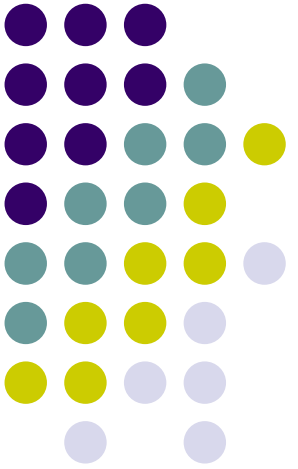
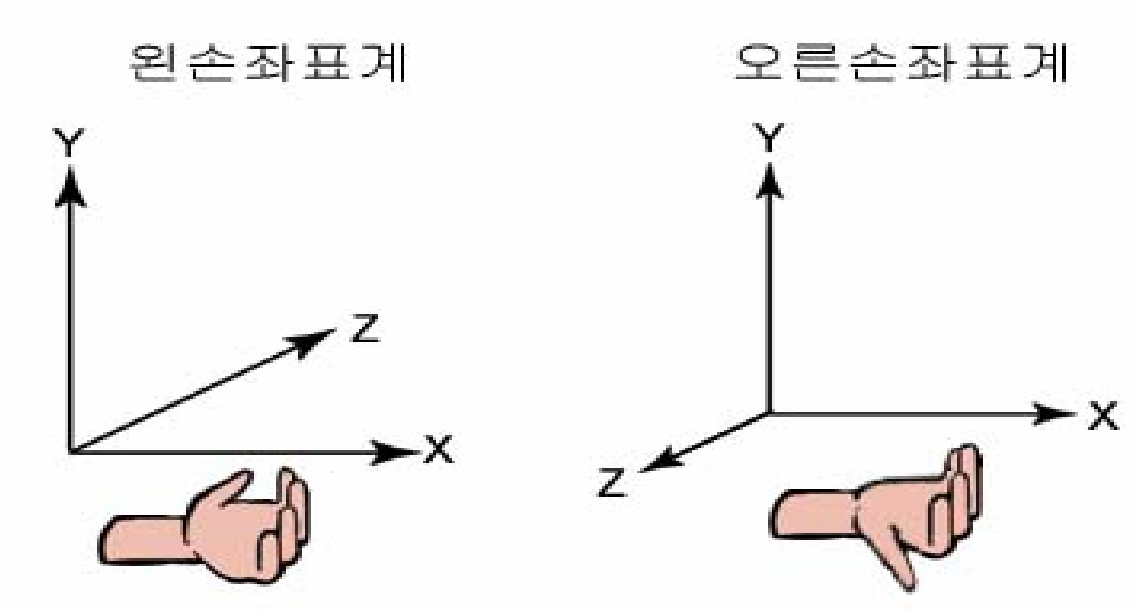
기초 이론



3차원 공간

- 왼손 좌표계와 오른손 좌표계



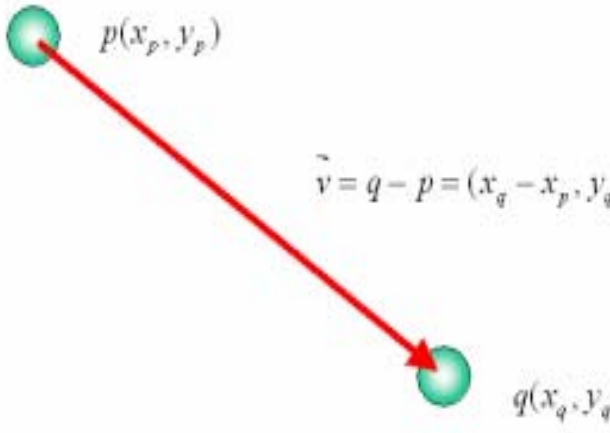
D3D　　OpenGL

벡터

- 벡터의 길이와 단위 벡터
  - 벡터의 크기가 1인 벡터를 단위벡터라고 하고, 임의의 벡터를 단위벡터를 만드는 것을 정규화라고 한다.
  - 단위 벡터의 용도는 일반적으로 회전의 중심축으로 사용되는 경우가 많다. 그 외에도 삼각형 면의 법선 벡터(normal vector)역시 단위 벡터를 사용한다.

벡터

- 두 점으로 정의한 벡터
  - 점 q에서 점 p로의 벡터 v는 q-p로 표시하며, 다음과 같이 그림으로 표시할 수 있다.
  - 옆의 그림은 2차원 벡터를 표시한 것이지만, 이것은 3차원에서도 적용된다.



벡터의 연산

- 벡터의 연산은 덧셈, 뺄셈, 스칼라 곱셉, 나눗셈, 내적, 외적이 있다.
- 2개의 벡터 p, q와 상수 c가 있다고 할 때 다음과 같은 연산들을 정의 할 수 있다.

1. 덧셈(addition)

$$p + q = (x_p, y_p, z_p) + (x_q, y_q, z_q) = (x_p + x_q, y_p + y_q, z_p + z_q)$$

2. 스칼라 곱셉(scalar multiplication)

$$c \otimes p = (cx_p, cy_p, cz_p)$$

3. 내적(dot product)

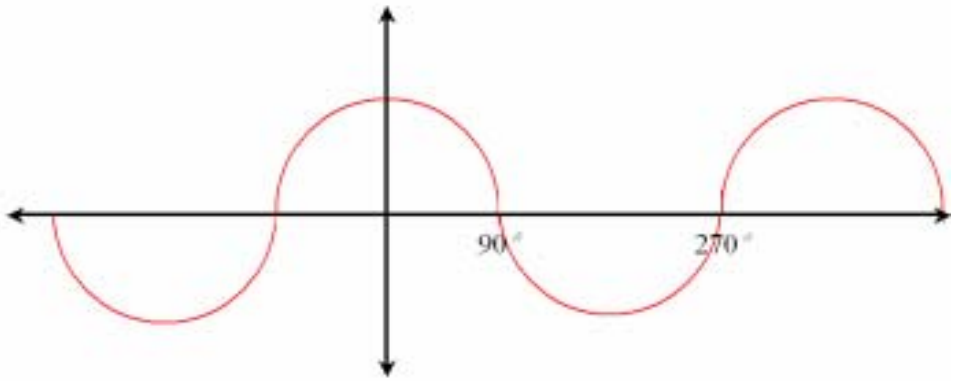
$$p \cdot q = x_p x_q + y_p y_q + z_p z_q = |p| \ |q| \cos\theta$$

4. 외적(cross product)

$$p \times q = (p_y q_z - p_z q_y, p_z q_x - p_x q_z, p_x q_y - p_y q_x)$$

벡터 내적 (inner product) 연산

- 스칼라 곱(scalar product) 또는 점 곱(dot product)으로도 불리운다.
- 두 벡터의 각 성분들끼리의 곱이다.
  - 정의: $P \cdot Q = \sum P_i \cdot Q_i$
- 내적은 일반적으로 두 벡터 사이의 각이 90도 이상인가를 판단하는 중요한 값이다.
- cos 그래프

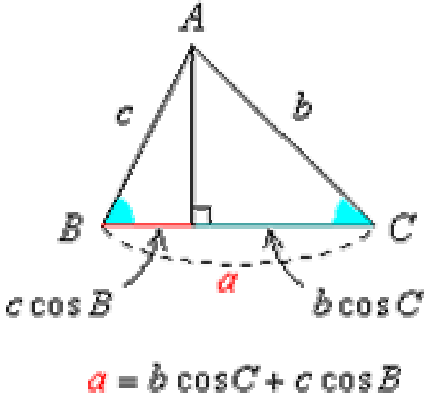


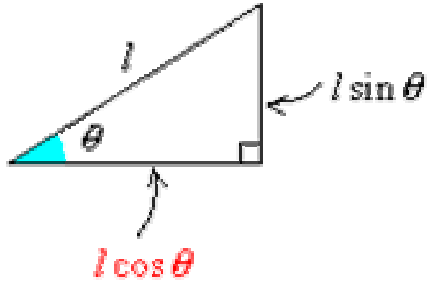
벡터 내적 연산

- 내적의 정의
  - $P*Q = \sum P_iQ_i$
- 내적의 계산
  - $P*Q = P_1Q_1 + P_2Q_{2+}P_3Q_3$
- 증명
  - Θ를 선분 OP와 OQ의 각도라 하면 코사인 법칙에 의해 다음이 성립
  - $\| P - Q \|^2 = \| P \|^2 + \| Q \|^2 - 2 \cdot \| P \| \cdot \| Q \| \cdot \cos\theta$
  - 이것을 전개하면
  - $\sum (P-Q)^2 = P^2_i + Q^2_i - 2 \cdot \| P \| \cdot \| Q \| \cdot \cos\theta$
  - 위 식을 간소화하면
  - $\sum -2 P_iQ_i = -2 \cdot \| P \| \cdot \| Q \| \cdot \cos\theta$
  - 최종적으로 $P*Q = \sum P_iQ_i$

코사인 제1법칙(the first law of cosines)

$a = b\cdot\cos C + c\cdot\cos B \rightarrow b = c\cdot\cos A + a\cdot\cos C \rightarrow c = a\cdot\cos B + b\cdot\cos A$





☞ 각 A 가 30°, 45°, 60° 와 같은 특수각이 아닐 때 각 A 의 대변 a 의 길이를 구할 때 씁니다. ^^

☞ 직각삼각형, 둔각삼각형일 때의 증명은 생략합니다. ^^ 3.

코사인 제2법칙(the second law of cosines)

$a^2 = b^2+c^2-2bc\cdot\cos A \rightarrow A = 90°$ 이면 피타고라스 정리가 됩니다. ^^
$b^2 = c^2+a^2-2ca\cdot\cos A$
$c^2 = a^2+b^2-2ab\cdot\cos A$
$a^2 = b^2+c^2-2bc\cdot\cos A$ 의 증명

$a = b\cdot\cos C + c\cdot\cos B$ …①
$b = c\cdot\cos A + a\cdot\cos C$ …②
$c = a\cdot\cos B + b\cdot\cos A$ …③ 에서 $\cos B$ 와 $\cos C$ 를 소거합니다. ^^

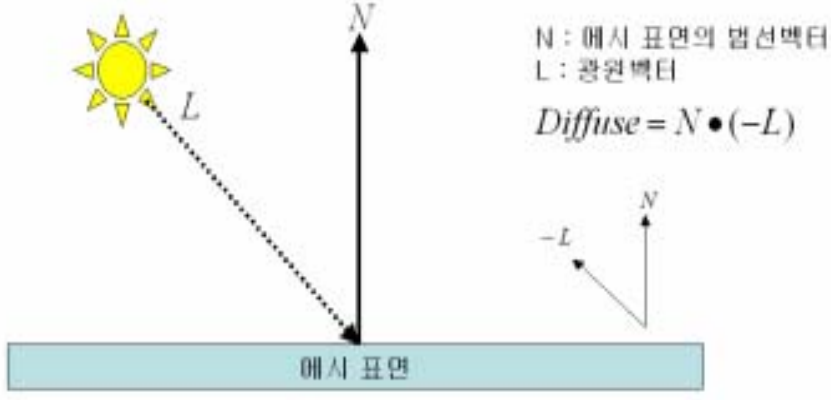
①×a : $a^2 = ab\cdot\cos C + ca\cdot\cos B$ …㉠
②×b : $b^2 = bc\cdot\cos A + ab\cdot\cos C$ …㉡
③×c : $c^2 = ca\cdot\cos B + bc\cdot\cos A$ …㉢

㉠−(㉡+㉢) 하면 $a^2-(b^2+c^2) = -2bc\cdot\cos A$ 에서 $a^2 = b^2+c^2-2bc\cdot\cos A$
☞ 두 변과 그 사이에 낀 각이 주어지면 각의 대변의 길이를 구할 수 있다는 뜻입니다. ^^

☞ $a^2 = b^2+c^2-2bc\cdot\cos A \Leftrightarrow \cos A = \frac{b^2+c^2-a^2}{2bc}$ → 세 변의 길이를 알면 모든 각을 구할 수 있다는 뜻! ^^

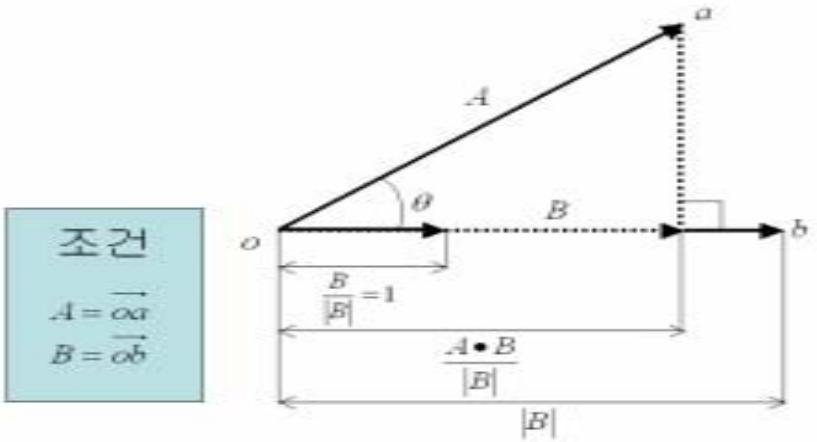
벡터 내적 연산

- 내적 연산의 유용한 성질
  - 내적 연산은 3D 그래픽 전반에 걸쳐서 사용된다. 특히 좌표계 변환과 조명 계산에 많이 사용된다.
  - 램버트 조명 모델 (Lambertian lighting model)



벡터 내적(inner product) 연산

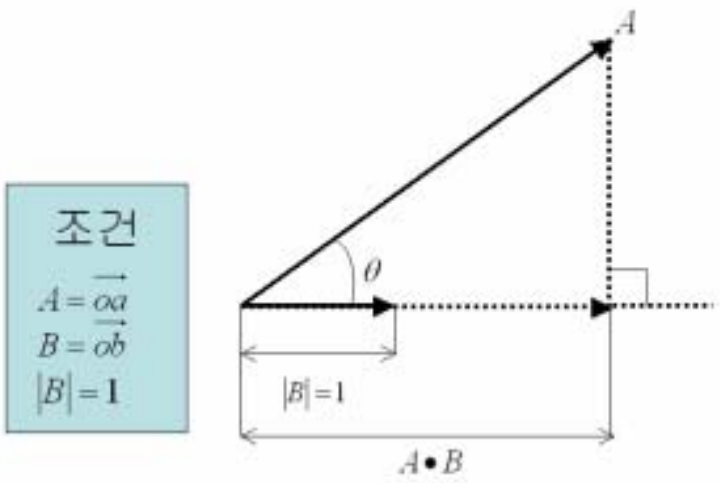
- 내적 연산의 기하학적 의미



- 벡터 A와 B가 있을 때 A · B 연산은 B벡터에서는 A벡터가 얼마만큼의 길이로 표현되는가를 나타낸다.

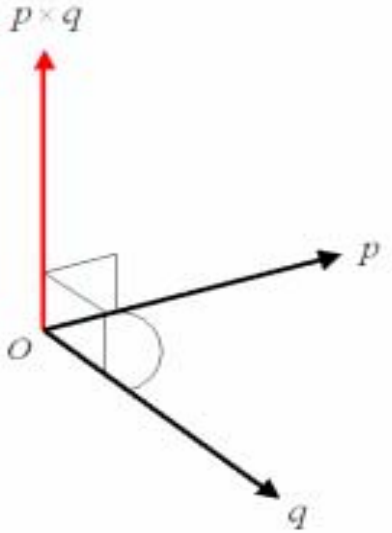
벡터 내적(inner product) 연산

- B를 좌표축이라고 가정한다면 A · B 연산은 A라는 벡터가 B라는 좌표축 위에서 얼마만큼의 크기를 갖고 있는가를 나타낼 수 있다.



벡터 외적(outer product) 연산

- 벡터 곱(vector product), cross product로도 불리운다.
- 두 벡터에 수직인 새로운 벡터이다.
  - PXQ=<$P_yQ_z - P_zQ_y$, $P_zQ_x - P_xQ_z$, $P_xQ_y - P_yQ_x$>
- 외적 연산의 특징은 결과가 2개의 벡터와 직교하는 벡터 값으로 나온다는 것이다.
- 외적이 가리키는 방향은 오른손 법칙을 따른다.



행렬

- 다음과 같이 사각형의 형태로 표기한 숫자 배열을 행렬 A라고 한다.
- 특히 n X n 행렬을 n차 정방행렬(square matrix)이라 한다.

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{1n} \\ a_{21} & a_{22} & a_{2n} \\ a_{m1} & a_{m2} & a_{mn} \end{pmatrix}$$

- 정방행렬에서 A에서 I = j일때 $a_{ij} = 1$이고, i ≠ j일 때 $a_{ij} = 0$인 행렬을 단위 행렬(identity matrix)이라고 하며, I또는 I_n이라 표시한다.

$$I_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

행렬의 성질

- 행렬을 사용하는 이유는 각종 변환을 행렬이라는 도구를 사용하여 간략하게 할 수 있기 때문이다. 또 결합법칙이 성립되기 때문이다.
- 행렬 곱셈
  - $A = (a_{ij})$, $B = (b_{ik})$일 때 $AB = \left[\sum_{j=1}^{n} a_{ij} b_{ik} \right]$ 이다.
  - DX 9.0 함수 구현
    - D3DXMATRIX *D3DXMatrixMultiply(D3DXMATRIX *pOut, CONST D3DXMATRIX *pM1, CONST D3DXMATRIX *pM2);

단위 행렬(Identity Matrix)

- $AI = IA = A$를 만족하는 행렬 I를 단위 행렬이라고한다.

$$I_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- DX 9.0 함수 구현
  - D3DXMATRIX *D3DXMatrixIdentity(D3DXMATRIX *pOut);

역행렬 (Inverse Matrix)

- $AA^{-1} = A^{-1}A = I$를 만족하는 행렬을 A^{-1}을 역행렬이라고 한다.

- DX 9.0 함수 구현
  - D3DMATRIX *D3DXMatrixInverse(D3DXMATRIX *pOut, FLOAT *pDeterminant, CONST D3DXMATRIX *pM);

기하 변환의 이해

- 2차원 변환
  - 2차원상의 어파인변환 (Affine transformation) : 한 점 p=(x , y) 를 2차원 어파인 사상(Affine mapping, 2차원 어파인)에 의해 p'=(x' , y') 로 바꾸어 주는 변환
- 3차원상의 좌표변환
  - 3차원 어파인 변환 : 3차원상의 한 점 p=(x, y, z)를 형태의 3차원 어파인 사상(3차원 어파인) 에 의해 또 다른 한 점 p'=(x', y', z')로 바꾸어 줌

기하 변환의 이해

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \nu_1 \\ \nu_2 \end{pmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \nu_1 \\ a_{21} & a_{22} & \nu_2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \mathrm{M} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

수학 이론의 일차변환 참조

표 2.4 2차원 어파인 변환

이동변환 (점 p를 (d_x, d_y) 만큼 이동)	$T(d_x,d_y)=\begin{pmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{pmatrix}$
회전변환 (점 p를 기준 원점에 대해 θ 만큼 회전)	$R(\theta)=\begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$
크기 변환 (점 p 의 x 좌표를 Sx, y 좌표를 Sy 만큼 확대)	$S(S_x,S_y)=\begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$
x축 방향 밀림 변환 (x 좌표만 sh_x 만큼 이동)	$SH_x(sh_x)=\begin{pmatrix} 1 & sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$
y축 방향 밀림 변환 (y 좌표만 sh_y 만큼 이동)	$SH_y(sh_y)=\begin{pmatrix} 1 & 0 & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

기하 변환의 이해

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = M \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \quad (\text{M은 } 4\times 4 \text{ 행렬})$$

표 2.5 3차원의 좌표변환

이동변환	$T(d_x d_y d_z) = \begin{pmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$
x축 회전 변환 (x 축을 중심으로 θ 만큼 회전)	$R_{x(\theta)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
y축 회전 변환	$R_{y(\theta)} = \begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
z축 회전 변환	$R_{z(\theta)} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
크기 변환	$S(s_x s_y s_z) = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
z 축 밀림 변환 (x 좌표를 sh_x, y 좌표를 sh_y 만큼 이동)	$SH_{xy}(sh_x sh_y) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ sh_x & sh_y & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

이동 행렬 (Translation Matrix)

- 주어진 위치로 이동하는 행렬 3 X 3크기만으로도 3차원 공간을 표현 가능하지만 이동행렬 때문에 4 X 4크기의 행렬이 되었다.
- 행렬 표현

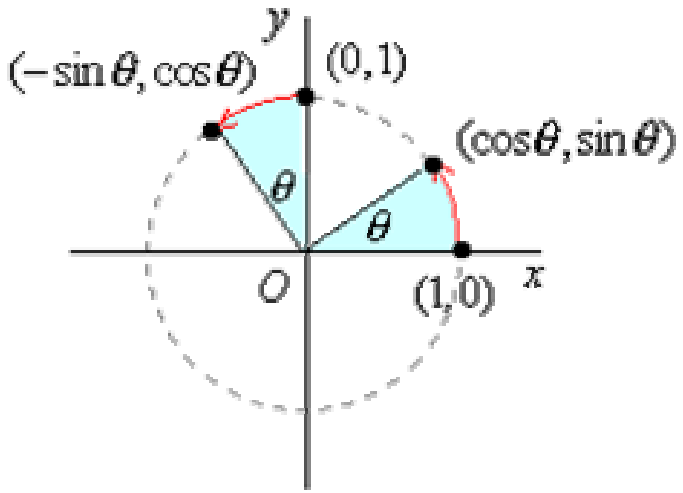
$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ dx & dy & dz & 1 \end{pmatrix}$$

- DX 9.0 함수 구현
  - D3DXMATRIX *D3DXMatrixTranslation(D3DXMATRIX *pOut, FLOAT x, FLOAT y, FLOAT z);

회전 이동

- 오른쪽 그림은 점 (1,0)과 점 (0,1)을 원점을 중심으로 θ 만큼 회전이동하면 각각
  - 점 (1,0) ➔ 점 $(\cos\theta, \sin\theta)$
  - 점 (0,1) ➔ 점 $(-\sin\theta, \cos\theta)$
- 점 (1,0)과 점 (0,1) 을 각각 점 (a,c)과 점 (b,d)로 옮기는 일차변환의 행렬은 $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$
  - 점 (1,0) ➔ 점 (a, b)
  - 점 (0,1) ➔ 점 (c, d)
- 원점을 중심으로 θ 만큼 회전이동 할 때의 행렬은

$$\begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$



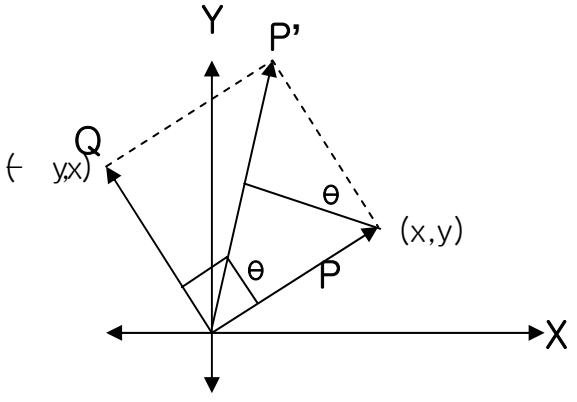
참고 http://mathteacher.pe.kr/

- 일차변환 $f:(x,y) \to (ax+by,\ cx+dy)$ 에서 $\begin{cases} x'=ax+by \\ y'=cx+dy \end{cases} \Leftrightarrow \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$ 와 같은 식을 얻을 수 있는 데, 이 식을 일차변환의 식이라 하고 이 때의 행렬 $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ 이 일차변환 f의 행렬

  (예) 일차변환 $f:(x,y) \to (x+y,\ x-y)$ 는 $\begin{cases} x'=x+y \\ y'=x-y \end{cases} \Leftrightarrow \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$

회전 변환

- 회전 행렬의 증명(2차원)
  - 2차원 벡터에 대한 90도 반시계 방향 회전은 x좌표와 y좌표를 교환하고 새 x좌표에 –를 붙인 것과 같다.
  - 밑의 그림에서 보듯이 벡터 P를 각도 θ만큼 회전한 결과로 생긴 벡터 P'는 P와 Q에 평행인 성분들로 표현가능
  - 간단한 삼각법적용시 p'는 다음수식과 같다.
  - P' = P*cosθ + Q*sinθ
  - 이에 기반해서 P'의 성분들을 표기하면
  - $P'_X = P_X*\cos\theta - P_Y*\sin\theta$
  - $P'_Y = P_Y*\cos\theta + P_X*\sin\theta$
  - 이를 행렬 형태로 쓴다면 다음과 같다.



$$P' = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} P$$

3차원 회전 표현 방식

- 오일러각(Euler-Angle) 방식: 시스템의 기본 좌표축을 중심으로 회전
  - 행렬(Matrix) 로 표현: 원주를 이동
  - 짐벌 록(gimbal lock: x, y, z축에 대한 회전각을 따로 정하므로, 두 개의 회전이 나머지 하나의 회전축을 상실케 하는 효과를 발생시켜, 결국 원하는 회전 결과를 얻을 수 없게 되는 현상)
  - http://www.anticz.com/eularqua.htm
- 각축(Angle-Axis) 방식: 임의의 회전축을 중심으로 회전
  - 행렬로 표현: 회전축을 시스템 좌표 축으로 이동하여 회전 후 다시 원래 위치로 이동
  - 사원수(Quaternion)로 표현: 구면을 이동
  - gimbal Lock 없어 일반적으로 많이 사용

회전 행렬(Rotation Matrix)

- DX 9.0 함수 구현
  - D3DXMatrixRotationX(), D3DXMatrixRotationY(), D3DXMatrixRotationZ(), D3DXMatrixRotationAxis(), D3DXMatrixRotationYawPitchRoll(), D3DXMatrixRotationQuaternion(),
- 행렬 표현

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

X축 회전 행렬

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

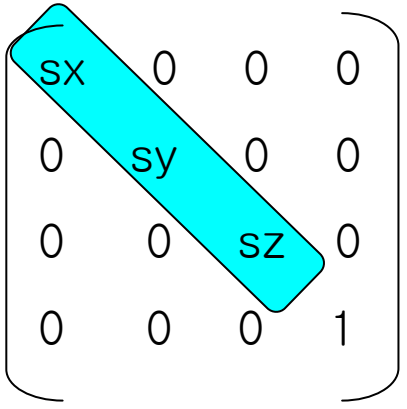
Y축 회전 행렬

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

Z축 회전 행렬

확대/축소 행렬(Scaling Matrix)

- DX 9.0 함수 구현
  - D3DXMATRIX *D3DXMatrixScaling(D3DXMATRIX *pOut, FLOAT sx, FLOAT sy, FLOAT sz);
- 행렬 표현



좌표계 변환행렬

- 행렬 표현

$$\begin{bmatrix} p_x & q_x & r_x & 0 \\ p_y & q_y & r_y & 0 \\ p_z & q_z & r_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  - 벡터 V를 $P=(p_x,p_y,p_z), Q(q_x,q_y,q_z), R=(r_x,r_y,r_z)$ 좌표계로 변환하려 다음과 같은 연산을 거쳐야 한다

$$v'_p = P \bullet V = (p_x v_x + p_y v_y + p_z v_z)$$
$$v'_q = Q \bullet V = (q_x v_x + q_y v_y + q_z v_z)$$
$$v'_p = R \bullet V = (r_x v_x + r_y v_y + r_z v_z)$$

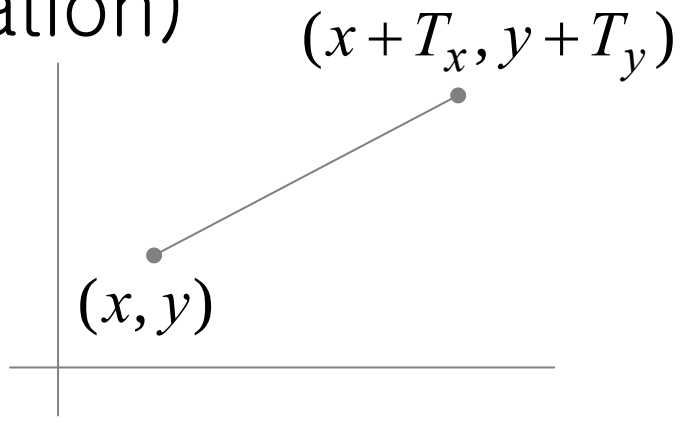
  - 그런데, 이 연산을 행렬로 표기하면 다음과 같다

$$V' = \begin{bmatrix} v_x & v_y & v_z & 1 \end{bmatrix} \times \begin{bmatrix} p_x & q_x & r_x & 0 \\ p_y & q_y & r_y & 0 \\ p_z & q_z & r_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

기본적인 변환

- 평행 이동 (translation)

$$x' = x + T_x$$
$$y' = y + T_y$$



(T_x, T_y) : 이동 벡터, 쉬프트 벡터(shift vector)

* Error : 장치 좌표계의 한계를 넘어가면 Error Message 를 보내거나, 표시하지 않거나, 잘못된 화상을 보여 준다.

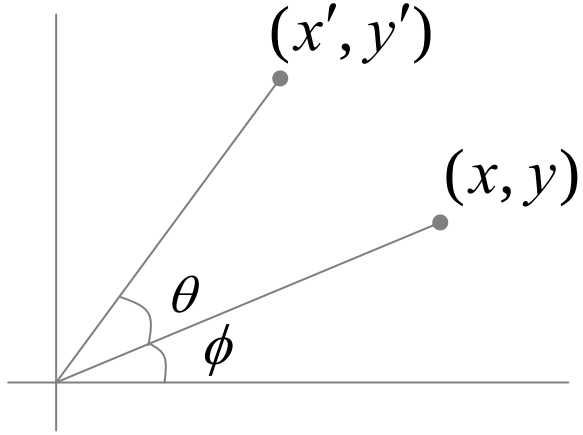
기본적인 변환

- 회전 (rotation)
  - 원주를 따른 점들의 변환

$$x = r\cos\phi$$
$$y = r\sin\phi \quad \text{(1)}$$

$$x' = r\cos(\phi+\theta)$$
$$= r\cos\phi\cos\theta - r\sin\phi\sin\theta$$
$$y' = r\sin(\phi+\theta) \quad \text{(2)}$$
$$= r\sin\phi\cos\theta + r\cos\phi\sin\theta$$

$$x' = x\cos\theta - y\sin\theta$$
$$y' = y\cos\theta + x\sin\theta$$



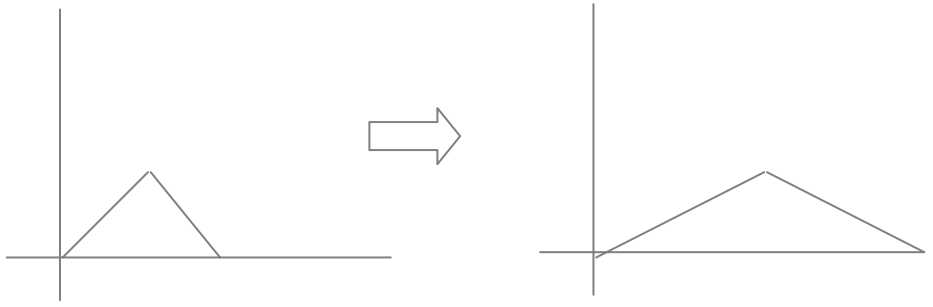
삼각함수의 덧셈 정리

- 다음을 증명하는 가장 쉬운 방법은 오일러 식을 이용하는 것이다. 탄젠트 공식은 위의 둘을 결합하여 얻는다.
  - $\sin(x + y) = \sin(x)\cos(y) + \cos(x)\sin(y)$
  - $\cos(x + y) = \cos(x)\cos(y) - \sin(x)\sin(y)$

기본적인 2차원 변환

- 신축 (scaling)

$$x' = x \cdot S_x$$
$$y' = y \cdot S_y$$



- 1 보다 작은 수 : 대상 크기를 축소
- 1 보다 큰 수 : 대상 크기를 확대
- 1 인 경우 : 변하지 않는다

* 균일 신축 : Sx와 Sy가 같은 값이 주어지면 상대적 비율 유지

행렬표현과 동차좌표

- 행렬 표현
  - 동차 좌표 (homogeneous coordinate)
    - 2차원 좌표 (x, y)를 $x_h = x \cdot \varpi,\ y_h = y \cdot \varpi$ 인 3차원 형태 $[x_h \quad y_h \quad \varpi]$ 로 나타내는 것을 의미한다.
      일반적으로 2차원 변환에서는 $\varpi = 1$ 로 놓을 수 있다.
      따라서 각 2차원 좌표는 동차 좌표 형태로 $[x \ y \ 1]$이다.
  - 평행이동 (LH)

$$[x' \quad y' \quad 1] = [x \quad y \quad 1] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_x & T_y & 1 \end{bmatrix}$$

행렬표현과 동차좌표

- 행렬 표현 (계속)
  - 회전

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

  - 신축

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

행렬표현과 동차좌표

- Short Notation

$$T(T_x, T_y) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_x & T_y & 1 \end{bmatrix}$$

$$R(\theta) = \begin{bmatrix} \cos\theta & \sin\vartheta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$S(S_x, S_y) = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

변환 방정식의 행렬형태 표현

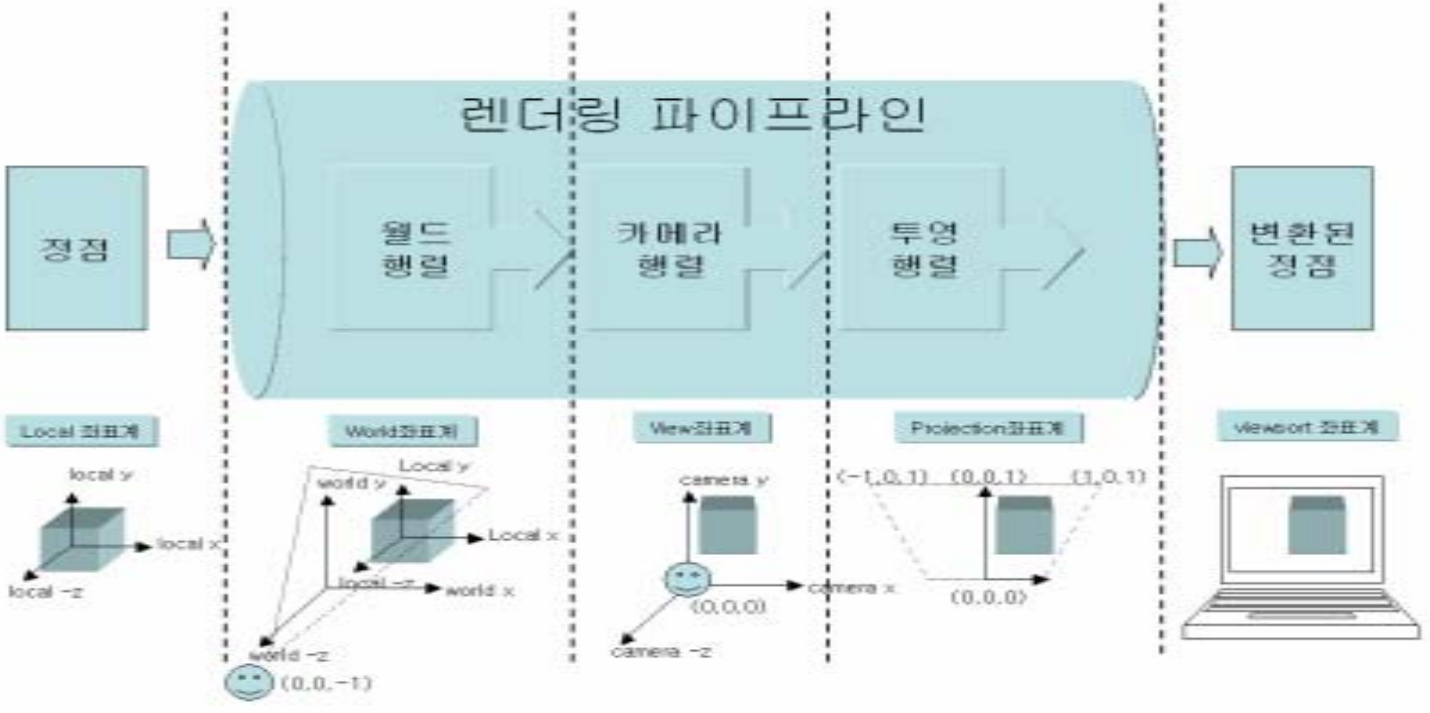
$$P' = P \cdot T(T_x, T_y)$$

$$P' = P \cdot R(\theta)$$

$$P' = P \cdot S(S_x, S_y)$$

렌더링 파이프라인

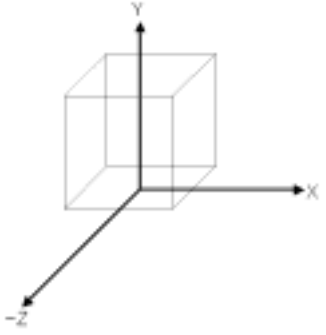
- 렌더링 파이프라인 개념도



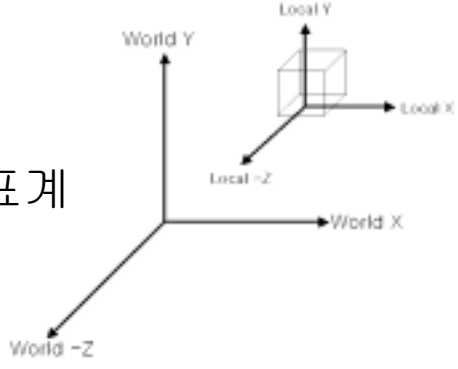
렌더링 파이프라인

- 월드 변환 행렬
  - 로컬 좌표계를 3차원 절대 죄표계인 월드 좌표계로 변환하는 행렬이다.

로컬 좌표계



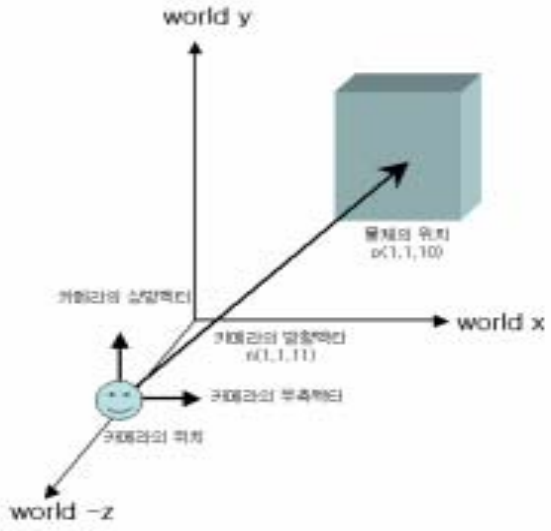
월드 좌표계



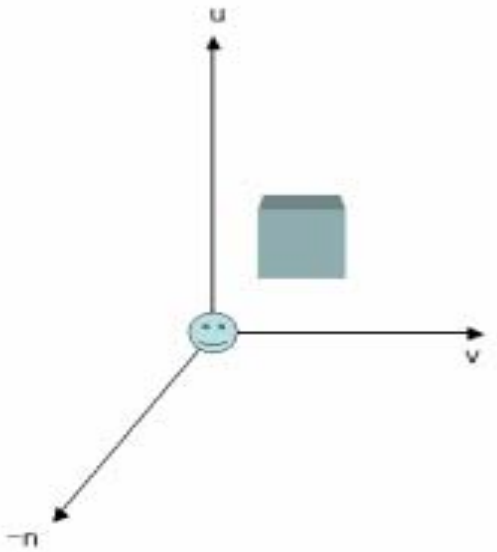
  - 모든 메시마다 각자의 고유한 로컬 좌표계가 존재하기 때문에 이들을 월드 좌표계로 변환하려면 메시마다 변환 행렬(transform matrix)이 있어야 한다.

렌더링 파이프라인

- 카메라 변환 행렬
  - 카메라가 존재하는 곳을 원점으로 하고 , 카메라가 바라보는 방향을 기준으로 한 새 카메라 좌표계를 만든 뒤, 이 좌표계로 월드 좌표계를 변환하는 것이다.



카메라 행렬 적용 전



카메라 행렬 적용 후

카메라 변환행렬

- 뷰좌표 = 월드 좌표 X 이동 행렬 X 좌표계 변환행렬

- 새로운 카메라 좌표계의 축 n, v, u
  - n축(z축) = normalize(p-c)
  - v축(x축) = normalize(upxn)
  - u축(y축) = nxv

  - p = 목표물의 좌표
  - c = 카메라의 좌표
  - up = 상방벡터(일반적으로 (0,1,0)

카메라 변환행렬

- [X',Y',Z',1] = [X,Y,Z,1] X

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -C_X & -C_Y & -C_Z & 1 \end{pmatrix} \begin{pmatrix} V_X & U_X & N_X & 0 \\ V_Y & U_Y & N_Y & 0 \\ V_Z & U_Z & N_Z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- 정리하면

[X',Y',Z',1] = [X,Y,Z,1] X

$$\begin{pmatrix} V_X & U_X & N_X & 0 \\ V_Y & U_Y & N_Y & 0 \\ V_Z & U_Z & N_Z & 0 \\ -(C*V) & -(C*U) & -(C*N) & 1 \end{pmatrix}$$

이것이 최종 카메라 행렬

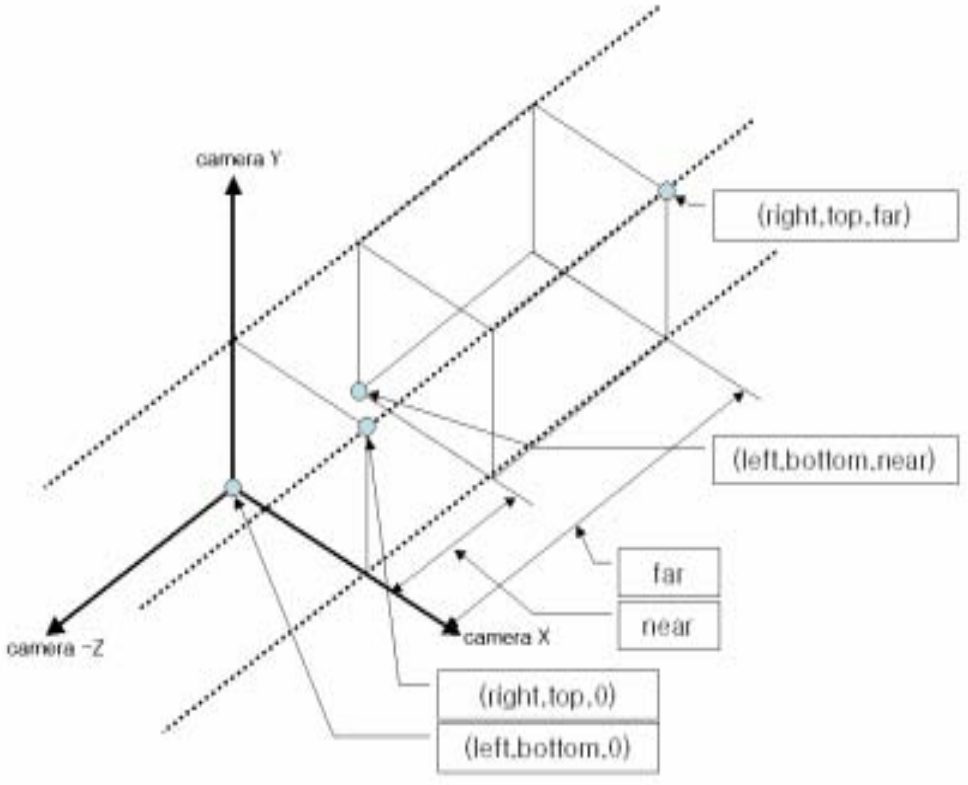
$$\begin{matrix} -(c_x \cdot v_x + c_y \cdot v_y + c_z \cdot v_z) & -(c_x \cdot u_x + c_y \cdot u_y + c_z \cdot u_z) & -(c_x \cdot n_x + c_y \cdot n_y + c_z \cdot n_z) & 1 \end{matrix}$$

렌더링 파이프라인

- 투영 변환 행렬
  - 3차원 좌표계를 2차원 좌표계로 바꾸는 변환이다.
  - 직교투영(orthographic projection)과 원근투영(perspective projection)이 있다.
  - 직교 투영법은 CAD/CAM 등의 모델링 등에서 사용되는 방법이다.
  - 직교 투영은 Z값의 거리에 따른 오차가 없기 때문에 정밀한 모델링이나 측량 등에 유리한 반면에 현실감은 떨어진다.
  - 원근 투영은 거리에 따라서 물체가 멀어지는 것을 표현한 변환 방법이다.
  - 게임 제작에 많이 사용된다.

렌더링 파이프라인

- 직교 투영(orthographic projection)



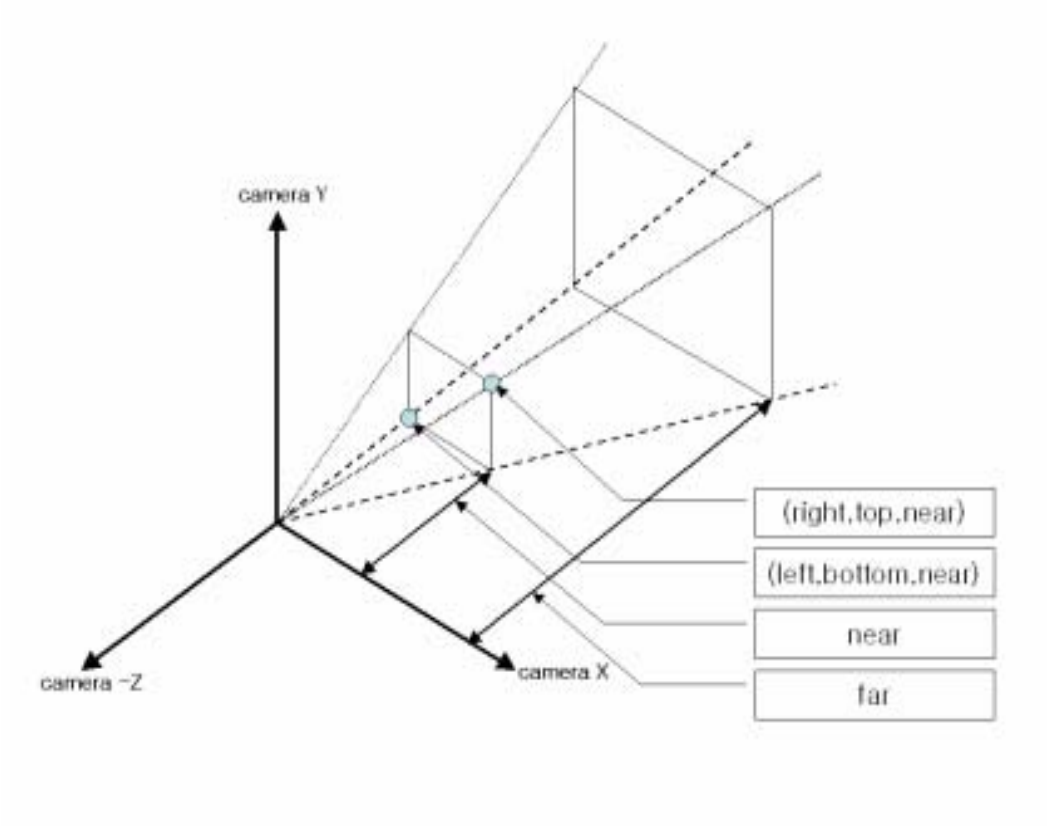
직교 투영의 시야

$$\begin{pmatrix} \frac{2}{w} & 0 & 0 & 0 \\ 0 & \frac{2}{h} & 0 & 0 \\ 0 & 0 & \frac{1}{f-n} & 1 \\ 0 & 0 & \frac{-n}{f-n} & 0 \end{pmatrix}$$

직교 투영의 결과

렌더링 파이프라인

- 원근 투영(perspective projection)

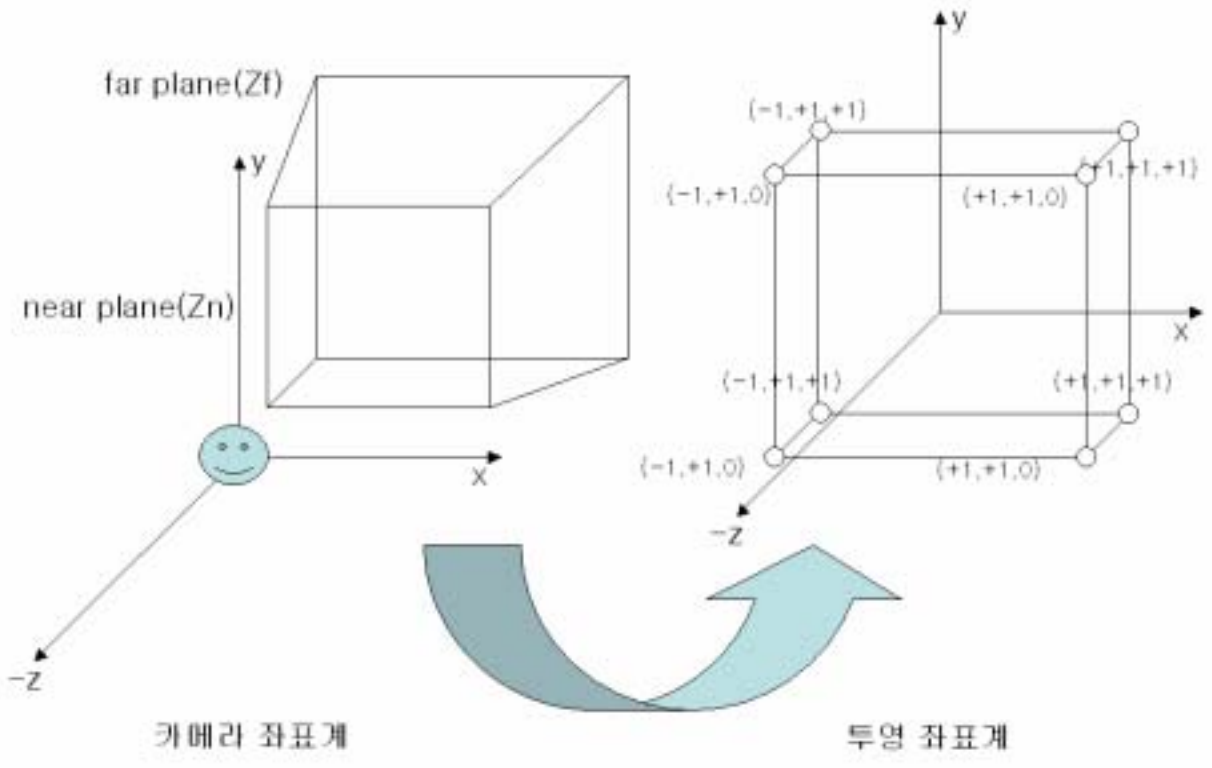


원근 투영의 시야

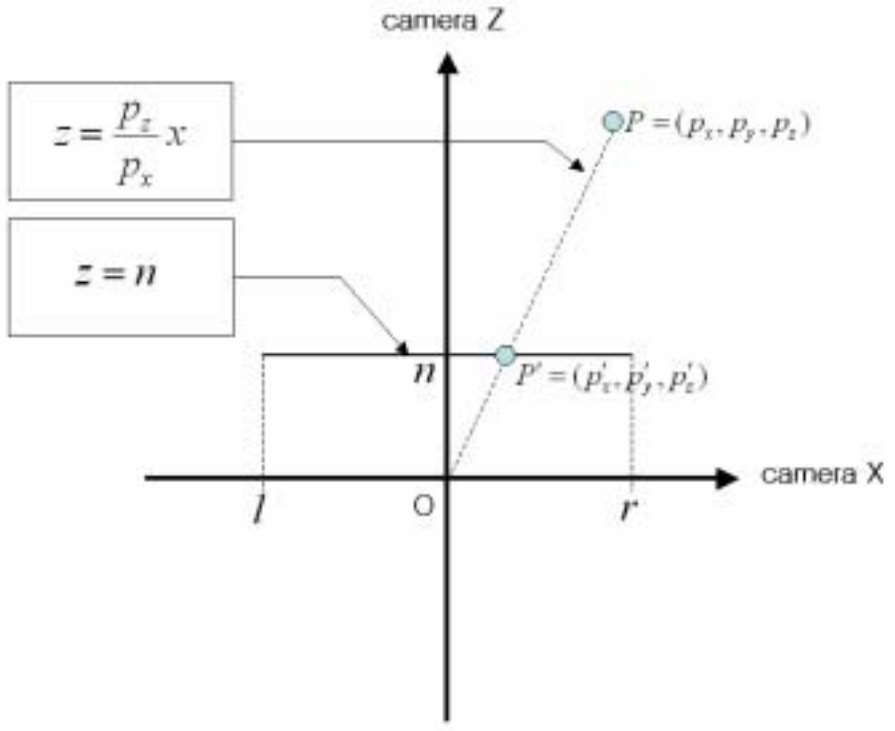
$$\begin{pmatrix} \frac{2n}{w} & 0 & 0 & 0 \\ 0 & \frac{2n}{h} & 0 & 0 \\ 0 & 0 & \frac{f}{f-n} & 1 \\ 0 & 0 & \frac{-fn}{f-n} & 0 \end{pmatrix}$$

원근 투영의 결과

카메라 좌표계에서 투영 좌표계로의 변환



XZ 평면에서의 직선의 방정식



사원수 (Quaternion; 4개가 한벌)

- 사원수란 3차원 그래픽에서 회전을 표현할 때, 행렬 대신에 사용하는 수학적 개념으로, 행렬과 달리 오류도 발생하지 않고, 데이터 크기도 작기 때문에 즐겨 사용되는 개념이다.
- 사원수와 행렬의 성능 차이

비교 항목	행렬	사원수
데이터량	float 16개	float 4개
연산 속도	float 16개 X float 16개	float 4개 X float 4개
결과의 질	Gimbal Lock등 오류 발생	최단호(shortest arc) 보간

사원수

- 사원수는 4차원 복소수 공간(complex space)벡터
  - $q = < x, y, z, w > = xi + yj + zk + w$ 혹은 $q = s + v$
    (단, s = w 회전각도, v = $< x, y, z >$ 벡터)

$$i^2 = j^2 = k^2 = -1 \qquad \text{(참고) 허수 } i^2 = -1$$
$$ij = -ji = k$$
$$jk = -ik = j$$
$$ki = -ik = j$$

- 단위 사원수
  - q = <0, 0, 0, 1>=0i + 0j + 0k + 1을 단위 사원수라 한다.
- 켤레 사원수(conjugate)
  - 사원수 $q = s + v$의 켤레를 $\bar{q}$ 로 표기하며 $\bar{q} = s - v$ 로 정의한다.
    이때, $q\bar{q} = \bar{q}q = q \bullet q = \|q\|^2 = q^2$ 이다.
- 역수
  - 0이 아닌 사원수 q의 역수(inverse)를 q 로 정의하면 $q^{-1} = \frac{\bar{q}}{q^2}$ 이다.

사원수

- 사원수의 회전
  - 사원수는 벡터의 회전과 밀접한 관계가 있는데, 축 A에 대하여 각도 θ만큼의 회전을 사원수로 나타내면

$$q = \cos\frac{\theta}{2} + A\sin\frac{\theta}{2}$$

  - 사원수로 점 p를 회전하려면 qPq^{-1} 계산
  - 최종적으로 얻어진 사원수를 실제 프로그램에서 회전에 사용하기 위해서는 다음과 같은 행렬로 변환

$$\begin{bmatrix} 1-2y^2-2z^2 & 2xy-2wz & 2xz+2wy & 0 \\ 2xy+2yz & 1-2x^2-2z^2 & 2yz+2wx & 0 \\ 2xz-2wy & 2yz+2wx & 1-2x^2-2y^2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

사원수와 관련된 D3D 객체

- 사원수 구조체

typedef struct D3DXQUATERNION
{ FLOAT x;
FLOAT y;
FLOAT z;
FLOAT w;
} D3DXQUATERNION; Members

x: The x-component.

y: The y-component.

z: The z-component.

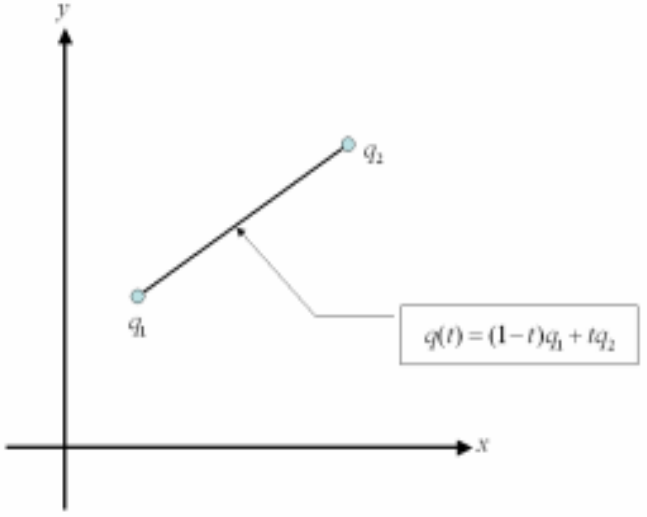
w: The w-component.

단, q.x = sin(theta/2) * axis.x
q.y = sin(theta/2) * axis.y
q.z = sin(theta/2) * axis.z q.w = cos(theta/2)

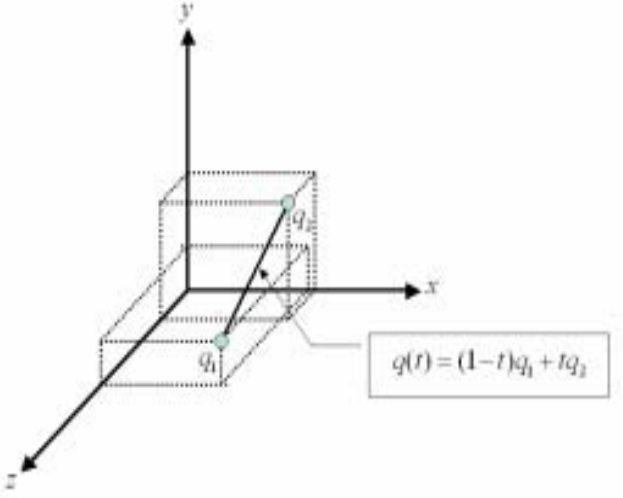
- DX 9.0 함수 구현
  - D3DXQUATERNION *D3DXQuaternionMultiply(D3DXQUATERNION **pOut*, CONST D3DXQUATERNION **pQ1*, CONST D3DXQUATERNION **pQ2*);
  - D3DXQUATERNION *D3DXQuaternionSlerp(D3DXQUATERNION **pOut*, CONST D3DXQUATERNION **pQ1*, CONST D3DXQUATERNION **pQ2*, FLOAT *t*);
- Quaternions add a fourth element to the [*x, y, z*] values that define a vector, resulting in arbitrary 4-D vectors. However, the following illustrates how each element of a unit quaternion relates to an axis-angle rotation (where *q* represents a unit quaternion (x, y, z, w), *axis* is normalized, and *theta* is the desired CCW rotation about the axis):

사원수

- 사원수 보간
  - 보간이란 한자 뜻 그대로, 중간을 채워 넣는다는 뜻이다.
  - 가장 쉬운 방식이 선형보간(linear interpolation) 방식이다.



선형 보간의 2차원 평면 기하학적 의미



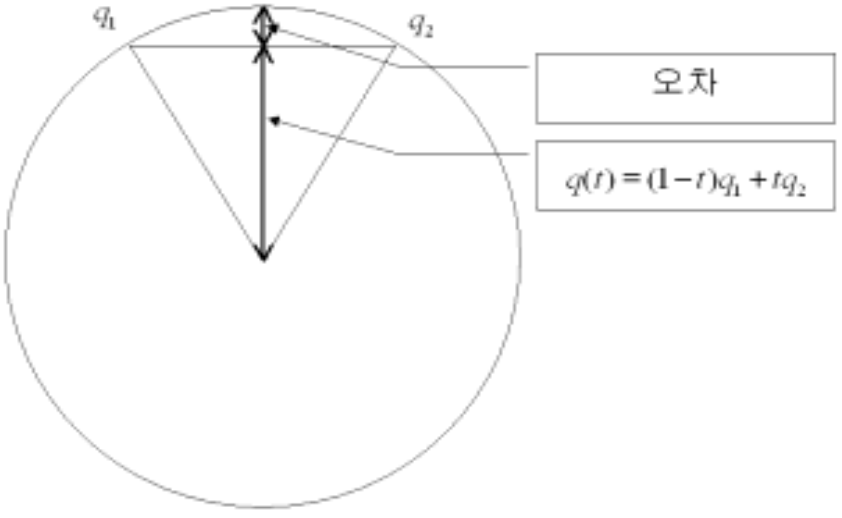
선형 보간의 3차원 공간 기하학적 의미

사원수

- 선형보간의 문제점
  - 사원수의 공간은 구면공간의 성격을 띠는데. 이를 선형 보간하게 되면 오차가 생긴다.
  - 정규화를 통해 보정할 수 있다.
  - 정규화한 최종 결과

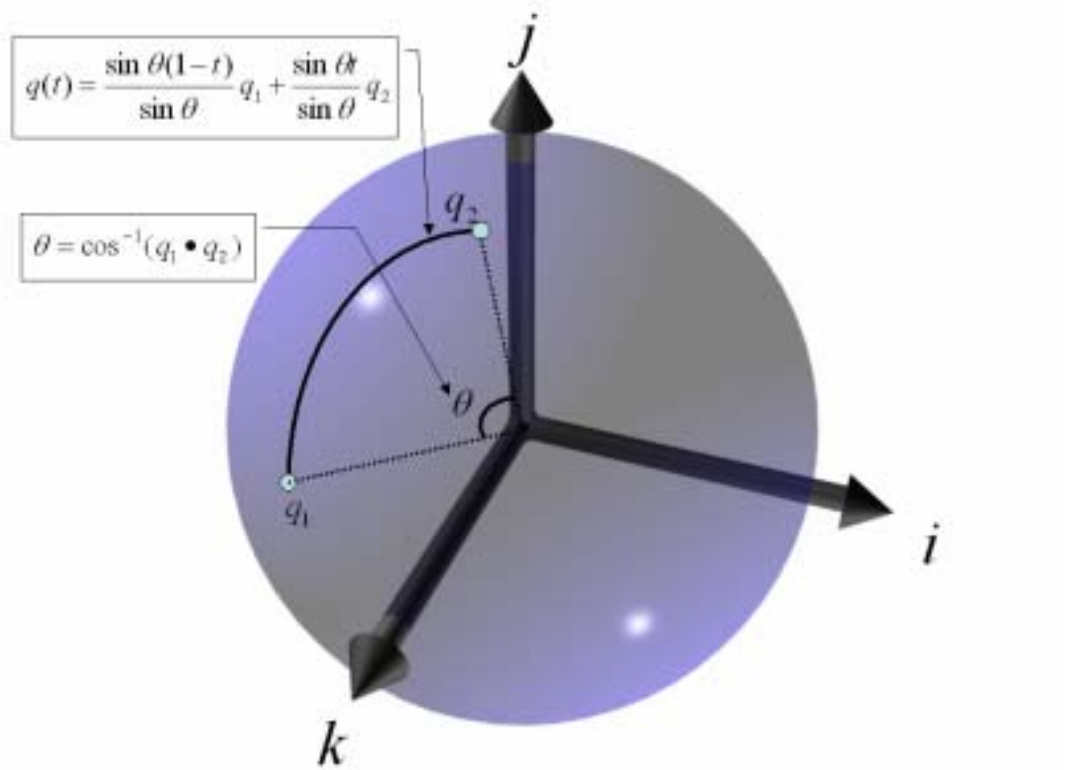
$$q(t) = \frac{(1-t)q_1 + tq_2}{\|(1-t)q_1 + tq_2\|}$$

  - 선형보간의 특성상 직선과 구면의 속도 오차가 를 해결하기 위해 구면 선형 보간을 사용한다.



사원수

- 구면 선형 보간
  - 구면 선형 보간의 기하학적 의미



구면 선형 보간의 사원수 공간 기하학적 의미

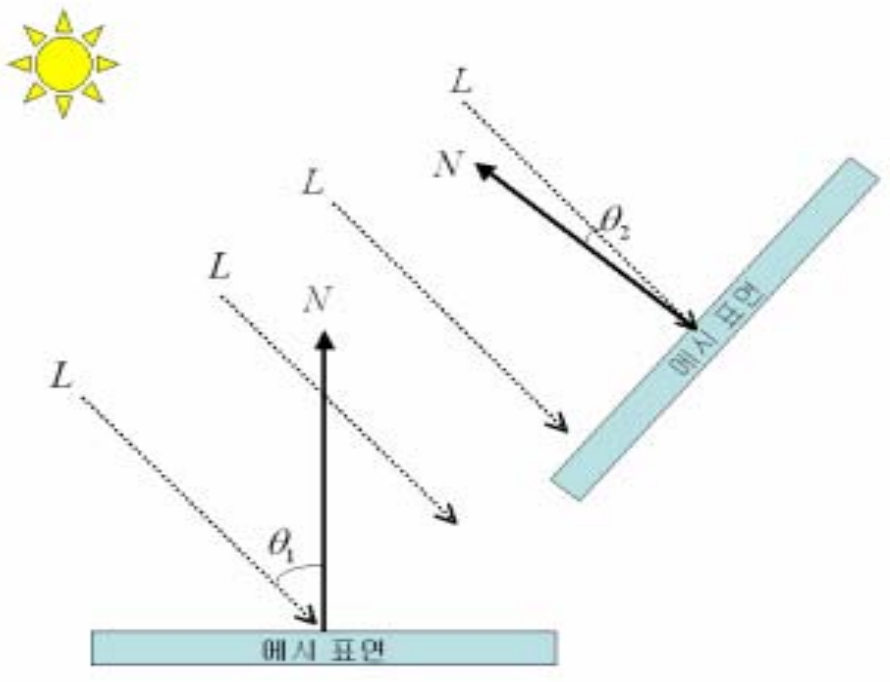
조명 모델

- 조명모델이란
  - 광원으로부터 점들까지의 조도를 계산하는 방법을 말한다.
  - 경험 모델과 물리 모델로 나뉜다.

- 광원
  - 빛의 출발점이다.
  - 주변 광원(ambient light)
    - 동일한 밝기의 빛이 장면 안의 모든 표면에 일정하게 반사되는 것을 말한다.
    - 이상적인 형태의 광원, 수학적 형태의 빛

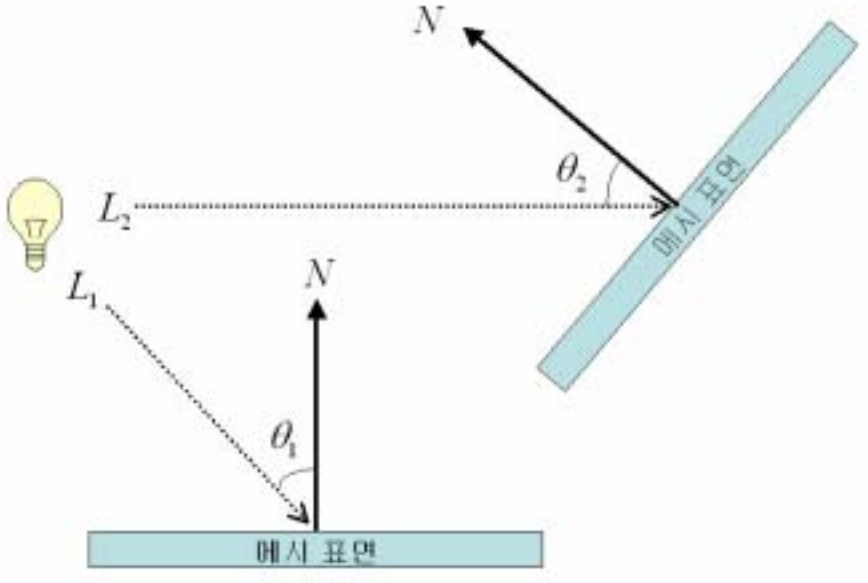
조명 모델

- 지향 광원(directional light)
  - 한 방향으로 무한히 뻗어나가는 빛이다. 비슷한 예로는 태양광이 있다.



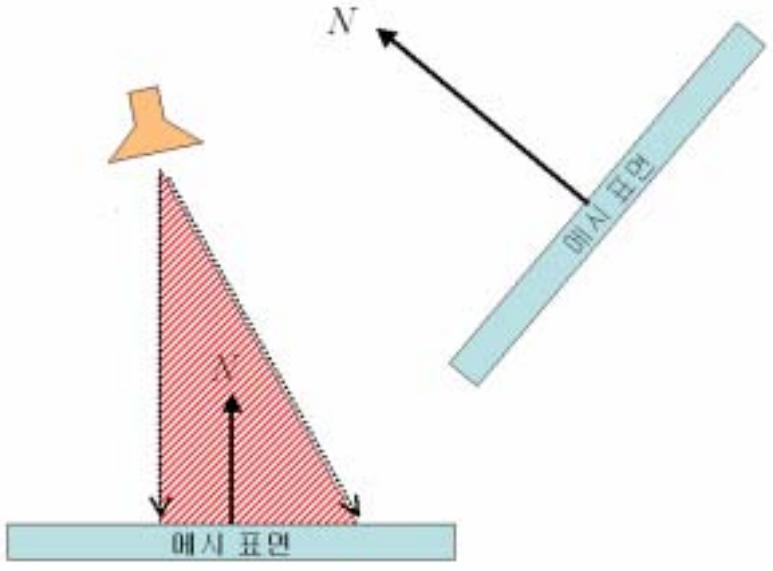
조명 모델

- 점 광원(point light)
  - 한 점을 중심으로 주변으로 퍼져나가는 빛을 나타낸다. 백열전구를 모델링한 것이라고 생각하면 된다.



조명 모델

- 점적 광원(spot light)
  - 점적 광원은 점 광원과 달리 특정한 한 방향으로만 퍼져나가도록 한 것이다.



조명 모델

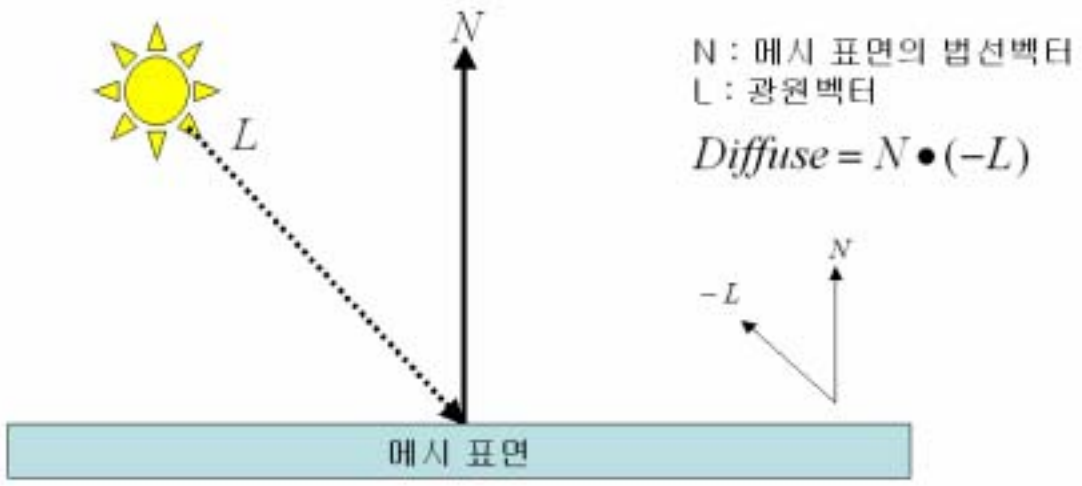
- 직접 조명
  - 조명에는 크게 직접 조명(direct illumination)과 전역 조명(global illumination)이 있다.
  - 직접조명은 물체 표면의 점들이 장면 내의 모든 광언들로부터 직접적으로 받는 빛을 말한다.
  - 전역조명이란 다른 물체를 거쳐서 받는 빛까지 모두 포함한 것을 말한다.
  - 주변 조명(ambient illumination)
    - 주변 광원에서 설명한 대로 정면 전반에 걸쳐서 일정한 빛의 강도가 주어지는 조명이다.

$$I_{local} = K_{ambient} I_{ambient}$$

조명 모델

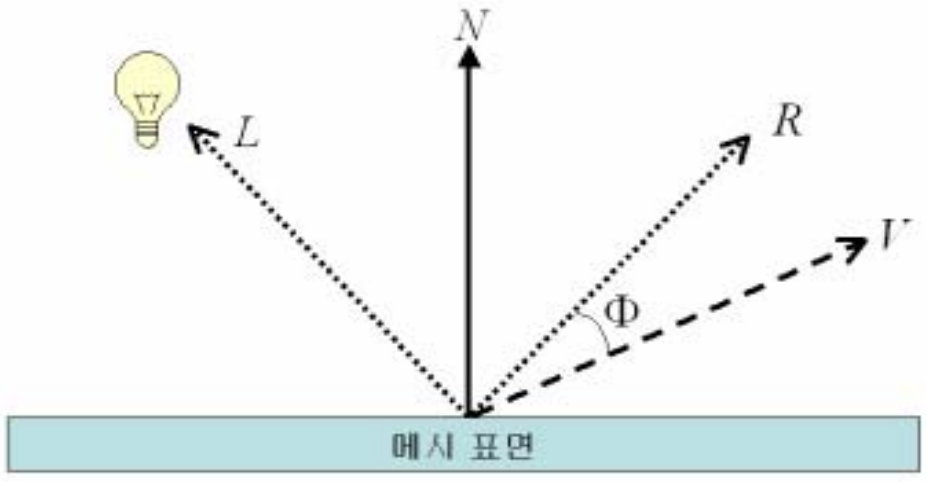
- 확산 조명(diffuse illumination)
  - 표면이 반사되는 빛의 강도는 view 방향과는 관계없이, 표면과 빛의 방향에 의해서 영향을 받는다. 이것을 램버트 반사 혹은 램버트 코사인 법칙이라고 한다.

$I_{da} = K_a I_a \cos\theta = I_d K_d \max(N \bullet L, 0)$ 단, K_a는 주변 조명 계수



조명 모델

- 반사 조명(specular illumination)
  - 코팅된 차량이나 금속성 하이라이트를 만들어 내는 것을 반사 조명이라고 한다.

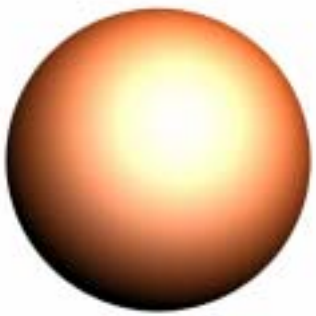


조명 모델

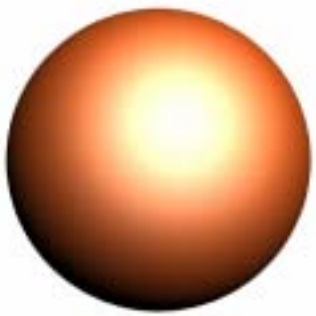
$$I_s = I_s K_s (\cos\Phi)^n = I_s K_s (V \bullet R)^n$$

1 단, K_s는 반사 조명 계수, V는 관찰자(카메라)벡터, R은 반사 벡터)

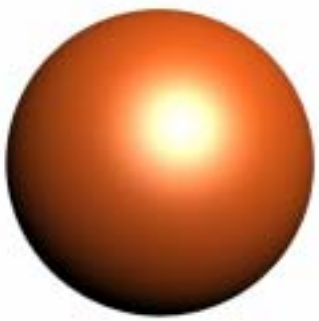
- 여기서 n제곱 값에 따라서 반사 조명은 다음과 같이 변한다.



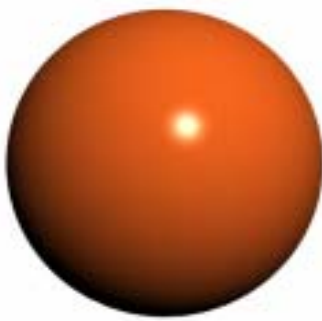
n = 1



n = 2



n = 3



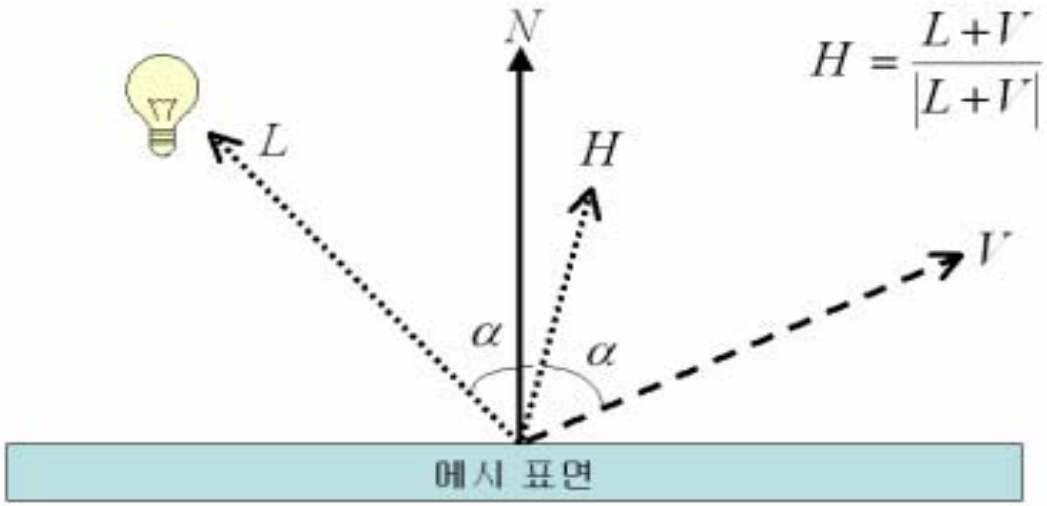
n = 4

조명 모델

- 반사 벡터 R을 구하지 않고 좀 더 쉽게 구할 수 있는 중간 벡터 H를 사용해도 비슷한 결과를 얻어내는 수식

$$I_s = I_s K_s \max(N \bullet H,\ 0)^n \quad (N \bullet L > 0)$$

(단, $N \bullet L > 0$ 일 때 1, $N \bullet L \le 0$ 일 때 0이다.



$$H = \frac{L+V}{|L+V|}$$

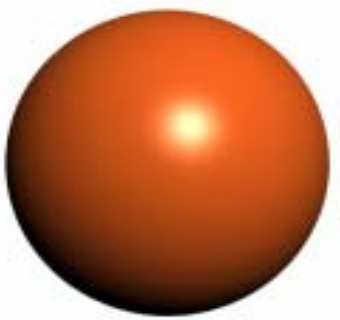
조명 모델

- 방출 조명(emissive illumination)
  - 특정한 물체들은 빛을 반사할 뿐만 아니라 빛을 뿜기도 하는데, 이러한 조명을 방출 조명이라고 한다.
  - $I = E$
- 직접 조명 정리
  - 직접 조명은 최종값을 얻기 위해서 주변 + 확산 + 반사 + 발광 조명값을 합친 것이다. 이를 수식으로 정리하면 다음과 같다.

$$I_{total} = K_a I_b + \sum (I_d K_d \max(N \bullet L_i, 0) + I_s K_s \max(N \bullet H_i, 0)^n (N \bullet L_i > 0)) + E$$

주변광

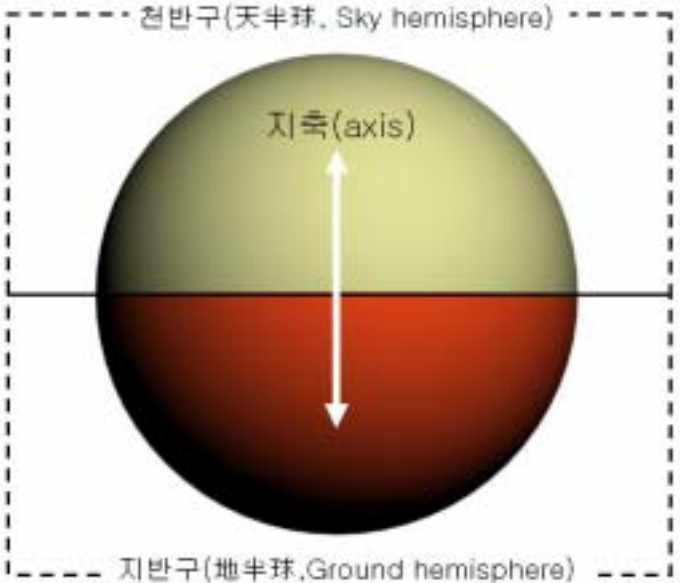
주변광 + 확산광



주변광 + 확산광 + 반사광

조명 모델

- 전역 조명(global illumination)
  - 라디오시티(radiosity)
  - 광선 추적(raytracing)
  - 포톤 맵핑(photon mapping)
  - 반구 조명(hemisphere illumination)
    - 천반구와 지반구의 광원색으로 구성하여 2개의 반구색을 합성하는 기법
    - 색 합성에는 반구의 축과 지축과의 각도(θ)를 사용한다.



천반구와 지반구 모형

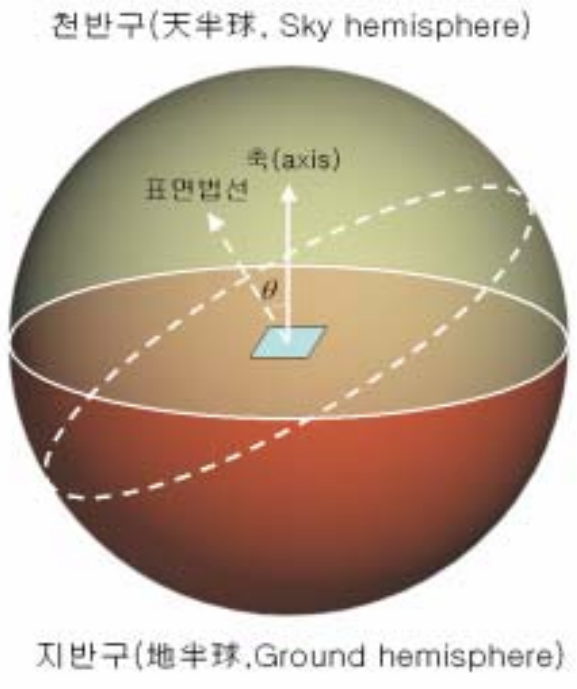
반구 조명 합성 공식

$C_{vertex} = (0.5 + \cos\theta)C_{sky} + (0.5 - \cos\theta)C_{ground}$

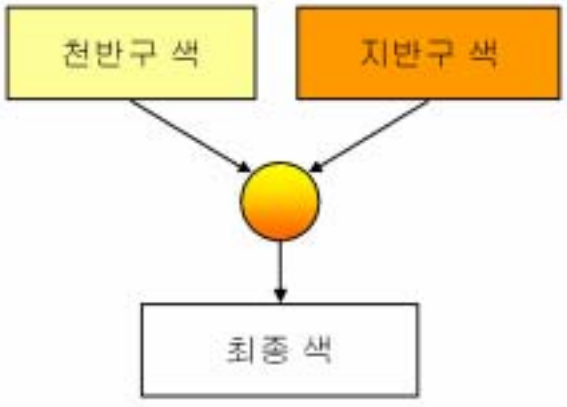
단, C_{sky}는 천반구색, C_{ground}는 지반구색)

조명 모델

- 반구 조명과 반구 조명의 색합성 과정



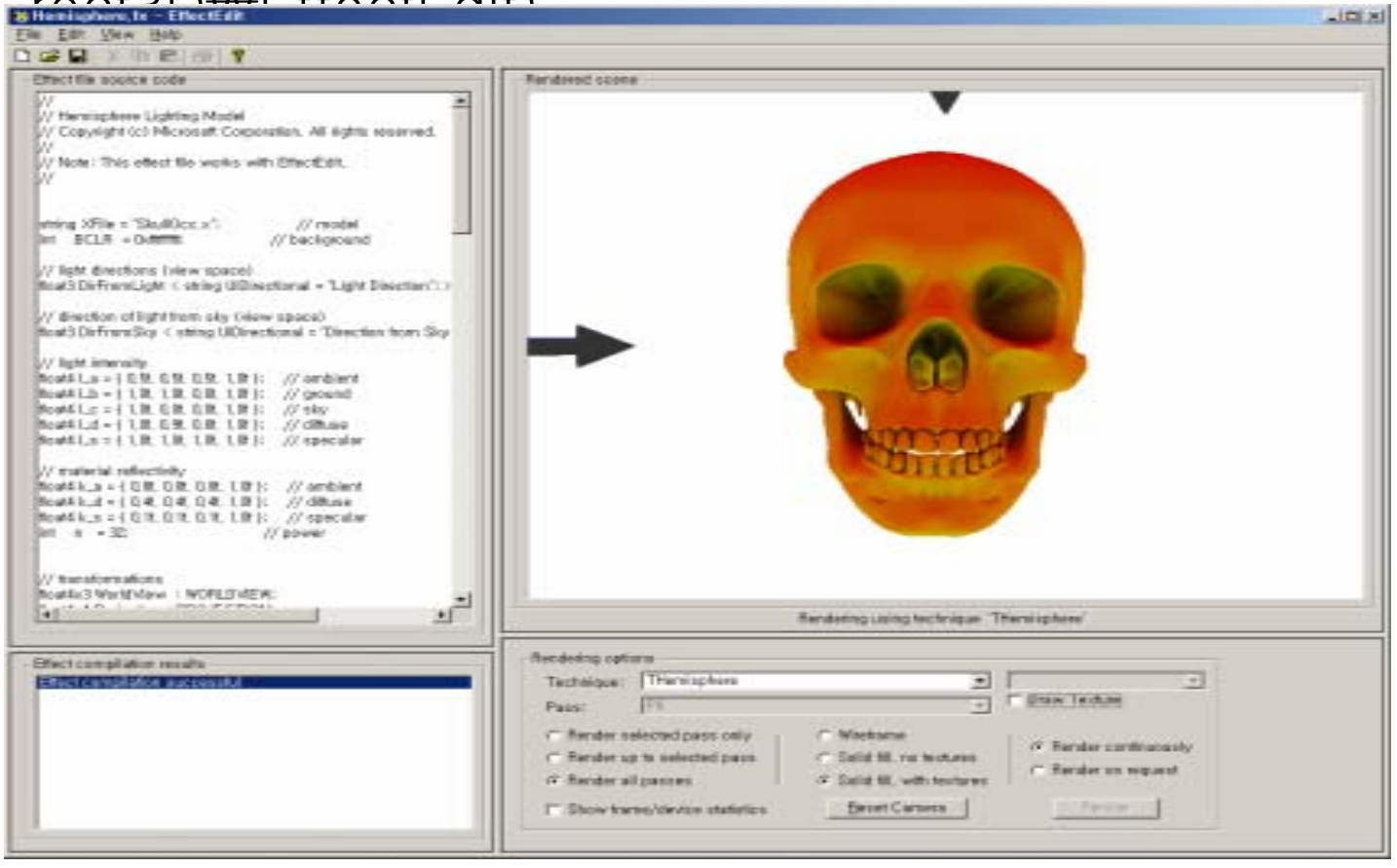
반구 조명



반구 조명의 색합성 과정

조명 모델

- 반구 조명 실행 결과 (C:₩DXSDK₩Samples₩C++₩Direct3D₩EffectEdit)

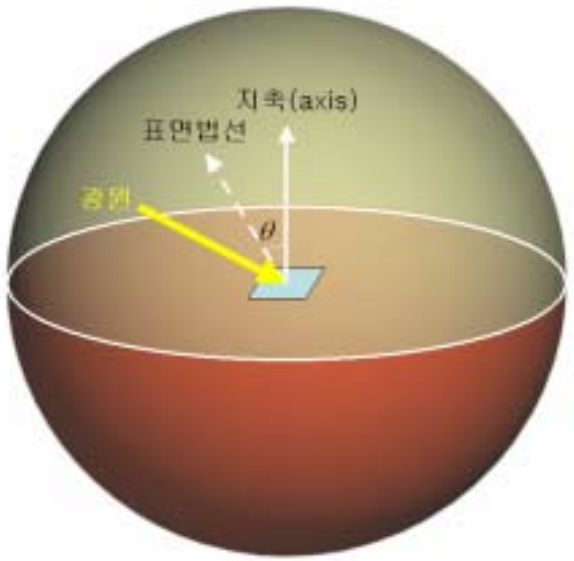


조명 모델

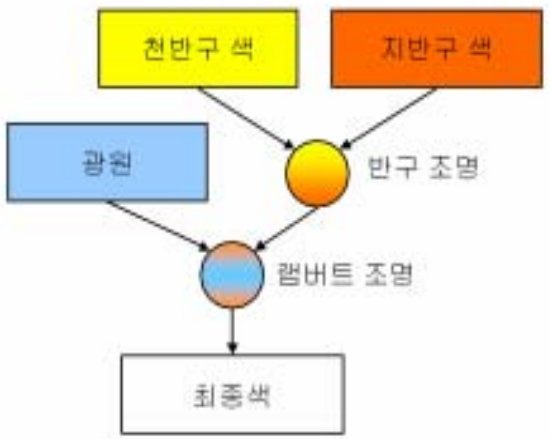
- 반구 조명 + 램버트 확산 조명

$$C_{vertex} = \max(N \bullet L) + (0.5 + \cos\theta)C_{sky} + (0.5 - \cos\theta)C_{ground}$$

(단, N은 표면 법선 벡터, L은 광원 벡터)



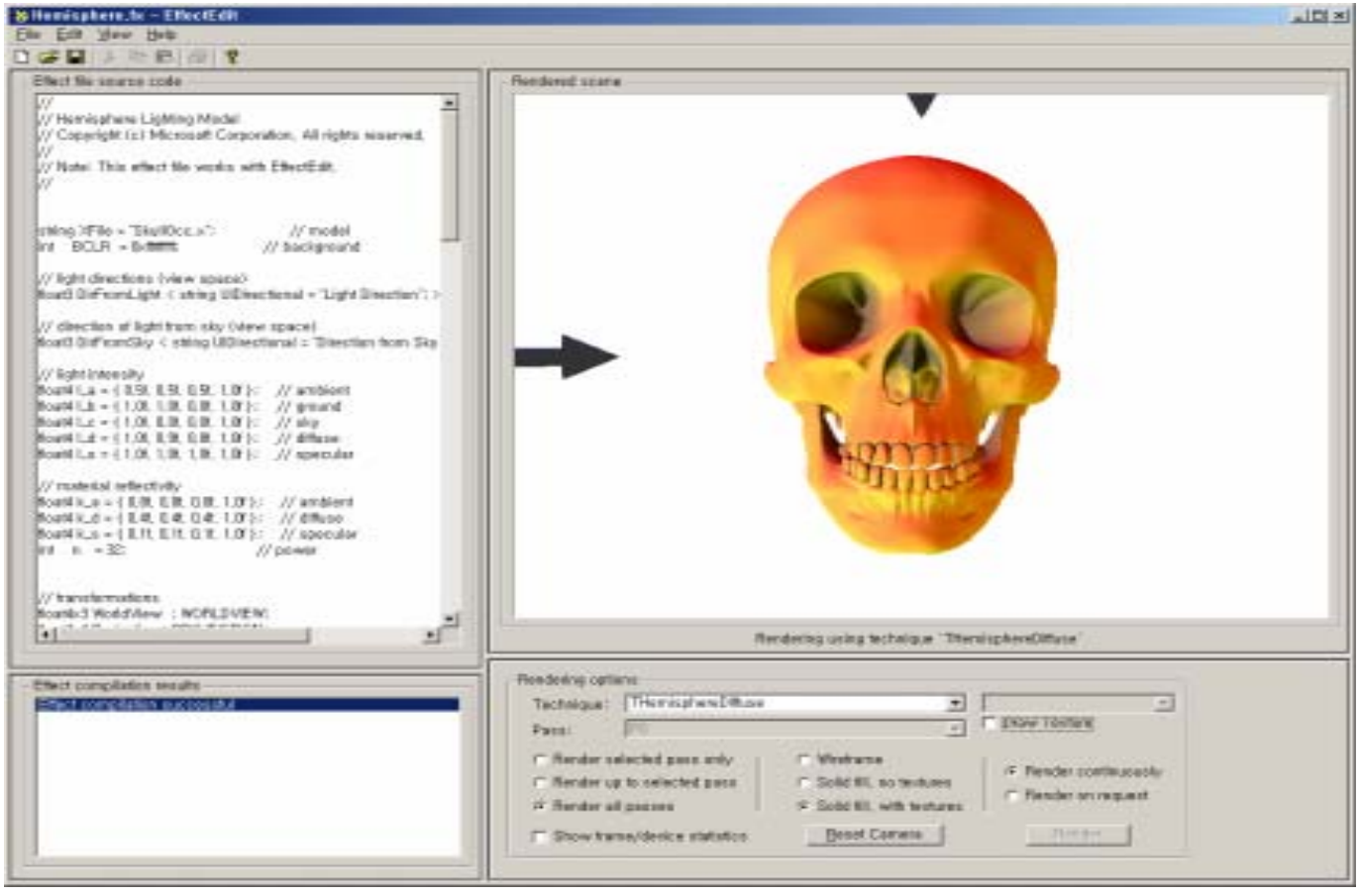
반구 조명 + 확산 조명



반구 조명 + 확산 조명의 색합성 과정

조명 모델

- 반구 조명 + 확산 조명의 실행 결과



조명 모델

- 반구 조명과 직접 조명 기법의 비교

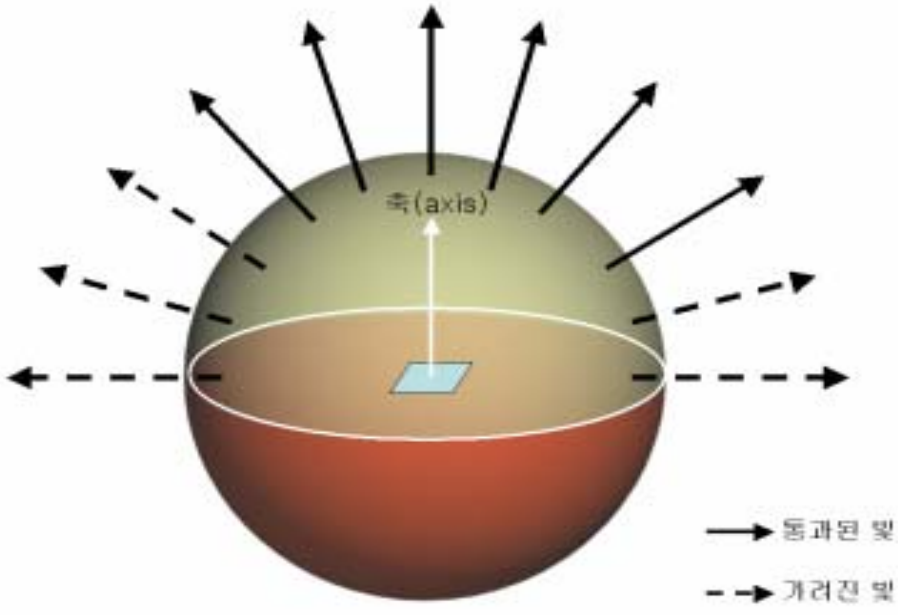
주변 조명 + 확산 조명

반구 조명

반구 조명 + 확산 조명

조명 모델

- 반구 조명 응용
  - 반구 조명을 진정한 전역 조명 모델이라고 하기는 무리이다.
  - 천반구, 지반구, 확산광 이외에도 주변 면들의 색에 의해서 영향을 받아야 하기 때문이다.
  - 실세계는 차폐물(occluder)이 의해 가려지는 면 처리를 해야 한다.



조명 모델

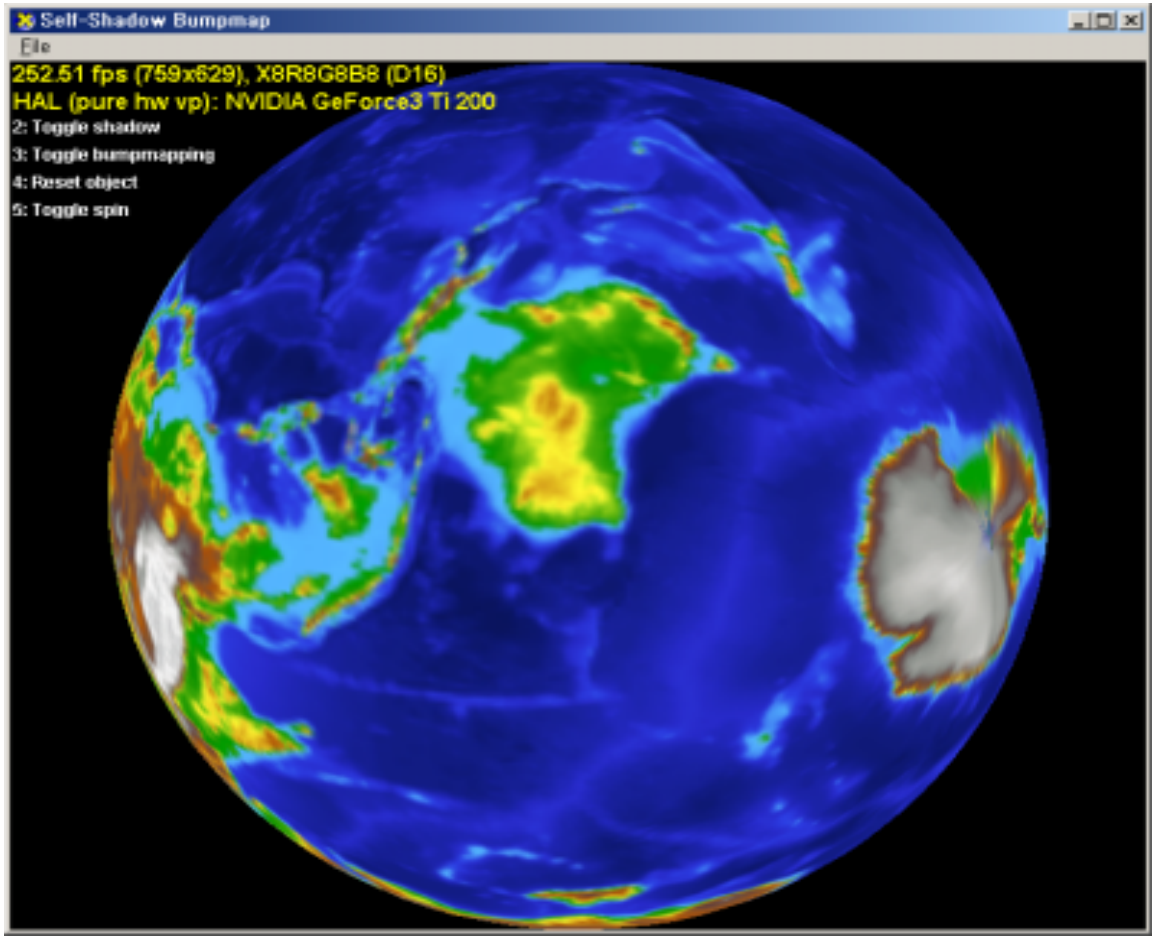
- 가림 처리 (masking)의 비교

가림 처리 없음

가림 처리 있음

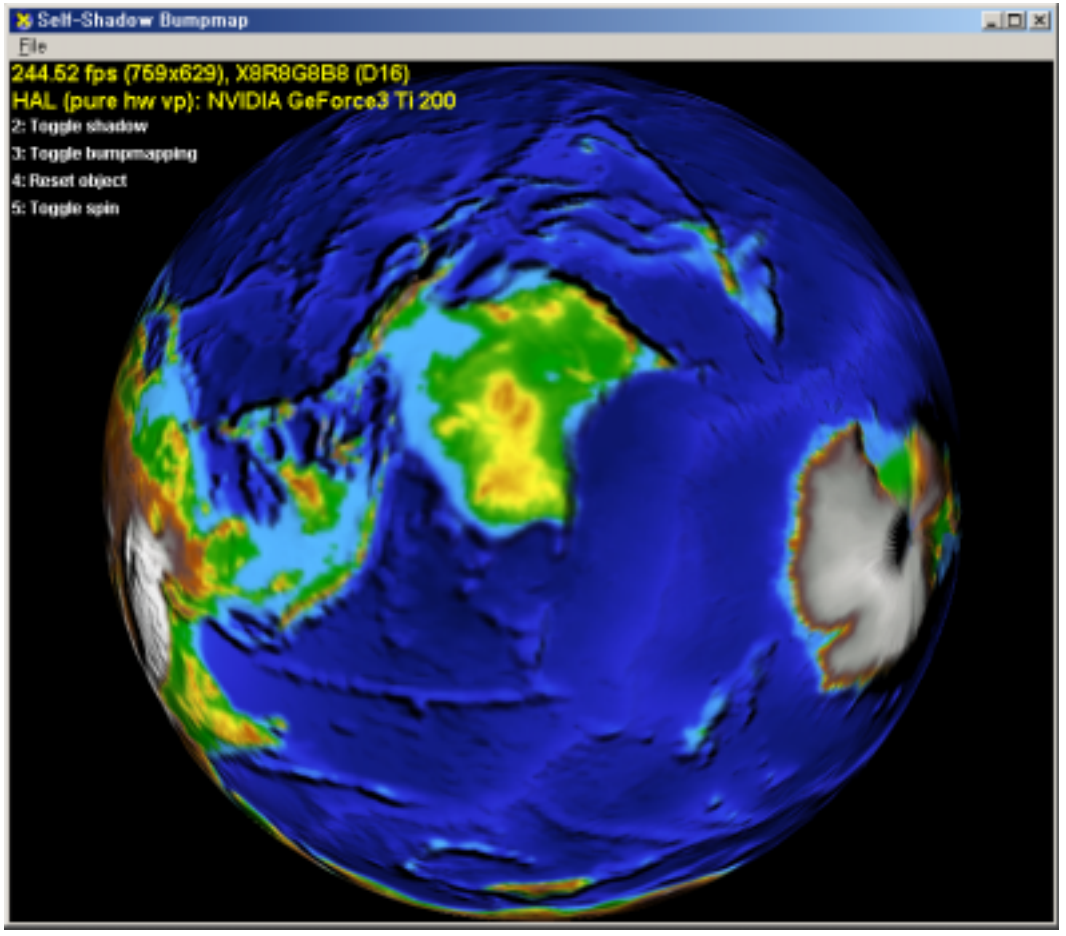
DX의 BumpSelfShadow 예제

- 가림 처리(masking)를 이용한 범프 맵핑



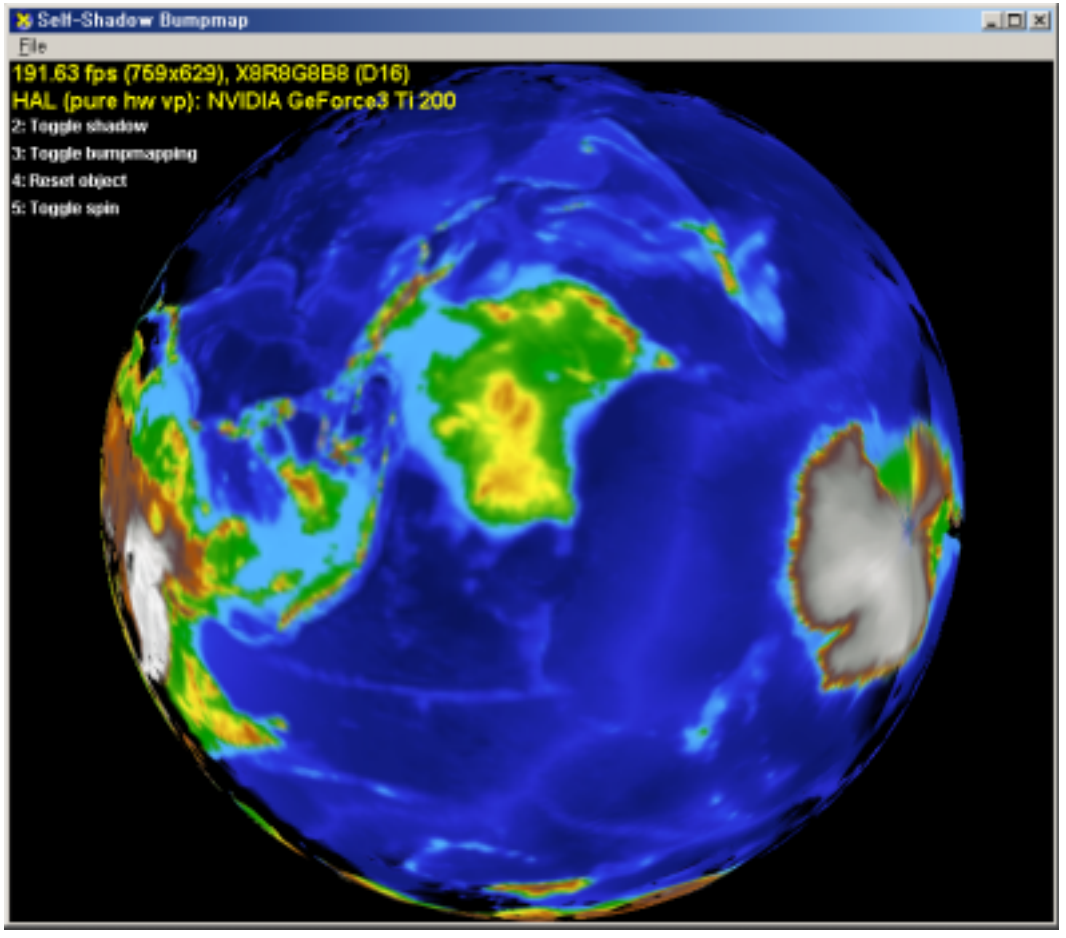
구

조명 모델


Self-Shadow Bumpmap
File
244.52 fps (769x629), X8R8G8B8 (D16)
HAL (pure hw vp): NVIDIA GeForce3 Ti 200
2: Toggle shadow
3: Toggle bumpmapping
4: Reset object
5: Toggle spin

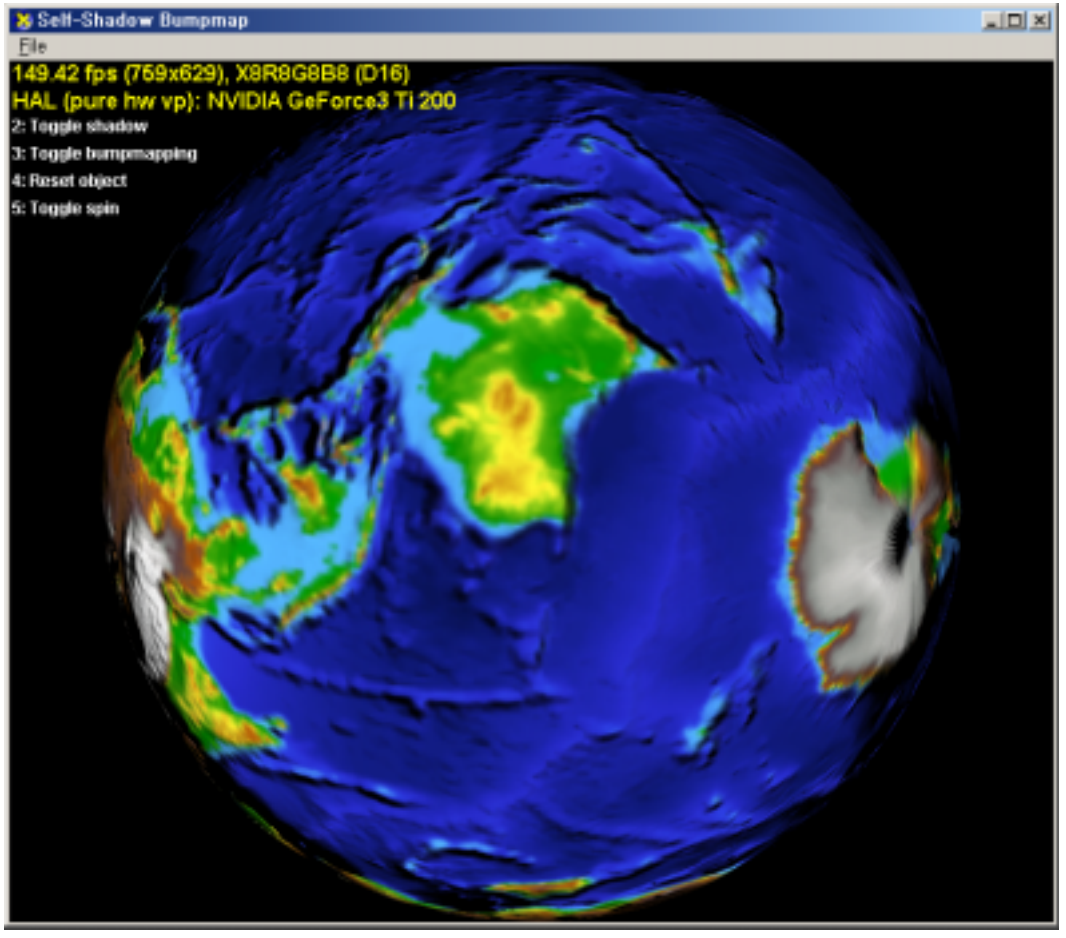
구 + 범프 맵

조명 모델


Self-Shadow Bumpmap
File
191.63 fps (769x629), X8R8G8B8 (D16)
HAL (pure hw vp): NVIDIA GeForce3 Ti 200
2: Toggle shadow
3: Toggle bumpmapping
4: Reset object
5: Toggle spin

구 + 그림자 맵

조명 모델



구 + 범프 맵 + 그림자 맵