■ 평면벡터의 내적

$$\vec{a} \cdot \vec{b} = \left|\vec{a}\right|\left|\vec{b}\right| \cos\theta$$

■ 공간벡터의 내적

$$\vec{a} = (a_1, a_2, a_3), \ \vec{b} = (b_1, b_2, b_3)$$

$$\vec{a} \cdot \vec{b} = a_1b_1 + a_2b_2 + a_3b_3$$

$$\cos\theta = \frac{\vec{a} \cdot \vec{b}}{\left|\vec{a}\right|\left|\vec{b}\right|}$$

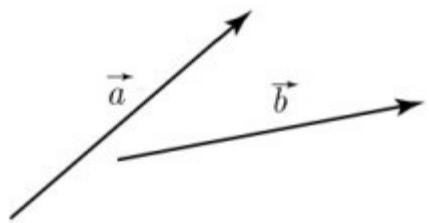
내적(스칼라곱 또는 점곱, dot product, inner product)

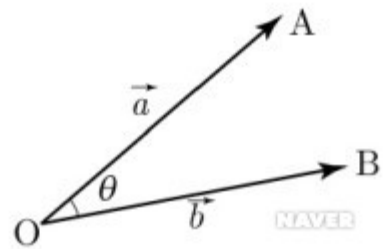
이제 벡터의 곱셈 중 첫 번째로 내적에 대해서 알아보겠습니다.
이제부터 슬슬 어려운 용어가 나오기 시작하는데요, 전혀 걱정하실 것 없습니다.
용어가 어려운 것이지 개념이 어려운 것은 아니기 때문입니다.
내적 이라는 단어가 한자라서 괜히 어려워 보이는 것 뿐이죠.

다음은 백과사전에서 검색한 내적의 정의입니다.

1) 내적의 정의

(1) 두 벡터 $\vec{a}, \vec{b}$가 이루는 각의 크기가θ일 때$\vec{a} \cdot \vec{b} = |\vec{a}||\vec{b}| \cos\theta$ $\cos\theta$를 두 벡터$\vec{a}, \vec{b}$의 내적이라 한다.

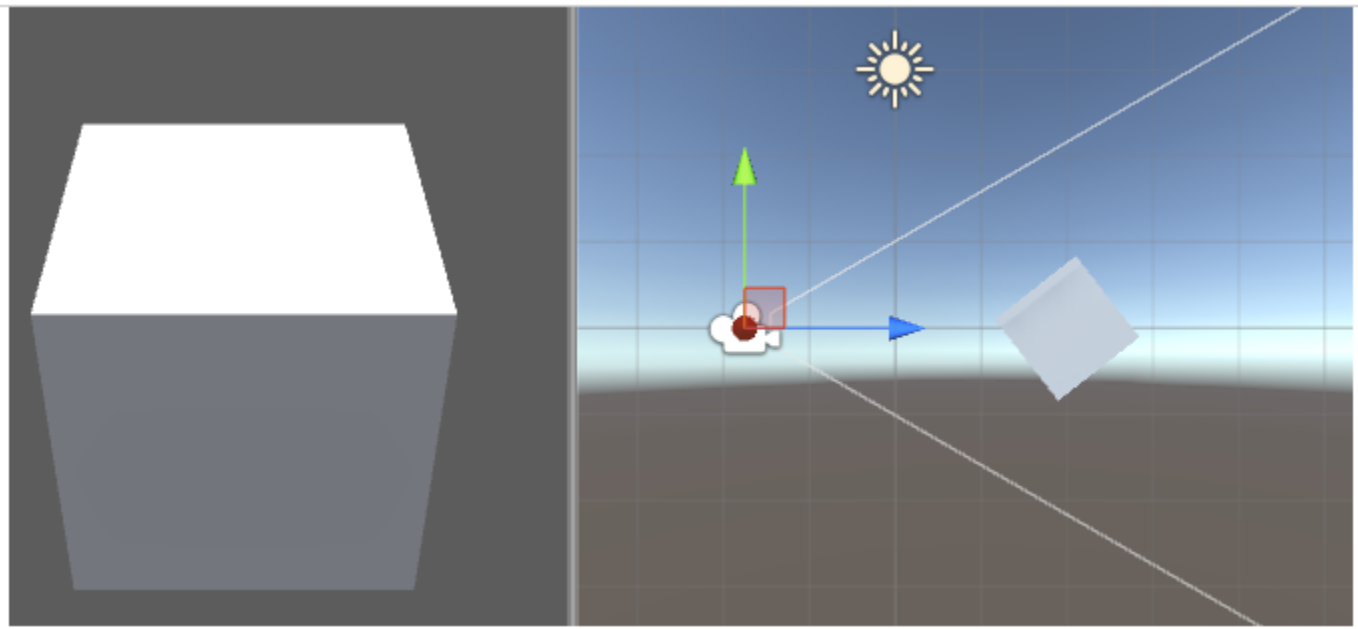




(출처 : http://terms.naver.com/entry.nhn?docId=2073869&cid=47324&categoryId=47324)

내적을 처음 접하는 사람한테는 뭐라고 읽어야 할지도 모를 만큼 어려운 설명입니다.
우리는 사전적인 정의 보다도 실제 게임 프로그래밍에서 내적이 어떻게 활용되는지
그 쓰임에 대해서 알아보도록 하겠습니다.
활용하는 법을 배우고 체득하고 나면 저 공식도 자연스럽게 이해가 되기 때문이죠.

물체가 빛을 받으면 밝은 부분과 어두운 부분이 생기는데
이러한 명암을 컴퓨터 그래픽으로 모델링 할 때 벡터의 내적 연산이 사용 됩니다.
또한 위 사진에서 사과의 뒷부분(보이지 않는)을 렌더링에서 제외 하기 위한 계산에도
벡터의 내적이 사용 됩니다.

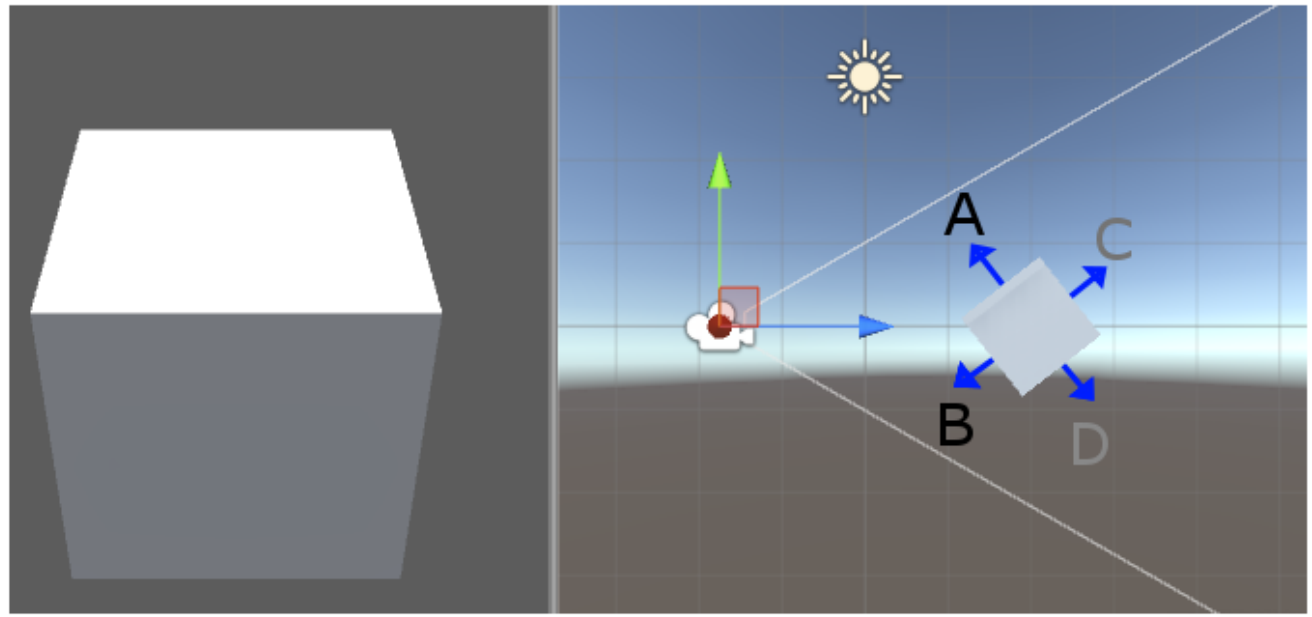


위 그림은 정육면체를 3D 그래픽으로 렌더링 한 모습입니다.
왼쪽 그림을 보면 정육면체의 면들 중 단 두 개의 면만 화면에 렌더링 되고 있는 것을 알 수 있습니다.
카메라의 입장에서 볼 때 정육면체의 뒷부분은 보이지 않는 부분이므로 렌더링 계산에서 제외할 수 있는 부분입니다.
이 때 벡터의 내적을 사용하여 어디가 뒷부분인지 선별하는 것이 가능합니다.
대략적으로 살펴본다면 다음과 같습니다.

> **폴리곤에서 카메라 쪽으로 향하는 벡터와 폴리곤의 법선 벡터를 내적하여**
> **그 결과값이 0보다 작다면(음수) 뒷부분으로 판단 한다.**

역시 글보다는 그림으로 이해 하는게 빠르겠죠?
아래는 위 내용을 그림으로 나타낸 것입니다.

역시 글보다는 그림으로 이해 하는게 빠르겠죠?
아래는 위 내용을 그림으로 나타낸 것입니다.



왼쪽이 렌더링된 결과물이며 오른쪽 그림은 카메라와 물체의 관계를 옆에서 바라본 모습입니다. 파란색으로 표시된 벡터 a, b, c, d는 각 폴리곤의 법선 벡터를 표시한 것입니다.

법선 벡터(노말 벡터라고도 합니다)란 해당 표면에 수직인 벡터를 의미 합니다.

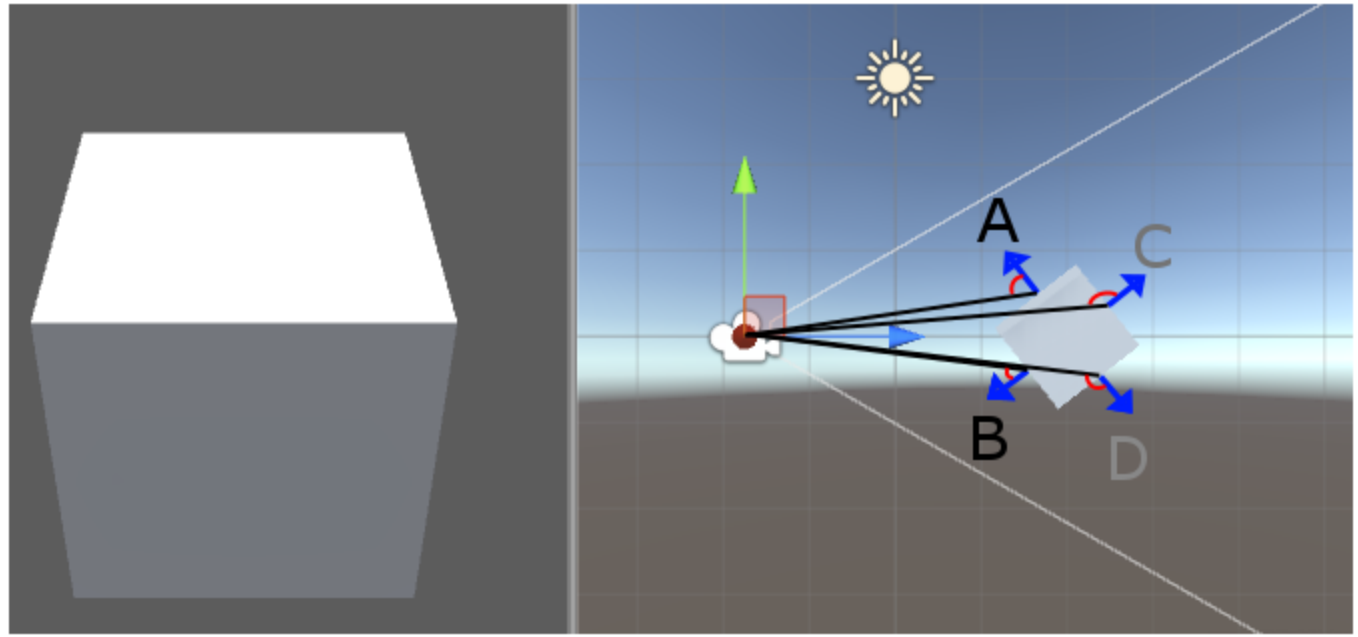
운동장 한가운데에 나무막대기를 세워놓았다고 상상해 봅시다.

나무 막대기는 넓은 운동장 표면(폴리곤)에 직각으로 세워져 있죠.

이 나무 막대기와 같은 벡터를 표면의 법선 벡터라고 부릅니다.

만약 나무 막대기가 평평한 운동장이 아닌 경사진 언덕길에 세워져 있다면 언덕길에 대해 직각일 테니 법선 벡터의 방향도 언덕길만큼 기울어져 있겠죠.

어느 표면에 붙어 있던 해당 표면에 대해 수직인 벡터라고 생각하시면 됩니다.



검은색으로 그려진 선은 각 폴리곤으로부터 카메라쪽으로 향하는 벡터를 나타낸 것입니다.
그림으로 봤을 때 이 두 벡터 사이의 각도가 90도 이하면 해당 폴리곤은 카메라쪽을
바라보고 있다고 판단할 수 있습니다(**a**, **b**).

반대로 각도가 90도 보다 크면 카메라를 등지고 있다고 판단할 수 있죠(**c**, **d**).
카메라를 등지고 있는 폴리곤은 어차피 보이지 않을 것이므로 계산에서 제외한다면 렌더링 성능이 훨씬 올라가겠죠.
그렇다면 도대체 내적이 어떻게 계산 되는 것이길래 위와 같은 구현이 가능한지 지금부터 알아보도록 하겠습니다.

내적의 계산 과정

벡터의 내적을 계산하기 위해서는 먼저 차원이 같은 두 벡터가 필요 합니다.
벡터의 각 성분들끼리 곱셈을 한 뒤 그 결과를 더해주면 내적이 계산 됩니다.
곱하기와 더하기만 할 줄 알면 되기 때문에 어려운건 없겠죠?

a = [2, 2]

b = [-1, 0]

a·**b** = [2 x -1] + [2 x 0]

= -2 + 0

= -2

c = [-2,-2]

b = [-1, 0)

c·**b** = [-2 x -1] + [-2 x 0]

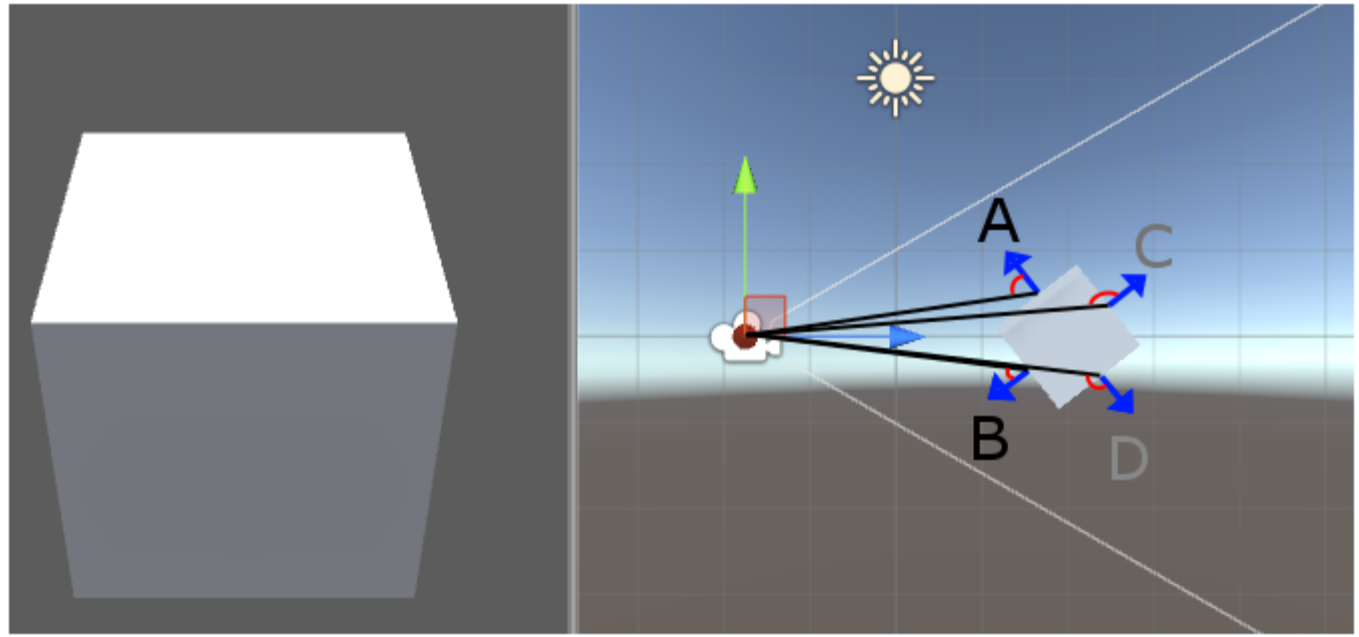
= 2 + 0

= 2

내적의 계산은 이게 끝입니다. 간단하죠?

(·는 내적의 기호입니다)

여기서 알 수 있는 사실은 두 벡터를 내적하면 벡터가 나오는 것이 아니라 단순한 숫자값이 나오게 된다는 것입니다.
또한 이 결과값의 부호를 통해서 폴리곤의 면이 카메라를 향하고 있는지 또는 등지고 있는지를 알아낼 수 있습니다.
내적의 결과가 양수(+)로 나온다면 두 벡터 사이의 각이 90도 보다 같거나 작다는 뜻이며
결과가 음수(-)로 나온다면 두 벡터 사이의 각이 90도 보다 크다는 뜻입니다.



이 그림을 다시 보면,
벡터 **a**와 카메라를 향하는 시선 사이의 각도가 90도 보다 작다는 것을 알 수 있습니다.
벡터 **b**도 마찬가지죠.
반면 벡터 **c**와 **d**는 카메라와의 각도가 90도 보다 크다는 것을 그림을 통해 알 수 있습니다.

즉,
벡터 **a**와 **b**를 각각 카메라를 향하는 벡터와 내적을 하면 그 결과값은 양수로 나오게 되며
두 벡터 사이의 각은 90도 보다 작다는 뜻입니다.
반면 벡터 **c**와 **d**를 각각 카메라를 향하는 벡터와 내적을 하면 그 결과값은 음수로 나오게 되며
두 벡터 사이의 각은 90도 보다 크다는 뜻입니다.

여기서 중요한 것은 각도가 몇 도 인지 까지는 계산할 필요 없이,
내적의 결과값의 부호만으로 각도의 범위를 알아낼 수 있다는 것입니다.
두 벡터 사이의 각도를 구하려면 삼각함수인 acos(아크코사인)함수를 사용해야 하는데
삼각함수를 많이 사용하면 프로그램의 성능이 하락될 수 있습니다.
따라서 몇 번의 곱셈과 덧셈만으로 끝나는 내적을 사용하는 것이 훨씬 이득이 되는 것이죠.

반사 벡터 구하기

내적을 이용하면 반사벡터도 쉽게 구할 수 있습니다.
벽에 부딪친 당구공이 어느 방향으로 튕겨 나가는지 상상해 본다면 반사벡터의 모습이 머릿속으로 그려질 것입니다.

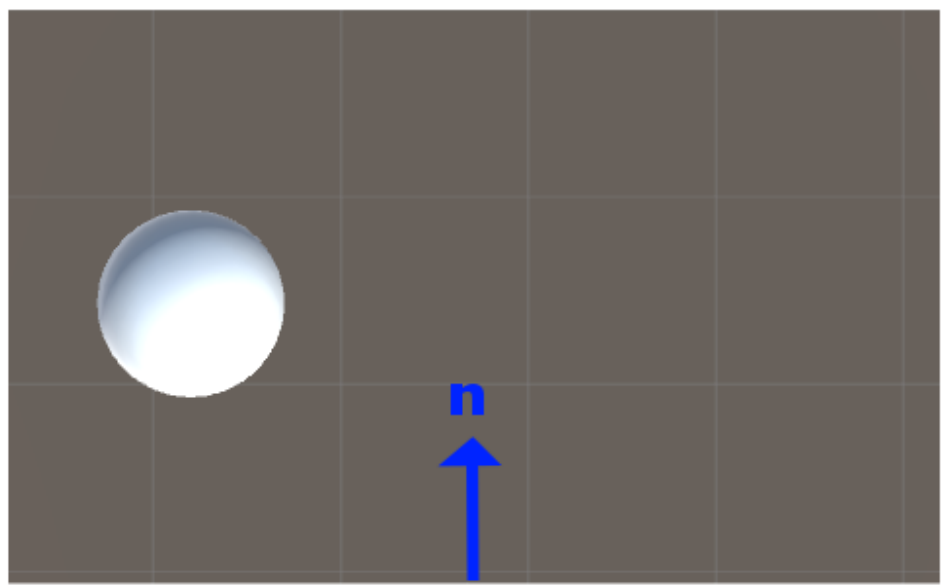
벡터 **p**가 공의 속도벡터라고 한다면 반사벡터를 구하는 공식은 다음과 같습니다.

$$reflect = p + 2n(-p \cdot n)$$

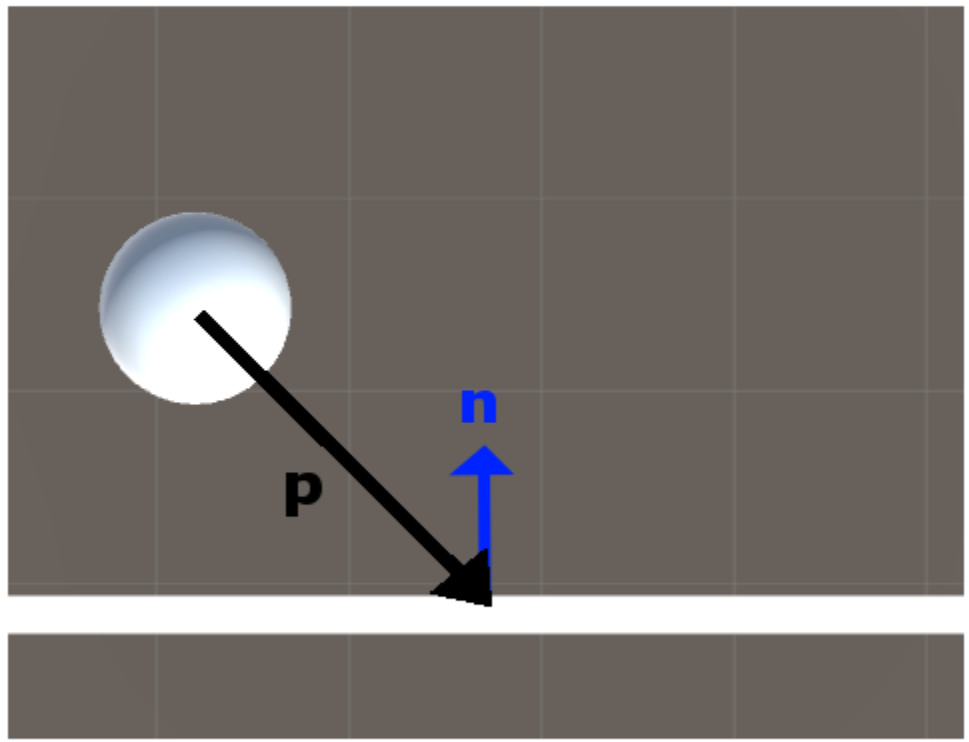
알기 쉽게 프로그래밍 언어로 표현한다면 이렇게 됩니다.

```
·미리보기 | 소스복사·
1.  Vector3 reflect = velocity + 2 * normal * Vector3.Dot(-velocity, normal);
```

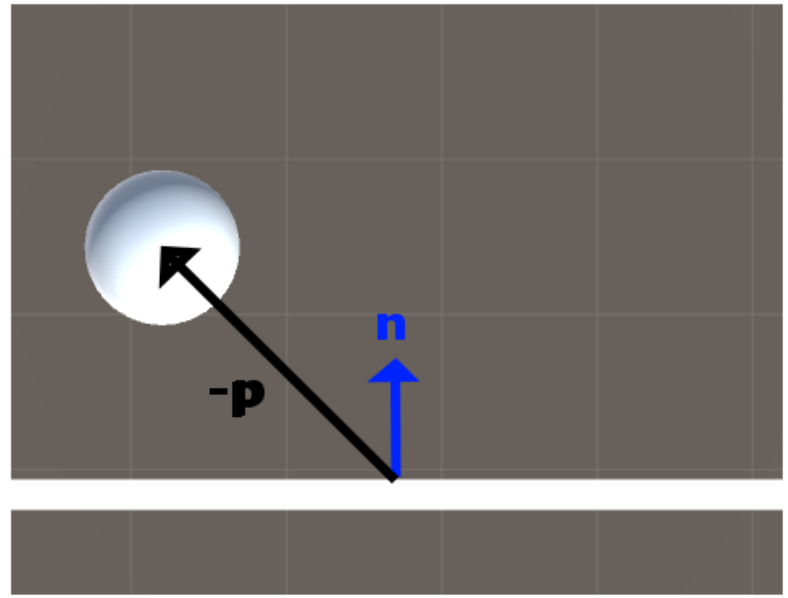
더 알기 쉽게 그림을 통해서 배워보겠습니다.



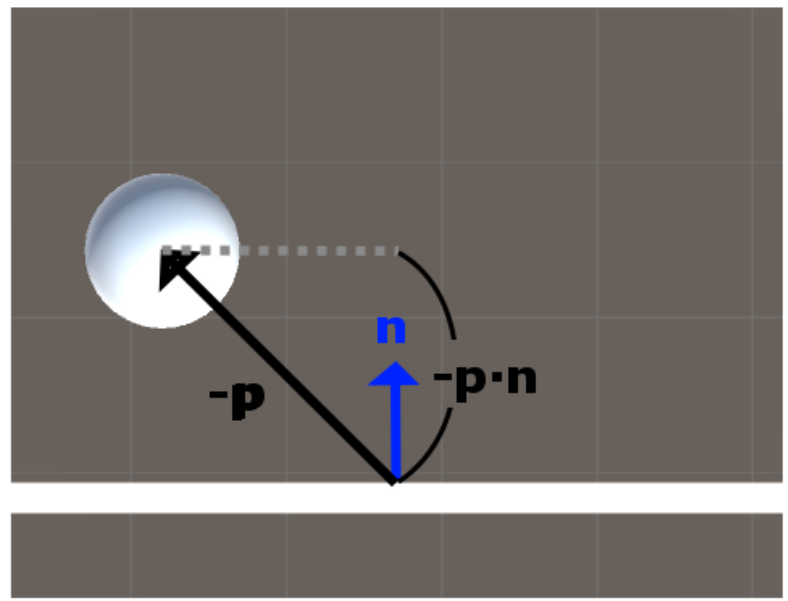
n은 벽의 법선벡터를 뜻합니다.



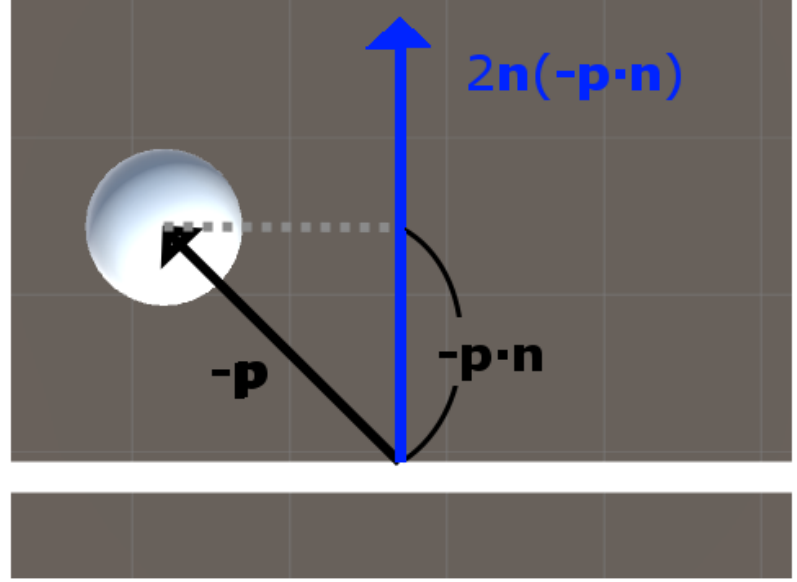
공은 벽을 향해 대각선 방향인 **p**의 속도로 움직이고 있습니다. 여기서 **p**는 공의 **속도 벡터**입니다.



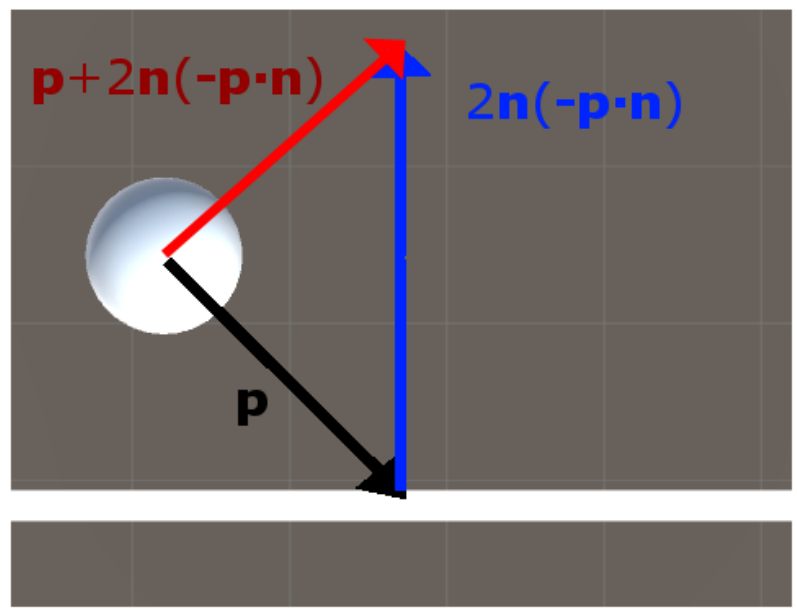
-p : 반사벡터를 구하기 위한 첫 번째 과정으로 **p**의 방향을 바꿔 줍니다.



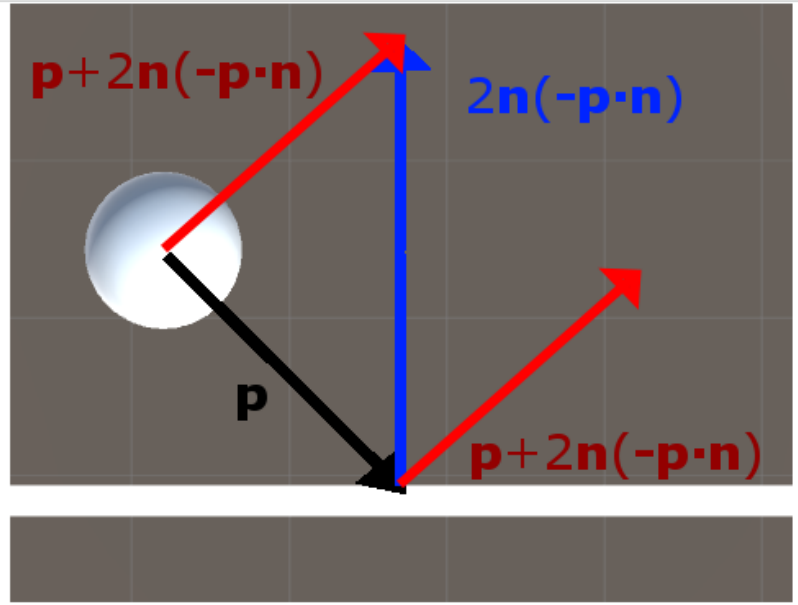
-p·n : **-p**와 벽의 법선벡터인 **n**을 내적 합니다. 내적의 결과로 스칼라값 하나가 나옵니다.



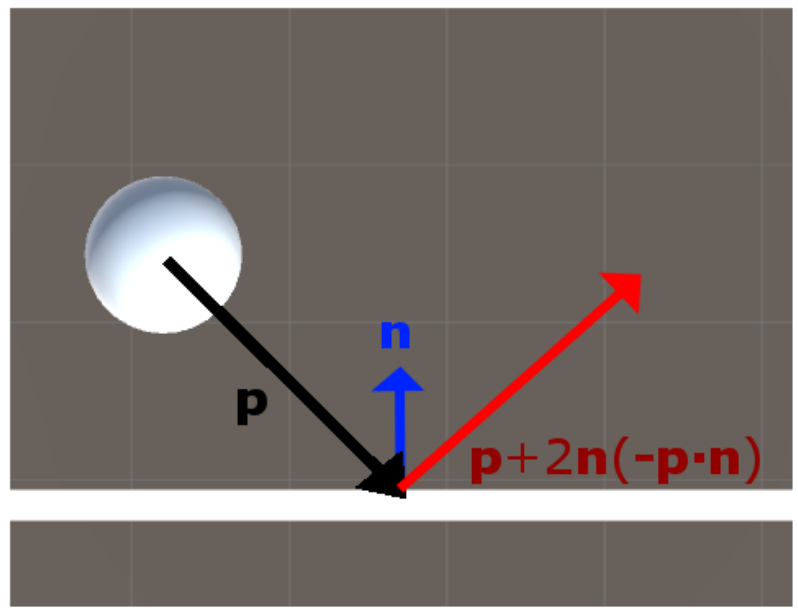
2n(-p·n) : 내적의 결과값과 법선벡터 **n**을 두배하여 곱해줍니다. 결과적으로 법선벡터 **n**을 길게 늘려준 벡터가 생기게 됩니다.



p+2n(-p·n) : 위 결과값에 **p**를 더해서 반사 벡터를 구합니다.



크기와 방향이 같으니 어디에 있던지 결국에는 똑같은 벡터입니다.

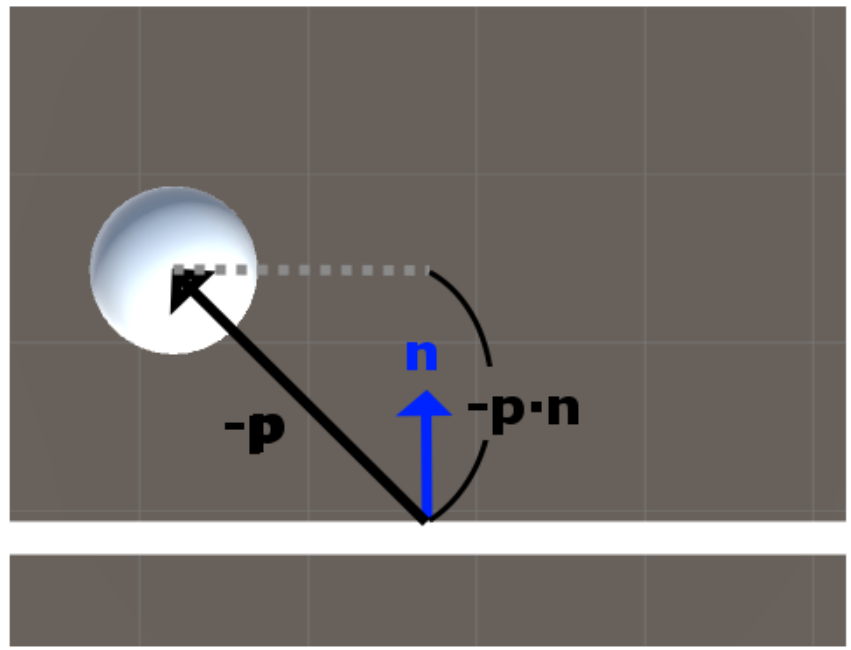


최종적으로 **p**의 반사 벡터가 구해졌습니다.

투영

반사 벡터를 구하면서 내적의 쓰임새에 대해서 알아봤습니다.
하지만 아직 첫부분에 나온 벡터의 정의를 이해하기에는 무리가 있습니다.
왜냐하면 우리는 아직 삼각함수를 배우지 않았기 때문이죠.
내적의 정의에 나온 cos(코사인)은 삼각함수의 한 부분인데 내적과 cos을 연결시켜서 이해하려면
벡터뿐만 아니라 삼각함수에 대한 이해까지도 필요합니다.
이 부분은 앞으로 나올 삼각함수 챕터에서 자세히 다루도록 하겠습니다.

여기서 이해해야 할 부분은 벡터의 내적을 통해서 한 벡터가 다른 벡터에 투영된 길이를 알 수 있다는 것입니다.
투영이란 그림자라고 이해하면 쉬울 겁니다.
반사벡터를 구할 때 벡터 **-p**를 벽의 법선 벡터인 **n**과 내적 하여 계산했는데요,
이 내적의 결과로 나온 값은 벡터 **-p**를 법선 벡터 **n**에 투영한 길이가 됩니다.
단, 법선 벡터 **n**은 반드시 단위 벡터 상태로 되어 있어야 합니다.
(단위 벡터는 길이가 1인 벡터를 의미 합니다)



아까 반사 벡터를 구할 때 봤던 그림입니다.
벡터 **n**을 위로 쭉 이은다고 생각하면 가상의 선이 그려지겠죠.
이 선을 향해서 벡터 **-p**를 직선으로 그었을 때 만나는 점, 그 점의 길이가 바로 내적의 결과값 입니다.
위 그림에서는 점선으로 표시한 부분과 벡터 **n**의 연장선이 만나는 지점이 됩니다.
이 값을 수학적인 방법 없이 구하려고 한다면 머리가 좀 아플겁니다.
(물론 자로 정확하게 그어서 그 길이를 재면 되겠지만 컴퓨터 메모리상에서는 자를 사용할 수 없으니
수학적인 방법을 쓸 수 밖에 없습니다)