Vector

이 장에서는 벡터에 대한 개념을 이해합니다. 먼저 필요한 용어를 정의하도록 하겠습니다.

1 용어 정의

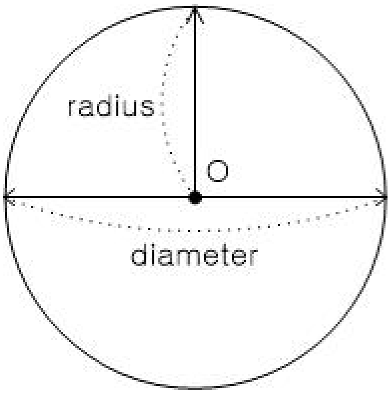
선에 대해서 양끝점이 정해지지 않은 선을 직선(line)이라고 합니다. 직선의 한쪽 끝점이 정해지면 레이(광선, 반직선, ray)라고 하며, 양 끝점이 모두 정해지면 라인 조각(line segment)이라고 합니다.

[그림] 라인 세그먼트, 레이 그리고 라인

라인 세그먼트의 경우 양 끝점을 begin, end로 표시할 수 있으며, 레이의 경우 시작하는 끝점을 테일(tail)이라고 하고, 광선이 진행하는 방향의 끝점을 헤드(head)라고 합니다.

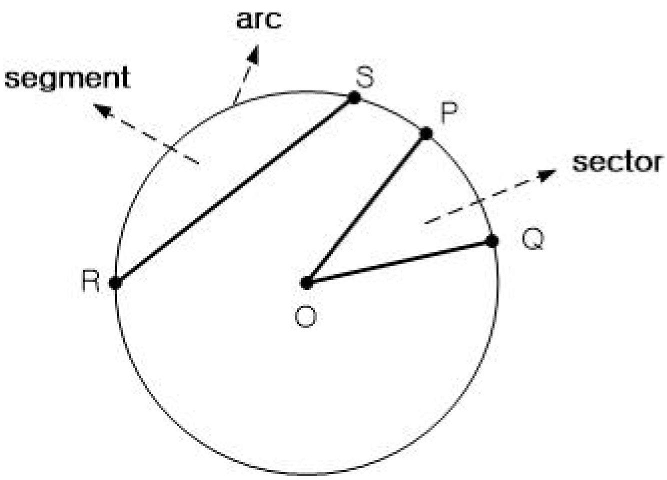
[그림] 레이의 테일과 헤드: 레이의 시작점을 테일(tail)이라고 하며, 방향이 진행하는 쪽을 헤드(head)라고 합니다.

이제 원(circle)과 관련된 용어를 정의하도록 하겠습니다.



[그림] 원의 지름과 반지름

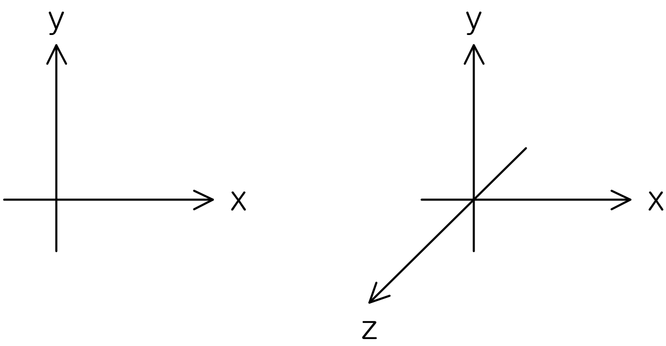
원의 중심 O를 지나는 직선이 원의 두 점과 만나서 이루는 라인 세그먼트의 길이를 지름(diameter)이라고 합니다. 지름의 절반 길이를 반지름(radius)이라고 합니다.



[그림] 섹터, 세그먼트와 원호(아크, arc)

원의 지름에서 시작하는 두 개의 반직선이 원과 만나는 점을 각각 P와 Q라고 하면, O,P,Q가 이루는 원의 일부를 **섹터(sector)**라고 합니다.
임의 직선이 원을 가로지를 때, 만나는 두 점을 R과 S라고 하면, R과 S가 잘라내는 원의 일부를 **세그먼트(segment)**라고 합니다.
그리고 세그먼트 혹은 섹터를 이루는 원의 선 일부를 **원호(아크, arc)**라고 합니다.

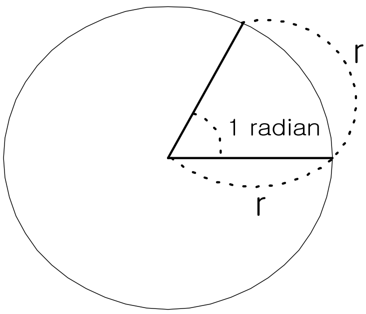
우리는 2차원과 3차원에 대해서 다음과 같은 축을 사용한다고 가정합니다.



[그림] 2차원과 3차원 축: 위의 3차원 축은 일반적으로 많이 사용되는 오른손 좌표계(right-hand coordinate system)입니다. OpenGL이나 3D Max 프로그램 등 대부분은 오른손 좌표계를 사용하지만, DirectX는 왼손 좌표계를 사용합니다. 오른손 좌표계를 사용하는 시스템에서 벡터는 대부분 열 벡터(column vector)로 표현합니다. 그러므로, 벡터는 변환 매트릭스(transform matrix)의 오른쪽에 위치합니다. DirectX는 행 벡터를 사용하므로, 변환 매트릭스의 왼쪽에 위치하는데 행벡터나 열벡터가 특정한 좌표계와 연관된 것은 아닙니다.

2 기본 개념

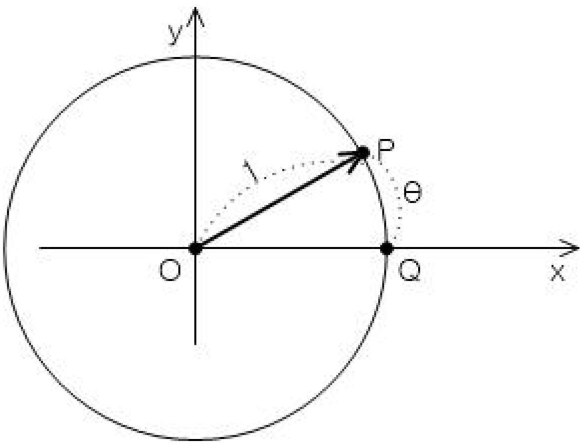
반지름이 실수 r인 원에서, 원주(circumference)의 길이가 r인 원호(arc)가 원의 중심(center)과 이루는 비율(ratio)은 원의 크기에 상관없이 일정합니다. 이 비율을 **라디안(radian)**이라고 합니다. 수학자들은 오래전에 단위원(unit circle)의 반호의 길이(원 둘레의 반)를 측정하려는 시도를 했는데, (놀랍게도) 이 값은 순환하지 않는 무한 소수(infinite decimal)였으며, 그 값을 정확하게 나타낼 수 없으므로 기호 **파이(pi, π)**로 나타냅니다. 모든 <u>수학 함수들은 각도(degree)를 표현하기 위해서 라디안을 단위로 사용</u>하며, 파이의 소수점 이하 6자리까지의 값은 3.141592 입니다.



[그림] 라디안: 반지름 r인 원에서, 원호의 길이가 r일 때 원의 중심과 원호가 이루는 비율을 1 라디안이라고 합니다. 반원의 길이는 pi로 정의하며, 근사값은 3.141592입니다.

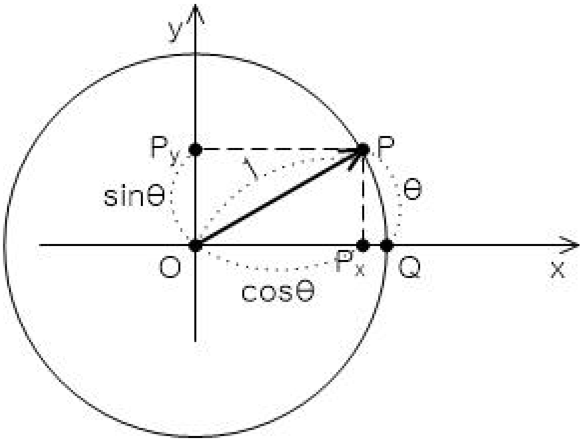
그러므로 라디안과 각도와의 관계는 다음과 같습니다.

pi 라디안(radian) = 180도(degree)



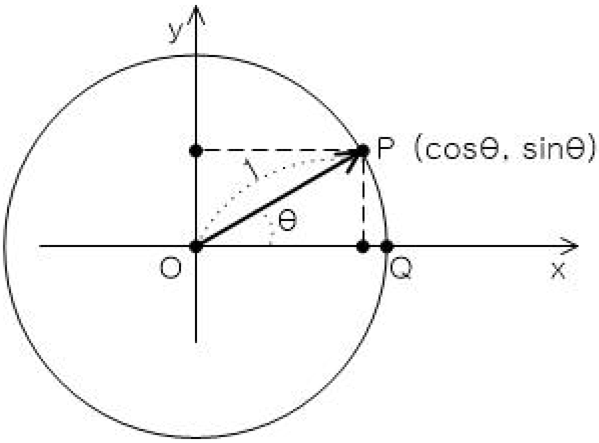
[그림] 단위 원(unit circle): 원호 PQ의 길이를 θ라고 합시다.

반지름의 길이가 1인 단위 원(unit circle)을 생각해 봅시다. 원점 O에서 출발한 반직선이 원과 만나는 점을 P, 원과 x축이 만나는 점을 Q라고 하고, 원호 PQ의 길이를 θ라고 합시다. 이제 단위원에서 원호의 길이 θ가 주어졌을 때, 점 P에서 x축에 투영한 점 P_x의 길이를 구하는 함수를 정의할 수 있습니다.



[그림] 단위원에서 원호의 길이에 대한 x축과 y축의 투영된 길이를 구하는 함수

단위 원에서 원호 θ의 길이가 주어졌을 때, 직선 OP_x의 길이를 구하는 함수를 코싸인(cosine)함수라고 합니다. 그리고 직선 OP_y의 길이를 구하는 함수를 싸인(sine)이라고 합니다.



[그림] 점 P의 좌표: 이제 점 P의 좌표는 (cosθ, sinθ)입니다.

원호의 길이 θ가 단위원의 유일한 각을 나타내는데 사용할 수 있습니다. 코싸인 함수를 cos(), 싸인 함수를 sin()이라고 정의하면 점 P의 좌표는 (cos(θ), sin(θ)) 입니다. 그리스 문자 θ가 파라미터로 사용된 것이 명확할 때 괄호를 생략하고 (cosθ, sinθ)라고 적습니다.

이 개념은 단위원이 아니라, 빗변의 길이가 1인 직각 삼각형(right triangle)

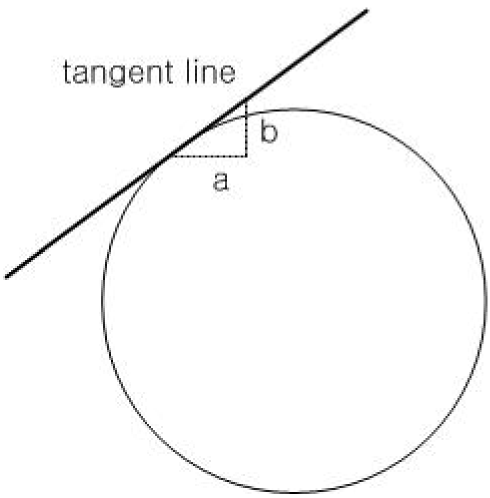
PQO를 생각하면, 빗변(hypotenuse)의 길이에 대한 밑변(adjacent)과 마주보는 변(opposite)의 길이에 대한 함수가 됩니다.

이제 단위 원이 아니라 임의의 직각 삼각형(right-angled triangle), ACB에 대해서 함수를 정의할 수 있습니다.

직각 삼각형 ACB에서 각 꼭지점의 각도가 같다면 삼각형의 크기와 상관없이 변들의 길이의 비율은 일정합니다. 이 비율을 이용해서 삼각형의 각 꼭지점의 각을 표현할 수 있는데, 사인(sine), 코사인(cosine) 및 탄젠트(tangent) 함수를 사용하여 나타낼 수 있습니다.

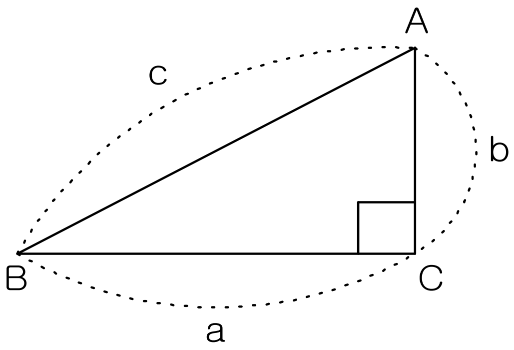
직각 삼각형의 밑변(adjacent)의 길이가 a, 빗변(hypotenuse)의 길이가 c, 높이(opposite)를 b라 하고 꼭지점 B에서의 각을 t라고 합시다. 그러면 b/c의 비율을 싸인(sine) t, a/c의 비율을 코사인(cosine) t, b/a의 비율을 탄젠트(tangent) t로 표현합니다.

sine은 인체 기관(organ)이나 조직(tissue)의 움푹한 곳을 가리키는 sinus에서 파생한 단어입니다. cosine은 sine이라는 단어에 함께를 의미하는 co-를 붙여서 만든 단어입니다. b/a를 탄젠트라고 하는 이유는 이 비율이 접선(tangent)의 기울기를 나타내는데 주로 사용되기 때문입니다.



[그림] 접선(tangent line)의 기울기는 b/a로 나타낼 수 있습니다.

그러므로 삼각형의 변의 길이를 안다면, 각도를 구할 수 있고, 반대로 각도와 몇 변에 대한 길이를 안다면 나머지 변의 길이를 구할 수 있습니다.



[그림] 삼각함수: 직각 삼각형의 각 변의 길이의 비는 삼각형의 크기와 상관없이 일정합니다. 꼭지점의 각도를 구하기 위해서, 혹은 변의 길이를 구하기 위해서 이 비율을 적절히 이용합니다.

피타고라스의 정리에 의하면 직각 삼각형 ACB에 대해서 다음의 식이 만족됩니다.

$$c^2 = a^2 + b^2,\ c = \sqrt{a^2 + b^2}$$

그러므로 c가 1이라면 아래의 식을 만족합니다.

$$1^2 = \sin^2\theta + \cos^2\theta, 1 = \sin^2\theta + \cos^2\theta$$

각(each) t값에 대한 cosine값을 보면 (대부분의 경우) 순환하지 않는 무한소수로써, 정확한 값을 구할 수 없습니다. 그래서 근사값을 사용하는데, 이러한 근사값은 **테일러 급수(Tayler series)**등의 방법을 통해 구할 수 있습니다.

이러한 비율을 구하는 함수를 **삼각함수(trigonometrix function)**라고 하는데, C 표준 라이브러리는 sine, cosine을 구하는 표준 삼각함수를 제공합니다. 그것은 아래와 같습니다.

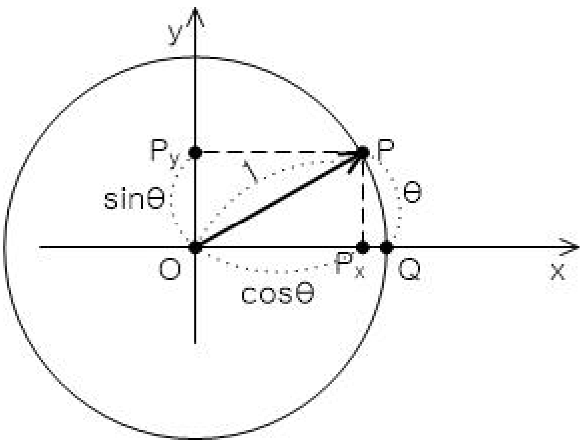
sin(), cos()

위 함수들은 t를 입력으로 받아 b/c와 a/c의 근사값을 리턴하는데, 반대로 b/c, a/c를 입력으로 받아 t를 구하기 위해서는 각 함수의 역함수에 해당하는 아래의 함수들을 사용할 수 있습니다.

asin(), acos()

□ 일반적인 역함수의 표현은 $\sin^{-1}()$, $\cos^{-1}()$이지만, 삼각함수는 이 표현을 사용하지 않고, 아크사인(arc sine), 아크코사인(arc cosine)을 사용합니다.

예를 들어 sin(3.141592)의 근사값은 0이며, asin(0)의 근사값은 3.141592입니다.



[그림] cos()의 역함수의 정의

cos()함수의 정의는 단위원에서 아크의 길이 θ를 주면, x축에 투영된 직선 조각 OP_x의 길이를 구하는 것입니다. 반대로 직선 조각 OP_x의 길이를 주면, 아크의 길이 θ를 구하는 함수는 cos()의 역함수(inverse function)입니다. 일반적으로 역함수는 $\cos^{-1}()$처럼 나타냅니다. 하지만 이 경우, 역함수가 아크의 길이를 리턴하므로 cos()함수의 역함수를 아크 코싸인(arc cosine)이라고 정의합니다. 표준 구현된 함수의 이름은 acos()입니다. 그러므로 아래의 식이 성립합니다.

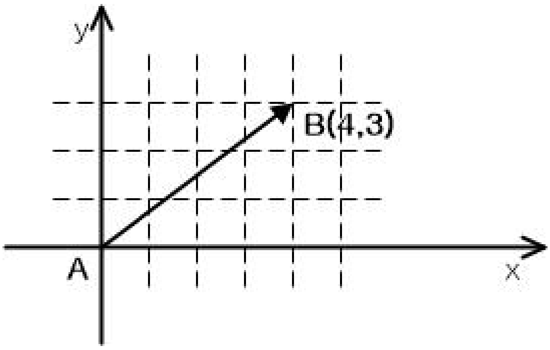
$$acos(\cos\theta) = \theta$$

3 벡터

실수 혹은 정수와는 다르게 **벡터(vector)**는 양(magnitude)과 방향(direction)으로 정의합니다. 양만을 가지는 값을 **스칼라(스케일러, scalar)**라고 합니다. 예를 들어 자동차가 속력(speed) 100km/h로 달리고 있을 때 속력은 스칼라 값입니다. 속력은 방향을 고려하지 않습니다. 자동차가 부산에서 대구 방향으로 속도(velocity) 100km/h로 달린다면 이는 벡터입니다. 속도의 양은 100이며 방향은

부산에서 대구입니다.

시작점(initial point) A와 끝점(terminal point) B로 표현되는 벡터 v는 $v = \overrightarrow{AB}$로 나타냅니다.



[그림] 벡터의 표현: 벡터 $v = \overrightarrow{AB}$는 (4,3) 혹은 $\begin{bmatrix}4\\3\end{bmatrix}$이라고 나타냅니다.

벡터 $v = \overrightarrow{AB}$는 행벡터(row vector) (4,3) 혹은 열벡터(column vector) $\begin{bmatrix}4\\3\end{bmatrix}$이라고 나타냅니다. 일반적인 컴퓨터 그래픽스 책에서는 $\begin{bmatrix}4\\3\end{bmatrix}$을 선호하는데, 왜냐하면 (4, 3)이라는 표현은 점(point)을 표현하는 것인지, 벡터를 표현하는 것인지 모호함이 발생하기 때문입니다. 위 그림에서 $v = \overrightarrow{AB}$는 벡터이지만, x축을 나타내는 화살표와 y축을 나타내는 화살표는 벡터가 아닙니다. x축과 y축을 벡터로 간주하는 것은 다음 장에서 다룰 예정입니다.

벡터를 열벡터로 표현하면, 후에 행렬(Matrix)을 다룰 때, 변환 행렬은 벡터의 왼쪽에 위치해야 합니다. 벡터를 입력으로 받는 f()함수와 g() 함수가 있다고 생각해 봅시다. 그러면 f($\begin{bmatrix}4\\3\end{bmatrix}$)의 결과를 g()에 적용하는 식은 다음과 같이 적을 수 있습니다.

$$g(\ f(\begin{bmatrix}4\\3\end{bmatrix})\ )$$

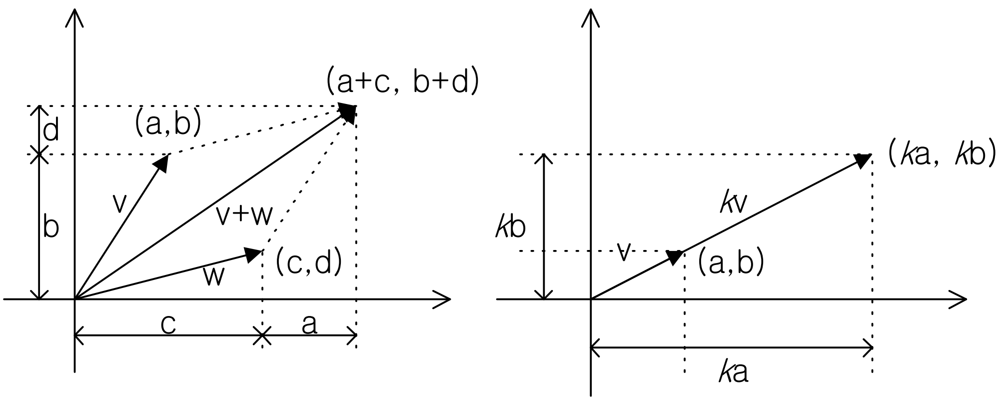
위 식은 벡터 $\begin{bmatrix}4\\3\end{bmatrix}$에 대해서 먼저 f()함수를 적용하고, 그 결과에 대해서 g()함수를 적용한 것입니다. 괄호를 생략하고 표현해 보면 다음과 같습니다.

$$g\ f\ \begin{bmatrix}4\\3\end{bmatrix}$$

이처럼 벡터를 열벡터로 표현하면, 점(point)과의 모호함을 없앨 수 있고, 합성 변환이 수학의 표현과 잘 어울린다는 이점이 있습니다.

이 책에서는 (4, 3)과 $\begin{bmatrix}4\\3\end{bmatrix}$를 혼용해서 사용할 것입니다. 그리고 점과의 구분이 명확이 필요한 경우 (4, 3)이 아니라, 벡터 (4, 3)이라고 표현할 것입니다. 그렇게 하는 이유는 벡터 (4, 3)은 소스 코드에서 $\begin{bmatrix}4\\3\end{bmatrix}$으로 표현하기 어렵기 때문입니다.

벡터에는 여러가지 연산이 있지만, 가장 간단한 연산은 덧셈과 스칼라 곱셈입니다.



[그림] 벡터의 합: 벡터의 합은 각 요소(component)를 합한 것입니다. 벡터의 스칼라 k곱은 각 요소에 k를 곱한 것입니다.

벡터 $\vec{v}=(a,b)$와 $\vec{w}=(c,d)$가 주어졌을 때, $\vec{v}+\vec{w}$의 의미는 $\vec{v}$만큼 이동 한 후에, $\vec{v}$의 head에서 $\vec{w}$만큼 이동하라는 의미입니다. 이것은 위 그림에서 보듯이 다음과 같이 나타낼 수 있습니다.

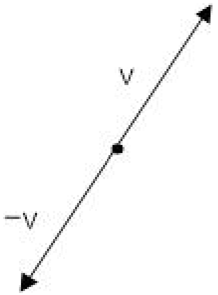
$$\vec{v}+\vec{w}=(a,b)+(c,d)=(a+c,b+d)$$

벡터 $\vec{v}=(a,b)$와 임의의 실수값 k에 대해서 곱을 정의할 수 있습니다. 그것은

$\vec{v}$방향은 유지한 채로 크기를 k만큼 증가시켰다는 의미로 다음과 같이 정의합니다.

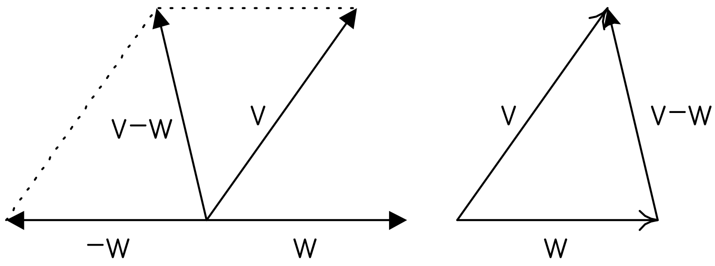
$$k\vec{v} = k(a,b) = (ka,kb)$$

이제 음 벡터를 이해할 수 있습니다. 음 벡터는 크기는 유지한 채로 방향이 반대가 되는 벡터입니다.



[그림] 음(negative)벡터

$v=(a,b)$ 일 때 $-v=(-a,-b)$입니다.



[그림] 벡터의 차(subtraction): 벡터의 차 v-w는 v에 -w를 더한 것과 같습니다

벡터 합의 정의에 의해서, $v-w=(a-c,b-d)$입니다.

벡터 v = (a,b)의 길이(length, norm)는 |v|로 나타내며, 다음과 같습니다. 아래의 식은 피타고라스의 정리(Pythagorean theorem)에 의해서 유도 가능합니다.

$$|v| = \sqrt{a^2+b^2}$$

벡터의 방향 성분만을 고려하기 위해서 |v|가 1이 되게 v를 바꿀 수 있는데, 이러한 과정을 **정규화(노멀라이즈, normalize)**라고 합니다.

4 KVector2 벡터 클래스의 구현

이제 2차원 벡터 클래스를 아래와 같이 구현 할 수 있습니다.

```
class KVector2
{
public:
    float   x;
    float   y;

public:
    KVector2(float tx, float ty) { x = tx; y = ty; }
    KVector2(int tx, int ty) { x = (float)tx; y = (float)ty; }
};

inline KVector2 operator+(const KVector2& lhs, const KVector2& rhs)
{
    KVector2 temp(lhs.x + rhs.x, lhs.y + rhs.y);
    return temp;
}

inline KVector2 operator*(float scalar, const KVector2& rhs)
{
    KVector2 temp(scalar*rhs.x, scalar*rhs.y);
    return temp;
}

inline KVector2 operator*(const KVector2& lhs, float scalar)
{
    KVector2 temp(scalar*lhs.x, scalar*lhs.y);
    return temp;
}
```

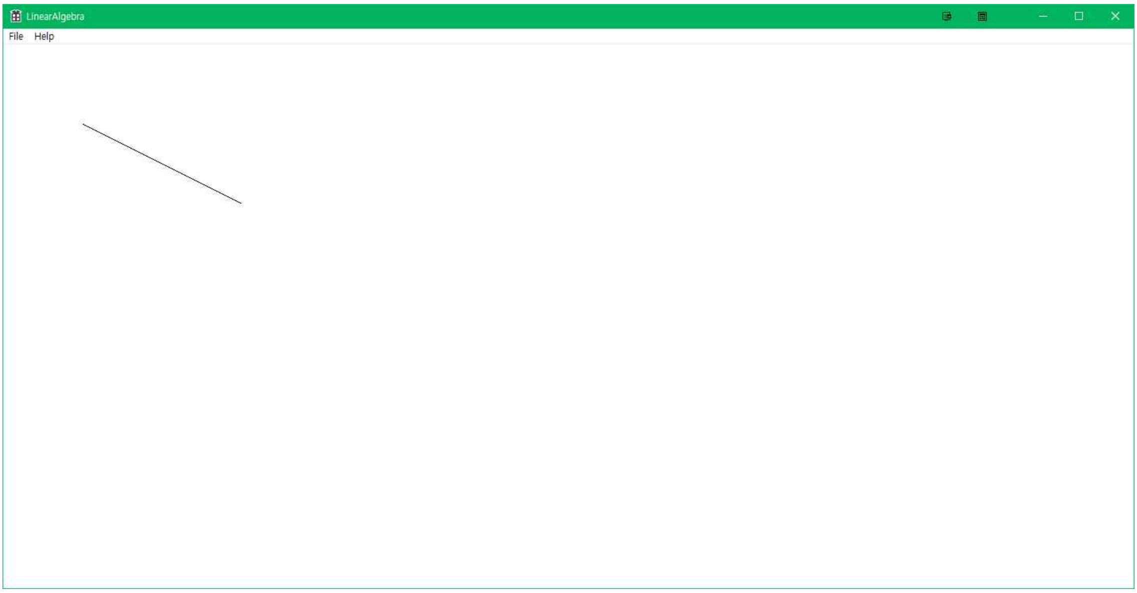
벡터와 스칼라 곱에 대해서 스칼라 값이 벡터의 왼쪽 혹은 오른쪽에 있을 수

있으므로, operator*()함수를 두 개 정의했습니다.

이제 두 벡터를 입력으로 받아, 두 벡터 사이의 head를 연결하는 선을 그리는 함수를 다음과 같이 정의할 수 있습니다.

```
void KVectorUtil::DrawLine(HDC hdc, const KVector2& v0, const KVector2&
v1)
{
    MoveToEx(hdc, (int)v0.x, (int)v0.y, nullptr);
    LineTo(hdc, (int)v1.x, (int)v1.y);
}
```

우리는 윈도우의 스크린 좌표계를 그대로 사용합니다. 스크린 좌표계는 클라이언트 영역의 좌측 상단이 원점이므로 실행 결과는 다음과 같습니다.



[그림] LinearAlgebra_Step02 Vectors 프로젝트의 실행 결과

LinearAlgebra_Step02 Vectors 프로젝트를 빌드해서 실행한 후 결과를 확인해 보기 바랍니다.

실습문제

1. 피타고라스의 정리가 성립함을 증명하세요.

2. 벡터의 연산에는 어떠한 것이 있는지 열거하고, 설명하세요.

3. KVector2에 Normalize()함수를 추가하세요.

@