2016년 3월 2일

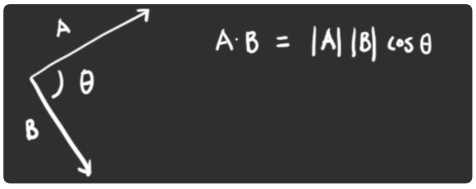
게임에서 내적과 외적(Dot Product and Cross Product in Game)

admin 「개발 이야기」 2 Comments

게임 개발에서는 벡터 연산(Vector Operation)을 자주 사용한다. 이는 벡터로서의 접근이 직관적이며, 방정식이나 복잡한 계산을 피하고 문제를 훨씬 쉽고 간단하며 효율적으로 해결하게 해주기 때문이다. 그래서 물리 엔진을 만들 때 벡터 연산의 적용을 가장 우선적으로 염두해 두는 것이 일반적이다. 속도, 바람, 저항, 충돌, 위치 판단 등 많은 것들이 벡터로 표현된다.

벡터 연산에서 가장 기본은 바로 내적(Inner Product or Dot Product)과 외적(Outer Product or Cross Product)이다.

먼저, 벡터의 내적은 아래와 같이 정의한다.



즉, A, B 벡터의 내적은 A 벡터와 B 벡터의 크기를 각각 곱한 다음 사이각의 cosθ 값을 곱한 스칼라 값이 된다. 벡터와 벡터의 내적의 결과는 벡터가 아닌 스칼라 값이다.

내적의 특성은 다음 몇가지로 정리된다.

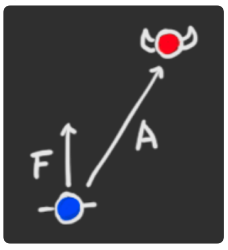
1) 자기 자신과 내적하면 제곱이다.
cosθ 값이 자기 자신이기 때문에 1이 된다. 결과적으로 같은 벡터 2개를 내적하면 제곱이 된다.

2) 두 단위벡터가 평행하면 절대값 1이다.
벡터 두개가 평행하는 경우는 같은 방향으로 향하거나, 반대 방향으로 향하는 것이다. 따라서 cosθ 값이 1 혹은 -1 이다. 절대값을 취하면 1이된다.

내적을 이용하는 예를 몇가지 들어보자.

1) 두벡터의 사이각이 얼마일까?
두 벡터의 사이각을 내적을 통해서 구할 수 있다. 벡터 A, B의 값을 알고 있다면 크기를 알 수 있고, 내적 계산을 이용하여 arcosθ를 통해 구할 수 있다.

2) 주인공의 방향벡터가 있고, 적이 주인공의 앞에 있는지 뒤에 있는지?



나와 적의 거리 차이로 나오는 벡터 A와 나의 Forward 벡터 간에 내적을 하면 각도가 -90 ~ 90 사이에 있으면 앞에 있는 것이고, 반대는 뒤에 있는 것이다. 따라서 내적시에 cosθ 값이 0보다 크면 앞쪽에, 0보다 작으면 뒤쪽에 있는 것으로 판별 가능하다.

About Rapapa

Rapapa는 즐거운 일터와 더 나은 세상을 꿈꾸는 Game & Web 개발자입니다. More...

카테고리

「非 개발 이야기」 (36)

「개발 이야기」 (55)

Recent Comments

Jung: 정말 좋네요.....

colanet: 다른건 다 이해했는데요.... 외적의 활용 2번이 잘 이��...

Cokwa: ?? 오히려 연산은 Tangent Space 쪽이 더 많은데요...

우택: 지식의 무료 나눔, 심도 있는 포스팅 감사합니다~!...

Bugger King: 명쾌한 포스트 감사합니다~!...

김동환: 저 같은 경우 예제를 따라 하고 링크 걸어 주신 리소��...

Victor: 정말 좋은 글입니다 +_+)b...

김선달 (VANDIT KING): 이해가 잘 가는 멋진 글입니다. 감사합니다....

incago: 도움이 많이되는 포스트네요. 감사합니다....

Joowon Yoo: 정리가 너무 잘되어있네요. 좋은자료 감사합니다. 근...

Latest Tweets

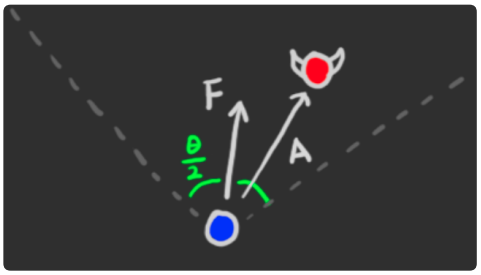
맥에서 Unity Editor인 MonoDevelop의 대안으로 Consulo가 있었는데 간만에 다운받아 보니 거의 완벽히 대체 가능하고 기능도 더 풍부함. v2.0으로 가며 안정화도 상당히 된듯. github.com/consulo/consulo

RT @russian_market: Soon. This is not a @netflix show. pic.twitter.com/4u26Evss6p

RT @unity3d: We're excited to expand our 2D offerings with skeletal animation software Anima2D. bit.ly/2gNXld3

RT @jbevain: Unity 5.5 ships with a new C# compiler with better debug information. Improved breakpoints, stepping and locals info in @VSToo...

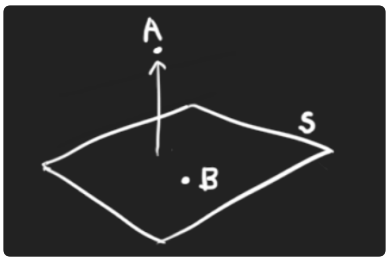
3) 적이 주인공의 시야각안에 들어와 있는지 아닌지?



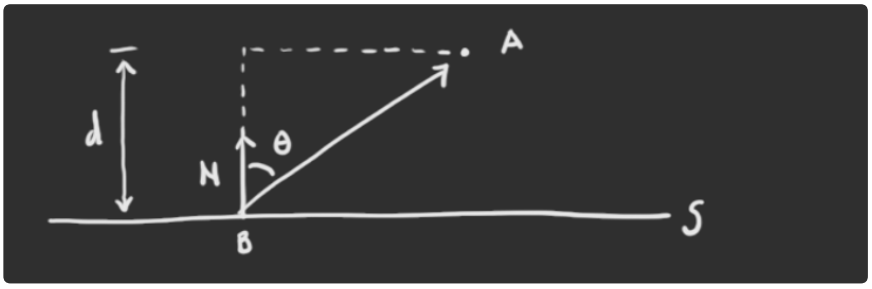
주인공의 시야각을 θ라고 하면, Forward 벡터와 적과 주인공의 거리 차이로 나오는 벡터 A 간의 내적을 통해 나오는 각도 값이 θ/2 를 넘지 않아야 시야 내에 존재한다는 것을 판별할 수 있다. 역시 각도를 다 구하기보다 cos 값으로 판단해야 연산이 적다.

4) 점 A와 평면 평면 S 간의 최단 거리는 아래와 같이 내적을 이용해 구할 수 있다. (B는 평면 상의 점, 평면과 d는 최단 거리)

$$(\overrightarrow{A-B}) \cdot \vec{n} = d$$



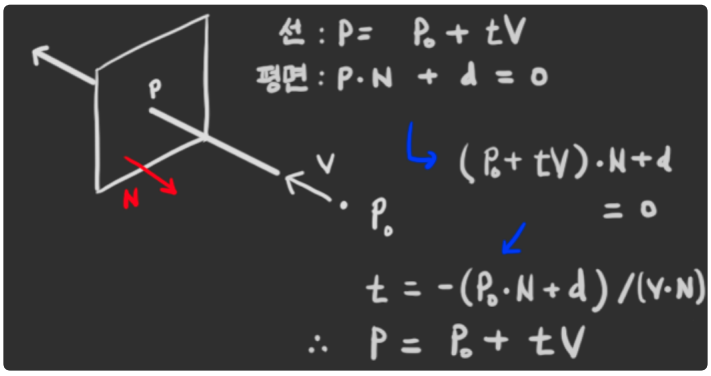
이는 단면을 그리면 조금 더 쉽게 이해된다.



벡터 BA와 평면의 Normal 벡터(단위 벡터)와 내적을 구하면, |BA| * cosθ 의 값이 된다. (평면의 Normal 벡터의 크기는 Normalized된 1이라고 가정). |BA|에 cosθ 값을 구한 값은 그림에서 보듯이 직각 삼각형의 형태가 되므로, 나머지 한변의 길이, 즉 d가 나오게 된다. 이는 BA 벡터의 Normal 벡터로의 정사영 개념이다.

5) 위에서 거리값이 0 이면 평면 위의 점이된다.

6) 선이 평면과 접하는 접점을 구할 때도 내적이 사용된다.



이 외에도 내적을 응용하거나 사용하는 부분은 많다.

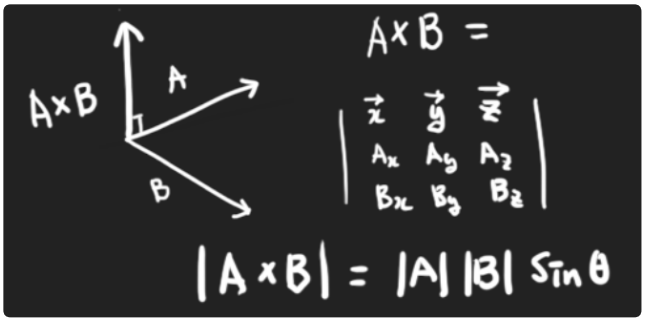
내적에 대해서는 이 정도로 정리하고, 이제 외적(Outer Product)에 대해서 알아보자. 외적은 아래와 같이 정의한다.

Race condition?
stackoverflow.com/questions/3451...

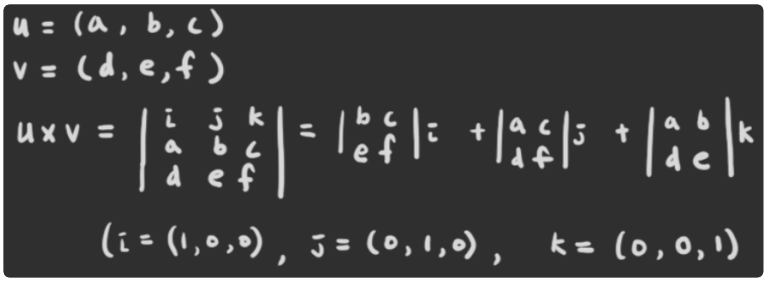
Blog Post : Command Pattern(명령 패턴) in Unity C# is.gd/o5H0Zq

RT @ThingsWork: This is how 3D drawing on the Microsoft Surface Studio works pic.twitter.com/LpQAdYsHs3

→ Follow me



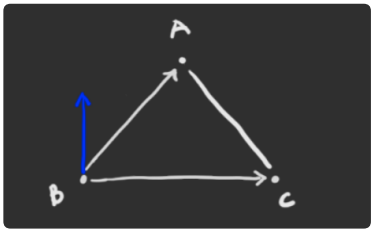
벡터의 외적의 결과는 내적과 달리 또 다른 벡터이다. 외적은 위 그림에서와 같이 Determinant(행렬식)으로 계산할 수 있는데, 주로 사용하는 3차원 벡터의 3×3 행렬식의 풀이는 중고등학교 때 배운 것 처럼 다음과 같다.



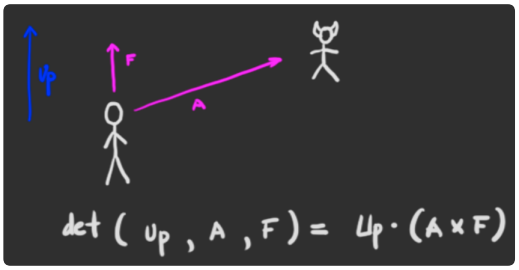
또한, AxB 외적 벡터의 방향은 수학에서는 오른손 법칙을 사용한다. 게임은 엔진이 사용하는 좌표계에 따라 다른데, OpenGL에서는 오른속 법칙, Unity는 왼손 법칙에 따른다. 두 벡터의 외적의 결과는 역시 벡터이다. 그리고 외적 벡터의 크기는 두 벡터의 크기에 sinθ를 곱한 값과 같다.

외적의 특성은 다음 몇가지로 정리된다.
1) A와 B 벡터의 외적, AxB는 A와도 수직(perpendicular)이고, B와도 수직이다.
2) 내적과 달리 교환 법칙이 성립하지 않으며 순서를 바꾸면 반대 방향의 벡터가 나온다.
v x u = -(u x v)
3) 내적과 동일하게 분배 법칙은 성립한다.
u x (v + w) = (u x v) + (u x w)
4) 외적 벡터의 크기는 평행 사변형의 넓이이다. (밑변 * 높이로 구하는 평행 사변형 넓이에서 sinθ 값이 높이)
5) 두 벡터가 평행하면 크기는 0이다. sin0 의 값이 0이기 때문이다.

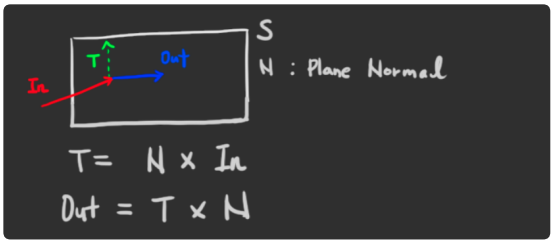
외적을 실제 개발에서 적용하는 예를 몇가지 들어보자.
1) 평면의 법선 벡터를 구할 때 쓴다.
당연하겠지만, 삼각형의 세점을 알고 있다고 할 때, 벡터 2개를 구할 수 있게 되고, 그 두 벡터의 외적을 구하면 해당 삼각형이 속한 평면의 법선 벡터를 구해낼 수 있게 된다. 보통 Modeling Data의 Polygon Normal 벡터를 구할 때 많이 쓴다.



2) 적이 캐릭터의 오른쪽에 있는지 왼쪽에 있는지 판별할 때도 이용된다. 월드 좌표계의 Up 방향 벡터를 Up, 캐릭터의 Forward 방향 벡터를 F, 캐릭터와 적간에 생기는 벡터를 A 벡터라고 하면, Up · (A × F) 값, 즉 Det(Up, A, F) 값이 0보다 크면 오른쪽, 0 보다 작으면 캐릭터의 왼쪽에 있음을 판별할 수 있다. 이는 외적으로 생긴 벡터의 방향이 Up 벡터의 방향과 -90~90도 즉 예각을 이루면 cos값이 0보다 큰 원리를 이용한 것이다.



3) 벽에 부딪혀서 미끄러지는 캐릭터의 방향을 계산할 때도 사용할 수 있다.



벡터와 선형 대수학을 잘 몰라도 물리가 Heavy하게 들어가는 게임이 아닌 이상 개발하는 데는 표면적으로 별 문제가 없다. 다만, 더 좋은 코드를 만들어 내려 욕심이 있는 개발자, 더 자유롭게 코딩하고 싶은 영혼이라면 이 부분을 놓쳐서는 안될 것이다.

Tweet This Post

« 리더는 제일 똑똑할 필요가 없다

Unity in GDC 2016 »