Chapter 09: 방향, 크기, 회전

레트로의
유니티
게임프로그래밍 에센스

이 제 민 지음

Contents

CHAPTER 09 방향, 크기, 회전

9.1 벡터 수학

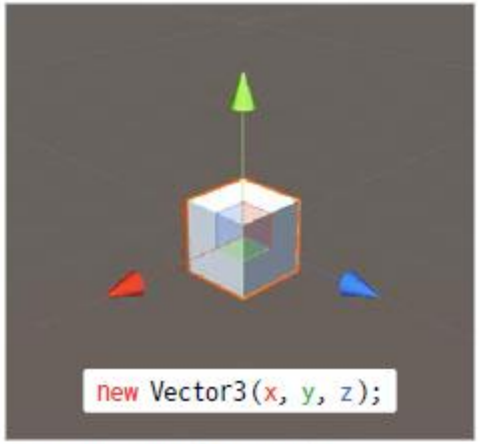
◦ 9.1.1 벡터의 정의

- 벡터는 위치, 방향, 회전, 속도, 크기를 비롯한 온갖 종류의 계산에 사용됨.
- 유니티는 3D 벡터를 나타내는 Vector3를 사용해서 3D 공간에서의 x, y, z 좌표를 표현함.

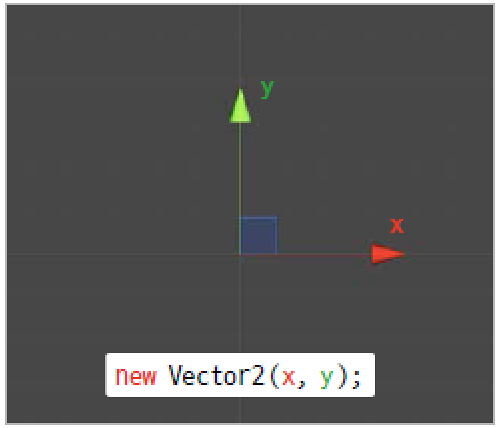
• 물리학자, 공학자, 게임 개발자에게 벡터는 공간상의 화살표로 사용됩니다.
  예) (10, 5, 0)은 오른쪽으로 10, 위쪽으로 5만큼 이동하는 화살표

• 데이터를 다루는 프로그래머에게 벡터는 나열된 숫자 데이터를 묶는 단위입니다.
  예) (172, 64)은 키 172, 몸무게 64를 나타내는 데이터

• 수학자에게는 벡터 연산을 만족하고 정해진 개수의 원소를 가지면 무엇이든 벡터입니다.



▶ Vector3는 3차원 공간에 대응



▶ Vector2는 2차원 공간에 대응

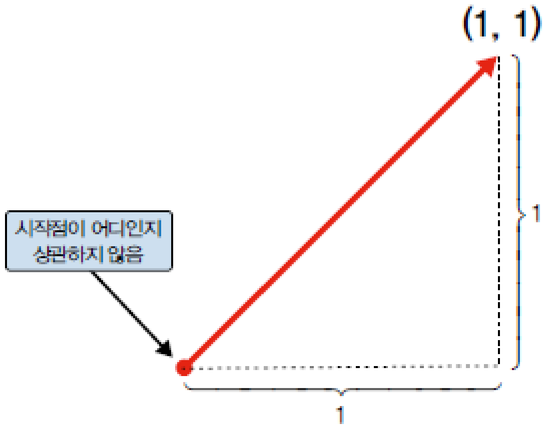
9.1 벡터 수학

◦ 9.1.2 절대 위치와 상대 위치

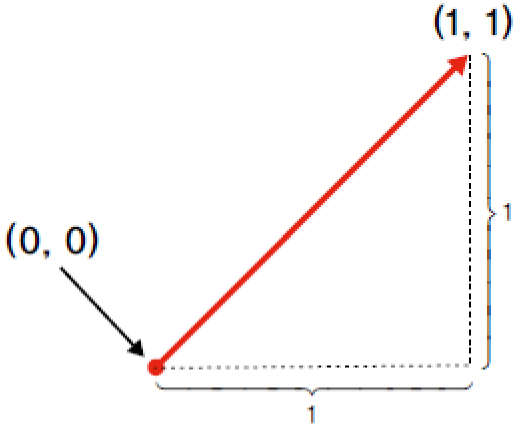
- 벡터는 방향과 크기를 가짐.

• **상대 좌표 :** (내가 어디 있는지는 모르겠지만) 현재 좌표에서 (1, 1)만큼 더 가려고 한다.

• **절대 좌표 :** 게임 세상 속에서 나의 좌표가 (1, 1)이다.



▶ 상대적인 방향과 크기로서의 (1, 1)



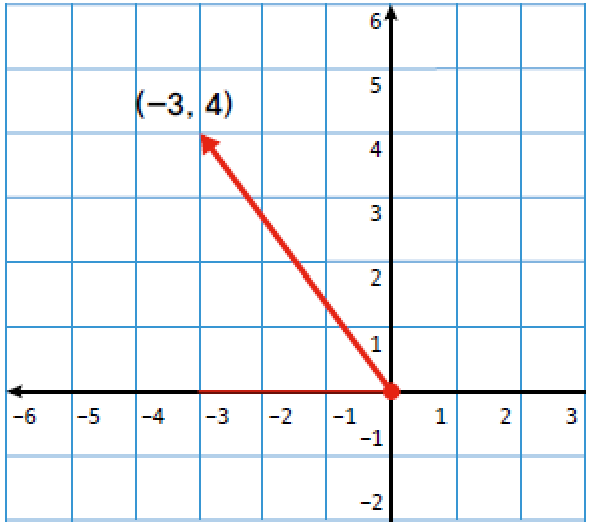
▶ 절대적인 좌표로서의 (1, 1)

9.1 벡터 수학

◦ 1.3 벡터의 크기

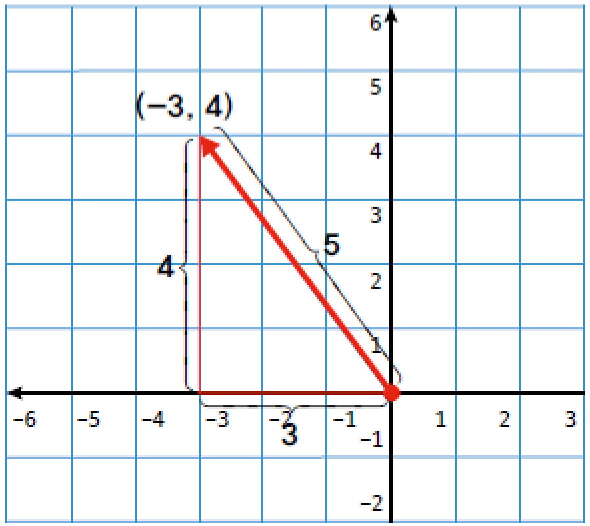
- 2D 벡터 (-3, 4)는 다음 그림과 같이 표현됨.

$$\text{벡터의 크기} = \sqrt{x^2 + y^2 + z^2}$$

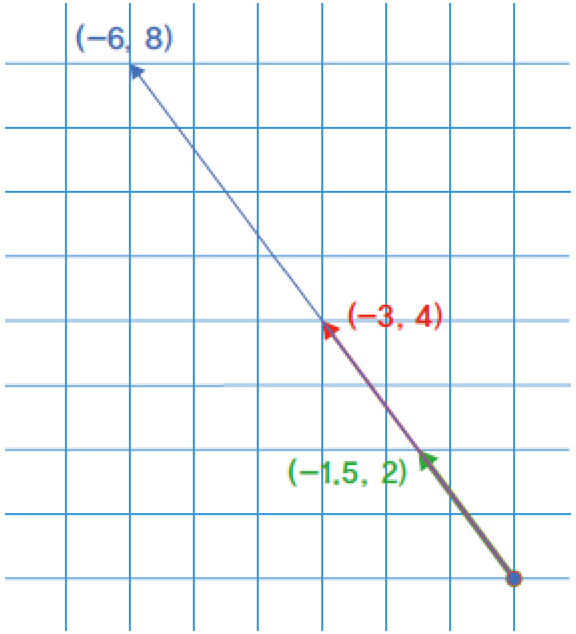


▶ (−3, 4)를 나타낸 벡터

$$(-3, 4)\text{의 크기} = \sqrt{(-3)^2 + 4^2} = 5$$



▶ Vector2 (−3, 4)의 크기



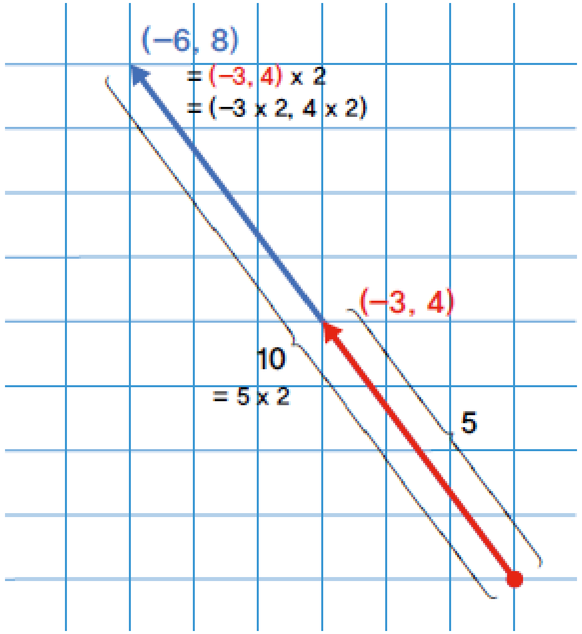
▶ (−3, 4)와 방향은 같지만 크기는 다른 벡터들

9.1 벡터 수학

◦ 9.1.4 벡터의 스칼라 곱

- (-6, 8)의 크기 10은 (-3, 4)의 크기 5의 2배임.

$$(-3, 4) \times 2 = (-3 \times 2, 4 \times 2) = (-6, 8)$$

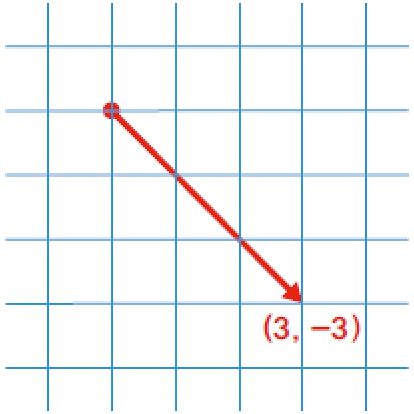


▶ 벡터에 배수 취하기

9.1 벡터 수학

- 9.1.5 방향벡터

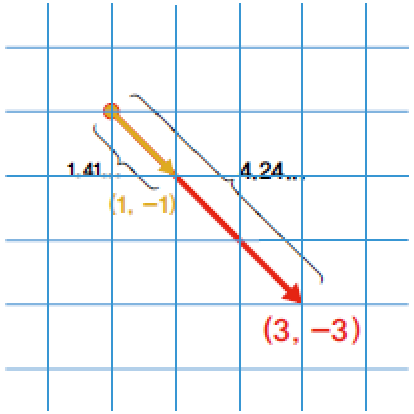
  - (3, -3)이라는 2D 벡터를 생각해봄.



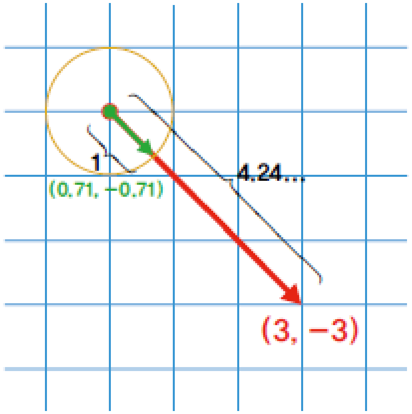
▶ 벡터 (3, −3)

(3, −3) = (방향) × (속력 또는 이동거리)

(3, −3) = (1, −1) × 3



▶ (1, −1)의 3배에 해당하는 (3, −3)



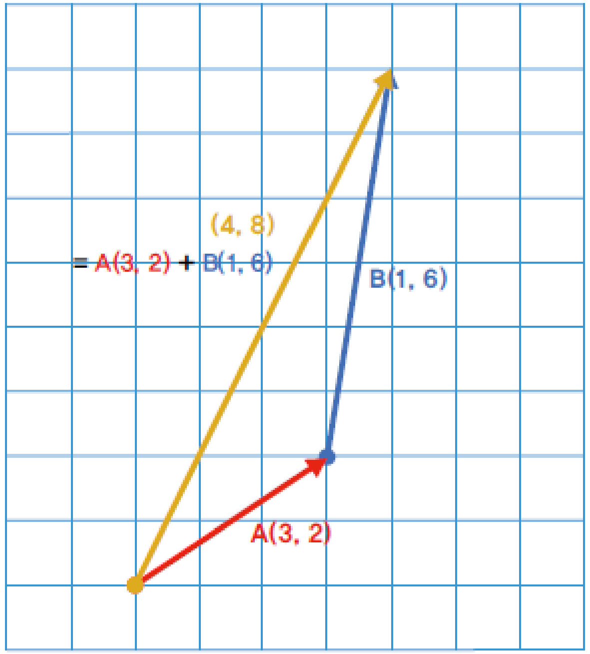
▶ 방향벡터 구하기

9.1 벡터 수학

◦ 9.1.6 벡터의 덧셈

- 벡터 간에는 덧셈이 가능함.

A + B = (3, 2) + (1, 6) = (3 + 1, 2 + 6) = (4, 8)

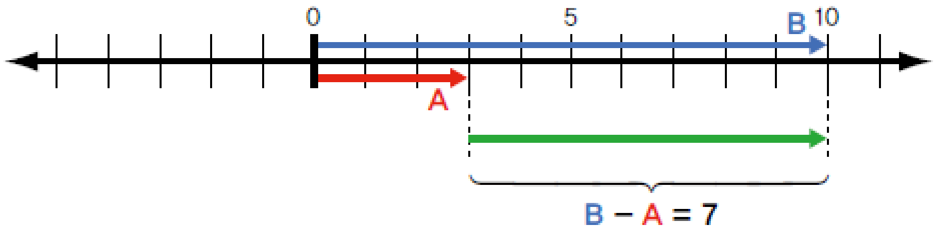


▶ A + B는 A만큼 이동한 상태에서 B만큼 더 이동한 것

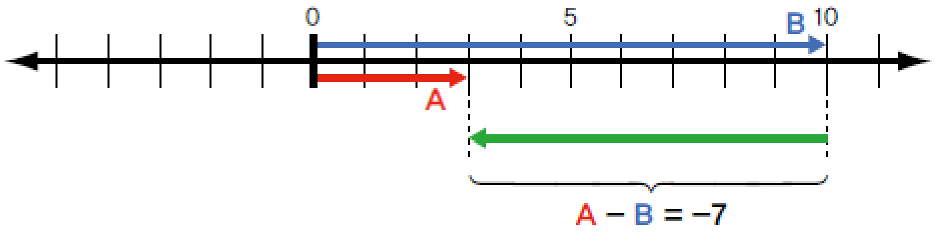
9.1 벡터 수학

◦ 9.1.7 벡터의 뺄셈

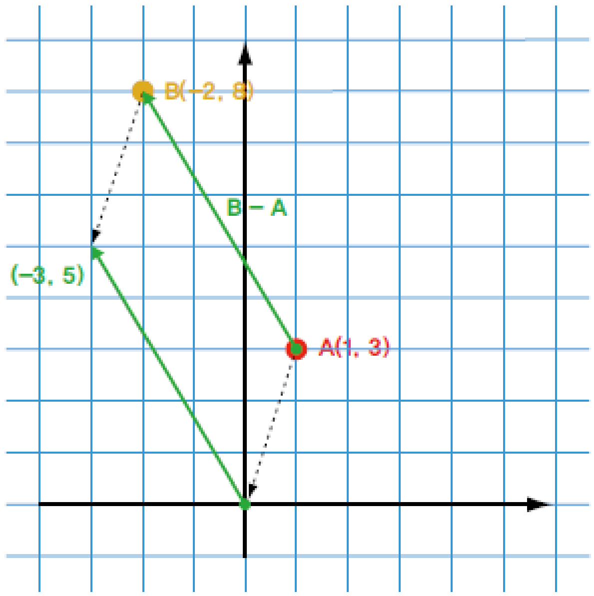
- 벡터의 뺄셈을 이해하기 전에 먼저 일반적인 뺄셈의 의미를 이해해봄.
- 벡터의 뺄셈으로 어떤 물체가 다른 물체를 추적할 때 어떤 방향으로 얼마만큼 가야 하는지 알 수 있음.



▸ B − A



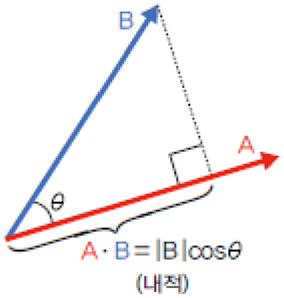
▸ A − B



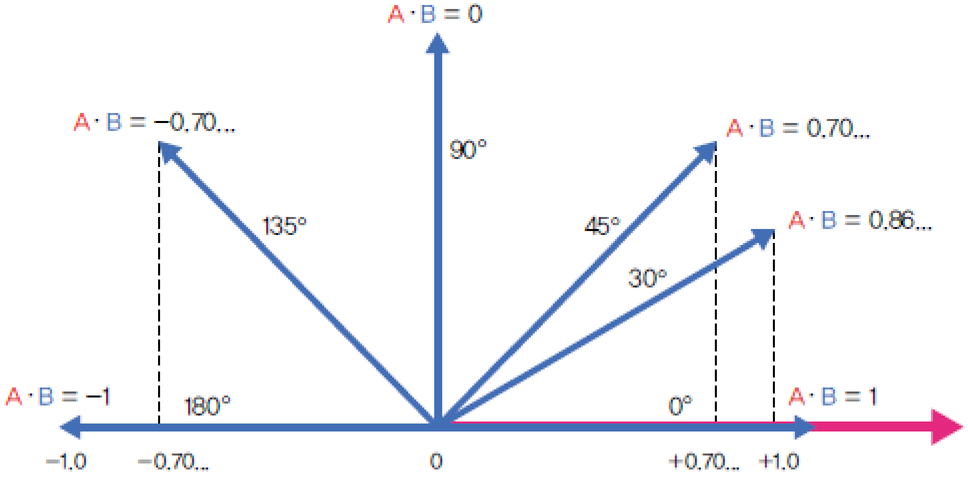
▸ 벡터의 뺄셈이 공간상에서 가지는 의미

9.1 벡터 수학

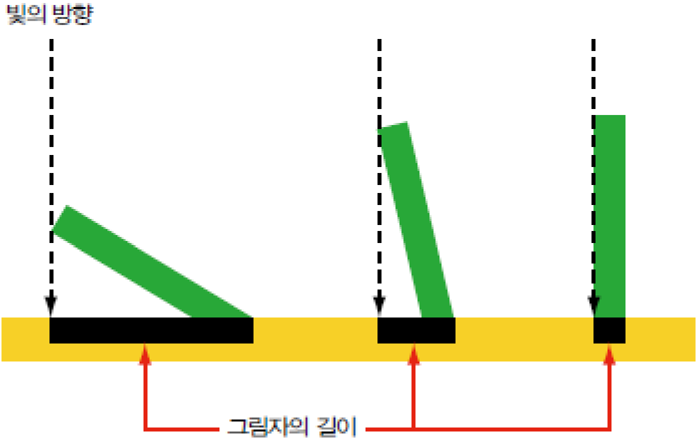
- 9.1.8 벡터의 내적
- \- 벡터의 내적은 어떤 벡터를 다른 벡터로 '투영'하는 연산으로, 점 연산이라고 부르기도 함.



▶ A · B를 구하는 공식

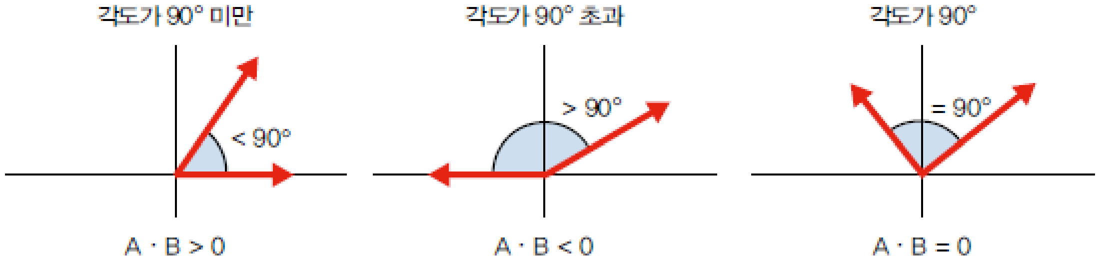


▶ 여러 각도에 대한 대표적인 내적값



▶ 수직에 가깝게 막대기를 세울수록 그림자가 짧아진다

둘 사이의 각도	내적 결과
0°	+1
0° ~ 90°	+1 ~ 0
90°	0
90° ~ 180°	0 ~ −1
180°	−1



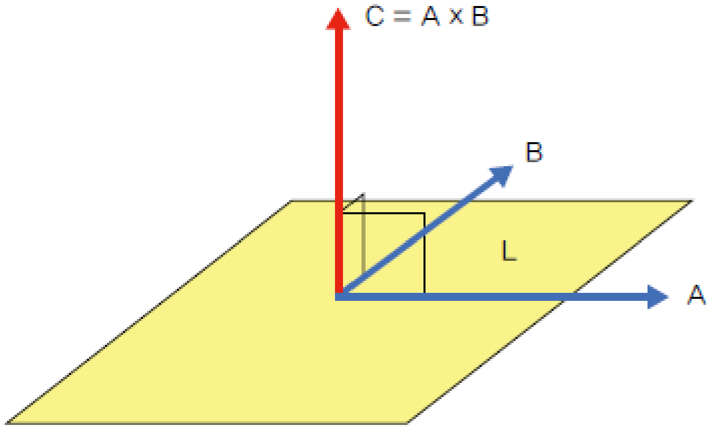
▶ 내적을 이용하면 각도 차이를 쉽게 파악할 수 있다

9.1 벡터 수학

- 9.1.9 벡터의 외적

  - 벡터의 외적은 두 벡터를 모두 수직으로 통과하는 벡터를 구하는 연산이며, 벡터 곱이나 교차 곱으로 부르기도 함.

  - 벡터 A를 벡터 B로 외적하는 표현은 A ×B임.



▶ A x B을 표현한 그림

9.2 유니티 C# 벡터

◦ 9.2.1 Vector 타입

- 유니티는 Vector2, Vector3, Vector4 타입을 지원함.

다음과 같은 형태로 생성자를 호출합니다.

- new Vector2(x, y);
- new Vector3(x, y, z);
- new Vector4(x, y, z, w);

```
Vector3 a = new Vector3(1, 2, 3); // (1, 2, 3) 벡터 생성

// (1, 2, 3)을 (10, 20, 30)으로 수정
a.x = 10;
a.y = 20;
a.z = 30;
```

9.2 유니티 C# 벡터

- 9.2.2 Vector3 연산

  - Vector3의 기본 연산은 모두 유니티의 C# 라이브러리에 정의되어 있음.

스칼라 곱

벡터에 배수를 취합니다.

- Vector3 * 스칼라;

예제 : (3, 6, 9) * 10 = (30, 60, 90)

```
Vector3 a = new Vector3(3, 6, 9);
a = a * 10; // a는 (30, 60, 90)이 됨
```

벡터의 덧셈과 뺄셈

두 벡터를 서로 더하거나 뺍니다.

- Vector3 + Vector3;

예제 : (2, 4, 8) + (3, 6, 9) = (5, 10, 17)

```
Vector3 a = new Vector3(2, 4, 8);
Vector3 b = new Vector3(3, 6, 9);

Vector3 c = a + b; // c는 (5, 10, 17)이 됨
```

- Vector3 - Vector3;

예제 : (2, 4, 8) - (3, 6, 9) = (-1, -2, -1)

```
Vector3 a = new Vector3(2, 4, 8);
Vector3 b = new Vector3(3, 6, 9);

Vector3 c = a - b; // c는 (-1, 2, -1)이 됨
```

벡터 정규화(방향벡터로 만들기)

해당 벡터와 방향은 같지만 크기가 1인 벡터를 생성합니다.

- Vector3.normalized;

예제 : (3, 3, 3)의 방향벡터인 (0.57..., 0.57..., 0.57...) 생성[3]

```
Vector3 a = new Vector3(3, 3, 3)
Vector3 b = a.normalized; // b는 대략 (0.6, 0.6, 0.6)이 됨
```

벡터의 크기

벡터의 크기(길이)를 구합니다.

- Vector3.magnitude;

예제 : (3, 3, 3)의 크기 5.19...

```
Vector3 a = new Vector3(3, 3, 3)
float b = a.magnitude; // b는 대략 5.19...가 됨
```

벡터의 내적

벡터 b를 벡터 a로 투영한 길이를 구합니다.

- Vector3.Dot(a, b);

예제 : (0, 1, 0) · (1, 0, 0) = 0

```
Vector3 a = new Vector3(0, 1, 0); // 위쪽으로 향하는 벡터
Vector3 b = new Vector3(1, 0, 0); // 오른쪽으로 향하는 벡터
float c = Vector3.Dot(a, b); // 수직인 벡터끼리 내적하면 결과는 0
```

9.2 유니티 C# 벡터

벡터의 외적

두 벡터 모두에 수직인 벡터를 구합니다.

- Vector3.Cross(a, b);

예제 : (1, 0, 0) x (0, 0, 1) = (0, 1, 0)

```
Vector3 a = new Vector3(0, 0, 1); // 앞쪽(Z) 방향벡터
Vector3 b = new Vector3(1, 0, 0); // 오른쪽(X) 방향벡터
// 외적 결과 c는 앞쪽과 오른쪽 모두에 수직인 위쪽(Y) 방향벡터
Vector3 c = Vector3.Cross(a, b); // c는 (0, 1, 0)
```

9.2 유니티 C# 벡터

- 9.2.3 Vector3 응용

  - 벡터 연산을 응용할 수 있는 대표적인 예제.

두 지점 사이의 거리

```
// 현재 위치(currentPos)에서 목적지(destPos)까지의 거리 구하기
Vector3 currentPos = new Vector3(1, 0, 1); // 현재 위치
Vector3 destPos = new Vector3(5, 3, 5);     // 목적지

// currentPos에서 destPos로 향하는 벡터
Vector3 delta = destPos - currentPos;

// currentPos에서 destPos까지의 거리(크기)
float distance = delta.magnitude;
```

```
Vector3 currentPos = new Vector3(1, 0, 1); // 현재 위치
Vector3 destPos = new Vector3(5, 3, 5);     // 목적지
// currentPos에서 destPos까지의 거리
float distance = Vector3.Distance(currentPos, destPos);
```

현재 위치에서 목적지로 향하는 방향

```
(destPos - currentPos).normalized;
```

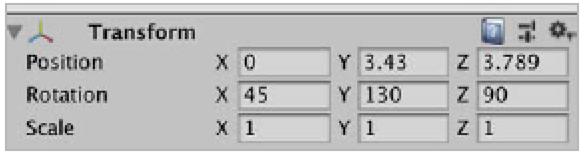
```
// 현재 위치(currentPos)에서 목적지(destPos)를 향해 10만큼 이동한 위치 구하기
Vector3 currentPos = new Vector3(1, 0, 1); // 현재 위치
Vector3 destPos = new Vector3(5, 3, 5);     // 목적지

// currentPos에서 destPos으로 향하는 방향벡터
Vector3 direction = (destPos - currentPos).normalized;

// 목적지를 향해 10만큼 현재 위치에서 이동한 새로운 위치
Vector3 newPos = currentPos + direction * 10;
```

9.3 쿼터니언

- 쿼터니언Quaternion은 회전을 나타내는 타입임.



▶ 트랜스폼의 위치, 회전, 스케일

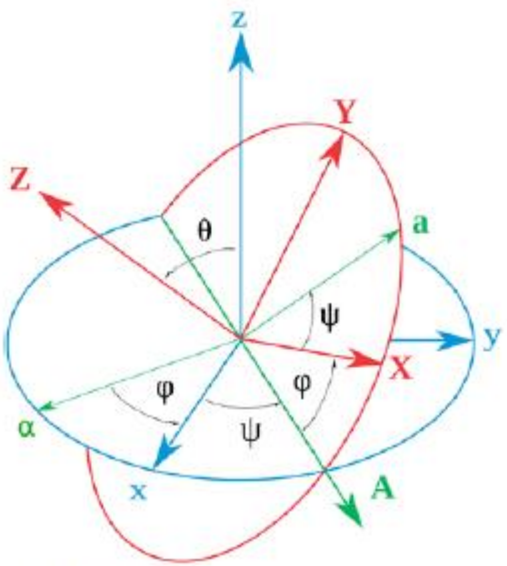
```
transform.position = new Vector3(0, 0, 10);
transform.localScale = new Vector3(1, 1, 1);

// rotation은 Vector3 타입이 아닌 Quaternion 타입이므로 에러 발생
transform.rotation = new Vector3(30, 60, 90);
```

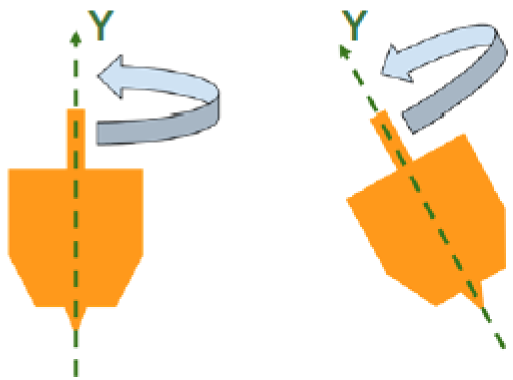
9.3 쿼터니언

◦ 9.3.1 짐벌락(Gimbal Lock)

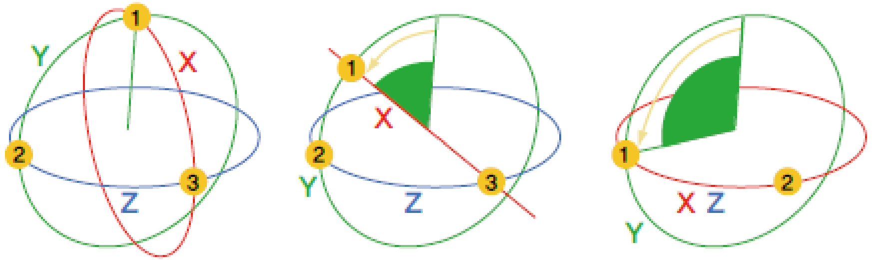
- 3D 벡터를 사용해 3D 회전을 나타내는 표현을 '오일러각'이라고 함.
- 어떤 축의 회전이 다른 축의 회전에 영향을 미친다는 사실과 세 번 나누어 축을 회전하는 방식때문에 오일러각 체계에서는 특정한 경우 앞선 두 번의 회전에 의해 세 번째 회전의 자유도가 상실되어 세 축 중 한 축의 회전을 사용없게 되는 현상이 발생함. 이것을 짐벌락이라 부름.



▶ 오일러각의 표현



▶ 서 있는 팽이의 회전과 기울어진 팽이의 회전



▶ 짐벌락의 예

9.3 쿼터니언

- 9.3.2 쿼터니언

  - 쿼터니언은 원소로 x, y, z 외에도 w를 가지는 값으로, 사원수라고 부르기도 함.

  - 쿼터니언은 ‘한 번에 회전하는’ 방식이기 때문에 오일러각과 달리 짐벌락 현상이 없으며 90도 회전을 제대로 표현할 수 있음.

  - 게임에서 회전을 구현할 때는 쿼터니언을 사용함.

9.3 쿼터니언

- 9.3.3 쿼터니언 예제
- - 유니티가 제공하는 쿼터니언 관련 편의 메서드를 사용해 회전값을 만들고 사용하는 방법 중 일부임.

새로운 회전 데이터 생성

오일러각을 표현하는 Vector3 값에서 새로운 Quaternion 값을 생성할 수 있습니다.

```
Quaternion.Euler(Vector3);
```

(0, 60, 0) 회전을 표현하는 쿼터니언 회전 데이터를 생성하는 코드는 다음과 같습니다.

```
Quaternion rotation = Quaternion.Euler(new Vector3(0, 60, 0));
```

회전을 Vector3(오일러각)로 가져오기

Quaternion 타입은 저장된 회전값을 Vector3 타입의 오일러각으로 변환한 변수 eulerAngles 를 제공합니다.

```
Quaternion rotation = Quaternion.Euler(new Vector3(0, 60, 0));
// Vector3 타입의 값으로 (0, 60, 0)이 나옵니다.
Vector3 eulerRotation = rotation.eulerAngles;
```

현재 회전에서 '더' 회전하기

```
Quaternion a = Quaternion.Euler(30, 0, 0);
Quaternion b = Quaternion.Euler(0, 60, 0);

// a만큼 회전한 상태에서 b만큼 더 회전한 회전값을 표현
Quaternion rotation = a * b;
```

9.4 마치며

- 이 장에서 배운 내용 요약

- 벡터는 방향과 크기를 가짐.
- 벡터는 절대적인 위치로 표현하거나 상대적인 위치로 표현할 수 있음.
- 벡터가 표현하는 화살표의 길이가 벡터의 크기임.
- 벡터는 스칼라 곱을 사용해 배수를 취할 수 있음.

  예 : (1, 1) × 5 = (5, 5)
- 방향벡터는 크기가 1인 벡터로, 방향을 표현함.
- 어떤 벡터를 크기가 1인 방향벡터로 만드는 것을 정규화라고 함.
- 벡터끼리의 덧셈이 가능함.

  예 : (1, 1) + (2, 2) = (3, 3)
- 벡터끼리의 뺄셈이 가능함.

  예: (1, 1) - (2, 2) = (-1, -1)

- B - A는 A에서 B로 향하는 벡터임.
- 벡터 A에 벡터 B를 내적한(A·B) 결과는 두 벡터 사이의 각도로 결정됨.
- 벡터 A에 벡터 B를 외적한(A ×B) 결과는 A와 B 모두에 수직인 벡터임.
- 노말벡터는 어떤 평면에 수직이며, 해당 평면의 방향을 표현하는 벡터임.
- 새로운 벡터값은 new Vector3(x, y, z);로 생성함.
- 어떤 벡터의 정규화된 벡터(방향벡터)는 Vector3.normalized;로 구함.
- 어떤 벡터의 크기는 Vector3.magnitude;로 구함.
- 트랜스폼 컴포넌트의 회전값은 Vector3 타입이 아니라 Quaternion 타입임.
- Vector3를 사용하는 오일러각은 짐벌락이 생길 수 있으므로 유니티는 회전을 Quaternion 타입으로 표현함.
- Quaternion.Euler( ) 메서드로 Vector3 값에서 쿼터니언 회전값을 생성할 수 있음
- 유니티는 쿼터니언에 대한 직접 접근을 막습니다. 대신 쿼터니언과 관련된 편의 메서드를 제공함.