DirectX Programming #2

Kang, Seong-tae

Computer Graphics, 2008 Spring

Contents

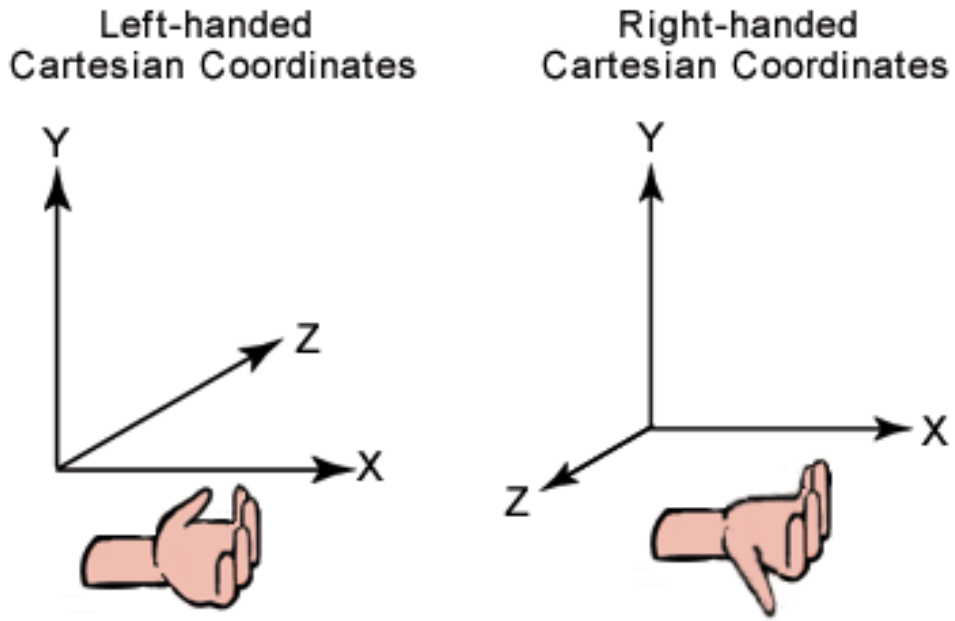
- The D3D coordinate system
- Transformation and matrix manipulation

The D3D Coordinate System

Computer Graphics, 2008 Spring

Coordinate System

- Left-handed coordinate
  - Right-handed coordinate is also available



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

```
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 Functions

- 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 Functions

- Matrix multiplication

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

                                pOut = pM1 X pM2
```

Transformation and Matrix Manipulation

Computer Graphics, 2008 Spring

Matrix Multiplication Order

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

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

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

Using Matrices

- World transformation
  - Model Space → World Space

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

- Viewing transformation
  - World Space → View Space

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

- Projection transformation
  - View Space → Projection Space

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

$$\begin{pmatrix} x' & y' & z' & w' \end{pmatrix} = \begin{pmatrix} x & y & z & w \end{pmatrix} \cdot M_{world} \cdot M_{view} \cdot M_{projection}$$

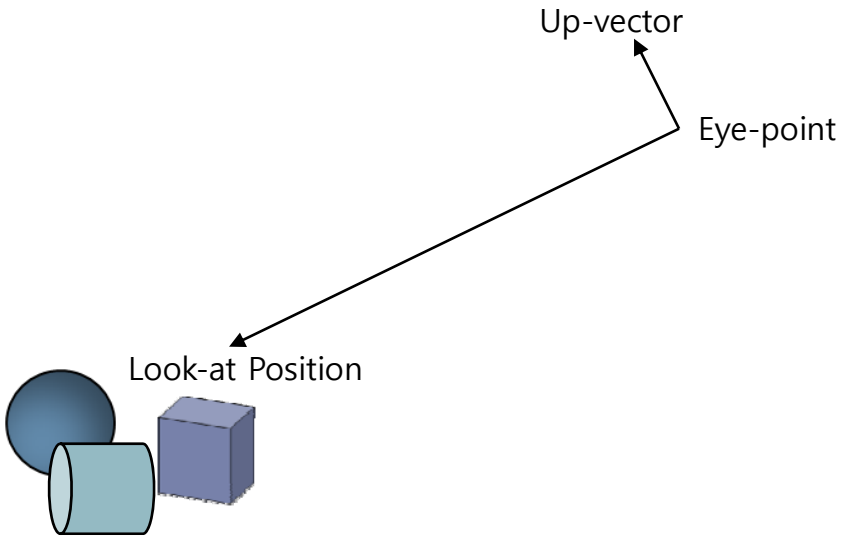
World Transformation

- Scaling, rotation and translation
  - Many pre-defined functions supported by DX
    - D3DXMatrixRotationAxis
    - D3DXMatrixRotationQuaternion
    - D3DXMatrixRotationX
    - D3DXMatrixScaling
    - D3DXMatrixTranslation
    - Et al. – see the SDK document
  - Composite predefined transforms using multiplication
    - D3DXMatrixMultiply

```
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



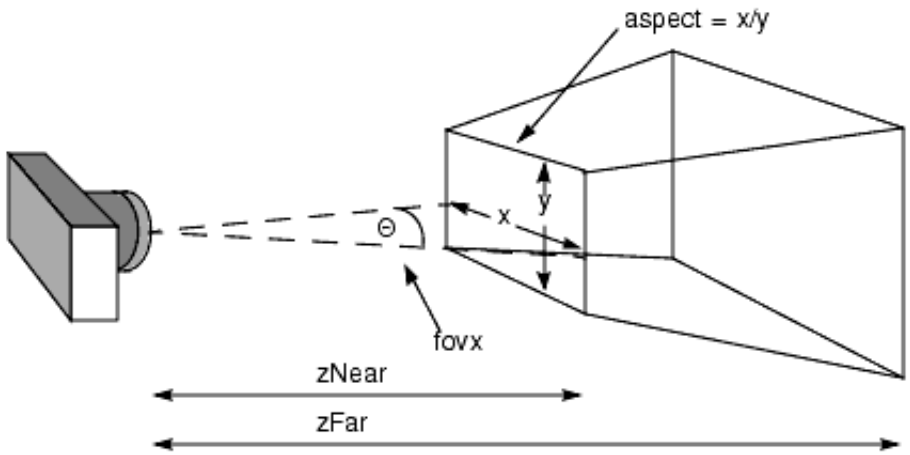

```
D3DXMATRIX matView;
D3DXVECTOR3 vEyePt( 0.0f, 3.0f,-5.0f );
D3DXVECTOR3 vLookatPt( 0.0f, 0.0f, 0.0f );
D3DXVECTOR3 vUpVec( 0.0f, 1.0f, 0.0f );
D3DXMATRIXA16 matView;
D3DXMatrixLookAtLH( &matView, &vEyePt, &vLookatPt, &vUpVec );
g_pd3dDevice->SetTransform( D3DTS_VIEW, &matView );
```

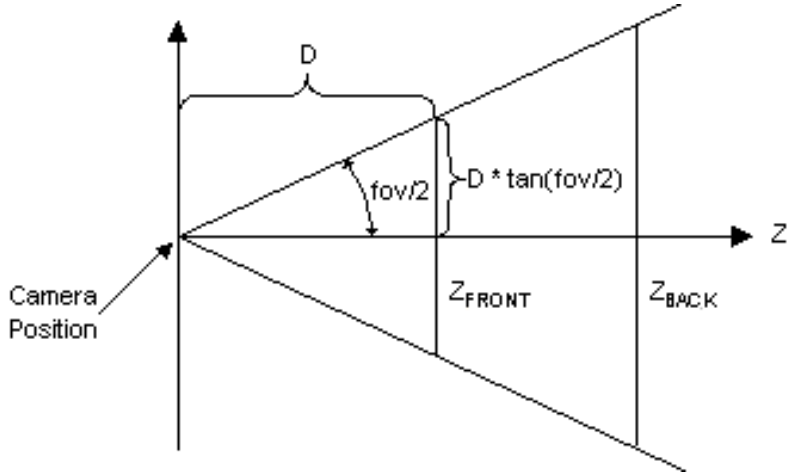
Projection Transformation

- View space → Projection space
  - 3D view space → 2D viewport space
- Perspective projection
  - D3DXMatrixPerspectiveLH
  - D3DXMatrixPerspectiveFovLH
  - D3DXMatrixPerspectiveOffCenterLH
- Orthogonal projection
  - D3DXMatrixOrthoLH
  - D3DXMatrixOrthoOffCenterLH

Projection Transform

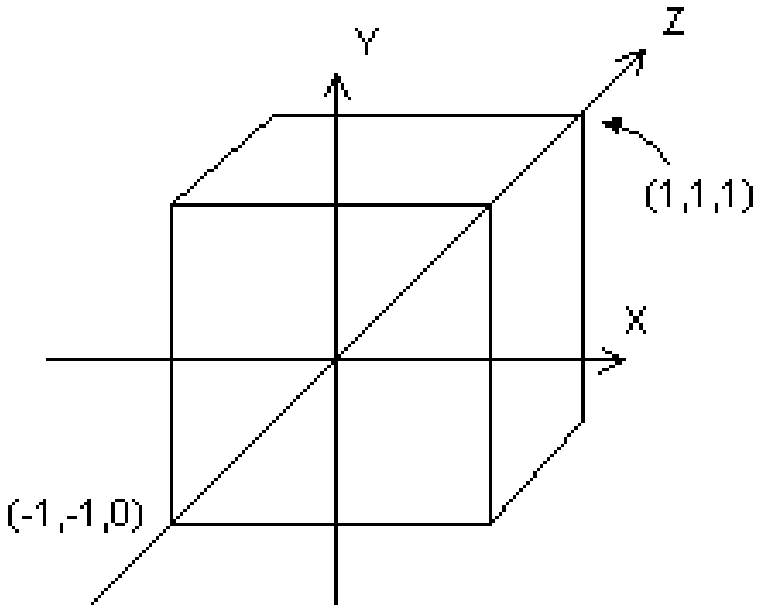
- Transformation parameters





Projection Transform

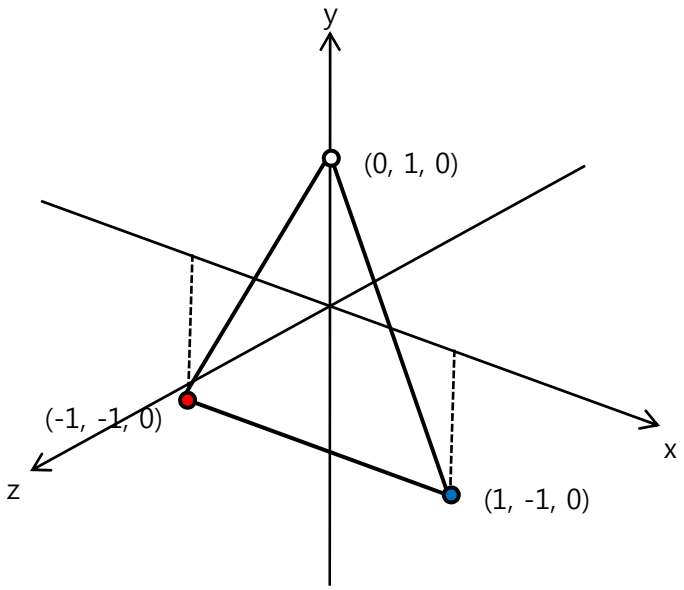
- Projection space
  - X, Y : [-1, 1], viewport
  - Z : [0, 1], depth



Tutorial 3

- Vertices

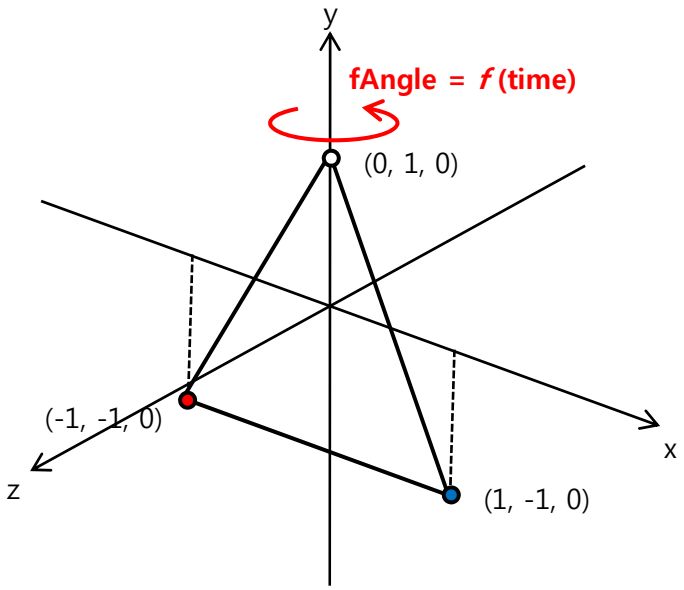
```
CUSTOMVERTEX g_Vertices[] =
  {
     { -1.0f,-1.0f, 0.0f, 0xffff0000, },
     {  1.0f,-1.0f, 0.0f, 0xff0000ff, },
     {  0.0f, 1.0f, 0.0f, 0xffffffff, },
  };
```



Tutorial 3

- World transform

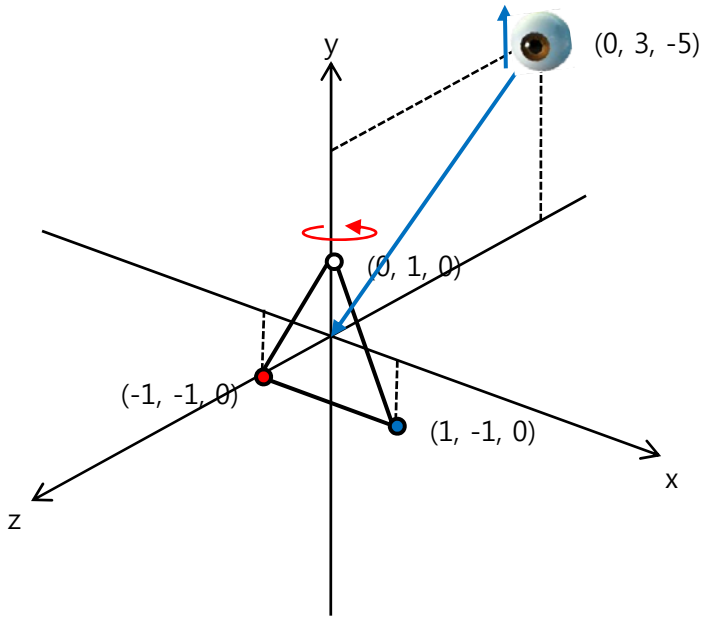
```
UINT  iTime  = timeGetTime() % 1000;
FLOAT fAngle = iTime * (2.0f * D3DX_PI) / 1000.0f;
D3DXMatrixRotationY( &matWorld, fAngle );
g_pd3dDevice->SetTransform( D3DTS_WORLD, &matWorld );
```



Tutorial 3

Viewing transform

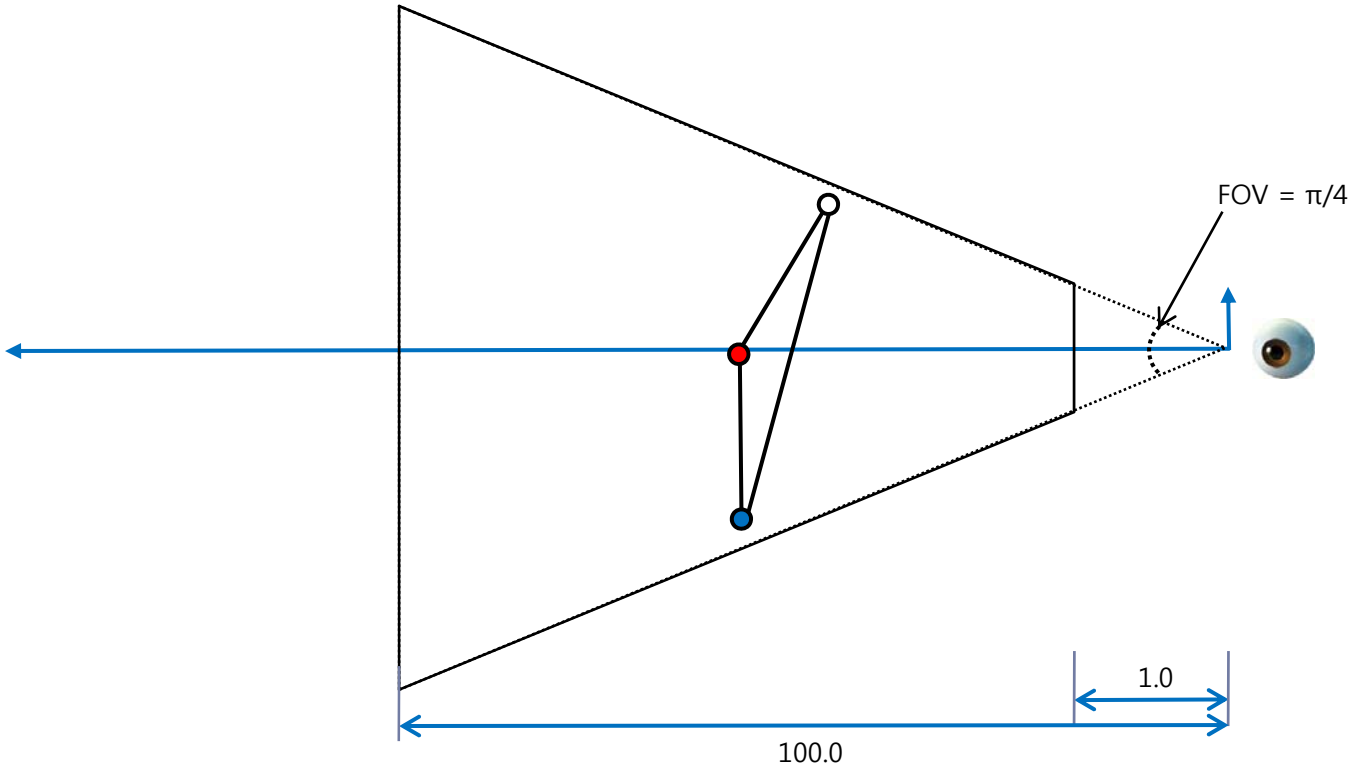
```
D3DXVECTOR3 vEyePt( 0.0f, 3.0f,-5.0f );
D3DXVECTOR3 vLookatPt( 0.0f, 0.0f, 0.0f );
D3DXVECTOR3 vUpVec( 0.0f, 1.0f, 0.0f );
D3DXMATRIXA16 matView;
D3DXMatrixLookAtLH( &matView, &vEyePt, &vLookatPt, &vUpVec );
g_pd3dDevice->SetTransform( D3DTS_VIEW, &matView );
```



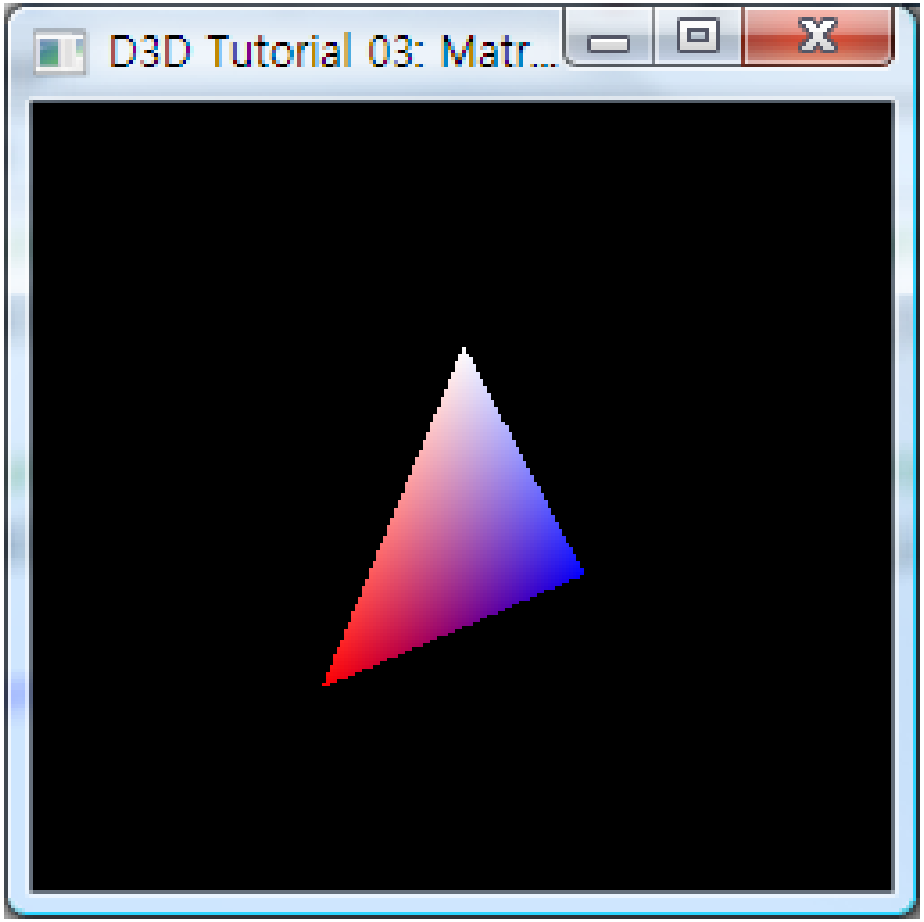
Tutorial 3

- Projection transform

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



Result of Tutorial 3



Any Question?