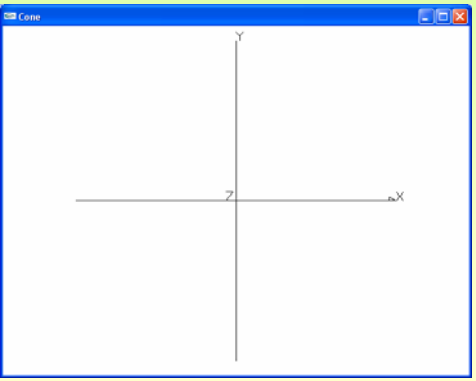
Contoh pemakaian beberapa sudut a dan b yang berbeda (dengan satuan radian).

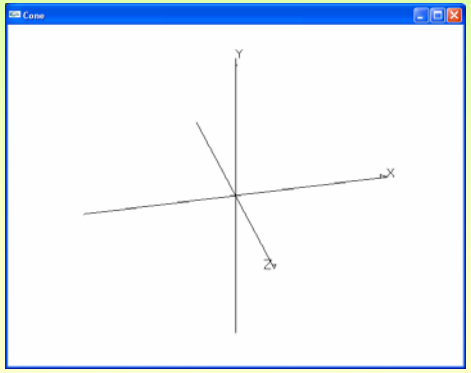
(1) $\alpha = 0$ dan $\beta = 0$

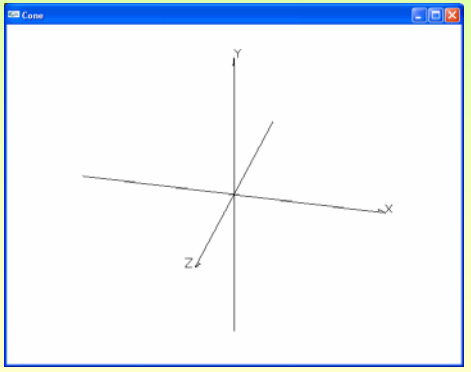
(2) $\alpha = 0.5$ dan $\beta = 0.25$

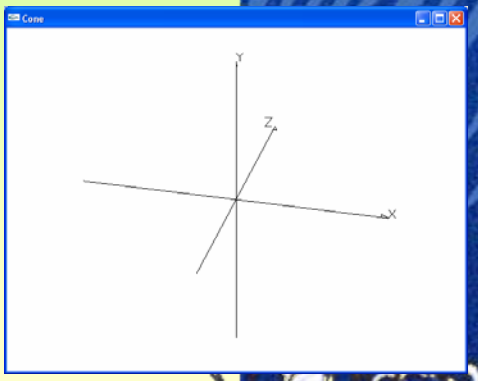
(3) $\alpha = 0.5$ dan $\beta = -0.25$

(4) $\alpha = -0.5$ dan $\beta = 0.25$









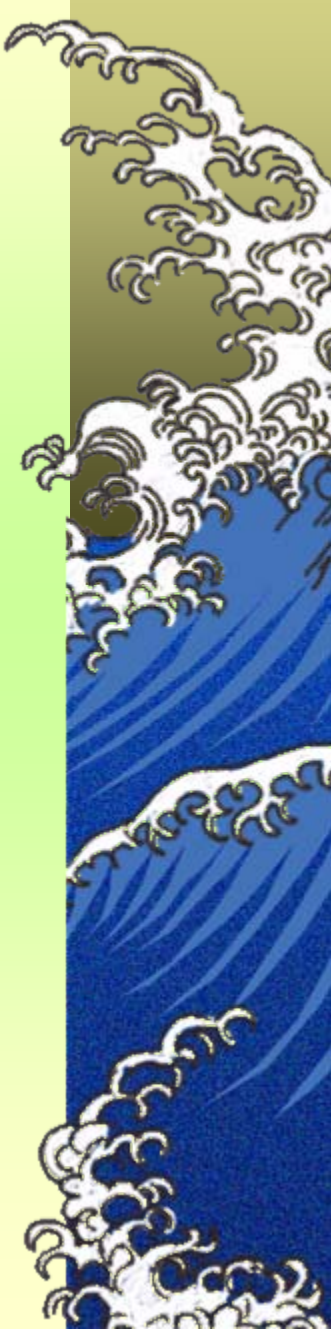
Implementasi Cara Menggambar Obyek 3D

```
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 di dalamnya.

```
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:

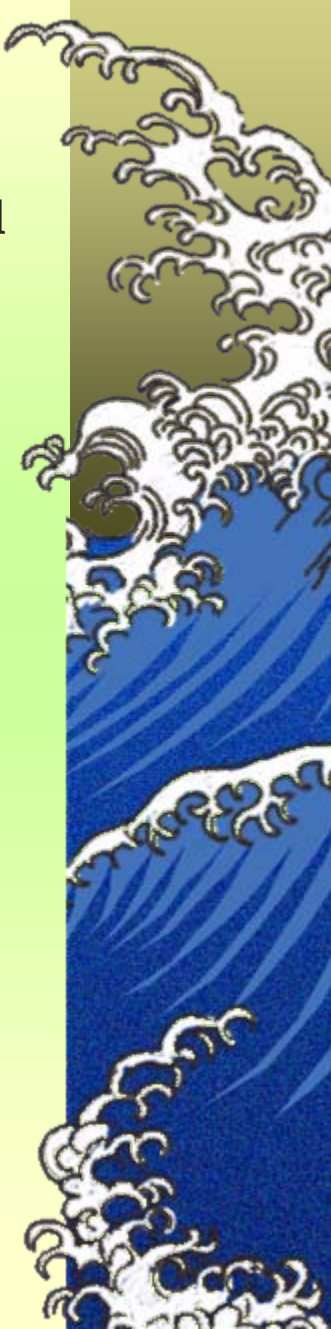
(1) Ambil vektor dari setiap titik pada face tersebut

(2) Konversikan setiap vektor 3D menjadi titik 2D

(3) Dari hasil konversi digambarkan polygon

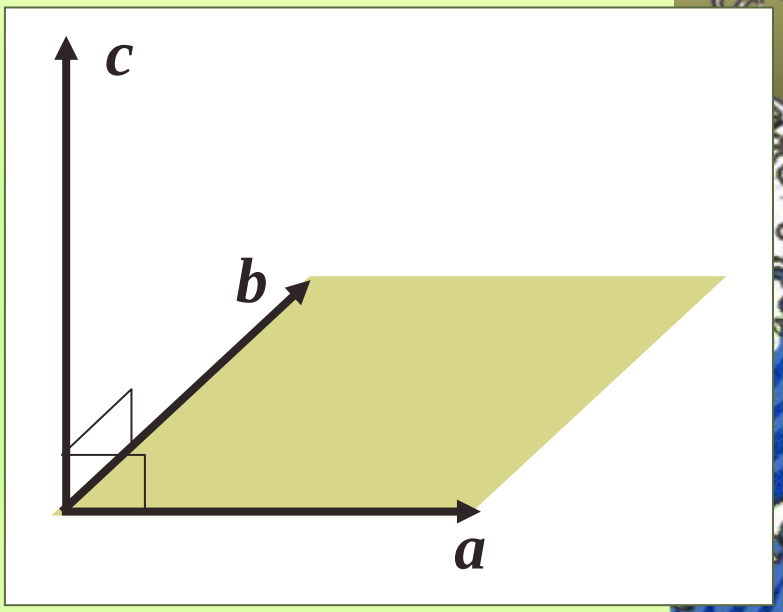
Visible dan Invisible

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



Vektor Normal

- Vektor normal adalah vektor yang arahnya tegak lurus dengan luasan suatu face
- Vektor normal adalah hasil perkalian silang vektor (cross-product) dari vektor-vektor yang ada pada luasan face



$$c = a \times b$$

Perkalian Silang (Cross Product)

Perkalian silang (cross product) dari vektor $a=(a_x,a_y,a_z)$ dan $b=(b_x,b_y,b_z)$ didefinisikan dengan

$$c = \begin{vmatrix} i & j & k \\ a_x & a_y & a_z \\ b_x & b_y & b_z \end{vmatrix}$$

$$= i(a_y b_z - a_z b_y) + j(a_z b_x - a_x b_z) + k(a_x b_y - a_y b_x)$$

$$= (a_y b_z - a_z b_y, a_z b_x - a_x b_z, a_x b_y - a_y b_x)$$

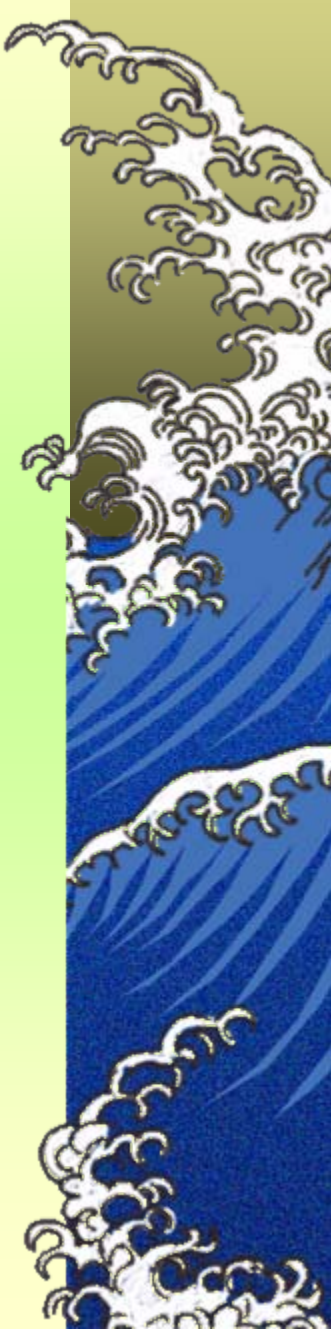
Implementasi Cross-Product

```
vector3D_t operator ^ (vector3D_t a, vector3D_t b)
{
        vector3D_t c;//c=a*b
        c.v[0]=a.v[1]*b.v[2]-a.v[2]*b.v[1];
        c.v[1]=a.v[2]*b.v[0]-a.v[0]*b.v[2];
        c.v[2]=a.v[0]*b.v[1]-a.v[1]*b.v[0];
        c.v[3]=1.;
        return c;
}
```

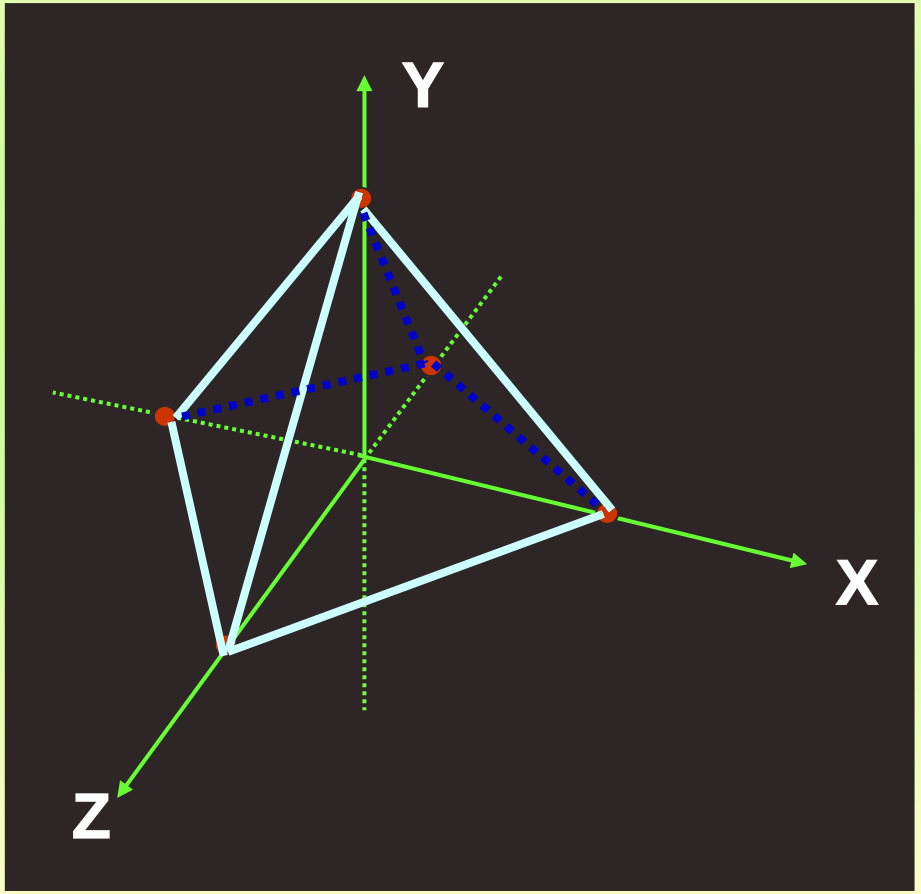
Cross product disimbolkan dengan operator ^
$a=(a_x,a_y,a_z)$ diubah sesuai struktur data dari vektor 3D menjadi (a.v[0],a.v[1],a.v[2])
$b=(b_x,b_y,b_z)$ diubah sesuai struktur data dari vektor 3D menjadi (b.v[0],b.v[1],b.v[2])

Implementasi Visible dan Invisible

- Untuk mengimplementasikan face visible dan invisible maka dilakukan penggambaran dua kali
- Pertama digambar dulu face-face yang invisible (NormalVector.v[2]<0)
- Kedua digambar face-face yang visible (NormalVector.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)