

# Rendering Pipeline

---

305890  
2008년 봄학기  
3/26/2008  
박경신

## Rendering Pipeline

---

- 모델 표현
- 가상 카메라 (virtual camera)
- 렌더링 파이프라인 (rendering pipeline)
- 예제

## Modeling

---

- 장면(scene)은 물체나 모델의 모음
- 물체(object)는 triangle mesh의 묘사로 이루어짐
- Triangle은 세 개의 정점 (vertex)로 정의됨
- Direct3D에서의 모델 표현
  - Vertex format
  - Triangle
  - Index

## Vertex Format

---

- 정점 (Vertex)
  - 공간적 위치
  - 부가적 특성: normal, color, texture 좌표
- 1단계: Custom vertex format 정의
  - 위치, 색상

```
struct ColorVertex {  
    float _x, _y, _z;           // position  
    DWORD _color;  
};
```
  - 위치, 법선, 텍스처 좌표

```
struct NormalTexVertex {  
    float _x, _y, _z;           // position  
    float _nx, _ny, _nz;       // normal vector  
    float _u, _v;              // texture coords  
};
```

## Vertex Format

### □ 2단계: Vertex formatting 방법을 지정

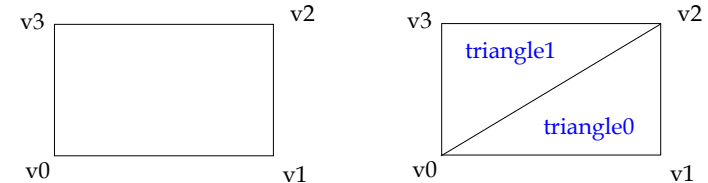
- FVF (Flexible vertex format) flag 조합을 이용
- Struct에 정의된 데이터의 순서와 FVF의 정의된 순서가 반드시 일치해야 함

```
#define FVF_COLOR(D3DFVF_XYZ | D3DFVF_DIFFUSE)
#define FVF_NORMAL_TEX(D3DFVF_XYZ | D3DFVF_NORMAL
| D3DFVF_TEX1)
```

## Triangle

### □ Triangle

- 3D 물체의 기본 구성 요소
- 예를 들어, 사각형은 두 개의 삼각형으로 표현할 수 있다.



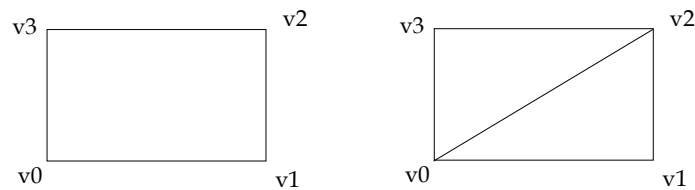
```
Vertex rect[6] = { v0, v1, v2, // triangle 0
                  v0, v2, v3 }; // triangle 1
```

- Vertex의 나열 순서(winding order)를 꼭 지켜야 함

## Index

### □ Index의 필요성

- 많은 vertex들이 중복으로 사용됨
- 따라서 vertex list와 index list를 구성해서
  - Vertex list는 모든 vertex들
  - Index list는 vertex list로의 index값들

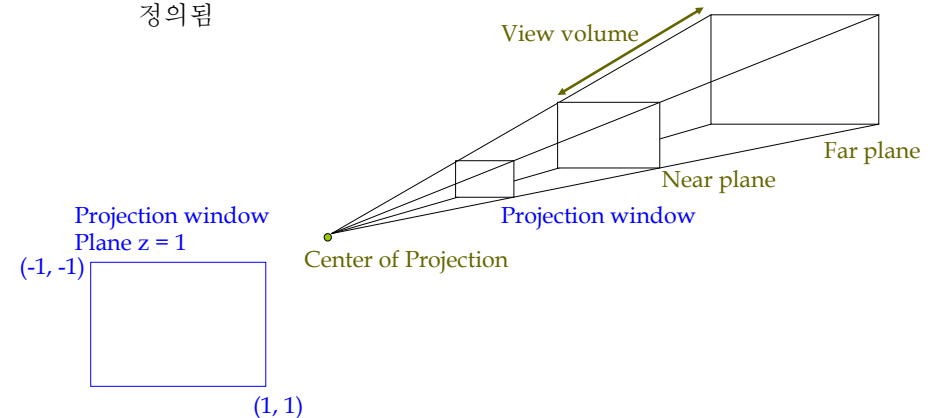


```
Vertex vertexList[4] = { v0, v1, v2, v3 }; // vertex list
WORD indexLIST[6] = { 0, 1, 2, // index list
                    0, 2, 3 }; // index list
```

## Virtual Camera

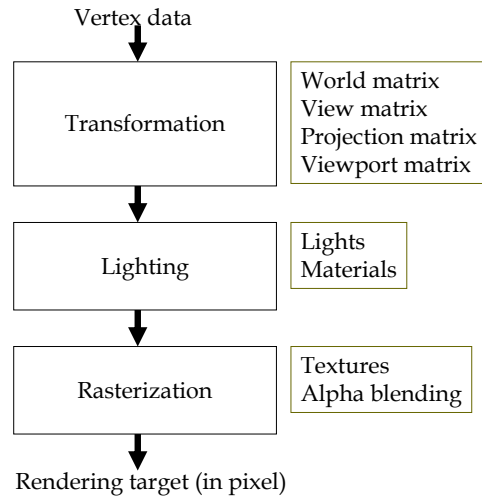
### □ 가상 카메라

- 관찰자가 볼 수 있는 세계의 부분을 결정함
- Projection window: Direct3D에서는 plane z=1과 일치하도록 정의됨



## Rendering Pipeline

- 3차원 장면으로부터 2D 이미지를 만드는 과정

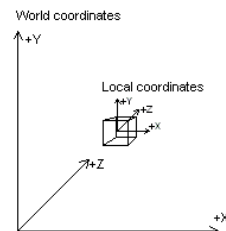


## Rendering Pipeline

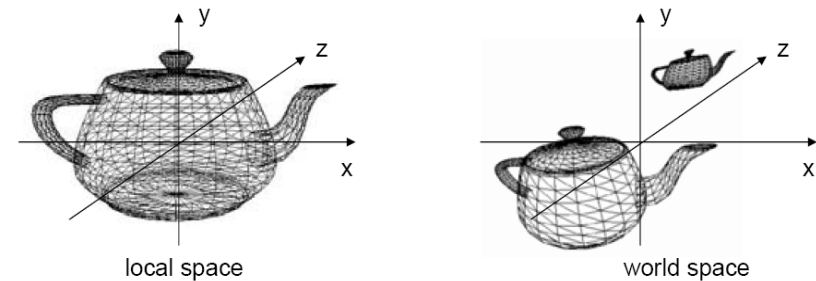
- Rendering pipeline
  - Local space
  - World space
  - View space
  - Backface culling
  - Lighting
  - Clipping
  - Projection
  - Viewport
  - Rasterize
- 이 변환 과정은 Direct3D가 책임 수행함
  - 해당 matrix를 구성하고 이를 IDirect3DDevice->SetTransform을 이용해 matrix를 Direct3D로 전달함

## Local Space & World Space

- Local space
  - 사용자가 물체를 정의하는 데 이용되는 물체 중심적 좌표계 (Modeling space)
  - 모델 구성의 편리성에 중점을 둠. World에서의 물체의 위치나 크기, world 내의 다른 물체와의 관계 등을 고려하지 않고도 모델을 구성할 수 있음.
- World space
  - 하나의 장면에 대한 좌표계로 여러 모델들을 포함함.
  - 각 물체의 관계를 정의함.



## Local Space & World Space



## Modeling Transformation

### Local space에서 World space로 변환

- 하나의 행렬로 표현.
- IDirect3DDevice::SetTransform(D3DTS\_WORLD, &worldMatrix);

```
// cube를 (-3, 2, 6)에 배치하고, sphere를 (5, 0, -2)에 배치하라.
D3DXMATRIX cubeWorldMatrix;
D3DXMatrixTranslation(&cubeWorldMatrix, -3.0, 2.0, 6.0);
D3DXMATRIX sphereWorldMatrix;
D3DXMatrixTranslation(&sphereWorldMatrix, 5.0, 0.0, -2.0);
// set transform for cube
Device->SetTransform(D3DTS_WORLD, &cubeWorldMatrix);
drawCube();
// set transform for sphere
// 이전 world matrix에 적용됨
Device->SetTransform(D3DTS_WORLD, &sphereWorldMatrix);
drawSphere();
```

## View Space

### 기하 물체와 카메라는 world space에서 정의됨

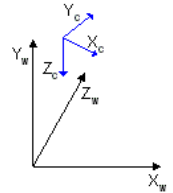
- Projection 등의 작업의 효율성을 위해서 view space로 변환함

### View space 변환

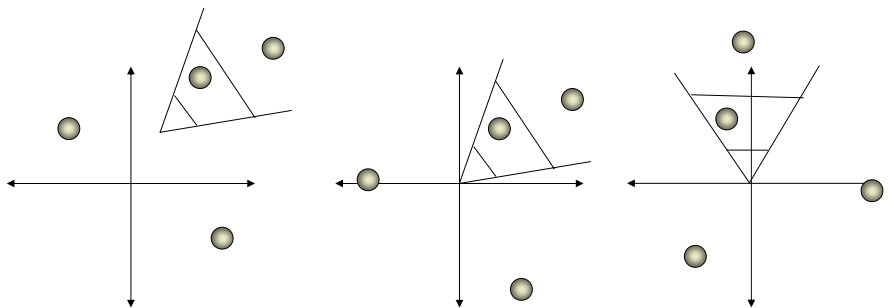
- 카메라를 world space의 원점으로 이동하고 +z축을 보도록 회전시킴.
- 다른 모든 기하 물체들도 동일하게 변환되어야 함.

### World space에서 view space 변환 행렬 계산

```
D3DXMATRIX LookAtLH (
D3DXMATRIX* pOut,
CONST D3DXVECTOR3* pEye, // camera position
CONST D3DXVECTOR3* pAt, // camera look-at position
CONST D3DXVECTOR3* pUp // world up 보통 y-축과 일치함 (0, 1, 0)
)
```



## View Space



World Space 내에서의 물체와 카메라

관찰점을 원점으로 변환.

Z-축에 맞게 관찰점을 회전. 물체들도 함께 회전.

## Viewing Transformation

### World space에서 View space로 변환

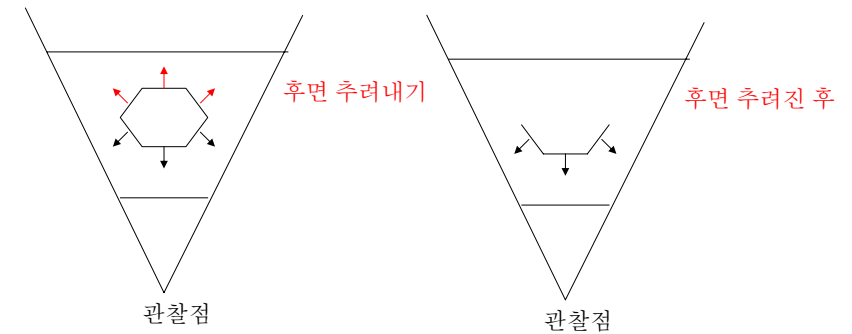
- IDirect3DDevice::SetTransform(D3DTS\_VIEW, &viewMatrix);

```
// camera를 (5, 3, -10)에 위치하고, world의 중앙 (0, 0, 0)을 바라보도록 함.
// set camera
D3DXVECTOR3 position(5.0, 3.0, -10.0);
D3DXVECTOR3 lookout(0.0, 0.0, 0.0);
D3DXVECTOR3 worldup(0.0, 1.0, 0.0);
// set view matrix
D3DMATRIX viewMatrix;
D3DXMatrixLookAtLH(&viewMatrix, &position, &lookat, &worldup);
Device->SetTransform(D3DTS_VIEW, &viewMatrix);
```

## Backface culling

- Backface culling
  - 카메라에 후면을 향하고 있는 polygon은 화면에 그리지 않음.
  - 후면 polygon을 미리 추려내면 이후의 계산에 상당한 이득이 있음.
- 후면 polygon의 결정
  - View space에서의 winding order가 시계(CW) 혹은 반시계(CCW) 방향일 경우
    - 전면/후면 polygon
    - Visibility test:  $\text{planeNormal} \cdot \text{viewVector} > 0$
- Culling 동작을 변경하기 원할 때
  - Device->setRenderState(D3DRS\_CULLMODE, Value);
  - Value
    - D3DCULL\_NONE: backface culling을 사용 안함
    - D3DCULL\_CW: 시계 방향 winding order를 가진 triangle들을 culling함
    - D3DCULL\_CCW: 반시계 방향 winding order를 가진 triangle들을 culling함 (default)

## Backface culling

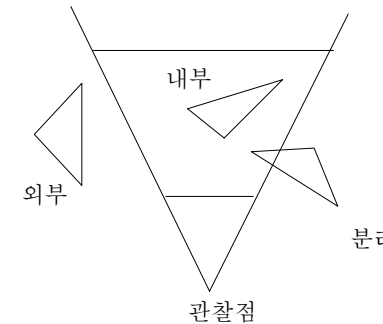


## Lighting

- Lighting
  - 물체에 명암을 추가하여 장면에 사실감을 더해줌

## Clipping

- Clipping
  - 시야 volume 외부의 기하 물체를 추려냄
  - Frustum에서의 triangle의 위치 분류
    - 완전한 내부: 보존
    - 완전한 외부: 추려냄
    - 부분적 내부: triangle을 두 부분으로 분리하여 내부의 부분만 보존



## Projection

### 투영 (Projection)

- View space에서의 3차원 장면의 2차원 표현을 얻음
- 원근 투영 (Perspective projection)은 원근법을 이용하여 기하 물체를 투사

### Projection matrix의 생성

```
D3DXMATRIX *D3DXMatrixPerspectiveFovLH(
    D3DXMATRIX *pOut,
    FLOAT fovY, // field of view in y-axis (in radian)
    FLOAT Aspect, // aspect ratio (= screen width/screen height)
    FLOAT zn, // z-value of near plane
    FLOAT zf // z-value of far plane
)
```

Aspect ratio는 projection window(정사각형)을 screen window space(직사각형)으로 만드는 과정에서 왜곡을 보정하는 역할

## Perspective Projection

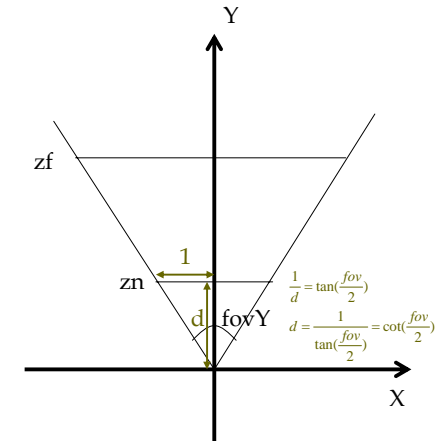
### Projection plane in front of the center of projection

$$\begin{pmatrix} xScale & 0 & 0 & 0 \\ 0 & yScale & 0 & 0 \\ 0 & 0 & \frac{zf}{zf - zn} & 1 \\ 0 & 0 & \frac{-zn * zf}{zf - zn} & 0 \end{pmatrix}$$

where  $yScale = \cot(fovY / 2)$

$xScale = yScale / Aspect$

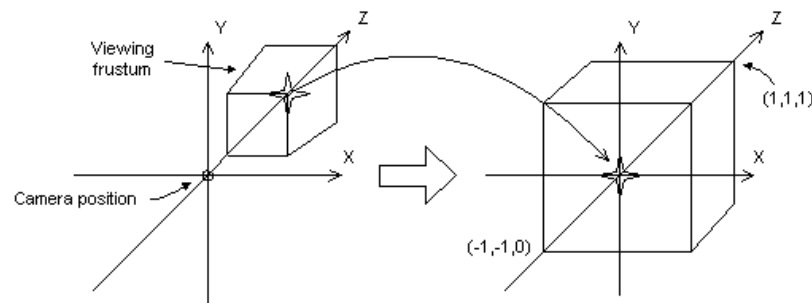
$Aspect = weight / height$



## Perspective Projection

### Direct3D에서의 뷰볼륨 정규화

- $(-x, -y, zn) \rightarrow (-1, -1, 0)$
- $(x, y, zf) \rightarrow (1, 1, 1)$



## Projection Transformation

### 투영변환

- IDirect3DDevice::SetTransform(D3DTS\_PROJECTION, &projMatrix);

// 90 도의 시야각, 거리 1의 가까운 평면,

// 거리 1000의 먼 평면을 가지는 frustum에 맞는 projection matrix.

// set camera

```
D3DXMATRIX projMatrix;
D3DXMatrixPerspectiveFovLH(
```

```
&projMatrix,
```

```
PI*0.5f,
```

```
(float)width / (float)height,
```

```
1.0f, 1000.0f);
```

```
Device->SetTransform(D3DTS_PROJECTION, &projMatrix);
```

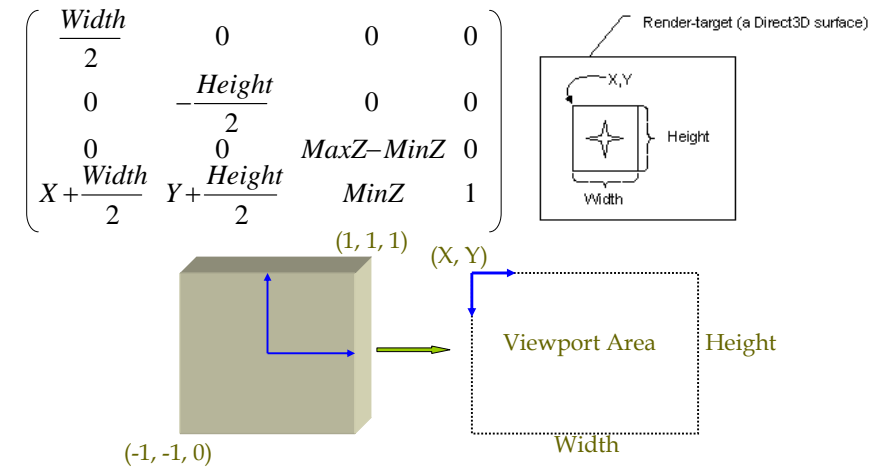
## Viewport Transformation

- 뷰포트변환
  - Projection window를 viewport (on screen)
- Viewport의 표현
 

```
typedef struct _D3DVIEWPORT9 {
    DWORD X;           // pixel coords of the upper-left corner
    DWORD Y;           // pixel coords of the upper-left corner
    DWORD Width;      // width in pixels
    DWORD Height;     // height in pixels
    float MinZ;       // range of depth values
    float MaxZ;       // range of depth values
} D3DViewPORT9;
```
- Viewport matrix 생성
  - `D3DVIEWPORT9 vp = {0, 0, 640, 480, 0, 1};`
  - `Device->SetViewport(&vp);`

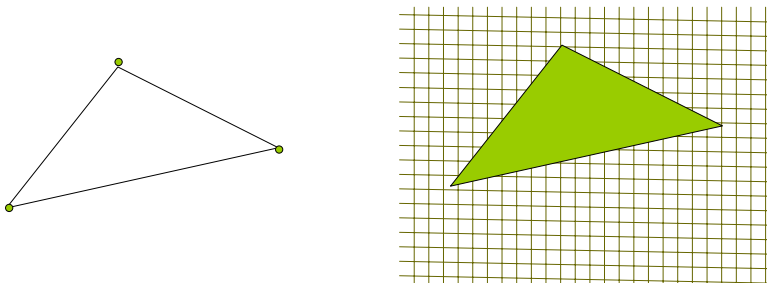
## Viewport

- 내부적으로 수행되는 Viewport행렬의 모습



## Rasterization

- 래스터화 (rasterization)
  - Screen 좌표로 변환된 2D triangle들을 그리기 위한 pixel color값들을 계산
  - 엄청난 작업 양을 필요로 함. 따라서 전용 그래픽스 하드웨어에서 처리됨
  - 결과물은 바로 display될 수 있는 이미지 형태임 [그림 2.17 참고]



## Reference

- Direct3D Transformation Pipeline - <http://msdn2.microsoft.com/en-us/library/bb206260.aspx>