ADVANCED COLLISION DETECTION

Homepage: https://sites.google.com/site/doc4code/

Email: sj6219@hotmail.com

2011/9/21

물리엔진에서 많이 사용되는 알고리즘이 GJK 와 EPA 이다. 이 문서에서는 이 두 알고리즘의 구현 방법에 대해서 알아보겠다.

목차

SUPPORT MAPPING

주어진 물체 P, vector $\mathbf{v}$ 에 대하여 다음을 만족하는 점으로 변환하는 함수 $S(P, \mathbf{v})$를 Support Mapping 이라고 부른다.

- $S(P, \mathbf{v}) \in P$
- $v \cdot S(P, \mathbf{v}) = \max \{\mathbf{v} \cdot \mathbf{a} : \mathbf{a} \in P\}$

그림 1.

BOX

Box B 의 중심이 $\begin{bmatrix} c_x \\ c_y \\ c_z \end{bmatrix}$ 이고, 모서리의 방향이 x, y, z 축이고 그 길이가 $2e_x$, $2e_y$, $2e_z$

일 때

$$S\left(B, \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}\right) = \begin{bmatrix} c_x \\ c_y \\ c_z \end{bmatrix} + \begin{bmatrix} \mathrm{sign}(v_x)e_x \\ \mathrm{sign}(v_y)e_y \\ \mathrm{sign}(v_z)e_z \end{bmatrix}$$

단,

$$\mathrm{sign}(x) = \begin{cases} x \geq 0, & 1 \\ x < 0, & -1 \end{cases}$$

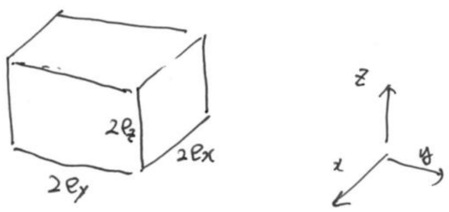

그림 2.

SPHERE

Sphere P 의 중심이 $\begin{bmatrix} c_x \\ c_y \\ c_z \end{bmatrix}$이고 반지름이 r 이라면,

$$S(P, \mathbf{v}) = \begin{bmatrix} c_x \\ c_y \\ c_z \end{bmatrix} + \begin{cases} \dfrac{r\mathbf{v}}{|\mathbf{v}|}, & \mathbf{v} \neq \mathbf{0} \\ r\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, & \mathbf{v} = \mathbf{0} \end{cases}$$

CYLINDER

원기둥 P 가 중심이 $\begin{bmatrix} c_x \\ c_y \\ c_z \end{bmatrix}$ 이고, 중심축이 z 축이고, 반지름이 r, 높이가 2h 라고 하면

$$S\left(P, \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}\right) = \begin{bmatrix} c_x \\ c_y \\ c_z \end{bmatrix} + \begin{cases} \begin{bmatrix} rv_x/\sqrt{{v_x}^2 + {v_y}^2} \\ rv_y/\sqrt{{v_x}^2 + {v_y}^2} \\ \mathrm{sign}(v_z)h \end{bmatrix}, & v_y \neq 0 \text{ or } v_y \neq 0 \\ \begin{bmatrix} 0 \\ 0 \\ \mathrm{sign}(v_z)h \end{bmatrix}, & \text{otherwise} \end{cases}$$

그림 3.

TRANSFORMATION

물체 Q 가 물체 P 를 행렬 M 과 벡터 **N** 에 의해 아래처럼 변환되었다고 하자.

$Q = \{ M\mathbf{x} + \mathbf{N} : \mathbf{x} \in P \}$

그런데, 벡터 v 에 대해,

$$\mathbf{v} \cdot \mathrm{S}(\mathrm{Q}, \mathbf{v}) = \max\{\mathbf{v} \cdot \mathbf{y} : \mathbf{y} \in \mathrm{Q}\}$$

$$= \max\{\mathbf{v} \cdot (\mathrm{M}\mathbf{x} + \mathbf{N}) : \mathbf{x} \in \mathrm{P}\}$$

$$= \max\{\mathbf{v}^{\mathrm{T}}\mathrm{M}\mathbf{x} + \mathbf{v} \cdot \mathbf{N} : \mathbf{x} \in \mathrm{P}\}$$

$$= \max\{(\mathrm{M}^{\mathrm{T}}\mathbf{v})^{\mathrm{T}}\mathbf{x} : \mathbf{x} \in \mathrm{P}\} + \mathbf{v} \cdot \mathbf{N}$$

$$= \max\{(\mathrm{M}^{\mathrm{T}}\mathbf{v}) \cdot \mathbf{x} : \mathbf{x} \in \mathrm{P}\} + \mathbf{v} \cdot \mathbf{N}$$

$$= (\mathrm{M}^{\mathrm{T}}\mathbf{v}) \cdot \mathrm{S}(\mathrm{P}, \mathrm{M}^{\mathrm{T}}\mathbf{v}) + \mathbf{v} \cdot \mathbf{N}$$

$$= \mathbf{v}^{\mathrm{T}}\mathrm{M}\,\mathrm{S}(\mathrm{P}, \mathrm{M}^{\mathrm{T}}\mathbf{v}) + \mathbf{v} \cdot \mathbf{N}$$

$$= \mathbf{v} \cdot (\mathrm{M}\,\mathrm{S}(\mathrm{P}, \mathrm{M}^{\mathrm{T}}\mathbf{v}) + \mathbf{N})$$

그러므로

$$\mathrm{S}(\mathrm{Q}, \mathbf{v}) = \mathrm{M}\,\mathrm{S}(\mathrm{P}, \mathrm{M}^{\mathrm{T}}\mathbf{v}) + \mathbf{N}$$

이다.

Oval 도 Sphere 를 Scale, Rotation, Position 변환한 것이므로, Oval 의 Support Mapping 도 위의 방법을 이용해 계산할 수 있다.

SUPPORT MAPPING OF CSO

CSO 의 Support Mapping 도 아래와 같이 구할 수 있다.

S(A-B, **v**) = S(A,**v**) – S(B,-**v**)

DOBKIN-KIRKPATRICK HIERARCHICAL REPRESENTATION

Dobkin-Kirkpatrick Hierarchical Representation 는 Polytope 에서 Support Mapping 을 효율적으로 계산하기 위한 자료구조이다. Polytope 의 vertex 가 n 개라면, O(log n) 시간 내에 계산할 수 있다

Polytope P 는 여러 개의 Polytope $P_1, P_2, \ldots, P_n$로 표현된다. P_1는 P 와 같고, $\mathrm{vert}(P_{i+1}) \subset \mathrm{vert}(P_i)$이다.

마지막 P_n는 simplex 이다.

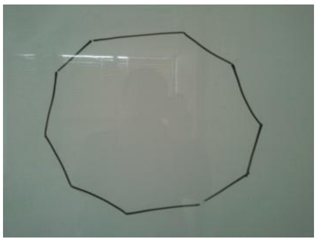

그림 4. P_1

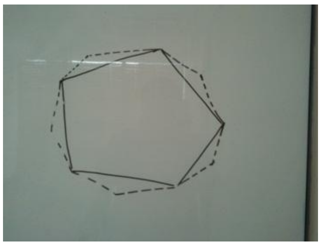

그림 5. P_2

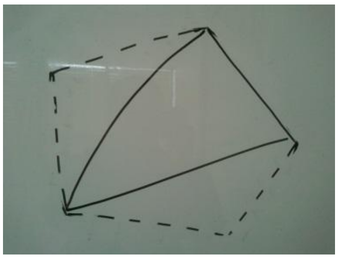

그림 6. P_3

그림에서 $S(P_1, \mathbf{v})$를 구하려면, 먼저 $S(P_3, \mathbf{v})$를 구한다. 그 다음 P_2에서 인접한 vertex 를 검사한다. 마지막으로 P_1에서 vertex 를 구한다.

P_i에서 P_{i+1}를 구하는 방법은 다음과 같다.

우선 다음 알고리즘을 이용해 집합 S_i를 구한다.

그 다음 $P_{i+1} = \mathrm{conv}(\mathrm{vert}(P_i) - S_i)$ 를 구한다.

```
Si = ϕ

for v ∈ vert(Pi) {
        if deg(v)≤8 and ! marked(v) {
                Si = Si ∪ {v}
                for w ∈ adj(v) {
                        mark(w)
                }
        }
}
```

위에서 $\deg(\mathbf{v})$는 $\mathbf{v}$와 인접해 있는 vertex 의 개수다. $\mathrm{adj}(\mathbf{v})$는 $\mathbf{v}$와 인접해 있는 vertex 의 집합이다.

GILBERT-JOHNSON-KEERTHI ALGORITHM

Gilbert-Johnson-Keerthi (GJK) 알고리즘은 두 convex 물체의 거리를 계산하는 데 사용된다.

우선 두 물체의 CSO(configuration space obstacle)를 구한 후, 원점에서 가장 가까운 점을 찾는다.

우선 주어진 물체 P 에서 원점에 가장 가까운 점을 계산하는 함수를 정의해 보자.

$$\upsilon(P) \in P \text{ and } |v(P)| = \min\{|\mathbf{x}| : \mathbf{x} \in P\}$$

GJK 알고리즘은 주어진 Convex 물체 A, B 에 대해서 $\upsilon(A - B)$ 를 찾게 된다.

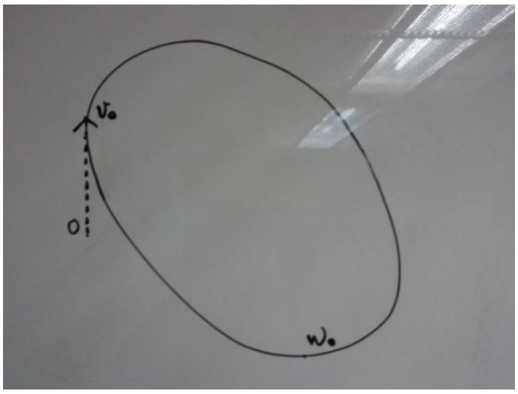

그림 7. $\mathbf{W_0} = \emptyset$

$\mathbf{v_0}$은 A-B 에 속한 임의의 점이다.

그리고, 점들의 집합 $W_0 = \emptyset$ (공집합)이다.

$\mathbf{w_i} = S(A - B, -\mathbf{v_i})$을 구한다.

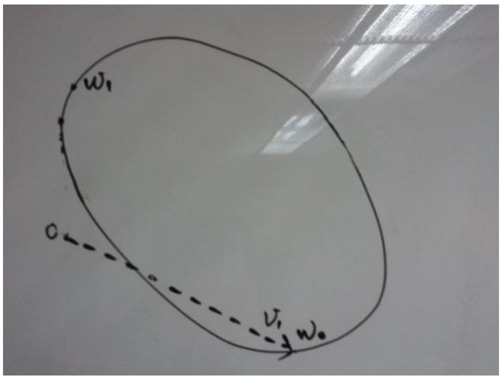

그림 8. $\mathbf{W_1} = \{\mathbf{w_0}\}$

$$\mathbf{v_{i+1}} = \upsilon\left(\mathrm{conv}(W_i \cup \{\mathbf{w_i}\})\right)$$

W_{i+1}는 다음 조건을 만족하는 점들의 집합 X 중 가장 작은 집합이다.

$$X \subseteq W_i \cup \{\mathbf{w_i}\}$$

$$\mathbf{v_{i+1}} \in \mathrm{conv}(X)$$

다르게 설명하면

$W_i \cup \{w_i\} = \{\mathbf{P_1}, \mathbf{P_2}, \ldots, \mathbf{P_n}\}$ 로 표시하면,

$$\mathbf{v_{i+1}} = \alpha_1 \mathbf{P_1} + \alpha_2 \mathbf{P_2} + \cdots + \alpha_n \mathbf{P_n}$$

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

$$\alpha_1 \geq 0, \alpha_2 \geq 0, \ldots, \alpha_n \geq 0$$

로 표시할 수 있다.

이때, $W_{i+1} = \{\mathbf{P_k} : \alpha_k > 0\}$ 즉, 계수가 양수인 점들의 집합이다.

W_i는 simplex 이므로, 최대 4 개의 원소까지 가질 수 있다.

$\mathbf{v_{i+1}}$ 와 W_{i+1} 를 구하는 방법에 대해서는 다음에 나오는 Johnson's distance Algorithm 에서 다루겠다.

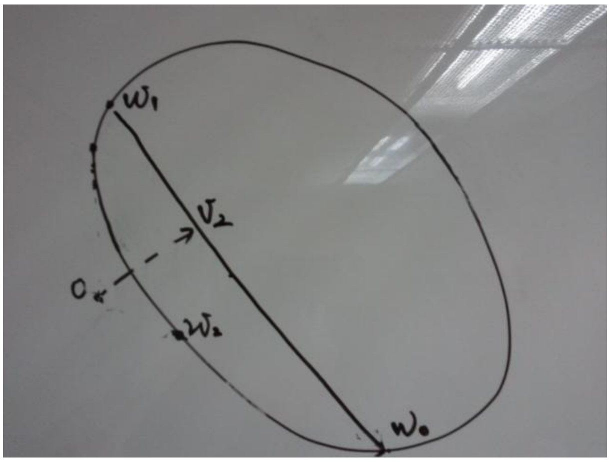

그림 9. $\mathbf{W_2} = \{\mathbf{w_0}, \mathbf{w_1}\}$

이렇게, 반복해 구하면 $\mathbf{v_i}$는 $\upsilon(A - B)$에 수렴한다고 한다. (수렴하는 걸 증명하는 건 까다로울 것 같으니 생략하자.)

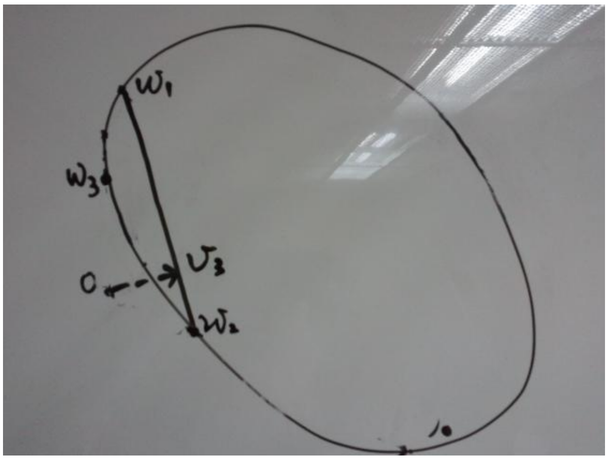

그림 10. $\mathbf{W_3} = \{\mathbf{w_1}, \mathbf{w_2}\}$

이제 언제까지 반복할 것인지에 대해서만 알아보면 된다.

그전에 다음 정리부터 증명해 보자.

정리 1. 모든 벡터 $\mathbf{x}$, $\mathbf{y}$ 에 대해, $|\upsilon(\mathrm{conv}(\{\mathbf{x},\mathbf{y}\})| < |\mathbf{x}| \leftrightarrow |\mathbf{x}|^2 - \mathbf{x}\cdot\mathbf{y} > 0$

$$|\mathrm{v}(\mathrm{conv}(\{\mathbf{x},\mathbf{y}\})|$$

$$= \min_{0\le\lambda\le1} |\mathbf{x} + (\mathbf{y}-\mathbf{x})\lambda|$$

$$= \min_{0\le\lambda\le1} |\mathbf{x} + (\mathbf{y}-\mathbf{x})\lambda|$$

$$= \sqrt{\min_{0\le\lambda\le1} |\mathbf{y}-\mathbf{x}|^2\lambda^2 + 2\mathbf{x}\cdot(\mathbf{y}-\mathbf{x})\lambda + |\mathbf{x}|^2}$$

그런데, $\mathrm{f}(\lambda) = |\mathbf{y}-\mathbf{x}|^2\lambda^2 + 2\mathbf{x}\cdot(\mathbf{y}-\mathbf{x})\lambda + |\mathbf{x}|^2$로 표기하면, $\mathrm{f}(0) = |\mathbf{x}|$ 이므로

$$|\upsilon(\mathrm{conv}(\{\mathbf{x},\mathbf{y}\})| < |\mathbf{x}|$$

$$\leftrightarrow \min_{0\le\lambda\le1} \mathrm{f}(\lambda) < \mathrm{f}(0)$$

$$\leftrightarrow \mathbf{x}\cdot(\mathbf{y}-\mathbf{x}) < 0$$

$$\leftrightarrow |\mathbf{x}|^2 - \mathbf{x}\cdot\mathbf{y} > 0$$

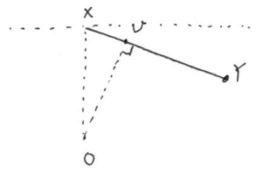

그림 11. $\mathbf{v} = \upsilon(\mathbf{conv}(\{\mathbf{x}, \mathbf{y}\}))$

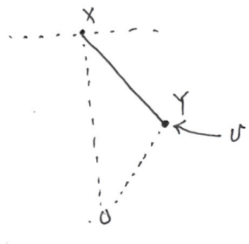

그림 12. $\mathbf{v} = \upsilon(\mathbf{conv}(\{\mathbf{x}, \mathbf{y}\}))$

정리 2. $|\mathbf{v_{i+1}}| \leq |\mathbf{v_i}|$

$\mathbf{v_{i+1}} = \upsilon\big(\mathrm{conv}(\mathrm{W_i} \cup \{\mathbf{w_i}\})\big)$ 이므로, $\forall \mathbf{x} \in \mathrm{conv}(\mathrm{W_i} \cup \{\mathbf{w_i}\}) :\ |\mathbf{v_{i+1}}| <= |\mathbf{x}|$ 이다.

그런데, $\mathbf{v_i} \in \mathrm{conv}(\mathrm{W_i})$ 이므로, $\mathbf{v_i} \in \mathrm{conv}(\mathrm{W_i} \cup \{\mathbf{w_i}\})$이다. $\mathbf{x} = \mathbf{v_i}$ 을 대입하면

$$|\mathbf{v_{i+1}}| \leq |\mathbf{v_i}|$$

이다.

정리 3. $|\mathbf{v_{i+1}}| = |\mathbf{v_i}| \ \leftrightarrow\ \mathbf{v_i} = \upsilon(\mathrm{A} - \mathrm{B})$

$\mathbf{v_i} = \upsilon(\mathrm{A} - \mathrm{B}) \rightarrow |\mathbf{v_{i+1}}| = |\mathbf{v_i}|$는 증명이 간단하다. 정리 1 에 의해 $|\mathbf{v_{i+1}}| \leq |\mathbf{v_i}|$ 여야 하고,

$\mathbf{v_{i+1}} \in \mathrm{A} - \mathrm{B}$ 이므로 $|\mathbf{v_{i+1}}| \geq \upsilon(\mathrm{A} - \mathrm{B})$여야 한다.

$|\mathbf{v_{i+1}}| = |\mathbf{v_i}|$ 를 가정해 보자. $\mathbf{v_i} \in \mathrm{conv}(\mathrm{W_i})$ 이므로 $\mathrm{conv}(\{\mathbf{v_i}, \mathbf{w_i}\}) \subseteq \mathrm{conv}(\mathrm{W_i} \cup \{\mathbf{w_i}\})$이다.

$$|\mathbf{v_{i+1}}| = |\upsilon(\mathrm{conv}(W_i \cup \{\mathbf{w_i}\}))| \leq |\upsilon(\mathrm{conv}(\{\mathbf{v_i}, \mathbf{w_i}\}))|$$

$$|\mathbf{v_i}| \leq |\upsilon(\mathrm{conv}(\{\mathbf{v_i}, \mathbf{w_i}\}))|$$

정리 1 에서 $\mathbf{x} = \mathbf{v_i},\ \mathbf{y} = \mathbf{w_i}$ 을 대입한 것을 위 식에 적용하면

$$|\mathbf{v_i}|^2 - \mathbf{v_i} \cdot \mathbf{w_i} \leq 0$$

$\forall \mathbf{z} \in A - B\ :\ -\mathbf{v_i} \cdot \mathbf{w_i} = -\mathbf{v_i} \cdot S(A - B, -\mathbf{v_i}) \geq -\mathbf{v_i} \cdot \mathbf{z}$ 을 위 식에 적용하면

$$\forall \mathbf{z} \in A - B :\ |\mathbf{v_i}|^2 \leq \mathbf{v_i} \cdot \mathbf{z}$$

그러므로 $\mathbf{v_i} = \upsilon(A - B)$ 이다.

정리 4. $|\mathbf{v_i} - \upsilon(A - B)|^2 \leq |\mathbf{v_i}|^2 - \mathbf{v_i} \cdot \mathbf{w_i}$

$$|\mathbf{v_i} - \upsilon(A - B)|^2 = |\mathbf{v_i}|^2 - 2\mathbf{v_i} \cdot \upsilon(A - B) + |\upsilon(A - B)|^2$$

정리 1 에서 $\mathbf{x} = \upsilon(A - B),\ \mathbf{y} = \mathbf{v_i}$ 을 대입하면, $\upsilon(\mathrm{conv}(\{\upsilon(A - B), \mathbf{v_i}\})) = \upsilon(A - B)$ 이므로

$$|\upsilon(A - B)|^2 - \upsilon(A - B) \cdot \mathbf{v_i} \leq 0$$

원래 식으로 돌아가면

$$\begin{aligned}|\mathbf{v_i} - \upsilon(A - B)|^2 &= |\mathbf{v_i}|^2 - 2\mathbf{v_i} \cdot \upsilon(A - B) + |\upsilon(A - B)|^2 \\ &\leq |\mathbf{v_i}|^2 - \mathbf{v_i} \cdot \upsilon(A - B)\end{aligned}$$

그런데, $\forall \mathbf{x} \in A - B :\ -\mathbf{v_i} \cdot \mathbf{w_i} = -\mathbf{v_i} \cdot S(A - B, -\mathbf{v_i}) \geq -\mathbf{v_i} \cdot \mathbf{x}$ 이므로, $\mathbf{x} = \upsilon(A - B)$를 대입하면

$$|\mathbf{v_i} - \upsilon(A - B)|^2 \leq |\mathbf{v_i}|^2 - \mathbf{v_i} \cdot \upsilon(A - B) \leq |\mathbf{v_i}|^2 - \mathbf{v_i} \cdot \mathbf{w_i}$$

정리 4 는 오차의 최대 값이 $|\mathbf{v_i}|^2 - \mathbf{v_i} \cdot \mathbf{w_i}$보다 작다는 것을 의미한다.

ALGORITHM

지금까지 배운 것을 요약하면 아래와 같다.

```
Vector GJK(Polytope A, Polytope B)
{
        Vector v = (A-B 에 속하는 임의의 점);
        Set<Vector> W = ϕ;
        for ( ; ; ) {
                Vector w = SupportMapping(A,B,-v);
                if (v · v – v · w <= ε_abs^2)
                        break;
                v = υ(conv(W∪{w}));
                W = (Convex Hull 이 v 를 포함하는 가장 작은 W∪{w}의 부분집합);
        }
        return v;
}
```

이제 **v** 와 **W** 를 계산하는 방법에 대해 좀 더 자세히 알아보자.

JOHNSON'S DISTANCE ALGORITHM

Johnson 의 알고리즘은 주어진 simplex 에서 원점에 가장 가까운 점을 계산하는 방법이다.

점들의 집합 $X = \{\mathbf{x_1}, \mathbf{x_2}, \dots, \mathbf{x_n}\}$ 이 affinely independent 하다고 가정하자.

이해를 돕기 위해 간단한 경우부터 알아보고, 나중에 일반적인 경우에 대해서 공부하겠다.

우선 n=2 인 경우부터 생각해보자.

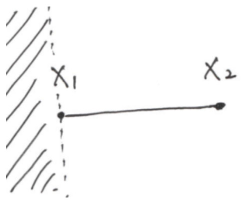

그림 13. $(\mathbf{0}-\mathbf{x_1})\cdot(\mathbf{x_2}-\mathbf{x_1})\le \mathbf{0}$

원점에서 가장 가까운 점을 찾는 방법은 원점이

그림 13 과 같이 빗금 친 공간에 있는지 검사하는 것이다. 조건식 $(\mathbf{0}-\mathbf{x_1})\cdot(\mathbf{x_2}-\mathbf{x_1})\le 0$이 맞으면, $\upsilon(\mathrm{conv}(\{\mathbf{x_1},\mathbf{x_2}\}))=\mathbf{x_1}$이다

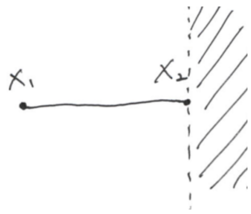

그림 14. $(\mathbf{0}-\mathbf{x_2})\cdot(\mathbf{x_2}-\mathbf{x_1})\ge \mathbf{0}$

마찬가지로, 원점이 그림 14 의 빗금 친 부분에 있는지 검사해서, 맞으면 $\upsilon(\mathrm{conv}(\{\mathbf{x_1},\mathbf{x_2}\}))=\mathbf{x_2}$ 가 된다.

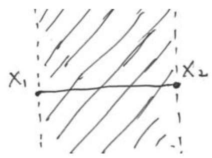

그림 15.

그림 13 과 14 의 경우가 아니면, $\upsilon(\mathrm{conv}(\{\mathbf{x_1},\mathbf{x_2}\}))=\upsilon(\mathrm{aff}(\{\mathbf{x_1},\mathbf{x_2}\}))$이다.

이번에는 n=3 인 경우에 대해 알아보자.

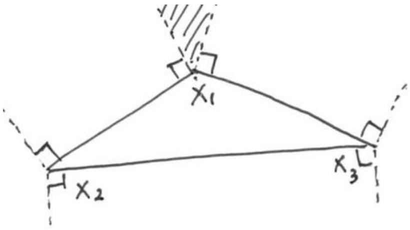

그림 16. $(\mathbf{0} - \mathbf{x_1}) \cdot (\mathbf{x_2} - \mathbf{x_1}) \le \mathbf{0}$ **and** $(\mathbf{0} - \mathbf{x_1}) \cdot (\mathbf{x_3} - \mathbf{x_1}) \le \mathbf{0}$

우선,

$$(\mathbf{0} - \mathbf{x_1}) \cdot (\mathbf{x_2} - \mathbf{x_1}) \le 0 \text{ and } (\mathbf{0} - \mathbf{x_1}) \cdot (\mathbf{x_3} - \mathbf{x_1}) \le 0$$

인지 검사해서 맞으면 $\upsilon\big(\mathrm{conv}(\{\mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}\})\big) = \mathbf{x_1}$이다. 꼭지점 $\mathbf{x_2}$와 $\mathbf{x_3}$에 대해서도 비슷하게 검사한다.

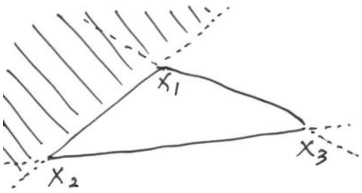

그림 17.

다음은 각 변에 대해서 검사할 차례이다.

원점이 빗금 친 공간에 있는지 검사해서, 조건에 맞으면 $\upsilon\big(\mathrm{conv}(\{\mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}\})\big) = \upsilon\big(\mathrm{aff}(\{\mathbf{x_1}, \mathbf{x_2}\})\big)$ 이 된다.

조건식은 조금 복잡하다.

$\mathbf{v} = \upsilon\big(\mathrm{aff}(\{\mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}\})\big)$라고 하자.

$\mathbf{v}$가 평면 $\mathrm{aff}(\{\mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}\})$에 포함돼야 하므로,

$$\mathbf{v} = \alpha_1\mathbf{x_1} + \alpha_2\mathbf{x_2} + \alpha_3\mathbf{x_3}$$

$$\alpha_1 + \alpha_2 + \alpha_3 = 1$$

이어야 한다.

그리고 $\mathbf{v}$가 평면 $\mathrm{aff}(\{\mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}\})$와 수직이어야 하므로, $\mathbf{v}\cdot(\mathbf{x_2}-\mathbf{x_1})=0$, $\mathbf{v}\cdot(\mathbf{x_3}-\mathbf{x_1})=0$이다.

$$(\mathbf{x_2}-\mathbf{x_1})\cdot\mathbf{x_1}\alpha_1+(\mathbf{x_2}-\mathbf{x_1})\cdot\mathbf{x_2}\alpha_2+(\mathbf{x_2}-\mathbf{x_1})\cdot\mathbf{x_3}\alpha_3=0$$

$$(\mathbf{x_3}-\mathbf{x_1})\cdot\mathbf{x_1}\alpha_1+(\mathbf{x_3}-\mathbf{x_1})\cdot\mathbf{x_2}\alpha_2+(\mathbf{x_3}-\mathbf{x_1})\cdot\mathbf{x_3}\alpha_3=0$$

연립 방정식으로 만들면

$$\begin{bmatrix} 1 & 1 & 1 \\ (\mathbf{x_2}-\mathbf{x_1})\cdot\mathbf{x_1} & (\mathbf{x_2}-\mathbf{x_1})\cdot\mathbf{x_2} & (\mathbf{x_2}-\mathbf{x_1})\cdot\mathbf{x_3} \\ (\mathbf{x_3}-\mathbf{x_1})\cdot\mathbf{x_1} & (\mathbf{x_3}-\mathbf{x_1})\cdot\mathbf{x_2} & (\mathbf{x_3}-\mathbf{x_1})\cdot\mathbf{x_3} \end{bmatrix}\begin{bmatrix}\alpha_1\\ \alpha_2\\ \alpha_3\end{bmatrix}=\begin{bmatrix}1\\0\\0\end{bmatrix}$$

Cramer's rule 을 이용하면 답을 구할 수 있다. Cramer 법칙을 잘 모르면, 공업 수학책이나 위키피디아를 검색해보면 된다.

$\mathrm{d}=\det\begin{bmatrix} 1 & 1 & 1 \\ (\mathbf{x_2}-\mathbf{x_1})\cdot\mathbf{x_1} & (\mathbf{x_2}-\mathbf{x_1})\cdot\mathbf{x_2} & (\mathbf{x_2}-\mathbf{x_1})\cdot\mathbf{x_3} \\ (\mathbf{x_3}-\mathbf{x_1})\cdot\mathbf{x_1} & (\mathbf{x_3}-\mathbf{x_1})\cdot\mathbf{x_2} & (\mathbf{x_3}-\mathbf{x_1})\cdot\mathbf{x_3} \end{bmatrix}$로 표기하면

$$\alpha_1=\frac{\det\begin{bmatrix} 1 & 1 & 1 \\ 0 & (\mathbf{x_2}-\mathbf{x_1})\cdot\mathbf{x_2} & (\mathbf{x_2}-\mathbf{x_1})\cdot\mathbf{x_3} \\ 0 & (\mathbf{x_3}-\mathbf{x_1})\cdot\mathbf{x_2} & (\mathbf{x_3}-\mathbf{x_1})\cdot\mathbf{x_3} \end{bmatrix}}{\mathrm{d}}=\frac{\det\begin{bmatrix} (\mathbf{x_2}-\mathbf{x_1})\cdot\mathbf{x_2} & (\mathbf{x_2}-\mathbf{x_1})\cdot\mathbf{x_3} \\ (\mathbf{x_3}-\mathbf{x_1})\cdot\mathbf{x_2} & (\mathbf{x_3}-\mathbf{x_1})\cdot\mathbf{x_3} \end{bmatrix}}{\mathrm{d}}$$

$$\alpha_2=\frac{\det\begin{bmatrix} 1 & 1 & 1 \\ (\mathbf{x_2}-\mathbf{x_1})\cdot\mathbf{x_1} & 0 & (\mathbf{x_2}-\mathbf{x_1})\cdot\mathbf{x_3} \\ (\mathbf{x_3}-\mathbf{x_1})\cdot\mathbf{x_1} & 0 & (\mathbf{x_3}-\mathbf{x_1})\cdot\mathbf{x_3} \end{bmatrix}}{\mathrm{d}}=\frac{-\det\begin{bmatrix} (\mathbf{x_2}-\mathbf{x_1})\cdot\mathbf{x_1} & (\mathbf{x_2}-\mathbf{x_1})\cdot\mathbf{x_3} \\ (\mathbf{x_3}-\mathbf{x_1})\cdot\mathbf{x_1} & (\mathbf{x_3}-\mathbf{x_1})\cdot\mathbf{x_3} \end{bmatrix}}{\mathrm{d}}$$

$$\alpha_3=\frac{\det\begin{bmatrix} 1 & 1 & 1 \\ (\mathbf{x_2}-\mathbf{x_1})\cdot\mathbf{x_1} & (\mathbf{x_2}-\mathbf{x_1})\cdot\mathbf{x_2} & 0 \\ (\mathbf{x_3}-\mathbf{x_1})\cdot\mathbf{x_1} & (\mathbf{x_3}-\mathbf{x_1})\cdot\mathbf{x_2} & 0 \end{bmatrix}}{\mathrm{d}}=\frac{\det\begin{bmatrix} (\mathbf{x_2}-\mathbf{x_1})\cdot\mathbf{x_1} & (\mathbf{x_2}-\mathbf{x_1})\cdot\mathbf{x_2} \\ (\mathbf{x_3}-\mathbf{x_1})\cdot\mathbf{x_1} & (\mathbf{x_3}-\mathbf{x_1})\cdot\mathbf{x_2} \end{bmatrix}}{\mathrm{d}}$$

여기서, α_3<=0 이면 원점이 그림 17 의 빗금 친 공간에 있는 것이다.

다른 변에 대해서도 마찬가지로 검사한다.

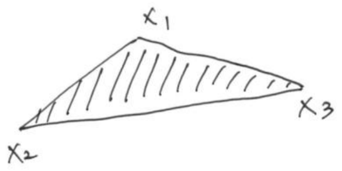

그림 18.

모든 검사에 실패했으면 $\upsilon(\mathrm{aff}(\{\mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}\}))$는 그림 18 처럼 삼각형 내부에 있는 것이다.

이 때, $\upsilon(\mathrm{conv}(\{\mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}\})) = \upsilon(\mathrm{aff}(\{\mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}\}))$ 이 된다.

이제 일반적인 경우에 대해서 생각해 보자.

$\mathrm{I} = \{1,2, \dots, \mathrm{n}\}$라고 표현하면

$$\upsilon(\mathrm{conv}(\mathrm{X})) = \sum_{\mathrm{i} \in \mathrm{I}} \alpha_{\mathrm{i}} \mathbf{x_i}$$

$$\sum_{\mathrm{i} \in \mathrm{I}} \alpha_{\mathrm{i}} = 1$$

$$\forall \mathrm{i} \in \mathrm{I} : \ \alpha_{\mathrm{i}} \geq 0$$

로 표현할 수 있다.

임의의 X에 대하여, 위 세 식을 만족하는 $\alpha_1, \alpha_2, \dots, \alpha_{\mathrm{n}}$ 이 존재한다.

집합 J, Y 를 $\mathrm{J} = \{\mathrm{i} : \ \alpha_{\mathrm{i}} > 0\}$, $\mathrm{Y} = \{\mathbf{x_i} : \ \mathrm{i} \in \mathrm{J}\}$로 정하면,

$$\upsilon(\mathrm{conv}(\mathrm{X})) = \upsilon(\mathrm{aff}(\mathrm{Y})) = \sum_{\mathrm{i} \in \mathrm{J}} \alpha_{\mathrm{i}} \mathbf{x_i}$$

$$\sum_{\mathrm{i} \in \mathrm{J}} \alpha_{\mathrm{i}} = 1$$

이다.

이때,

$$(\text{조건 1})\ \forall i \in J :\ \alpha_i > 0$$

을 만족해야 한다. 다르게 표현하면 $\upsilon(\mathrm{aff}(Y))$는 $\mathrm{conv}(Y)$ 의 내부에 있어야 한다.

그리고,

$$\forall j \in (I - J) : \upsilon\left(\mathrm{aff}\left(Y \cup \{\mathbf{x_j}\}\right)\right) = \sum_{i \in J \cup \{j\}} \beta_{J \cup \{j\}, i} \mathbf{x_i}$$

$$\sum_{i \in J \cup \{j\}} \beta_{J \cup \{j\}, i} = 1$$

로 표시될 수 있다.

여기서,

$$(\text{조건 2})\ \forall j \in (I - J) :\ \beta_{J \cup \{j\}, j} \leq 0$$

여야 한다. 만약, $\beta_{J \cup \{j\}, j} > 0$ 이라면, $\upsilon(\mathrm{aff}(Y \cup \{\mathbf{x_j}\})) < \upsilon(\mathrm{aff}(Y))$여야 하기 때문에 모순이 발생한다.

조건 2 를 다르게 표현하면 $\upsilon\left(\mathrm{aff}\left(Y \cup \{\mathbf{x_j}\}\right)\right)$는 $\mathrm{conv}(X)$ 의 외부 또는 경계에 있어야 한다.

그래서, 조건 1 과 2 를 만족시키는 J 를 찾아야 한다.

이번엔 집합 M 이 I 의 부분집합일 때, 계수 $\beta_{M,i}$를 구하는 방법에 대해 알아보자.

$$\upsilon\left(\mathrm{aff}(\{\mathbf{x_i} :\ i \in M\})\right) = \sum_{i \in M} \beta_{M,i} \mathbf{x_i}$$

$$\sum_{i \in M} \beta_{M,i} = 1$$

를 만족하는 계수를 구해보자.

편의상 M 의 원소 $\mathbf{x_i}$ 을 $\mathbf{y_j}$ 으로 표기하자.

$$\upsilon\left(\mathrm{aff}(\{\mathbf{x_i} :\ i \in M\})\right) = \upsilon\left(\mathrm{aff}(\{\mathbf{y_j} :\ 1 \leq j \leq m\})\right) = \sum_{j=1}^{m} \lambda_j \mathbf{y_j}$$

$$\sum_{j=1}^{m} \lambda_j = 1$$

그리고, $\upsilon\left(\mathrm{aff}(\{\mathbf{y_j} : 1 \le j \le m\})\right)$는 $\mathrm{aff}(\{\mathbf{y_j} : 1 \le j \le m\})$ 에 수직이어야 한다.

$$\forall j \in \{2, \dots, m\} : \ (\mathbf{y_j} - \mathbf{y_1}) \cdot \sum_{j=1}^{m} \lambda_j \mathbf{y_j} = 0$$

행렬식으로 다시 정리하면,

$$\begin{bmatrix} 1 & \cdots & 1 \\ (\mathbf{y_2} - \mathbf{y_1}) \cdot \mathbf{y_1} & \cdots & (\mathbf{y_2} - \mathbf{y_1}) \cdot \mathbf{y_m} \\ \vdots & \ddots & \vdots \\ (\mathbf{y_m} - \mathbf{y_1}) \cdot \mathbf{y_1} & \cdots & (\mathbf{y_m} - \mathbf{y_1}) \cdot \mathbf{y_m} \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_m \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

$\begin{bmatrix} 1 & \cdots & 1 \\ (\mathbf{y_2} - \mathbf{y_1}) \cdot \mathbf{y_1} & \cdots & (\mathbf{y_2} - \mathbf{y_1}) \cdot \mathbf{y_m} \\ \vdots & \ddots & \vdots \\ (\mathbf{y_m} - \mathbf{y_1}) \cdot \mathbf{y_1} & \cdots & (\mathbf{y_m} - \mathbf{y_1}) \cdot \mathbf{y_m} \end{bmatrix} = \mathrm{A}$로 표시하면

행렬 방정식을 Cramer's rule 에 의해 풀면

$$\lambda_j = \frac{\det \begin{bmatrix} 1 & \cdots & 1 & 1 & 1 & \cdots & 1 \\ (\mathbf{y_2} - \mathbf{y_1}) \cdot \mathbf{y_1} & \cdots & (\mathbf{y_2} - \mathbf{y_1}) \cdot \mathbf{y_{j-1}} & 0 & (\mathbf{y_2} - \mathbf{y_1}) \cdot \mathbf{y_{j+1}} & \cdots & (\mathbf{y_2} - \mathbf{y_1}) \cdot \mathbf{y_m} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ (\mathbf{y_m} - \mathbf{y_1}) \cdot \mathbf{y_1} & \cdots & (\mathbf{y_m} - \mathbf{y_1}) \cdot \mathbf{y_{j-1}} & 0 & (\mathbf{y_m} - \mathbf{y_1}) \cdot \mathbf{y_{j+1}} & \cdots & (\mathbf{y_m} - \mathbf{y_1}) \cdot \mathbf{y_m} \end{bmatrix}}{\det(\mathrm{A})}$$

행렬 A 에서 1 행과 j 열을 삭제한 행렬을 $\mathrm{A_{1j}}$로 나타내면

$$\beta_{M,i} = \lambda_j = \frac{(-1)^{j+1} \det(\mathrm{A_{1j}})}{\det(\mathrm{A})}$$

$$\mathrm{A_{1j}} = \begin{bmatrix} (\mathbf{y_2} - \mathbf{y_1}) \cdot \mathbf{y_1} & \cdots & (\mathbf{y_2} - \mathbf{y_1}) \cdot \mathbf{y_{j-1}} & (\mathbf{y_2} - \mathbf{y_1}) \cdot \mathbf{y_{j+1}} & \cdots & (\mathbf{y_2} - \mathbf{y_1}) \cdot \mathbf{y_m} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ (\mathbf{y_m} - \mathbf{y_1}) \cdot \mathbf{y_1} & \cdots & (\mathbf{y_m} - \mathbf{y_1}) \cdot \mathbf{y_{j-1}} & (\mathbf{y_m} - \mathbf{y_1}) \cdot \mathbf{y_{j+1}} & \cdots & (\mathbf{y_m} - \mathbf{y_1}) \cdot \mathbf{y_m} \end{bmatrix}$$

가 된다. (이 부분이 이해가 안 가면, 선형수학에서 cofactor 를 공부하면 된다.)

그리고, 분자 $(-1)^{j+1} \det(\mathrm{A_{1j}}) = \Delta_{M,i}$로 표시하자.

$\sum_{i \in M} \beta_{M,i} = 1$이므로, $\det(\mathrm{A}) = \sum_{i \in M} \Delta_{M,i}$로 $\det(\mathrm{A})$를 계산하는 편이 더 쉽다.

또, 항상 $\det(A) \geq 0$인 성질이 있다. 특히, $\{\mathbf{x_i} : i \in M\}$이 affinely independent 하면 $\det(A) > 0$ 이다.

이것에 대한 증명은 E. G. Gilbert, D. W. Johnson, and S.S. Keerthi, A fast procedure for computing the distance between complex objects in three-dimensional space 라는 논문 끝부분에 나와있다.

증명하기는 어렵지 않으나, Matrix 수학에 대한 지식이 필요하므로 생략하겠다.

$\beta_{M,i}$를 부호를 구할 때 $\Delta_{M,i}$의 부호를 구해도 된다는 것만 알아두자.

이번에는 $\Delta_{M,i}$를 재귀적으로 정의해 보겠다.

$$\forall i \in I : \Delta_{\{i\},i} = 1$$

으로 정한다.

$M \subset I$ 이고 $k \in I - M$ 이라고 하자.

$\mathbf{x_k} = \mathbf{y_{m+1}}$로 생각하면

$$\Delta_{M\cup\{k\},k} = (-1)^{m+2}\det(B)$$

$$단, B = \begin{bmatrix} (\mathbf{y_2} - \mathbf{y_1}) \cdot \mathbf{y_1} & \cdots & (\mathbf{y_2} - \mathbf{y_1}) \cdot \mathbf{y_m} \\ \vdots & \ddots & \vdots \\ (\mathbf{y_m} - \mathbf{y_1}) \cdot \mathbf{y_1} & \cdots & (\mathbf{y_m} - \mathbf{y_1}) \cdot \mathbf{y_m} \\ (\mathbf{y_{m+1}} - \mathbf{y_1}) \cdot \mathbf{y_1} & \cdots & (\mathbf{y_{m+1}} - \mathbf{y_1}) \cdot \mathbf{y_m} \end{bmatrix}$$

$$\det(B) = \sum_{j=1}^{m} (-1)^{m+j} (\mathbf{y_{m+1}} - \mathbf{y_1}) \cdot \mathbf{y_j} \det(A_{1j})$$

$$\Delta_{M\cup\{k\},k} = (-1)^{m+2} \sum_{j=1}^{m} (-1)^{m+j} (\mathbf{y_{m+1}} - \mathbf{y_1}) \cdot \mathbf{y_j} \det(A_{1j})$$

$$= \sum_{j=1}^{m} (-1)^{2m+2+j} (\mathbf{y_{m+1}} - \mathbf{y_1}) \cdot \mathbf{y_j} \det(A_{1j})$$

$$= \sum_{j=1}^{m} (-1)^{1+j} (\mathbf{y_1} - \mathbf{y_{m+1}}) \cdot \mathbf{y_j} \det(A_{1j})$$

$$= \sum_{i \in M} \Delta_{M,i} \left(\mathbf{x_p} - \mathbf{x_k}\right) \cdot \mathbf{x_i}$$

마지막 줄에서 $\mathbf{y_1} = \mathbf{x_p}$로 표현하였다. $\mathbf{x_p}$ 는 $\mathbf{y_j}$의 순서와 상관없으므로, $p \in M$ 이면 된다.

요약하면,

$$\forall \mathrm{i} \in \mathrm{I} : \Delta_{\{\mathrm{i}\},\mathrm{i}} = 1$$

$$\Delta_{\mathrm{M}\cup\{\mathrm{k}\},\mathrm{k}} = \sum_{\mathrm{i}\in\mathrm{M}} \Delta_{\mathrm{M},\mathrm{i}}(\mathbf{x_p} - \mathbf{x_k}) \cdot \mathbf{x_i} \text{ where } \mathrm{k} \in \mathrm{I} - \mathrm{M} \text{ and } \mathrm{p} \in \mathrm{M}$$

$$\upsilon(\mathrm{aff}(\{\mathbf{x_i} : \mathrm{i} \in \mathrm{M}\})) = \sum_{\mathrm{i}\in\mathrm{M}} \beta_{\mathrm{M},\mathrm{i}}\mathbf{x_i} \text{ where } \beta_{\mathrm{M},\mathrm{i}} = \frac{\Delta_{\mathrm{M},\mathrm{i}}}{\sum_{\mathrm{j}\in\mathrm{M}} \Delta_{\mathrm{M},\mathrm{j}}}$$

그리고, I 의 모든 부분 집합 중에서 다음 조건을 만족하는 J 를 찾는다.

$$\forall \mathrm{i} \in \mathrm{J} : \Delta_{\mathrm{J},\mathrm{i}} > 0$$

$$\forall \mathrm{i} \in \mathrm{I} - \mathrm{J} : \Delta_{\mathrm{J}\cup\{\mathrm{i}\},\mathrm{i}} \leq 0$$

이 때, 우리가 원하는 답을 다음과 같이 얻을 수 있다.

$$\upsilon(\mathrm{conv}(\mathrm{X})) = \upsilon(\mathrm{aff}(\{\mathbf{x_i} : \mathrm{i} \in \mathrm{J}\})) = \frac{\sum_{\mathrm{i}\in\mathrm{J}} \Delta_{\mathrm{J},\mathrm{i}}\mathbf{x_i}}{\sum_{\mathrm{i}\in\mathrm{J}} \Delta_{\mathrm{J},\mathrm{i}}}$$

설명이 좀 어려울 수 있다.

정리하기 위해 실제 예를 들어보겠다.

그림 16 로 돌아가서, $\upsilon(\mathrm{conv}(\{\mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}\}))$를 구해보자.

우선 $\Delta_{\{1\},1} = 1$, $\Delta_{\{2\},2} = 1$, $\Delta_{\{3\},3} = 1$ 부터 시작한다.

일단, $\upsilon(\mathrm{aff}(\{\mathbf{x_1}\}))$ 이 원하는 답인지 검사하는 것부터 시작한다.

조사를 위해서는 $\Delta_{\{1,2\},2}$와 $\Delta_{\{1,3\},3}$를 구해야 한다.

$$\Delta_{\{1,2\},2} = \Delta_{\{1\},1}(\mathbf{x_1} - \mathbf{x_2}) \cdot \mathbf{x_1}$$

마찬가지로 $\Delta_{\{1,3\},3}$를 구한다.

$$\Delta_{\{1,3\},3} = \Delta_{\{1\},1}(\mathbf{x_1} - \mathbf{x_3}) \cdot \mathbf{x_1}$$

만약 $\Delta_{\{1,2\},2}$<=0 이고 $\Delta_{\{1,3\},3} \le 0$ 이면 $\upsilon(\mathrm{aff}(\{\mathbf{x_1}\}))$가 원하는 답이 된다. 즉 $\upsilon(\mathrm{conv}(\{\mathbf{x_1},\ \mathbf{x_2},\ \mathbf{x_3}\})) = \mathbf{x_1}$이다.

그 다음 $\upsilon(\mathrm{aff}(\{\mathbf{x_2}\}))$에서 대해서 검사해 본다. 조사를 위해서 $\Delta_{\{1,2\},1}$과 $\Delta_{\{2,3\},3}$를 계산하고, 부호를 검사한다.

다음은 $\upsilon(\mathrm{aff}(\{\mathbf{x_3}\}))$에서 처리한다. $\Delta_{\{1,3\},1}$과 $\Delta_{\{2,3\},2}$를 계산한다.

그 다음 과정은 $\upsilon(\mathrm{aff}(\{\mathbf{x_1},\ \mathbf{x_2}\}))$를 처리한다. $\Delta_{\{1,2,3\},3}$를 구하고 그 부호를 검사한다.

$$\Delta_{\{1,2,3\},3} = \Delta_{\{1,2\},1}(\mathbf{x_1} - \mathbf{x_3}) \cdot \mathbf{x_1} + \Delta_{\{1,2\},2}(\mathbf{x_1} - \mathbf{x_3}) \cdot \mathbf{x_2}$$

으로 계산하고, 만약, $\Delta_{\{1,2,3\},3} \le 0$ 이면 $\upsilon(\mathrm{aff}(\{\mathbf{x_1},\ \mathbf{x_2}\})) = \frac{\Delta_{\{1,2\},1}\mathbf{x_1} + \Delta_{\{1,2\},2}\mathbf{x_2}}{\Delta_{\{1,2\},1} + \Delta_{\{1,2\},2}}$

가 답이 된다.

마찬가지 방법으로 $\upsilon(\mathrm{aff}(\{\mathbf{x_1},\ \mathbf{x_3}\}))$과 $\upsilon(\mathrm{aff}(\{\mathbf{x_2},\ \mathbf{x_3}\}))$에 대해서도 조사해 본다.

마지막으로, 지금까지 검사에 모두 실패했다면, 답은 $\upsilon(\mathrm{aff}(\{\mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}\}))$이 된다.

$$\upsilon(\mathrm{aff}(\{\mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}\})) = \frac{\Delta_{\{1,2,3\},1}\mathbf{x_1} + \Delta_{\{1,2,3\},2}\mathbf{x_2} + \Delta_{\{1,2,3\},3}\mathbf{x_3}}{\Delta_{\{1,2,3\},1} + \Delta_{\{1,2,3\},2} + \Delta_{\{1,2,3\},3}}$$

GJK 와 DISTANCE ALGORITHM

GJK 에서 $W_i \cup \{\mathbf{w_i}\}$을 입력으로 $\mathbf{v_{i+1}}$ 와 W_{i+1} 를 구할 때, distance 알고리즘을 사용한다.

그런데, $\mathbf{w_i} \in \mathrm{W_{i+1}}$인 성질을 이용하면, 모든 부분 집합을 검사할 필요가 없다.

$\mathrm{W_{i+1}}$가 $\mathbf{w_i}$ 를 포함해야 하므로, 검사할 부분 집합을 반으로 줄일 수 있다.

다음을 증명해 보자.

정리 5. $\mathbf{v_i} \neq \upsilon(A - B) \rightarrow \mathbf{w_i} \in \mathrm{W_{i+1}}$

$\mathbf{w_i} \notin \mathrm{W_{i+1}}$라고 가정해 보자. 그러면, $\upsilon(\mathrm{conv}(\mathrm{W_{i+1}})) = \upsilon(\mathrm{conv}(\mathrm{W_i}))$이어야 하므로, $|\mathbf{v_{i+1}}| = |\mathbf{v_i}|$ 이다.

그런데, 정리 3 에 의해 $\mathbf{v_i} = \upsilon(A - B)$이므로, 모순이 된다.

오차

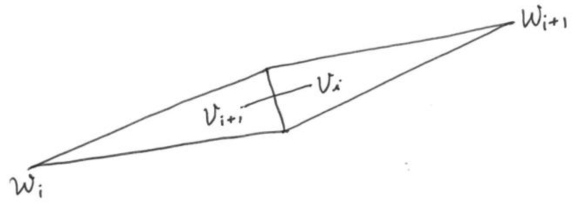

그림 19.

오차 때문에 GJK 알고리즘이 무한 반복되는 경우가 있을 수 있다.

그림처럼 삼각형이 길쭉하고, $|\mathbf{v_i}|$ 가 크면 오차가 더 심해진다.

이런 경우, $\mathbf{v_i}$ 와 $\mathbf{v_{i+1}}$가 교대로 반복될 수 있다.

정리 2 와 정리 3 의 성질을 이용하여, $|\mathbf{v_{i+1}}| \geq |\mathbf{v_i}|$조건이 만족될 때 반복을 중단하도록 하면, 무한 반복을 피할 수 있다.

INTERSECTION TEST

게임에서 GJK 를 사용될 때, 접촉하고 있는지 여부만 검사하는 경우가 대부분이다.

이럴 때는 반복 종료 조건을 완화하여, 더 효율적으로 구현할 수 있다.

Incremental Separating Axis GJK(ISA-GJK) Algorithm 에서는 접촉 여부만 검사하는 경우 사용된다.

이 알고리즘에서는 두 물체를 분리하는 Separating Axis 를 발견하는 즉시 반복을 종료하게 되어있다..

```
bool ISA_GJK(Polytope A, Polytope B)
{
	Vector v = (임의의 점);
	Set<Vector> W = ϕ;
	for ( ; ; ) {
		Vector w = SupportMapping(A,B,-v);
		if (v · w > 0)
			// v is separating axis
			return false;
		v = υ(conv(W∪{w}));
		W = (Convex Hull 이 v 를 포함하는 가장 작은 W∪{w}의 부분집합);
		if (v · v <= ε_abs^2)
			return true;
	}
}
```

EXPANDING-POLYTOPE ALGORITHM

주어진 물체 P 에서 다음을 만족하는 점을 구하는 함수 $\varphi(P)$를 다음과 같이 정의하자.

$$\varphi(\mathrm{P}) \in \mathrm{P} \text{ and } |\varphi(\mathrm{P})| = \inf\{|\mathbf{x}| : \mathbf{x} \notin \mathrm{P}\}$$

Expanding-Polytope Algorithm(EPA)는 convex 물체 P 가 입력으로 주어졌을 때, 그 $\varphi(\mathrm{P})$를 계산하는 알고리즘이다. 주로 GJK 를 사용해, 두 물체가 충돌한 경우, 충돌 위치를 찾는데 사용된다.

EPA 는 GJK 에서 구한 simplex 를 입력으로 받아서, Penetration Depth 를 찾게 된다.

설명을 쉽게 하기 위해서 우선 2 차원인 경우부터 설명하겠다.

EPA IN 2D SPACE

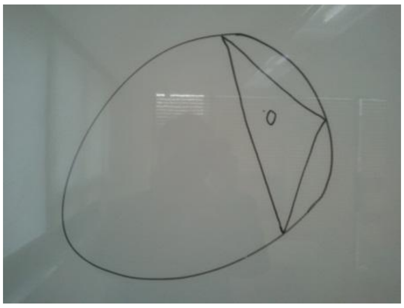

그림 1.

Convex 한 물체 P 의 $\varphi(\mathrm{P})$를 구해 보자.

알고리즘은 원점을 포함하고 있는 삼각형부터 시작한다. 이 삼각형을 P_0로 표기하자. 그리고, $\mathrm{vert}(\mathrm{P}_0) = \{\mathbf{x_0}, \mathbf{x_1}, \mathbf{x_2}\}$ 라고 하자.

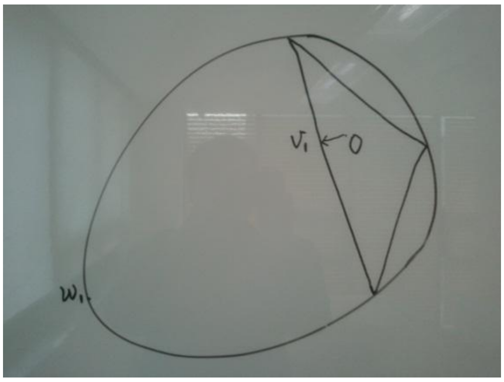

그림 2.

먼저 $\mathbf{v_i} = \varphi(P_i)$를 계산한다. 그 다음 $\mathbf{w_i} = S(P, \mathbf{v_i})$ 를 계산한다. $P_{i+1} = \mathrm{conv}(\mathrm{vert}(P_i) \cup \{\mathbf{w_i}\})$를 이용하여 다각형(Polygon)을 확장한다. 이렇게 계속 반복하게 되면 $\mathbf{v_i}$ 와 $\mathbf{w_i}$가 $\varphi(P_i)$에 가까워진다.

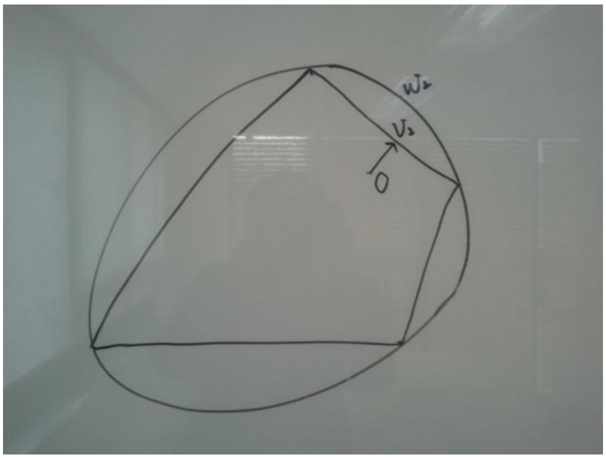

그림 3.

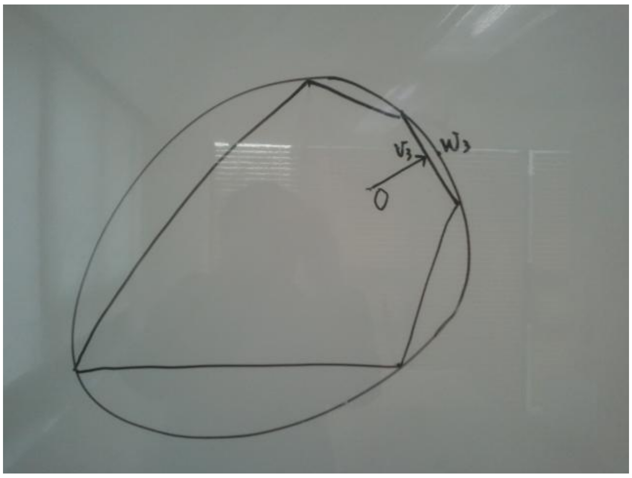

그림 4.

$\mathbf{v_0} = \varphi(\mathrm{P}_0)$를 구하려면, 삼각형의 세 변(edge)을 검사해 보면 된다. 즉, $\upsilon(\mathrm{aff}(\{\mathbf{x_0}, \mathbf{x_1}\}))$, $\upsilon(\mathrm{aff}(\{\mathbf{x_1}, \mathbf{x_2}\}))$, $\upsilon(\mathrm{aff}(\{\mathbf{x_0}, \mathbf{x_2}\}))$ 중 크기가 가장 작은 점이 $\mathbf{v_0}$ 가 된다.

$\mathbf{v_i} = \varphi(\mathrm{P_i})$도 마찬가지로 $\mathrm{P_i}$의 모든 변을 검사한다. 이 때, $\mathbf{v_i}$가 변 $\overline{\mathbf{x_p x_q}}$ 위에 있다고 하면, $\mathbf{v_i} = \upsilon\left(\mathrm{aff}(\{\mathbf{x_p}, \mathbf{x_q}\})\right)$ 여야 한다.

$\mathbf{v_i} = \lambda_p \mathbf{x_p} + \lambda_q \mathbf{x_q}$ (단, $\lambda_p + \lambda_q = 1$)로 표현한다면, $\lambda_p > 0, \lambda_q > 0$이어야 한다.

그리고, $\mathrm{P_i}$의 변의 집합에서 $\overline{\mathbf{x_p x_q}}$를 빼고, $\overline{\mathbf{x_p w_i}}$와 $\overline{\mathbf{w_i x_q}}$ 를 추가하면 $\mathrm{P_{i+1}}$의 변의 집합이 된다.

지금까지 배운 것을 구현해 보자

2D ALGORITHM

```
struct Entry
{
        Vector   x[2];
        Vector   v;
        float lambda[2];
        float distance;
```

```
    bool Internal() { return lambda[0] > 0 && lambda[1] > 0; }
    bool operator < (Entry &other) { return this->distance > other->distance; }
};
```

Polygon P_i는 변(edge)들의 집합으로 표현한다. 각 변은 Entry 구조체로 표현하였다.

x[0] 과 x{1] 은 Edge 의 양 끝점이다.

```
Entry MakeEntry(Vertex x0, Vertex x1)
{
    Entry entry;
    entry.x[0] = x0;
    entry.x{1] = x1;
    entry.lambda = (Johnson's distance algorithm);
    entry.v = entry.lambda[0] * x0 + entry.lambda[1] * x1;
    entry.distance = |entry.v|;
    return entry;
}
```

MakeEntry() 함수는 Edge 의 양 끝점을 입력으로 받아서 구조체의 각 필드 값을 계산한다. $\text{entry.v} = \upsilon(\text{aff}(\{\text{entry.x}[0], \text{entry.x}[1]\}))$ 를 만족해야 하고, entry.lambda 는 이때 계수의 값들이다.

그리고, 반복할 때마다, entry.distance 가 작은 edge 를 효율적으로 찾기 위해 priority_queue 를 사용하였다. priority_queue 에 관해서는 C++ Standard Template Library(STL) manual 을 참조하기 바란다.

```
Vector EPA2D(Polytope& P)
{
    priority_queue<Entry> queue;
    for each edge x_p x_q of P
    {
        Entry entry = MakeEntry(x_p, x_q);
        if (entry.Internal())
            queue.push(entry);
    }
```

```
	Vector v;
	for ( ; ; )
	{
		Entry entry = queue.top();
		queue.pop();
		v = entry.v;
		Vector w = SupportMapping(P, v);
		if (v · w / entry.distance - entry.distance >= εabs)
			break;
		Entry entry1 = MakeEntry(entry.x[0], w);
		if (entry1.Internal())
			queue.push(entry1);
		Entry entry2 = MakeEntry(w, entry.x[1]);
		if (entry2.Internal())
			queue.push(entry2);
		}
	return v;
}
```

$\mathrm{entry} = \mathrm{queue.top}()$에서 entry 는 $\mathrm{P_i}$의 edge 중에서 가장 원점에서 가까운 edge 이므로,

entry.Internal()은 true 이어야 한다. 그러므로 queue.push()할 때, entry.Internal()을 검사해서 false 이면 추가하지 않는 게 더 효율적이다.

그리고, $\mathbf{v_i}$ 와 $\mathbf{w_i}$를 비교해서, 서로 가까우면 반복을 중단한다.

EPA IN 3D SPACE

이번엔 3 차원 공간에서 Convex 한 물체 P 의 $\varphi(\mathrm{P})$를 구해 보자.

알고리즘은 원점을 포함하고 있는 사면체(tetrahedron)부터 시작한다. 이 사면체를 $\mathrm{P_0}$로 표기하자. 그리고, $\mathrm{vert}(\mathrm{P_0}) = \{\mathbf{x_0}, \mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}\}$ 라고 하자.

먼저 $\mathbf{v_i} = \varphi(P_i)$를 계산한다. 그 다음 $\mathbf{w_i} = S(P, \mathbf{v_i})$ 를 계산한다. $P_{i+1} = \mathrm{conv}(\mathrm{vert}(P_i) \cup \{\mathbf{w_i}\})$를 이용하여 다면체(Polyhedron)을 확장한다. 이렇게 계속 반복하게 되면 $\mathbf{v_i}$ 와 $\mathbf{w_i}$가 $\varphi(P_i)$에 가까워진다.

$\mathbf{v_0} = \varphi(P_0)$를 구하려면, 사면체$P_0$의 4 개의 면(face)을 검사해 보면 된다. 즉, $\upsilon(\mathrm{aff}(\{\mathbf{x_0}, \mathbf{x_1}, \mathbf{x_2}\}))$, $\upsilon(\mathrm{aff}(\{\mathbf{x_0}, \mathbf{x_1}, \mathbf{x_3}\}))$, $\upsilon(\mathrm{aff}(\{\mathbf{x_0}, \mathbf{x_2}, \mathbf{x_3}\}))$,$\upsilon(\mathrm{aff}(\{\mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}\}))$ 중 크기가 가장 작은 점이 $\mathbf{v_0}$ 가 된다.

$\mathbf{v_i} = \varphi(P_i)$도 마찬가지로 P_i의 모든 변을 검사한다. 이 때, $\mathbf{v_i}$가 삼각형 $\mathbf{x_p x_q x_r}$ 위에 있다고 하면, $\mathbf{v_i} = \upsilon\left(\mathrm{aff}(\{\mathbf{x_p}, \mathbf{x_q}, \mathbf{x_r}\})\right)$여야 한다.

$\mathbf{v_i} = \lambda_p \mathbf{x_p} + \lambda_q \mathbf{x_q} + \lambda_r \mathbf{x_r}$ (단, $\lambda_p + \lambda_q + \lambda_r = 1$)로 표현한다면, $\lambda_p > 0, \lambda_q > 0, \lambda_r > 0$이어야 한다.

여기까지는 2 차원일 때의 경우와 비슷하다.

그렇지만, 3 차원일때는 P_i의 면의 집합에서 삼각형 $\mathbf{x_p x_q x_r}$를 빼고, 삼각형 $\mathbf{x_p x_q w_i}$, $\mathbf{x_p x_q w_i}$, $\mathbf{x_p x_q w_i}$, $\mathbf{x_p x_q w_i}$를 추가하면 P_{i+1}의 면의 집합이 되지 않는 경우가 있다.

즉, 선분 $\mathbf{x_p x_q}$ 가 P_{i+1} 의 내부가 되는 경우가 있다.

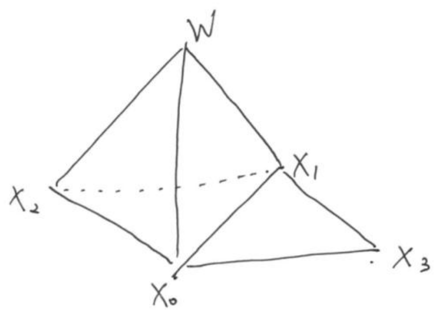

그림 5.

그림에서 삼각형 $\mathbf{x_0 x_1 x_2}$ 와 삼각형 $\mathbf{x_0 x_3 x_1}$ 가 P_i 의 면이고, 삼각형 $\mathbf{x_0 x_1 x_2}$에서 Support Mapping Point $\mathbf{w}$ 를 구했다고 가정하자. 이 때, 선분 $\mathbf{x_0 x_1}$ 가 P_{i+1}의 내부에 있다면, 삼각형 $\mathbf{w x_0 x_1}$ 대신에 삼각형 $\mathbf{w x_0 x_3}, \mathbf{w x_3 x_1}$을 추가해야 될 수도 있다.`

그래서, P_{i+1}의 면을 구하려면, $\mathbf{w_i}$의 위치에서 보이는 P_i 의 면은 빼고, 삭제된 면들의 테두리와 $\mathbf{w_i}$로 만들어진 면을 추가해야 한다.

3D ALGORITHM

```
struct Entry
{
        Vector   x[3];
        Vector   v;
        float lambda[3];
        float distance;
        Entry *adj[3];
        int      index[3];
        bool     deleted;

        bool Internal() { return lambda[0] > 0 && lambda[1] > 0 && lambda[2] > 0;}
};

bool operator < (Entry* entry1, Entry *entry2)
         { return entry1->distance > entry2->distance; }
```

3 차원 환경에서는 P_i의 면을 Entry 구조체로 표현한다. 그리고, 각 면의 이웃 관계도 저장할 필요가 있다.

adj 는 이웃한 면들의 포인터이다.

그리고, index 는 이웃한 면에서의 인덱스이다.

$i=0, 1, 2$ 이면, $\text{entry}{->}\, adj[i]{->}\, adj[entry{->}\, index[i]]\ ==\ \text{entry}$ 여야 한다.

```
struct Edge
{
        Entry *entry;
```

```
    int     index;
};

void silhouette(Entry *entry,  int index,  Vector w,  queue<Edge> &edge_queue)
{
    if (entry->deleted)
        return;
    if (entry->v · w < entry->v · entry->v)
    {
        Edge edge;
        edge.entry = entry;
        edge.index = index;
        edge_queue.push(edge);
    }
    else {
        entry->deleted = true;
        silhouette(entry->adj[(index+1)%3], entry->index[(index+1)%3],
            w, edge_queue);
        silhouette(entry->adj[(index+2)%3], entry->index[(index+2)%3],
            w, edge_queue);
    }
}
```

silhouette ()함수를 보면, 입력으로 주어진 w 에서 entry 가 보이는 지 검사한다.

그 면이 보이는 지 여부는 entry->v · w < entry->v · entry->v 로 검사한다.

만약, 보이는 면이라면, deleted = true 로 표시하고, 나중에 삭제한다. 보이지 않는 면이라면 테두리를 저장하는 큐에 추가한다. 나중에, 테두리와 w 로 면(face)을 만들어 추가하게 된다..

```
Entry MakeEntry(Vertex x0, Vertex x1, Veretx x2)
{
    Entry *entry = new Entry();
    entry->x[0] = x0;
    entry->x{1] = x1;
    entry->x[2] = x2;
    entry->lambda = (Johnson's distance algorithm);
```

```
        entry->v = entry->lambda[0] * x0 + entry->lambda[1] * x1+ entry->lambda[1] * x2;
        entry->distance = |entry->v|;
        entry->deleted = false;
        return entry;
}

Vector EPA3D(Polytope& P)
{
        priority_queue<Entry *> queue;
        for each face {xp,xq,xr} of P
        {
                Entry* entry = MakeEntry(xp, xq, xr);
                entry->adj[] = ...;
                entry->index[] = ...;
                if (entry->Internal())
                        queue.push(entry);
        }
        Vector v;
        float upper_bound = FLT_MAX;
        for ( ; ; )
        {
                Entry* entry = queue.top();
                queue.pop();
                if (entry->deleted)
                        continue;
                v = entry.v;
                Vector w = SupportMapping (P, v);
                upper_bound = min(upper_bound, v·w/entry->distance);
                if (upper_bound - entry.distance >= ε_abs)
                        break;
                queue<Edge> edge_queue;
                for (int i = 0; i < 3; i++)
                        silhouette(entry->adj[i], entry->index[i], w, edge_queue);
                for each edge in edge_queue
                {
                        Entry *new_entry = MakeEntry(edge.entry->x[(edge.index+1)%3],
                                edge.entry->x[edge.index], w);
                        new_entry ->adj[] = ...;
                        new_entry ->index[] = ...;
```

```
                    edge.entry[edge.index] = new_entry;
                    edge.index[edge.index] = 0;
                    if (entry->Internal() && new_entry->distance >= entry->distance
                    && new_entry->distance < upper_bound)
                            queue.push(entry);
                    }
        }
        return v;
}
```

이번에는 오차 때문에 무한 반복되는 것을 피하기 위해 upper_bound 라는 변수를 사용하였다.

upper_bound 는 $\mathbf{v_0} \cdot \mathbf{w_0}/|\mathbf{v_0}|, \mathbf{v_1} \cdot \mathbf{w_1}/|\mathbf{v_1}|, \dots, \mathbf{v_i} \cdot \mathbf{w_i}/|\mathbf{v_i}|$ 중에서 최소 값이다.

설명을 쉽게 하기 위해 Entry 구조체를 메모리 반환하는 부분은 생략하였다.

초기 POLYTOPE 설정

지금까지 P_0를 사면체(tetrahedron)로 가정하였다.

GJK 의 Simplex 가 사면체가 아닌 경우에 대해서도 알아보자.

Simplex 가 점인 경우에는 Penetration Depth 는 0 이기 때문에 EPA 를 실행할 필요가 없다..

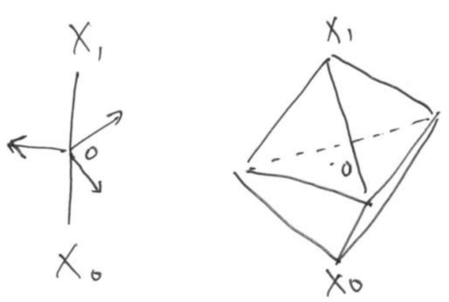

그림 6.

Simplex 가 선분인 경우, 그 선분에 수직인 평면에 속한 3 개의 Support Mapping 을 추가한다.

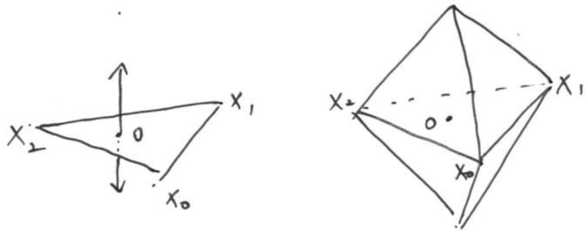

그림 7.

Simplex 가 삼각형인 경우, 삼각형 평면에 수직인 두 개의 Support Mapping 을 추가한다.

GJK 와 EPA 의 결합

EPA 는 Penetration Depth 가 0 에 가까운 물체에 대해서는 잘 동작하지 않는다.

그리고, 결과에 빨리 수렴하지 않아서 비효율적이다.

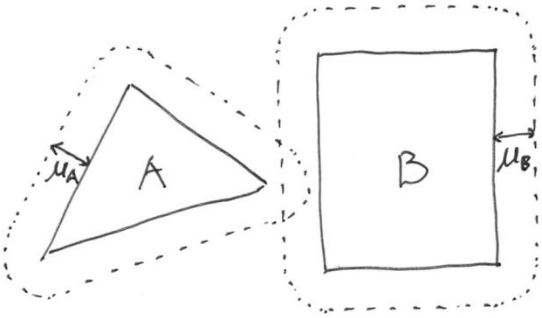

그림 8.

그래서, 물리엔진에서 사용될 때, 물체를 약간 확대하여 처리하면 더 효율적이다.

일단 GJK 를 이용해서, 확대된 두 물체의 충돌 여부를 검사한다.

확대된 두 물체가 충돌하지 않았을 때는 접촉하지 않은 것으로 처리한다.

그리고, 원래 물체는 충돌하지 않았지만, 확대된 물체에서 충돌되었다면, EPA 를 사용하지 않고, GJK 에서 구한 충돌 지점을 그대로 이용한다.

그리고, 더 깊숙하게 충돌한 경우(원래 물체에서 충돌한 경우)에만 EPA 를 이용하여 충돌한 점을 찾는다.

두 물체가 깊숙하게 충돌한 경우에, 물리 엔진의 기능에 의해서 살짝 접촉한 상태로 보정되므로, EPA 를 사용하는 경우가 줄어든다.

또, 물리엔진에서 사용될 때, 이전 frame 의 결과를 보관하고 있다가 다시 사용하면, 더 빨리 수렴하게 된다. 기존 알고리즘에서는 $\mathbf{v_0} \in A - B$ 이어야 했기 때문에, 이전 결과를 그대로 사용할 수 없었다. upper_bound 변수를 이용하여 $\mathbf{v_0}$가 어떤 값이든 상관없도록, 알고리즘을 수정하였다.

```
float PenetrationDepth(Polytope A, Polytope B)
{
        Vector v = (임의의 점);
        float upper_bound = FLT_MAX;
        Set<Vector> W = ϕ;
        for ( ; ; ) {
                Vector w = SupportMapping(A,B,-v);
                if (v · w > 0 && (v · w)^2 / (v · v ) > (μ_A + μ_B)^2)
                        return 0; // v is separating axis
                if ((1 − ε_rel)^2 * upper_bound ≤ v · w)
                        return μ_A + μ_B − √upper_bound;
                v = υ(conv(W∪{w}));
                W = (Convex Hull 이 v 를 포함하는 가장 작은 W∪{w}의 부분집합);
                upper_bound = v · v;
                if (upper_bound <= ε_abs^2 )
                        break;
        }
        return (EPA 를 사용하여 |φ(확대된 A − B)| 를 계산한다. );
```

```
}
```