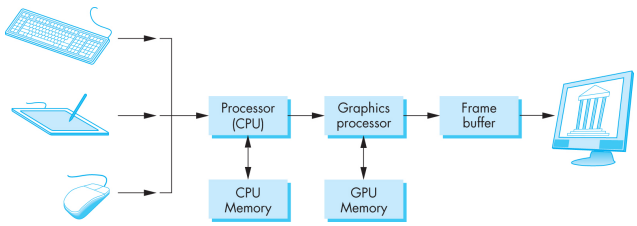
1 Graphics Systems and Models

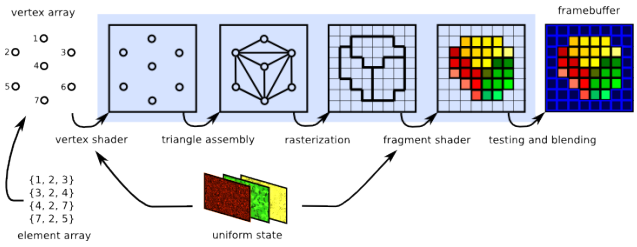
그래픽스 시스템은 컴퓨터 시스템의 일부. 그래픽스는 3차원의 빛을 2차원 디스플레이에 표현하는 것.



프레임버퍼는 그래픽스 시스템의 핵심 요소로 각 픽셀의 정보(색상, 명암 등)에 대한 정보를 디스플레이 크기만큼 저장하고 있음. 즉, 화면 정보를 변경하는 것은 프레임버퍼를 변경하는 것. 커서를 움직이면 프레임버퍼 내용이 바뀐다.

프레임버퍼에 내용을 그릴 때마다 즉시 화면을 갱신하면 깜박임이 생김. 화면을 그릴 때 이전 프레임을 지우는 과정이 보이기 때문. 그래서 별도의 버퍼(back buffer)에 화면을 다 그려놓고 front buffer와 교체(swap)한다. 즉, back buffer는 쓰기 전용, front buffer는 읽기 전용으로 사용. 이를 double buffering 이라고 함.

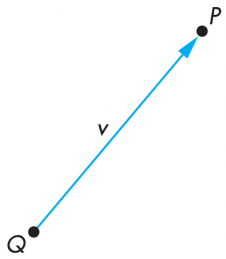
오늘날 대부분의 그래픽스 시스템은 래스터 베이스. 이미지가 프레임버퍼에 있는 픽셀의 배열로 만들어짐. 해상도는 프레임 버퍼에 있는 픽셀의 개수에, 선명도는 각 픽셀이 얼마나 많은 색을 표현할 수 있는지(사용하는 비트 수)에 따른다. 1비트는 2가지 색, 8비트는 256가지 색(RGB 당 8비트). 요즘엔 10비트, 12비트도 쓰인다. 그래픽스 파이프라인은 3차원 개체를 2차원 래스터 이미지로 표현하기 위한 일련의 처리 과정.



1. vertex processor: 정점의 위치와 카메라의 위치, 원근법 등을 조정한다. 3차원 상의 점을 2차원 상의 점으로 표현한다.
2. clipper and primitive assembler: 카메라에 보이지 않는 물체를 잘라내고(clipping) 각 정점을 이어 객체를 구성(primitive assembly)한다.
3. rasterizer: 객체가 픽셀의 배열(raster)로 이뤄진 화면에서 어떤 픽셀에 해당하는지 계산하여 프래그먼트 집합을 구성한다.
4. fragment processor: 프래그먼트에 색을 표현한다. 음영이나 깊이감이 반영된다.

대부분 하드웨어에 구현되어 있기 때문에 우리는 vertex processor(vertex shader)와 gragment processor(fragment shader)에 개입한다.

2 Geometry



기하학의 기본적인 3 요소는 스칼라, 벡터, 점.

- 스칼라: 크기
- 벡터: 방향과 길이
  - 모든 벡터는 인버스를 갖는다: $\mathbf{v}$, $-\mathbf{v}$
  - 모든 벡터에는 스칼라를 곱할 수 있다: $\alpha\mathbf{v}$ (α가 0이라면 영벡터)
  - 영벡터: 길이 0, 방향이 정해지지 않은 벡터.
  - 어떤 두 벡터의 합은 벡터다: $\mathbf{u} + \mathbf{w} = \mathbf{v}$
  - 벡터 공간에서 벡터에는 위치가 없기 때문에 방향과 길이가 같다면 어디에 있든 모두 같은 벡터.
- 점: 공간 위의 어떤 위치
  - 벡터는 두 점의 위치 차를 표현한 것.
  - 따라서 점-점 뺄셈의 결과는 벡터: $\mathbf{v} = P - Q$, $P = Q + \mathbf{v}$

2.1 Affine Spaces

벡터 공간을 확장해 점과 벡터가 관계를 맺는 공간. 원점이 어디인지 모르는 벡터 공간이기도 하다. 벡터 공간과 달리 아핀 공간에는 위치를 특정하는 점에 대한 개념이 있음. 따라서 아래와 같은 연산이 가능하다.

- 벡터-벡터 덧셈 → 벡터
- 벡터-스칼라 곱셈 → 벡터
- 점-점 뺄셈 → 벡터
- 점-벡터 덧셈 → 점
- 점-점 덧셈 → ???

모든 점은 $P(\alpha) = P_0 + \alpha d$와 같이 표현 가능하다. 이때 P_0는 임의의 점, d는 임의의 벡터, α는 스칼라다. α가 주어지면 직선을 얻을 수 있으며, 여기서 α는 직선과 관계없이 '매개'하는 역할만 하므로 이러한 표현은 parametric form. 직선을 표현하는 경우:

- explicit: $y = mx + h$, x를 알면 y를 구할 수 있음. y축과 평행한 직선은 표현 불가.
- implicit: $ax + by + c = 0$, x, y를 알면 직선 위의 점인지 알 수 있음.
- parametric: $x(\alpha) = \alpha x_0 + (1-\alpha)x_1$, $y(\alpha) = \alpha y_0 + (1-\alpha)y_1$

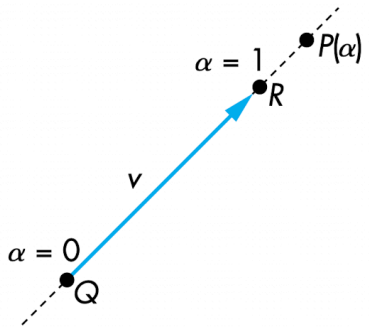
원의 방정식의 경우 implicit form으로 $x^2 - y^2 = 1$인데, parametric으로는 $x = \cos\theta, y = \sin\theta$임.

2.2 Affine Sums

원래 점-점 덧셈과 점-스칼라 곱셈은 불가능하지만 아핀 합으로는 할 수 있다.

- 각 점들 앞의 계수(α) 합이 1일 때 점-점 덧셈이 허용.
- n개의 점의 덧셈에서 계수의 합이 1이 되는 경우를 아핀 합이라고 한다.

어떤 점 Q, 벡터 $\mathbf{v}$, 스칼라 α가 있을 때,



점 Q에서 $\mathbf{v}$ 방향의 직선 위에 있는 모든 점을 아래와 같은 parametric form으로 표현할 수 있다.

$$P = Q + \alpha\mathbf{v}$$

이때 다음을 만족하는 점 R을 항상 찾을 수 있다.

$$\mathbf{v} = R - Q$$
$$\therefore P = Q + \alpha(R - Q) = \alpha R + (1-\alpha)Q$$

이는 점-점 덧셈으로 볼 수도 있다.

$$P = \alpha_1 R + \alpha_2 Q \quad \text{where} \quad \alpha_1 + \alpha_2 = 1$$

따라서 아핀 공간에서는 점-점 덧셈이 가능하다.

2.3 Convexity

convex는 개체 내 임의의 두 점을 선택했을 때 두 점을 이은 선이 해당 개체 안에 포함되는 것. n개의 점 $P_1, P_2, \cdots, P_n$으로 정의되는 개체에 대한 아핀 합은 아래와 같다.

$$P = \alpha_1 P_1 + \alpha_2 P_2 + \cdots + \alpha_n P_n$$

다음 조건을 만족할 때 위의 식이 의미를 가짐을 귀납적으로 보일 수 있다.

$$\alpha_1 + \alpha_2 + \cdots + \alpha_n = 1$$

이를 만족하는 점 P들은 모두 개체 안에 포함된다.

convex hull은 점 $P_1, P_2, \cdots P_n$들을 모두 포함하는 최소의 convex 개체. $\alpha_i >= 0$이면 $P_1, P_2, \cdots P_n$의 convex hull이 된다.

삼각형은 convex하기 때문에 내부에 있는 어떤 점을 아핀 합으로 표현할 수 있다

$$P(\alpha_1, \alpha_2, \alpha_3) = \alpha_1 P + \alpha_2 Q + \alpha_3 R$$
$$\text{where} \quad \alpha_1 + \alpha_2 + \alpha_3 = 1, \quad \alpha_i >= 0$$

후술할 무게 중심과 관련있다.

2.4 Dot and Cross Products

$|\mathbf{v}|$는 벡터 $\mathbf{v}$의 길이.

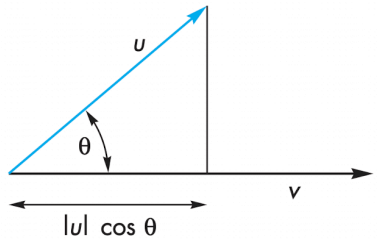
- $|\mathbf{v}| = \sqrt{\mathbf{v}_x{}^2 + \mathbf{v}_y{}^2 + \mathbf{v}_z{}^2}$
- $\mathbf{v} \cdot \mathbf{v} = |\mathbf{v}|^2$

Dot(inner) Product

$$\mathbf{u} \cdot \mathbf{v}$$

두 벡터 사이의 스칼라(두 벡터의 사이각 등)를 구할 때, 폴리곤의 안팎을 구별할 때 사용. $\mathbf{u} \cdot \mathbf{v} = 0$이라면 두 벡터는 수직이다.

아래와 같은 두 벡터 사이각의 코사인은

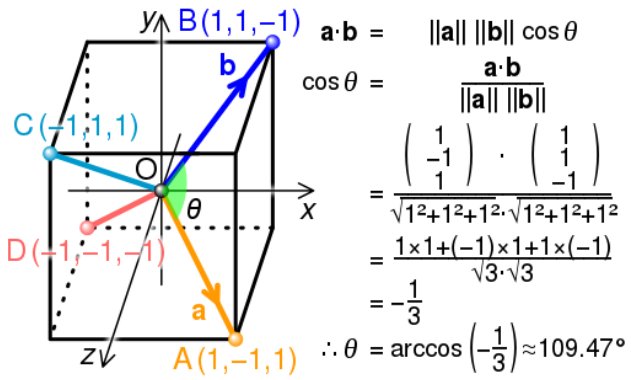


$$\mathbf{u} \cdot \mathbf{v} = |\mathbf{u}||\mathbf{v}| \cos\theta$$
$$\therefore \cos\theta = \frac{\mathbf{u} \cdot \mathbf{v}}{|\mathbf{u}||\mathbf{v}|}$$

코사인 값은 두 벡터가 가까워지면 1, 멀어지면 0(수직)이 된다. 피사체가 카메라 앞에 있는지, 뒤에 있는지 계산: 카메라의 정면으로 향하는 벡터 $\mathbf{u}$와 피사체로 향하는 벡터 $\mathbf{v}$가 있을 때 $\frac{\mathbf{u} \cdot \mathbf{v}}{|\mathbf{u}||\mathbf{v}|}$가 0보다 크면 앞, 0보다 작으면 뒤에 있는 것. (∵ 앞뒤는 180도를 기준으로 판단, -90도에서 90도 사이에 피사체가 있다면 앞에 있는 것이므로.)

예시: $\mathbf{u} = (1, 2, 3), \mathbf{v} = (4, -5, 6)$

$$\begin{aligned}\mathbf{u} \cdot \mathbf{v} &= \mathbf{u}_1\mathbf{v}_1 + \mathbf{u}_2\mathbf{v}_2 + \mathbf{u}_3\mathbf{v}_3 \\ &= 1(4) + 2(-5) + 3(6) \\ &= 12\end{aligned}$$

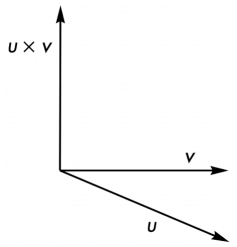


$$\mathbf{a}\cdot\mathbf{b} = \|\mathbf{a}\|\,\|\mathbf{b}\|\cos\theta$$
$$\cos\theta = \frac{\mathbf{a}\cdot\mathbf{b}}{\|\mathbf{a}\|\,\|\mathbf{b}\|}$$
$$= \frac{\begin{pmatrix}1\\-1\\1\end{pmatrix}\cdot\begin{pmatrix}1\\1\\-1\end{pmatrix}}{\sqrt{1^2+1^2+1^2}\cdot\sqrt{1^2+1^2+1^2}}$$
$$= \frac{1\times1+(-1)\times1+1\times(-1)}{\sqrt{3}\cdot\sqrt{3}}$$
$$= -\frac{1}{3}$$
$$\therefore \theta = \arccos\left(-\frac{1}{3}\right) \approx 109.47°$$

Cross(outer) Product

$$\mathbf{u} \times \mathbf{v}$$

3차원에서 두 벡터와 수직인 벡터를 구할 때, 한 평면의 normal vector를 구할 때 사용.



$$\mathbf{u} \times \mathbf{v} = |\mathbf{u}||\mathbf{v}| \sin\theta$$
$$\therefore \sin\theta = \frac{\mathbf{u} \times \mathbf{v}}{|\mathbf{u}||\mathbf{v}|}$$

$\mathbf{v} = (v_1, v_2, v_3), \mathbf{w} = (w_1, w_2, w_3)$인 경우 $\mathbf{v} \times \mathbf{w}$:

$$\begin{aligned}\mathbf{v} \times \mathbf{w} &= \begin{bmatrix}\mathbf{i} & \mathbf{j} & \mathbf{k} \\ v_1 & v_2 & v_3 \\ w_1 & w_2 & w_3\end{bmatrix} \\ &= (v_2w_3 - v_3w_2)\mathbf{i} + (v_3w_1 - v_1w_3)\mathbf{j} \\ &\quad + (v_1w_2 - v_2w_1)\mathbf{k}\end{aligned}$$

사실 $\mathbf{j}$의 스칼라 부분 연산은 $(v_3w_1 - v_1w_3)\mathbf{j}$로 순서가 반대이지만, 시각적으로 연산 순서를 기억하기 쉽도록 $\mathbf{j}$의 부호를 음의 부호로 바꿔서 표현할 수 있다.

$$(v_2w_3 - v_3w_2)\mathbf{i} \underline{-(v_1w_3 - v_3w_1)\mathbf{j}} + (v_1w_2 - v_2w_1)\mathbf{k}$$

예시: $\mathbf{v} = (3, -3, 1), \mathbf{w} = (4, 9, 2)$

$$\begin{aligned}\mathbf{v} \times \mathbf{w} &= \begin{bmatrix}\mathbf{i} & \mathbf{j} & \mathbf{k} \\ 3 & -3 & 1 \\ 4 & 9 & 2\end{bmatrix} \\ &= [-3(2) - 1(9)]\mathbf{i} - [3(2) - 1(4)]\mathbf{j} \\ &\quad + [3(9) - -3(4)]\mathbf{k} \\ &= -15\mathbf{i} - 2\mathbf{j} + 39\mathbf{k}\end{aligned}$$

예시: 점 $P(0,0,0), Q(2,4,6), R(-1,2,7)$로 이뤄진 평면의 법선 벡터는 벡터 $Q-P$와 벡터 $R-P$에 공통으로 수직인 벡터.

$$\mathbf{v} = Q - P = (2,4,6) - (0,0,0) = (2,4,6)$$
$$\mathbf{w} = R - P = (-1,2,7) - (0,0,0) = (-1,2,7)$$

$$\begin{aligned}\mathbf{v} \times \mathbf{w} &= \begin{bmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ 2 & 4 & 6 \\ -1 & 2 & 7 \end{bmatrix} \\ &= [4(7) - 6(2)]\mathbf{i} - [2(7) - 6(-1)]\mathbf{j} \\ &\quad + [2(2) - 4(-1)]\mathbf{k} \\ &= 16\mathbf{i} - 20\mathbf{j} + 8\mathbf{k} \\ &= (16, -20, 8)\end{aligned}$$

2.5 Planes

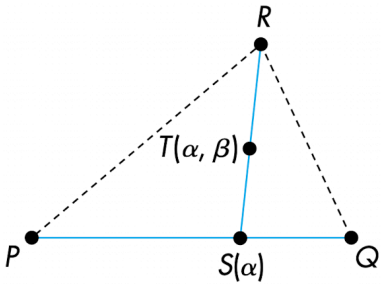
평면은 하나의 점과 두 개의 벡터, 또는 세 개의 점으로 정의할 수 있다. 하나의 점, 두 개의 벡터로 정의하는 경우

$$P(\alpha, \beta) = R + \alpha\mathbf{u} + \beta\mathbf{v}$$

세 개의 점으로 정의하는 경우

$$P(\alpha, \beta) = R + \alpha(Q - R) + \beta(P - Q)$$

아래와 같이 아핀 공간의 점 P, Q, R로 정의된 삼각형이 있을 때,



점 P, Q를 잇는 선 위에 있는 임의의 점 S는

$$S(\alpha) = \alpha P + (1 - \alpha)Q, \quad 0 \le \alpha \le 1$$

점 S, R을 잇는 선 위에 있는 임의의 점 T는

$$T(\beta) = \beta S + (1 - \beta)R, \quad 0 \le \beta \le 1$$

점 S, T는 α, β로 결정되며, 면은 P, Q, R로 결정되므로 아래와 같은 삼각형 평면의 방정식을 얻을 수 있다.

$$\begin{aligned}T(\alpha, \beta) &= \beta[\alpha P + (1 - \alpha)Q] + (1 - \beta)R \\ &= P + \beta(1 - \alpha)(Q - P) + (1 - \beta)(R - P)\end{aligned}$$

$Q - P$와 $R - P$는 임의의 벡터이므로, 이를 각각 $\mathbf{u}, \mathbf{v}$로 바꾸면 하나의 점과 두 개의 벡터로 평면을 표현하는 셈이 된다.

$$\begin{aligned}T(\alpha, \beta) &= P_0 + \alpha\mathbf{u} + \beta\mathbf{v} \\ &= \beta\alpha P + \beta(1 - \alpha)Q + (1 - \beta)R \\ T(\alpha, \beta', \gamma) &= \alpha' P + \beta' Q + \gamma' R \\ \text{where} \quad & \alpha' + \beta' + \gamma' = 1\end{aligned}$$

이때 $(\alpha', \beta', \gamma')$을 무게중심 좌표(barycentric coordinate)라고 한다. 삼각형 내에 있는 점 T에 대한 barycentric coordinate 표현은

$$\begin{aligned}T &= \beta\mathbf{u} + \gamma\mathbf{v} + P \\ &= \beta(Q - P) + \gamma(R - P) + P \\ &= \underbrace{(1 - \beta - \gamma)}_{\alpha} P + \beta Q + \gamma R\end{aligned}$$

이때 무게중심 좌표는 (α, β, γ). 앞서 본 $T(\alpha, \beta) = P_0 + \alpha\mathbf{u} + \beta\mathbf{v}$로부터 $\mathbf{u} \times \mathbf{v}$(평면에 직교하는 normal vector)를 구하면 평면을 표현하는 또 다른 방정식을 얻을 수 있다.

$$\mathbf{u} \times \mathbf{v} \cdot (P - P_0) = 0$$

3 3D Object Modeling

3.1 Polygonal Representation

boundary representations은 외부에 보이는 표면만 모델링하는 방식. 점을 이어서 망(mesh)를 만든다. 직관적이고 단순한 방법으로 대부분 이 방식으로 모델링한다. 사용자 친화적이지 않고 기계 친화적이라는

단점이 있음.

보통 폴리곤은 삼각형으로 모델링한다. 삼각형은 완전히 평면적임을 보장할 수 있고, cross product로 평면이 어느 쪽을 바라보고 있는지 쉽게 구할 수 있기 때문. 삼각형을 사용하면 임의의 점 3개로 유니크한 평면을 만들 수 있다. 단, 점 3개가 같은 직선 상에 있다면, 또는 두 벡터가 같은 방향이라면 유니크한 평면이 불가능하다.

만약 점 4개를 선택한다면 유니크한 평면을 만들지 못하고 사각형이 아니라 삼각뿔을 만들게 될 수 있다.

폴리곤을 어떻게 저장할까?

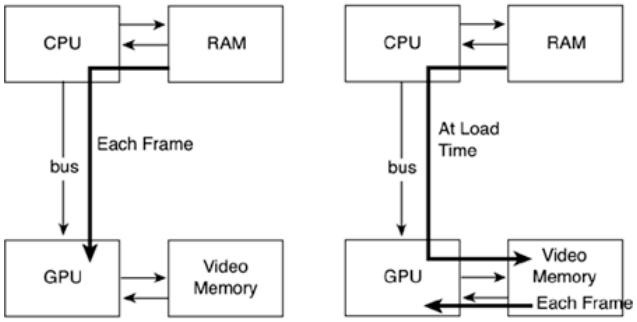
- Face Set (STL): 삼각형을 구성하는 3개의 점 좌표를 모두 저장하는 방식.
  - 같은 점에 대한 중복 데이터를 저장하게 되는 문제.
  - 한 점의 위치를 변경하면 그 점을 참조하는 삼각형 전체의 위치가 바뀜.
- Shared Vertex (OBJ, OFF): 점과 삼각형 정보를 따로 저장하는 방식.
  - vertices 테이블에는 점의 x, y, z 좌표, triangles 테이블에는 하나의 삼각형을 구성하는 3개의 점을 참조.
  - triangles:vertices를 1:N 데이터베이스로 구성하는 형태.

3.2 Other Representations

- Sweep: 평면을 회전시켜 입체 도형을 표현. 쉽게 안팎을 구별할 수 있음.
- Fractal: 프랙탈로 표현. 멀리서는 물체가 한 덩어리로 보이다가 가까워질수록 정교하게 보임.
- Constructive Solid Geometry: 집합 연산(union, intersection, difference)를 이용해 도형을 표현. 겉부분이 아니라 보이지 않는 내부까지 구현해야 할 때 사용.
- Quadtree: 트리로 표현. 트리가 깊어질수록 정교한 표현. 중간 정도에서는 개략적인 모양을 얻을 수 있음.
- Voxel: 2차원 평면을 쌓아 부피를 만들어 표현.
- Implicit Surface: $f(\mathbf{p}) = Q$를 만족하는 점의 집합으로 표현.

3.3 Vertex Array and Buffer

매 프레임마다 RAM으로부터 CPU를 거쳐 GPU로 데이터를 불러오면 CPU와 CPU-GPU 사이 버스에 오버헤드가 커짐. 특히 프레임 사이 변경이 크게 없다면 불필요한 데이터를 계속 불러오는 것. 그래서 프로그램을 로드할 때 GPU에 데이터를 보내두고 매 프레임 데이터는 GPU가 처리.



- Vertex Buffer Object(VBO): vertex data를 받아 GPU 메모리에 전달하기 위한 버퍼.
- Vertex Array Object(VAO): vertex buffer와 연결된 배열. vertex attributes를 담고 있으며, vertex shader가 이 attributes를 참조해 렌더링을 한다.

아래와 같은 과정을 거친다.

1. 새로운 VBO 생성
2. 버퍼 바인딩
3. 비디오 메모리 공간 할당 및 vertex 데이터를 버퍼 객체에 복사
4. 새로운 VAO 생성
5. 배열 바인딩
6. vertex attribute 배열 정의 및 VBO와 VAO 연결.

4 2D Geometric Transformation

2D 평면을 변형(이동: translation, 회전: rotation, 신축: scaling)한다. 어떤 점 $\mathbf{p}$를 2D에서는 $\mathbf{p} = (x, y)$, 3D에서는 $\mathbf{p} = (x, y, z)$로 표현한다. 이를 컬럼 벡터로 표현할 수도 있다.

$$\mathbf{p} = \begin{bmatrix} x \\ y \end{bmatrix} \quad \text{or} \quad \mathbf{p} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

2D 변형을 행렬 M으로 표현하여 컬럼 벡터 $\mathbf{p}$에 적용하면: $\mathbf{p}' = M\mathbf{p}$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ax + by \\ cx + dy \end{bmatrix}$$

4.1 Translation

점 $\mathbf{p}$를 $\mathbf{t}$만큼 이동시켜 점 $\mathbf{p}'$으로 변형한다: $\mathbf{p}' = \mathbf{p} + \mathbf{t}$

$$\mathbf{p}' = \begin{bmatrix} x' \\ y' \end{bmatrix} \quad \mathbf{p} = \begin{bmatrix} x \\ y \end{bmatrix} \quad \mathbf{t} = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

$$\mathbf{p}' = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \end{bmatrix}$$

4.2 Rotation

점 $\mathbf{p}$를 θ만큼 회전시켜 점 $\mathbf{p}'$으로 변형: $\mathbf{p}' = R(\theta)\mathbf{p}$

$$\mathbf{p}' = \begin{bmatrix} x' \\ y' \end{bmatrix} \quad \mathbf{p} = \begin{bmatrix} x \\ y \end{bmatrix} \quad R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

$$\mathbf{p}' = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta \end{bmatrix}$$

4.3 Scaling

점 $\mathbf{p}$를 x축으로 s_x배, y축으로 s_y배만큼 신축해 점 $\mathbf{p}'$으로 변형한다: $\mathbf{p}' = S(s_x, s_y)\mathbf{p}$

$$\mathbf{p}' = \begin{bmatrix} x' \\ y' \end{bmatrix} \quad \mathbf{p} = \begin{bmatrix} x \\ y \end{bmatrix} \quad S(s_x, s_y) = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$$

$$\mathbf{p}' = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} s_x x \\ s_y y \end{bmatrix}$$

4.4 Homogeneous Coordinates

앞서 translation은 scaling, rotation과 다르게 행렬 합으로 표현했다. 그런데 homogeneous coordination을 사용하면 모든 변형을 행렬 곱으로 표현할 수 있다.

2D 좌표계 (x, y)를 확장해 세 개의 요소로 표현할 수 있음: (x_h, y_h, h). 이를 다시 cartesian coordinates로 표현하면: $x = \frac{x_h}{h}$, $y = \frac{y_h}{h}$ 일반적으로 h는 1로 둔다. $(x_h, y_h, 0)$이면 점의 좌표가 무한하다는 의미. $(2, 1, 1) = (4, 2, 2) = (6, 3, 3)$. $(0, 0, 0)$은 불가능.

Translation

$$\mathbf{p}' = T(t_x, t_y)\mathbf{p}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \\ 1 \end{bmatrix}$$

$$T^{-1}(t_x, t_y) = T(-t_x, -t_y)$$

$$T(t_{x2}, t_{y2}) \cdot T(t_{x1}, t_{y1}) = T(t_{x1} + t_{x2}, t_{y1} + t_{y2})$$

Rotation

$$\mathbf{p}' = R(\theta)\mathbf{p}$$

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

$$R^{-1}(\theta) = R(-\theta)$$

$$R(\theta_2) \cdot R(\theta_1) = R(\theta_1 + \theta_2)$$

Scaling

$$\mathbf{p}' = S(s_x, s_y)\mathbf{p}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$S^{-1}(s_x, s_y) = S(\frac{1}{s_x}, \frac{1}{s_y})$$

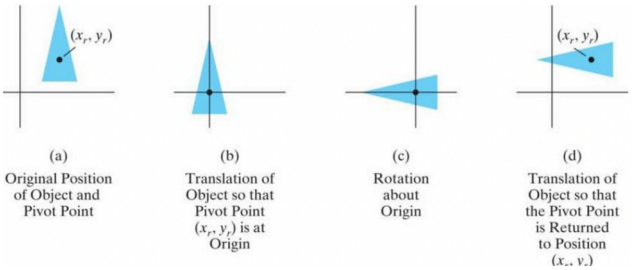
$$S(s_{x2}, s_{y2}) \cdot S(s_{x1}, s_{y1}) = S(s_{x1} \cdot s_{x2}, s_{y1} \cdot s_{y2})$$

4.5 Pivot-Point Transformations

원점에서 변형하는 것과 달리 특정 점을 기준으로 회전, 신축 변환하려면 절차가 더 필요하다. 일단 개체를 원점으로 옮겨서 변환하고, 다시 원래 자리로 돌려 놓아야 함. 주의: $\mathbf{v}$에 A, B, C 순서로 적용한다면 $C(B(A\mathbf{v})) = CBA(\mathbf{v})$이다.

Rotation

(a) 개체의 원위치는 (x_r, y_r).



(b) 개체를 좌표계의 원점으로 이동(translation).

(c) 개체를 회전(rotation).

(d) 개체를 다시 원위치로 이동(translation).

$$R(x_r, y_r, \theta)$$

$$= \overbrace{T(x_r, y_r)}^{(d)} \cdot \overbrace{R(\theta)}^{(c)} \cdot \overbrace{T(-x_r, -y_r)}^{(b)}$$

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

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

$$= \begin{bmatrix} \cos\theta & -\sin\theta & -x_r\cos\theta + y_r\sin\theta + x_r \\ \sin\theta & \cos\theta & -x_r\sin\theta - y_r\cos\theta + y_r \\ 0 & 0 & 1 \end{bmatrix}$$

예시: 점 $A(4,6), B(2,2), C(6,2)$으로 정의된 삼각형 T를 고정점 $(3,3)$ 기준 반시계 방향으로 90도 회전시켰을 때 T'이라고 하면

$$R(3,3,\frac{\pi}{2}) = T(3,3) \cdot R(\frac{\pi}{2}) \cdot T(-3,-3)$$

$$= \begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & 3 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\frac{\pi}{2} & -\sin\frac{\pi}{2} & 0 \\ \sin\frac{\pi}{2} & \cos\frac{\pi}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -3 \\ 0 & 1 & -3 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\frac{\pi}{2} & -\sin\frac{\pi}{2} & 3 \\ \sin\frac{\pi}{2} & \cos\frac{\pi}{2} & 3 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -3 \\ 0 & 1 & -3 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\frac{\pi}{2} & -\sin\frac{\pi}{2} & -3\cos\frac{\pi}{2} + 3\sin\frac{\pi}{2} + 3 \\ \sin\frac{\pi}{2} & \cos\frac{\pi}{2} & -3\sin\frac{\pi}{2} - 3\cos\frac{\pi}{2} + 3 \\ 0 & 0 & 1 \end{bmatrix}$$

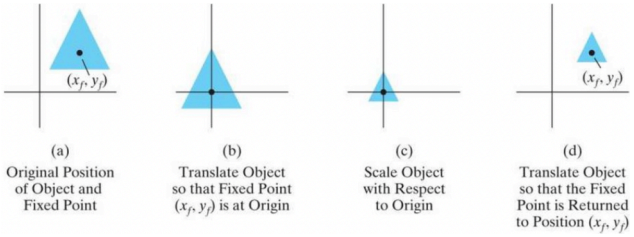
$$= \begin{bmatrix} 0 & -1 & 6 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\therefore T' = R(3,3,\frac{\pi}{2}) \cdot T$$

$$= \begin{bmatrix} 0 & -1 & 6 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 4 & 2 & 6 \\ 6 & 2 & 2 \\ 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 4 & 4 \\ 4 & 2 & 6 \\ 1 & 1 & 1 \end{bmatrix} \quad \begin{aligned} A' &= (0,4) \\ B' &= (4,2) \\ C' &= (4,6) \end{aligned}$$

Scaling



(a) 개체의 원위치는 (x_f, y_f).

(b) 개체를 좌표계의 원점으로 이동(translation).

(c) 개체를 확대/축소(scaling).

(d) 개체를 다시 원위치로 이동(translation).

$$S(x_f, y_f, s_x, s_y)$$

$$= \overbrace{T(x_f, y_f)}^{(d)} \cdot \overbrace{S(s_x, s_y)}^{(c)} \cdot \overbrace{T(-x_f, -y_f)}^{(b)}$$

$$= \begin{bmatrix} 1 & 0 & x_f \\ 0 & 1 & y_f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_f \\ 0 & 1 & -y_f \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} s_x & 0 & x_f \\ 0 & s_y & y_f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_f \\ 0 & 1 & -y_f \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} s_x & 0 & -s_x x_f + x_f \\ 0 & s_y & -s_y y_f + y_f \\ 0 & 0 & 1 \end{bmatrix}$$

4.6 Other Transformations

Reflection

x축을 기준으로 반사: $\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

y축을 기준으로 반사: $\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

원점을 기준으로 반사: $\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

직선 $y = x$를 기준으로 반사: $\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Shear

직선 $y = y_{ref}$에 x축 방향: $\begin{bmatrix} 1 & sh_x & -sh_x \cdot y_{ref} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

직선 $x = x_{ref}$에 y축 방향: $\begin{bmatrix} 1 & 0 & 0 \\ sh_y & 1 & -sh_y \cdot x_{ref} \\ 0 & 0 & 1 \end{bmatrix}$

4.7 Change of Coordinate Systems

새로운 좌표축을 그려서 기존 개체를 다루고 싶을 때는 좌표축의 변형 과정을 역으로 하면 된다. 만약 축을 이동, 회전했다면 아래와 같이 한다.

$$T(-x_0, -y_0) = \begin{bmatrix} 1 & 1 & -x_0 \\ 0 & 1 & -y_0 \\ 0 & 0 & 1 \end{bmatrix}$$

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

$$M_{xy,x'y'} = R(-\theta)T(-x_0, -y_0)$$

예시: $(7, 7)$에 있는 점 P는 축이 시계 방향으로 90도$(-\frac{\pi}{2})$ 회전한 뒤에는 $(-7, 7)$로 좌표가 바뀐다.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos -\frac{\pi}{2} & \sin -\frac{\pi}{2} & 0 \\ -\sin -\frac{\pi}{2} & \cos -\frac{\pi}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 7 \\ 7 \\ 1 \end{bmatrix}$$
$$= \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 7 \\ 7 \\ 1 \end{bmatrix} = \begin{bmatrix} -7 \\ 7 \\ 1 \end{bmatrix}$$

새로운 좌표축에서 y축 벡터 $\mathbf{V}$의 단위 벡터는 $\mathbf{v}$.

$$\mathbf{v} = \frac{\mathbf{V}}{|\mathbf{V}|} = (v_x, v_y)$$

예시: $\mathbf{V} = (3, 3, 0)$이라면

$$\mathbf{v} = \frac{\mathbf{V}}{|\mathbf{V}|} = \frac{(3, 3, 0)}{\sqrt{3^2 + 3^2 + 0^2}}$$
$$= (\frac{3}{3\sqrt{2}}, \frac{3}{3\sqrt{2}}, 0)$$
$$= (\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0)$$

x축 벡터는 $\mathbf{v}$를 시계 방향으로 회전한 벡터와 같으므로, $\mathbf{u} = (v_y, -v_x) = (u_x, u_y)$. 이때 y축 벡터 $\mathbf{v}$와 x축 벡터 $\mathbf{u}$는 아래와 같다.

$$\mathbf{v} = \begin{bmatrix} -\sin\theta \\ \cos\theta \end{bmatrix} \quad \mathbf{u} = \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix}$$

따라서 새 축이 회전만 한 것이라면 $R(\theta)$을 아래와 같이 쓸 수 있다. (두 벡터의 길이가 1이라는 점에 유의.)

$$R = \begin{bmatrix} u_x & u_y & 0 \\ v_x & v_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

5 3D Geometric Transformations

5.1 3D Translation

$$P' = T \cdot P$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

5.2 3D Coordinate-Axis Rotation

어떤 좌표축을 중심축으로 회전하는 경우. 3차원 회전은 축이 적어도 3개 이상이다. 이때 중심축이란 회전하지 않는 점이 있는 축.

z-axis Rotation

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

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

x-axis Rotation

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

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

y-axis Rotation

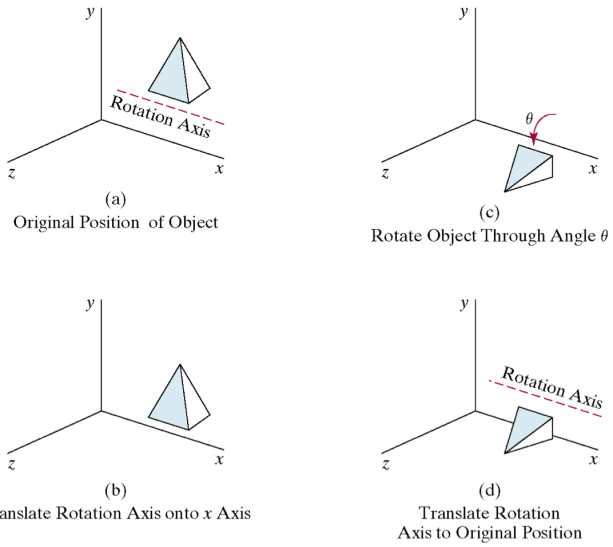
$$P' = R_y(\theta) \cdot P$$

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

시계방향으로 회전한다면 $-\theta$. 참고로 $AA^{-1} = I$, $(TSR)^{-1} = R^{-1}S^{-1}T^{-1}$.

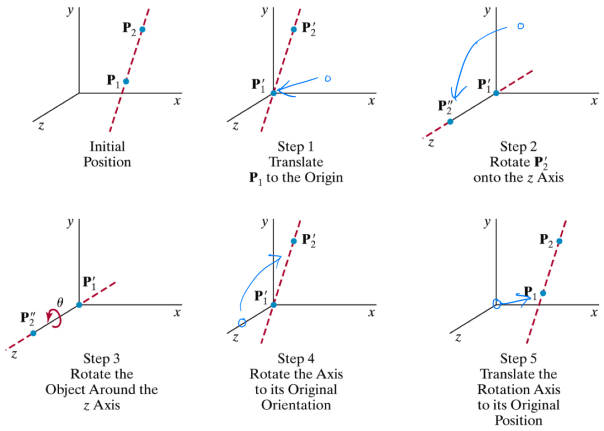
5.3 General 3D Rotation

좌표축이 아니라 임의의 축을 중심으로 회전하는 경우. 점 P를 좌표축으로 옮기고, 회전한 뒤 원래 자리로 옮겨놓는다.



$$P' = T^{-1} \cdot R_x(\theta) \cdot T \cdot PR(\theta) = T^{-1} \cdot R_x(\theta) \cdot T$$

축이 아니라 임의의 방향으로 회전하는 경우. 가령 P_2를 P_1 기준으로 회전하는 경우.



5.4 3D Scailing

원점을 중심으로 크기 변환하는 경우.

$$P' = S \cdot P$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

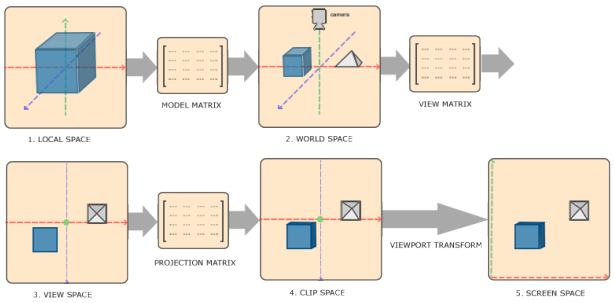
원점이 아니라 고정점을 중심으로 변환하는 경우에는 원점으로 이동, 변환, 다시 제자리로 이동.

$$T(x_f, y_f, z_f) \cdot S(s_x, s_y, s_z) \cdot T(-x_f, -y_f, -z_f) = \begin{bmatrix} s_x & 0 & 0 & (1-s_x)x_f \\ 0 & s_y & 0 & (1-s_y)y_f \\ 0 & 0 & s_z & (1-s_z)z_f \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

6 Viewing

3차원 월드에서 카메라의 시점.

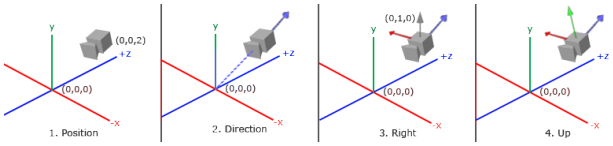
6.1 3D Viewing Pipeline



1. **Model Coordinates**: 개별 개체의 좌표계.
2. Modeling Transform: 모델을 원하는 위치에 배치.
3. **World Coordinates**: 개별 개체가 통합된 세계의 좌표계.
4. Viewing Transform: 월드 좌표계를 카메라 중심으로 조정.
5. **Viewing(Camera) Coordinates**: 카메라로 보는 좌표계.
6. Projection/Normalize Transform: 좌표계 정규화, 원근감 표현.
7. **Normalized Coordinates**: -1에서 1 사이 값을 갖는 표준화된 좌표계.
8. Viewport Transform: 정규화된 좌표를 출력 장치 좌표계에 실제 화면 좌표로 2D 매핑.
9. **Screen Coordinates**: 화면의 좌표계.

6.2 Viewing Coordinate Params

뷰잉 좌표계는 카메라의 위치, 방향, 위쪽 방향에 따라 설정된다. 카메라의 위치 P_0, 카메라의 up vector, 카메라의 방향(-z). 이때 점 P_{ref}를 바라본다고 가정.
normal vector n (z_{view}): 카메라가 바라보는 방향.



카메라의 위치 P_0에서 카메라가 바라보는 대상의 위치 P_{ref}를 빼면 얻을 수 있다. 카메라는 z축의 음의 방향을 가리키고 있다.

$$N = P_0 - P_{\text{ref}}, \quad \therefore n = \frac{N}{|N|}$$

up vector v (y_{view}): 카메라의 위쪽 축. 사용자가 지정한 V'에 따라 결정된다.

$$V' = (V' \cdot n)n + V$$
$$V = V' - (V' \cdot n)n \quad \therefore v = \frac{V}{|V|}$$

right vector u (x_{view}): 카메라의 오른쪽 축.

$$u = v \times n$$

6.3 Viewing Transformation (WC-VC)

월드 좌표계를 카메라 좌표 $P_0 = (x_0, y_0, z_0)$를 중심으로 조정하는 변환. T로 좌표계의 원점을 카메라의 원점으로 옮기고, R로 회전시킨다.

$$T = \begin{bmatrix} 1 & 0 & 0 & -x_0 \\ 0 & 1 & 0 & -y_0 \\ 0 & 0 & 1 & -z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M_{WC,VC} = R \cdot T$$

6.4 Projection/Normalize Transformation (VC-NC)

가상의 평면에 개체를 투영(projection)할 때 투영 방식을 선택해 변환할 수 있음.

- Parallel projection: 평행선을 따라 투영하는 것. 개체를 구성하는 모든 점에 대해 투영 방향이 평행하다. 원근법이 반영되지 않기 때문에 있는 그대로의 길이를 측정할 때 쓴다. 어느 방향으로 투영할지에 대한 정보만 있으면 된다: Direction of projection(DOP)
- Perspective projection: 소실선을 따라 투영하는 것. 원근감이 적용된다.

Orthogonal Projection

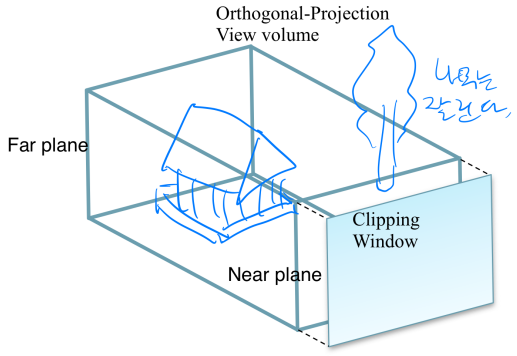
직육면체를 view volume으로 사용하는 projetion. view volume의 near와 far의 크기가 동일하기 때문에 원근감이 적용되지 않는다.

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

아래 나오는 normalize를 한 다음 위 projection을 적용하면 된다.

Clipping Window & View Volume

clipping window는 view plane 앞에 놓인 윈도우. 밖에 있는 개체가 잘린다(clipping). clipping volume은



far plane과 near plane을 정의해 멀리있는 물체가 잘리게 된다.

Normalized View Volume

x, y, z 좌표는 각각 0에서 1 또는 -1에서 1 범위로 정규화된다. orthogonal한 박스에 넣는 것. 이는 장치에서 독립적인 좌표다.

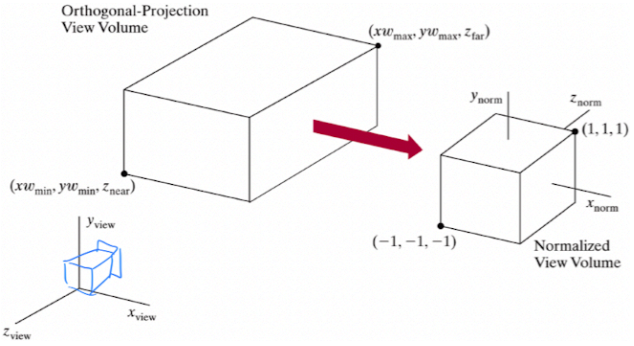
$$w = x_{\max} - x_{\min}$$

$$y = y_{\max} - y_{\min}$$

$$d = z_{\text{near}} - z_{\text{far}}$$

$$M_{ortho,norm} = \begin{bmatrix} \frac{2}{w} & 0 & 0 & -\frac{x_{\max}+x_{\min}}{w} \\ 0 & \frac{2}{h} & 0 & -\frac{y_{\max}+y_{\min}}{h} \\ 0 & 0 & \frac{2}{d} & \frac{z_{\text{near}}+z_{\text{far}}}{d} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

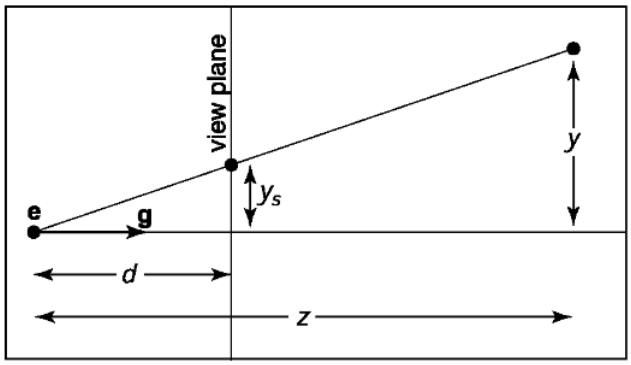
$M_{ortho,norm}$는 orthographic projections를 normalized 좌표로 변환한다. oblique projection은 DOP를



비스듬하게 만든 것.

Perspective Projection

절두체를 view volume으로 사용하는 projection. 삼각형의 닮음비를 이용해 변환할 수 있다. 이때 $y_s = d\frac{y}{z}$.



$$M_{\text{perspective}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \text{ or } \begin{bmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & d & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

3차원 좌표에 $M_{\text{perspective}}$를 곱하면 원근감이 적용된다. 만약 $(x, y, z, 1)$에 적용하면 $(x, y, z, \frac{z}{d})$가 되고, 이를 cartesian coordinate로 바꾸면 $(x\frac{d}{z}, y\frac{d}{z}, d)$. 따라서 거리 d가 멀어질수록 작아보이는 것. 여기서 homogeneous coordinate를 써야 하는 이유를 또 하나 발견. 마지막 row를 이용해 나눗셈을 할 수 있음.

만약 어떤 크기 변환 s_z와 이동 변환 t_z가 있을 때 perspective projection을 적용한다면:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & s_z & t_z \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$
$$= \begin{bmatrix} dx \\ dy \\ s_z z + t_z \\ z \end{bmatrix} = \begin{bmatrix} \frac{dx}{z} \\ \frac{dy}{z} \\ s_z + \frac{t_z}{z} \\ 1 \end{bmatrix}$$
$$= (\frac{x}{z/d}, \frac{y}{z/d}, s_z + \frac{t_z}{z})$$

카메라 중심으로 변환했으므로 모든 대상물이 -z 축에 있음. z값은 음수일 때, $[-z_{\text{near}}, -z_{far}]$를 $[-1, 1]$로 normalize하고 싶다면:

$$A = -\frac{z_{\text{far}} + z_{\text{near}}}{z_{\text{far}} - z_{\text{near}}}$$
$$B = -\frac{2z_{\text{far}} z_{\text{near}}}{z_{\text{far}} - z_{\text{near}}}$$
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} z_{\text{near}} & 0 & 0 & 0 \\ 0 & z_{\text{near}} & 0 & 0 \\ 0 & 0 & A & B \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$
$$= \begin{bmatrix} z_{\text{near}} x \\ z_{\text{near}} y \\ Az + B \\ -z \end{bmatrix} = \begin{bmatrix} \frac{z_{\text{near}} x}{-z} \\ \frac{z_{\text{near}} y}{-z} \\ -A - \frac{B}{z} \\ 1 \end{bmatrix}$$

여기서 주목할 점은 마지막 행의 세 번째 열에 -1이 들어갔다는 점. perspetive projection을 적용한다.

6.5 Viewport Transformation (NC-SC)

normalized coordinates를 screen coordinates로 변환. 우선 view volume을 viewport의 width와 height에 맞게 크기를 변환한다. view port가 $(x_{\min}, y_{\min})$와 $(x_{\max}, y_{\max})$로 정의된 사각형이라면, width $w = x_{\max} - x_{\min}$, height $h = y_{\max} - y_{\min}$.

$$\hat{S} = \begin{bmatrix} \frac{w}{2} & 0 & 0 & 0 \\ 0 & \frac{h}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

크기 변환 후에는 view volume을 viewport의 원점$(\frac{w}{2}, \frac{h}{2})$을 기준으로 오프셋 $x_{\min}, y_{\min}$만큼 이동한 위치에 배치한다.

$$\hat{T} = \begin{bmatrix} 1 & 0 & 0 & \frac{x_{\max}+x_{\min}}{2} \\ 0 & 1 & 0 & \frac{y_{\max}+y_{\min}}{2} \\ 0 & 0 & 1 & \frac{1}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

따라서,

$$M_{\text{NC, SC}} = \hat{T}\hat{S} = \begin{bmatrix} \frac{w}{2} & 0 & 0 & \frac{x_{\max}+x_{\min}}{2} \\ 0 & \frac{h}{2} & 0 & \frac{y_{\max}+y_{\min}}{2} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

이때 w를 스크린의 width pixel 개수, h를 스크린의 height pixel 개수로 생각해도 된다.

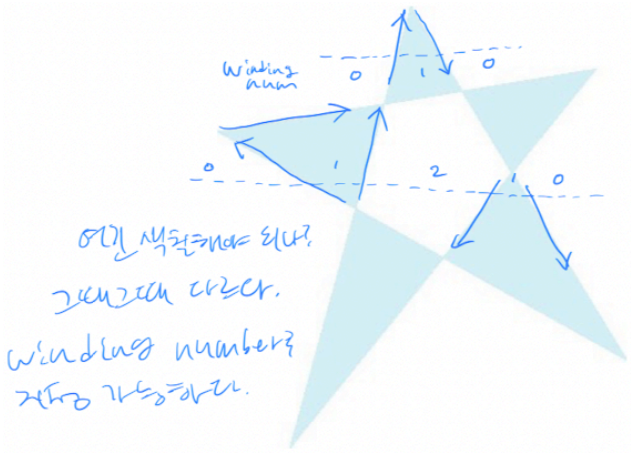
7 Rasterization

- Scan conversion: 몇 개의 픽셀을 선택해 표현할지 고르는 과정. 프래그먼트의 집합을 만들어낸다. 프래그먼트는 픽셀과 일대일 대응이 아닐 수 있다.
- DDA(Digital Differential Analyzer) Algorithm: 개체를 포함하는 픽셀을 칠하는 방식. x축을 기준으로 y 좌표를 포함하는 픽셀을 칠한다. 기울기가 1보다 크면 끊겨 보이기 때문에 y축을 기준으로 그린다.

- Bresenham's Algorithm: flaot point 연산에 비용이 많이 들기 때문에 고안된 방법. 요즘에는 컴퓨터 성능이 좋기 때문에 scan conversion하면 된다.

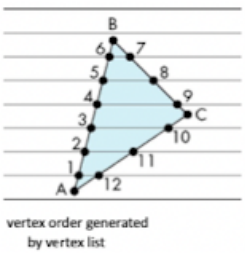
7.1 Polygon Scan Conversion

winding number를 이용해 칠해야할 영역을 찾는 방식. 개체 밖에서 개체를 관통하는 수평선을 그렸을 때 볼록 다각형은 항상 두 지점에서 수평선과 맞닿는다. 이때 y가 증가하는 부분에서는 winding nubmer를 1 증가시키고, 감소하는 부분에서는 1 감소시켜 그 값이 n인 부분만 칠한다. 홀수일때만 채우거나, 0이 아닐

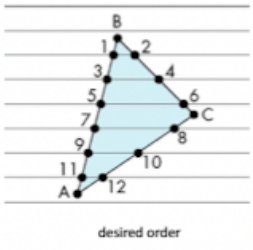


때만 채우도록 할 수도 있기 때문에 odd-even fill이라고 함.

만약 칠하려는 개체가 만약 볼록 다각형이 아니라면 문제가 생긴다. 무조건 convex하게 만들기 위해 한 개체를 여러 개의 삼각형으로 쪼갠다. 이걸 tessellation이라고 한다. 개체를 관통하는 가상의 sacn lines를 그



vertex order generated by vertex list



desired order

린다. 이때 scan line의 두 교점 사이를 칠하는 방식.

각 정점의 색깔을 다르게 칠할 수도 있음. 삼각형의 꼭짓점 A, B, C의 색이 주어졌을 때 임지의 점 T의 색은 내분점으로 interpolate해서 구할 수 있다.

7.2 Polygon Aliasing

개체에 비해 픽셀이 너무 큰 경우 문제가 됨. 샘플링이 부족해서 생기는 모든 문제를 aliasing이라고 한다. 레티나 디스플레이는 25cm 정도 떨어진 위치에서 화면을 봤을 때 aliasing 문제가 생기지 않을 정도의 해상도.

7.3 Anti-aliasing by Area Averaging

개체가 픽셀을 차지한 비율만큼 색을 칠하면 aliasing 문제를 해결할 수 있다(area averaging). 근데 다른 방법도 있다.

- Supersampling: 각 픽셀에 대해 여러 점을 샘플링해서 색상의 평균을 낸다. fragment shader를 여러 번 돌리는 셈.
- Multisampling: 평면까지 샘플링할 필요는 없다는 발상. 개체의 외각선이 지나가는 곳, 즉, 경계 부분에서만 샘플링한다.

8 Visibility

개체가 실제로 앞에 있는지, 뒤에 있는지와 상관없이 렌더링을 마지막에 한 개체가 위에 보이는 문제. 간단히 뒤에 있는 개체를 먼저 그릴 수도 있지만, cyclic overlap, penetration 같은 경우에는 해결이 안 됨.

8.1 Back-Face Removal (Culling)

개체의 뒷면은 안 보이니까 그리지 않아도 된다. 그렇다면 어디가 앞인지 알아야 함. 이때 개체의 normal vector 구하면 된다. normal vector가 -z 축이면 안 그리고, +z 축이면 그린다. 0인 경우 실루엣.

$a \times b = n$인데, $b \times a = -n$이 되는 문제. 정점의 번호가 중요하다. 정면에서 봤을 때 반시계 방향으로 번호를 부여하면 된다.

8.2 z-buffer Algorithm

가까운 개체 순서로 그리고, 멀어서 가려지면 렌더링하지 않는 방법. 두 개의 버퍼를 두고 하나는 개체를, 하나는 개체를 구성하는 픽셀별 거리를 저장. 이때 z-buffer(depth buffer)에 카메라로부터 가장 가까운 개체까지의 거리를 저장해둔다. '가까운 물체가 멀리

있는 물체를 가린다'라는 직관적 깊이 개념을 구현하는 것.

9 Lighting and Shading

광원으로부터 오는 빛을 물체가 반사하는 것만 표현하는 모델은 local reflection model(local lighting). 개체, 조명, 카메라만 사용한다. 물체가 서로를 반사하는 빛까지 표현하는 global reflection model(global lighting)이 현실적이지만 구현이 어려움. 언리얼 엔진은 local reflection만으로 global reflection을 표현하는 도전.

조명에는 point light, directional light, spot light 가 있음. point는 광원 위치에서 물체 위치를 뺀 벡터 L이 정점 위치에 의존하는 경우. directional은 L이 상수. 광원이 충분히 멀다면 위치보다는 방향이 중요하다.

9.1 Phong Reflection Model

반사된 빛 I를 아래와 같이 수식으로 표현할 수 있음.

$$I = k_a I_a + k_d I_d + k_s I_s$$
$$where \quad k_a + k_d + k_s = 1$$

- $k_a I_a$: ambient component, 주변광. 수많은 반사를 거쳐 광원이 불분명한 빛. 현실적으로 수많은 빛들의 상호작용을 구현하기 어려워 일정한 밝기와 색으로 근사한다.
- $k_d I_d$: diffuse component, 분산광. 물체의 표면에서 분산되는 빛. 각도에 따라 밝기가 달라진다.
- $k_s I_s$: specular component, 반사광. 분산광과 달리 한 방향으로 완전히 반사되는 빛. 반사되는 부분이 흰색으로 반짝여 보인다.

Diffuse reflection (Lambertian)

$$I_d = I_i \cos\theta = I_i (L \cdot N)$$

빛이 닿는 면의 경사도에 따라 단위 면적당 빛이 적어짐. 기울기가 점점 기울어져 음수가 되면 0으로 한계를 설정해야 한다.

- I_i: indensity of incident light.
- L: 빛의 방향 (unit)
- N: 점에서의 법선. 빛이 들어오는 양이 N에 따라 달라짐. (unit)
- θ: N과 L 사이의 각.

N과 L 만으로 반사되는 빛의 양을 계산하는 것. N과 L이 unit vector라면 $\cos\theta = N \cdot L$.

Specular component

$$I_s = I_i \cos^n \Omega = I_i (R \cdot V)^n$$

보는 방향 V와 빛이 반사되는 방향 R이 일치하면 반짝여보임.

- V: viewing direction (unit)
- R: reflection direction (unit)
- Ω: V와 R 사이의 각.
- n: $\cos^n$에서 n이 커질수록 뾰족한 그래프가 만들어진다. 각이 좁아질수록(n이 클수록) 더 좁은 영역이 더욱 반짝임. 이때 n이 shininess.

N과 L을 알면 R을 구할 수 있다: $R = 2(L \cdot N)N - L$.

Final Phong Reflection

$$I = k_a I_a + I_i (k_d (L \cdot N) + k_s (R \cdot V)^n)$$

k_d(알베도)는 물체의 색깔. k_s는 항상 흰색. 금속의 경우 k_s는 표면의 색상으로, k_d는 0으로 하면 된다.

9.2 Blinn-Phong Model

Phong 모델의 floating point 연산을 피하기 위한 모델. H는 L과 V의 중간에 있는 halfway 벡터($(L+V)/2$)인데, 만약 V가 L의 반사 벡터인 R에 있다면 H는 N과 같아진다. 따라서 $R \cdot V$ 대신 $N \cdot H$를 쓸 수 있다.

이때 멀리있는 광원 L은 고정되어 있고, V도 카메라가 충분히 멀다면 그냥 z 방향이라고 취급할 수 있음. 결과적으로 H를 상수로 만듦으로써 픽셀마다 빛을 계산하던 연산을 줄일 수 있다. 사실 미묘하게 결과가 달라서 잘못된 모델이지만 레거시라서 제공은 된다.

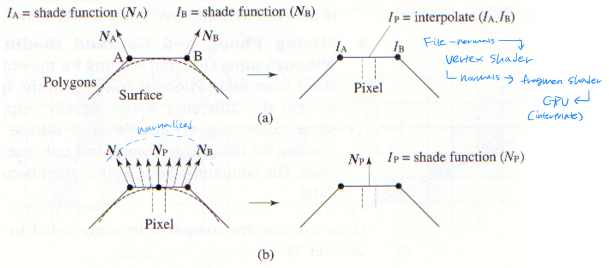
9.3 Flat & Gouraud Shading

lighting은 물체와 조명간의 상호작용으로 물체가 빛을 어떻게 반사하는지 계산하는 것. shading은 라이팅을 했을 때 물체의 색깔을 결정하는 것.

Flat은 lighting 계산을 면마다 한번만 하는 shading 방식. 구를 못 만들어서 면으로 이어붙인건데 면 하나를 통째로 같은 명암으로 설정하는건 취지에 맞지 않으므로 잘못된 방식.

Gouraud는 lighting을 삼각형의 꼭짓점마다 한번만 계산하는 방식. 정점 주변 4개 면의 normal을 모두 더해서 그 개수로 나눈 평균값을 해당 정점의 normal로 사용. 연산량이 줄어들지만 어색.

9.4 Phong Shading



어떤 면이 주어졌을 때 디자이너의 의도는 곡면이었을 것이라는 추측 가능. 한 도형에서도 각 정점의 normal이 다르기 때문에 Phong shading은 정점마다 normal을 보간한다. 각 점의 normal을 구하고 normalize하면 길이가 맞춰짐.

9.5 sRGB vs Linear

모니터에 색상을 0으로 주면 검은색, 1로 주면 흰색이 나올 것. 0.5라면? 회색이 나오긴 하지만, 실제 중간값이 아니라 훨씬 어두운 색상이 나옴. 에너지 값을 그래픽 카드에 보냈을 때 그대로 빛으로 바뀌주지 않기 때문. 해결하려면 모니터에 걸리는 전압, 그 에너지 값을 고려해 모니터의 색상값으로 바꾸는 과정이 필요, 그 표준이 sRGB.

9.6 Advanced Rendering

실제 세상에서 빛의 반사는 반구 Ω에 들어오는 빛 ω_i가 점 x에서 반사해 ω_o 방향으로 나간다. 아래는 L_o, 즉 t 시점에 λ 파장으로 점 x에서 반사되어 w_o로 들어오는 빛을 구하는 식.

$$L_o(x, \omega_o, \lambda, t) = L_e(x, \omega_o, \lambda, t) + \int_\Omega f_r(x, \omega_i, \omega_o, \lambda, t) L_i(x, \omega_i, \lambda, t)(\omega_i \cdot n) d\omega_i$$

여기서 f_r은 반사율. BRDF(Bidirectional Reflectance Distribution Function)라고 한다. 실제 물질의 빛 반사를 실측한 값을 사용한다. 이때 물체를 돌려도 빛이 동일한 것은 isotropic, CD처럼 돌리면 빛이 변하는 것은 anisotropic.

디자이너가 실측하기 어려우므로 BRDF를 근사하도록 만들어뒀다.

10 Texture Mapping

텍스처 매핑은 텍스처 이미지를 3D 개체 표면에 매핑하는 것. window 좌표계의 특정 점을 world 좌표계에서 찾고, 그것을 parametric 좌표계에서 찾고, 그것을 또 texture 좌표계에서 찾아야 한다. 몇가지 mapping 방법이 있음.

- Planar: X-Y 좌표계를 텍스처 좌표계로 대응. 하지만 z축으로 깊어지면 텍스처가 그냥 늘려짐.
- Spherical & Cylindrical & Box: 구, 원통, 박스에 매핑하는 방식. 다 문제가 있음. cylinder의 경우 위에서 보면 planar와 같은 문제.

실제로는 디자인 단계에서 3D 개체에 2차원 텍스처 좌표를 저장해둔다.

10.1 Diffuse map

개체의 표면의 색상을 정의하는 텍스처. vertex마다 위치와 색깔을 설정할 수는 있지만, vertex를 너무 잘게 쪼개야 하기 때문에 텍스처를 사용함. vertex shader에서 fragment shader로 vertex 정보를 전달할 때 GPU가 interpolation을 해 지정한 슬롯의 텍스처에서 지정된 픽셀 색상을 가져오게 된다.

10.2 Bump map (normal map)

개체 표면의 울퉁불퉁함을 정의하는 텍스처. original surface의 geometry를 조작하는 것이 아니라, 반사되

는 빛을 조정하는 것. 홈이 파인 부분이 너무 어둡게 나오기 때문에 어느정도로 어두워야 하는지 설정하기 위해 ambient occlusion map을 사용할 수도.

10.3 Aliasing Problem

텍스처의 픽셀을 텍셀이라고 부름. 원근법이 적용된 상태에서 멀리있는 픽셀의 텍스처를 가져오기 위해 해당하는 텍셀을 선택하면 너무 넓은 면적의 텍셀이 선택되는 문제가 있다.

샘플링을 많이 해도 되긴 하지만, 상대적으로 거대한 텍셀에서 몇 개 더 샘플링한다고 큰 차이가 없음. 다른 방법은 텍셀을 평균낸 값을 가져오는 것. 하지만 매번 평균을 내면 연산이 너무 많다. 그 해결책은 Mip-Mapping. 큰 텍스처의 여러 영역을 평균내어 작은 버전의 텍스처를 미리 만들어 두는 것. 작은 개체의 텍스처를 작은 텍스처에서 가져오면 된다.

10.4 Anisotropic Filtering

텍스처가 뿌옇게 보이는 문제. 텍스처를 x로만 줄인 버전, y로만 줄인 버전, xy 모두 줄인 버전을 모두 만들어두고 사용하면 된다.

11 Shadows and Multi-Pass Rendering

빛을 가려 그림자를 만드는 개체는 Creator. 그림자가 비춰지는 개체는 Reciever. 한 개체가 creator이자 receiver일 수 있다. point source는 딱 떨어지는 umbra를 형성하지만, 광원의 크기가 커져 area source가 되면 softshadow가 일어나는 penumbra가 함께 형성된다.

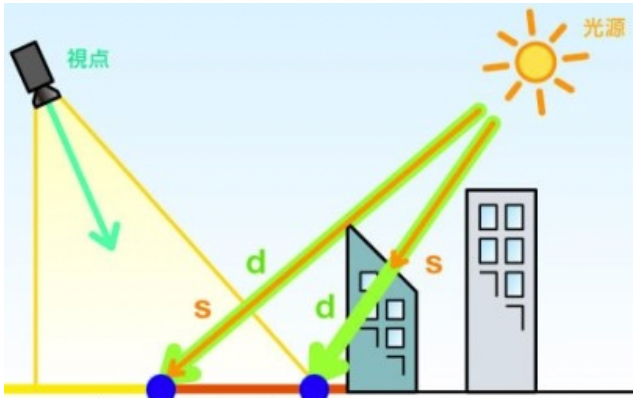
어떻게 그림자를 표현할까? 그림자 텍스쳐를 입힐 수도 있음(baking). 하지만 조명과 물체를 움직일 수가 없다.

11.1 Shadow Volume

creator로 빛이 가려지면 뒤쪽의 절두체 볼륨이 그림자라고 간주할 수 있음. 카메라 시점에서 shadow volume 면이 먼저 보이면 그림자 안에 있다고 판단. 지금은 안 쓰임.

11.2 Shadow Mapping

빛의 위치에서 월드를 봤을 때 안 보이는 부분을 그림자로 간주. 이때 빛에서 각 픽셀까지의 depth를 프레임 버퍼 오브젝트에 저장해둔다. 다만 이렇게 만든 shadow depth map은 사용자에게 보이지 않게 렌더링한다. 이렇게 화면에 그리지 않는 렌더링은 off-screen rendering.



카메라가 어떤 점을 봤을 때, 해당 픽셀과 광원의 거리가 shadow map의 해당 픽셀에 쓰여진 광원과 creator의 거리보다 크다면 그림자. 즉, shadow map에 카메라가 보는 픽셀의 depth가 아니라 다른 픽셀의 depth가 쓰여 있다면 그림자로 간주.

빛 위치에서 세상을 보려면 새로운 view projection matrix를 만들어야 한다. (MVP: Model View Projection) 카메라가 보는 픽셀을 shadow map의 픽셀에 대응시킬 때도 MVP가 필요하다. 이때 shadow map의 projection은 orthogonal이다.

11.3 Shadow Acne

그림자가 없어야 하는 곳에 나타나는 그림자 무늬. shadow map의 해상도가 무한하지 않으므로 shadow map은 픽셀을 샘플링해 구성된다. 그런데 샘플링의 오차로 인해 shadow map의 텍셀 기준으로 특정 픽셀의 그림자를 판정하려 할 때, 실제로는 자신을 가리는 개체가 없음에도 참조한 shadow map의 텍셀에는 자신을 가리는 다른 픽셀의 depth가 저장되어 있는 케이스가 발생한다. 때문에 필연적으로 shadow acne가 나타난다.

shadow map에 bias를 적용해 depth를 미세하게 조정하면 해결할 수 있다. 하지만 bias가 너무 크면 개체로부터 그림자가 분리되는 peterpanning이 일어난다.

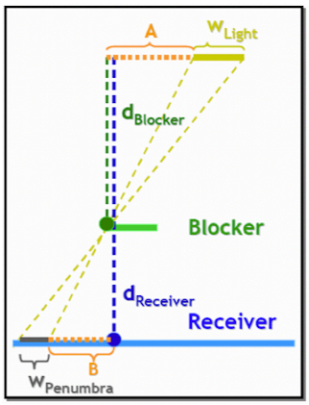
따라서 bias를 적절히 설정하거나, shadow acne를 가리도록 개체의 위치나 두께 등을 조정해야 한다.

11.4 Shadow Aliasing

그림자 경계 부분이 계단처럼 울퉁불퉁해 보이는 현상. shadow map을 키워 샘플링을 많이 하면 해결된다. 그게 아니면...

Multi sampling (Percentage Closer Filtering): poisson distribution에 따라 픽셀을 여러 번 샘플링한다. poisson disk 값이 커지면 그림자의 sharpness가 줄어들어 soft shadow가 표현된다. sharpness가 커져서 날카로워지면 텍셀이 보이는 문제가 생긴다. 이때는 shadow map의 크기를 키울 수 밖에 없다.

Percentage-Closer Soft Shadows: penumbra는 광원의 크기, 빛과 blocker 사이의 거리, 빛과 receiver 사이의 거리에 따른다. 따라서 blocker를 찾고, penumbra의 크기를 계산한다.

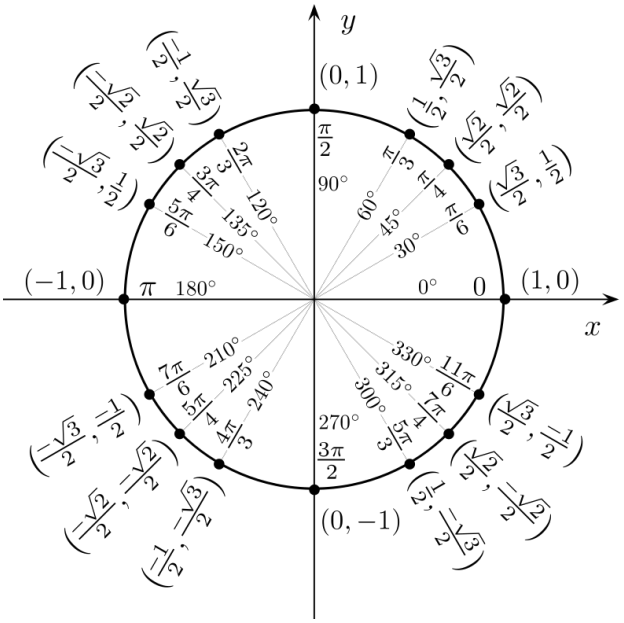


$$w_{Penumbra} = \frac{(d_{Receiver} - d_{Blocker}) \cdot w_{Light}}{d_{Blocker}}$$

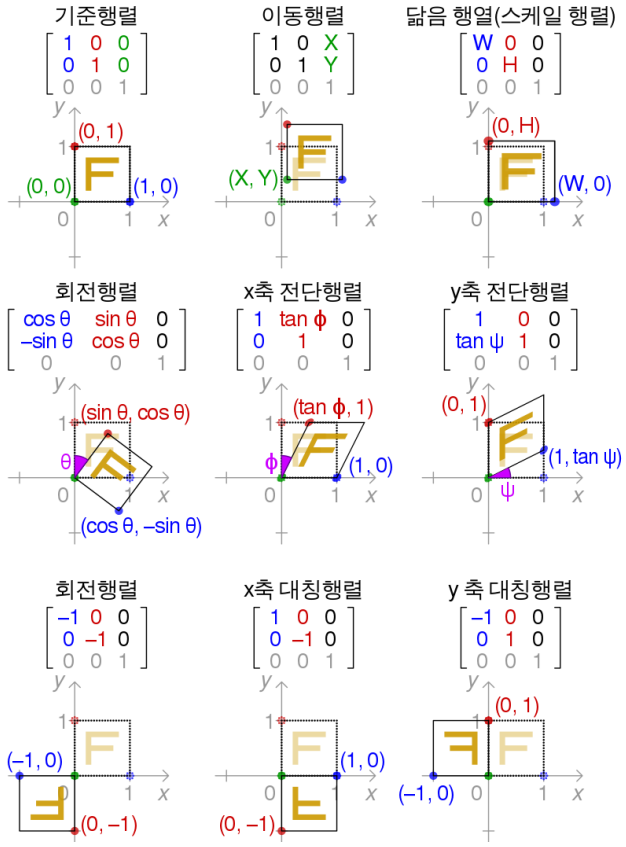
- Assumes that blocker, receiver, and light are parallel

12 부록

π 라디안은 180도. 1라디안은 $\frac{180}{\pi}$ 도. 1도는 $\frac{\pi}{180}$ 라디안. 아래 그림은 각도, 라디안, cos, sin 순서.



반시계 방향으로 θ는 시계 방향으로 $-\theta$, 이때: $\sin -\theta = -\sin\theta$, $\cos -\theta = \cos\theta$




```
mat3 getTBN(vec3 N) {
  vec3 Q1 = dFdx(worldPosition);
  vec3 Q2 = dFdy(worldPosition);
  vec2 st1 = dFdx(texCoords);
  vec2 st2 = dFdy(texCoords);
  float D = st1.s * st2.t - st2.s * st1.t;
  return mat3(
    normalize((Q1 * st2.t - Q2 * st1.t) * D),
    normalize((-Q1 * st2.s + Q2 * st1.s) * D),
    N,
  );
}

void main(void) {
  vec3 l = lightPosition - worldPosition;
  vec3 L = normalize(l);
  vec3 N = normalize(normal);

  mat3 tbn = getTBN(N);
  float dBdu = texture(
    bumpTex, texCoords + vec2(0.00001, 0)
  ).r - texture(bumpTex,
                texCoords - vec2(0.00001, 0)).r;
  float dBdv = texture(
    bumpTex, texCoords + vec2(0, 0.00001)
  ).r - texture(bumpTex,
                texCoords - vec2(0, 0.00001)).r;
  N = normalize(N - dBdu * tbn[0] * 100 -
    dBdv * tbn[1] * 100);

  vec3 V = normalize(cameraPosition -
                     worldPosition);
  vec3 R = 2 * dot(L, N) * N - L;
  vec3 I = lightColor / dot(l, l);
  vec4 diffColor = texture(diffTex, texCoords);
  vec3 c = diffColor.rgb * max(0, dot(L, N)) *
           I + diffColor.rgb * ambientLight;
  c += pow(max(0, dot(R, V)), shininess) * I;

  out_Color = vec4(pow(c, vec3(1 / 2.2)), 1);
}
```