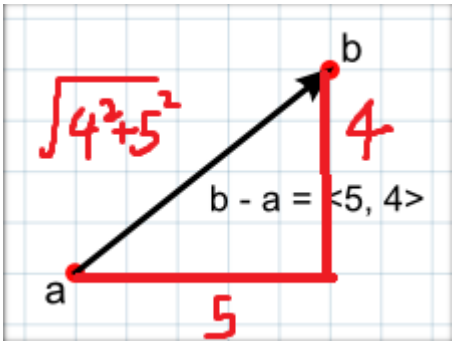
1. Vector

* 요약:
벡터란 방향과 크기를 설명할 수 있는 수학적 개념입니다.
어떠한 객체가 지닌 방향성과 그 방향으로 진행하고 있는 힘의 크기를 표현하고자 할 때 벡터가 사용됩니다.

* 세부 설명:

예를 들어 선풍기 앞에서 종이를 잡고 있다가 놓은 순간, 종이가 바람에 의해 어떤 방향으로 얼만큼 빠르게 날아가는지 벡터로 표현할 수 있을 것입니다.



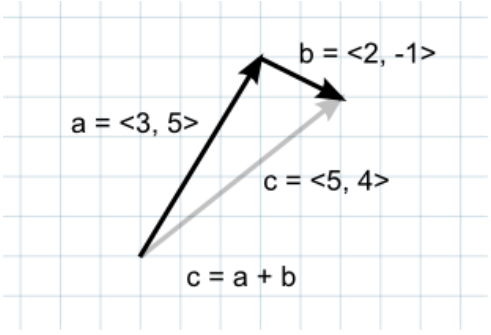
<5, 4> 벡터가 있다면 x축 방향으로 5만큼, y축 방향으로 4만큼 이동하고 있고,
힘의 크기는 삼각형의 빗변의 길이인 √41 이 될 것입니다.

1.1 단위 벡터

단위 벡터는 normalized vector라고도 불리며, 힘의 크기가 1로 고정이 된 벡터를 의미합니다.
위의 그림에 적용한다면 삼각형의 빗변의 길이를 1로 변환시키는 것을 의미하며,
그러기 위해서는 x축, y축 값을 빗변의 길이인 힘의 크기로 나눠야 합니다.
벡터의 힘의 크기를 유지한 채 방향만 반영하고 싶을 경우 단위 벡터를 활용할 수 있습니다.

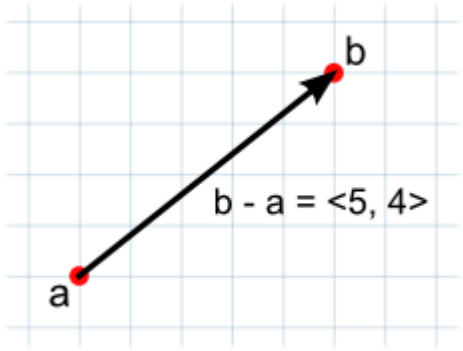
게임 내에서 캐릭터의 이동 속도나, 캐릭터와 적 간의 거리 등을 계산할 때 벡터를 활용할 수 있습니다. 벡터의 활용이란 구체적으로 벡터끼리 덧셈 연산을 하거나, 뺄셈 연산 등을 하는 것을 의미합니다.

1.2 벡터의 덧셈



벡터 a를 로켓의 진행방향과 힘, 벡터 b를 바람의 진행방향과 힘이라고 생각한다면,
바람의 영향을 받은 로켓의 이동방향과 힘이 벡터 c에 일치할 것입니다.
벡터의 덧셈 연산은 벡터들의 순서와 관계없이 동일한 결과값을 가지게 됩니다.

1.3 벡터의 뺄셈



벡터의 뺄셈은 두 객체 간의 거리를 구할 때 주로 사용됩니다. 벡터의 덧셈과 달리 뺄셈 연산 순서에 따라 결과값 벡터의 크기는 동일하지만 나아가는 방향이 달라집니다.

1.4 벡터의 스칼라 곱셈, 나눗셈

스칼라는 벡터와 달리 힘의 크기에 대한 정보만을 가지고 있습니다. 그렇기 때문에 하나의 벡터에 스칼라 값을 곱하거나 나누게 되면 벡터의 방향성을 유지한 채 힘의 크기를 자유롭게 조절할 수 있습니다.

1.5 위치 정보

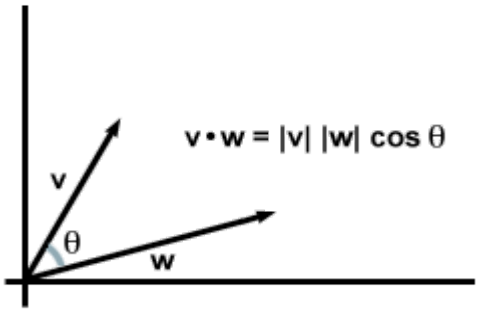
Unity에서 벡터 자료형 자체에는 벡터 연산을 위한 함수들을 포함하고 있지만, 객체의 위치를 표현할 때 활용되기도 합니다.

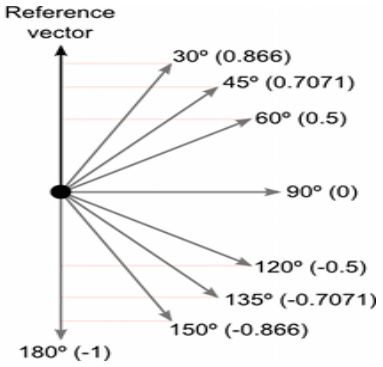
예를 들어 Unity의 Transform 클래스는 position 이라는 프로퍼티를 가지고 있는데, Transform.position은 월드 공간에서의 위치 정보를 Vector3 자료형으로 담고 있습니다. 이러한 사용법은 현재 객체의 진행방향이나 힘의 크기보다는 위치 정보인 x, y, z 좌표에만 관심이 있기 때문에 수학적 개념에서의 벡터와 차이가 조금 존재합니다.

2. 내적 **Dot Product**

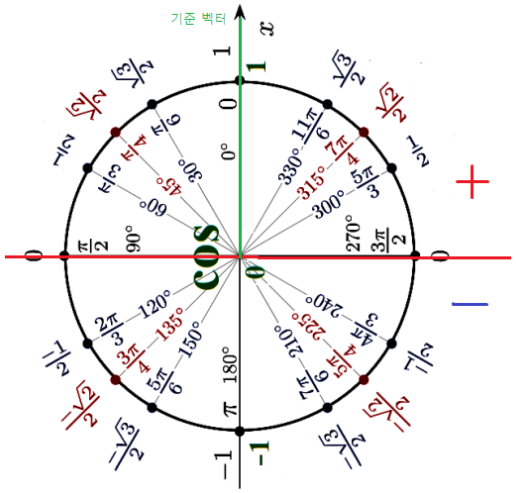
* 요약 :
내적은 두 벡터의 힘의 크기를 곱한 값에 두 벡터 사이의 각에 대한 cos값을 곱한 스칼라 결과값입니다. 한 벡터를 기준으로 다른 벡터가 얼만큼 떨어져 있는지 cos값의 특징을 활용하여 유추할 수 있습니다. 기준이 되는 벡터가 뒤바뀌어도 내적값은 동일합니다.

* 세부설명:



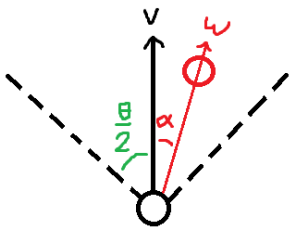


왼쪽 그림을 참고하면서 오른쪽 그림의 Reference vector를 벡터 v로 놓는다면, 각도는 θ값을 의미하고, 괄호 안 값은 cosθ 값을 의미합니다. 만약 θ가 180도일 경우 cosθ은 -1이 되고, 벡터 w가 벡터 v 정반대편을 향하고 있음을 알 수 있습니다.



cosθ값은 각도 θ가 90도 미만일 경우 양수, 90도를 초과할 경우 음수의 값을 가집니다. 일반적으로 힘의 크기는 항상 양수이기 때문에 내적값의 부호는 cosθ값의 부호를 항상 따르게 됩니다. 따라서 내적값의 부호가 양수일 경우 두 벡터가 같은 편에 있고, 음수일 경우 서로 반대편에 있다는 것을 알 수 있습니다.

만약 벡터 v, w가 모두 단위 벡터일 경우 |v| = |w| = 1 이므로 내적값은 cos값과 동일하게 됩니다. 따라서 이 내적값으로 두 벡터가 얼만큼 떨어져 있는지 (얼만큼 회전을 해야 하는지) 알 수 있습니다.



또한 내적을 활용해서 게임에서 플레이어의 시야각 내에 물체가 있는지 판단할 수 있습니다.
위 그림에서 플레이어는 벡터 v 방향으로 바라보고 있고, 플레이어의 시야각은 θ입니다. 빨간색 물체는 플레이어 기준으로 w벡터 방향으로, w벡터 크기만큼 떨어져 있습니다.

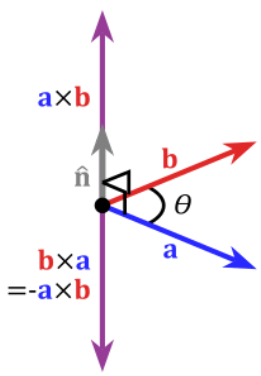
v의 단위벡터와 w의 단위 벡터의 내적값은 cosα 과 일치할 것입니다.
물체가 플레이어의 시야각 내에 존재하기 위해서는 각도 α가 각도 θ/2 보다 작아야 하며,
이는 곧 cosα 값이 cos(θ/2) 보다 커야 한다는 뜻입니다.

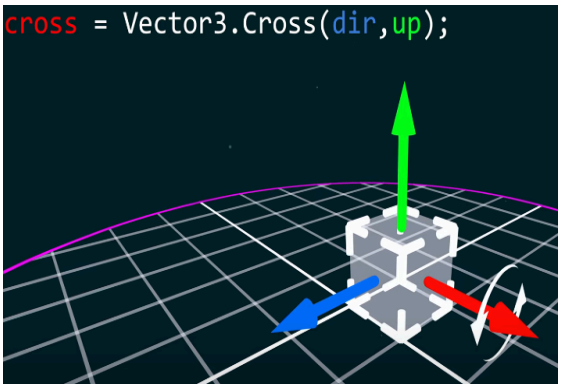
두 벡터 사이의 각에 대한 cos값을 직접적으로 구하지 않고 내적을 활용하는 이유는
내적 함수 연산이 CPU 시간이 훨씬 적게 걸리기 때문입니다.

3. 외적 **Cross Product**

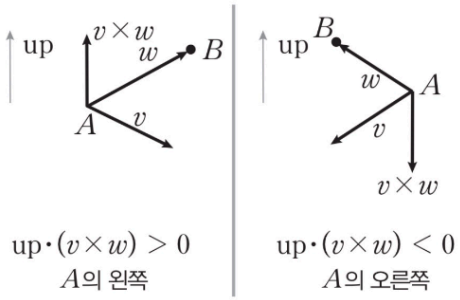
* 요약:
외적은 두 벡터가 이루는 평면에 수직인 벡터입니다. 외적은 두 벡터의 크기를 곱한 값에 두 벡터 사이의 각에 대한 sin값을 곱한 것과 같습니다. 내적과 달리 외적 연산은 입력 벡터의 순서에 따라 외적 벡터의 크기는 동일하되 방향이 달라지게 됩니다.

* 세부 설명:





위의 큐브 사진에서 큐브를 파란색 벡터 방향으로 굴리고 싶을 경우,
파란색 벡터와 초록색 벡터의 외적을 구하면 빨간색 벡터가 나올 것입니다.
그렇다면 빨간색 벡터를 축으로 삼아 반시계 방향으로 90도 회전하면 될 것입니다.



또 다른 활용법은 플레이어의 시선을 파란색 벡터, 적이 있는 위치 벡터를 빨간색으로 가정한다면, 파란색 벡터와 빨간색 벡터의 외적은 초록색 벡터가 될 것입니다.

초록색 벡터의 방향이 게임 내 상향 벡터(Up)와 같은 편일 경우, 연산 순서가 바뀌지 않았으므로 플레이어 기준 적이 왼쪽에 위치하고 있음을 알 수 있습니다 (b는 a의 왼쪽).

외적이 상향 벡터와 반대편일 경우, 입력 벡터의 연산 순서가 바뀐 결과와 같으므로
플레이어 기준 적이 오른쪽에 위치하고 있음을 알 수 있습니다 (b는 -a의 오른쪽).
같은 편인지 판별하는 방법은 두 벡터 간의 내적을 사용하는 것입니다.
즉, 초록색 벡터와 상향 벡터를 내적했을 때 양수가 나올 경우 같은 편,
음수가 나올 경우 서로 반대편에 있음을 알 수 있습니다.

또한 두 벡터의 평행 여부를 판단하고 싶을 때 외적을 이용할 수도 있습니다.

angle	0°	30°	45°	60°	90°	120°	135°	150°	180°
	0	$\pi/6$	$\pi/4$	$\pi/3$	$\pi/2$	$2\pi/3$	$3\pi/4$	$5\pi/6$	π
sin	$\frac{\sqrt{0}}{2}$	$\frac{\sqrt{1}}{2}$	$\frac{\sqrt{2}}{2}$	$\frac{\sqrt{3}}{2}$	$\frac{\sqrt{4}}{2}$	$\frac{\sqrt{3}}{2}$	$\frac{\sqrt{2}}{2}$	$\frac{\sqrt{1}}{2}$	$\frac{\sqrt{0}}{2}$

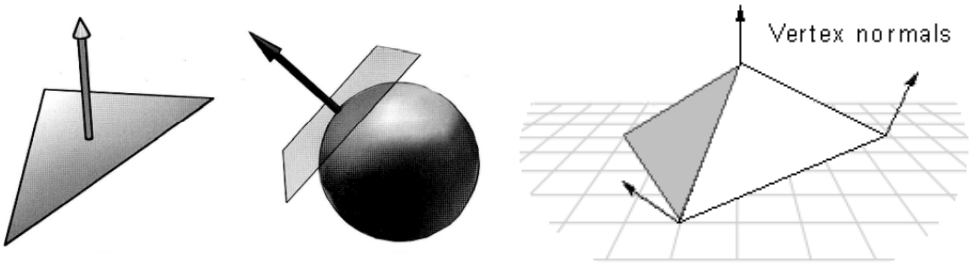
sin값은 0도와 180도에서 0이 됩니다. 그러므로 외적을 했을 때 0이 나올 경우 두 벡터가
평행하고 있다는 사실을 알 수 있습니다.

내적으로도 평행 여부를 알 수 있지만, 그러기 위해서는 두 벡터들을 단위 벡터로 변환해야
하므로 평행 여부를 판단할 때 외적이 통상적으로 쓰입니다.

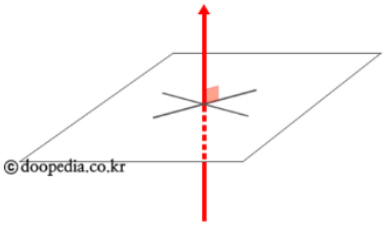
4. 노말벡터

* 요약: 평면에서 수직인 벡터로, 3차원 공간에서 표면의 방향을 나타내는 단위 벡터입니다.

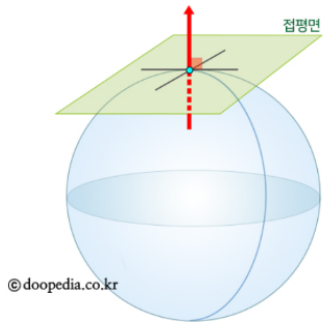
* 세부설명:



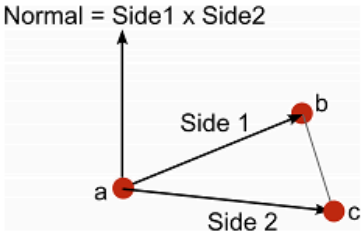
노말벡터는 법선 벡터라고도 불립니다. 빛의 방향과 색상 등과 결합하여 모델의 셰이딩을
만들거나 그림자의 방향 등을 연산할 때 활용됩니다.



면 법선 (face normal) 은 다각형의 모든 점에 수직인 단위 벡터입니다. 평면 위 모든 법선
벡터들은 동일한 방향을 향할 것입니다. 면에서의 빛 반사를 계산하기 위해서는 메시
표면의 모든 점들의 표면 법선이 필요합니다.



표면 법선은 표면의 한 점이 바라보는 방향의 벡터입니다. 휘어진 곡면의 경우에는 특정 점에 접하는 평면을 구해야 합니다. 곡면상의 어느 점을 기준으로 하느냐에 따라 법선 벡터가 매번 달라지며, 모두 다른 방향을 향하고 있을 것입니다.




```
Vector3 a;
Vector3 b;
Vector3 c;

Vector3 side1 = b - a;
Vector3 side2 = c - a;

Vector3 normal = Vector3.Cross(side1, side2);
```

노말 벡터를 구하는 방법은 두 벡터(side1, side)의 외적을 구하는 것입니다.
b - a는 a에서 b로 나아가는 방향의 벡터, c - a는 a에서 c로 나아가는 방향의 벡터를 의미합니다. 즉, 노말 텍터는 side1 과 side2 가 이루는 평면에 대해 수직으로 나아가는 방향의 벡터입니다.

Vector3 z = Vector3.Cross(Vector x, Vector y)

3D 그래픽 라이브러리마다 채택하고 있는 좌표계가 다르며,
DirectX의 경우 왼손 좌표계, OpenGL의 경우 오른손 좌표계를 사용합니다.