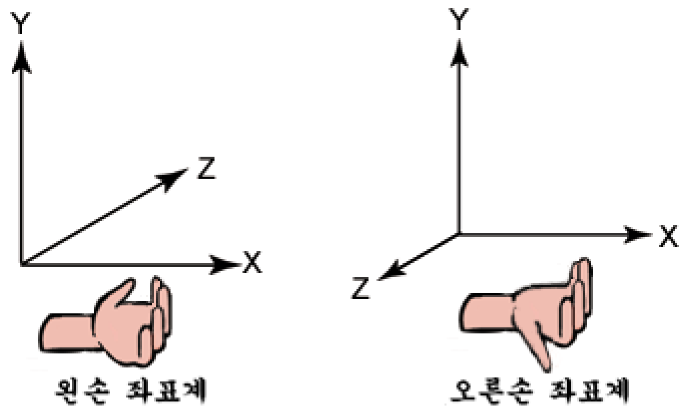
Vector and Matrix

김성익(noerror@hitel.net)
2005.02.20

Spirit of Flame
3D RealTime Graphics Programming Study
Kasa

벡터

- 3D 카테시안 좌표계 *3d cartesian coordinates*
- 직교 좌표계
- x, y, z 스칼라 값 집합



- 왼손 좌표계

Spirit of Flame
3D RealTime Graphics Programming Study
Kasa

벡터 표현

- Direct3d 의 벡터형

  $[\,x \quad y \quad z\,]$

- OpenGL 의 벡터형

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$V_{direct3d} = V_{opengl}{}^{T}$$

- Direct3d vs OpenGL : 전치 *transpose* 된 형태
- 전치 : 행과 열을 바꿈

Spirit of Flame

3D RealTime Graphics Programming Study

Kasa

벡터 의미

- 공간 *space* : 방향 *direction*, 위치 *position*

- *명시적인 표현 제안*

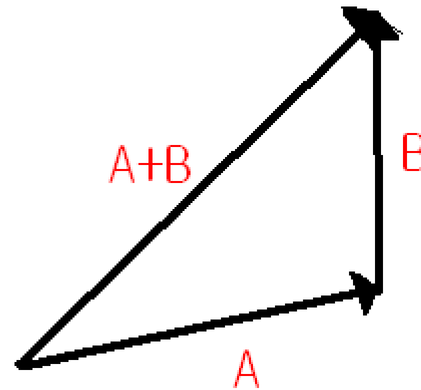
```
struct _vector3 {
    float x, y, z;
} ;

#define _point3 _vector3

float Rayintersect(_vector3 * pos, _vector3 * direction) ;

float Rayintersect(_point3 * pos, _vector3 * direction) ;
```

- affine

벡터 합, 길이

- 벡터합 *addition*

```
D3DXVECTOR3 * _D3DXVec3Add(D3DXVECTOR3 *pOut,
    CONST D3DXVECTOR3 *pV1,
    CONST D3DXVECTOR3 *pV2)
{
    pOut->x = pV1->x + pV2->x;
    pOut->y = pV1->y + pV2->y;
    pOut->z = pV1->z + pV2->z;
    return pOut;
}
```



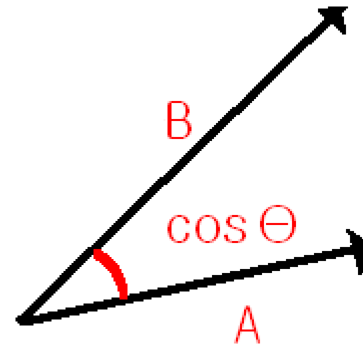
- 벡터길이 *length*

```
FLOAT _D3DXVec3Dot(CONST D3DXVECTOR3 *pV1,
    CONST D3DXVECTOR3 *pV2
)
{
    return pV1->x * pV2->x + ₩
        pV1->y * pV2->y + pV1->z * pV2->z;
}
```

벡터 내적

- 벡터내적 *dotproduct*
- A • B = |A|*|B|*cosΘ

```
FLOAT _D3DXVec3Length(CONST D3DXVECTOR3 *pV)
{
    return (float) sqrt(pV->x * pV->x + ₩
        pV->y * pV->y + pV->z * pV->z);
}
```

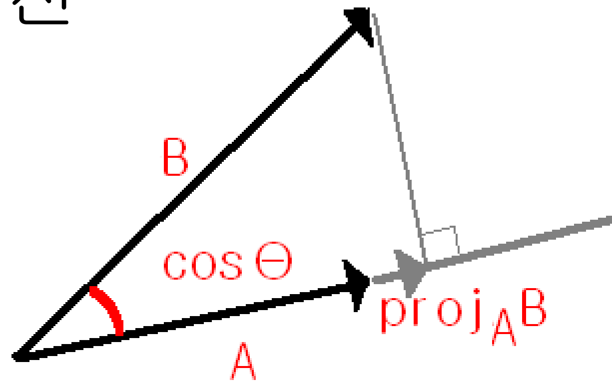


- 내적 > 0 : 90도 이내 (같은 방향)
- 내적 = 0 : 직각
- 단위벡터 *unit vector* 끼리의 내적 : 두 벡터 사이의 cos

Spirit of Flame
3D RealTime Graphics Programming Study
Kasa

벡터 프로젝션

- $A \cdot A = |A|^2$
- 프로젝션

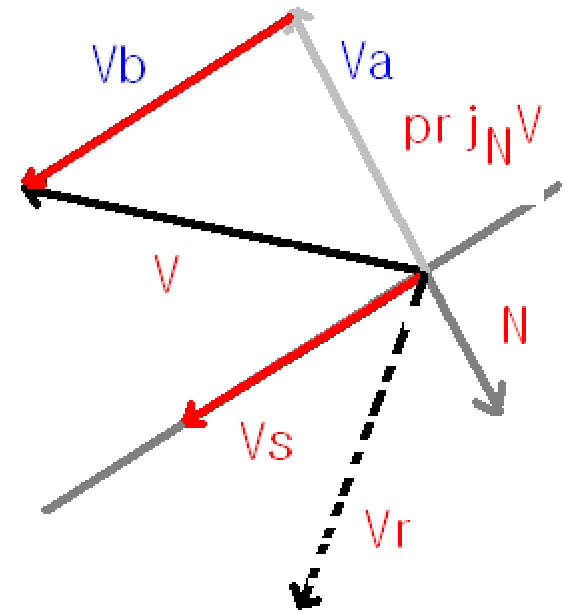


- $proj_A B = (|B| * \cos\Theta) * (A / |A|)$
  $= \{(A \cdot B) / (A \cdot A)\} \cdot A$

Spirit of Flame
3D RealTime Graphics Programming Study
Kasa

슬라이딩 벡터

- 슬라이딩 벡터
- 벡터V를 두 벡터로 분해
- Vs = Vb = V – Va
- Va 는 N 벡터의 투영 벡터



- 반사벡터
- Vr = Vs – Va = V – Va * 2

Spirit of Flame
3D RealTime Graphics Programming Study
Kasa

벡터외적

- 벡터외적 *crossproduct*

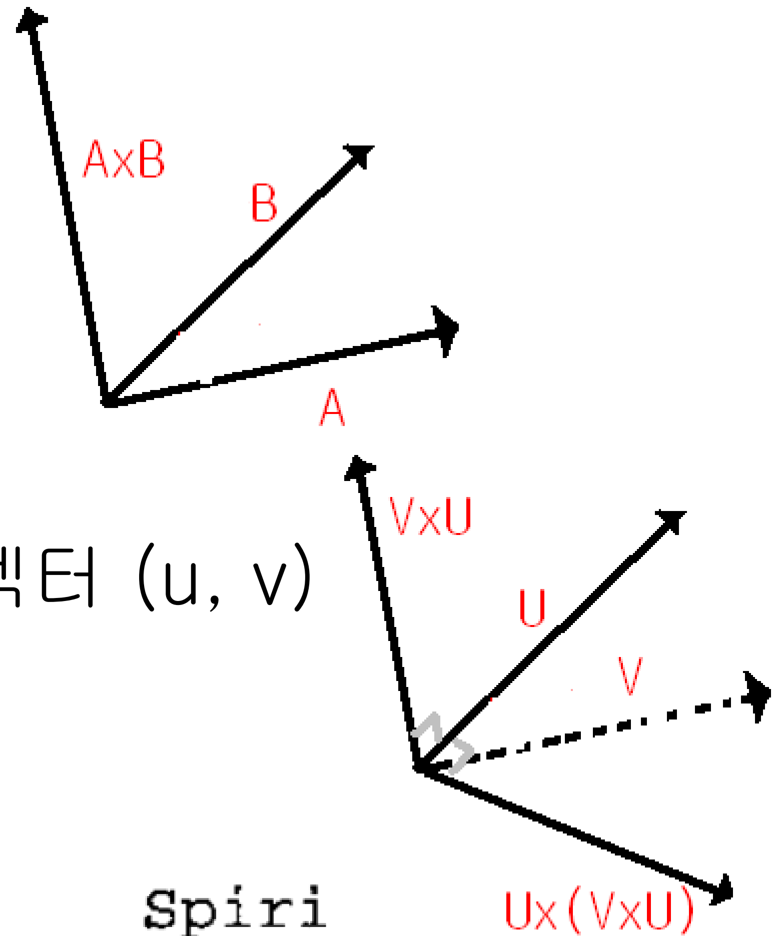
```
D3DXVECTOR3 * _D3DXVec3Cross(D3DXVECTOR3 *pOut,
    CONST D3DXVECTOR3 *pV1,
    CONST D3DXVECTOR3 *pV2)
{
    pOut->x = pV1->y * pV2->z - pV1->z * pV2->y;
    pOut->y = pV1->z * pV2->x - pV1->x * pV2->z;
    pOut->z = pV1->x * pV2->y - pV1->y * pV2->x;
    return pOut;
}
```

- 오른손
- 직교 기저 *orthogonal basis* 벡터 (u, v)

  u

  v x u

  u x (v x u)



Spiri
3D Realtime Graphics Programming Study
Kasa

매트릭스

- 행렬 *matrix*
- 4 x 4 형태의 행렬

$$\begin{bmatrix} M_{11} & M_{12} & M_{13} & M_{14} \\ M_{21} & M_{22} & M_{23} & M_{24} \\ M_{31} & M_{32} & M_{33} & M_{34} \\ M_{41} & M_{42} & M_{43} & M_{44} \end{bmatrix}$$

- 변환 *transform*
  스케일 *scaling*, 회전 *rotation*, 이동 *translation*

변환

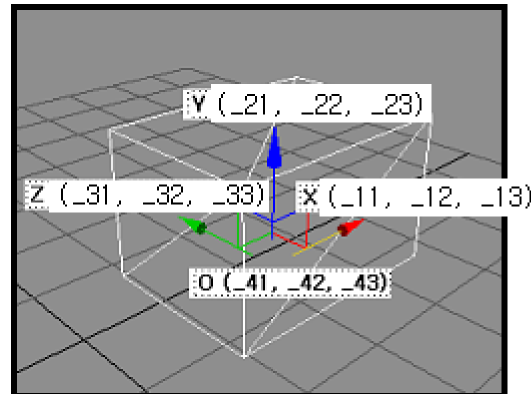
- 동차좌표homogeneous coordinate
  (x / w, y / w, z / w) = (x, y, z, w)
- 벡터 매트릭스 곱 (가로벡터 Direct3D)

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

```
D3DXVECTOR4 * _D3DXVec3Transform(D3DXVECTOR4 *pOut,
    CONST D3DXVECTOR3 *pV,
    CONST D3DXMATRIX *pM)
{
    pOut->x = pV->x * pM->_11 + pV->y * pM->_21 + pV->z * pM->_31 + 1.0f * pM->_41;
    pOut->y = pV->x * pM->_12 + pV->y * pM->_22 + pV->z * pM->_32 + 1.0f * pM->_42;
    pOut->z = pV->x * pM->_13 + pV->y * pM->_23 + pV->z * pM->_33 + 1.0f * pM->_43;
    pOut->w = pV->x * pM->_14 + pV->y * pM->_24 + pV->z * pM->_34 + 1.0f * pM->_44;
    return pOut;
}
```

직각좌표계

- X축 ‘끝점 (1, 0, 0) – 시작점(0, 0, 0)’
- 변환해보면
  끝점 = (_11+_41, _12+42, _13+43)
  시작점 = (_41, _42, _43)
  x축의 방향 벡터 = (_11, _12, _13)
- 행렬의 각 요소는
  축의 좌표

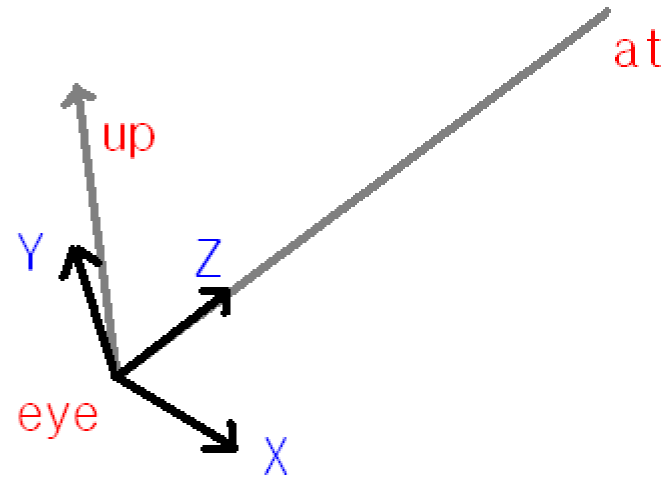


of Flame
e Graphics Programming Study
Kasa

바라보는 매트릭스

- z축은 바라보는 방향
- 외적을 이용한 직교 기저 벡터 공식

```
D3DXMATRIX * _D3DXMatrixLookAtLH(D3DXMATRIX *pOut,
    CONST D3DXVECTOR3 *pEye,
    CONST D3DXVECTOR3 *pAt,
    CONST D3DXVECTOR3 *pUp)
{
    D3DXVECTOR3 u, x, y, z, t;
    D3DXMATRIX m;

    D3DXVec3Subtract(&u, pAt, pEye);
    D3DXVec3Normalize(&z, &u);

    D3DXVec3Cross(&t, pUp, &u);
    D3DXVec3Normalize(&x, &t);

    D3DXVec3Cross(&t, &u, &x);
    D3DXVec3Normalize(&y, &t);

    m._11 = x.x, m._12 = x.y, m._13 = x.z, m._14 = 0.0f;
    m._21 = y.x, m._22 = y.y, m._23 = y.z, m._24 = 0.0f;
    m._31 = z.x, m._32 = z.y, m._33 = z.z, m._34 = 0.0f;

    m._41 = pEye->x;
    m._42 = pEye->y;
    m._43 = pEye->z;
    m._44 = 1.0f;
    return D3DXMatrixInverse(pOut, NULL, &m);;
}
```



매트릭스성질

- 결합 *Concatenation*
- v1 = v • M1, v2 = v1 • M2
  M = M1•M2 , v3 = v • M
  v2 == v3
- (M1•M2)•M3 = M1•(M2•M3)=M1•M2•M3

- 교환 성질
- M1 • M2 != M2 • M1

매트릭스성질

- 단위벡터 : $A \cdot I = A$
- $I = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
- 역행렬 : $A \cdot A^{-1} = I$

  A가 직교*orthogonal* 행렬이면 $A^T = A^{-1}$
- $(AB)^{-1} = B^{-1} A^{-1}$

스케일 매트릭스

- 벡터의 x, y, z 스케일*scaling* 매트릭스

$$\begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- 계층적인 구조에서의 스케일 변환

회전 변환

- X축 회전*rotation* 매트릭스

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

- 다른 도메인의 회전 변환
  오일러 각 *euler angle* : Yaw(y축), Roll(z축), Pitch(x축)
  회전 각도와 회전 축 *axis and angle*
  사원수 *quaternion*

변환

- 이동*translation* 매트릭스

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ Tx & Ty & Tz & 1 \end{bmatrix}$$

- 회전, 스케일 변환에서의 origin

- Rigid Body Transform :
  회전, 이동 변환

참고

- Rick Parent, Computer Animation Algorithms and techniques, Morgan Kaufmann publishers, chapter 2.1 spaces and transformations
- Issac Victor Kerlow, The art of 3-d computer animation and imaging, Wiley, pages 77-89
- Eric Lengyel, Mathematics fo 3d game programming & computer graphics, charles river media, chapter 1, 2, 3
- Gareth Williams, Linear algebra with applications
- Math in Direct3DX, http://mormegil.wz.cz/prog/3dref/D3DXmath_en.htm