DirectX Programming #2

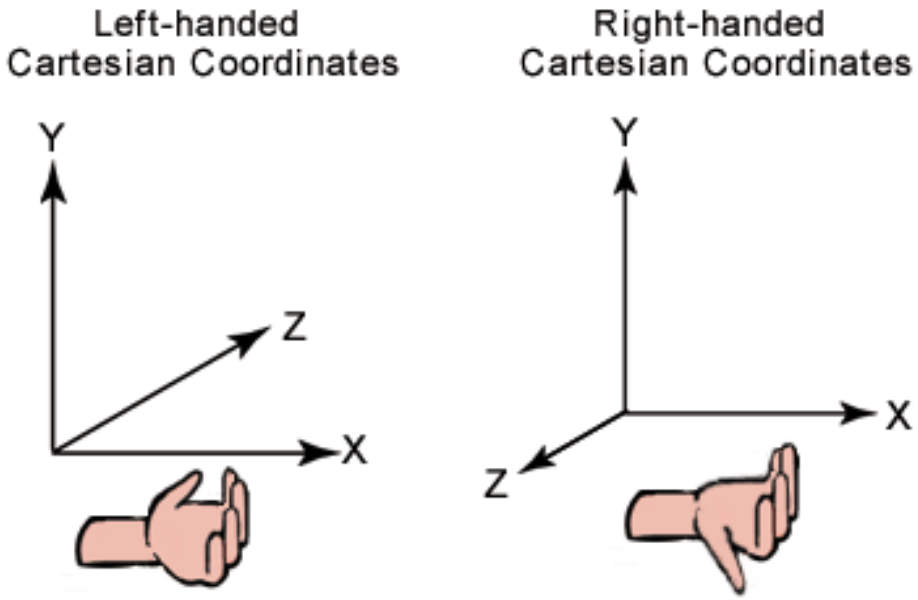
Kang, Seongtae
Computer Graphics, 2009 Spring

Contents

- The D3D coordinate system
- Transformation and matrix manipulation
- Material and Lighting

The D3D Coordinate System

- Left-handed
  - Right-handed coordinate is also available
  - Take care of backface-culling order



Vector Representation

- D3DXVECTOR*n* class
  - Float elements
    - x, y for D3DXVECTOR2
    - x, y, z for D3DXVECTOR3
  - Basic operators
    - Scalar multiplication/division
    - Vector addition/subtraction and equality comparison

Matrix Representation

- D3DXMATRIX Structure
  - 4x4 homogeneous matrix
  - Row-major order
    - Column-major order matrices must be transposed
  - Can be accessed as float[16] array

```
typedef struct D3DXMATRIX {
    FLOAT _11, FLOAT _12, FLOAT _13, FLOAT _14,
    FLOAT _21, FLOAT _22, FLOAT _23, FLOAT _24,
    FLOAT _31, FLOAT _32, FLOAT _33, FLOAT _34,
    FLOAT _41, FLOAT _42, FLOAT _43, FLOAT _44 );
} D3DXMATRIX;
```

Basic Matrix Operations

- Identity matrix

```
D3DXMATRIX * D3DXMatrixIdentity(D3DXMATRIX * pOut);
```

pOut = I

- Transpose

```
D3DXMATRIX * D3DXMatrixTranspose(D3DXMATRIX *pOut, CONST D3DXMATRIX *pM);
```

$pOut = (pM)^T$

- Inverse

```
D3DXMATRIX * D3DXMatrixInverse(D3DXMATRIX *pOut, FLOAT *pDeterminant,
                               CONST D3DXMATRIX *pM);
```

$pOut = (pM)^{-1}$, pDeterminant = det(pM)

Basic Matrix Operations

- Multiplication

```
D3DXMATRIX * D3DXMatrixMultiply(D3DXMATRIX *pOut,
                                CONST D3DXMATRIX *pM1, CONST D3DXMATRIX *pM2);

                                pOut = pM1 X pM2
```

Matrix Multiplication Order

- Row-major matrix
- Row vector
- Post-multiplication

$$[x'\ y'\ z'\ 1] = [x\ \ y\ \ z\ \ 1]\begin{bmatrix} M_{11} & M_{12} & M_{13} & M_{14} \\ M_{21} & M_{22} & M_{23} & M_{24} \\ M_{31} & M_{32} & M_{33} & M_{34} \\ M_{41} & M_{42} & M_{43} & M_{44} \end{bmatrix}$$

$$C = M_1 \cdot M_2 \cdot \ M_{n-1} \cdot \ M_n \longrightarrow$$

Setting Transform Matrices

- World transformation
  - Model space → World space

```
IDirect3DDevice::SetTransform(D3DTS_WORLD, &matWorld);
```

- Viewing transformation
  - World space → Viewing space

```
IDirect3DDevice::SetTransform(D3DTS_VIEW, &matView);
```

- Projection transformation
  - Viewing space → Projection space

```
IDirect3DDevice::SetTransform(D3DTS_PROJECTION, &matProjection);
```

$$(x', y', z', w') = (x, y, z, w) \times M_{world} \times M_{view} \times M_{proj}$$

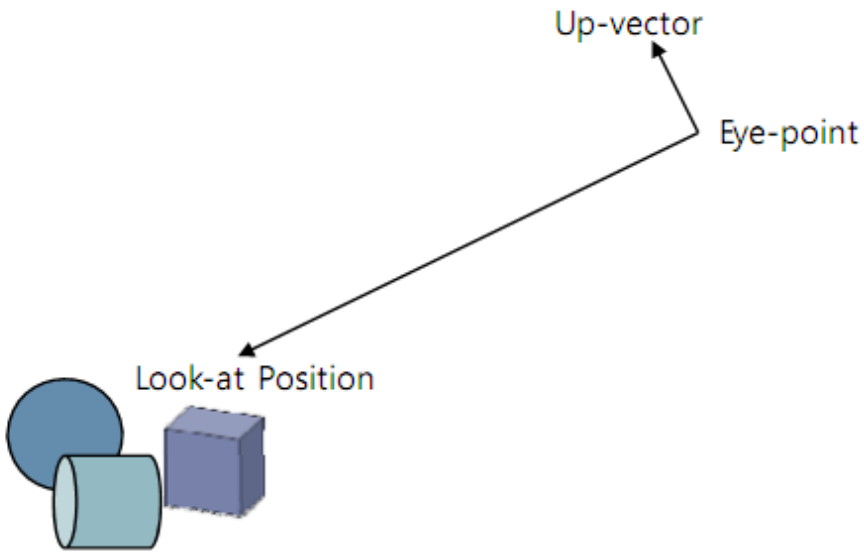
World Transformation

- Scaling, rotation and translation
  - Many predefined functions are defined in DX library
    - D3DXMatrixRotationAxis
    - D3DXMatrixRotationQuaternion
    - D3DXMatrixRotationX
    - D3DXMatrixScaling
    - D3DXMatrixTranslation
    - Et al. – See the SDK document
  - Composite predefined transformations using multiplication

```
D3DXMATRIX matWorld;
D3DXMatrixRotationX( &matRotX, RotateX );
D3DXMatrixRotationY( &matRotY, RotateY );
D3DXMatrixRotationZ( &matRotZ, RotateZ );
D3DXMatrixMultiply( &matWorld, &matRotZ, &matRotY );
D3DXMatrixMultiply( &matWorld, &matWorld, &matRotX );
g_pD3DDevice->SetTrasform( D3DTS_WORLD, &matWorld );
```

Viewing Transformation

- “Camera” transformation
- Transformation specifier
  - Eye point
  - Look-at position
  - Up vector




```
D3DXMATRIX matView;
D3DXVECTOR3 vEyePt( 0.0f, 3.0f,-5.0f );
D3DXVECTOR3 vLookatPt( 0.0f, 0.0f, 0.0f );
D3DXVECTOR3 vUpVec( 0.0f, 1.0f, 0.0f );
D3DXMATRIXA16 matView;
D3DXMatrixLookAtLH( &matView, &vEyePt, &vLookatPt, &vUpVec );
g_pd3dDevice->SetTransform( D3DTS_VIEW, &matView );
```

Projection Transformation

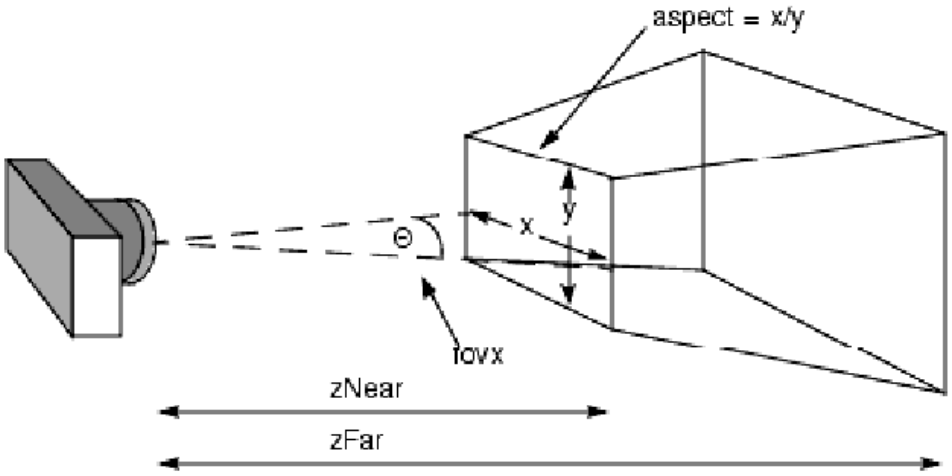
- 3D viewing space → 2D viewport space

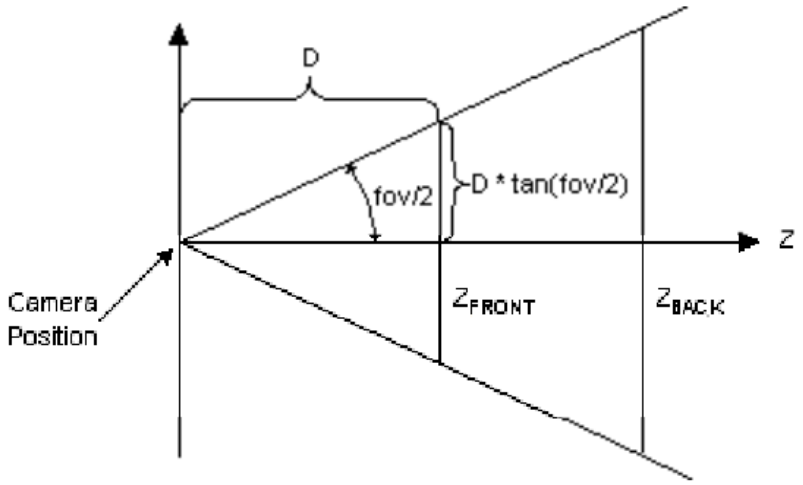
- Perspective projection
  - D3DXMatrixPerspectiveLH
  - D3DXMatrixPerspectiveFovLH
  - D3DXMatrixPerspectiveOffCenterLH

- Orthogonal projection
  - D3DXMatrixOrthoLH
  - D3DXMatrixOrthoOffCenterLH

Projection Transformation

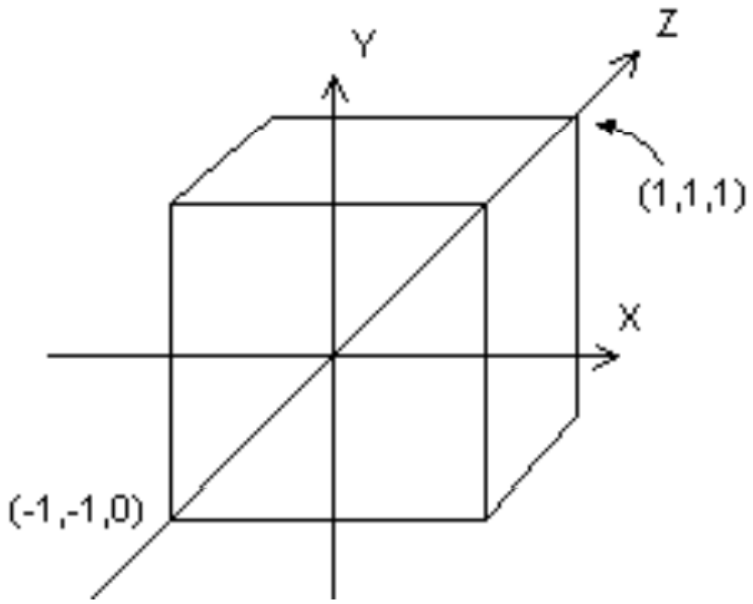
- Transformation parameters





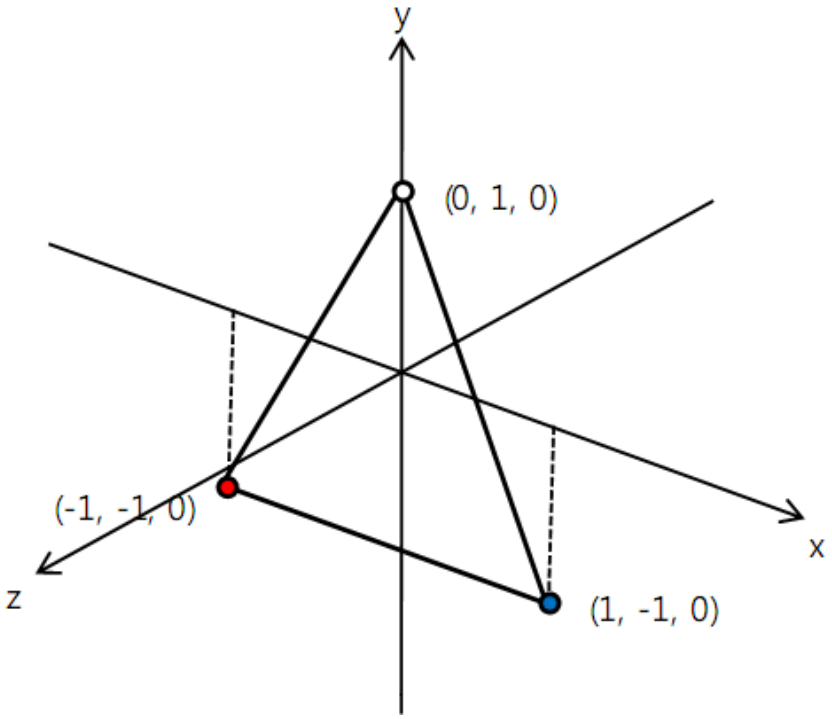
Projection Transformation

- Projection space
  - X, Y : [-1, 1], viewport
  - Z : [0,1], depth



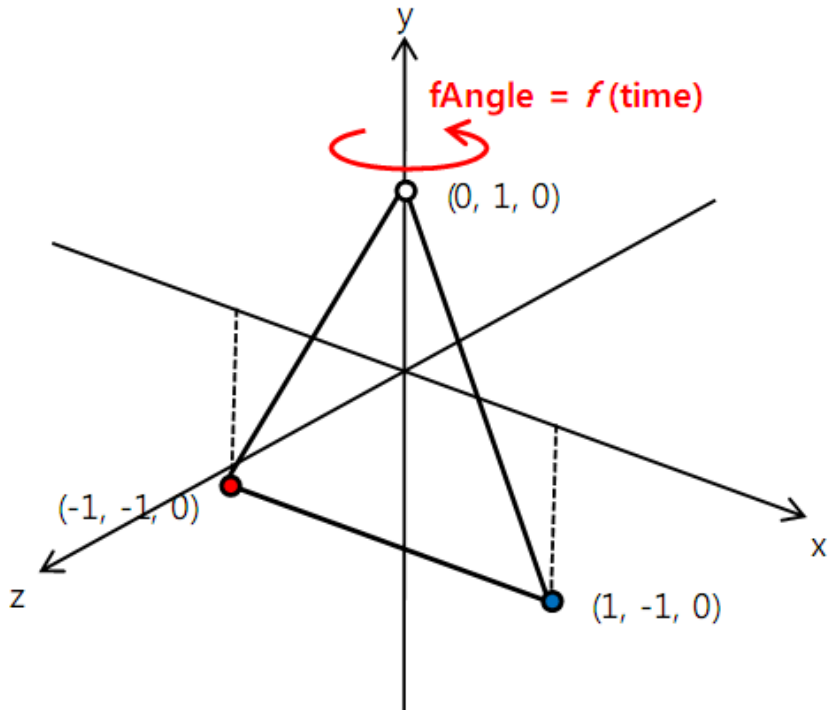
Setting Vertices : Tutorial 3

```
CUSTOMVERTEX g_Vertices[] =
  {
     { -1.0f,-1.0f, 0.0f, 0xffff0000, },
     {  1.0f,-1.0f, 0.0f, 0xff0000ff, },
     {  0.0f, 1.0f, 0.0f, 0xffffffff, },
  };
```



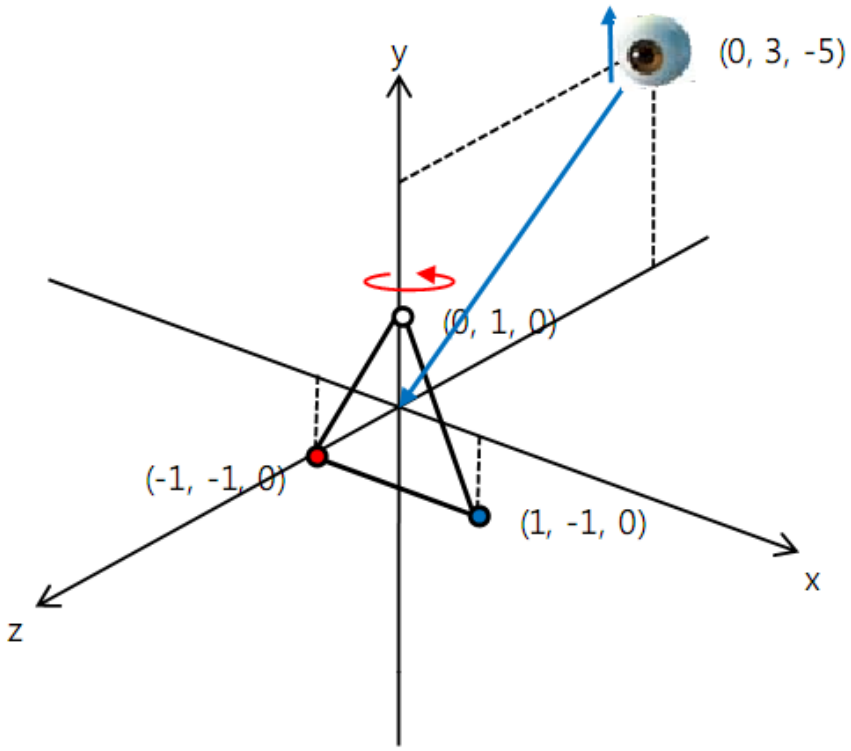
World Transformation : Tutorial 3

```
UINT  iTime  = timeGetTime() % 1000;
FLOAT fAngle = iTime * (2.0f * D3DX_PI) / 1000.0f;
D3DXMatrixRotationY( &matWorld, fAngle );
g_pd3dDevice->SetTransform( D3DTS_WORLD, &matWorld );
```



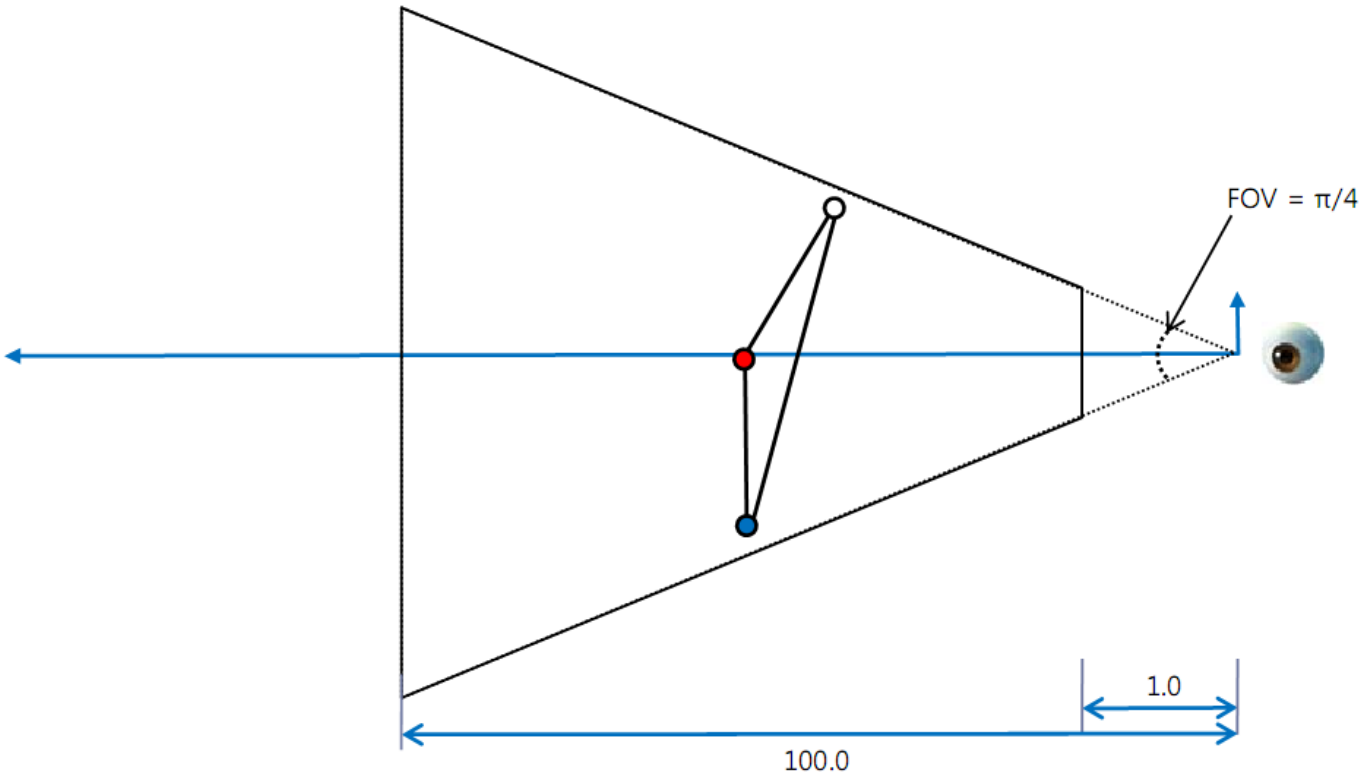
Viewing Transformation : Tutorial 3

```
D3DXVECTOR3 vEyePt( 0.0f, 3.0f,-5.0f );
D3DXVECTOR3 vLookatPt( 0.0f, 0.0f, 0.0f );
D3DXVECTOR3 vUpVec( 0.0f, 1.0f, 0.0f );
D3DXMATRIXA16 matView;
D3DXMatrixLookAtLH( &matView, &vEyePt, &vLookatPt, &vUpVec );
g_pd3dDevice->SetTransform( D3DTS_VIEW, &matView );
```

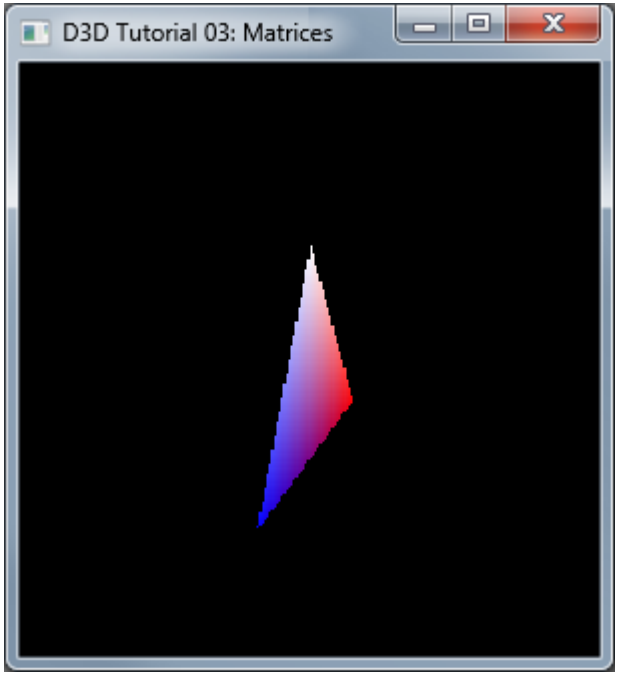


Projection Transformation : Tutorial 3

```
D3DXMATRIXA16 matProj;
D3DXMatrixPerspectiveFovLH( &matProj, D3DX_PI/4, 1.0f, 1.0f, 100.0f );
g_pd3dDevice->SetTransform( D3DTS_PROJECTION, &matProj );
```



Result : Tutorial 3

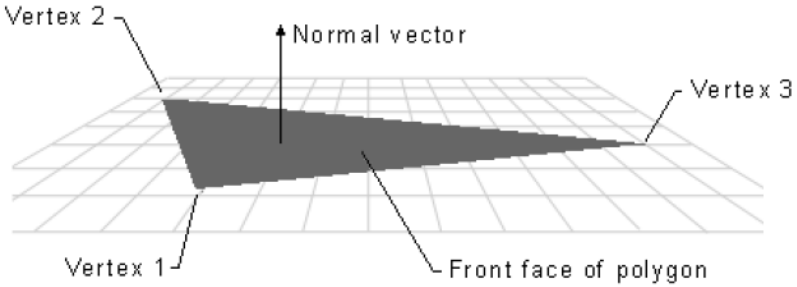


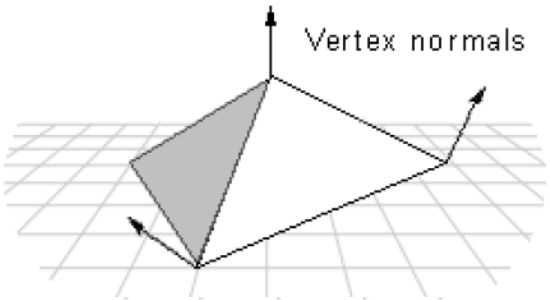
Shading

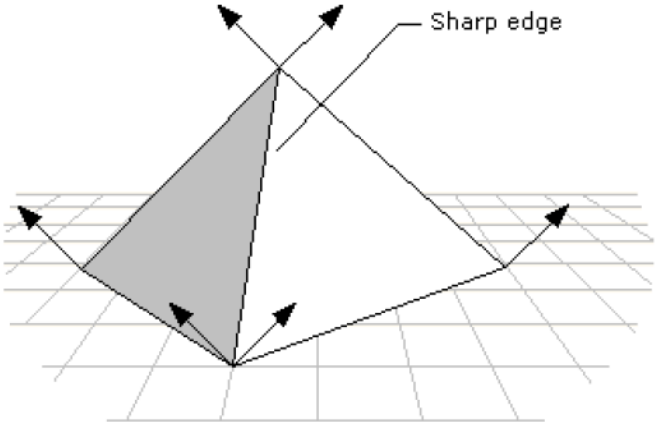
- Calculating intensity for each pixel based on
  - Object descriptions
  - Optical features for surface
  - Viewing parameters

Normal

- For shading, you have to specity the vertex normal








```
struct CUSTOMVERTEX
{
    D3DXVECTOR3 position; // The 3D position for the vertex.
    D3DXVECTOR3 normal;   // The Quad normal for the vertex.
};

#define D3DFVF_CUSTOMVERTEX (D3DFVF_XYZ|D3DFVF_NORMAL)
```

Material

- Defines characteristics of the quad of a geometric object
  - Color
    - Diffuse
    - Ambient
    - Specular
    - Emission
  - Glossiness
    - Power (of exponent term)

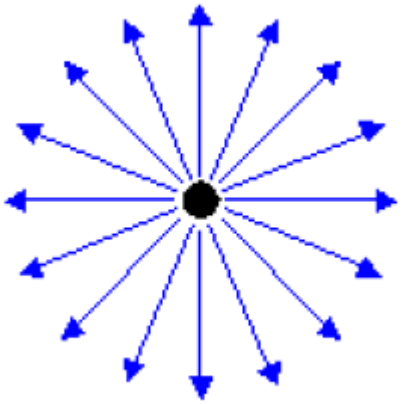
Per-vertex colors

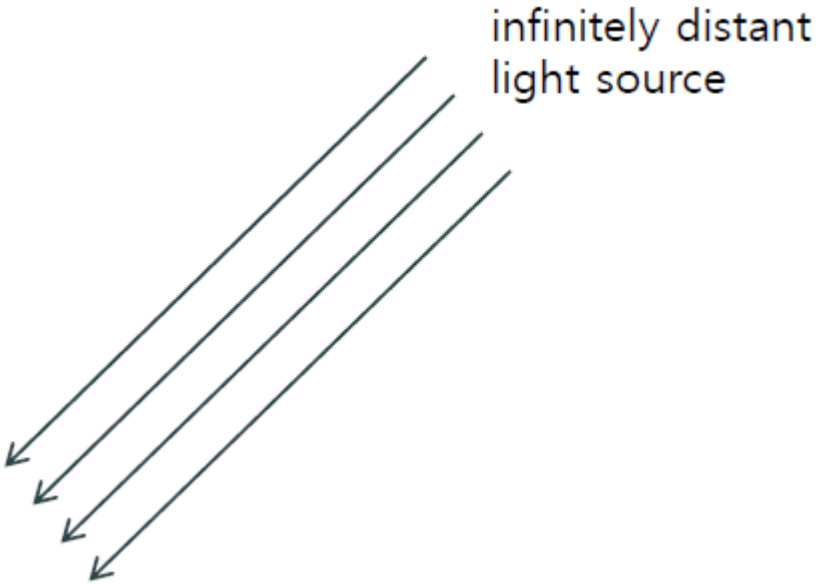
- We have used per-vertex color already
  - D3DFVF_DIFFUSE : diffuse color
- Either per-vertex information or material based information can be used as shading factors
  - D3DMCS_MATERIAL : material color
  - D3DMCS_COLOR1 : per-vertex diffuse color(D3DFVF_DIFFUSE)
  - D3DMCS_COLOR2 : per-vertex specular color(D3DFVF_SPECULAR)

```
g_pd3dDevice->SetRenderState( D3DRS_DIFFUSEMATERIALSOURCE, D3DMCS_COLOR1 );
g_pd3dDevice->SetRenderState( D3DRS_SPECULARMATERIALSOURCE, D3DMCS_MATERIAL);
```

Light Type

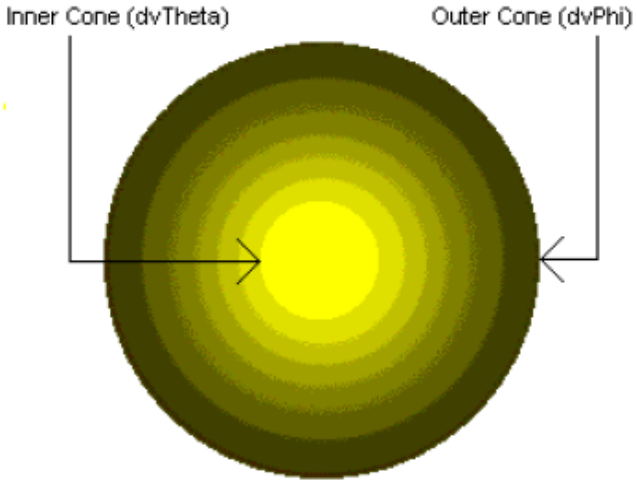
- Point light
  - Position and color
  - Light bulb without lampshade
- Directional light
  - Color and direction
  - Sunray



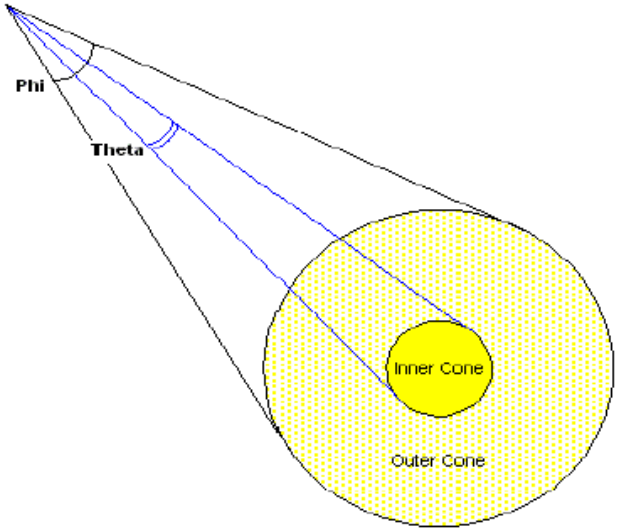


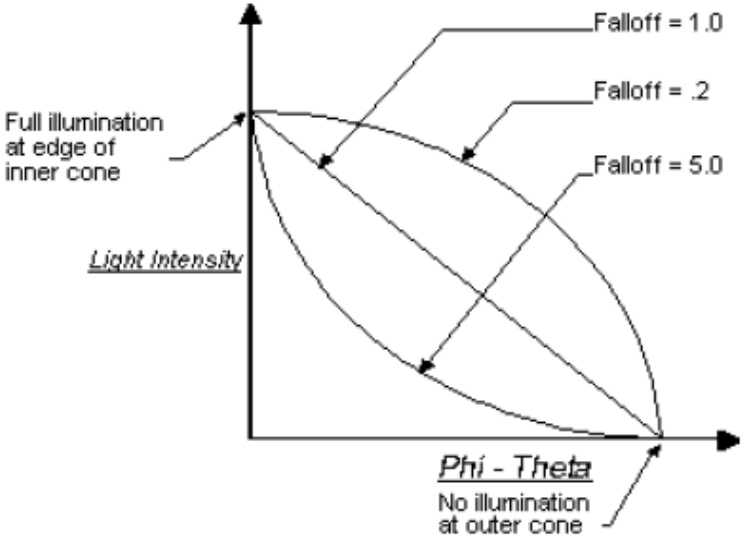
Light Type

- Spotlight
  - Color, position, direction
  - Cone shape illumination area
  - Theta, phi, falloff, attenuation0/1/2



$$Atten = 1/(att0_i + att1_i * d + att2_i * d^2)$$





Using Lights

- Light configutation

```
D3DXVECTOR3 vecDir;

D3DLight9 light;
ZeroMemory( &light, sizeof(light) );
light.Type = D3DLIGHT_DIRECTIONAL;

light.Diffuse.r = 1.0f; light.Diffuse.g = 1.0f; light.Diffuse.b = 1.0f;

vecDir=D3DXVECTOR3(1.0f, 2.0f, 0.0f);
D3DXVec3Normalize( (D3DXVECTOR3*)&light.Direction, &vecDir );
```

- Attach light to the device

```
g_pd3dDevice->SetLight( 0, &light );
```

- Light direction vector need not be normalized.

Using Lights

- Enable lighting

```
g_pd3dDevice->SetRenderState( D3DRS_LIGHTING, TRUE );   // global configuration
g_pd3dDevice->LightEnable( 0, TRUE);                    // per-light configuration
```

- Ambient color setting
  - All object will be lighted up by this color, regardless of its material variables

```
g_pd3dDevice->SetRenderState( D3DRS_AMBIENT, 0x00202020 );
```

Shading Model

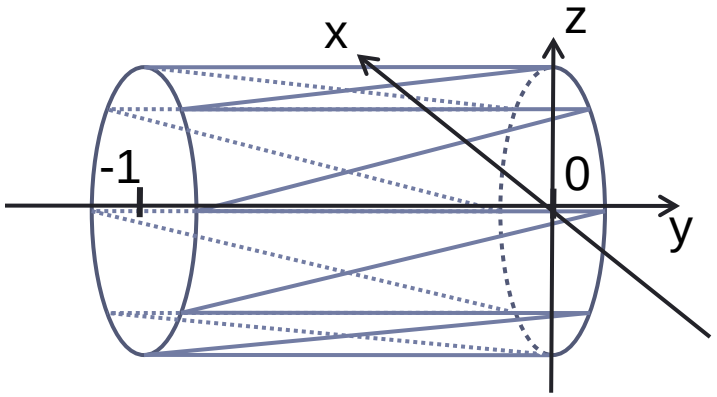
- Three shading model
  - Flat shading : D3DSHADE_FLAT
  - Gouraud shading : D3DSHADE_GOURAUD
  - Phong shading : D3DSHADE_PHONG
- D3D9 does not support implicit Phong shading
  - Setting the shading model to D3D_SHADE_PHONG will lead unpredictable behavior
  - Can be implemented using the programmable shader
    - (This will be given as the final assignment)

```
pDev->SetRenderState(D3DRS_SHADEMODE, D3DSHADE_FLAT);
```

- The default shading model is D3DSHADE_GOURAUD

Creating Polygon : Tutorial 4

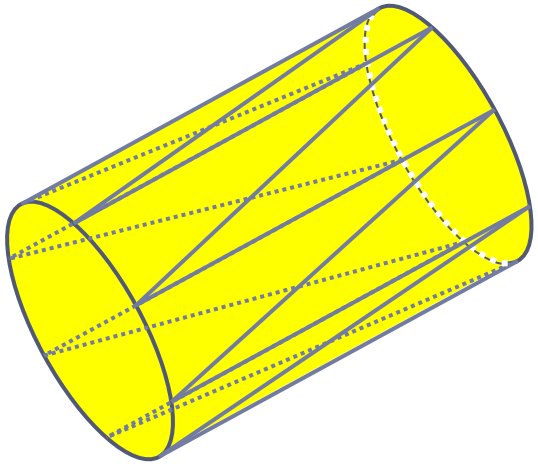
- Generating a cylinder
  - as a sequence of triangle strips




```
CUSTOMVERTEX* pVertices;
   if( FAILED( g_pVB->Lock( 0, 0, ( void** )&pVertices, 0 ) ) )
       return E_FAIL;
   for( DWORD i = 0; i < 50; i++ )
   {
       FLOAT theta = ( 2 * D3DX_PI * i ) / ( 50 - 1 );
       pVertices[2 * i + 0].position = D3DXVECTOR3( sinf( theta ), -1.0f, cosf( theta ) );
       pVertices[2 * i + 0].normal = D3DXVECTOR3( sinf( theta ), 0.0f, cosf( theta ) );
       pVertices[2 * i + 1].position = D3DXVECTOR3( sinf( theta ), 1.0f, cosf( theta ) );
       pVertices[2 * i + 1].normal = D3DXVECTOR3( sinf( theta ), 0.0f, cosf( theta ) );
   }
   g_pVB->Unlock();
```

Material Setup : Tutorial 4

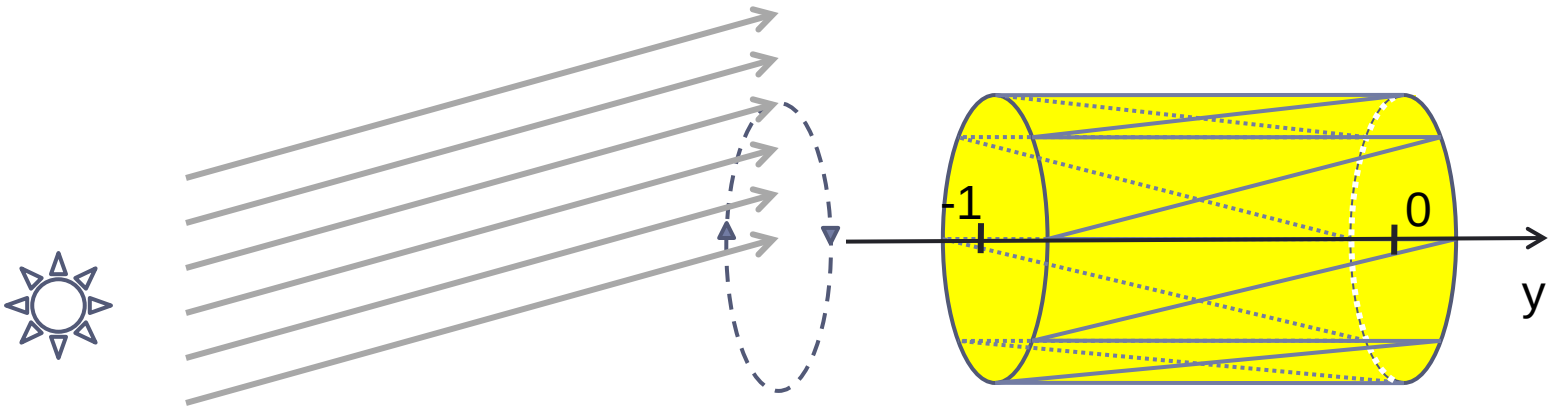
- Creating a yellow material
  - With yellow ambient and diffuse color



```
D3DMATERIAL9 mtrl;
  ZeroMemory( &mtrl, sizeof( D3DMATERIAL9 ) );
  mtrl.Diffuse.r = mtrl.Ambient.r = 1.0f;
  mtrl.Diffuse.g = mtrl.Ambient.g = 1.0f;
  mtrl.Diffuse.b = mtrl.Ambient.b = 0.0f;
  mtrl.Diffuse.a = mtrl.Ambient.a = 1.0f;
  g_pd3dDevice->SetMaterial( &mtrl );
```

Light Setup : Tutorial 4

- A directional light source
  - With oscillating direction




```
vecDir = D3DXVECTOR3( cosf( timeGetTime() / 350.0f ),
                      1.0f,
                      sinf( timeGetTime() / 350.0f ) );
D3DXVec3Normalize( ( D3DXVECTOR3* )&light.Direction, &vecDir );
light.Range = 1000.0f;
```

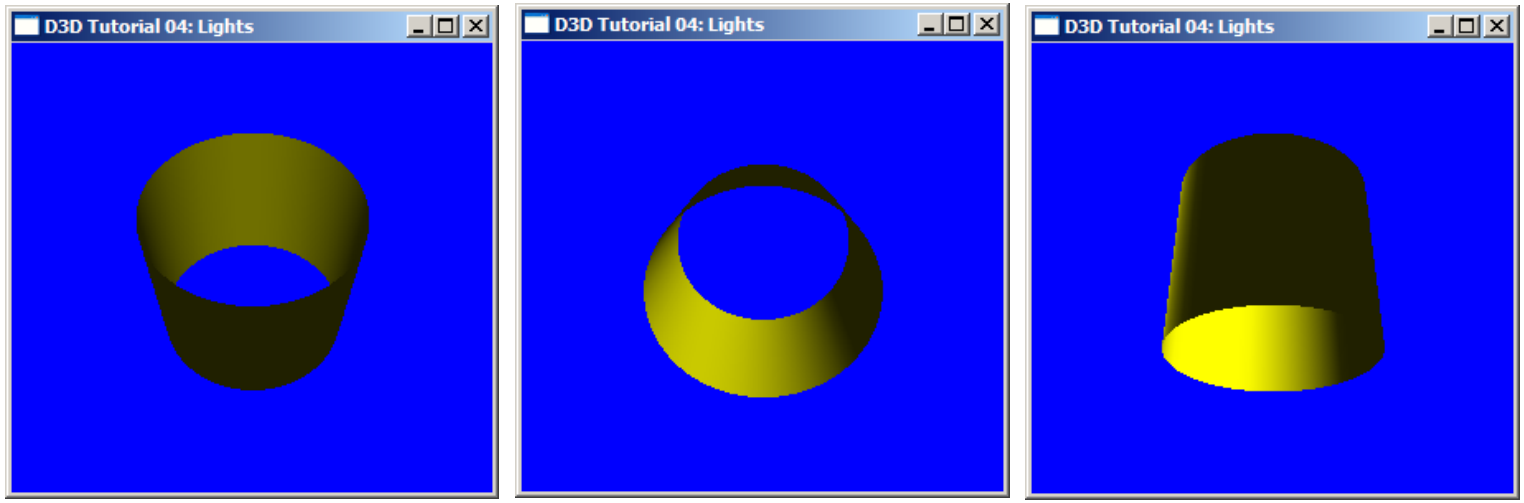
Light Setup : Tutorial 4

- Enabling a light and lighting
- Turning on the ambient light

```
g_pd3dDevice->SetLight( 0, &light );
g_pd3dDevice->LightEnable( 0, TRUE );
g_pd3dDevice->SetRenderState( D3DRS_LIGHTING, TRUE );

g_pd3dDevice->SetRenderState( D3DRS_AMBIENT, 0x00202020 );
```

Result : Tutorial 4



Practice Assignments

- Compile and run Tutorial 3, 4
  - Try other transformations (translation, rotation, etc.)
  - Try each light types and variation on material properties
- Review texture mapping