GaLic. Heesung Oh
2009, Copyright All rights reserved

3D Game Programming 09
- Camera

afewhee@gmail.com

0. 목차

2009, Copyright All rights reserved

- 카메라 기초
  - 카메라 클래스
  - 뷰 행렬
  - 투영 행렬
- 1, 3인칭 카메라
  - 전진과 후진
  - 카메라 회전
  - 3인칭 카메라
  - 충돌 검사를 위한 카메라 Frustum
- 카메라 응용
  - 빌보드 효과
  - 다중 카메라(Multi-Camera)
- 실습

1. 카메라 클래스

2009, Copyright All rights reserved

- 뷰 행렬 설정
  - D3DXMatrixLookAtLH();

- 뷰행렬과 카메라 클래스의 예

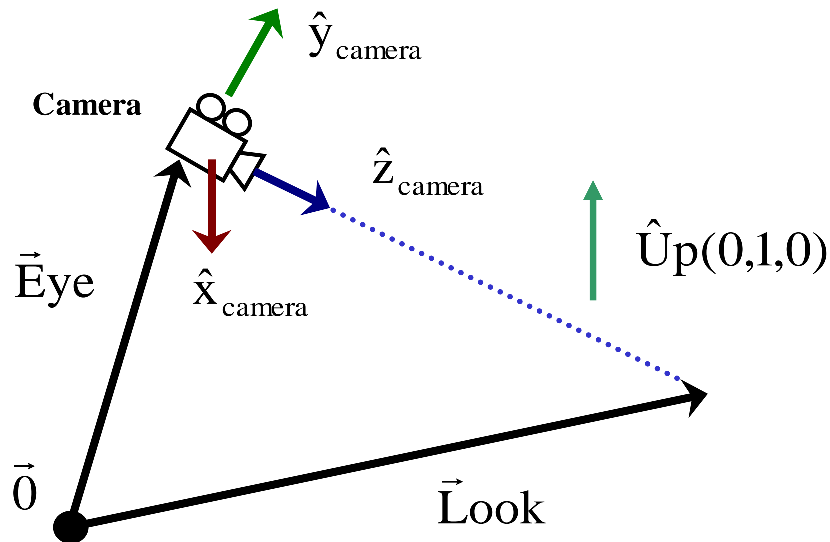
```
class CMcCamera
{
protected:
    LPDIRECT3DDEVICE9       m_pDev;
    D3DXMATRIX              m_mtViw;        // View Matrix
    D3DXMATRIX              m_mtPrj;        // Projection Matrix
    D3DXVECTOR3             m_vcEye;        // Camera position
    D3DXVECTOR3             m_vcLook;       // Look vector
    D3DXMATRIX              m_vcUp;         // up vector

public:
    CMcCamera();
    virtual ~CMcCamera();
    INT  Create(LPDIRECT3DDEVICE9 pDev);
    INT  FrameMove();
};

CMcCamera::CMcCamera()
{
    m_pDev = NULL;
}
...
```

GaLic. Heesung Oh
2009, Copyright All rights reserved

- 뷰 행렬 설정
  - D3DXMatrixLookAtLH();
- 뷰 행렬 계산



$$\vec{z} = \vec{Look} - \vec{Eye}$$

$$\hat{z} = \frac{\vec{z}}{|\vec{z}|}$$

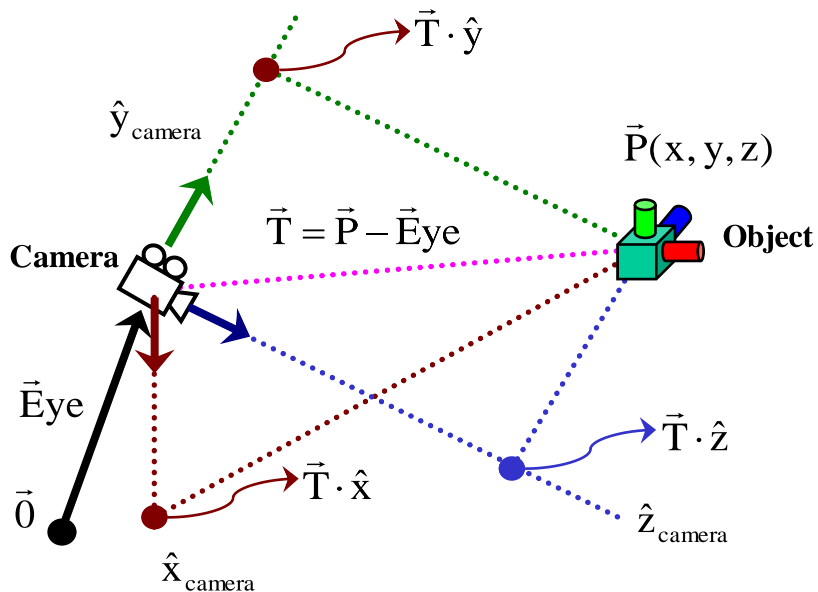
$$\vec{x} = \vec{Up} \times \hat{z}$$

$$\hat{x} = \frac{\vec{x}}{|\vec{x}|}$$

$$\hat{y} = \hat{z} \times \hat{x}$$

GaLic.Heesung Oh
2009, Copyright All rights reserved

뷰 행렬 계산



$$(x',y',z',1)=(x,y,z,1)\begin{pmatrix}\hat{x}_x & \hat{y}_x & \hat{z}_x & 0\\ \hat{x}_y & \hat{y}_y & \hat{z}_y & 0\\ \hat{x}_z & \hat{y}_z & \hat{z}_z & 0\\ 0 & 0 & 0 & 1\end{pmatrix}$$

$$+(x,y,z,1)\begin{pmatrix}0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\\ -\vec{Eye}\cdot\hat{x} & -\vec{Eye}\cdot\hat{y} & -\vec{Eye}\cdot\hat{z} & 1\end{pmatrix}$$

$$(x',y',z',1)=(x,y,z,1)\begin{pmatrix}\hat{x}_x & \hat{y}_x & \hat{z}_x & 0\\ \hat{x}_y & \hat{y}_y & \hat{z}_y & 0\\ \hat{x}_z & \hat{y}_z & \hat{z}_z & 0\\ -\vec{Eye}\cdot\hat{x} & -\vec{Eye}\cdot\hat{y} & -\vec{Eye}\cdot\hat{z} & 1\end{pmatrix}$$

$$M_{view}=\begin{pmatrix}\hat{x}_x & \hat{y}_x & \hat{z}_x & 0\\ \hat{x}_y & \hat{y}_y & \hat{z}_y & 0\\ \hat{x}_z & \hat{y}_z & \hat{z}_z & 0\\ -\vec{Eye}\cdot\hat{x} & -\vec{Eye}\cdot\hat{y} & -\vec{Eye}\cdot\hat{z} & 1\end{pmatrix}$$

$$ViewMatrix^{-1}=\begin{pmatrix}\hat{x}_x & \hat{x}_y & \hat{x}_z & 0\\ \hat{y}_x & \hat{y}_y & \hat{y}_z & 0\\ \hat{z}_x & \hat{z}_y & \hat{z}_z & 0\\ \vec{Eye}_x & \vec{Eye}_y & \vec{Eye}_z & 1\end{pmatrix}$$

1. 카메라 클래스

GaLic.Heesung Oh
2009, Copyright All rights reserved

- 뷰 행렬 계산

- 카메라의 X, Y, Z축을 구한다.

```
Zaxis  = 카메라가 보고 있는 지점 위치(Look) – 카메라의 위치(Eye);
Zaxis = normalize(Zaxis);
Xaxis = Cross(Up, Zaxis);
Xaxis = normalize(Xaxis);
Yaxis = Cross( Zaxis, Xaxis);
```

- 뷰 행렬의 _41, _42, _43 값을 정한다.

```
_41 = -dot(Xaxis, eye)
_42 = -dot(Yaxis, eye)
_43 = -dot(Zaxis, eye)
```

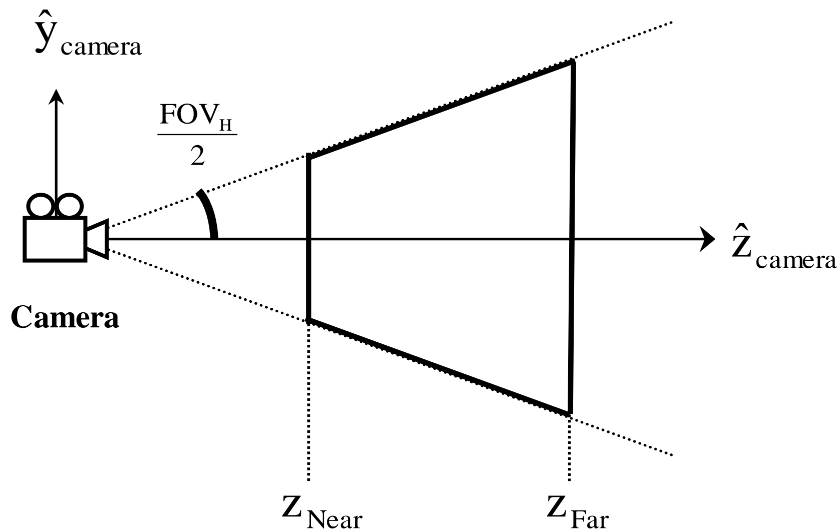
- 뷰 행렬을 설정한다.

Xaxis.x	Yaxis.x	Zaxis.x	0
Xaxis.y	Yaxis.y	Zaxis.y	0
Xaxis.z	Yaxis.z	Zaxis.z	0
-dot(Xaxis, eye)	-dot(Yaxis, eye)	-dot(Zaxis, eye)	1

GaLic. Heesung Oh
2009, Copyright All rights reserved

투영 행렬 계산

카메라의 FOV



$$\begin{pmatrix} w & 0 & 0 & 0 \\ 0 & h & 0 & 0 \\ 0 & 0 & Q & 1 \\ 0 & 0 & -Q*Z_{Near} & 0 \end{pmatrix}$$

$$, \ Q = \frac{Z_{Far}}{Z_{Far} - Z_{Near}}, \ h = \cot\left(\frac{FOV_H}{2}\right), \ w = \frac{h}{Aspect}$$

GaLic. Heesung Oh
2009, Copyright All rights reserved

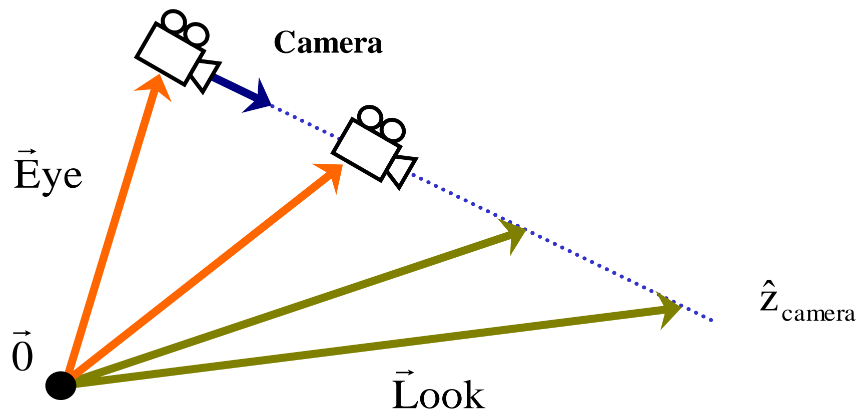
- **투영 행렬 계산**

```
float Near_Plane;
float Far_Plane;
float FOV; //Field of View
float Aspect = ScreenWidth/ScreenHeight;
float h, w, Q;
h = cot(FOV/2.f);
w = h/ Aspect;
Q = Far_Plane /( Far_Plane - Near_Plane);
D3DXMATRIX ProjectionMatrix(
                        w, 0,    0,    0,
                        0, h,    0,    0,
                        0, 0,    Q,    1,
                        0, 0, -zn*Q, 0);
```

GaLic. Heesung Oh
2009, Copyright All rights reserved

● 전진과 후진

카메라의 전진



카메라의 위치: Eye Vector
카메라가 보고 있는 지점: Look Vector
카메라의 이동 스피드: Speed

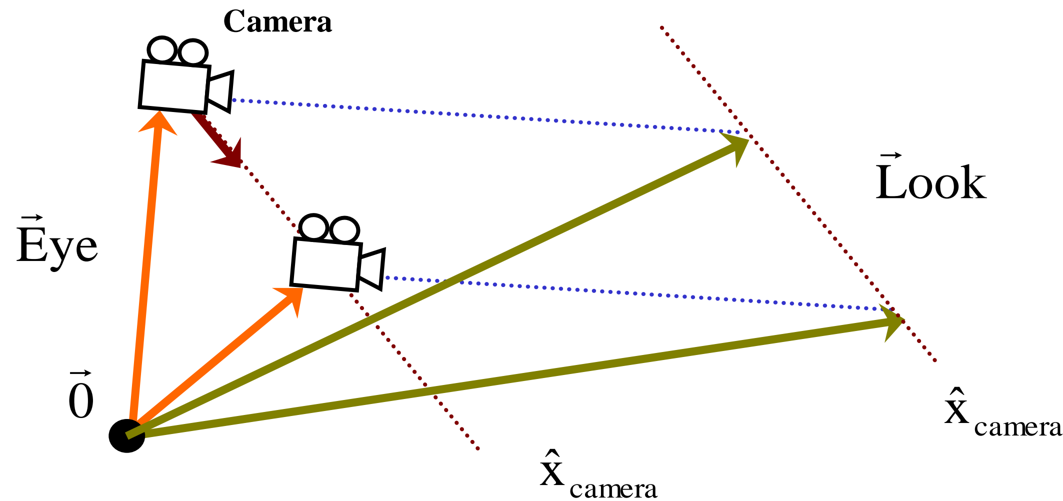
```
Float3 vcZ = Look – Eye;
Normalize(vcZ);
Eye  += vcZ * Speed;
Look += vcZ * Speed;
```

뷰 행렬을 이용할 경우 _13, _23, _33 은 카메라의 z 축에 해당

```
Float3 vcZ(mtView._13, mtView._23, mtView._33);
Eye  += vcZ * Speed;
Look += vcZ * Speed;
```

GaLic. Heesung Oh
2009, Copyright All rights reserved

- 측면이동: 카메라의 X축에 대한 평행 이동




```
Float3 vcX(mtView._11, 0, mtView._31);
Normalize(vcX);
Eye  += vcX * Speed;
Look += vcX * Speed;
```

2009, Copyright All rights reserved

- 회전
  - 1인칭 카메라: 카메라의 Eye를 중심으로 회전
  - 3인칭 카메라: 카메라의 Look 위치(캐릭터의 중심 위치)를 중심으로 회전
  - Gimbal Lock 문제 주의 ➔ 해결책: 회전을 누적
  - 회전을 누적하면 오차 발생 가능 ➔ 특정한 각도에서 초기화

2. 1, 3인칭 카메라

GaLic.Heesung Oh
2009, Copyright All rights reserved

- 1인칭 카메라 회전 구하기
  - Eye를 기준으로 Look을 구함

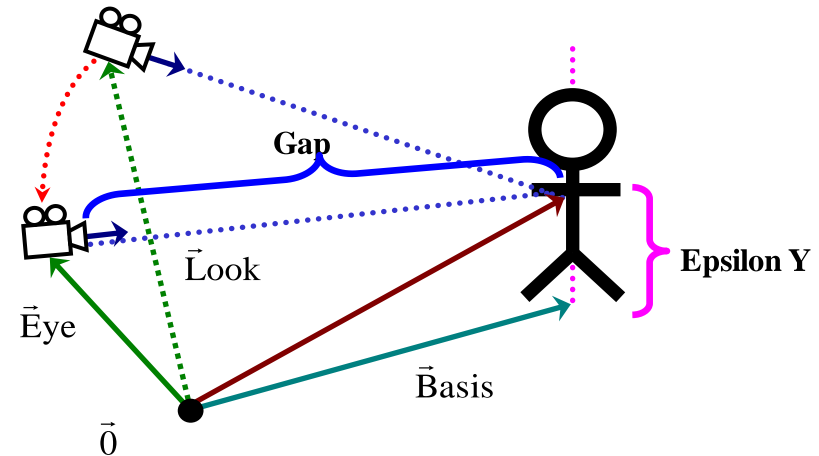
- 카메라의 X축으로 회전할 때 프로그램 순서
  - 1. 카메라에 대한 x, y축, z축 벡터를 뷰 행렬에서 얻는다.
    - vcX = Float3(View._11, View._21, View._31);
    - vcY = Float3(View._12, View._22, View._32);
    - vcZ = Float3(View._13, View._23, View._33);
  - 2. 카메라의 x축에 대한 회전 행렬을 구한다.
    - Pitch = D3DXToRadian(Angle * Speed);
    - rtX: MatrixRotationAxis(&rtX, &vcX, Pitch);
  - 3. 카메라의 y, z 축을 회전 행렬을 이용해서 회전 시킨다.
    - vcZ : D3DXVec3TransformCoord(&vcZ, &vcZ, &rtX);
    - vcY : D3DXVec3TransformCoord(&vcY, &vcY, &rtX);
  - 4. Look 벡터와 Up 벡터를 다시 계산한다.
    - Look = vcZ + Eye;
    - Up = vcY;
  - 5. D3DXMatrixLookAtLH() 함수를 이용해서 뷰 행렬을 완성한다.
    - D3DXMatrixLookAtLH(View, Eye, Look, Up);

GaLic.Heesung Oh
2009, Copyright All rights reserved

- 카메라의 Y축으로 회전할 때 프로그램 순서
  - 1. 카메라에 대한 x, y축, z축 벡터를 뷰 행렬에서 얻는다.
    - vcX = Float3(View._11, View._21, View._31);
    - vcY = Float3(View._12, View._22, View._32);
    - vcZ = Float3(View._13, View._23, View._33);
  - 2. 카메라의 y축에 대한 회전 행렬을 구한다.
    - Yaw = D3DXToRadian(Angle * Speed);
    - rtY: MatrixRotationAxis(&rtY, &vcY, Yaw);
  - 3. 카메라의 x, z 축을 회전 행렬을 이용해서 회전 시킨다.
    - vcX : D3DXVec3TransformCoord(&vcX, &vcX, & rtY);
    - vcZ : D3DXVec3TransformCoord(&vcZ, &vcZ, & rtY);
  - 4. Look 벡터와 Up 벡터를 다시 계산한다.
    - Look = vcZ + Eye;
    - Up = vcY;
  - 5. D3DXMatrixLookAtLH() 함수를 이용해서 뷰 행렬을 완성한다.
    - D3DXMatrixLookAtLH(View, Eye, Look, Up);

GaLic.Heesung Oh
2009, Copyright All rights reserved

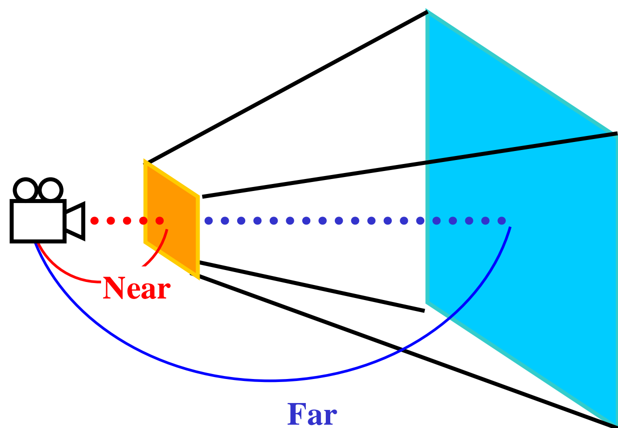
- 3인칭 카메라 회전 구하기
  - Look을 기준으로 Eye를 구함
  - 1인칭과 비슷
  - Look에서의 거리 Gap이 필요
  - 캐릭터를 사용할 경우 캐릭터의 중심 좌표에서 Y축으로 Epsilon 값 필요



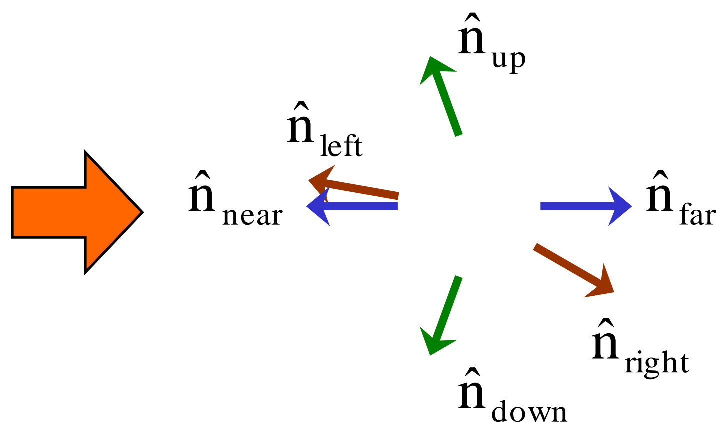
- 프로그램 순서
  - 마우스의 상대 좌표 값을 읽는다.
  - 카메라에 대한 x, y축 벡터는 뷰 행렬에서 얻고 Eye – Look 벡터를 구한다.
    - ...
    - vcZ = Eye - Look;
  - 이 마우스의 x축에 대한 상대적인 값을 라디안 값으로 변경한 후 카메라의 x에 대한 회전 행렬을 구한다.
  - 4. 카메라의 y, z 축을 3의 행렬을 이용해서 회전 시킨다.
  - 5. Look 벡터와 Up 벡터를 다시 계산한다.
    - Eye = vcZ + Look;
    - Up = vcY;
  - 6. D3DXMatrixLookAtLH() 함수를 이용해서 뷰 행렬을 완성한다.
    - D3DXMatrixLookAtLH(View, Eye, Look, Up);

GaLic.Heesung Oh
2009, Copyright All rights reserved

● 카메라 Frustum



Viewing Frustum
Normal Vector



$(-1, -1, 0)^{-1}$, $(-1, 1, 0)^{-1}$, $(1, 1, 0)^{-1}$, $(1, -1, 0)^{-1}$ 네 점이 뷰 행렬 * 투영 행렬의 역 행렬을 통해서 변환된 값이라 하면

- Left 평면은 (-1, 1, 0)′, (-1, -1, 0)′, 카메라 위치(Eye), 세 점으로 구한다.
- Right 평면은 (1, -1, 0)′, (1, 1, 0)′, 카메라 위치(Eye), 세 점으로 구한다.
- Up 평면은 (1, 1, 0)′, (-1, 1, 0)′, 카메라 위치(Eye), 세 점으로 구한다.
- Down 평면은 (-1, -1, 0)′, (1, -1, 0)′, 카메라 위치(Eye), 세 점으로 구한다.

GaLic.Heesung Oh
2009, Copyright All rights reserved

- 카메라 Frustum 프로그램 순서
  - 1. 뷰와 투영 행렬의 곱의 역 행렬을 구한다.
    - Matrix ViewProjInv = View Matrix * Projection Matrix;
    - ViewProjInv = InverseMatrix(ViewProjInv);
  - 2. (-1, -1, 0), (-1, 1, 0), (1, 1, 0), (1, -1, 0)을 변환한다.
    - A = Transform(ViewProjInv, (-1, -1, 0));
    - B = Transform(ViewProjInv, (-1, 1, 0));
    - C = Transform(ViewProjInv, (1, 1, 0));
    - D = Transform(ViewProjInv, (1, -1, 0));
  - 3. Left, Right, Up, Down 평면의 방정식을 각각의 세 점을 이용해서 구한다.
    - Left 평면: PlaneFromPoints(B, A, Eye)
    - Right 평면: PlaneFromPoints(D, C, Eye)
    - Up 평면: PlaneFromPoints(C, B, Eye)
    - Down 평면: PlaneFromPoints(A, D, Eye)

GaLic.Heesung Oh
2009, Copyright All rights reserved

- 빌보드 효과
  - 렌더링 오브젝트가 항상 카메라를 바라보고 있는 것
  - 파티클, 화염효과 같은 이펙트, 간단한 렌더링 오브젝트에 많이 이용
  - 보통 하나의 텍스처와 4개의 정점 이용 ➔D3D POINT LIST를 이용하기도 함

- 구현 방법
  - 정점은 그래픽 파이프 라인의 뷰 변환을 거치므로 뷰 변환 전에 뷰 행렬을 적용하면 렌더링 객체는 뷰 변환을 거치지 않은 것과 같은 효과

- 프로그램 순서
  - 빌보드 행렬 설정: 뷰행렬의 역행렬을 구하고 _41, _42, _43 값을 0으로 주어 렌더링 객체에 회전만 적용하도록 함
  - 정점을 Z=0으로 설정하고 X, Y 값에 너비와 높이 설정
  - 정점을 빌보드 행렬에 변환
  - 렌더링

- Pseudo-Code
  - 1. 카메라의 뷰 행렬의 역행렬을 통해서 _41, _42, _43 변수 값을 0으로 만들어 빌보드 행렬을 얻는다.

```
BillBoardMatrix = Inverse(View Materix);
BillBoardMatrix._41 = 0;
BillBoardMatrix._42 = 0;
BillBoardMatrix._43 = 0;
```

  - 2. 물체의 위치를 월드 좌표의 x 축과 y 축에 평행하도록 설정한다.

```
Vtx[0].position = Float3(-1, -1, 0);
Vtx[1].position = Float3(-1,  1, 0);
Vtx[2].position = Float3( 1,  1, 0);
Vtx[3].position = Float3( 1, -1, 0);
```

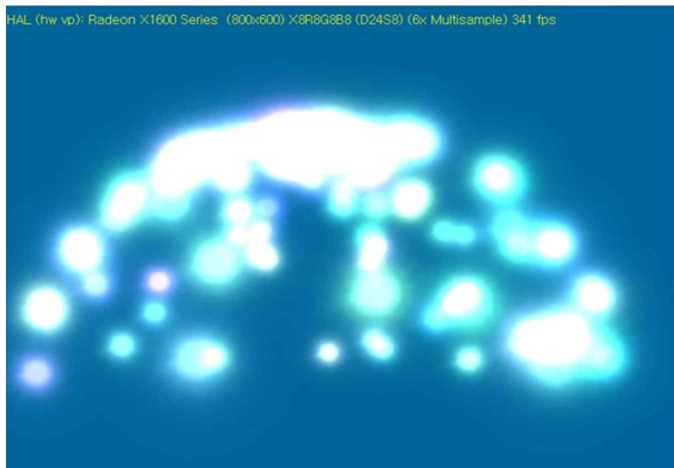
  - 3. 빌보드 행렬을 통해서 이 물체의 위치를 이동 시킨다.

```
Vtx[0].p += 물체 위치;
Vtx[1].p += 물체 위치;
Vtx[2].p += 물체 위치;
Vtx[3].p += 물체 위치;프로그램 예
```

3. Billboard(빌보드 효과)

2009, Copyright All rights reserved

- **파티클**



- **나무**

4. 다중 카메라

2009, Copyright All rights reserved

- 다중 카메라: 렌더링에서 여러 개의 카메라를 두어 장면을 연출함으로써 영화와 같은 영상을 구현
- 예: 자동자의 Room Mirror, 그룹으로 움직이는 캐릭터의 모니터링 효과,
- 렌더링 타켓을 Texture의 Surface로 설정해서 사용

-

4. 다중 카메라

2009, Copyright All rights reserved

- 렌더링 타겟 변경 방법

- 렌더 타겟용 텍스처와 서피스 얻기

```
LPDIRECT3DTEXTURE9 pTexRender;
LPDIRECT3DSURFACE9 pSufRender;
D3DXCreateTexture(                  ...

                                    , USAGE: D3DUSAGE_RENDERTARGET
                                    , D3DFMT_X8R8G8B8I
                                    , D3DPOOL_DEFAULT, ...);

pTexRender->GetSurfaceLevel(0, pSufRender);
...
```

- 디바이스의 렌더 타겟 보존

```
pDevice->GetRenderTarget(0, &pSufColorOld);
pDevice->GetDepthStencilSurface(&SufDepthOld);
```

- 렌더 타겟 변경

```
pDevice->SetRenderTarget(pSufRender);
```

렌더링

```
pDevice->BeginScene();......
```

- 렌더 타겟 환원

```
pDevice->SetRenderTarget(0, pSufColorOld);
pDevice->SetDepthStencilSurface(SufDepthOld);
```

3. 실습

GaLic. Heesung Oh
2009, Copyright All rights reserved

- 카메라 클래스를 작성하되 추상화 하시오
- 다중 카메라를 구현해 보시오