cub3D 가이드

저자: minckim

목차

기호 표기

벡터 :굵은 알파벳 소문자 ex) $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{vector}$

행렬 :굵은 알파벳 대문자 ex) $\mathbf{A}, \mathbf{B}, \mathbf{C}, \boldsymbol{Metrix}$

스칼라 :일반 알파벳 소문자 ex) $a, b, c, scalar$

수식을 보다 보면 이게 벡터인지, 행렬인지, 스칼라인지 혼란스럽습니다.

한눈에 구분할 수 있도록 굵기나 대문자 여부로 구분할 수 있게 하고 있습니다.

표현 방법은 책마다 사람마다 다릅니다.

어떤 책에서는 벡터를 표현할 때 벡터 위에 화살표를 씌우기도 합니다.

수학적인 배경지식

과제를 수행하는데에 필요한 기초 수학

42에는 사원수를 통한 화면회전을 하는 분이 있는가 하면, 벡터가 생소한 분들도 있습니다.
수학 실력 스펙트럼이 다양한데, 벡터가 생소한 분들이라도 쉽게 이해할 수 있도록 강의 슬라이드를 만들어 보았습니다.
특히 수학파트는 슬라이드를 보면서 이해하기 쉽게 아래에 자막 비슷한 것을 달아보았습니다.

참고로 minckim은 건축학과 출신입니다.
[비전공자], [6두품], [사파], [유사이과]임에 주의하십시오.
(용어나 내용의 엄밀함이 다소 부족할 수 있다는 뜻입니다)

뒷부분은 녹화 영상을 참고하시면 좋을 것 같습니다.
(자막 쓸 공간이 없어요ㅠㅠ)

여러분…

벡터와 행렬은

여러분을

괴롭히기 위해 존재하는 것이

아닙니다

수학적인 배경지식::벡터

벡터

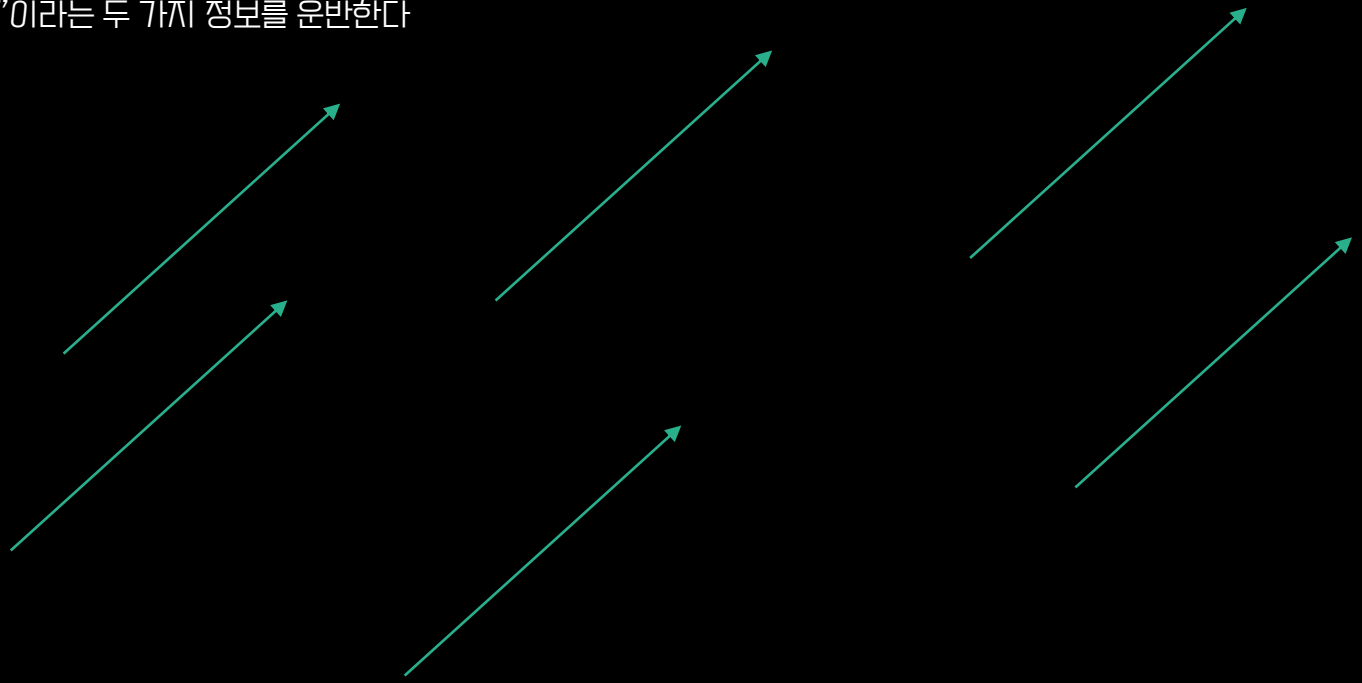
벡터 == 길이와 방향을 가진 값

벡터(vector)의 어원

라틴어, 운반한다

➔ 점을 주어진 방향으로 길이만큼 운반한다

➔ "길이"와 "방향"이라는 두 가지 정보를 운반한다



수학적인 배경지식::벡터

벡터

스칼라 scalar

숫자 하나로 충분한 것들:

- 온도
- 속력(단순한 빠르기)
- 몸무게(~~tmi 무게는 힘이라서 방향이 있 읍읍~~)

벡터 vector

숫자 하나만으로는 설명하기 애매한 것들:

- 속도(방향이 포함된 빠르기)
- 힘
- 위치

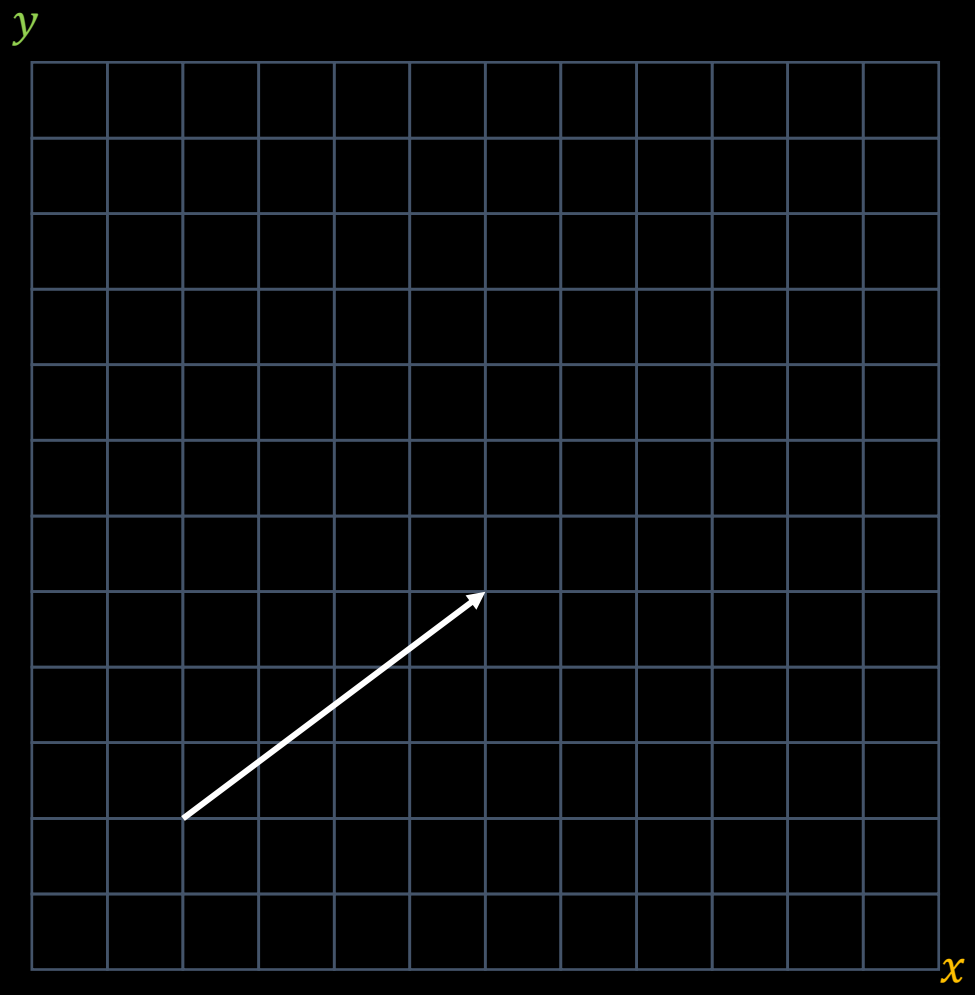
텐서 tensor

숫자 하나만으로는 설명하기 애매한 것들 중 매운맛:

- 응력

수학적인 배경지식::벡터

벡터 표현방법



좌표 $(3,4)$

길이와 각도 $(5, 37°)$

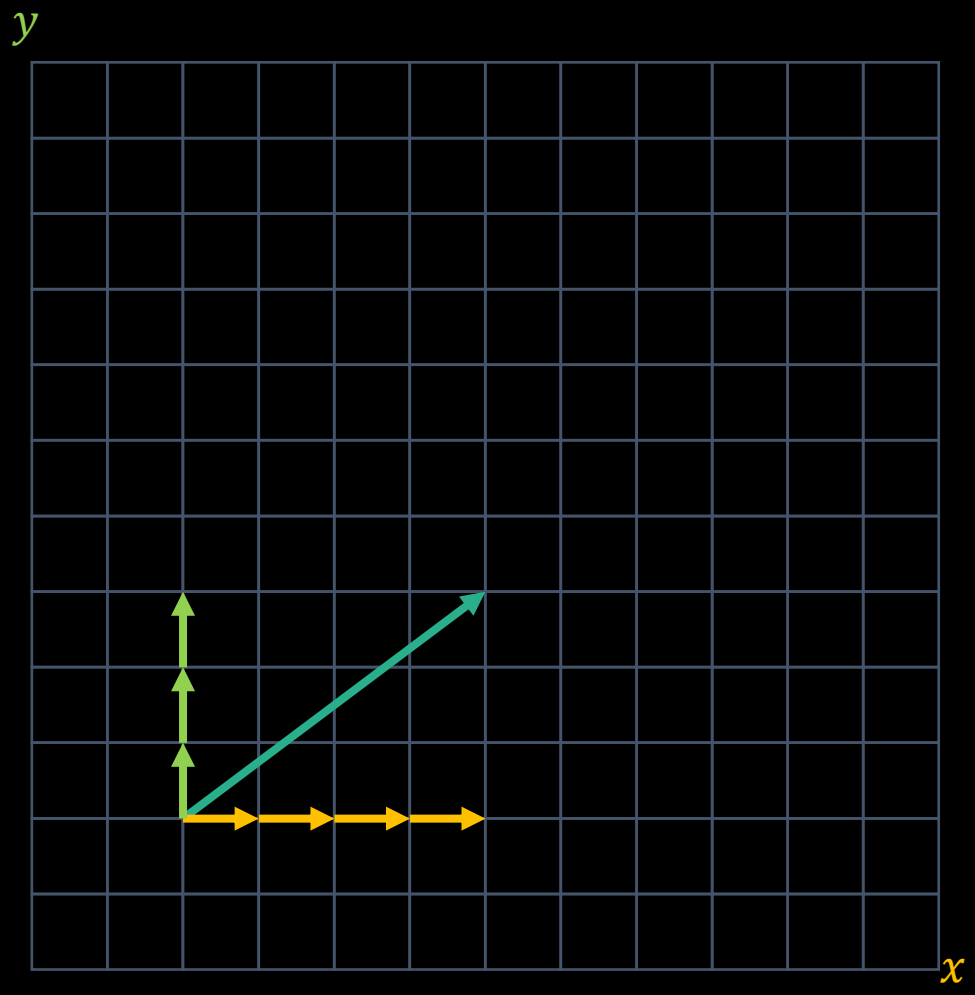
벡터를 정량적으로 표현하는 방법에는 여러가지가 있습니다.

그 중 가장 많이 쓰이는 방법은 벡터를 직교좌표계로 표현하는 것입니다.

그리고 경우에 따라서는 길이와 방향으로 표현할 수도 있습니다. 각자 특정 계산에 대해 장단점이 있습니다.

수학적인 배경지식::벡터

벡터 표현방법



$$(4,3) = (4,0) + (0,3)$$

$$(4,3) = 4 \cdot (1,0) + 3 \cdot (0,1)$$

$$(4,3) = 4\hat{x} + 3\hat{y}$$

길이와 방향으로 표현하는 것을 극좌표계라고 하는데, 다루기 어렵기 때문에 먼저 직교좌표계부터 살펴보겠습니다.

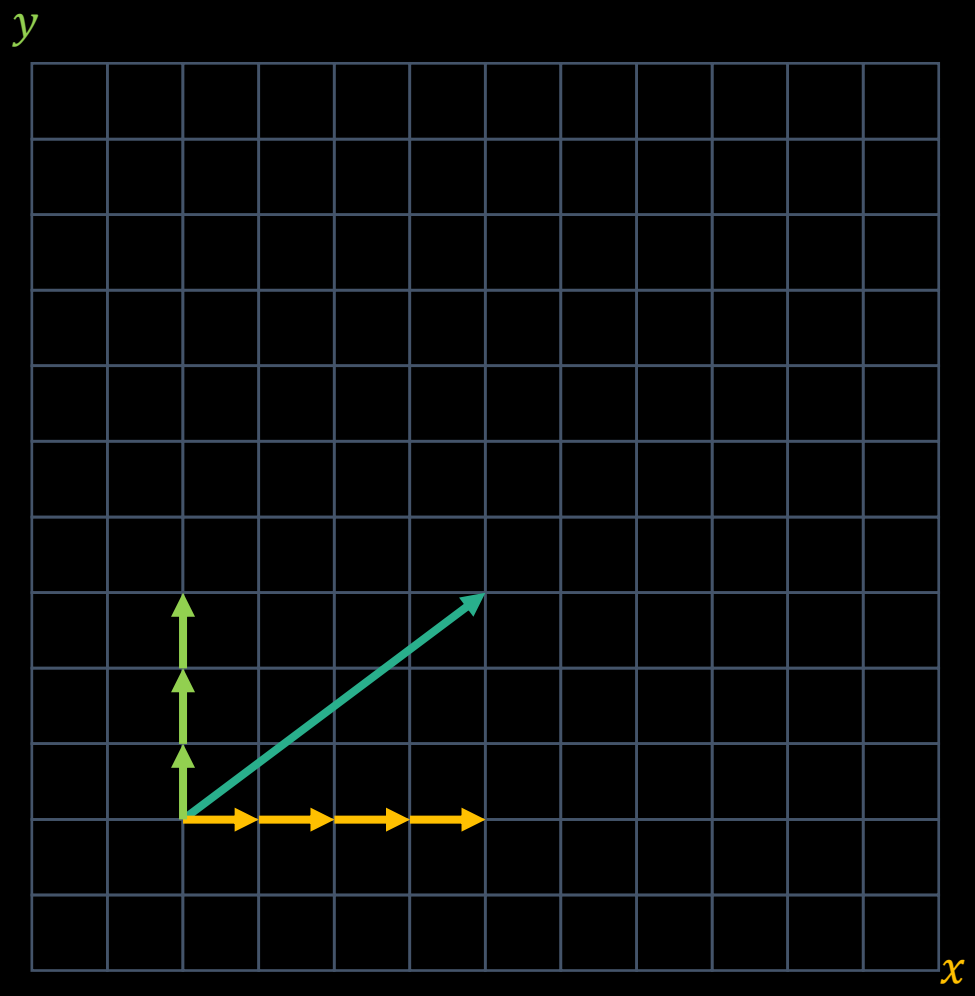
직교좌표계는 직교하는 축들을 기준으로 표현하는 것입니다.

간단하게 (4,3)은 x축방향으로 4만큼, y축방향으로 3만큼 떨어진 곳을 표현한 것입니다.

눈치채셨나요? 벡터가 가진 정보는 길이와 방향 뿐이라서, 벡터를 표현하는 것은 점을 표현하는 것과 다를 바 없습니다.

수학적인 배경지식::벡터

벡터 구조체




```c
typedef struct      s_vec{
    double          x;
    double          y;
}                   t_vec;

t_vec   vec_new(double x, double y)
{
    t_vec   result;

    result.x = x;
    result.y = y;
    return (result);
}
```

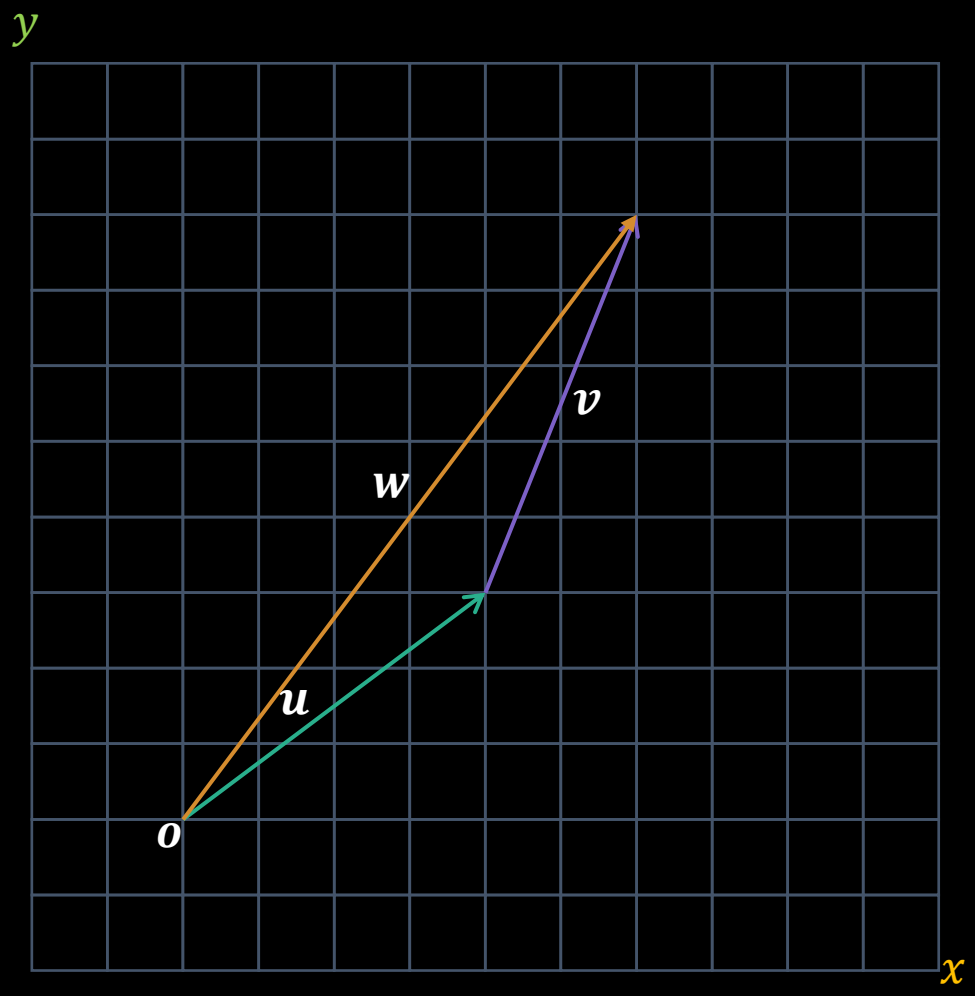
벡터 구조체를 만들어봅시다. 일단 만들어 두면 쓸모가 많습니다.

숫자를 일일이 더하고, 빼고, 곱하는 일을 벡터 연산으로 할 수 있게 됩니다.

숫자 두 개를 받아서 벡터 구조체를 반환하는 함수를 만들어 봅시다.

수학적인 배경지식::벡터

벡터 덧셈

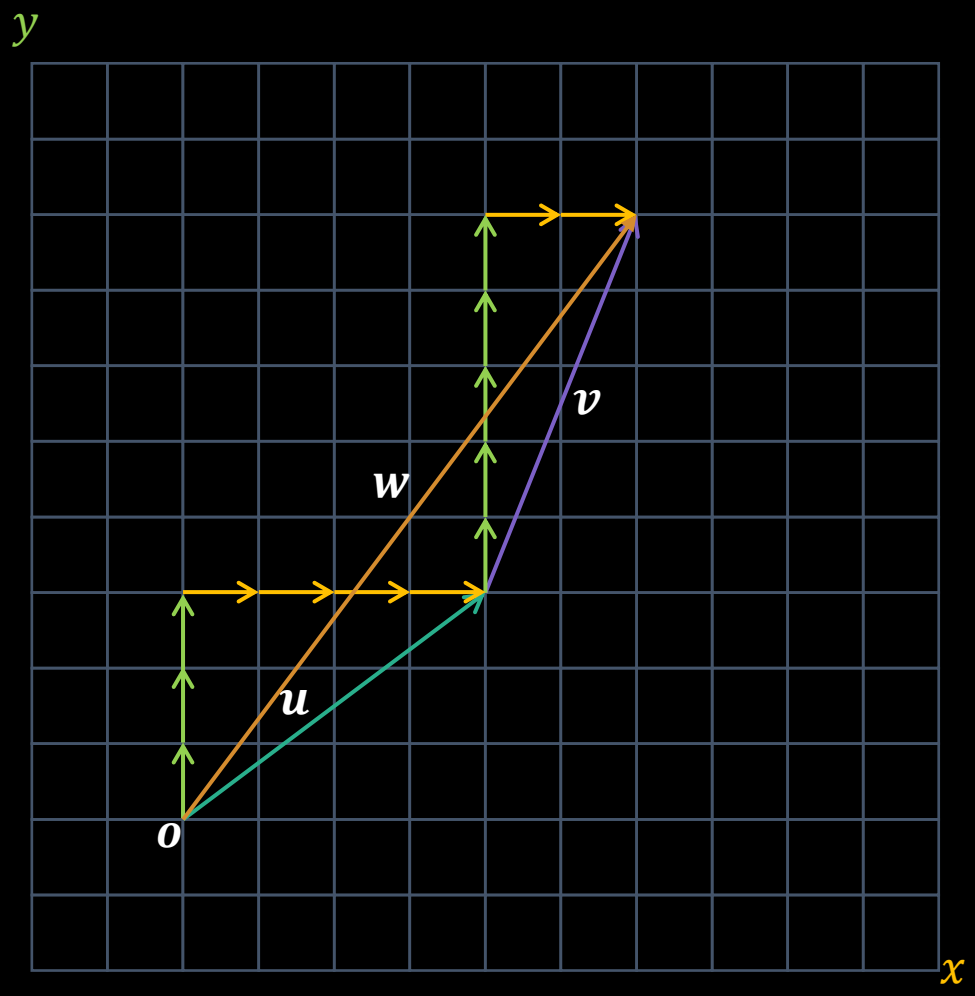


1. 플레이어는 $o(0,0)$에 있다.
2. 플레이어를 $u(4,3)$만큼 움직였다.
3. 플레이어를 $v(2,5)$만큼 움직였다.
4. 최종 위치는?

벡터 덧셈이 필요한 경우를 생각해 봅시다.

수학적인 배경지식::벡터

벡터 덧셈



1. 플레이어는 $\boldsymbol{o}(0,0)$에 있다.
2. 플레이어를 $\boldsymbol{u}(4,3)$만큼 움직였다.
3. 플레이어를 $\boldsymbol{v}(2,5)$만큼 움직였다.
4. 최종 위치는?

$$\boldsymbol{u}+\boldsymbol{v}=\boldsymbol{w}$$
$$=(4,3)+(2,5)$$
$$=(4\hat{x}+3\hat{y})+(2\hat{x}+5\hat{y})$$
$$=(4+2)\hat{x}+(3+5)\hat{y}$$
$$=(4+2,3+5)$$
$$=(6,8)$$

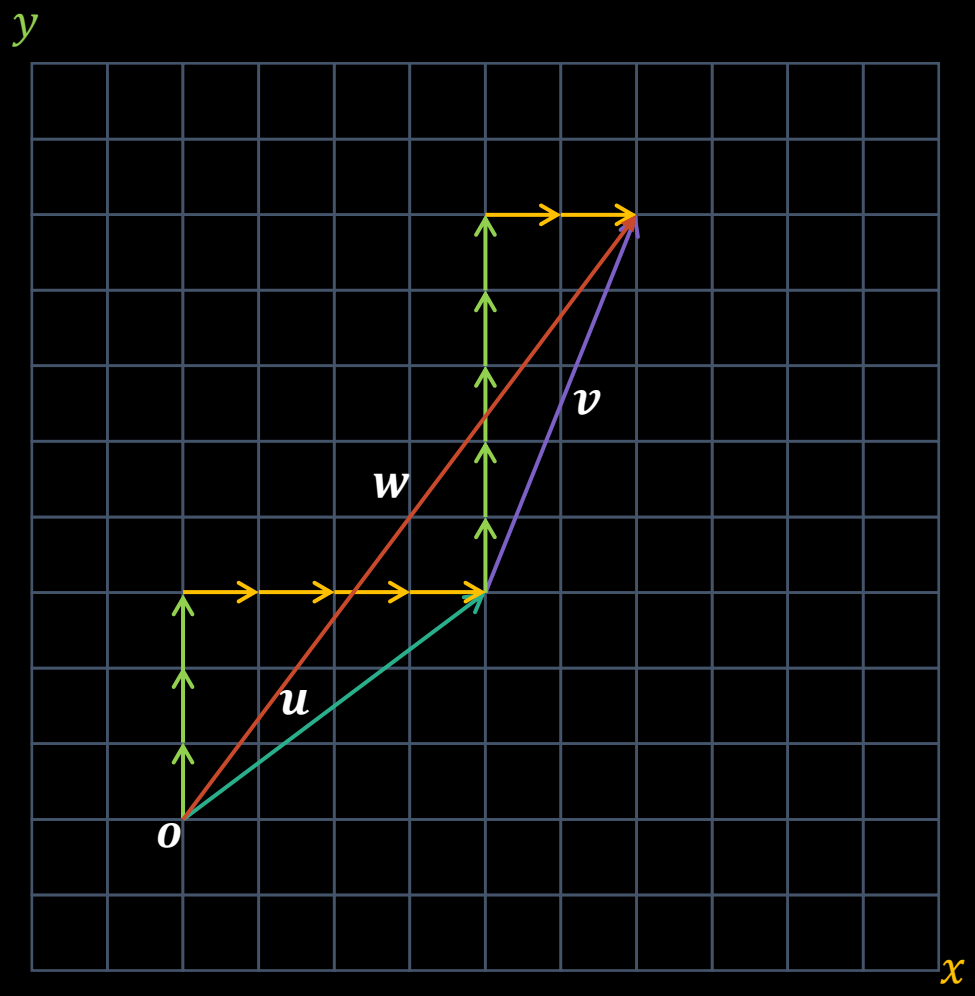
벡터 덧셈은 그냥 같은 자리에 있는 것 끼리 더하면 됩니다.

벡터의 덧셈을 정의했기때문에 마치 실수 연산을 하듯, 연산자를 써서 식을 쓸 수도 있습니다.

앞으로는 스칼라 수와 벡터를 구분하기 위해 벡터는 특별히 **굵은 소문자**로 쓰겠습니다.

수학적인 배경지식::벡터

벡터 덧셈 함수

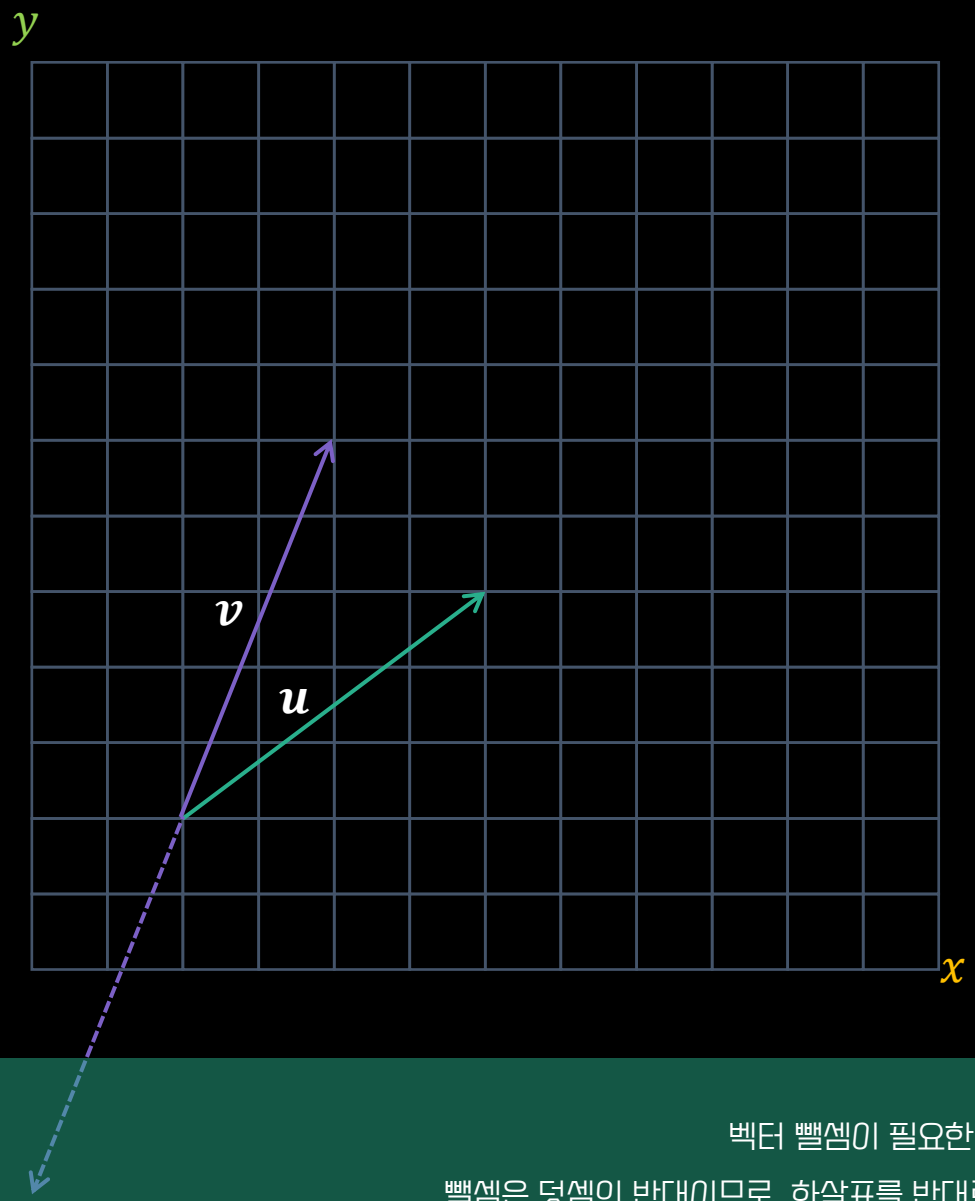



```
t_vec   vec_add(t_vec a, t_vec b)
{
    a.x += b.x;
    a.y += b.y;
    return (a);
}
```

벡터를 더하는 함수를 만들어 봅시다

수학적인 배경지식::벡터

벡터 뺄셈



1. 플레이어는 $(0, 0)$에 있다.
2. 플레이어를 $(4, 3)$만큼 움직였다.
3. 플레이어를 $(2, 5)$만큼 반대로 움직였다.
4. 최종 위치는?

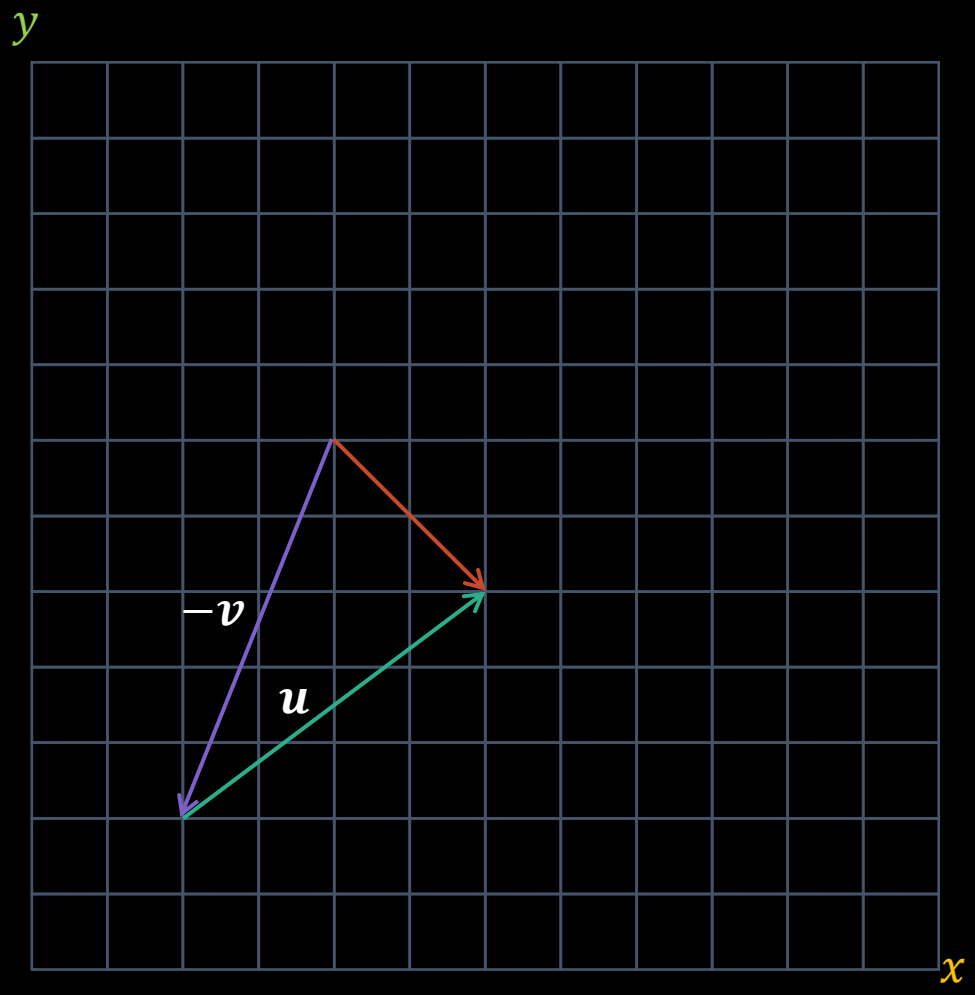
$$\boldsymbol{u} - \boldsymbol{v} =$$
$$= (4, 3) - (2, 5)$$
$$= (4 - 2, 3 - 5)$$
$$= (2, -2)$$

벡터 뺄셈이 필요한 경우를 생각해 봅시다.

뺄셈은 덧셈의 반대이므로, 화살표를 반대로 그린 다음 덧셈을 수행하면 될 것 같습니다.

수학적인 배경지식::벡터

벡터 뺄셈



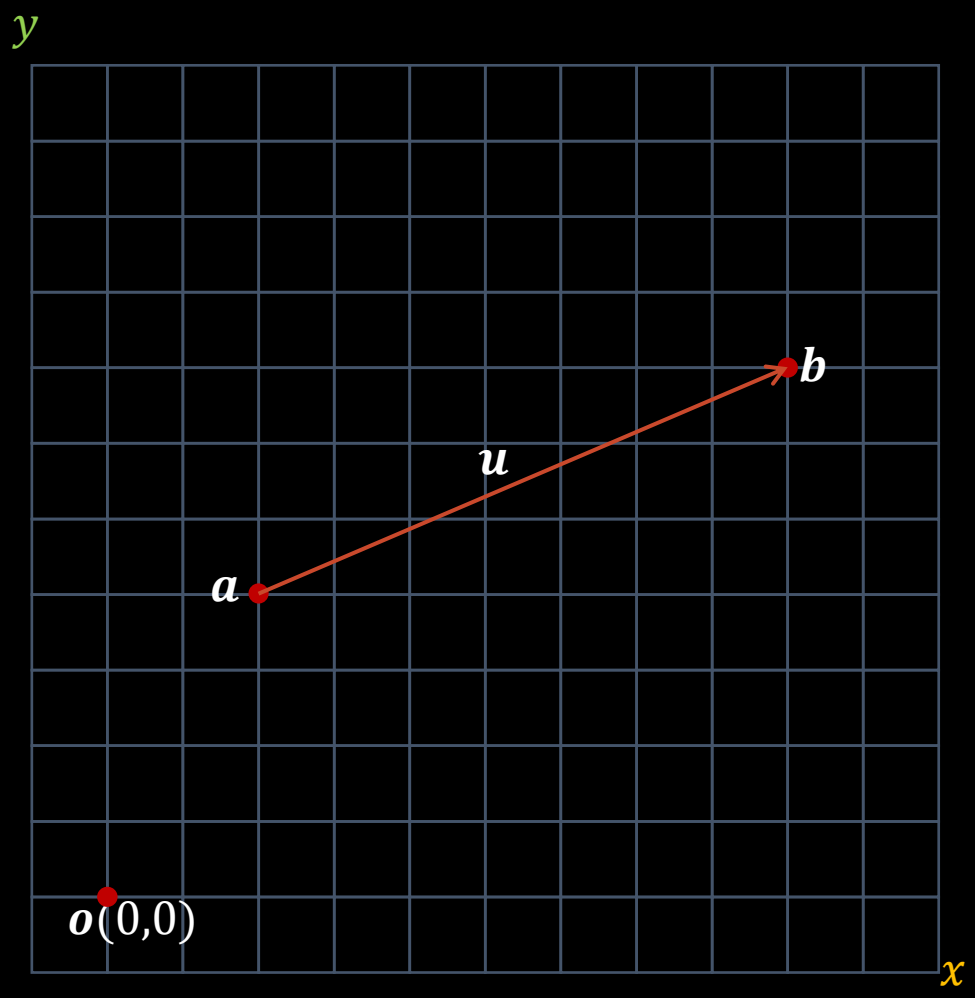
1. 플레이어는 $(0, 0)$에 있다.
2. 플레이어를 $(4, 3)$만큼 움직였다.
3. 플레이어를 $(2, 5)$만큼 반대로 움직였다.
4. 최종 위치는?

$$\boldsymbol{u} - \boldsymbol{v} =$$
$$= (4, 3) - (2, 5)$$
$$= (4 - 2, 3 - 5)$$
$$= (2, -2)$$

맞는 말이지만, 그것보다는 $\boldsymbol{u}$의 머리와 $\boldsymbol{v}$머리를 잇는 것으로 이해하는 편이 쓸모가 많습니다.

수학적인 배경지식::벡터

벡터 뺄셈



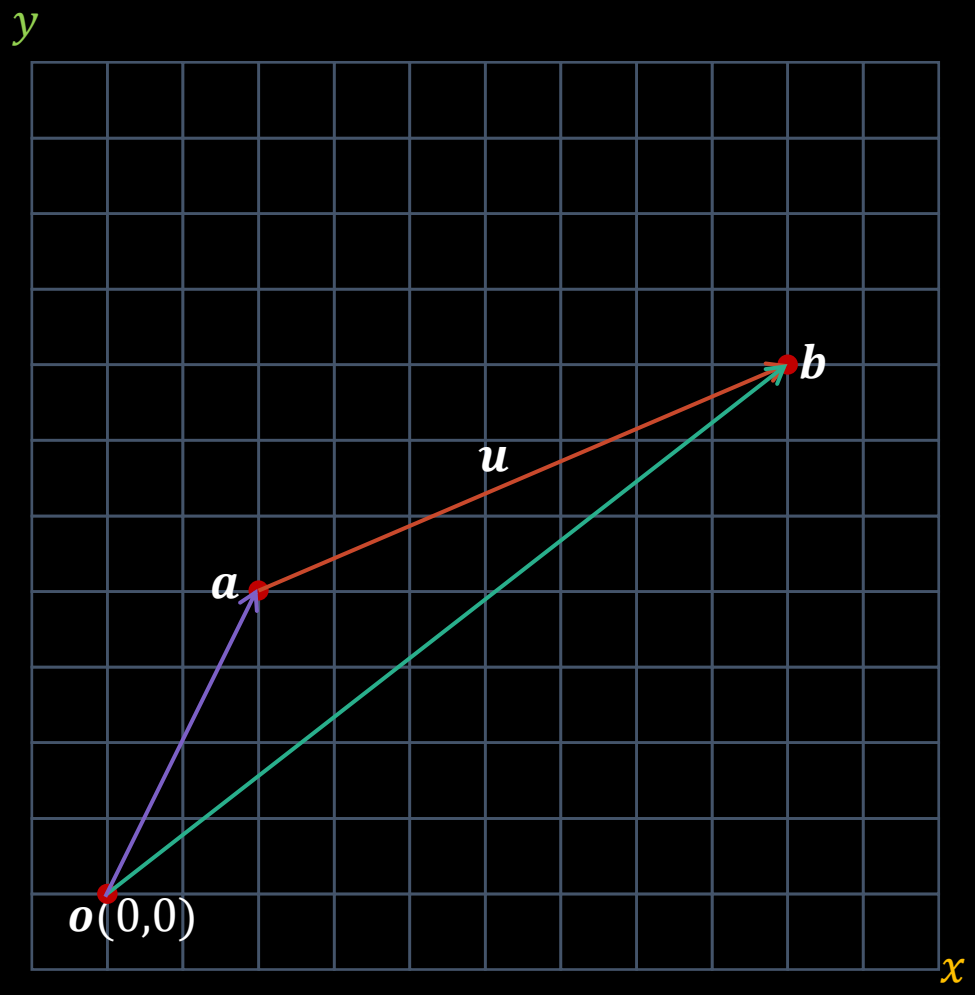
1. 플레이어는 $a(2,4)$에 있다.
2. 물체는 $b(9,7)$에 있다.
3. 플레이어를 기준으로 한 물체의 상대적인 위치는?

벡터는 점과 다를 바 없다는 것을 떠올리면서 이 문제를 고민해봅시다.

벡터의 뺄셈은 상대적인 위치를 파악할 때 요긴하게 쓰입니다.

수학적인 배경지식::벡터

벡터 뺄셈



1. 플레이어는 $\boldsymbol{a}(2,4)$에 있다.
2. 물체는 $\boldsymbol{b}(9,7)$에 있다.
3. 플레이어를 기준으로 한 물체의 상대적인 위치는?

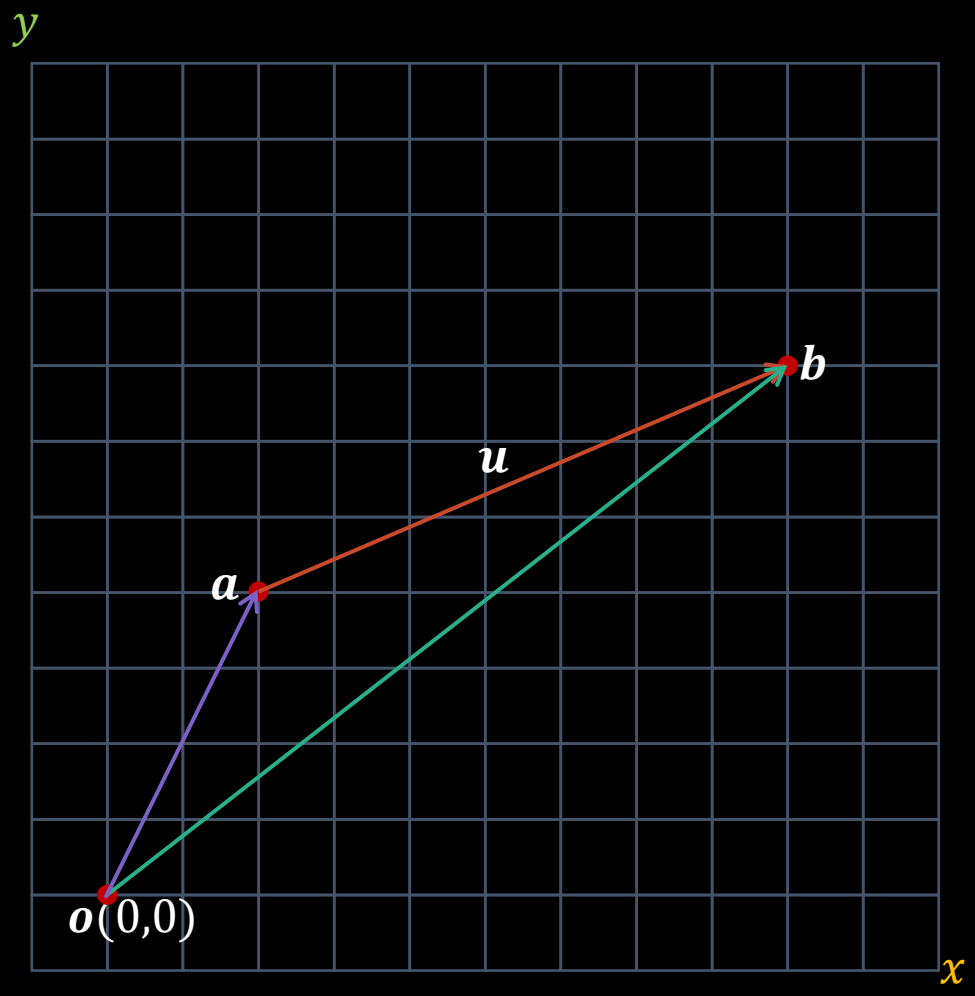
   $\rightarrow \boldsymbol{u} = \boldsymbol{b} - \boldsymbol{a}$

관찰 대상의 좌표(벡터)에서 나의 좌표를 빼면, 나를 기준으로 한 관찰대상의 좌표를 구할 수 있습니다.

매운맛) 내가 보는 방향이 바뀌기도 합니다. 그러면 뺄셈만으로는 해결이 안됩니다. 그건 나중에 다뤄 보기로 합시다.

수학적인 배경지식::벡터

벡터 뺄셈 함수

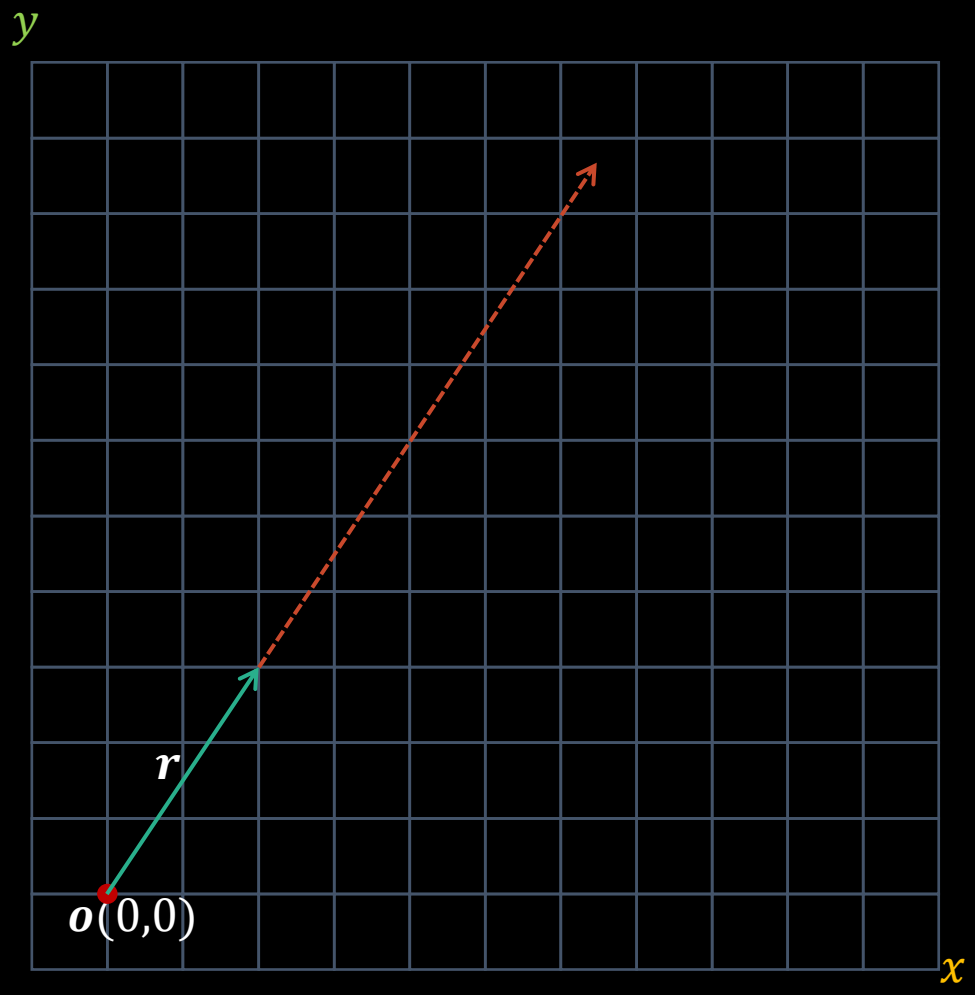



```
t_vec   vec_sub(t_vec a, t_vec b)
{
    a.x -= b.x;
    a.y -= b.y;
    return (a);
}
```

벡터 뺄셈 함수를 만들어 봅시다.

수학적인 배경지식::벡터

벡터 스칼라곱



1. 플레이어는 $\boldsymbol{a}(0,0)$에 있다.
2. 플레이어가 광선을 쏜다.

   광선은 1초에 $\boldsymbol{r} = (2,3)$만큼 움직인다.
3. 3초 뒤 광선의 위치$\boldsymbol{h}$는?

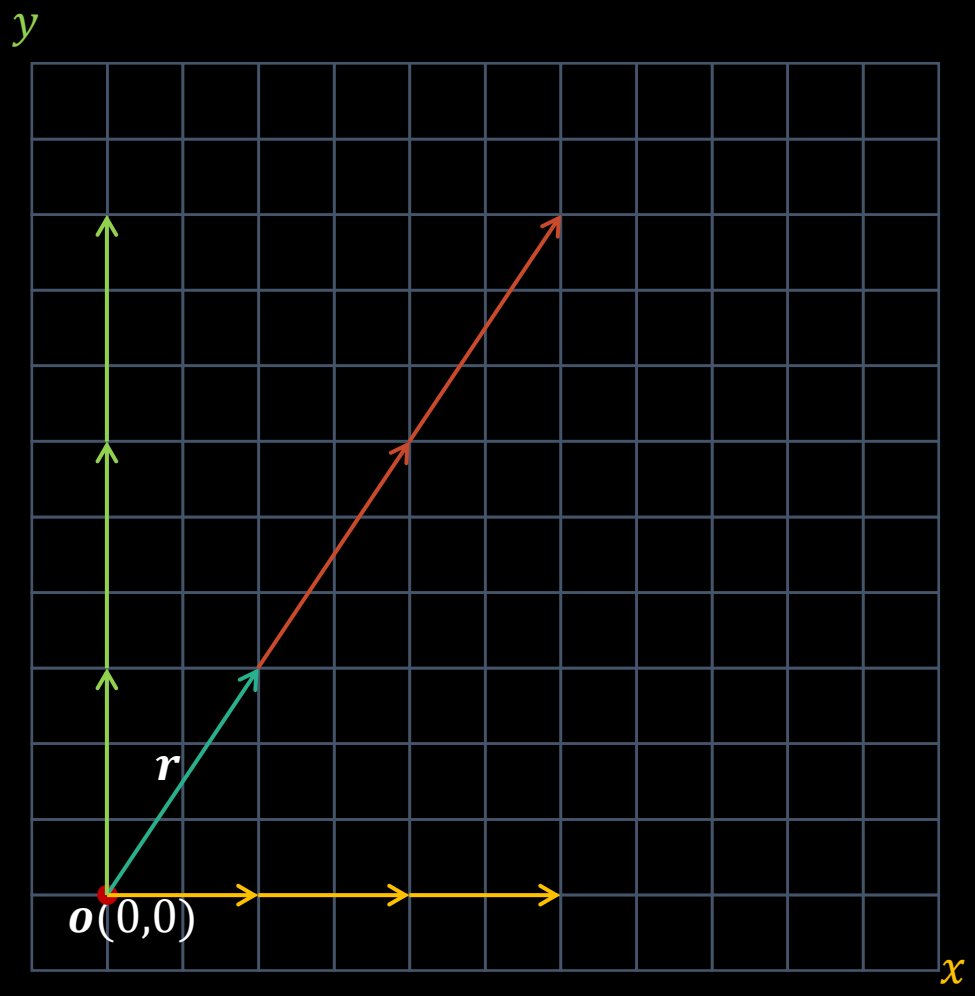
이번엔 스칼라 곱에 대해 알아봅시다.

벡터에 스칼라곱을 한다는 것은 벡터를 잡아 늘리거나 줄이는 것으로 이해해도 무방합니다.

그냥 곱셈이 아니라 “스칼라 곱”인 이유는 벡터의 곱셈 방법이 여러가지이기 때문입니다.

수학적인 배경지식::벡터

벡터 스칼라곱



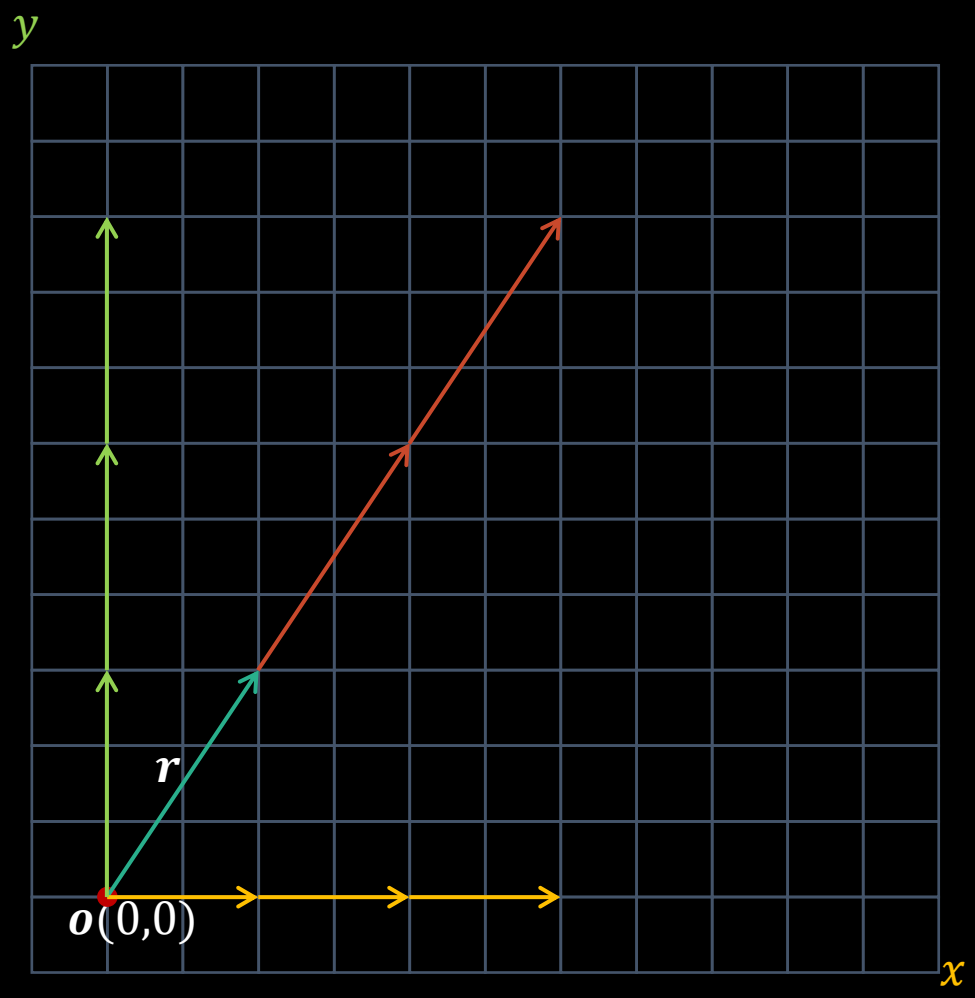
1. 플레이어는 $a(0,0)$에 있다.
2. 플레이어가 광선을 쏜다.
   광선은 1초에 $r = (2,3)$만큼 움직인다.
3. 3초 뒤 광선의 위치h는?

$$3 \cdot r$$
$$= 3 \cdot (2,3)$$
$$= (3 \cdot 2, 3 \cdot 3)$$
$$= (6,9)$$

스칼라 곱은 수를 각 요소에 곱해주면 됩니다.

수학적인 배경지식::벡터

벡터 스칼라곱 함수



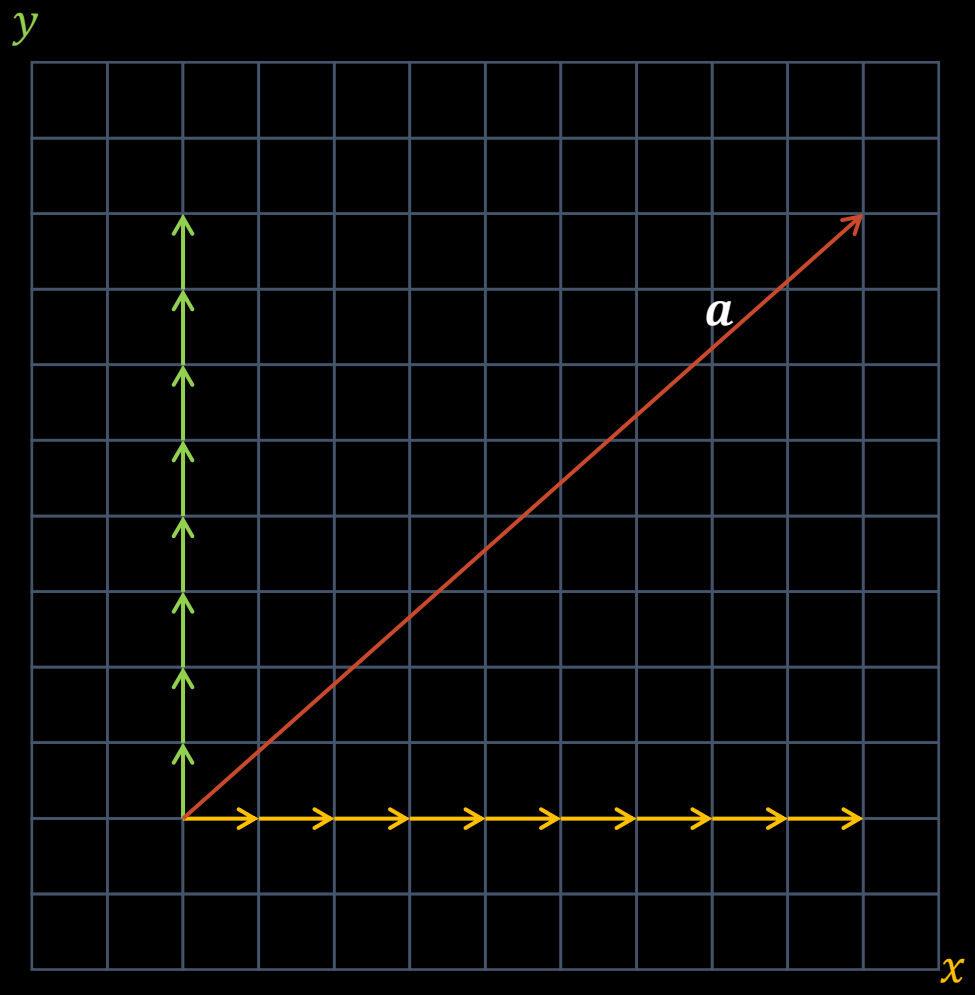

```
t_vec   vec_mul(t_vec a, double b)
{
    a.x *= b;
    a.y *= b;
    return (a);
}
```

스칼라곱 함수를 만들어 봅시다.

벡터에는 점곱(dot product), 가위곱(cross product)도 있고, 내적이니 외적이니 하는 것도 있습니다. 하지만 당장 우리 과제에 필요하지는 않으므로 건너뛰도록 하겠습니다.

수학적인 배경지식::벡터

벡터와 연립방정식



$$(9,8) = 9\hat{x} + 8\hat{y}$$

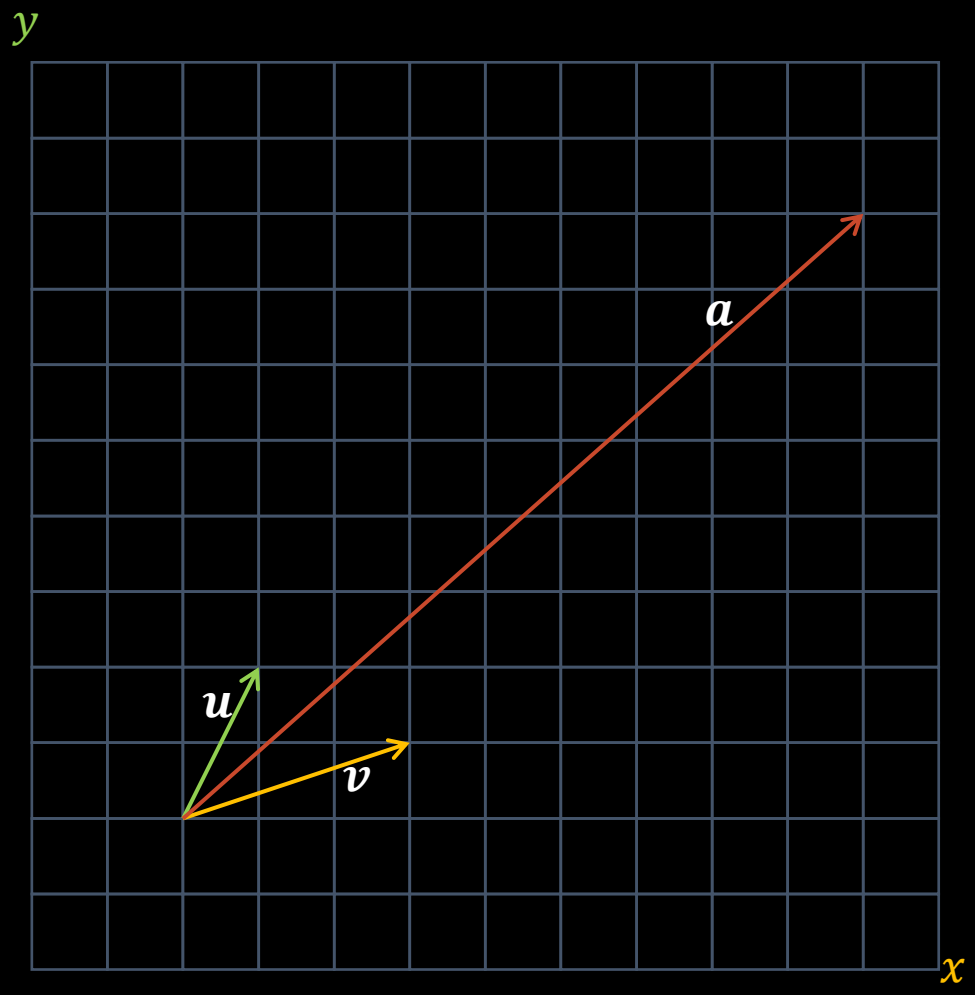
(9,8) 이라는 벡터는 x축방향으로 9만큼, y축방향으로 8만큼 떨어진 곳을 가리킵니다.

$\hat{x}(1,0)$을 9배하고 $\hat{y}(0,1)$을 8배 한 벡터를 더한 것을 (9,8)이라고 표현한다면

$\hat{x}$ $\hat{y}$가 아닌 다른 벡터로는 좌표계를 만들 수 없을까요?

수학적인 배경지식::벡터

벡터와 연립방정식

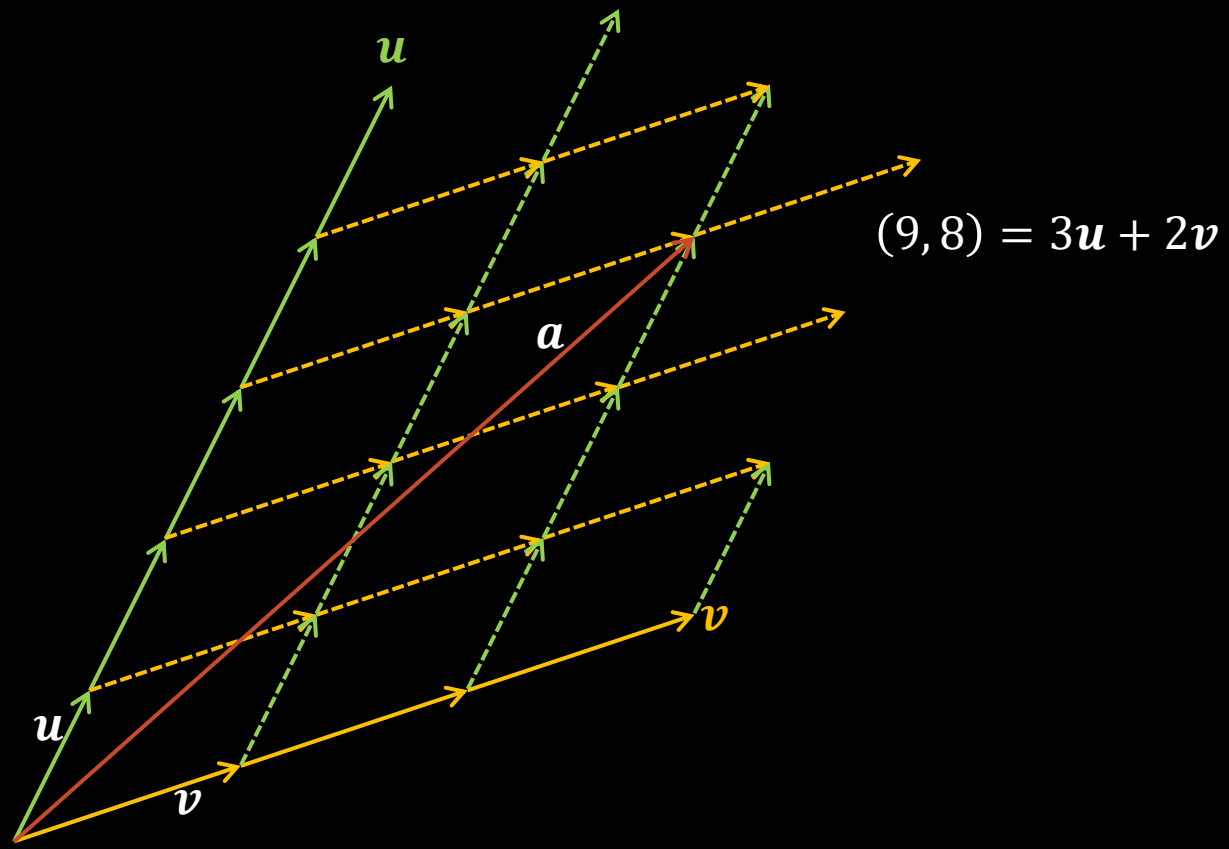


$$(9,8) = s\boldsymbol{u} + t\boldsymbol{v}$$

이렇게 $\boldsymbol{u}(1,2)$ 하고 $\boldsymbol{v}(3,1)$를 가지고 새로운 좌표계를 만들어보면 어떨까요?

수학적인 배경지식::벡터

벡터와 연립방정식



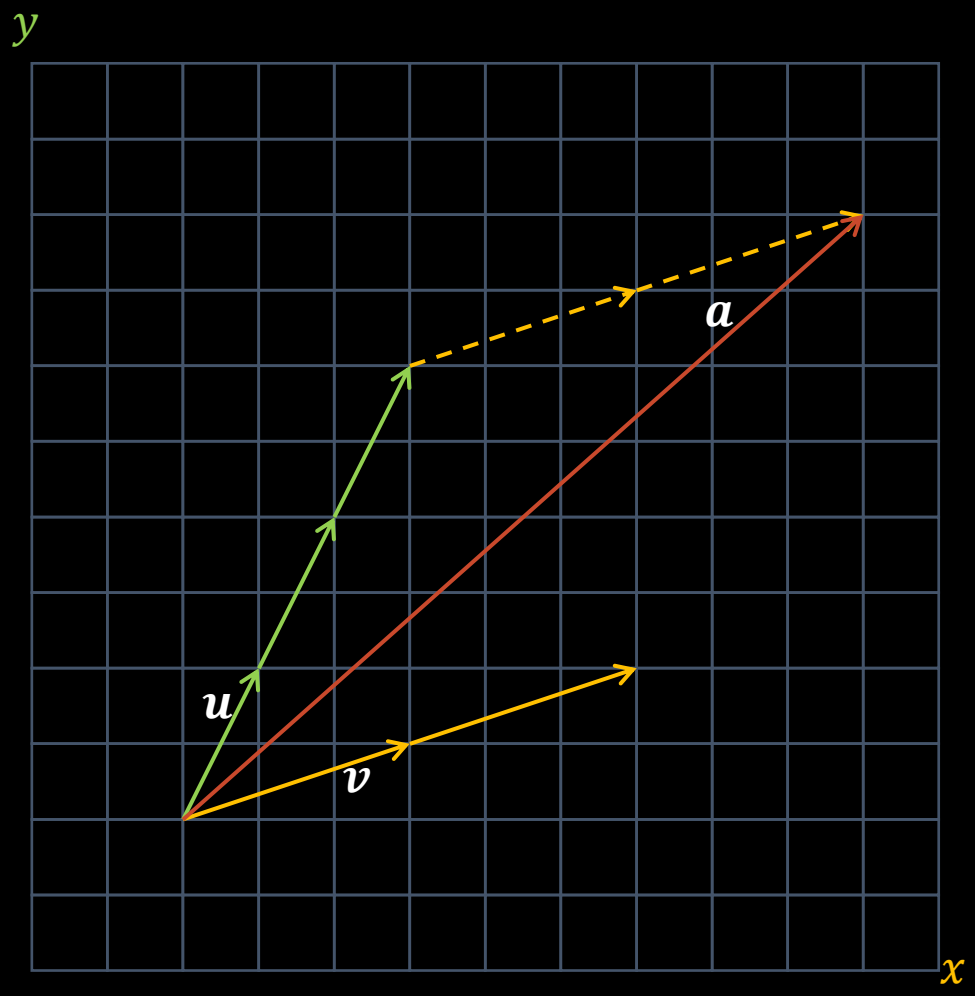
$$(9,8) = 3\boldsymbol{u} + 2\boldsymbol{v}$$

이렇게 $\boldsymbol{u}(1,2)$ 하고 $\boldsymbol{v}(3,1)$를 가지고 새로운 좌표계를 만들어보면 어떨까요?

보아하니 이 새로운 좌표계에서는 $\boldsymbol{a}$를(3,2)로 쓸 수 있겠네요.

수학적인 배경지식::벡터

벡터와 연립방정식



$$(9,8) = 3\boldsymbol{u} + 2\boldsymbol{v}$$

$$\boldsymbol{a} = (9,8)?$$
$$or$$
$$\boldsymbol{a} = (3,2)?$$

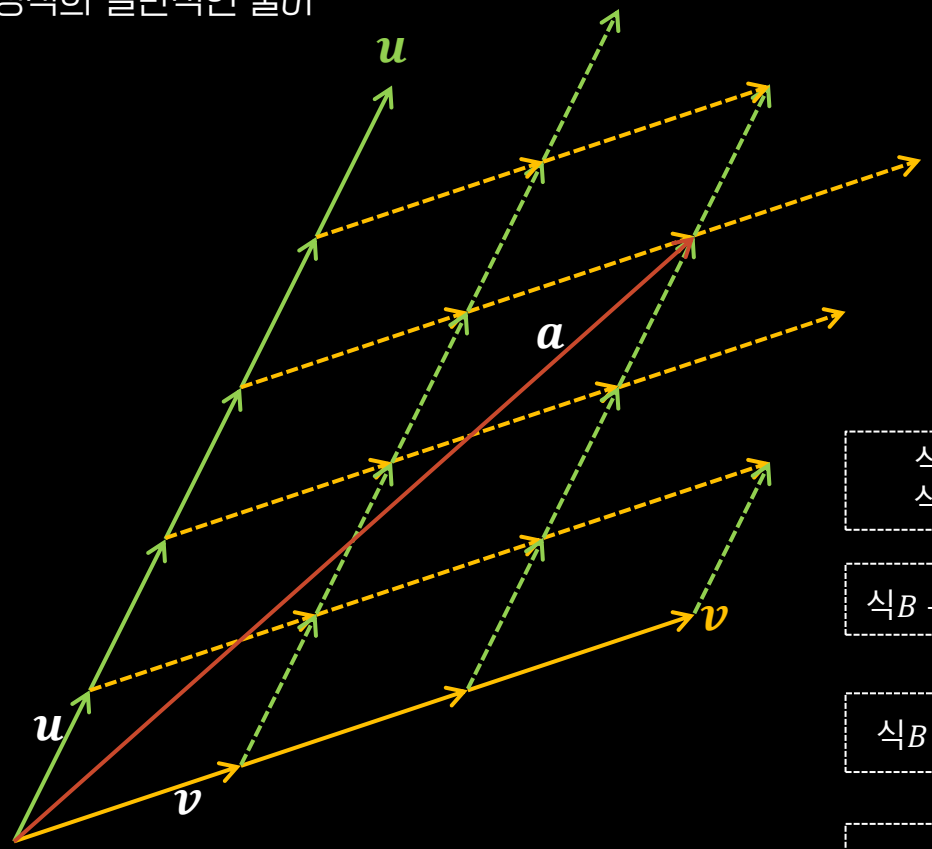
이렇게 벡터가 두 개 있으면 새로운 좌표계를 만들 수 있습니다.

2 개 있으면 2차원 평면을, 3개 있으면 3차원 공간을 만들 수 있고, 4개 있으면 상상하기 어렵지만 4차원 공간을 만들 수도 있습니다.

하지만 이 문제는 지금 다루기에는 어려운 문제인 것 같고, 우선은 (3,2)를 작도가 아니라 계산으로 알아내는 방법을 알아봅시다.

수학적인 배경지식::벡터

벡터와 연립방정식::연립방정식의 일반적인 풀이



$$\boldsymbol{a} = s\boldsymbol{u} + t\boldsymbol{v}$$

$$u = (1,2), v = (3,1), a = (9,8)$$

$$s(u_x, u_y) + t(v_x, v_y) = (a_x, a_y)$$

$$\begin{cases} u_x s + v_x t = a_x \\ u_y s + v_y t = a_y \end{cases}$$

$$\begin{cases} 1s + 3t = 9 \cdots 식A \\ 2s + 1t = 8 \cdots 식B \end{cases}$$

식$A \div 1$
식$B \div 2$ → $\begin{cases} s + 3t = 9 \\ s + 0.5t = 4 \end{cases}$

식$B -$ 식A → $\begin{cases} s + 3t = 9 \\ -2.5t = -5 \end{cases}$

식$B \div (-2.5)$ → $\begin{cases} s + 3t = 9 \\ t = 2 \end{cases}$

식$A -$ 식$B \times 3$ → $\begin{cases} s = 3 \\ t = 2 \end{cases}$

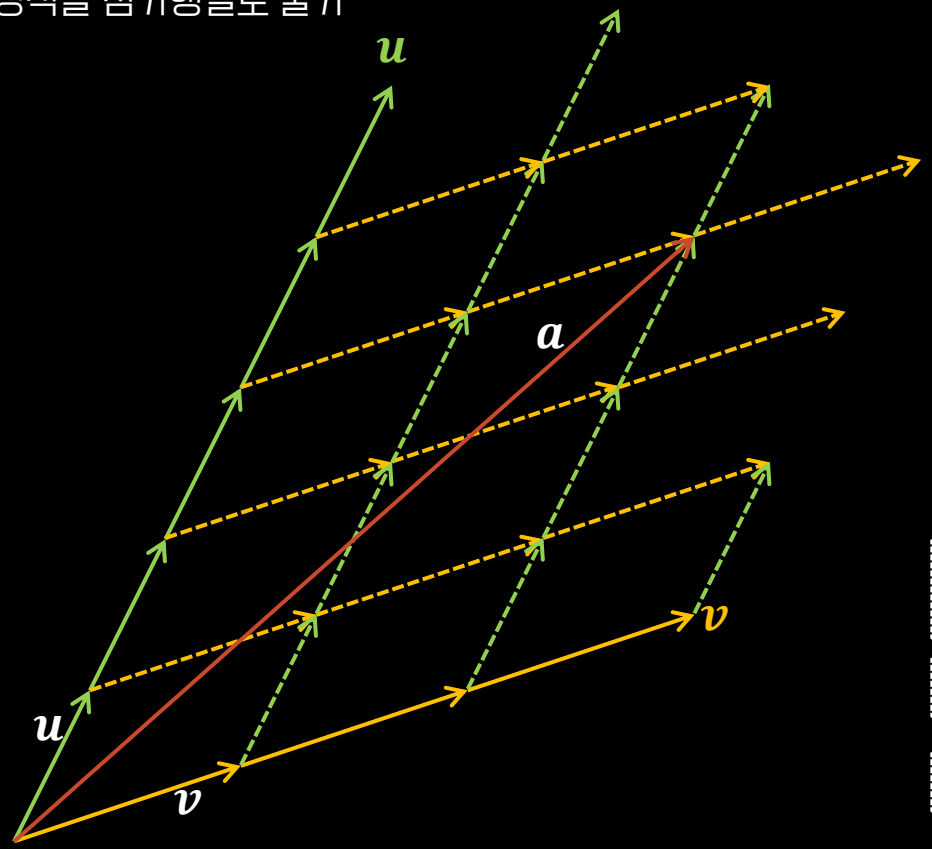
$\boldsymbol{u}$를 적당히 잡아 당긴 벡터와 $\boldsymbol{v}$를 적당히 잡아 당긴 벡터를 더하면 $\boldsymbol{a}$가 되는 것을 토대로 식을 세워봅시다.

식을 세우면 전형적인 연립방정식이 됩니다.

연립방정식을 푸는 방법은 중학교때 배웠습니다.

수학적인 배경지식::벡터

벡터와 연립방정식::연립방정식을 첨가행렬로 풀기



$$\boldsymbol{a} = s\boldsymbol{u} + t\boldsymbol{v}$$

$$u = (1,2), v = (3,1), a = (9,8)$$

$$s(u_x, u_y) + t(v_x, v_y) = (a_x, a_y)$$

$$\begin{cases} u_x s + v_x t = a_x \\ u_y s + v_y t = a_y \end{cases}$$

$$\left[\begin{array}{cc|c} u_x & v_x & a_x \\ u_y & v_y & a_y \end{array}\right]$$

$$\left[\begin{array}{cc|c} 1 & 3 & 9 \\ 2 & 1 & 8 \end{array}\right]$$

$line1 \div 1$
$line2 \div 2$ → $\left[\begin{array}{cc|c} 1 & 3 & 9 \\ 1 & 0.5 & 4 \end{array}\right]$

$line2 - line1$ → $\left[\begin{array}{cc|c} 1 & 3 & 9 \\ 0 & -2.5 & -5 \end{array}\right]$

$line2 \div (-2.5)$ → $\left[\begin{array}{cc|c} 1 & 3 & 9 \\ 0 & 1 & 2 \end{array}\right]$

$line1 - line2 \times 3$ → $\left[\begin{array}{cc|c} 1 & 0 & 3 \\ 0 & 1 & 2 \end{array}\right]$

옛날 옛적에 컴퓨터도 없고 그런 시절에 손으로 방정식을 풀던 수학자들이 현타에 빠지기 시작했습니다.

"아, 이거 미지수 뻔한 거 맨날 쓰기도 귀찮고, 방법도 획일화 하는 좋겠다"

그래서 나온게 행렬입니다. 행렬 오른쪽에 상수항을 "첨가"해서 방정식을 풀기 시작합니다. 이 행렬을 "첨가행렬"이라고 합니다.

수학적인 배경지식::벡터

첨가행렬로 연립방정식을 푸는 알고리즘

$$\left[\begin{array}{ccc|c} a_{11} & a_{12} & a_{13} & b_1 \\ a_{21} & a_{22} & a_{23} & b_2 \\ a_{31} & a_{32} & a_{33} & b_3 \end{array}\right] \begin{array}{c} = A_1 \\ = A_2 \\ = A_3 \end{array}$$

$$\left[\begin{array}{ccc|c} 1 & a_{12} & a_{13} & b_1 \\ 0 & 1 & a_{23} & b_2 \\ 0 & 1 & a_{33} & b_3 \end{array}\right] \begin{array}{c} A_1 \\ \div a_{22} \\ \div a_{32} \end{array}$$

$$\left[\begin{array}{ccc|c} 1 & a_{12} & a_{13} & b_1 \\ 0 & 1 & 0 & b_2 \\ 0 & 0 & 1 & b_3 \end{array}\right] \begin{array}{c} A_1 \\ A_2 - A_3 * a_{23} \\ A_3 \end{array}$$

$$\left[\begin{array}{ccc|c} 1 & a_{12} & a_{13} & b_1 \\ 1 & a_{22} & a_{23} & b_2 \\ 1 & a_{32} & a_{33} & b_3 \end{array}\right] \begin{array}{c} \div a_{11} \\ \div a_{21} \\ \div a_{31} \end{array}$$

$$\left[\begin{array}{ccc|c} 1 & a_{12} & a_{13} & b_1 \\ 0 & 1 & a_{23} & b_2 \\ 0 & 0 & a_{33} & b_3 \end{array}\right] \begin{array}{c} A_1 \\ A_2 \\ A_3 - A_2 \end{array}$$

$$\left[\begin{array}{ccc|c} 1 & 0 & 0 & b_1 \\ 0 & 1 & 0 & b_2 \\ 0 & 0 & 1 & b_3 \end{array}\right] \begin{array}{c} A_1 - A_2 \times a_{12} - A_3 \times a_{13} \\ A_2 \\ A_3 \end{array}$$

$$\left[\begin{array}{ccc|c} 1 & a_{12} & a_{13} & b_1 \\ 0 & a_{22} & a_{23} & b_2 \\ 0 & a_{32} & a_{33} & b_3 \end{array}\right] \begin{array}{c} A_1 \\ A_2 - A_1 \\ A_3 - A_1 \end{array}$$

$$\left[\begin{array}{ccc|c} 1 & a_{12} & a_{13} & b_1 \\ 0 & 1 & a_{23} & b_2 \\ 0 & 0 & 1 & b_3 \end{array}\right] \begin{array}{c} A_1 \\ A_2 \\ \div a_{33} \end{array}$$

"a_{11}로 나눴는데 왜 그대로냐??"

컴퓨터 변수처럼 생각하면 됩니다. 변화 하는 걸 모두 쓰기가 너무 힘듭니다ㅠㅠ

아무튼 이렇게 방정식이 3개가 되든 4개가 되든 일정한 방식으로 처리할 수 있게 되었습니다. 이것을 가우스-조단 소거법이라고 합니다.

수학적인 배경지식::벡터

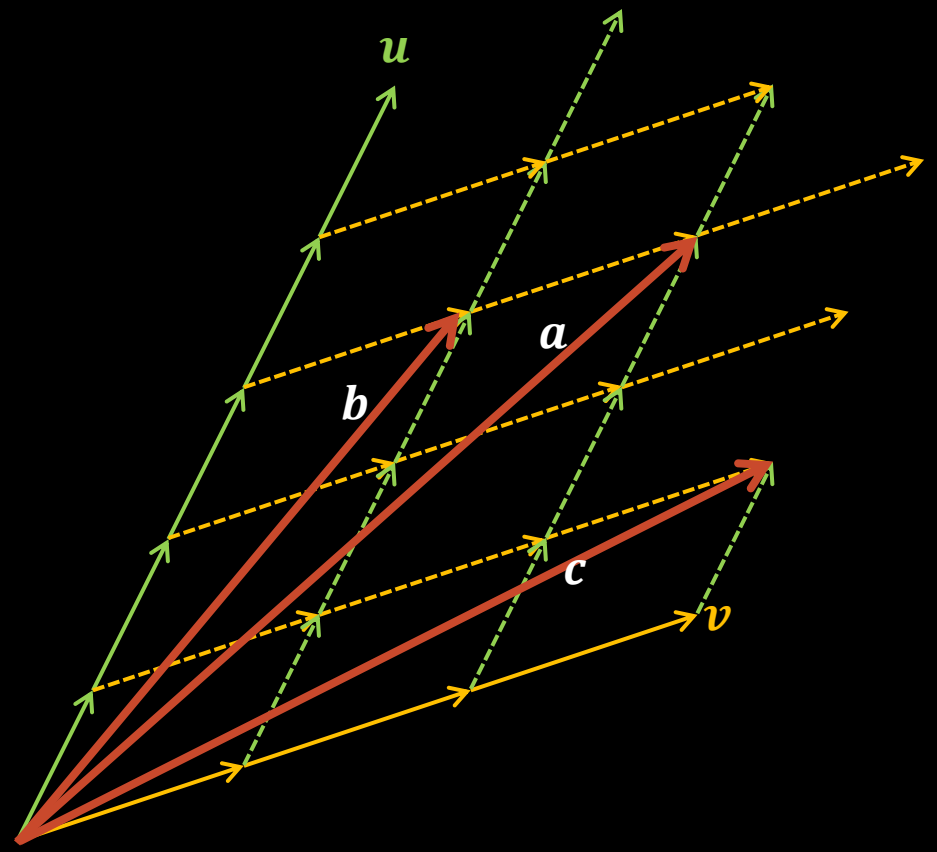
행렬 연산 정의

덧셈	$\begin{bmatrix} a & b \\ c & d \end{bmatrix} + \begin{bmatrix} p & q \\ r & s \end{bmatrix} = \begin{bmatrix} a+p & b+q \\ c+r & d+s \end{bmatrix}$	같은 행,열에 있는 요소끼리 덧셈
뺄셈	$\begin{bmatrix} a & b \\ c & d \end{bmatrix} - \begin{bmatrix} p & q \\ r & s \end{bmatrix} = \begin{bmatrix} a-p & b-q \\ c-r & d-s \end{bmatrix}$	같은 행,열에 있는 요소끼리 뺄셈
곱셈	$\begin{bmatrix} a & b \\ c & d \end{bmatrix}\begin{bmatrix} p & q \\ r & s \end{bmatrix} = \begin{bmatrix} ap-br & aq-bs \\ cp-dr & cq-ds \end{bmatrix}$	~~같은 행,열에 있는 요소끼리 곱셈~~ 복잡함
	$\begin{bmatrix} a & b \\ c & d \end{bmatrix}\begin{bmatrix} p & q \\ r & s \end{bmatrix} \neq \begin{bmatrix} p & q \\ r & s \end{bmatrix} \times \begin{bmatrix} a & b \\ c & d \end{bmatrix}$	교환법칙이 성립하지 않음 (자리바꾸기 금지)
단위행렬	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$	단위행렬, 항등행렬 == 곱셈의 항등원 마치 실수 사칙연산에서 1과 같은 것
역행렬	$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1}\begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1}\begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$	나눗셈은 없지만, 곱셈의 역원(사칙연산에서1/x같은 것)은 있음
	$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad-bc}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$	2차 정사각행렬의 역행렬

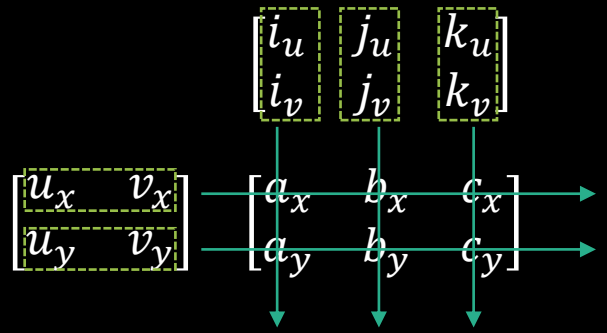
아무튼 행렬은 처음에 그렇게 탄생했는데, 만들고보니 쓸모가 많습니다. 마치 실수 연산과 비슷하게 행렬 연산을 정의해서 식으로 표현하기도 하죠.

수학적인 배경지식::벡터

행렬 곱셈에 대하여



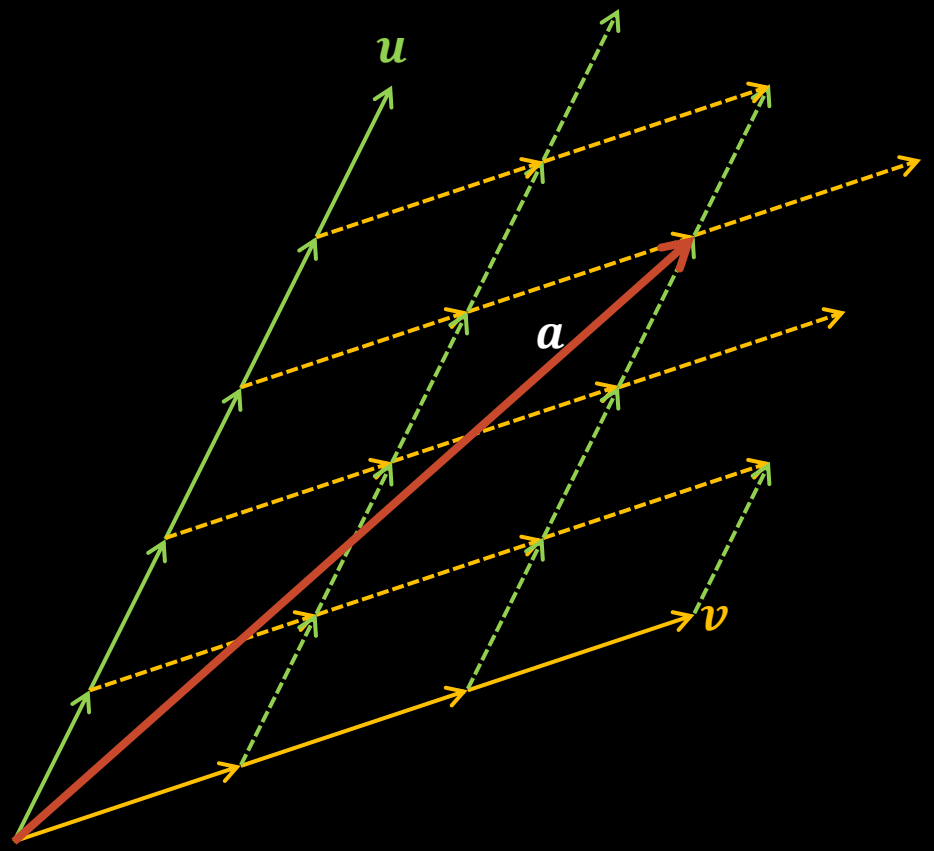
$$\begin{bmatrix} u_x & v_x \\ u_y & v_y \end{bmatrix} \begin{bmatrix} i_u & j_u & k_u \\ i_v & j_v & k_v \end{bmatrix} = \begin{bmatrix} a_x & b_x & c_x \\ a_y & b_y & c_y \end{bmatrix}$$



곱셈이 많이 복잡한데, 이렇게 이해하면 좋을 것 같습니다.

수학적인 배경지식::벡터

행렬 곱셈에 대하여



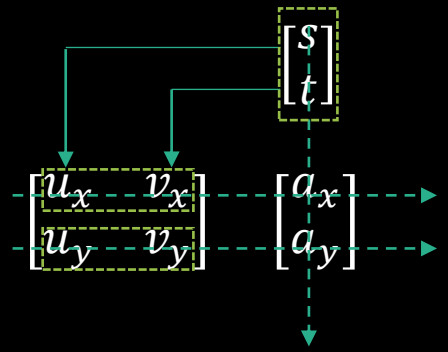
$$\boldsymbol{a} = s\boldsymbol{u} + t\boldsymbol{v}$$

$$u = (1,2), v = (3,1), a = (9,8)$$

$$s(u_x, u_y) + t(v_x, v_y) = (a_x, a_y)$$

$$\begin{cases} u_x s + v_x t = a_x \\ u_y s + v_y t = a_y \end{cases}$$

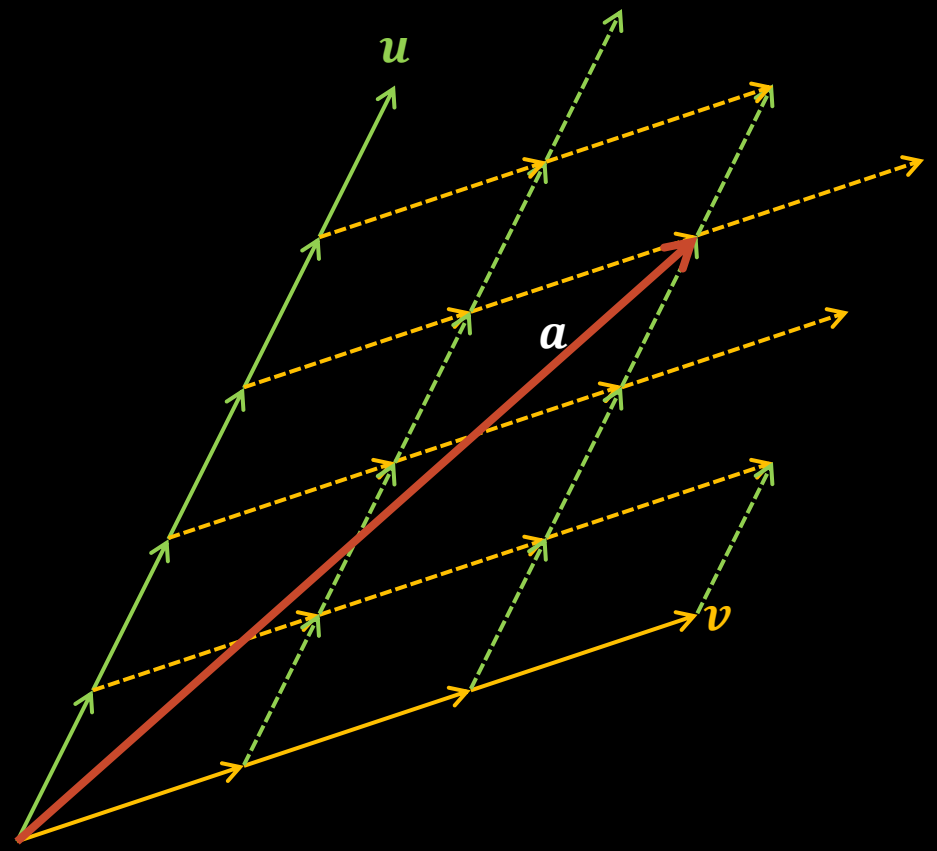
$$\begin{bmatrix} u_x & v_x \\ u_y & v_y \end{bmatrix} \begin{bmatrix} s \\ t \end{bmatrix} = \begin{bmatrix} a_x \\ a_y \end{bmatrix}$$



아무튼 우리에게 당장 필요한 것은 벡터로 만든 방정식이 연립방정식이라는 것입니다.

수학적인 배경지식::벡터

역행렬로 방정식 풀기



$$\begin{matrix} \boldsymbol{B} & \boldsymbol{x} & \boldsymbol{a} \end{matrix}$$

$$\begin{bmatrix} u_x & v_x \\ u_y & v_y \end{bmatrix} \begin{bmatrix} s \\ t \end{bmatrix} = \begin{bmatrix} a_x \\ a_y \end{bmatrix}$$

$$\boldsymbol{Bx} = \boldsymbol{a}$$

$$\boldsymbol{x} = \frac{\boldsymbol{a}}{\boldsymbol{B}}$$

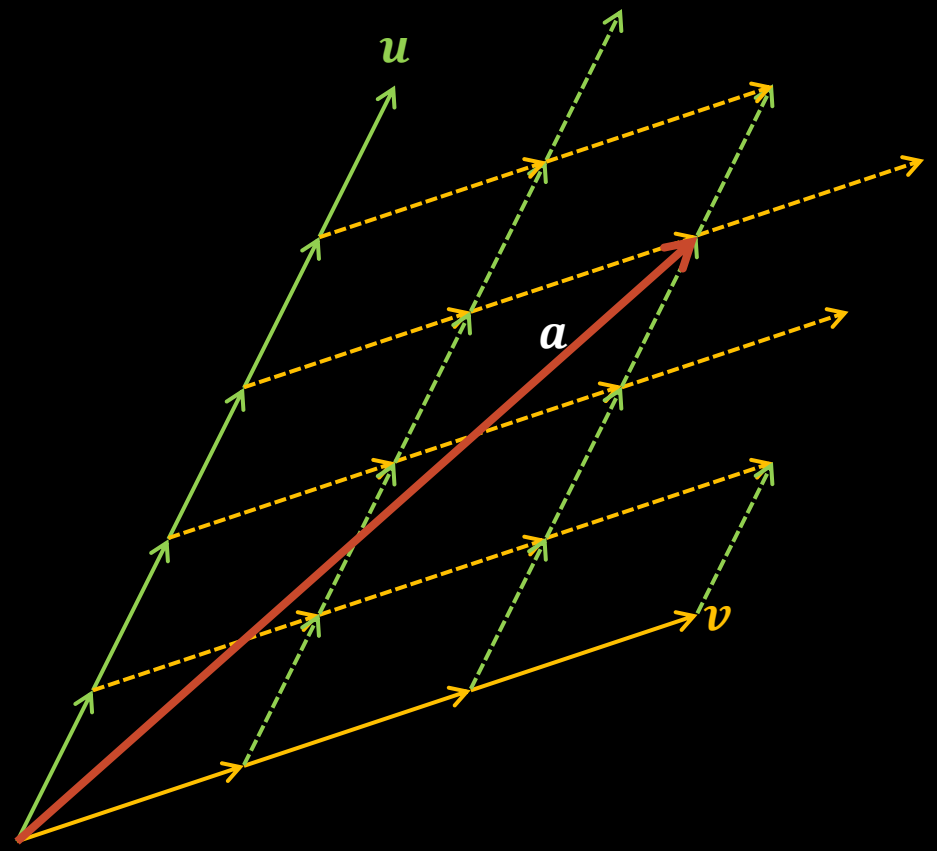
$$\boldsymbol{B^{-1}Bx} = \boldsymbol{B^{-1}a}$$

$$\boldsymbol{x} = \boldsymbol{B^{-1}a}$$

그리고 방정식을 가우스-조단 소거법으로 푸는 것 말고, 역행렬을 구해서 풀 수도 있습니다.

수학적인 배경지식::벡터

역행렬로 방정식 풀기



$$u = (1,2), v = (3,1), a = (9,8)$$

$$\begin{bmatrix} u_x & v_x \\ u_y & v_y \end{bmatrix} \begin{bmatrix} s \\ t \end{bmatrix} = \begin{bmatrix} a_x \\ a_y \end{bmatrix}$$

$$\begin{bmatrix} 1 & 3 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} s \\ t \end{bmatrix} = \begin{bmatrix} 9 \\ 8 \end{bmatrix}$$

$$\begin{bmatrix} s \\ t \end{bmatrix} = \begin{bmatrix} 1 & 3 \\ 2 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 9 \\ 8 \end{bmatrix}$$

$$\begin{bmatrix} s \\ t \end{bmatrix} = \frac{1}{1 \cdot 1 - 3 \cdot 2} \begin{bmatrix} 1 & -3 \\ -2 & 1 \end{bmatrix} \begin{bmatrix} 9 \\ 8 \end{bmatrix}$$

$$\begin{bmatrix} s \\ t \end{bmatrix} = \begin{bmatrix} -0.2 & 0.6 \\ 0.4 & -0.2 \end{bmatrix} \begin{bmatrix} 9 \\ 8 \end{bmatrix}$$

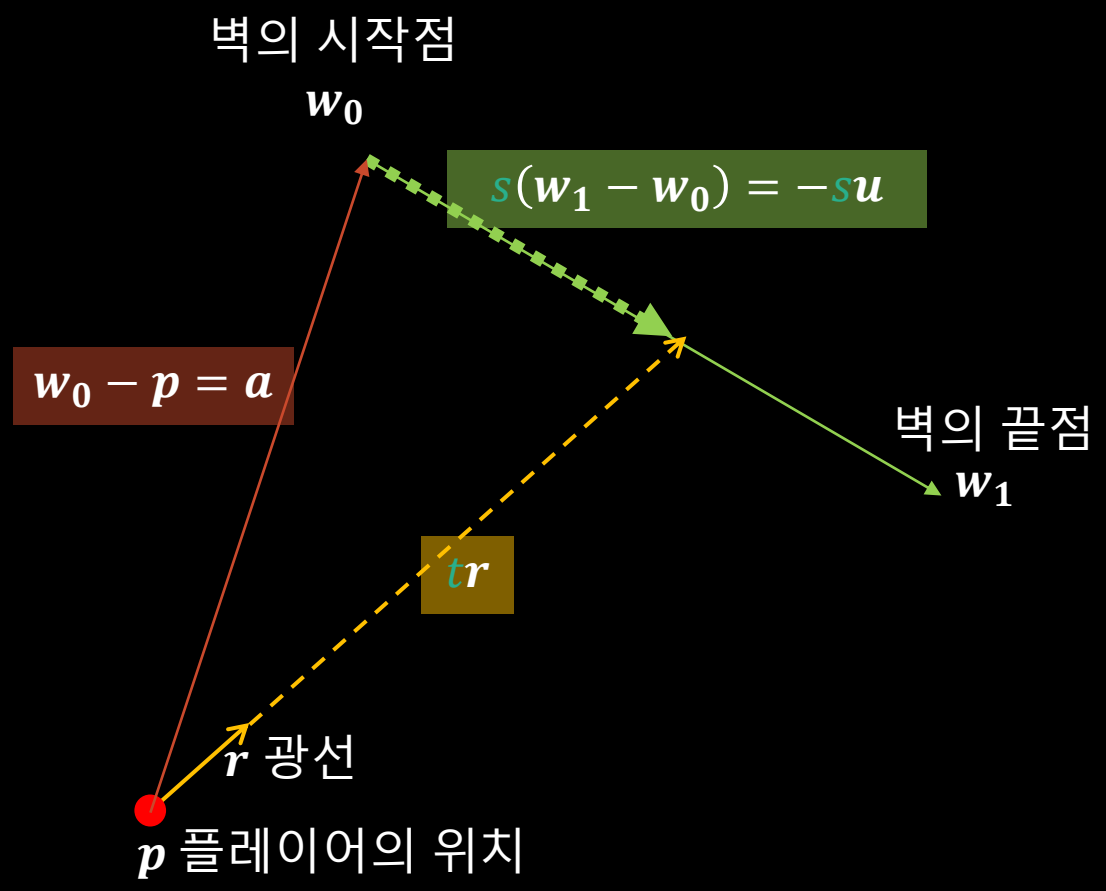
$$\begin{bmatrix} s \\ t \end{bmatrix} = \begin{bmatrix} -0.2 \cdot 9 + 0.6 \cdot 8 \\ 0.4 \cdot 9 - 0.2 \cdot 8 \end{bmatrix}$$

$$\begin{bmatrix} s \\ t \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

직접 해봅시다

수학적인 배경지식::벡터

벡터와 연립방정식 사용례



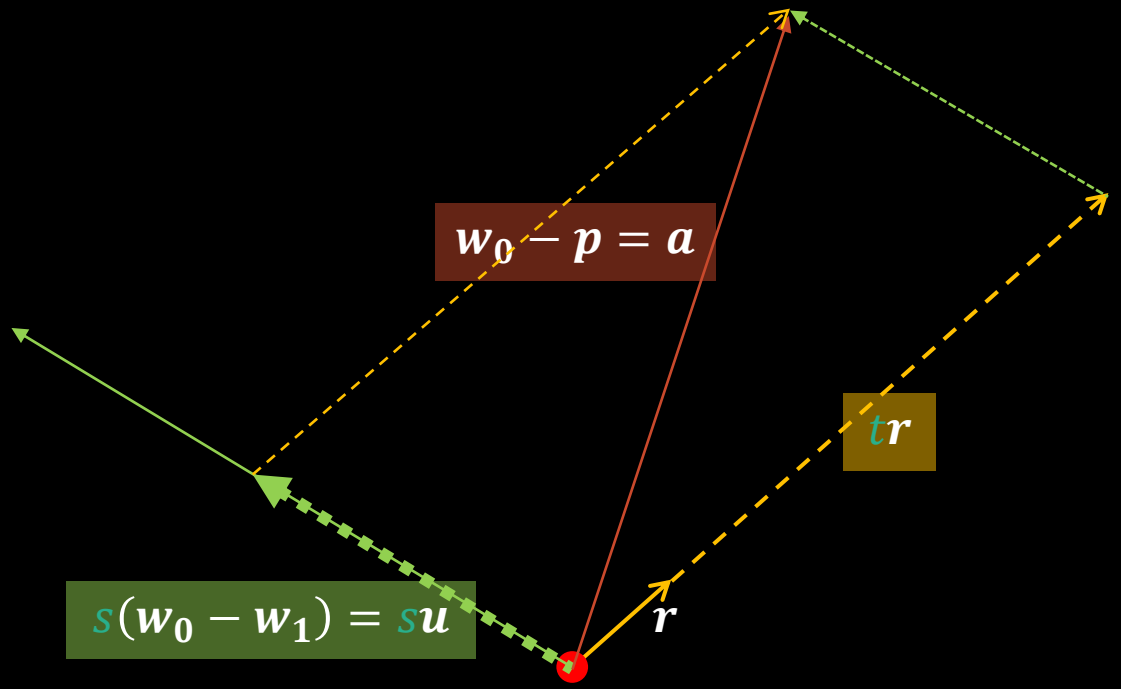
이걸 왜 하느냐면 우리 과제에 쓸모가 있기 때문입니다.

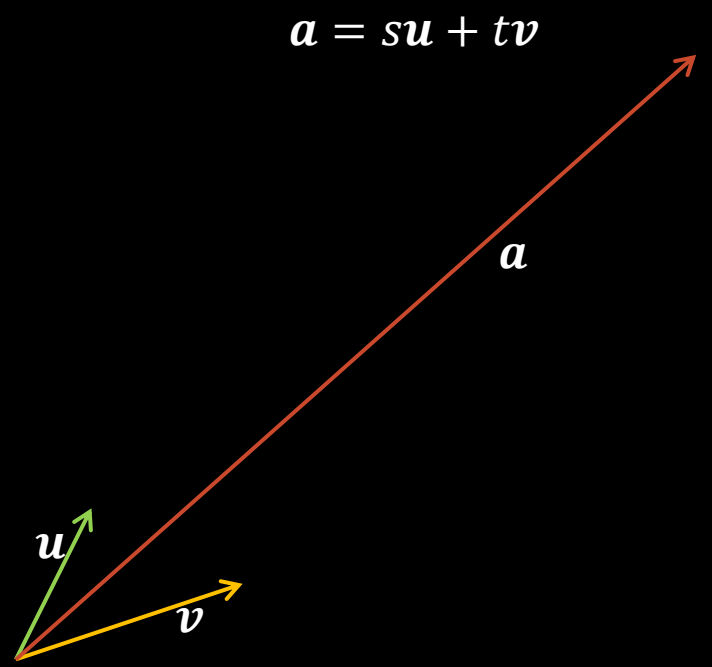
벡터는 그림표현을 숫자표현으로 이어주는 좋은 도구입니다.

3개 벡터가 이렇게 삼각형을 이루고 있는데, 이 그림을 토대로 우리가 벡터연산을 응용해서 식을 세워볼 수 있습니다.

수학적인 배경지식::벡터

벡터와 연립방정식 사용례





둘이 같은 꼴입니다. 오른쪽 문제를 해결할 수 있으면 왼쪽 문제도 해결가능합니다.

수학적인 배경지식::벡터

연립방정식 푸는 함수

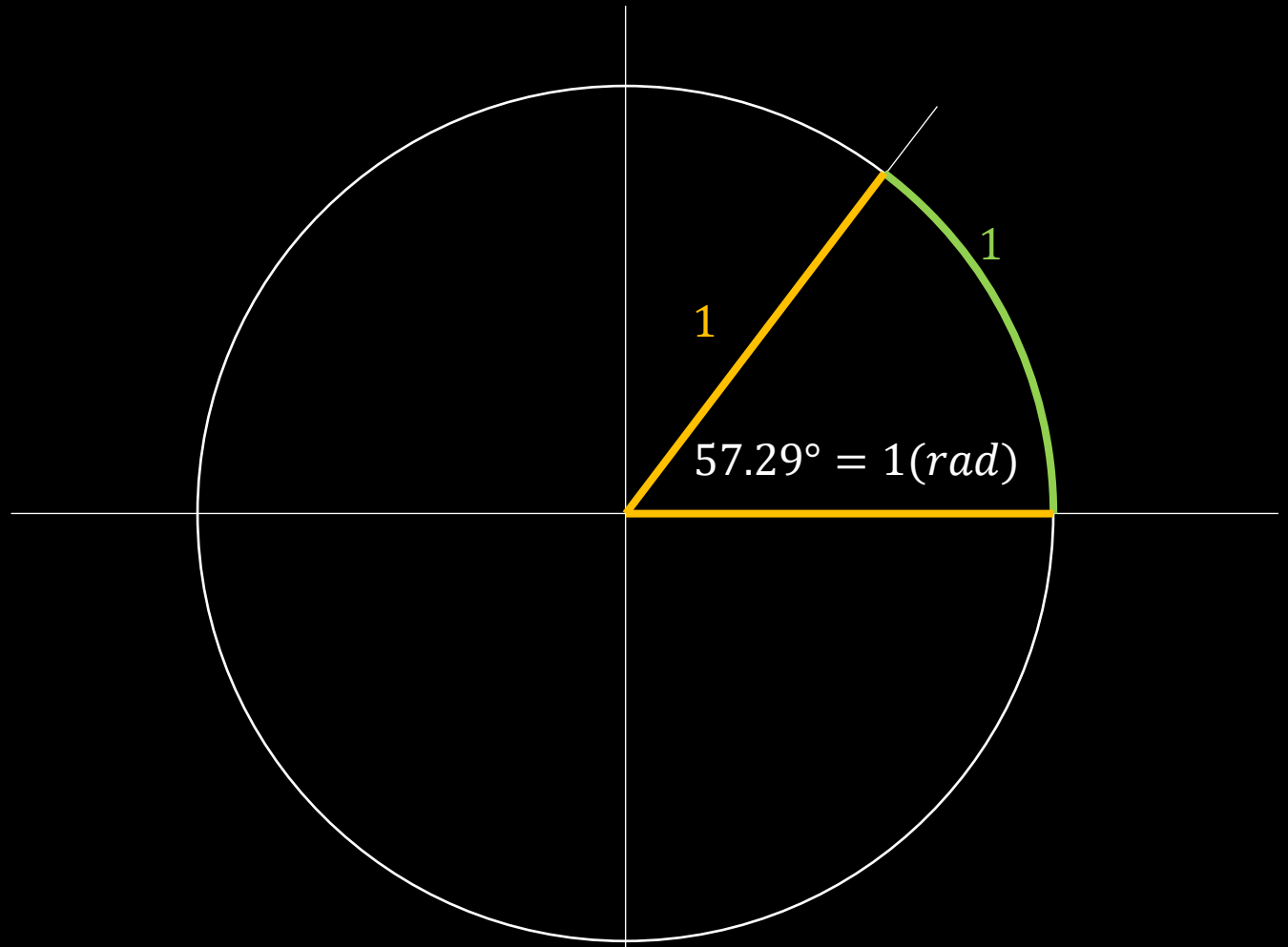
```
/*
**  equation_solver
**  input:
**      coeff0: coefficient0 column vector ptr
**      coeff1: coefficient1 column vector ptr
**      constant: constant column vector
**
**  explain:
**
**  |coeff0_x coeff1_x | |result_x|   |constant_x |
**  |                  | |        | = |           |
**  |coeff0_y coeff1_y | |result_y|   |constant_y |
**
**  return:
**      result vector
*/

t_vec   equation_solver(t_vec *coeff0, t_vec *coeff1, t_vec *constant)
{
    t_vec   result;
    double  det;

    det = coeff0->x * coeff1->y - coeff1->x * coeff0->y;
    result.x = (coeff1->y * constant->x - coeff1->x * constant->y) / det;
    result.y = (coeff0->x * constant->y - coeff0->y * constant->x) / det;
    return (result);
}
```

det == 0이면 해가 없지만, 그런 경우에는 알아서 해가 nan으로 됨

역행렬을 통해 연립방정식을 푸는 코드를 짜봅시다. 별로 길지 않습니다.

수학적인 배경지식::라디안

라디안의 정의



$$1° = \frac{\pi}{180}$$

$$1(rad) = \frac{180}{\pi}$$

$$x° = \frac{x}{180}\pi(rad)$$

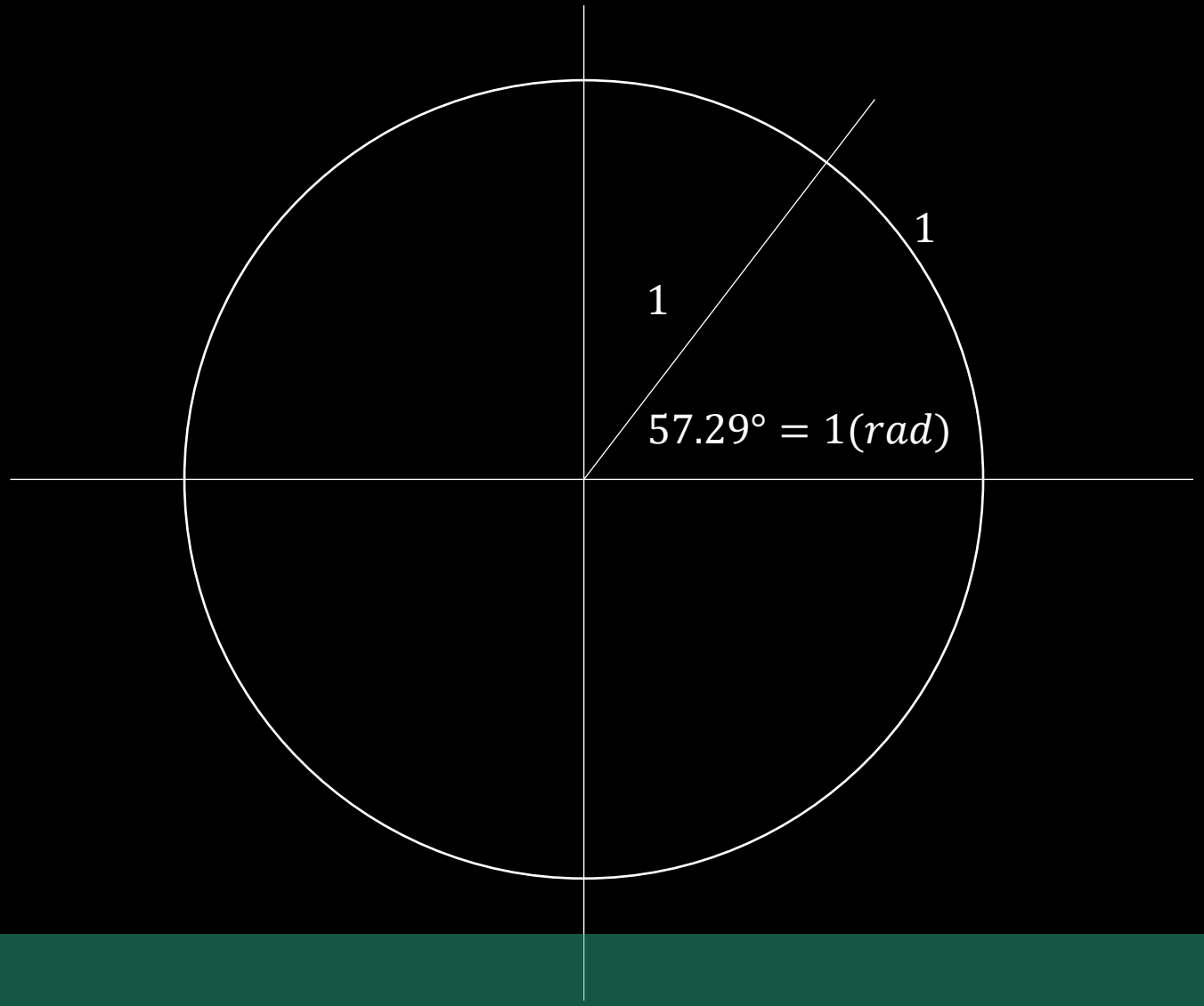
$$x(rad) = x\frac{180°}{\pi}$$

이번엔 라디안에 대해 알아봅시다. 우리가 과제를 할 때는 math.h의 sin함수나 cos함수를 써야 합니다. 이 함수의 인자의 단위는 도(degree)가 아니라 라디안 입니다.

라디안의 정의는 반지름의 길이와 호의 길이가 같을 때, 그때 호의 각도입니다.

수학적인 배경지식::라디안

자주 쓰이는 각도와 라디안의 관계



$$1° = \frac{\pi}{180}$$

$$30° = \frac{\pi}{6}$$

$$45° = \frac{\pi}{4}$$

$$60° = \frac{\pi}{3}$$

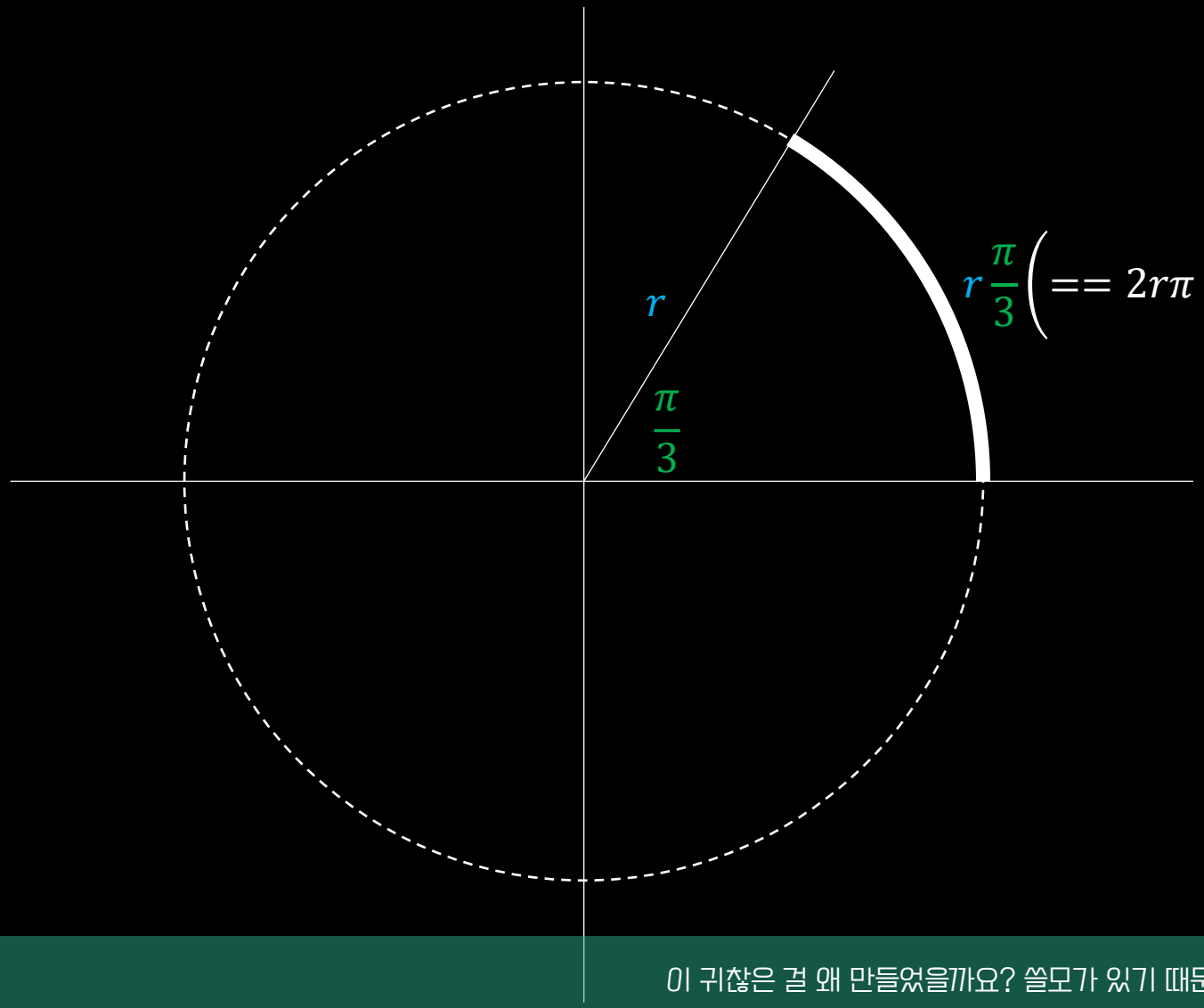
$$90° = \frac{\pi}{2}$$

$$180° = \pi$$

일상에서 자주 쓰이는 30도, 45도, 60도, 90도 같은 값이 라디안으로 몇인지 미리 외워두면 좋습니다.

수학적인 배경지식::라디안

라디안의 쓰임새



$$1° = \frac{\pi}{180}$$

$$30° = \frac{\pi}{6}$$

$$45° = \frac{\pi}{4}$$

$$60° = \frac{\pi}{3}$$

$$90° = \frac{\pi}{2}$$

$$180° = \pi$$

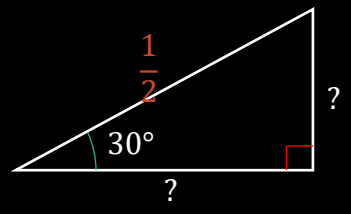
이 귀찮은 걸 왜 만들었을까요? 쓸모가 있기 때문입니다.

예를 들어 원의 호의 길이는 반지름 x 각(라디안) 으로 쉽게 구할 수 있습니다.

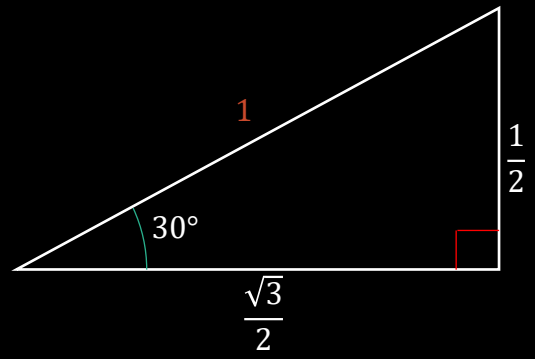
초딩때처럼 파이 곱하기 360분의 어쩌고 저쩌고 할 필요가 없죠.

수학적인 배경지식::삼각함수

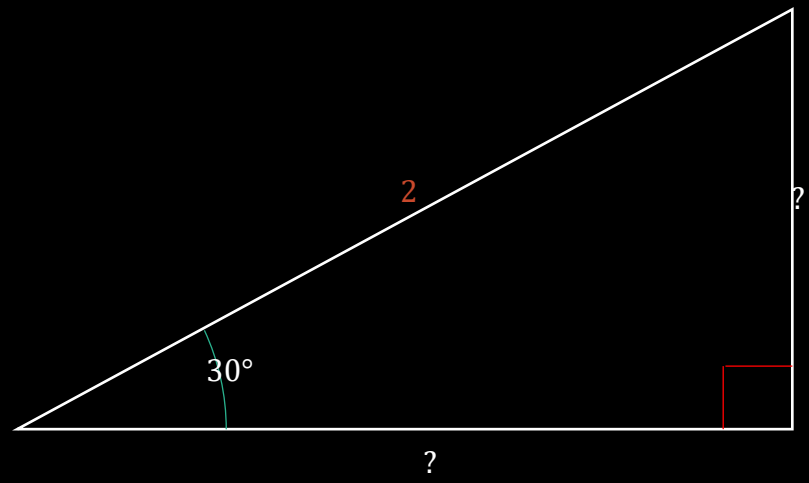
삼각함수의 쓰임새



[0]



[1]



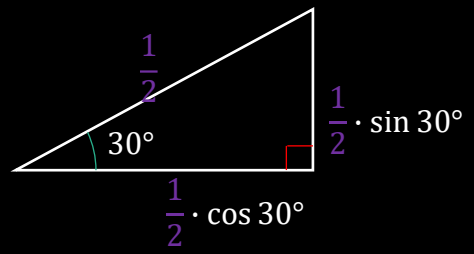
[2]

삼각형은 형태가 비슷비슷하게 생긴 친구들 끼리는 비율을 공유합니다.

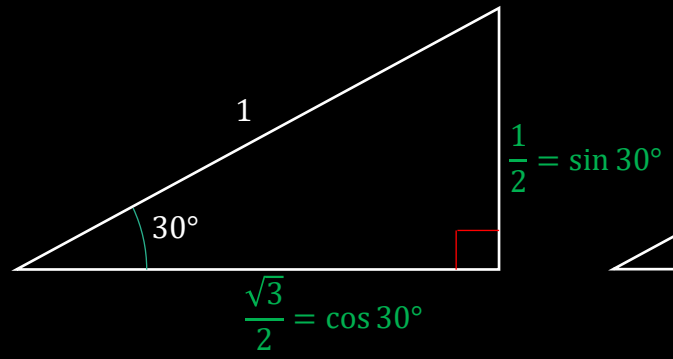
그런 성질을 이용하면, 삼각형 하나만 확실히 알아 두면 나머지 친구들의 신상정보를 쉽게 알아낼 수 있습니다.

수학적인 배경지식::삼각함수

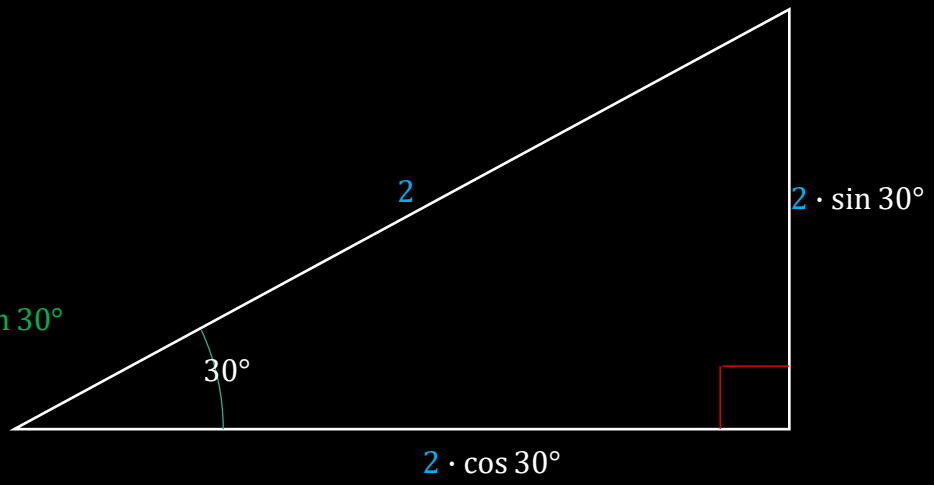
삼각함수의 쓰임새



[0]



[1]

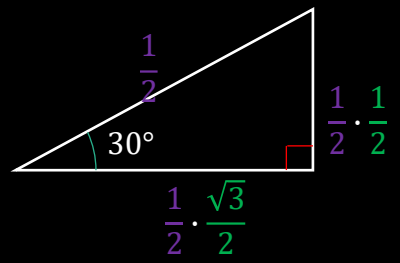


[2]

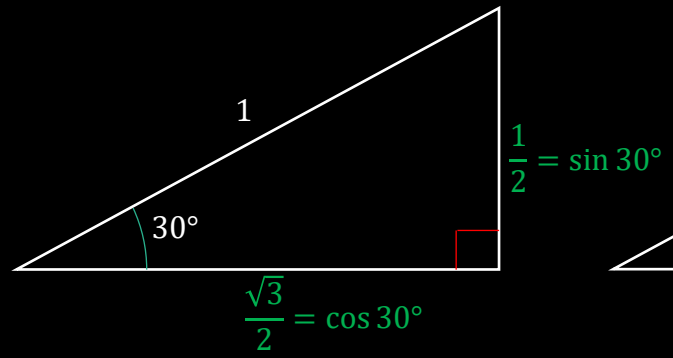
삼각비를 알고, 빗변의 길이를 알면 나머지는 미리 알아둔 비율을 곱해서 쉽게 알아낼 수 있습니다.

수학적인 배경지식::삼각함수

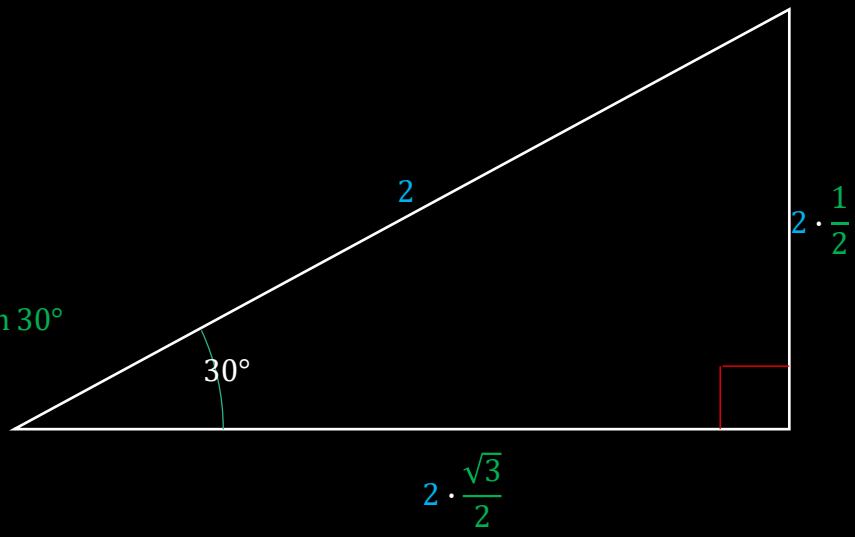
삼각함수의 쓰임새



[0]



[1]

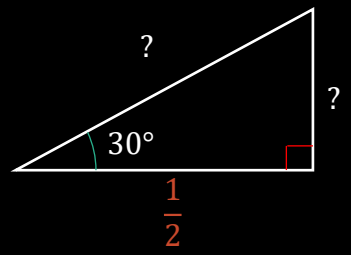


[2]

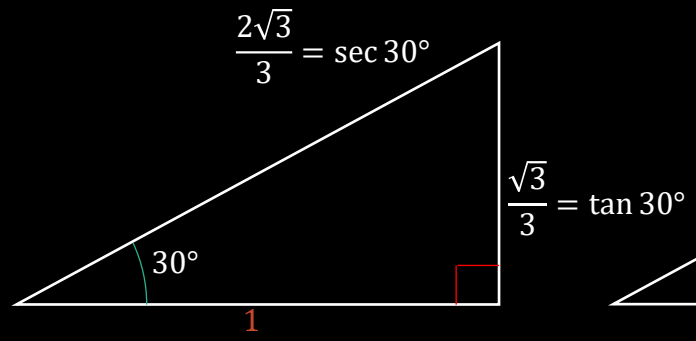
삼각비를 알고, 빗변의 길이를 알면 나머지는 미리 알아둔 비율을 곱해서 쉽게 알아낼 수 있습니다.

수학적인 배경지식::삼각함수

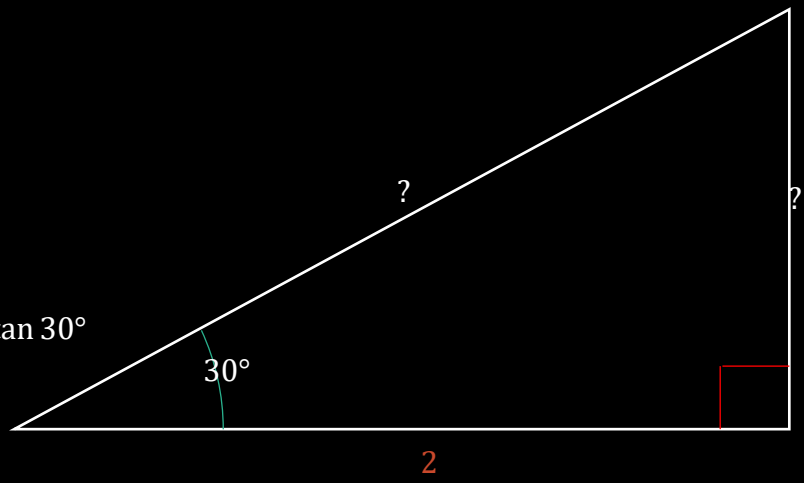
삼각함수의 쓰임새



[0]



[1]



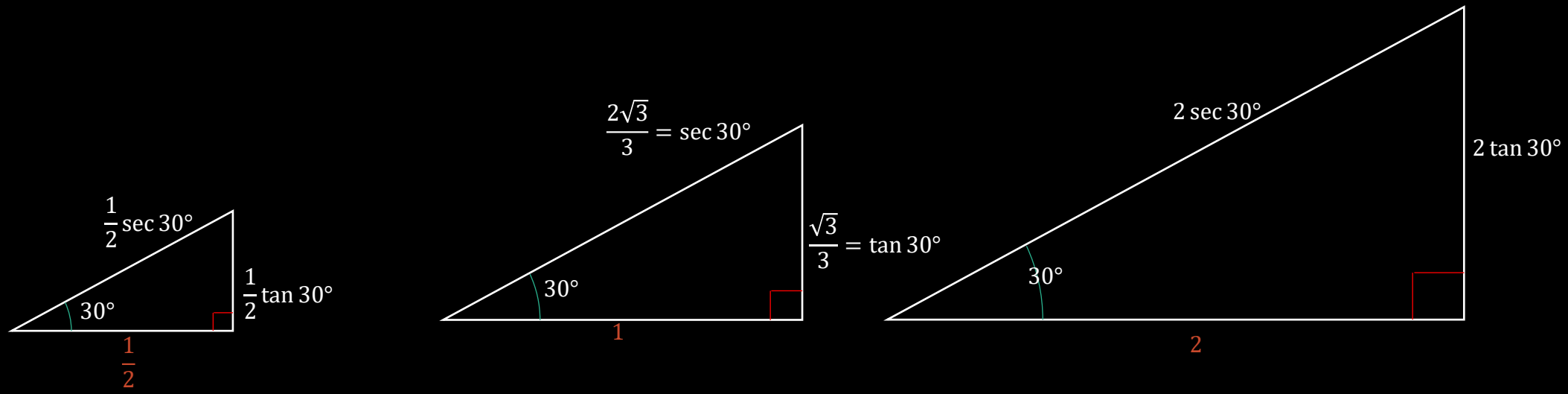
[2]

탄젠트는 높이/밑변 입니다.

마찬가지로 밑변의 길이를 알고 있을 때 높이 값을 구하기 좋습니다.

수학적인 배경지식::삼각함수

삼각함수의 쓰임새



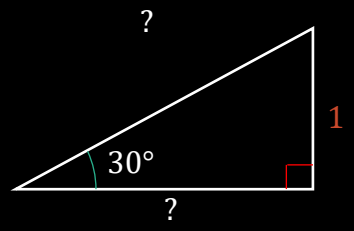
[0] **[1]** **[2]**

탄젠트는 높이/밑변 입니다.

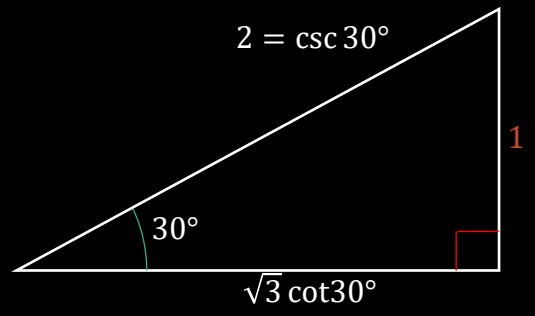
마찬가지로 밑변의 길이를 알고 있을 때 높이 값을 구하기 좋습니다.

수학적인 배경지식::삼각함수

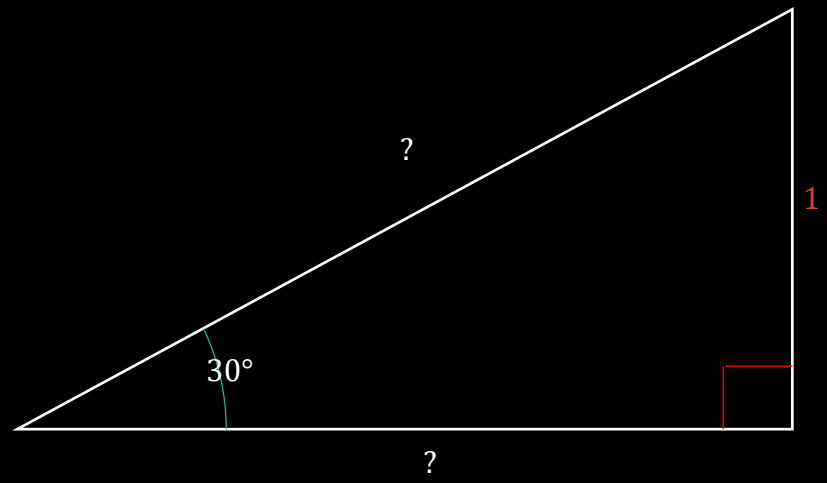
삼각함수의 쓰임새



[0]



[1]



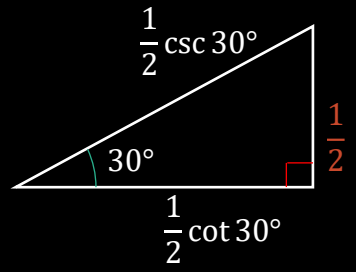
[2]

탄젠트는 높이/밑변 입니다.

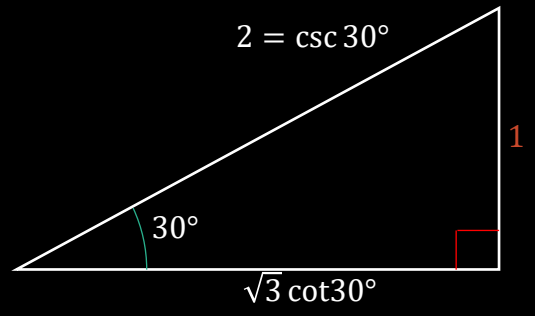
마찬가지로 밑변의 길이를 알고 있을 때 높이 값을 구하기 좋습니다.

수학적인 배경지식::삼각함수

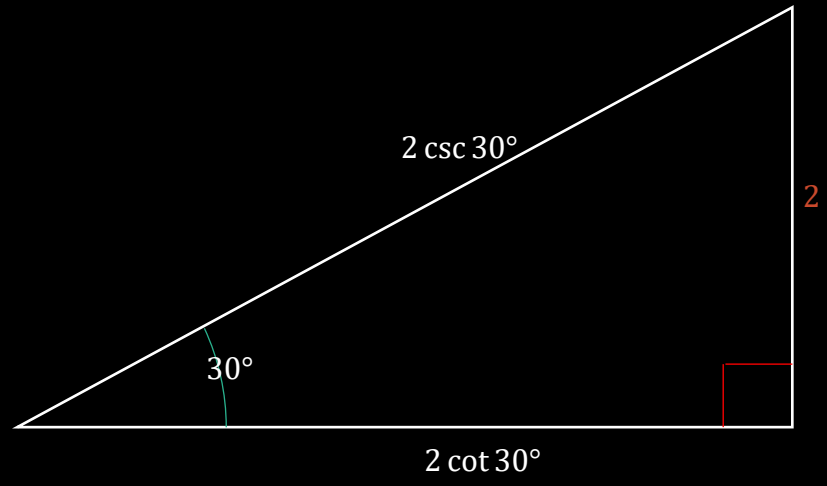
삼각함수의 쓰임새



[0]



[1]



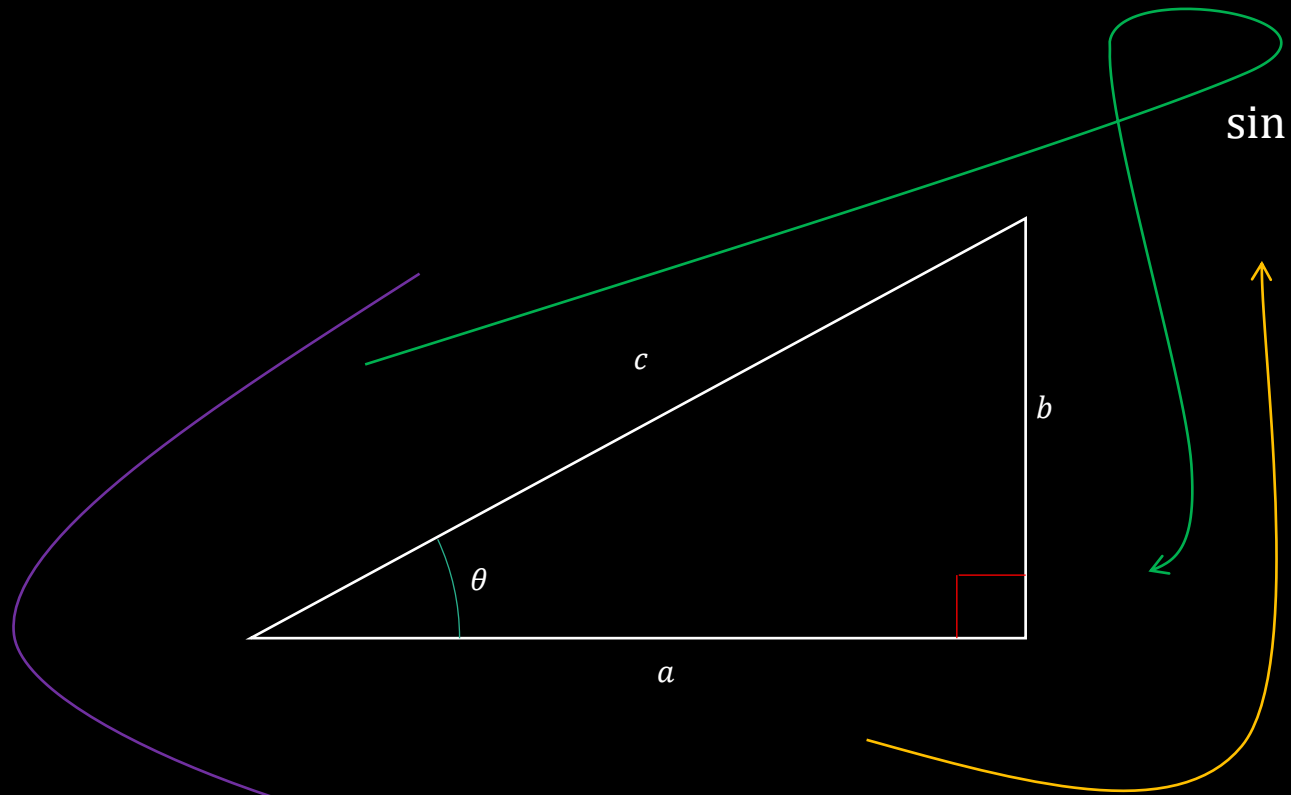
[2]

탄젠트는 높이/밑변 입니다.

마찬가지로 밑변의 길이를 알고 있을 때 높이 값을 구하기 좋습니다.

수학적인 배경지식::삼각함수

삼각함수 요약 정리



$$\sin\theta = \frac{b}{c} \qquad \csc\theta = \frac{1}{\sin\theta} = \frac{c}{b}$$

$$\cos\theta = \frac{a}{c}$$

$$\sec\theta = \frac{1}{\cos\theta}\frac{c}{b}$$

$$\tan\theta = \frac{b}{a}$$

$$\cot\theta = \frac{1}{\tan\theta} = \frac{a}{b}$$

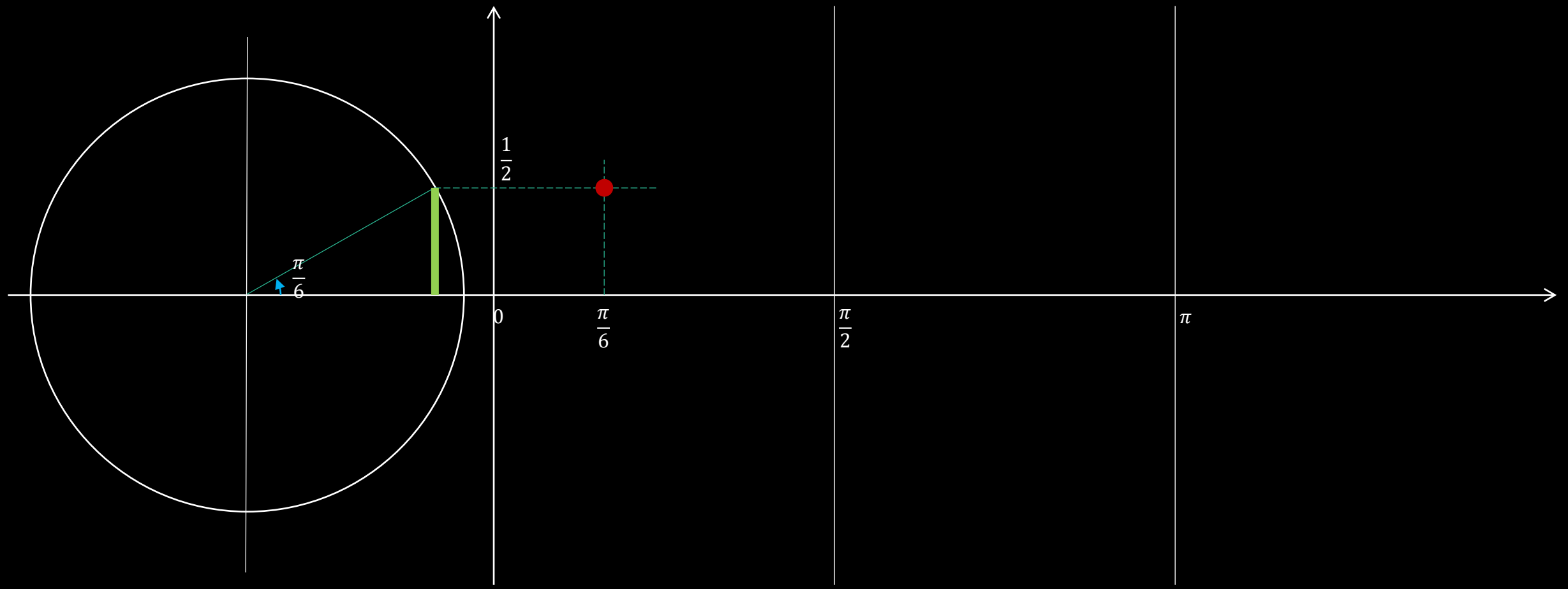
처음 보면(~~사실 여러분들 모두 중학교 3학년때 배웠습니다 읍읍~~) 무엇이 사인이고 무엇이 코사인인지 헷갈릴 텐데,

사인은 s모양, 코사인은 c모양, 탄젠트는 t~~가 되려다 만~~ 모양으로 외울 수 있습니다.

그리고 각각 역수를 취한 것들이 있습니다. 사인은 코시컨트, 코사인은 시컨트, 탄젠트는 코탄젠트 입니다.

수학적인 배경지식::삼각함수

사인함수 그래프



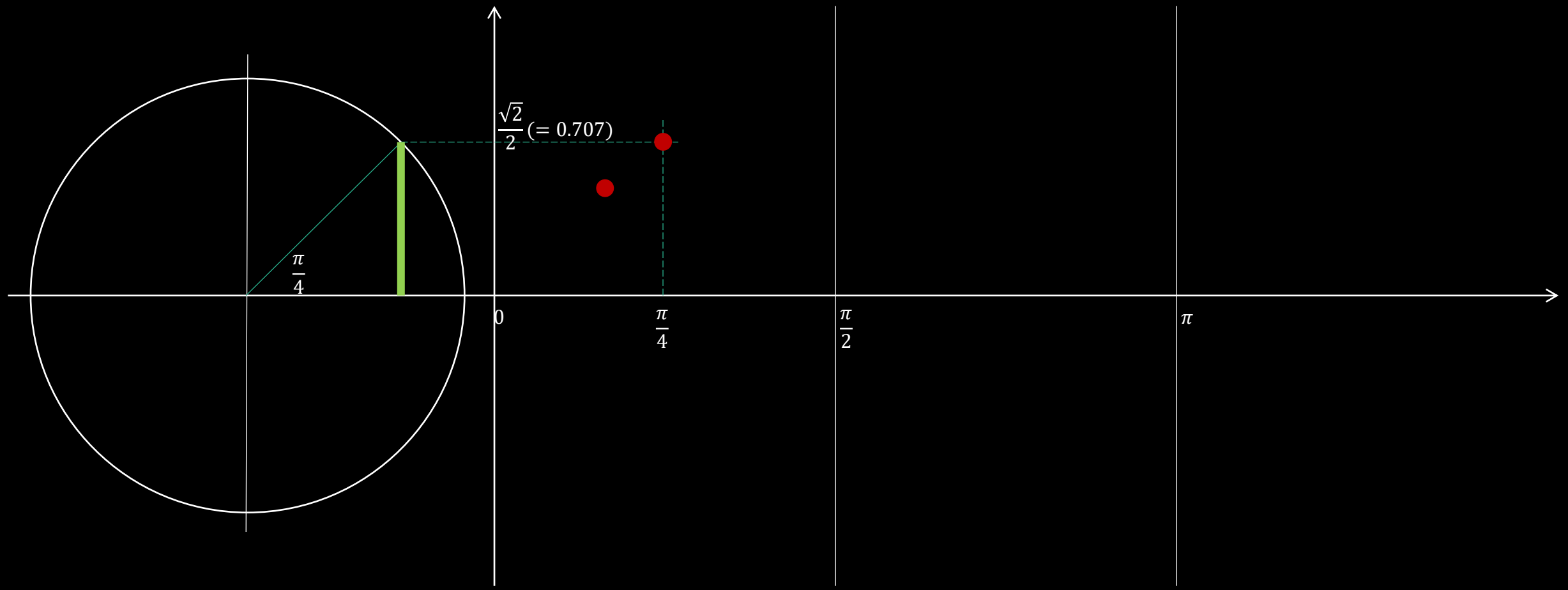
그래서 빗변의 길이가 1인 삼각형을 기준으로 sin값을 계산해두면 다른 모든 삼각형에도 써먹을 수 있습니다.

각도(입력)에 대응하는 sin값(반환값)이 있기때문에 sin도 곧 함수입니다.

그 값을 추적해서 그래프를 그려보면 이런 식입니다.

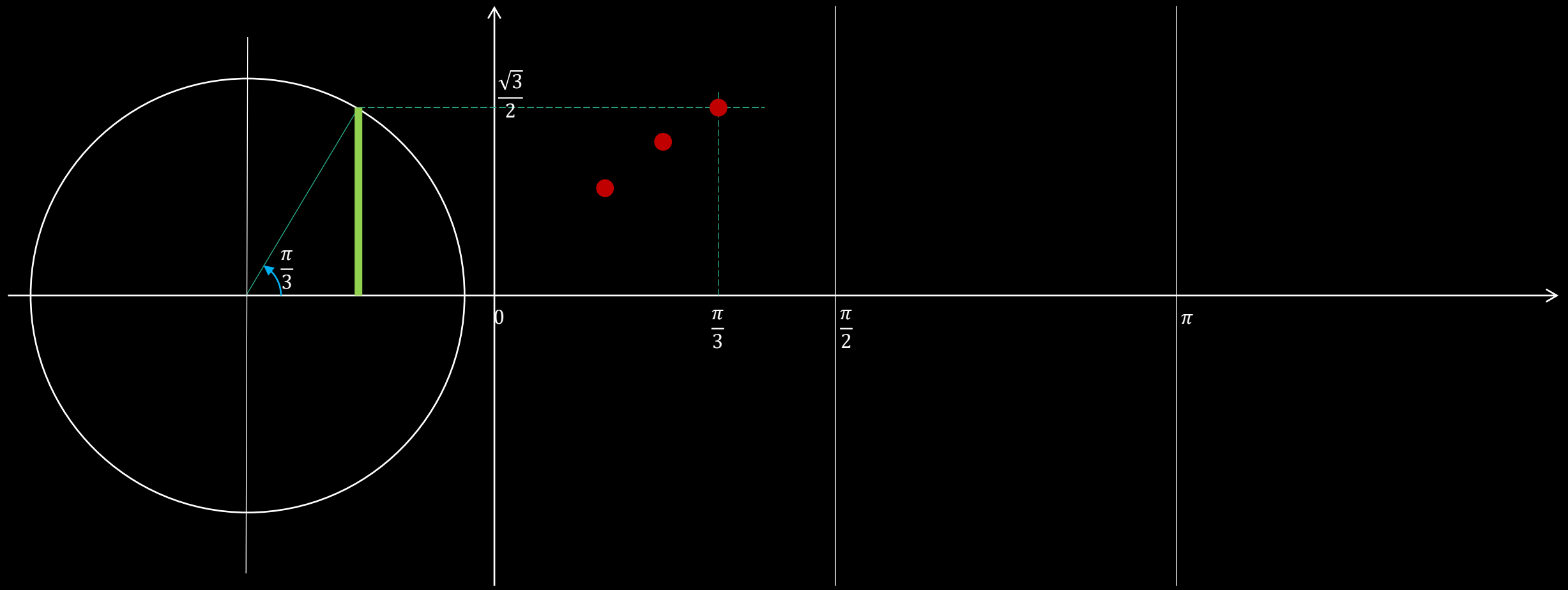
수학적인 배경지식::삼각함수

사인함수 그래프



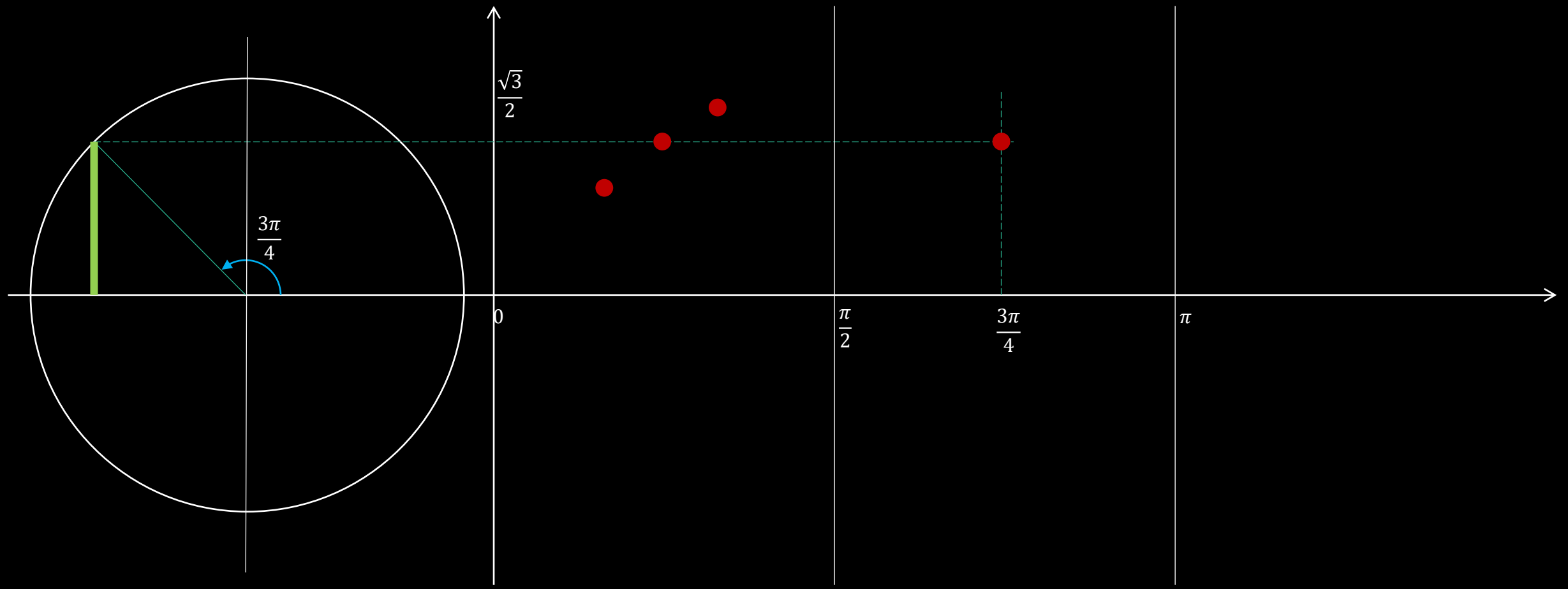
수학적인 배경지식::삼각함수

사인함수 그래프



수학적인 배경지식::삼각함수

사인함수 그래프

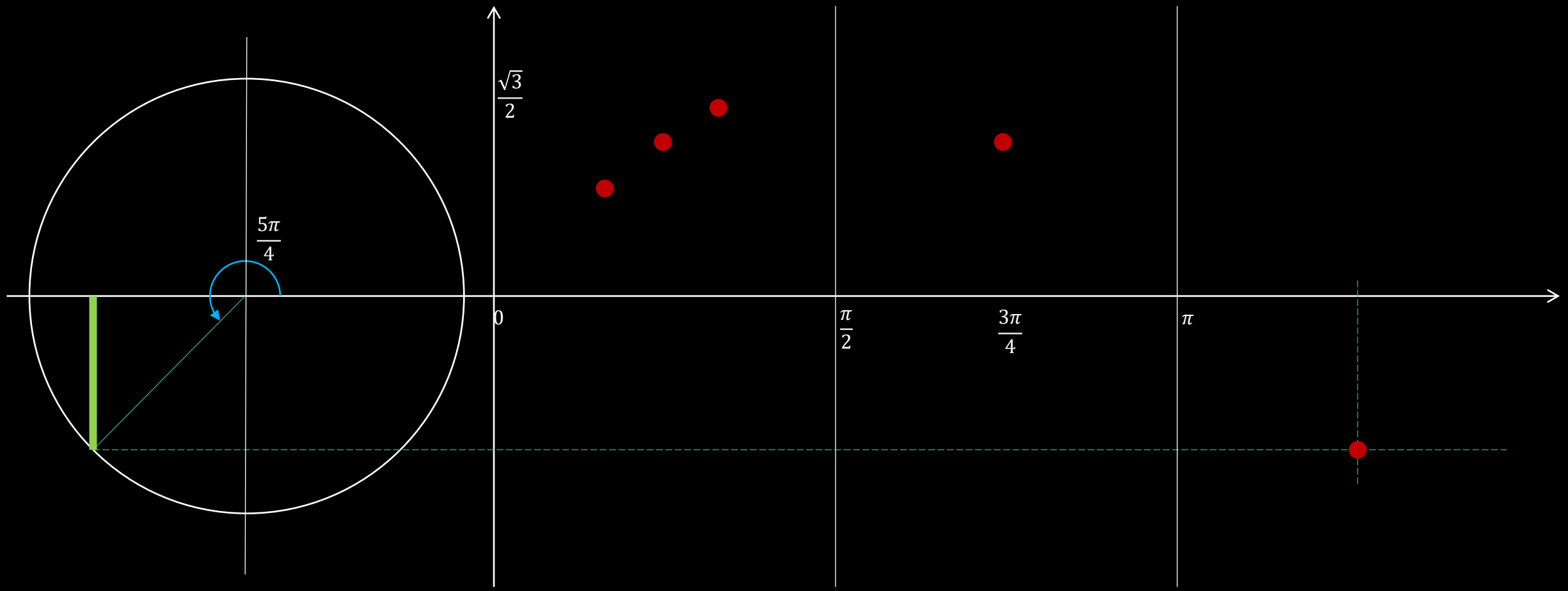


상식 밖인 것 같지만, 90도를 넘는 삼각형에 대해서도 sine값이 정의됩니다.

이게 말이 되느냐 싶겠지만, 정의하기 나름입니다. 예외가 적을수록 쓸모가 많기 때문이죠.

수학적인 배경지식::삼각함수

사인함수 그래프

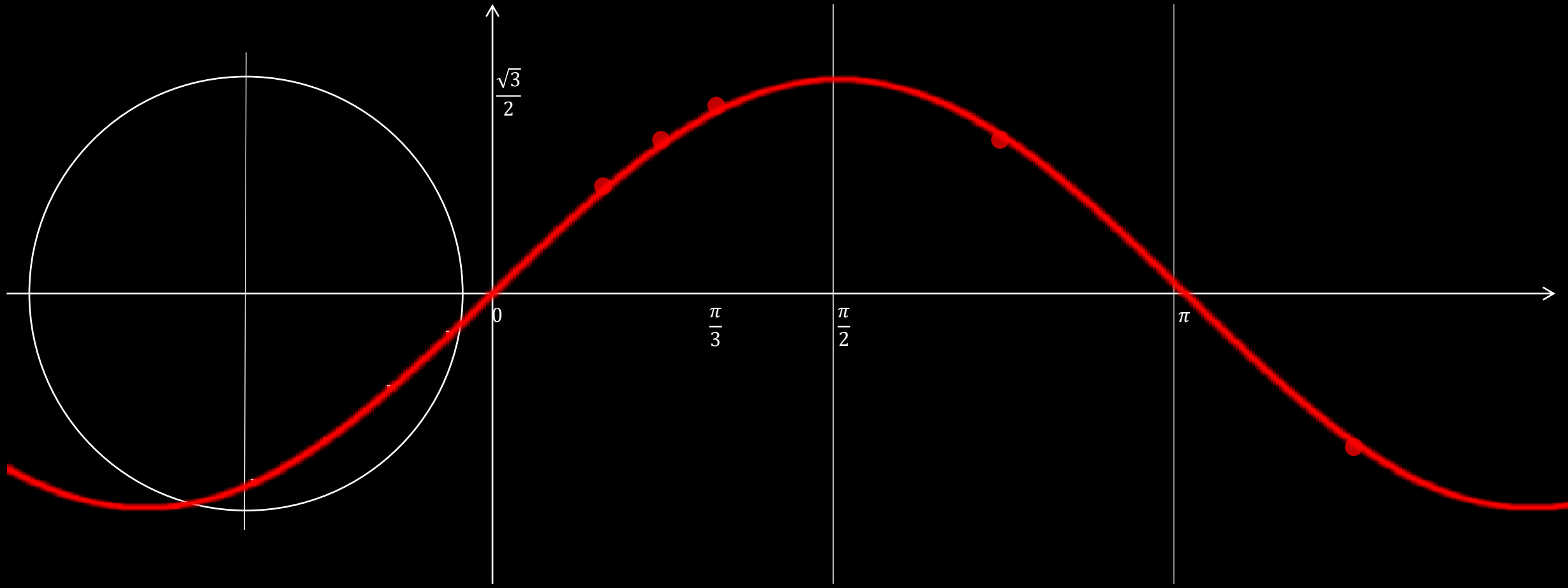


심지어 180도를 넘어가기도 합니다.

sine의 값은 y좌표로 하기로 합니다.

수학적인 배경지식::삼각함수

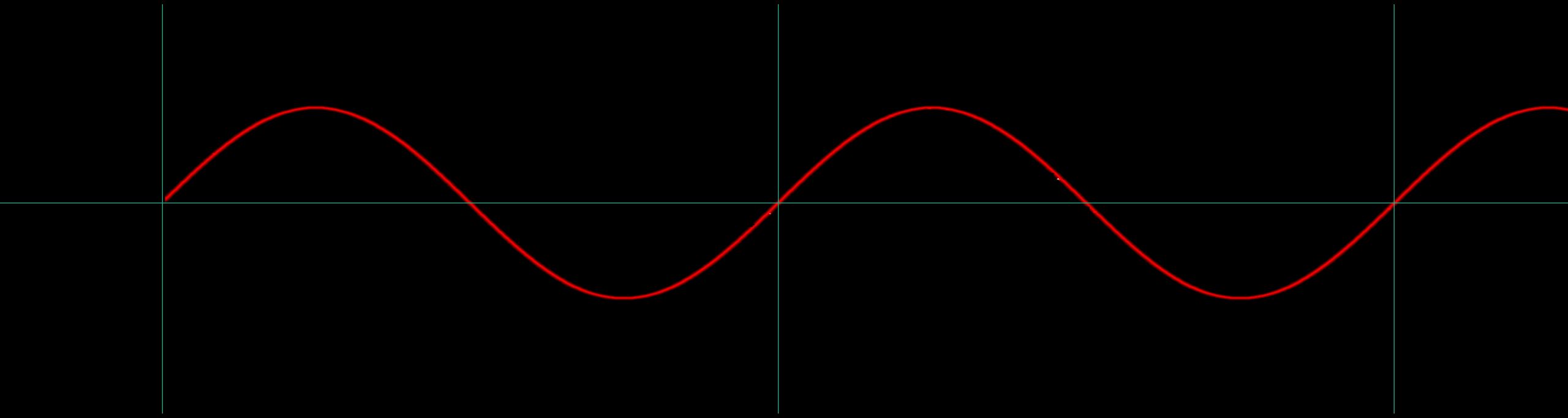
사인함수 그래프



함수를 세밀하게 이어주면 이런 형태가 됩니다.

수학적인 배경지식::삼각함수

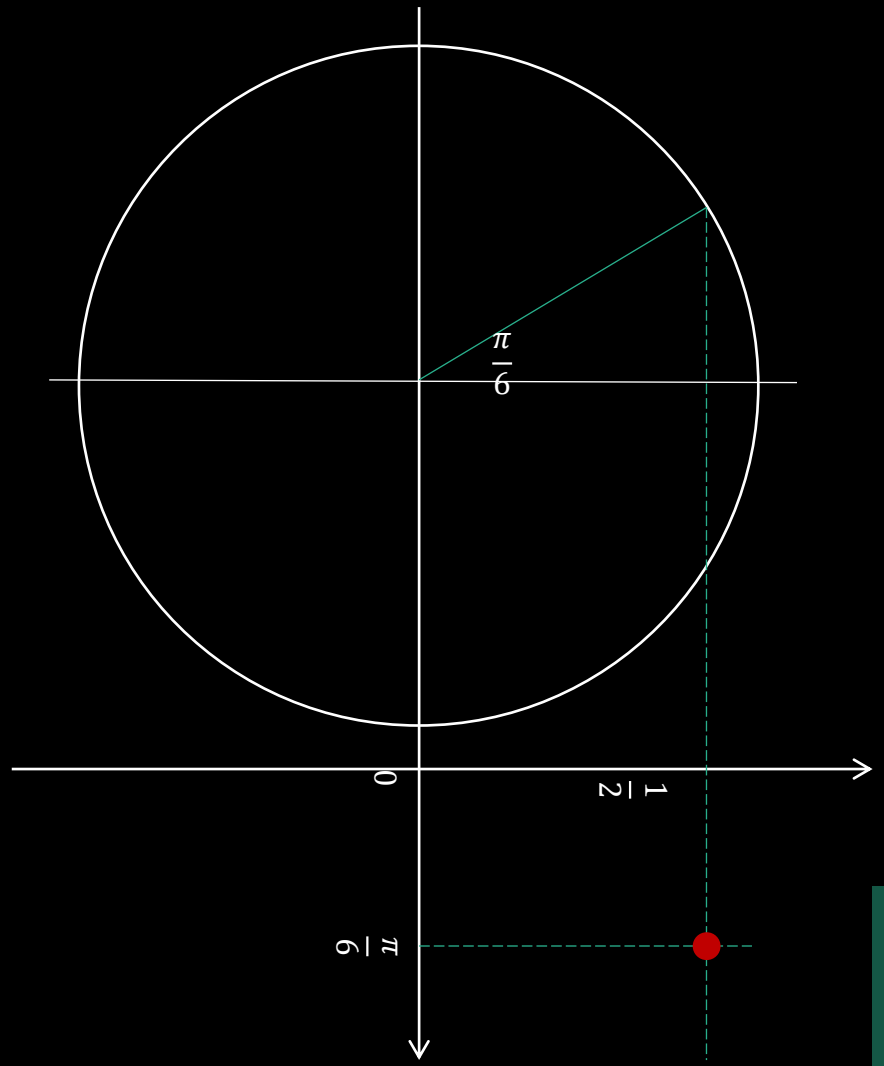
사인함수 그래프



$2\pi(== 360°)$ 주기로 반복되는 함수입니다.

수학적인 배경지식::삼각함수

코사인함수 그래프

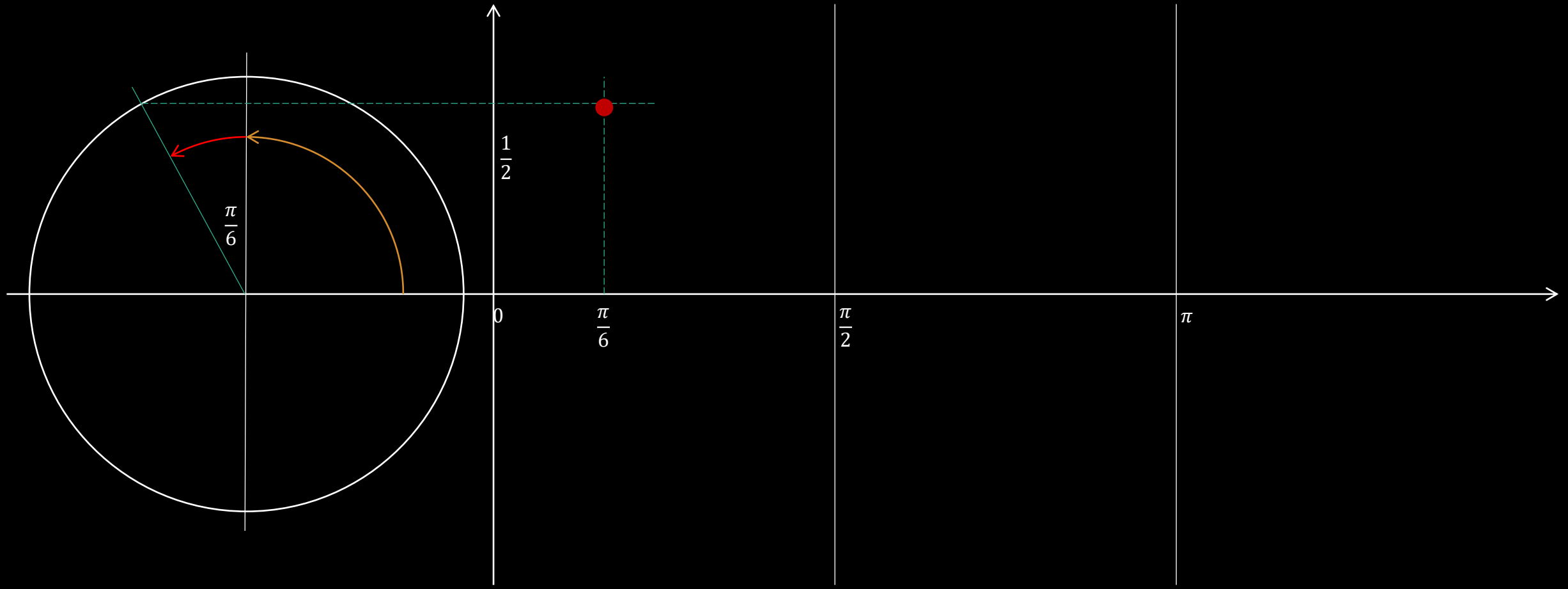


cosine 함수는 코사인의 정의를 따르자면 밑변의 길이를 재면 됩니다.

확장해서 생각해보자면, sine 함수에서와 비슷하게 x값 자체를 반환값으로 하면 좋을 것 같습니다.

수학적인 배경지식::삼각함수

코사인함수 그래프

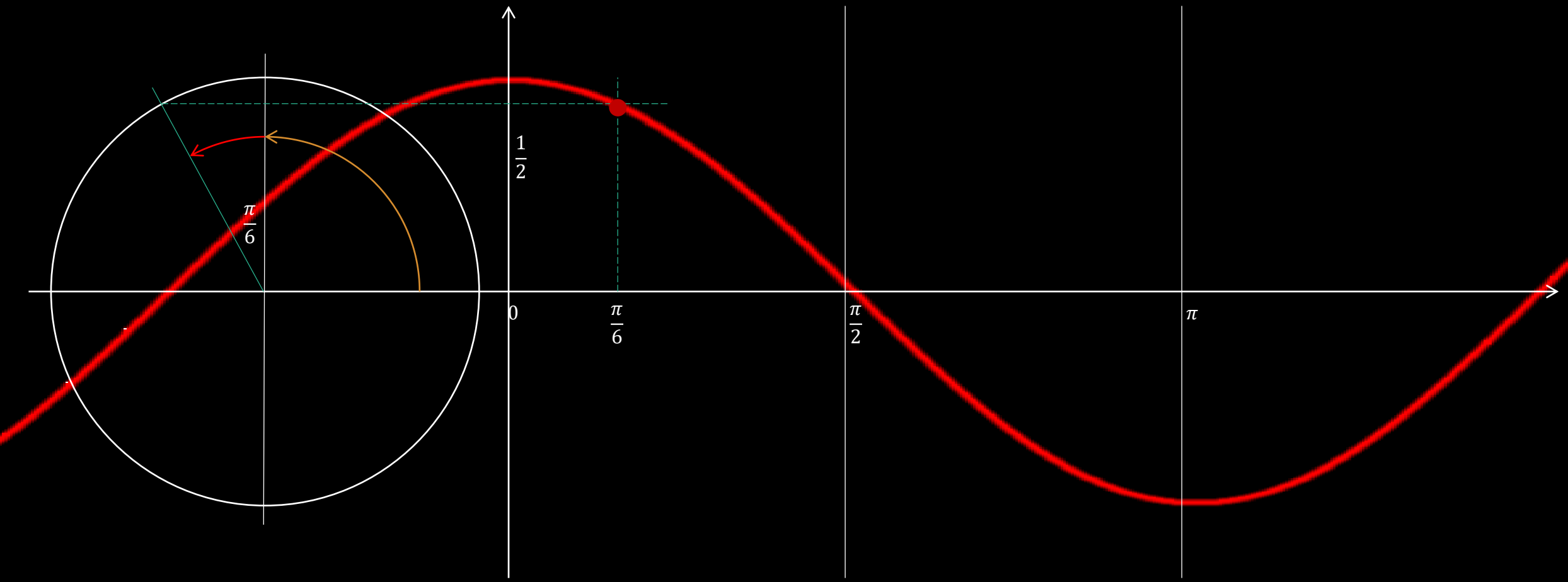


그래프가 아래로 향하는 것이 표현하기 어렵기 때문에 반시계방향으로 90도 돌려봅시다.

그러고 나면 사인함수랑 똑같습니다. 다만 이미 90인 상태에서 시작한 것일 뿐.

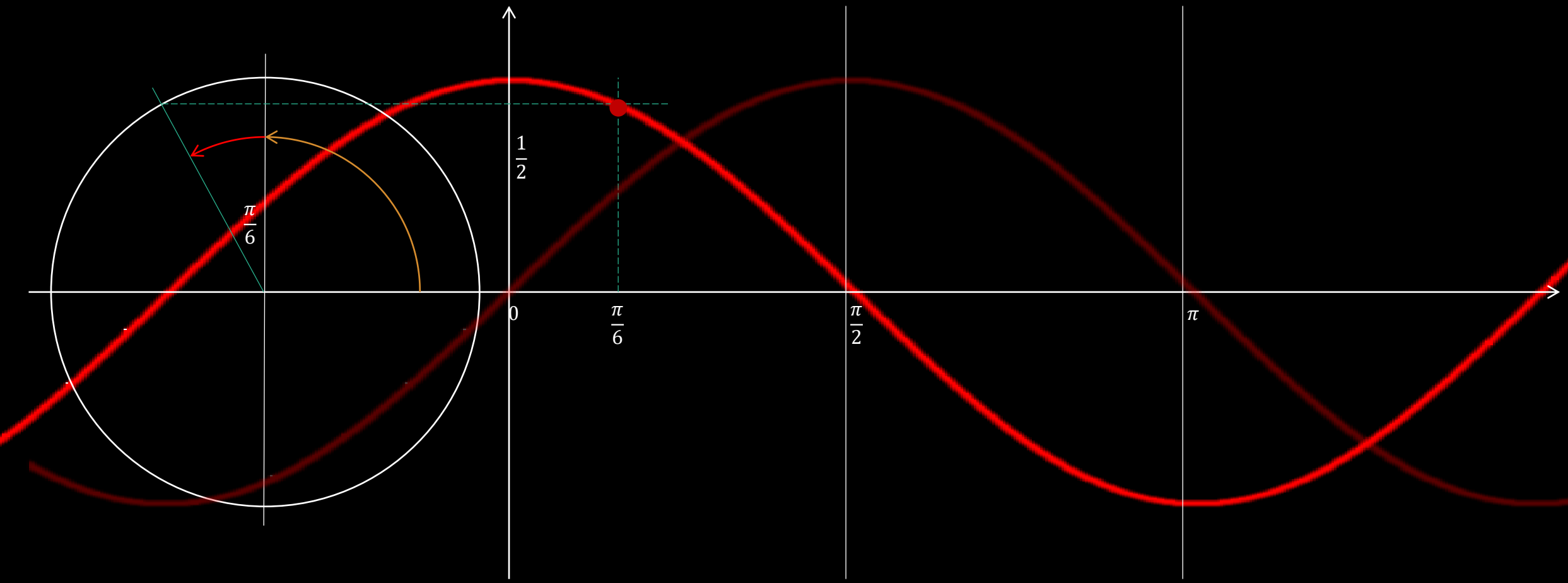
수학적인 배경지식::삼각함수

코사인함수 그래프



수학적인 배경지식::삼각함수

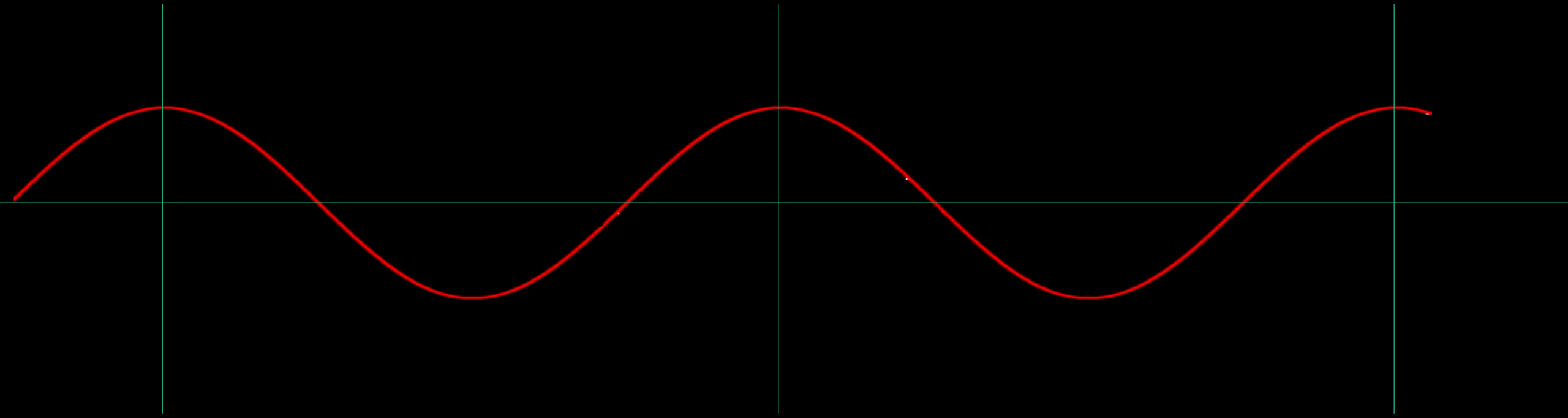
코사인함수 그래프



사인함수와 90도 차이 납니다.

수학적인 배경지식::삼각함수

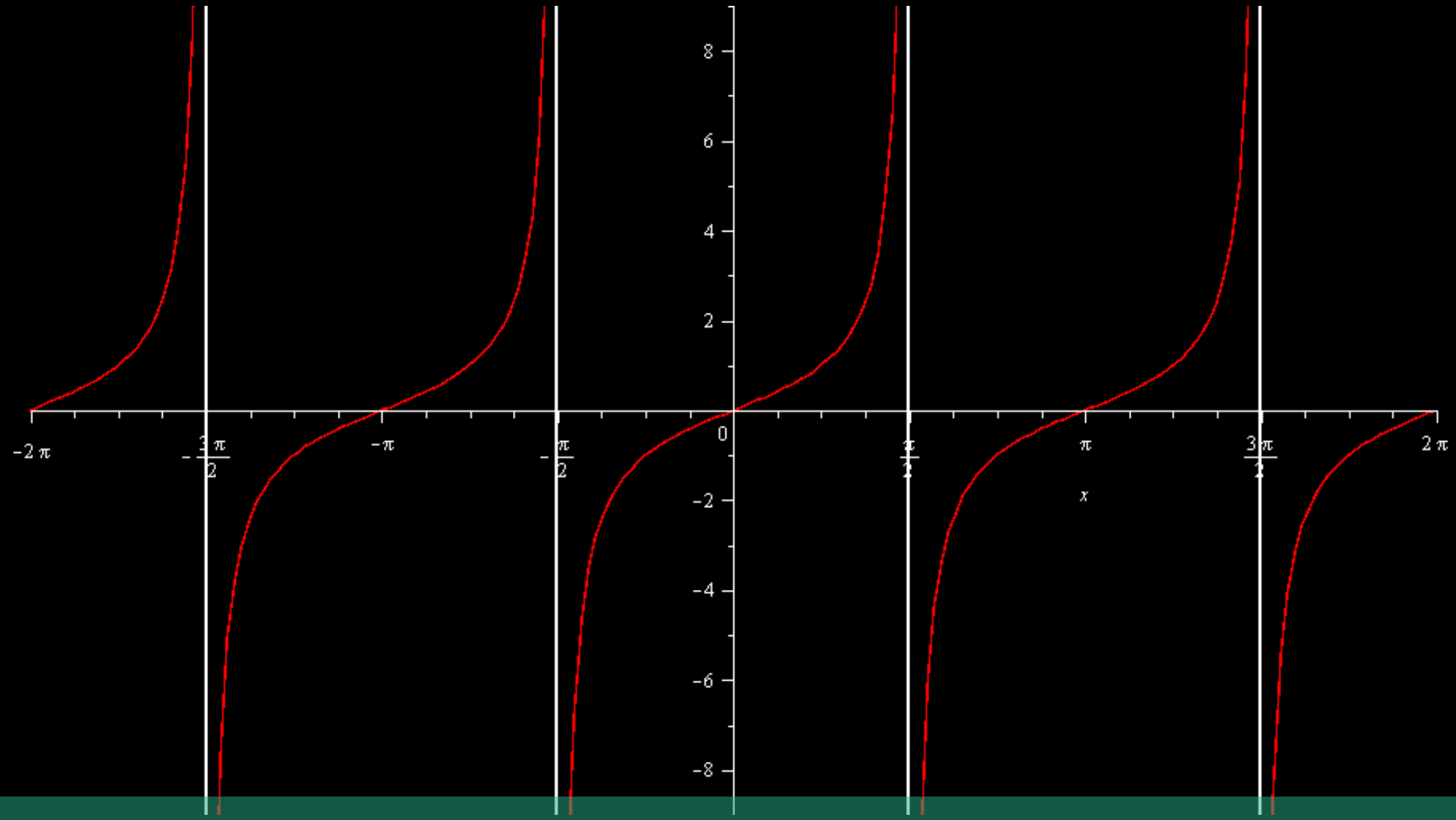
코사인함수 그래프



코사인 함수도 마찬가지로 $2\pi(== 360°)$ 주기로 반복됩니다.

수학적인 배경지식::삼각함수

탄젠트 그래프

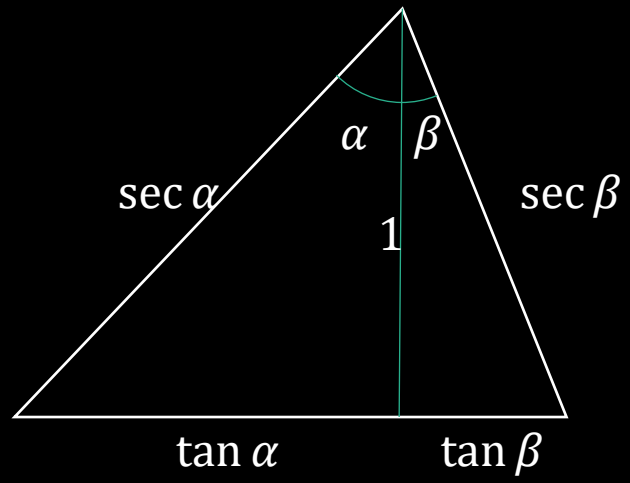


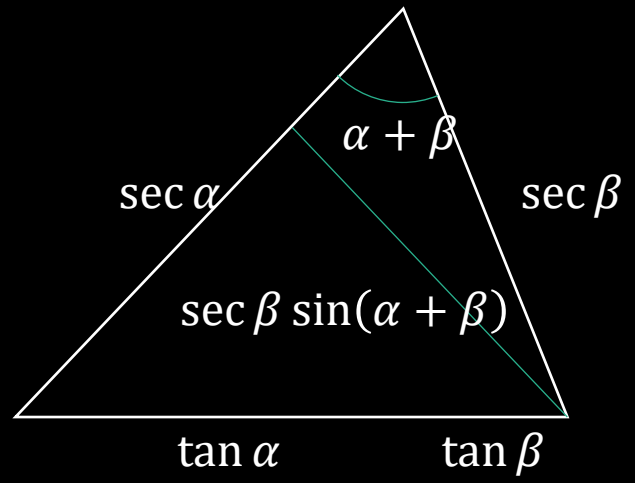
탄젠트 함수는 이렇게 생겼고, $\pi(== 180°)$를 주기로 반복됩니다.

수학적인 배경지식::삼각함수

삼각함수의 덧셈/뺄셈

$$\sin(\alpha+\beta)=\sin\alpha\cos\beta+\cos\alpha\sin\beta$$
$$\cos(\alpha+\beta)=\cos\alpha\cos\beta-\sin\alpha\sin\beta$$





$$(\tan\alpha+\tan\beta)=\sec\alpha\sec\beta\sin(\alpha+\beta)$$

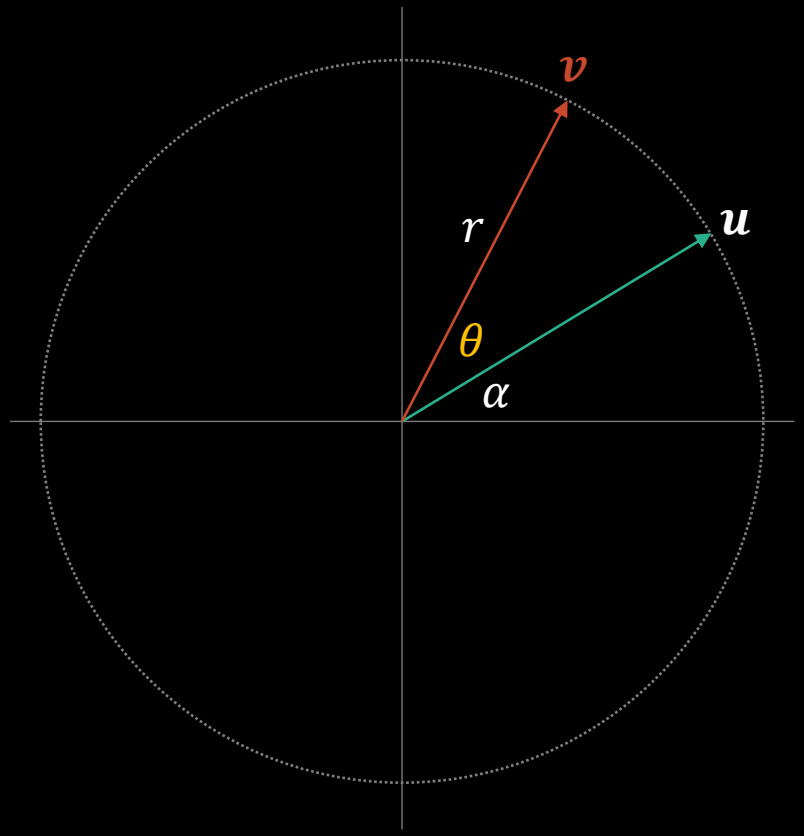
$$\cos\alpha\cos\beta\left(\frac{\sin\alpha}{\cos\alpha}+\frac{\sin\beta}{\cos\beta}\right)=\sin(\alpha+\beta)$$

$$\sin(\alpha+\beta)=\sin\alpha\cos\beta+\cos\alpha\sin\beta$$

삼각함수를 알아봤으니, 이번에는 덧셈 뺄셈에 대해 알아봅시다. $\sin(\alpha+\beta)\neq\sin\alpha+\sin\beta$ 입니다.

수학적인 배경지식::회전

벡터를 회전시키기



플레이어가 현재 바라보고 있는 방향은 u이다

플레이어가 보는 방향을 θ만큼 반시계방향으로 돌린다

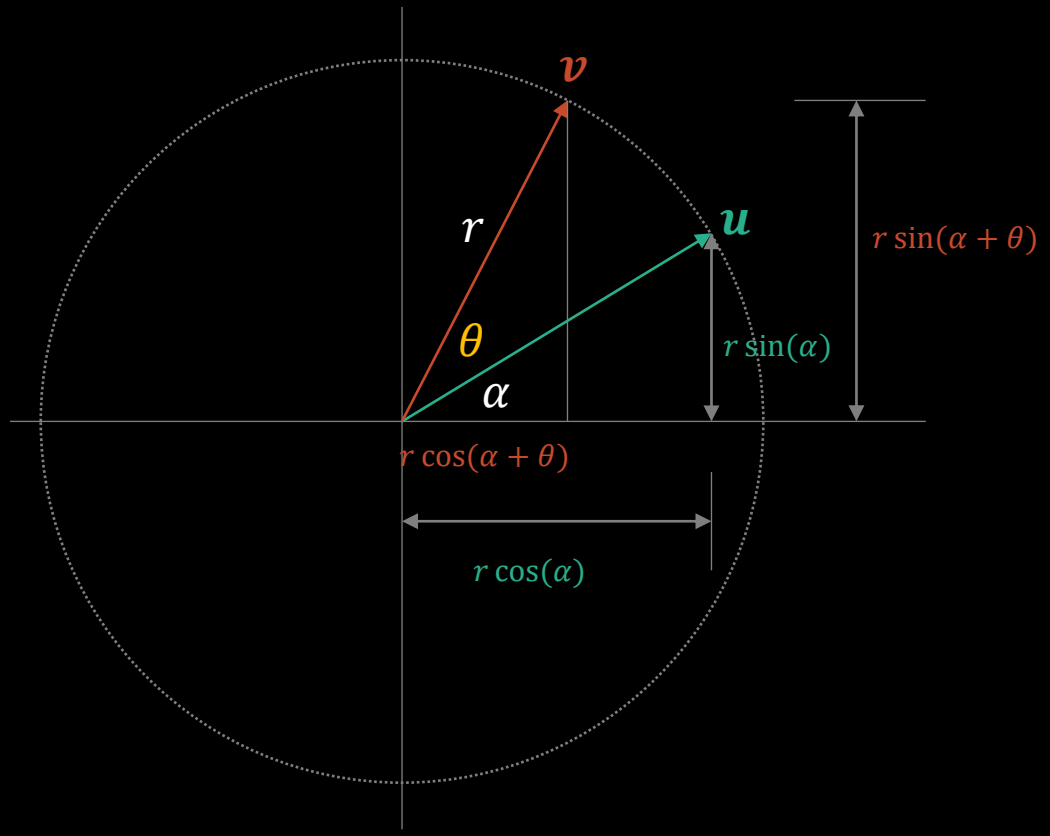
플레이어가 바라보는 방향의 벡터 v 를 직교좌표계로 표현하면?

이렇게 힘겹게 삼각함수를 공부한 이유는 벡터를 회전시키는 데에 쓸모가 있기 때문입니다.

수학적인 배경지식::회전

벡터를 회전시키기

$$\sin(\alpha+\beta)=\sin\alpha\cos\beta+\cos\alpha\sin\beta$$
$$\cos(\alpha+\beta)=\cos\alpha\cos\beta-\sin\alpha\sin\beta$$



$$\boldsymbol{v}=\begin{bmatrix}v_x\\v_y\end{bmatrix}=\begin{bmatrix}r\cos(\alpha+\theta)\\r\sin(\alpha+\theta)\end{bmatrix}$$

$$=r\begin{bmatrix}\cos\alpha\cos\theta-\sin\alpha\sin\theta\\\sin\alpha\cos\theta+\cos\alpha\sin\theta\end{bmatrix}$$

$$=r\begin{bmatrix}\cos\theta&-\sin\theta\\\sin\theta&\cos\theta\end{bmatrix}\begin{bmatrix}\cos\alpha\\\sin\alpha\end{bmatrix}$$

$$=\begin{bmatrix}\cos\theta&-\sin\theta\\\sin\theta&\cos\theta\end{bmatrix}\cdot r\begin{bmatrix}\cos\alpha\\\sin\alpha\end{bmatrix}$$

$$=\begin{bmatrix}\cos\theta&-\sin\theta\\\sin\theta&\cos\theta\end{bmatrix}\boldsymbol{u}$$

$$\boldsymbol{R}=\begin{bmatrix}\cos\theta&-\sin\theta\\\sin\theta&\cos\theta\end{bmatrix}$$

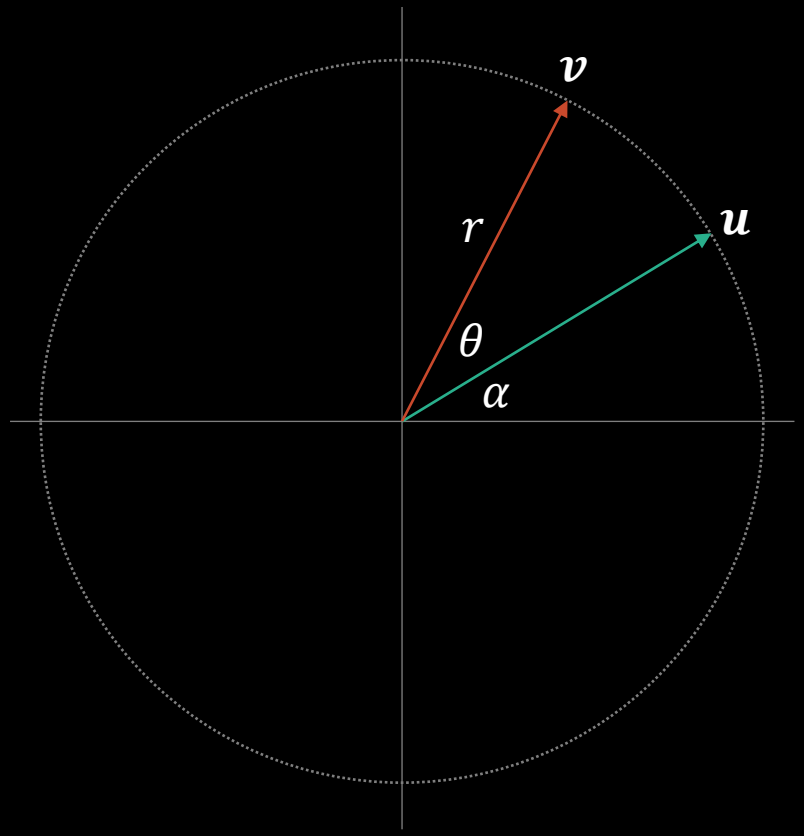
$$\boldsymbol{R}^{-1}=\boldsymbol{R}^{T}$$

$$\boldsymbol{v}=\boldsymbol{R}\boldsymbol{u}$$

벡터u를 θ만큼 회전시켜서 얻는 벡터 v의 값은, θ에 대응하는 회전행렬R을 u에 곱해줌으로써 얻을 수 있습니다.

수학적인 배경지식::회전

벡터를 회전시키기



플레이어가 현재 바라보고 있는 방향은 u이다

플레이어가 보는 방향을 θ만큼 반시계방향으로 돌린다

플레이어가 바라보는 방향의 벡터 v 를 직교좌표계로 표현하면?

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

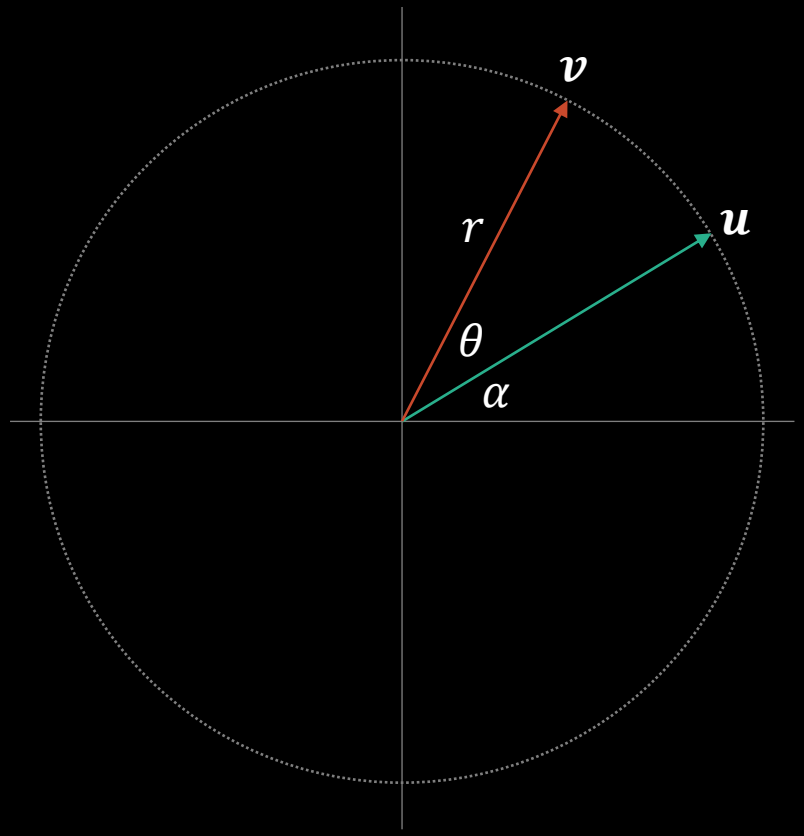
$$R^{-1} = R^{T}$$

$$v = Ru$$

사용례는 다음과 같습니다.

수학적인 배경지식::회전

벡터를 회전시키기



$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

$$R^{-1} = R^{T}$$

$$v = Ru$$

u θ

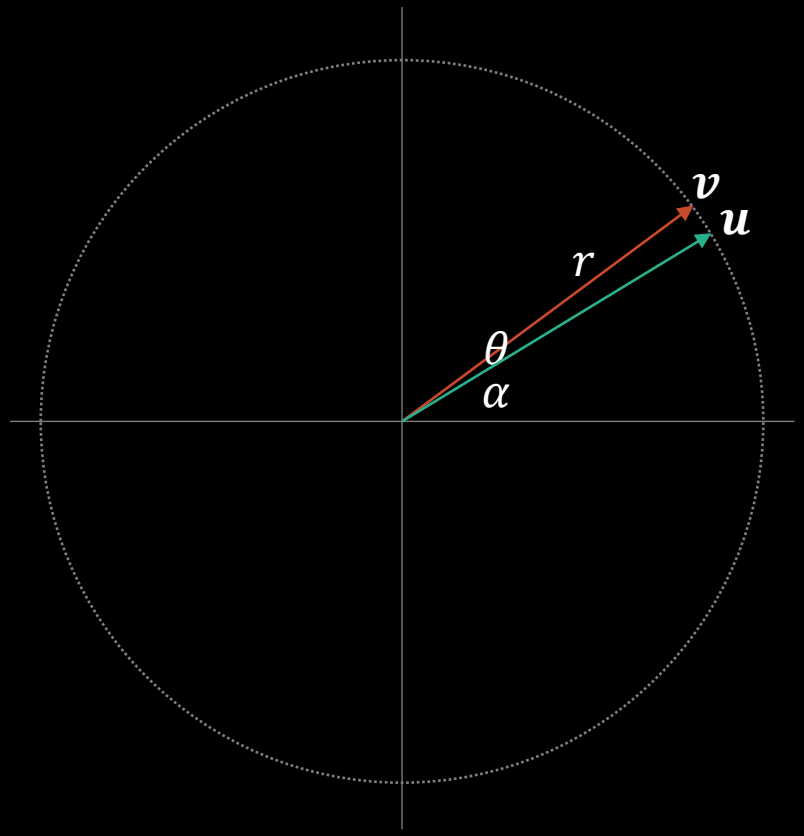
```
t_vec   vec_rot_ccw(t_vec a, double angle)
{
    double  sin_angle;
    double  cos_angle;
    t_vec   result;

    sin_angle = sin(angle);
    cos_angle = cos(angle);
    result.x = cos_angle * a.x - sin_angle * a.y;
    result.y = sin_angle * a.x + cos_angle * a.y;
    return (result);
}
```

v

이것을 토대로 함수를 만들어 봅시다.

수학적인 배경지식::회전

벡터를 회전시키기



$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

$$R^{-1} = R^{T}$$

$$v = Ru$$

u θ

```
t_vec   vec_rot_min_ccw(t_vec a)
{
    static double   sin_unit;
    static double   cos_unit;
    t_vec           result;

    sin_unit = sin_unit ? sin_unit : sin(M_PI * ANGLE_MIN / 180);
    cos_unit = cos_unit ? cos_unit : cos(M_PI * ANGLE_MIN / 180);
    result.x = cos_unit * a.x - sin_unit * a.y;
    result.y = sin_unit * a.x + cos_unit * a.y;
    return (result);
}
```

v

그런데 어차피 시야각이 급변할 일이 없기 때문에, 이렇게 미소한 각도에 대해서는 계산을 단순화해두면 더 좋을 수도 있습니다.

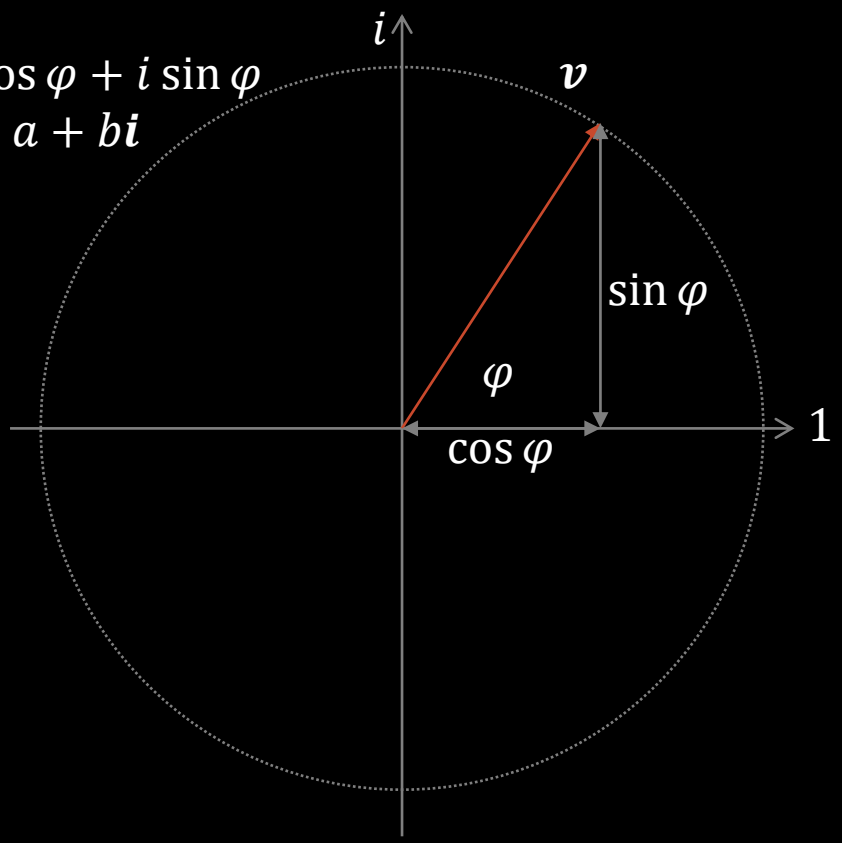
(그런데 전체 연산에 비해 회전하는 연산이 차지하는 비중이 크지 않으므로 효과가 미비할 것입니다.)

수학적인 배경지식::회전

사원수::복소수를 이용한 회전

복소수

$$e^{i\varphi} = \cos\varphi + i\sin\varphi$$
$$= a + b\boldsymbol{i}$$



새로운 수 체계

$$q = w + x\boldsymbol{i} + y\boldsymbol{j} + z\boldsymbol{k}$$

$$i^2 = j^2 = k^2 = ijk = -1$$

$$i \neq j$$

$$j \neq k$$

$$k \neq i$$

$$jk = -kj = i$$

$$ki = -ik = j$$

$$ij = -ji = k$$

~~근본이 없네~~

회전행렬을 이용한 회전은 이해하기 쉽긴 한데, 3차원에서는 가끔 말썽을 일으킵니다. (#짐벌락)

다행히도 우리는 2차원만 다루면 돼서 이런 일을 겪지는 않습니다. "이드소프트웨어"에서는 이걸 썼다고 합니다.

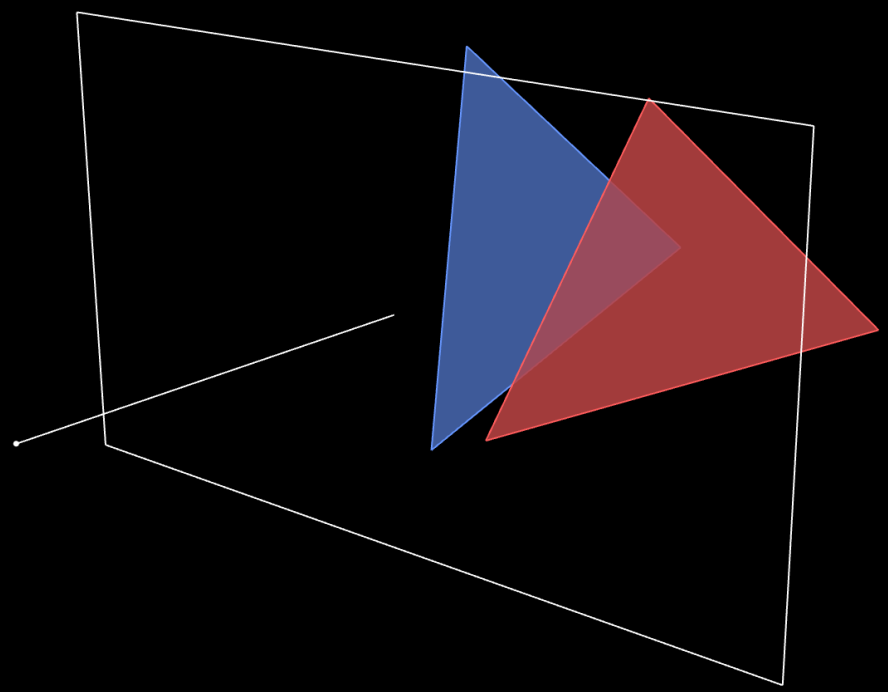
복소수를 이용한 회전은 이런 문제에서 자유롭습니다. 관심있는 분들은 찾아보시기를 바랍니다.

레이캐스팅 원리 이해하기

레이캐스팅 맛보기

레이캐스팅 원리 이해하기

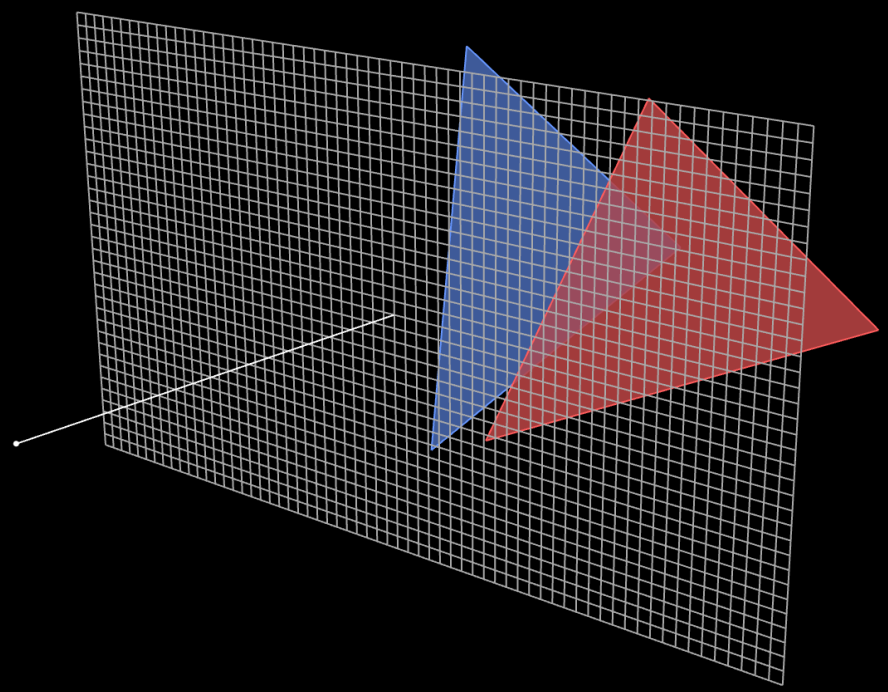
레이캐스팅



가상의 화면과 삼각형 두 개가 있습니다.

레이캐스팅 원리 이해하기

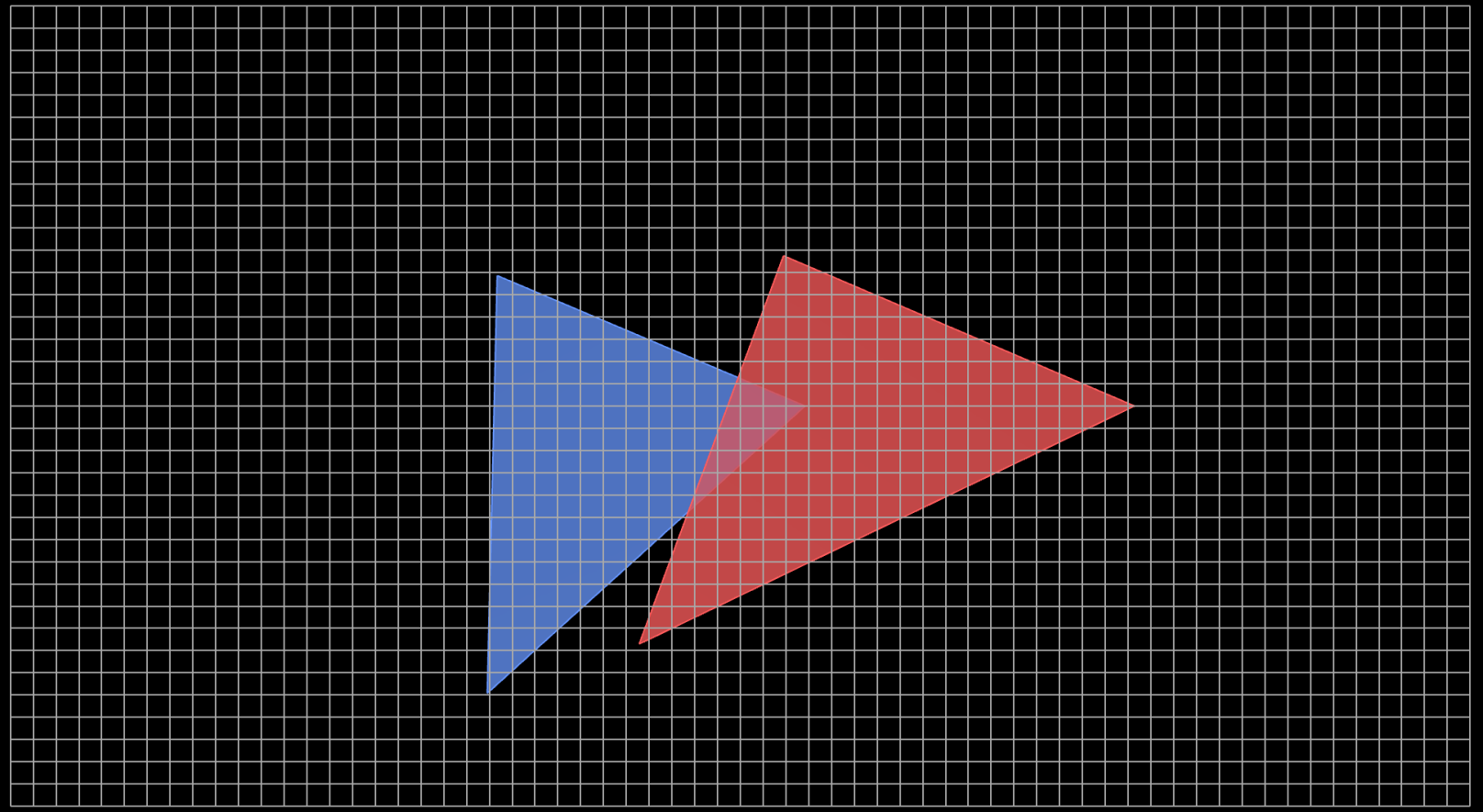
레이캐스팅



우리의 목표는 삼각형을 저 화면에 투영시키는 것입니다.

레이캐스팅 원리 이해하기

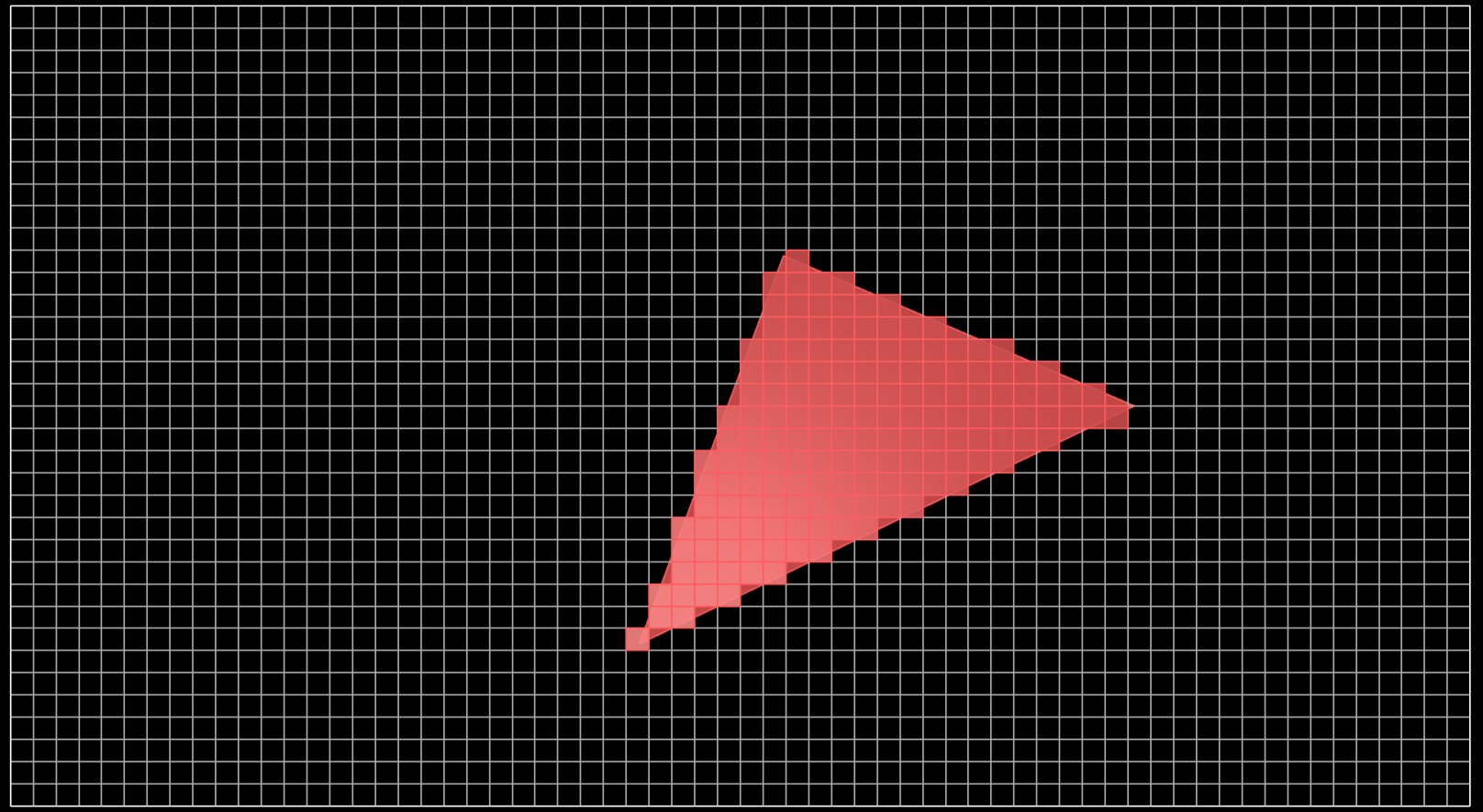
레이캐스팅



정면에서 보면 이런 모습일 것입니다.

레이캐스팅 원리 이해하기

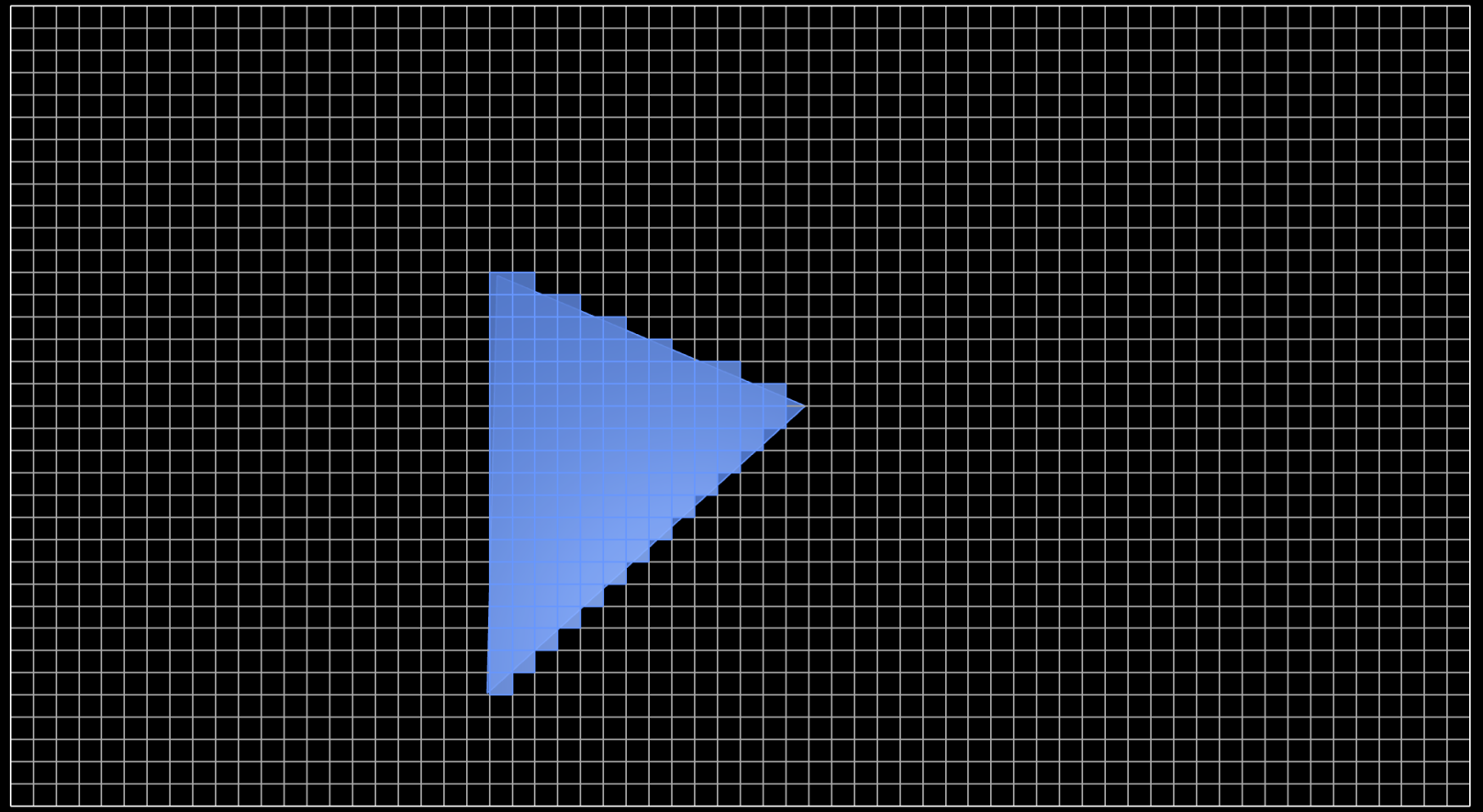
레이캐스팅



픽셀로 표현해 봅니다.

레이캐스팅 원리 이해하기

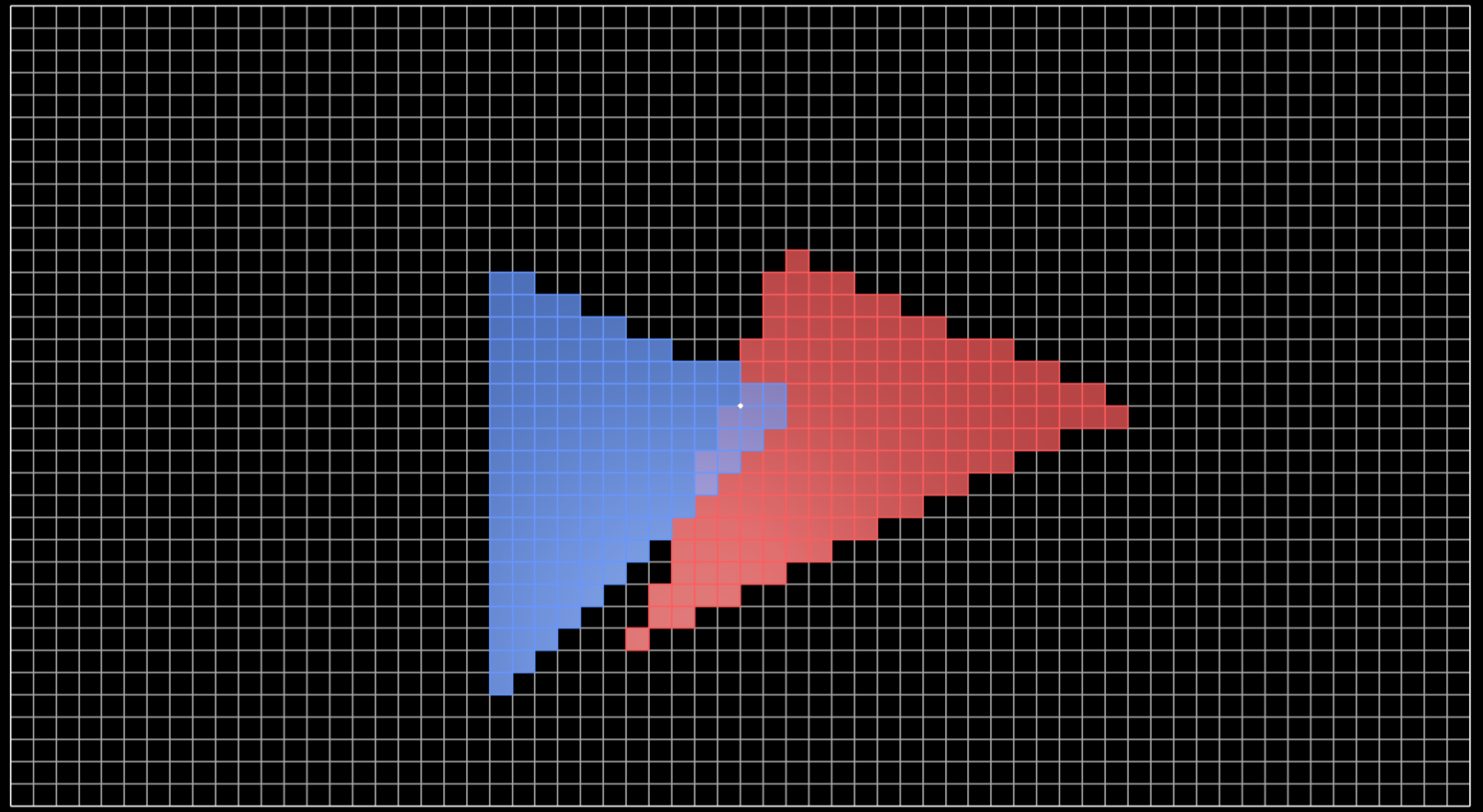
레이캐스팅



픽셀로 표현해 봅니다.

레이캐스팅 원리 이해하기

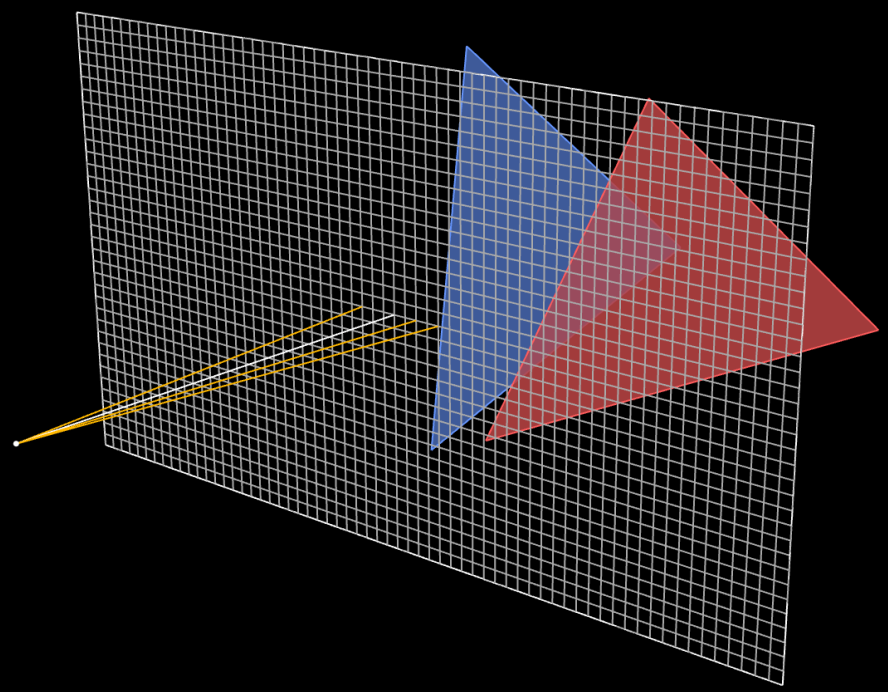
레이캐스팅



그런데 두 면이 겹치는 부분이 있습니다.
둘 중 무엇이 앞에 있는지 컴퓨터는 어떻게 판단할까요?

레이캐스팅 원리 이해하기

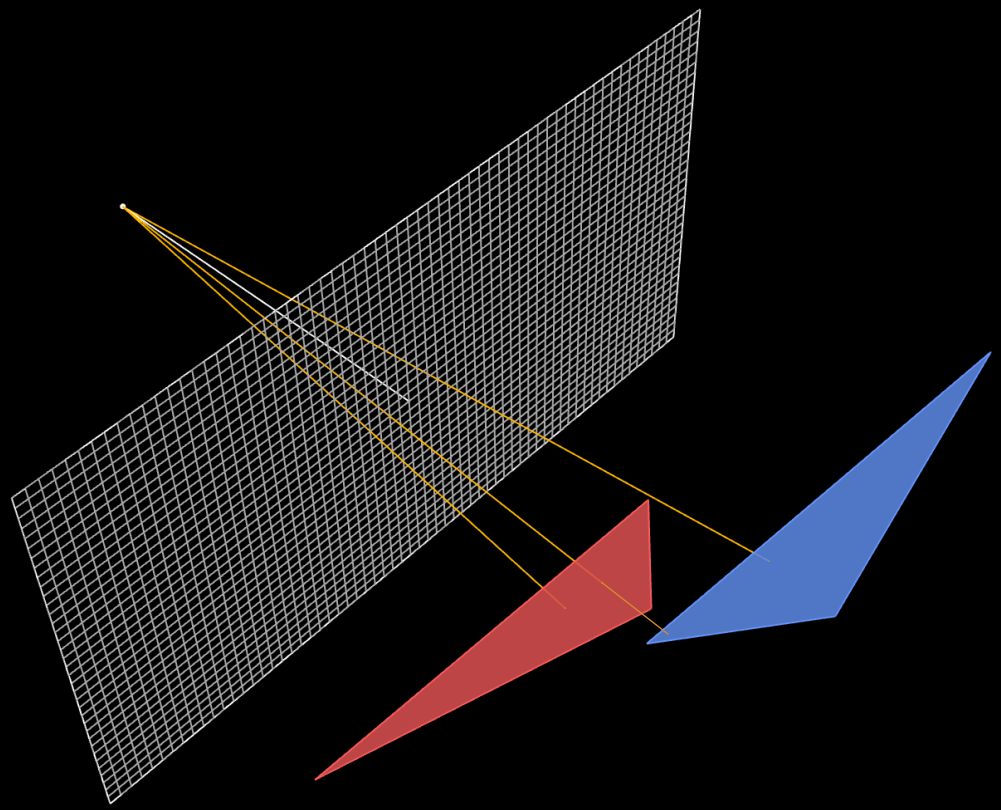
레이캐스팅



시점에서 픽셀을 향하는 반직선을 그어봅니다.

레이캐스팅 원리 이해하기

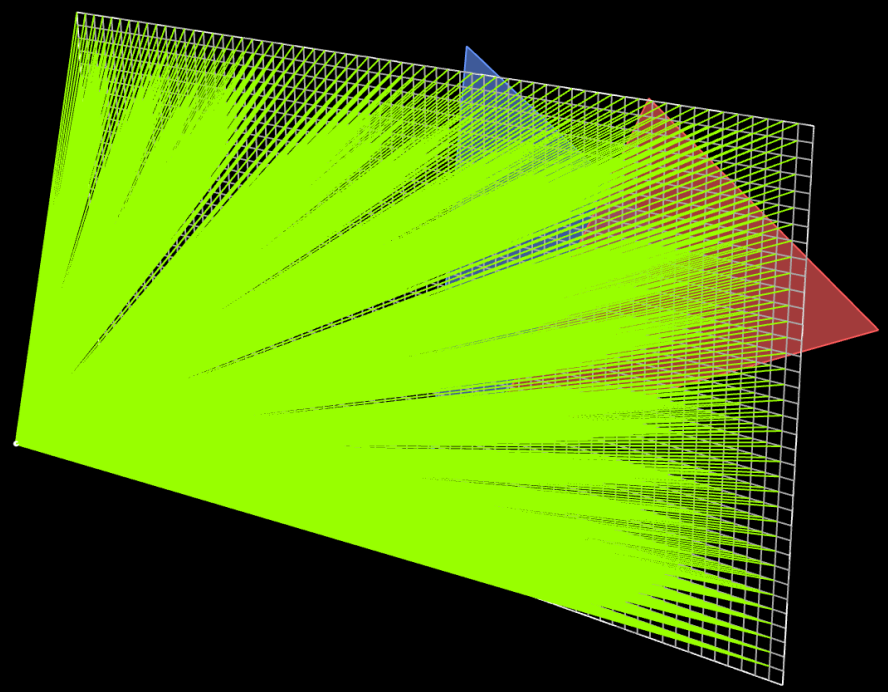
레이캐스팅



시점에서 픽셀을 이은 선을 쭉 연장해 봅니다.
우리가 그은 선들은 면과 만납니다. 한 면과 만나는 선도 있고, 두 면과 만나는 선도 있습니다.
선이 면과 만날 때의 거리와 색깔을 저장해두고, 거리에 따라 갱신하다 보면 가장 가까운 면의 색깔을 찾을 수 있습니다.

레이캐스팅 원리 이해하기

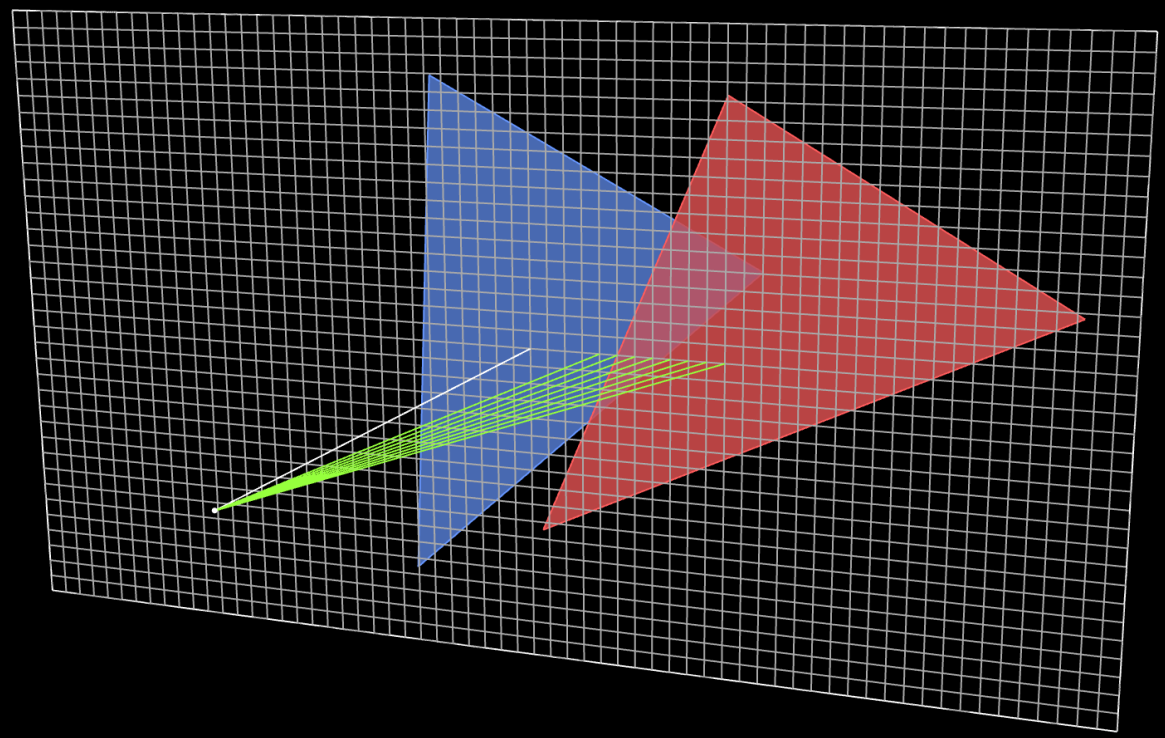
레이캐스팅



따라서 픽셀 전체에 반직선을 만들어 놓고, 각각의 반직선이 면과 만날 때,
그 거리와 그 면의 색을 저장하고, 갱신하면 우리가 원하는대로 화면을 출력할 수 있습니다.
이것을 레이캐스팅(Raycasting)이라고 합니다.

레이캐스팅 원리 이해하기

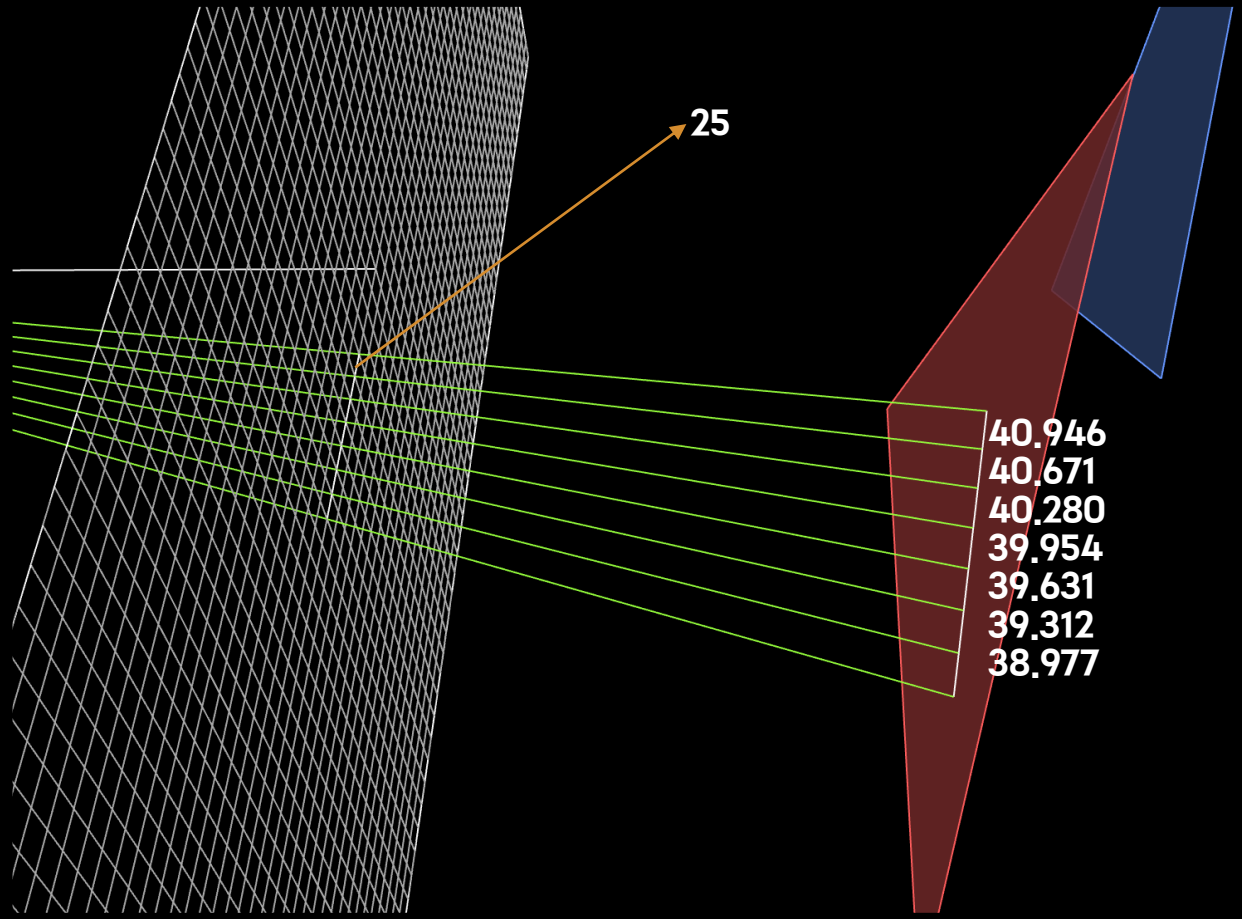
레이캐스팅



그러나 이런 방법은 비용이 클 것 같습니다.
픽셀의 수가 수만개~수백만개에 달하고, 면의 개수가 수천개에 달하면 시간 복잡도가 너무 큽니다.
한 면이 모든 픽셀에서 나오는 선과 만나는 건 아닙니다. 굳이 모든 면과 모든 픽셀을 서로 검증할 필요는 없을 것 같습니다.

레이캐스팅 원리 이해하기

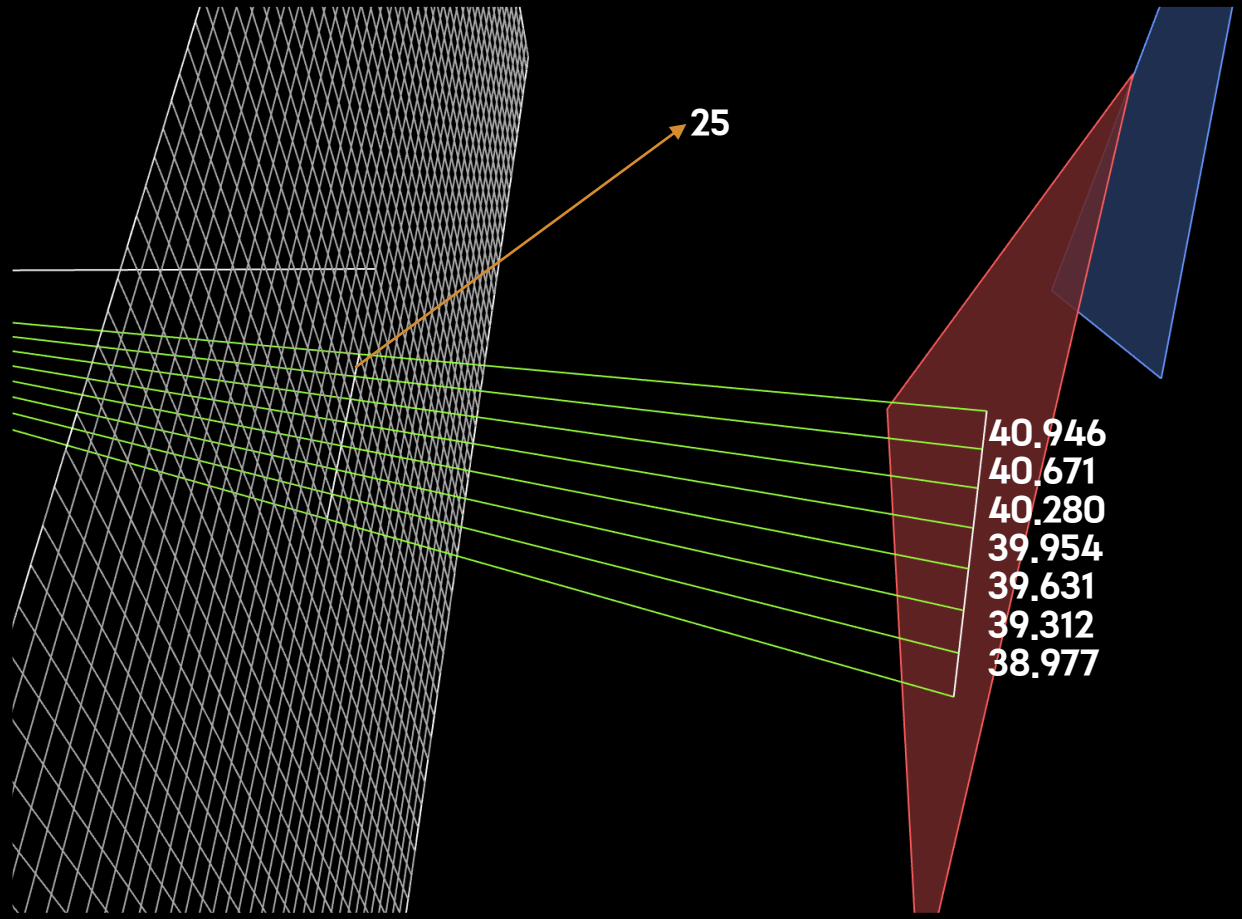
레이캐스팅



그래서 딱 삼각형이 담기는 픽셀에서만 거리를 재고 색을 담으면 좋을 것 같습니다. 한 번 상상해 봅시다.
면에 점을 흩뿌려 봅시다. 꼭지점을 화면에 투영했던 것처럼 면 위에 흩뿌려진 점을 화면에 투영하는 것은 어렵지 않습니다.
그러나 흩뿌려진 점들이 화면에 투영되었을 때 너무 듬성듬성 하지도, 쓸 데 없이 너무 빽빽하지도 않아야 합니다.

레이캐스팅 원리 이해하기

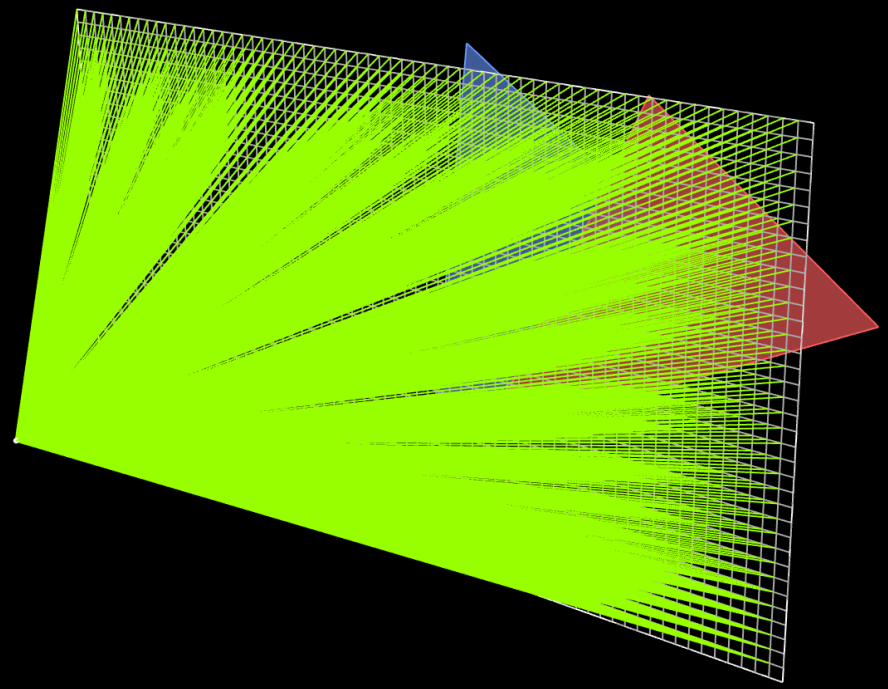
레이캐스팅



그러나 화면에 투영했을 때 픽셀 간격에 딱딱 맞는 배치는 쉽게 구할 수 없습니다.
그림에서 보이듯, 간격이 일정하지 않습니다.

레이캐스팅 원리 이해하기

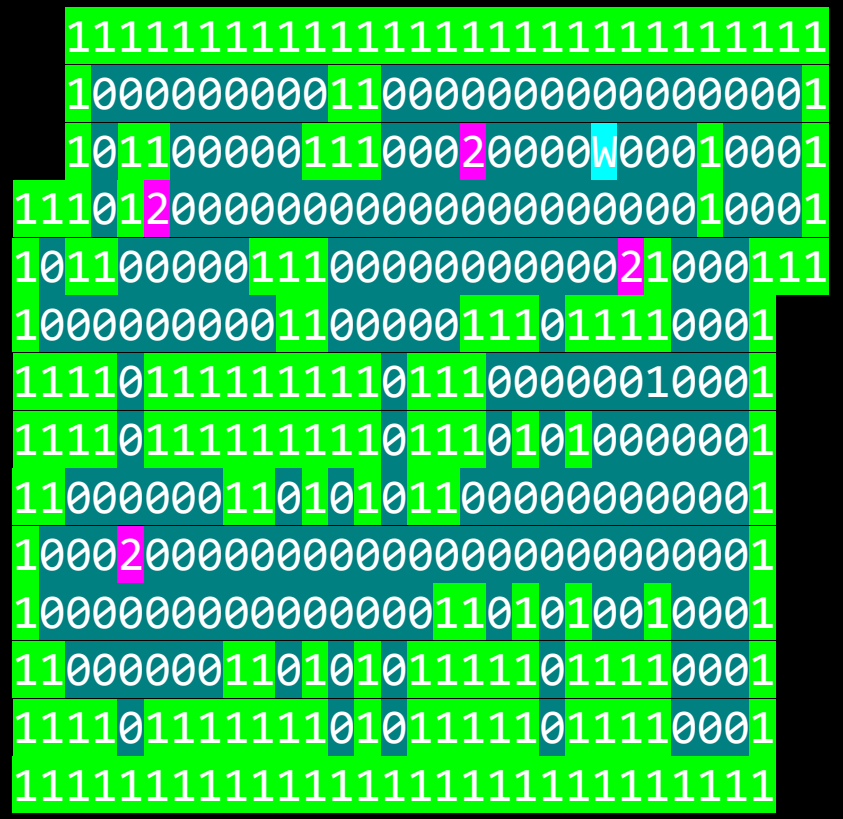
레이캐스팅



결국 각 픽셀에서 빛을 쏠 수밖에 없습니다.

레이캐스팅 원리 이해하기

레이캐스팅



하지만 우리 과제는 레이캐스팅 대상이 2차원입니다.
따라서 스크린의 세로방향으로는 고려하지 않아도 됩니다.
연산 부하가 수 백 배는 줄어들었습니다!

화면 초기화

mlx 초기화 하기

화면 초기화

mlx 구조체 다루기

```
typedef struct      s_screen{
    void            *mlx;
    void            *win;
    t_img           img;
    t_pixel         **pixel;
    t_vec           origin;
    t_vec           dir;
    t_vec           plane;
    double          sin_unit;
    double          cos_unit;
    double          distance;
    t_ray           *ray;
    int             w;
    int             h;
}                   t_screen;

typedef struct      s_img
{
    int             w;
    int             h;
    int             bits_per_pixel;
    int             size_line;
    int             endian;
    void            *ptr;
    unsigned int    *addr;
}                   t_img;
```

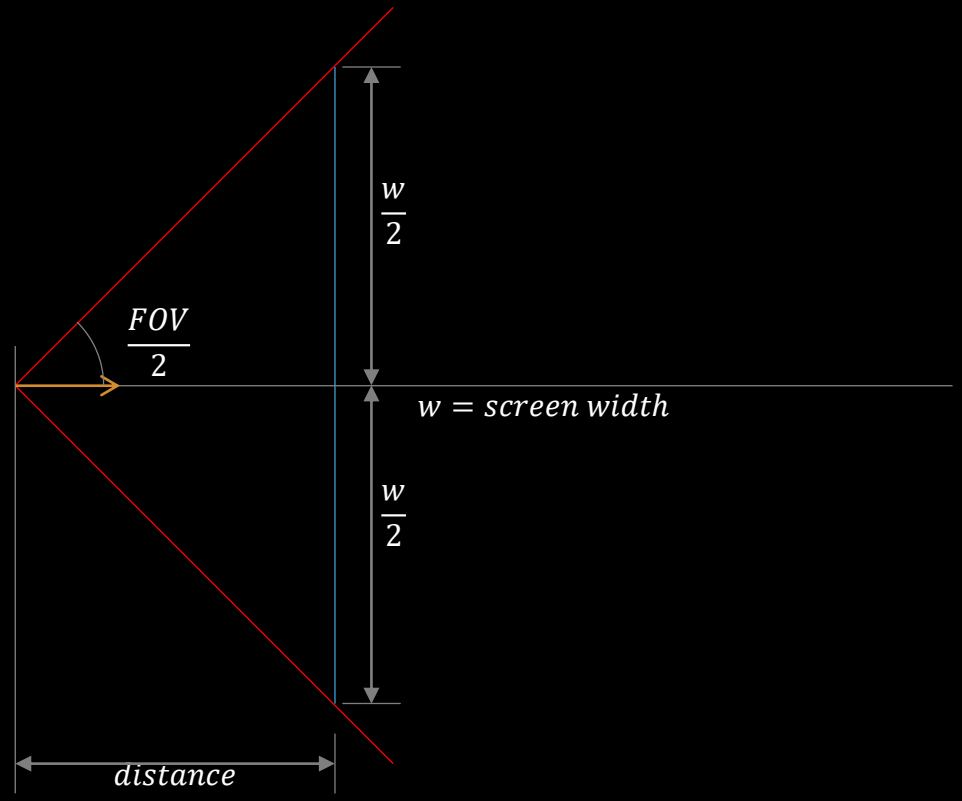
```
    t_screen    screen;
    t_screen    *s;

    s = &screen;
    s->mlx = mlx_init();
    s->win = mlx_new_window(s->mlx, s->w, s->h, "cub3D LESSON");
    s->img.ptr = mlx_new_image(s->mlx, s->w, s->h);
    s->img.addr = (unsigned int*)mlx_get_data_addr(s->img.ptr, \
        &(s->img.bits_per_pixel), &(s->img.size_line), &(s->img.endian));
```

화면 초기화

스크린 거리 구하기

```
typedef struct      s_screen{
    void            *mlx;
    void            *win;
    t_img           img;
    t_pixel         **pixel;
    t_vec           origin;
    t_vec           dir;
    t_vec           plane;
    double          sin_unit;
    double          cos_unit;
    double          distance;
    t_ray           *ray;
    int             w;
    int             h;
}                   t_screen;

typedef struct      s_img
{
    int             w;
    int             h;
    int             bits_per_pixel;
    int             size_line;
    int             endian;
    void            *ptr;
    unsigned int    *addr;
}                   t_img;
```

```
    t_screen    screen;
    t_screen    *s;

    s = &screen;
    s->mlx = mlx_init();
    s->win = mlx_new_window(s->mlx, s->w, s->h, "cub3D LESSON");
    s->img.ptr = mlx_new_image(s->mlx, s->w, s->h);
    s->img.addr = (unsigned int*)mlx_get_data_addr(s->img.ptr, \
        &(s->img.bits_per_pixel), &(s->img.size_line), &(s->img.endian));
    s->distance = 1 / tan(FOV / 2) * s->w / 2;
```

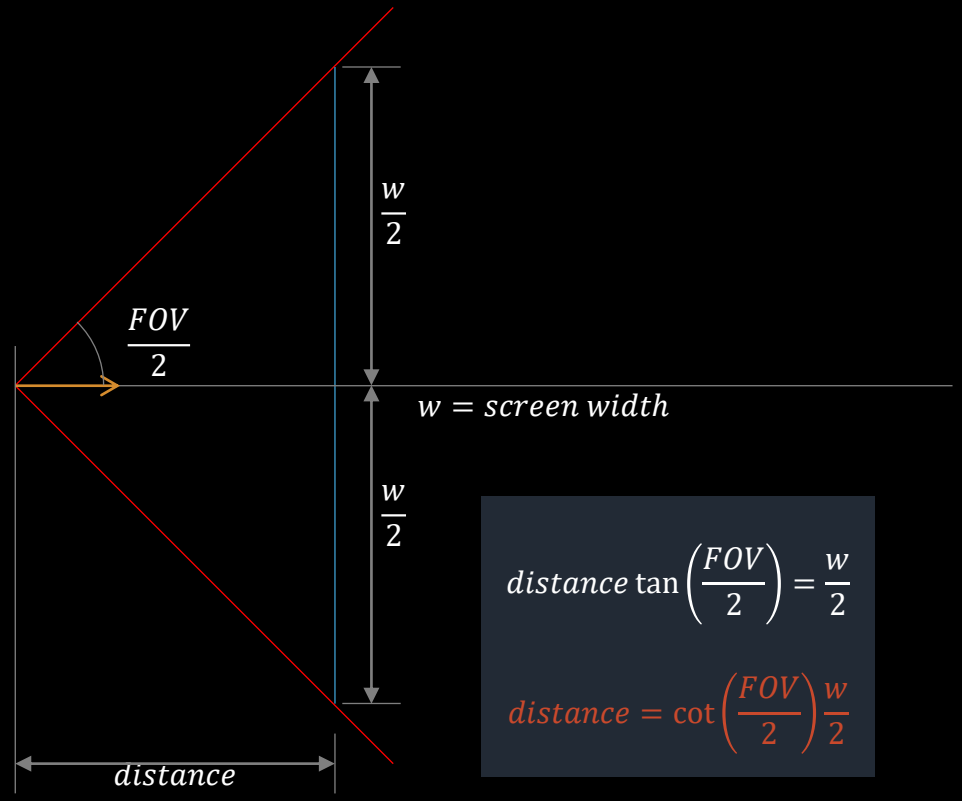


화면 초기화

스크린 거리 구하기

```
typedef struct      s_screen{
    void            *mlx;
    void            *win;
    t_img           img;
    t_pixel         **pixel;
    t_vec           origin;
    t_vec           dir;
    t_vec           plane;
    double          sin_unit;
    double          cos_unit;
    double          distance;
    t_ray           *ray;
    int             w;
    int             h;
}                   t_screen;

typedef struct      s_img
{
    int             w;
    int             h;
    int             bits_per_pixel;
    int             size_line;
    int             endian;
    void            *ptr;
    unsigned int    *addr;
}                   t_img;
```

```
    t_screen    screen;
    t_screen    *s;

    s = &screen;
    s->mlx = mlx_init();
    s->win = mlx_new_window(s->mlx, s->w, s->h, "cub3D LESSON");
    s->img.ptr = mlx_new_image(s->mlx, s->w, s->h);
    s->img.addr = (unsigned int*)mlx_get_data_addr(s->img.ptr, \
        &(s->img.bits_per_pixel), &(s->img.size_line), &(s->img.endian));
    s->distance = 1 / tan(FOV / 2) * s->w / 2;
```



$$distance \tan\left(\frac{FOV}{2}\right) = \frac{w}{2}$$

$$distance = \cot\left(\frac{FOV}{2}\right)\frac{w}{2}$$

화면 초기화

픽셀 초기화

```
typedef struct      s_screen{
    void            *mlx;
    void            *win;
    t_img           img;
    t_pixel         **pixel;
    t_vec           origin;
    t_vec           dir;
    t_vec           plane;
    double          sin_unit;
    double          cos_unit;
    double          distance;
    t_ray           *ray;
    int             w;
    int             h;
}                   t_screen;

typedef struct      s_img
{
    int             w;
    int             h;
    int             bits_per_pixel;
    int             size_line;
    int             endian;
    void            *ptr;
    unsigned int    *addr;
}                   t_img;

typedef struct      s_pixel{
    double          distance;
    unsigned int    *color;
}                   t_pixel;
```

```
    t_screen    screen;
    t_screen    *s;

    s = &screen;
    s->mlx = mlx_init();
    s->win = mlx_new_window(s->mlx, s->w, s->h, "cub3D LESSON");
    s->img.ptr = mlx_new_image(s->mlx, s->w, s->h);
    s->img.addr = (unsigned int*)mlx_get_data_addr(s->img.ptr, \
        &(s->img.bits_per_pixel), &(s->img.size_line), &(s->img.endian));
    s->distance = 1 / tan(FOV / 2) * s->w / 2;
    s->pixel = init_pixel(s->w, s->h, &s->img);
```

```
t_pixel     **init_pixel(int w, int h, t_img *img)
{
    t_pixel **pixel;
    int     x;
    int     y;

    pixel = malloc(sizeof(t_pixel*) * w);
    x = -1;
    while (++x < w)
    {
        y = -1;
        pixel[x] = malloc(sizeof(t_pixel) * h);
        while (++y < h)
        {
            pixel[x][y].distance = INFINITY;
            pixel[x][y].color = (unsigned int*)((char*)img->addr\
             + img->size_line * y + img->bits_per_pixel / 8 * x);
        }
    }
    return (pixel);
}
```

광선

광선 다발 만들기

광선

광선 구조체 초기화(메모리 공간 동적 할당)

```
typedef struct      s_screen{
    void            *mlx;
    void            *win;
    t_img           img;
    t_pixel         **pixel;
    t_vec           origin;
    t_vec           dir;
    t_vec           plane;
    double          sin_unit;
    double          cos_unit;
    double          distance;
    t_ray           *ray;
    int             w;
    int             h;
}                   t_screen;

typedef struct      s_img
{
    int             w;
    int             h;
    int             bits_per_pixel;
    int             size_line;
    int             endian;
    void            *ptr;
    unsigned int    *addr;
}                   t_img;

typedef struct      s_ray{
    t_vec           dir;
    double          distance;
}                   t_ray;
```

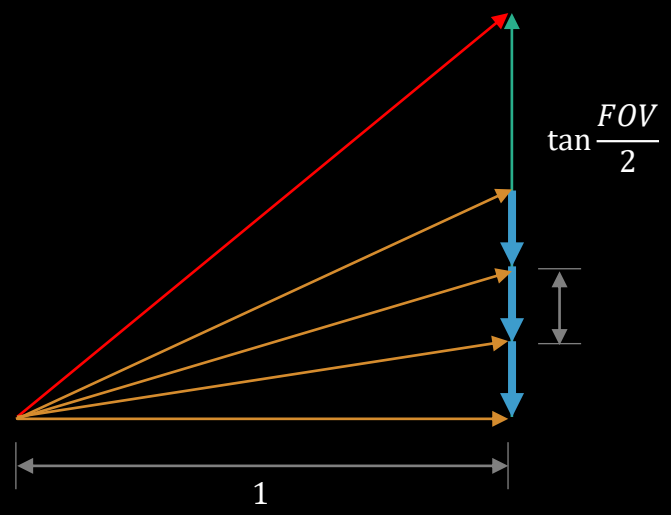
```
    t_screen    screen;
    t_screen    *s;

    s = &screen;
    s->mlx = mlx_init();
    s->win = mlx_new_window(s->mlx, s->w, s->h, "cub3D LESSON");
    s->img.ptr = mlx_new_image(s->mlx, s->w, s->h);
    s->img.addr = (unsigned int*)mlx_get_data_addr(s->img.ptr, \
        &(s->img.bits_per_pixel), &(s->img.size_line), &(s->img.endian));
    s->distance = 1 / tan(FOV / 2) * s->w / 2;
    s->pixel = init_pixel(s->w, s->h, &s->img);
    s->ray = malloc(sizeof(t_ray) * s->w);

//r == 모든 데이터를 들고 다니는 구조체
    r->player_plane = set_player_plane(&a->map, s->w);
```

```
t_vec   set_player_plane(t_map *map, int w)
{
    double  min;

    min = tan(FOV / 2) / w * 2;
    if (map->dir_init == 'E')
        return vec_new(0, -min);
    if (map->dir_init == 'N')
        return vec_new(min, 0);
    if (map->dir_init == 'W')
        return vec_new(0, min);
    return vec_new(min, 0);
}
```

화면 초기화

평면의 벡터 구하기

```c
typedef struct      s_screen{
    void            *mlx;
    void            *win;
    t_img           img;
    t_pixel         **pixel;
    t_vec           origin;
    t_vec           dir;
    t_vec           plane;
    double          sin_unit;
    double          cos_unit;
    double          distance;
    t_ray           *ray;
    int             w;
    int             h;
}                   t_screen;

typedef struct      s_img
{
    int             w;
    int             h;
    int             bits_per_pixel;
    int             size_line;
    int             endian;
    void            *ptr;
    unsigned int    *addr;
}                   t_img;

typedef struct      s_ray{
    t_vec           dir;
    double          distance;
}                   t_ray;
```



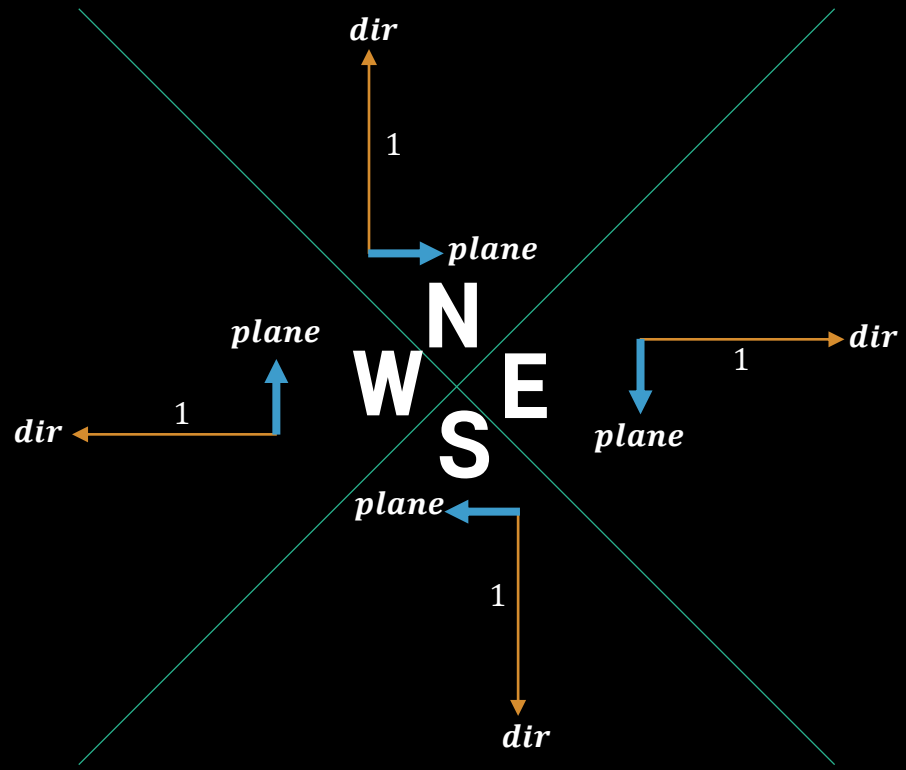
$$min = \tan\frac{FOV}{2} / \left(\frac{w}{2}\right)$$

```c
t_vec   set_player_plane(t_map *map, int w)
{
    double  min;

    min = tan(FOV / 2) / w * 2;
    if (map->dir_init == 'E')
        return vec_new(0, -min);
    if (map->dir_init == 'N')
        return vec_new(min, 0);
    if (map->dir_init == 'W')
        return vec_new(0, min);
    return vec_new(min, 0);
}
```

화면 초기화

평면의 벡터 구하기

```
typedef struct      s_screen{
    void            *mlx;
    void            *win;
    t_img           img;
    t_pixel         **pixel;
    t_vec           origin;
    t_vec           dir;
    t_vec           plane;
    double          sin_unit;
    double          cos_unit;
    double          distance;
    t_ray           *ray;
    int             w;
    int             h;
}                   t_screen;

typedef struct      s_img
{
    int             w;
    int             h;
    int             bits_per_pixel;
    int             size_line;
    int             endian;
    void            *ptr;
    unsigned int    *addr;
}                   t_img;

typedef struct      s_ray{
    t_vec           dir;
    double          distance;
}                   t_ray;
```




```
t_vec   set_player_plane(t_map *map, int w)
{
    double  min;

    min = tan(FOV / 2) / w * 2;
    if (map->dir_init == 'E')
        return vec_new(0, -min);
    if (map->dir_init == 'N')
        return vec_new(min, 0);
    if (map->dir_init == 'W')
        return vec_new(0, min);
    return vec_new(min, 0);
}
```

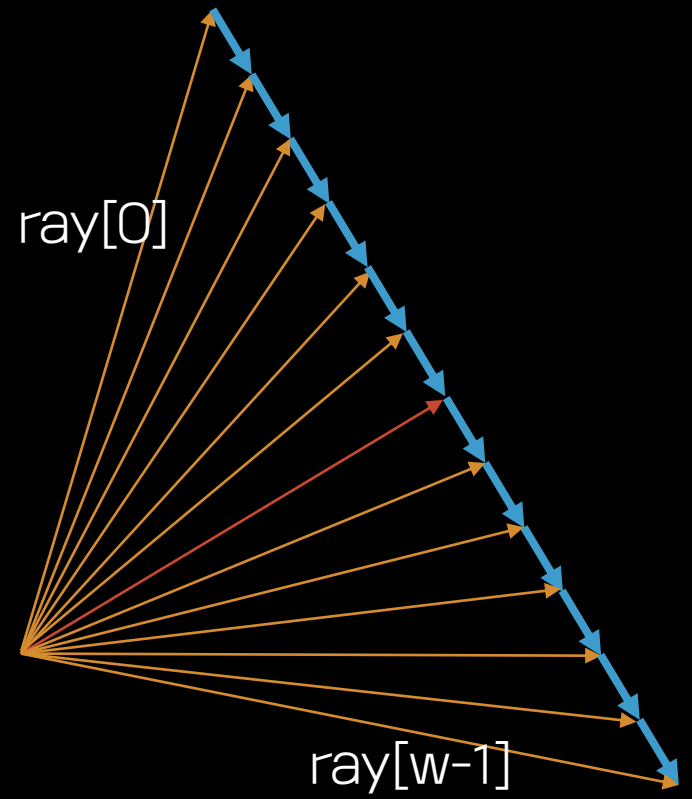
$$min = \tan\frac{FOV}{2} / \left(\frac{w}{2}\right)$$

화면 초기화

출력 전에 할 일- 광선의 내용 초기화

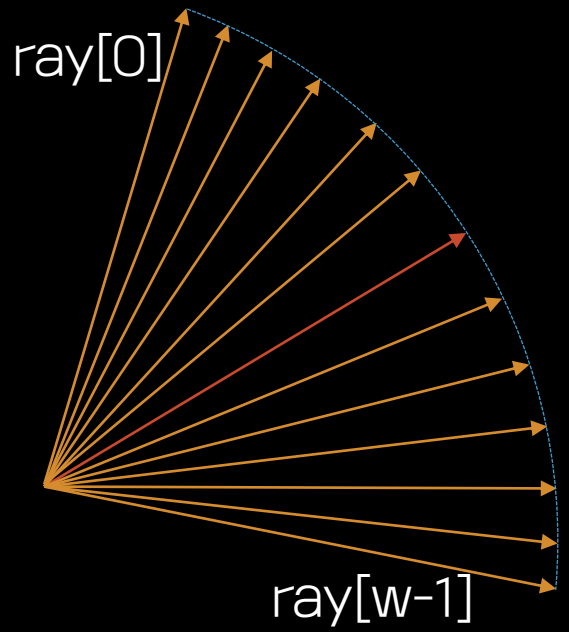
```
typedef struct      s_ray{
    t_vec           dir;
    double          distance;
}                   t_ray;
```

```
void    refresh_ray(t_ray *ray, t_vec *dir, t_vec *plane, int w)
{
    int     x;
    t_vec   plane_tmp;

    plane_tmp = vec_mul(*plane, -(double)w / 2);
    ray[0].distance = INFINITY;
    ray[0].dir = vec_add(*dir, plane_tmp);
    x = 0;
    while (++x < w)
    {
        ray[x].dir = vec_add(ray[x - 1].dir, *plane);
        ray[x].distance = INFINITY;
    }
}
```

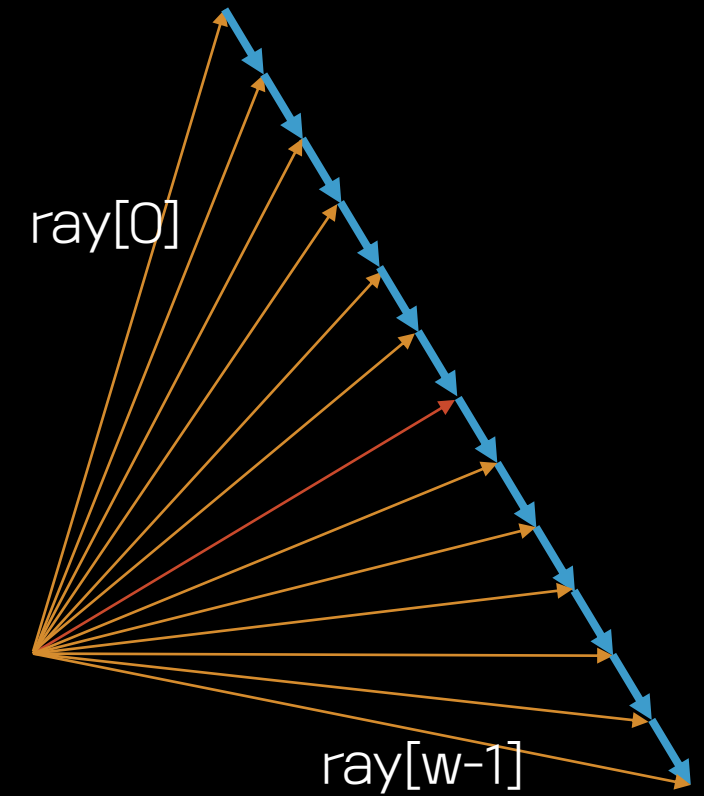


화면 초기화

출력 전에 할 일- 광선의 내용 초기화



VS



키 입력

키 입력 받는 방법

키 입력

동시입력 하는 방법

```
typedef struct      s_key
{
    char            w;
    char            s;
    char            a;
    char            d;
    char            arr_l;
    char            arr_r;
}                   t_key;
```

```
//r == 모든 데이터를 담고있는 구조체
    mlx_hook(r.screen.win, 2, 1, key_press_manager, &r.key);
    mlx_hook(r.screen.win, 3, 2, key_release_manager, &r.key);
    mlx_hook(r.screen.win, 17, 1L << 5, cub_close, 0);
```

```
int     cub_close(void)
{
    printf("bye\n");
    exit(0);
}
```

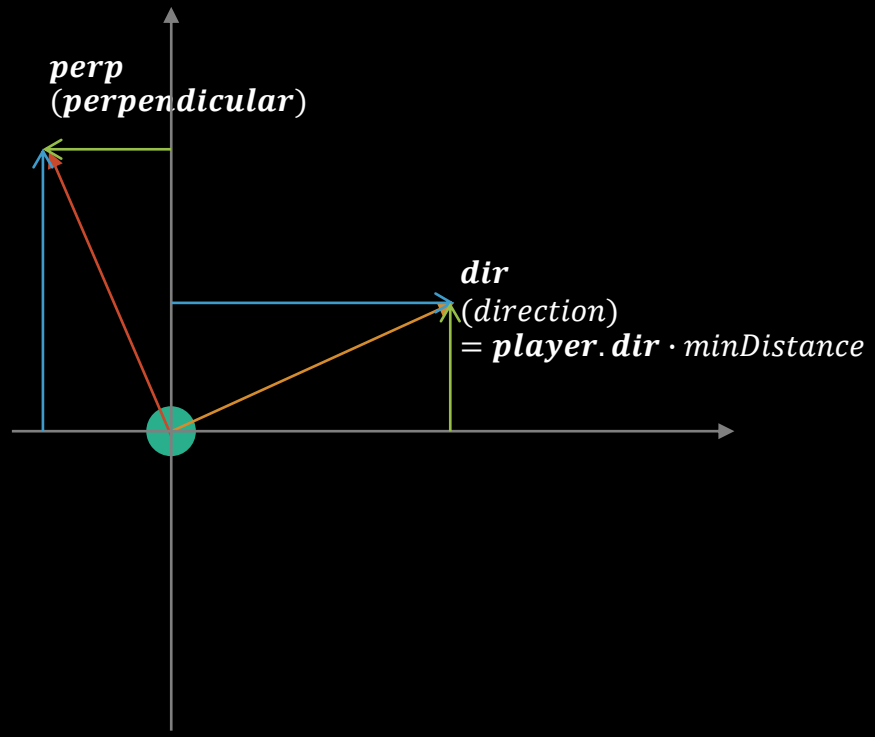
```
int     key_press_manager(int key, t_key *key_storage)
{
    if (key == KEY_A)
        key_storage->a = 1;
    if (key == KEY_S)
        key_storage->s = 1;
    if (key == KEY_D)
        key_storage->d = 1;
    if (key == KEY_W)
        key_storage->w = 1;
    if (key == KEY_ARR_L)
        key_storage->arr_l = 1;
    if (key == KEY_ARR_R)
        key_storage->arr_r = 1;
    if (key == KEY_ESC)
        cub_close();
    return (0);
}
```

```
int     key_release_manager(int key, t_key *key_storage)
{
    if (key == KEY_A)
        key_storage->a = 0;
    if (key == KEY_S)
        key_storage->s = 0;
    if (key == KEY_D)
        key_storage->d = 0;
    if (key == KEY_W)
        key_storage->w = 0;
    if (key == KEY_ARR_L)
        key_storage->arr_l = 0;
    if (key == KEY_ARR_R)
        key_storage->arr_r = 0;
    return (0);
}
```

키 입력

동시입력 하는 방법

```
typedef struct      s_key
{
    char            w;
    char            s;
    char            a;
    char            d;
    char            arr_l;
    char            arr_r;
}                   t_key;
```

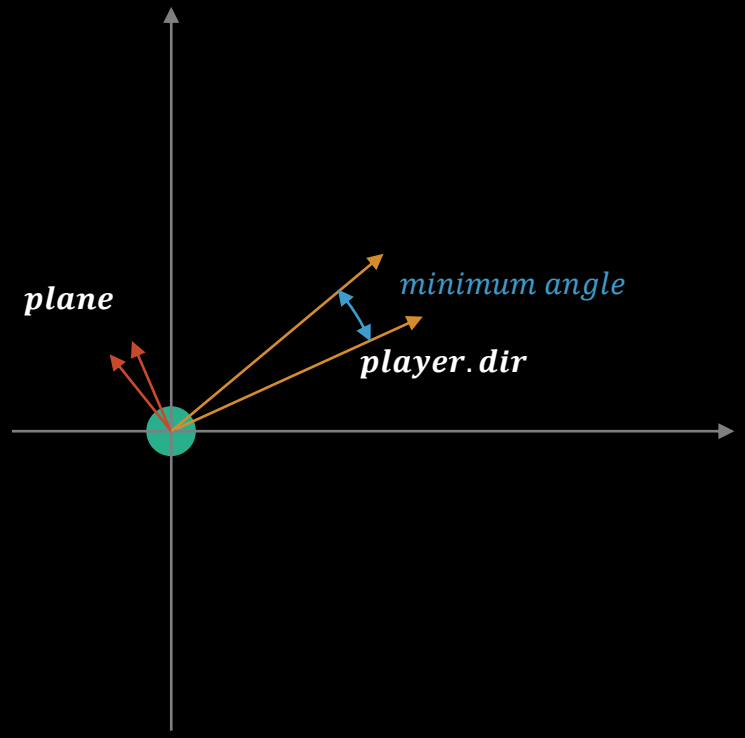



```
int     player_move(t_runtime *r)
{
    t_vec   dir;
    t_vec   perp;

    dir = vec_mul(r->player_dir, MOVE_MIN);
    perp.x = -dir.y;
    perp.y = dir.x;
    if (r->key.w)
        r->player_origin = vec_add(r->player_origin, dir);
    if (r->key.a)
        r->player_origin = vec_add(r->player_origin, perp);
    if (r->key.s)
        r->player_origin = vec_sub(r->player_origin, dir);
    if (r->key.d)
        r->player_origin = vec_sub(r->player_origin, perp);
    if (r->key.arr_l)
    {
        r->player_dir = vec_rot_min_ccw(r->player_dir);
        r->player_plane = vec_rot_min_ccw(r->player_plane);
    }
    if (r->key.arr_r)
    {
        r->player_dir = vec_rot_min_cw(r->player_dir);
        r->player_plane = vec_rot_min_cw(r->player_plane);
    }
    return 0;
}
```

키 입력

동시입력 하는 방법

```
typedef struct      s_key
{
    char            w;
    char            s;
    char            a;
    char            d;
    char            arr_l;
    char            arr_r;
}                   t_key;
```



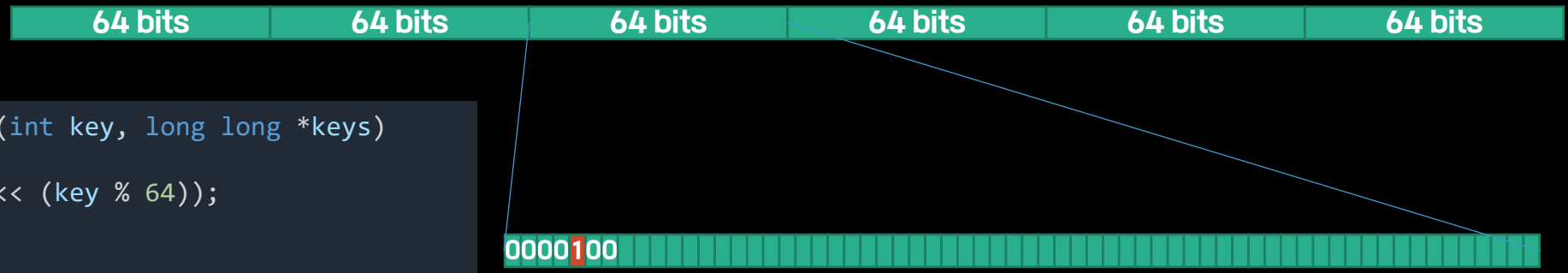

```
int     player_move(t_runtime *r)
{
    t_vec   dir;
    t_vec   perp;

    dir = vec_mul(r->player_dir, MOVE_MIN);
    perp.x = -dir.y;
    perp.y = dir.x;
    if (r->key.w)
        r->player_origin = vec_add(r->player_origin, dir);
    if (r->key.a)
        r->player_origin = vec_add(r->player_origin, perp);
    if (r->key.s)
        r->player_origin = vec_sub(r->player_origin, dir);
    if (r->key.d)
        r->player_origin = vec_sub(r->player_origin, perp);
    if (r->key.arr_l)
    {
        r->player_dir = vec_rot_min_ccw(r->player_dir);
        r->player_plane = vec_rot_min_ccw(r->player_plane);
    }
    if (r->key.arr_r)
    {
        r->player_dir = vec_rot_min_cw(r->player_dir);
        r->player_plane = vec_rot_min_cw(r->player_plane);
    }
    return 0;
}
```

키 입력

동시입력 하는 방법 - 매운맛

```
long long  keys[6];
```

if key == 132 → 64 + 64 + 4




```
int     key_press_manager(int key, long long *keys)
{
    keys[key / 64] |= (1 << (key % 64));
    return (0);
}

int     key_release_manager(int key, long long *keys)
{
    keys[key / 64] &= ~(1 << (key % 64));
    return (0);
}

int     is_pressed(int key, long long *keys)
{
    if (keys[key / 64] & 1 << (key % 64))
        return (1);
    else
        return (0);
}
```

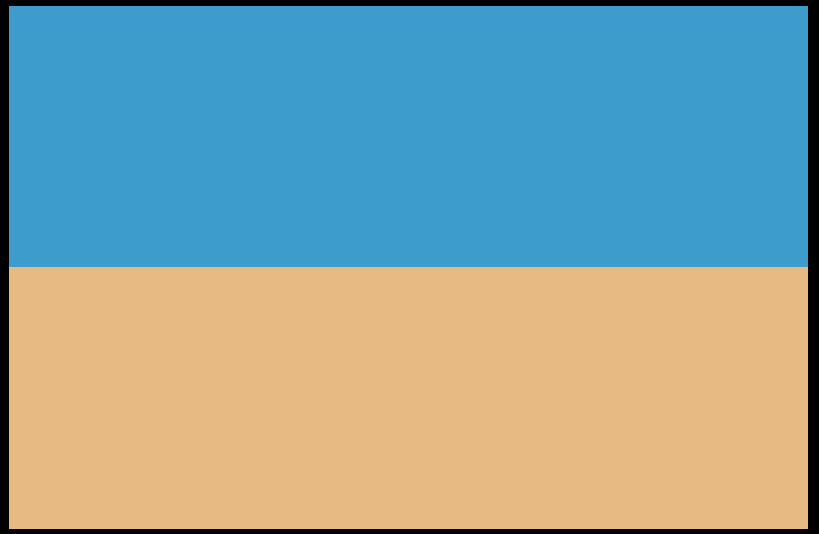
천장과 바닥 그리기

천장 바닥 색칠

천장과 바닥 그리기

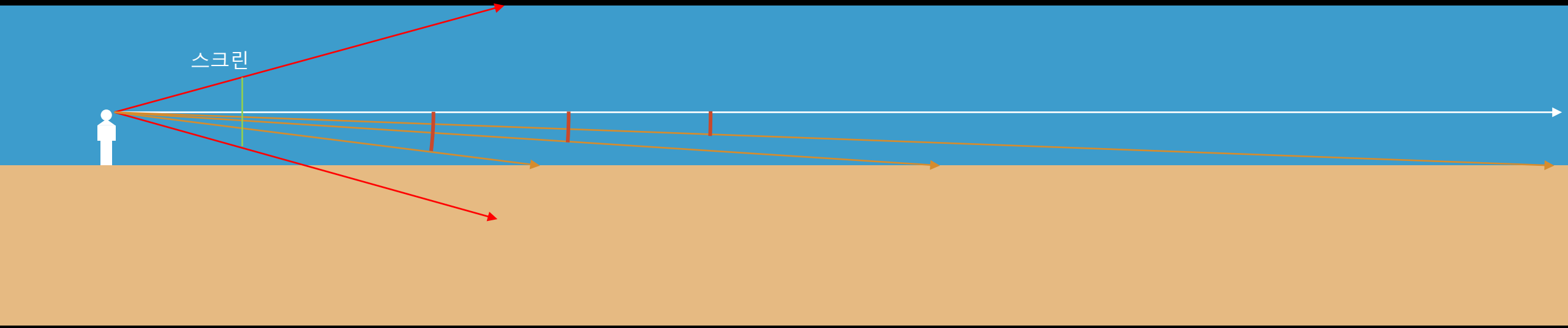
절반씩 색칠하기

```
void    draw_floor(t_screen *screen, int color_floor, int color_ceiling)
{
    int     x;
    int     y;
    t_pixel **pixel;

    pixel = screen->pixel;
    y = -1;
    while (++y < screen->h / 2)
    {
        x = -1;
        while (++x < screen->w)
            *(pixel[x][y].color) = color_ceiling;
    }
    while (++y < screen->h)
    {
        x = -1;
        while (++x < screen->w)
            *(pixel[x][y].color) = color_floor;
    }
}
```



천장과 바닥 그리기

절반씩 칠하는 이유

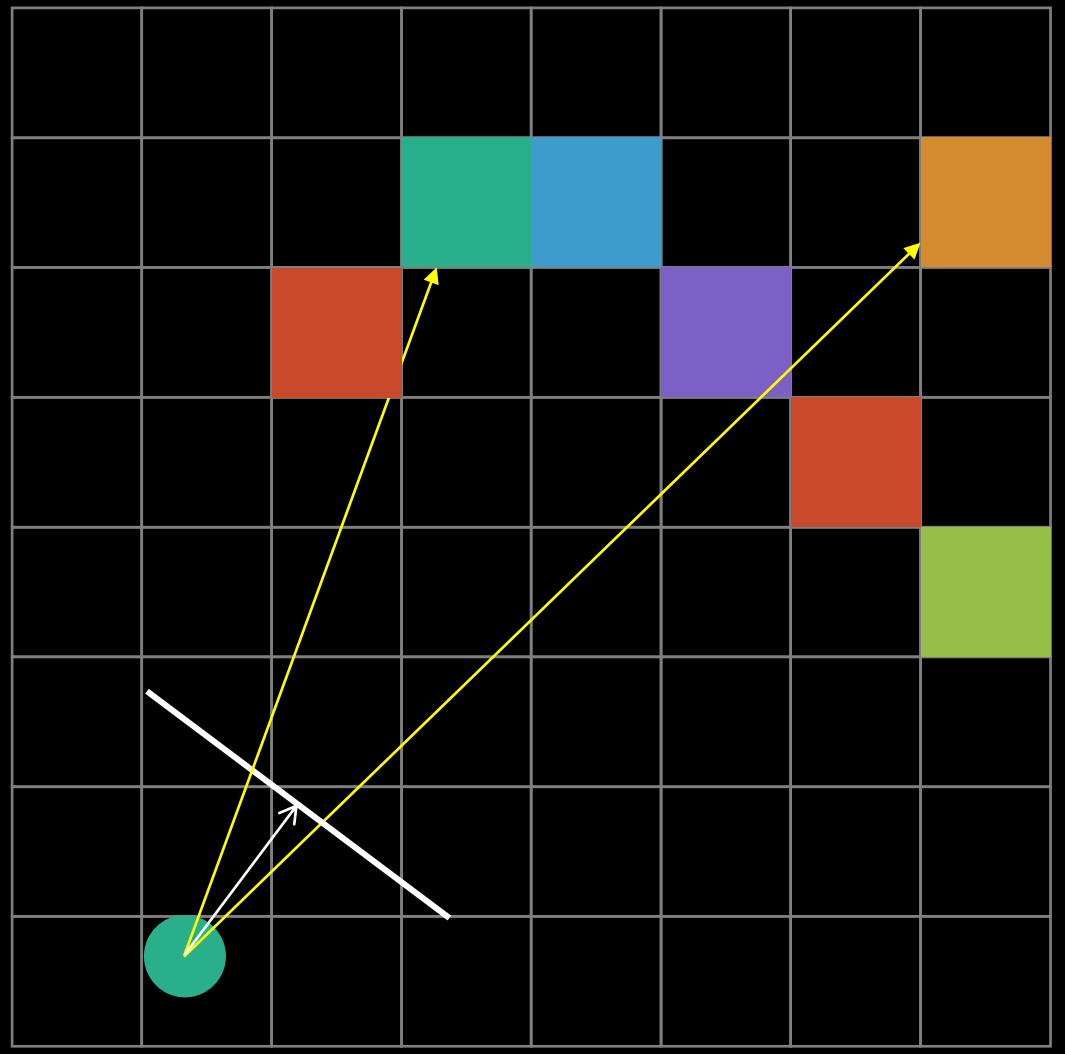


DDA알고리즘

#로데브 #정통 알고리즘 #정파

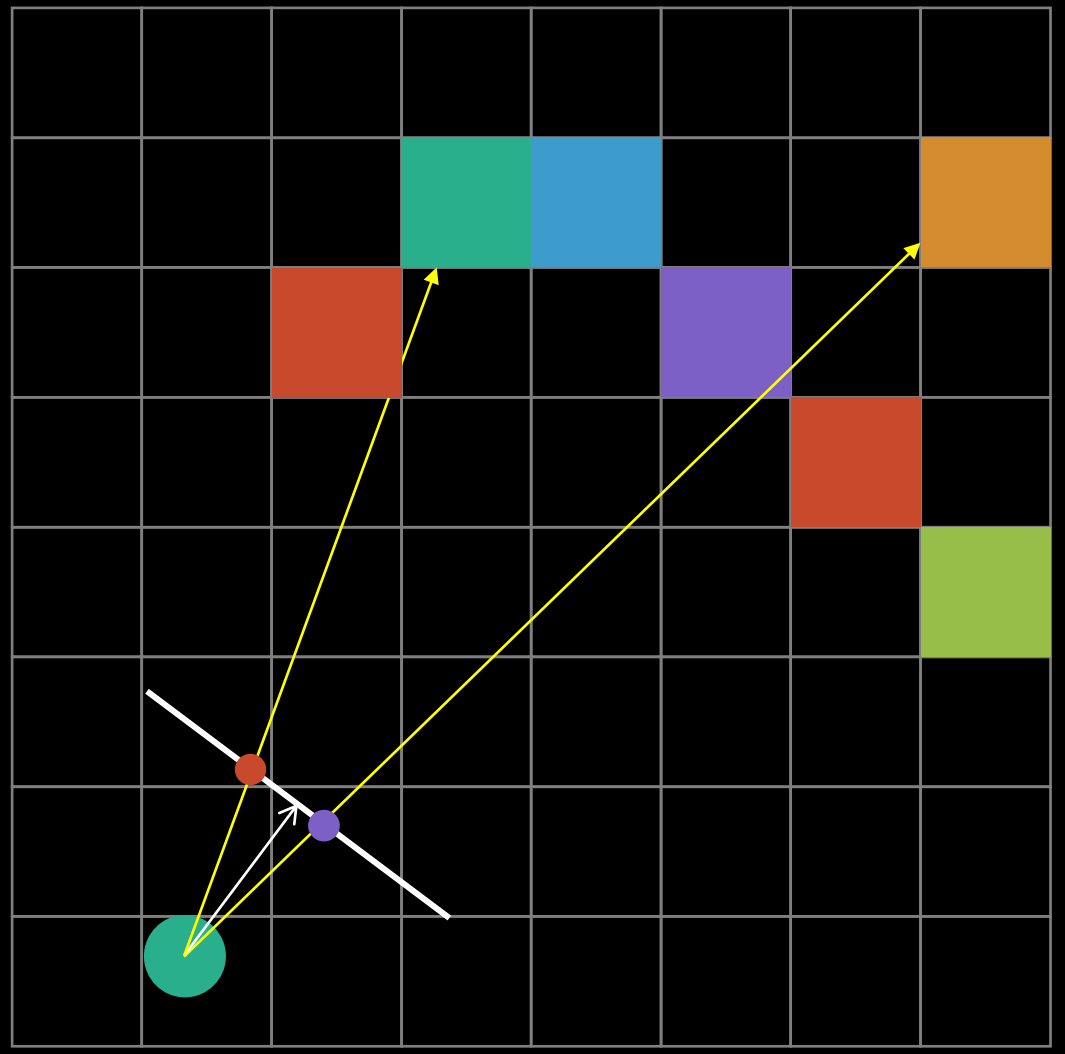
DDA알고리즘

워밍업



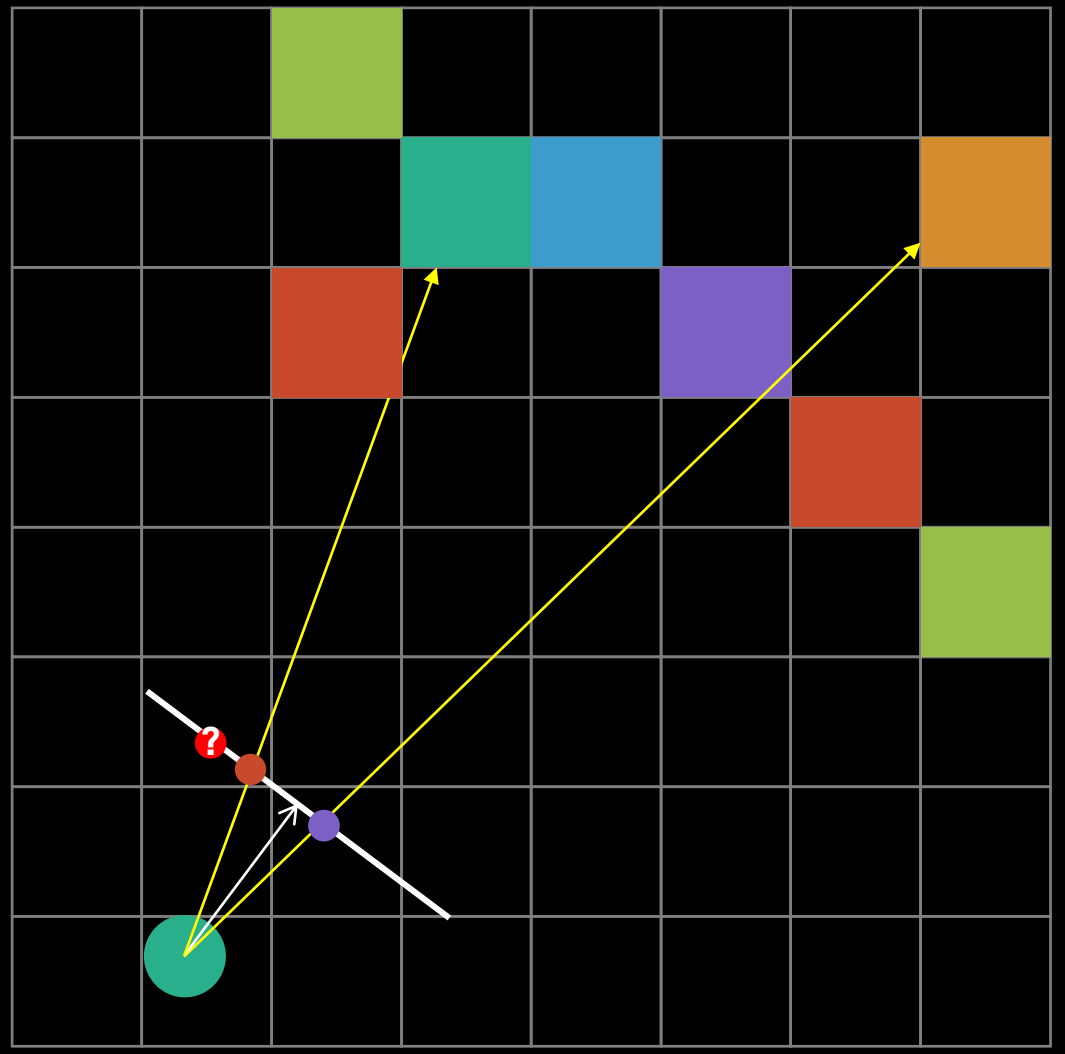
DDA알고리즘

워밍업



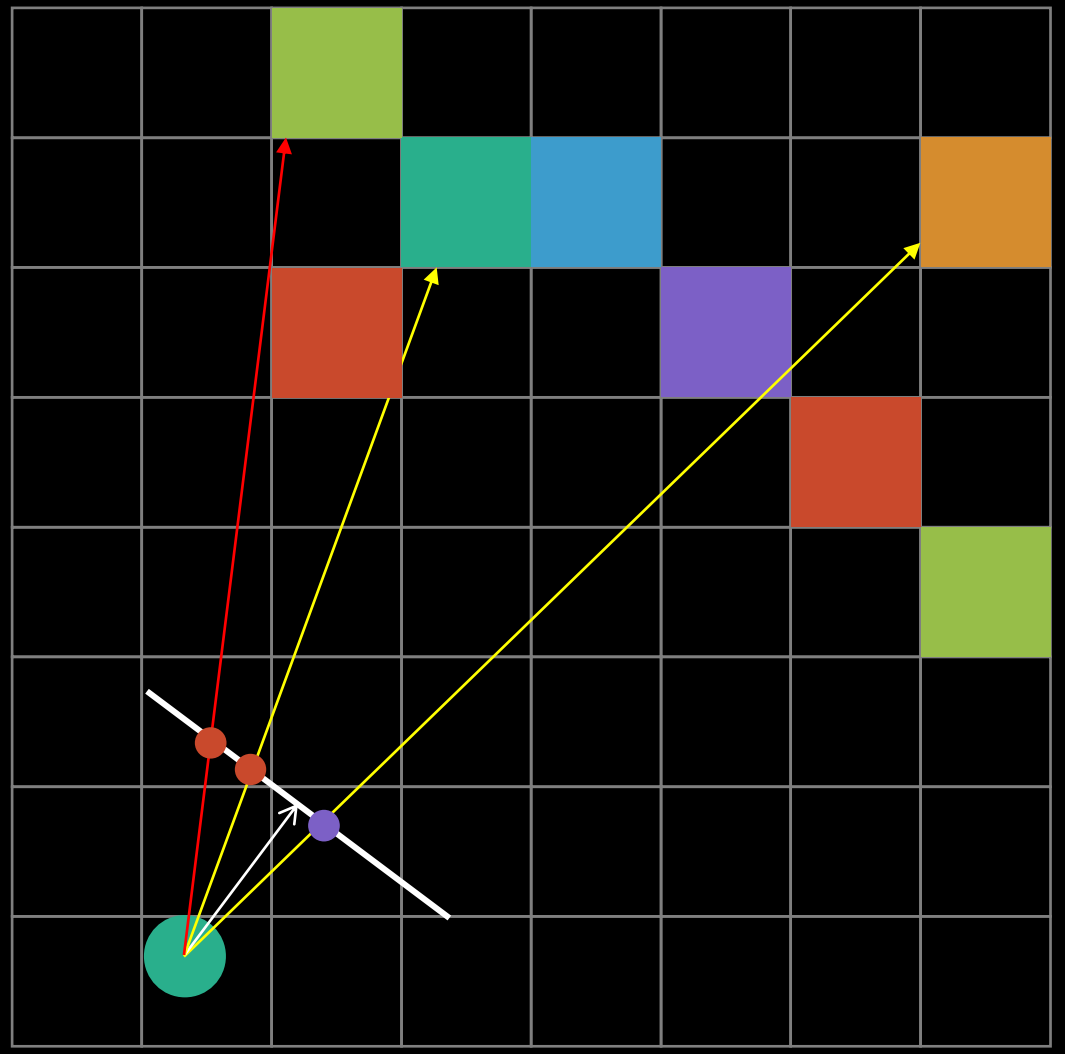
DDA알고리즘

워밍업



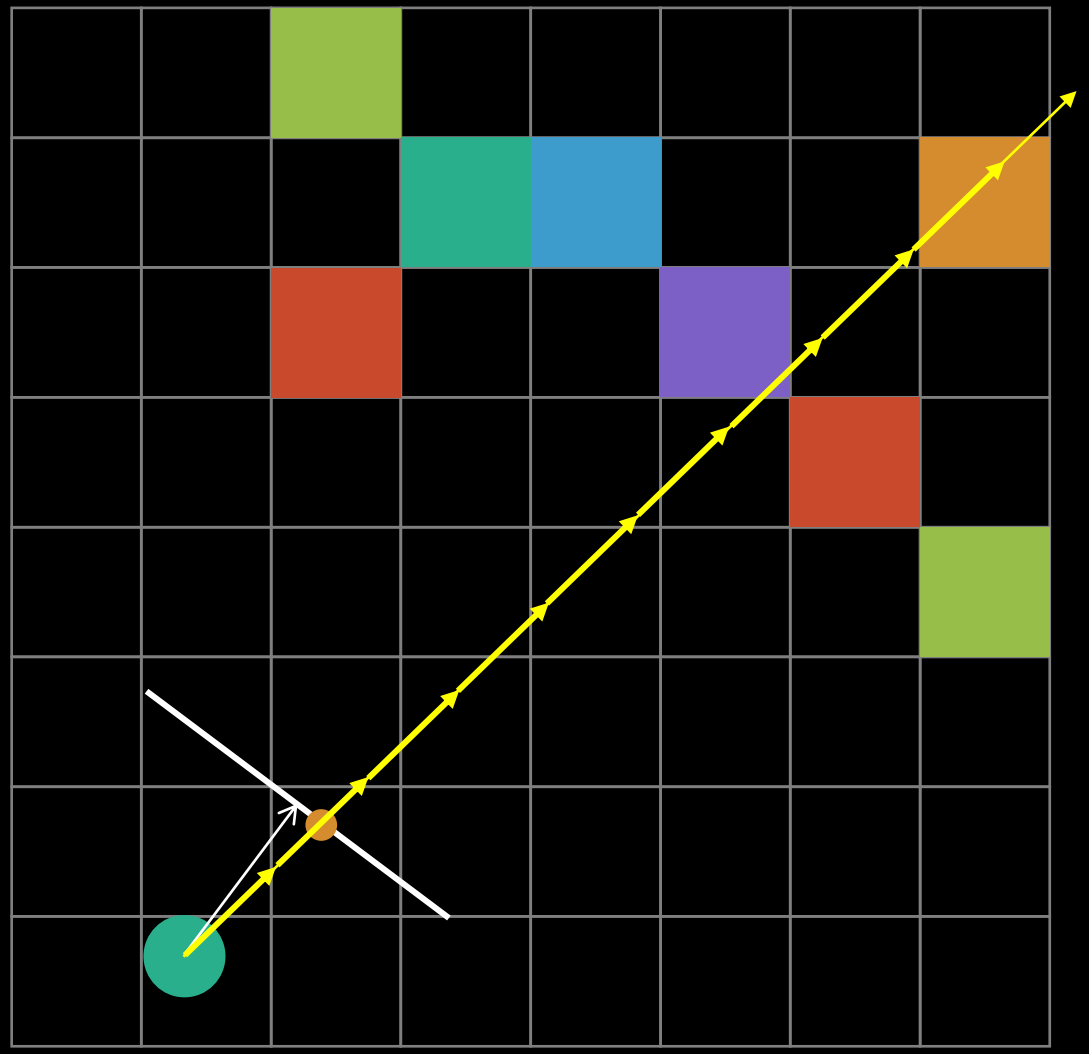
DDA알고리즘

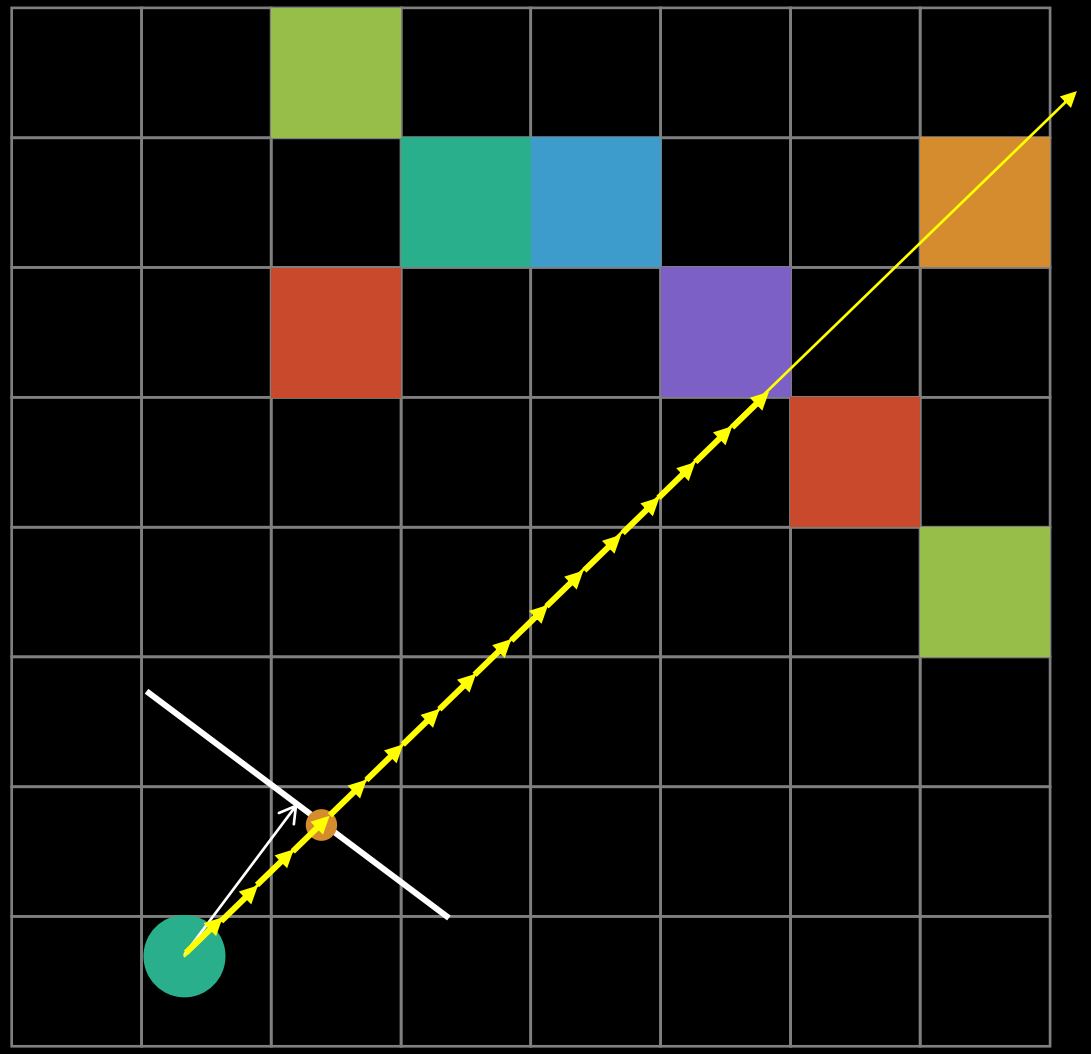
워밍업



DDA알고리즘

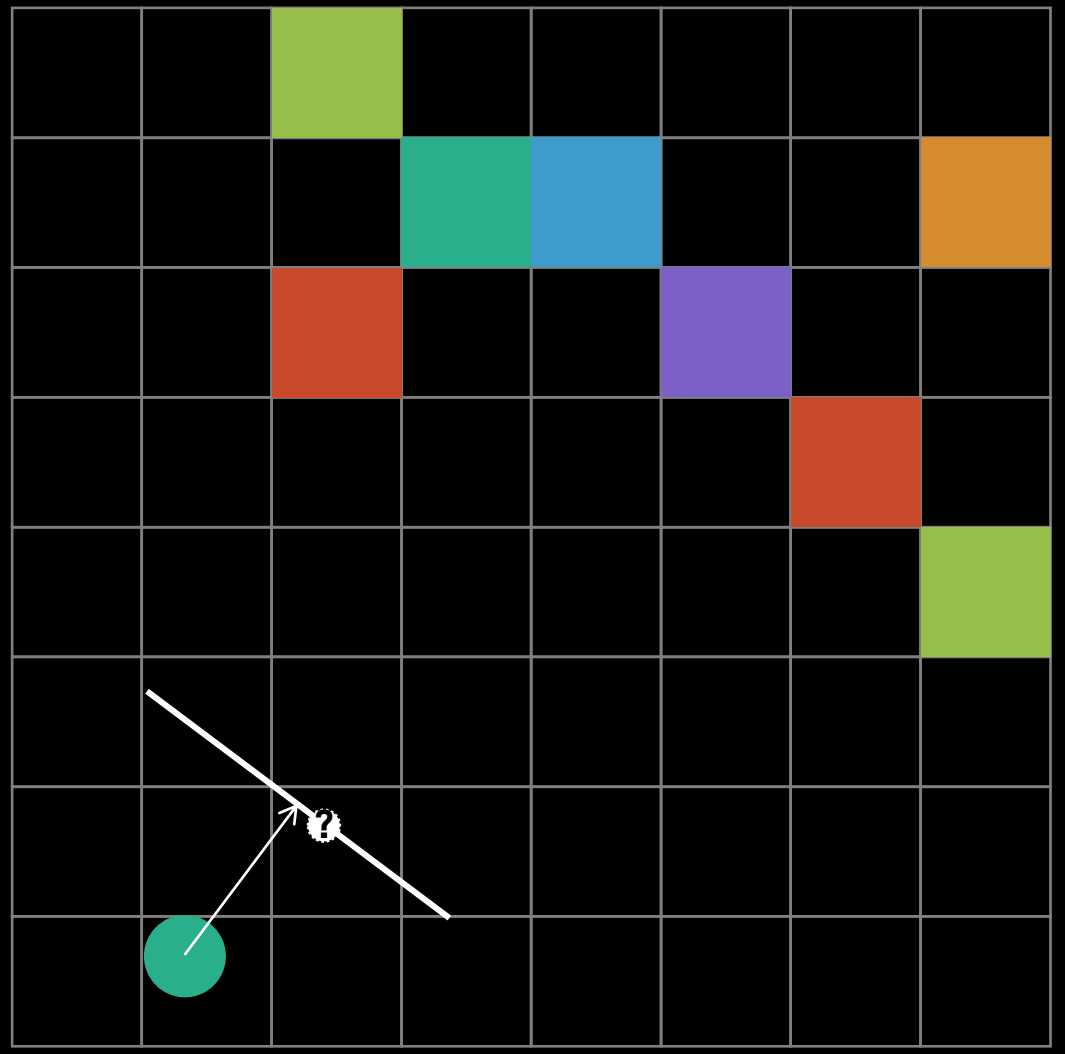
DDA알고리즘을 쓰는 이유





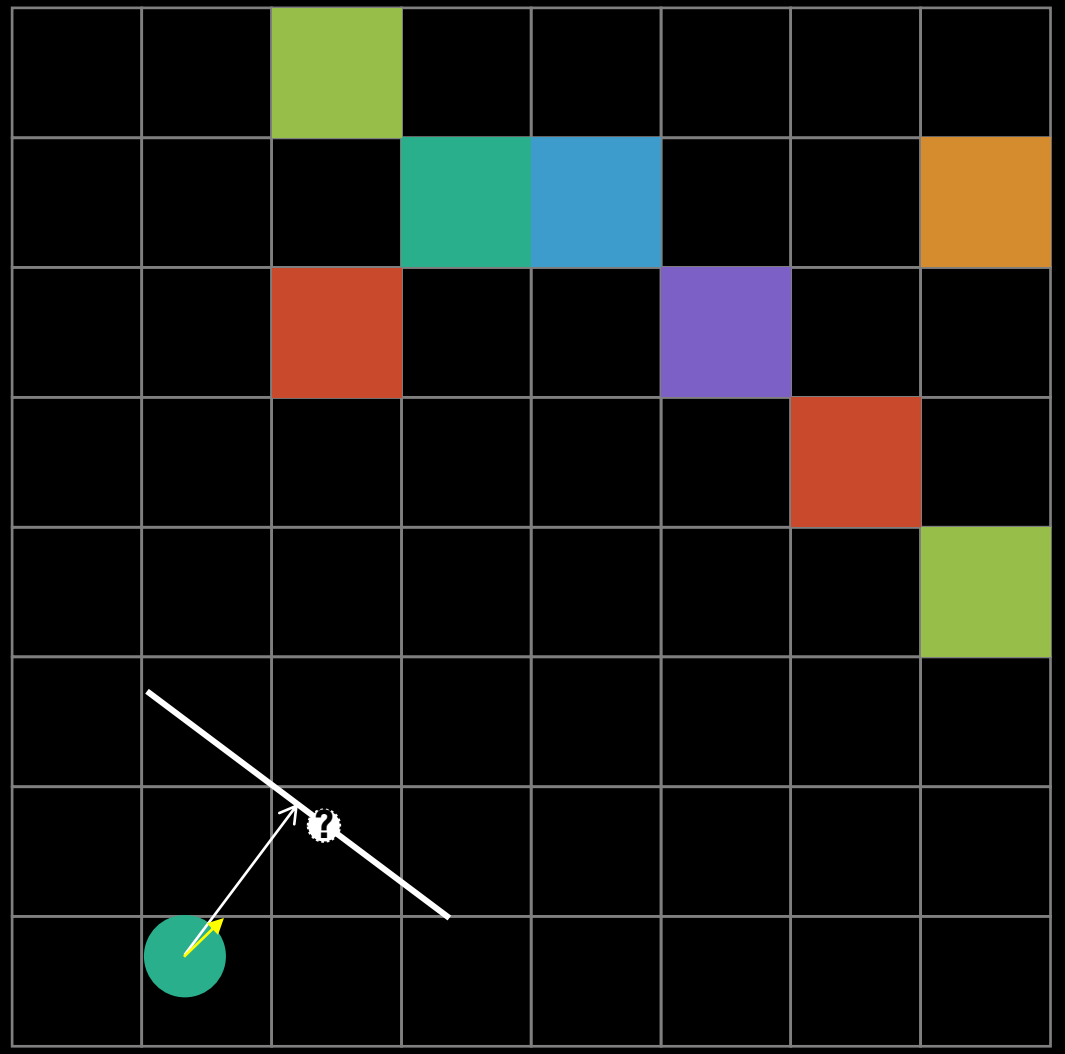
DDA알고리즘

워밍업



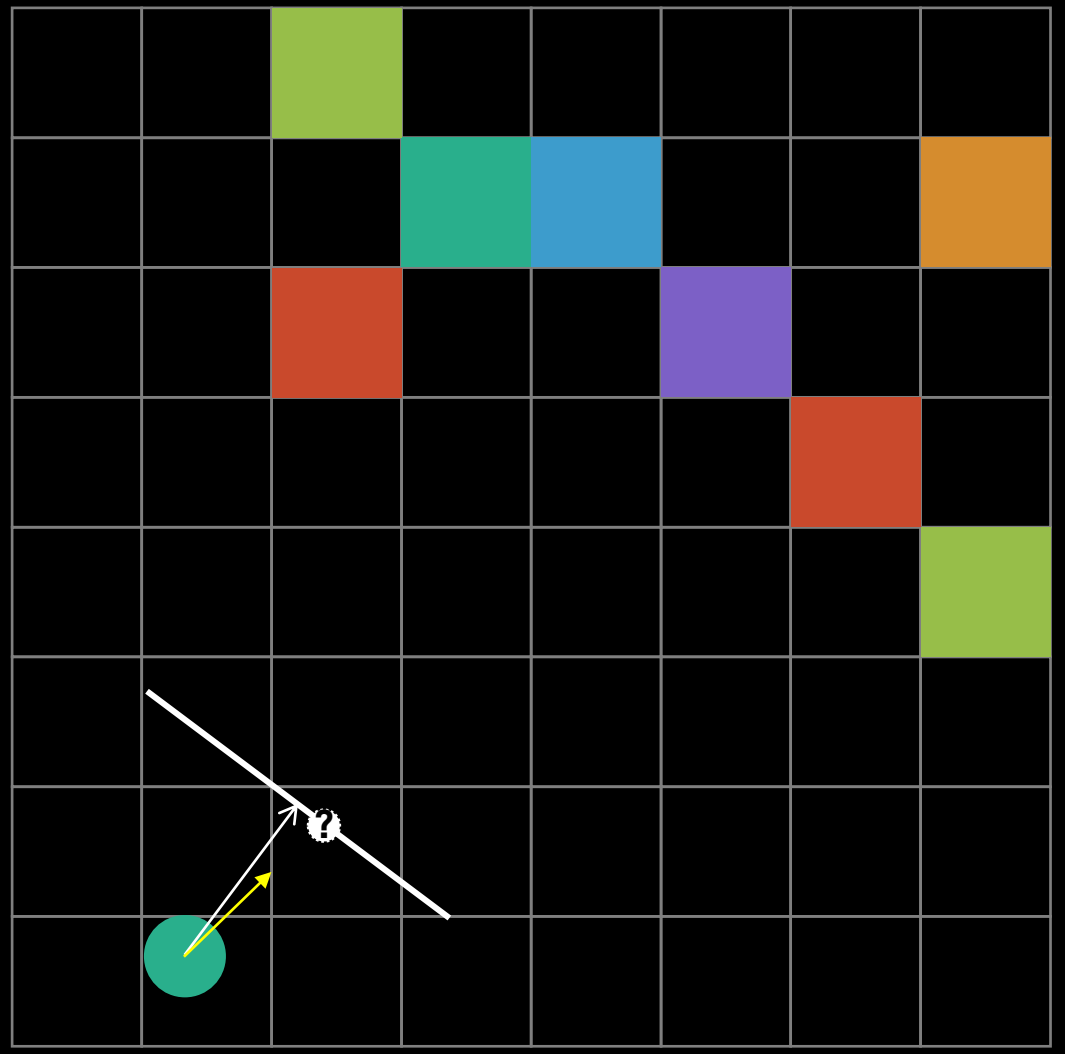
DDA알고리즘

DDA알고리즘을 직관적으로 이해하기



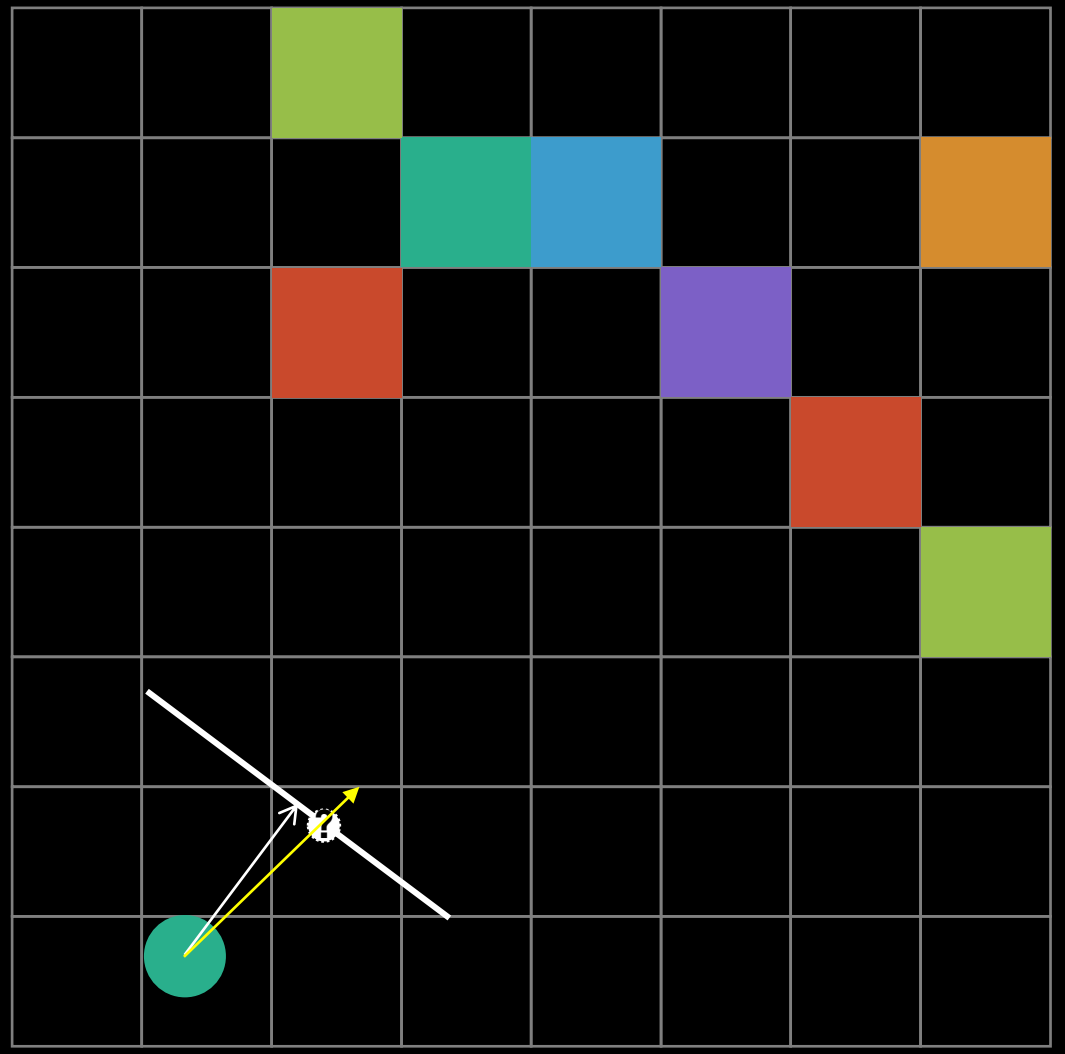
DDA알고리즘

DDA알고리즘을 직관적으로 이해하기



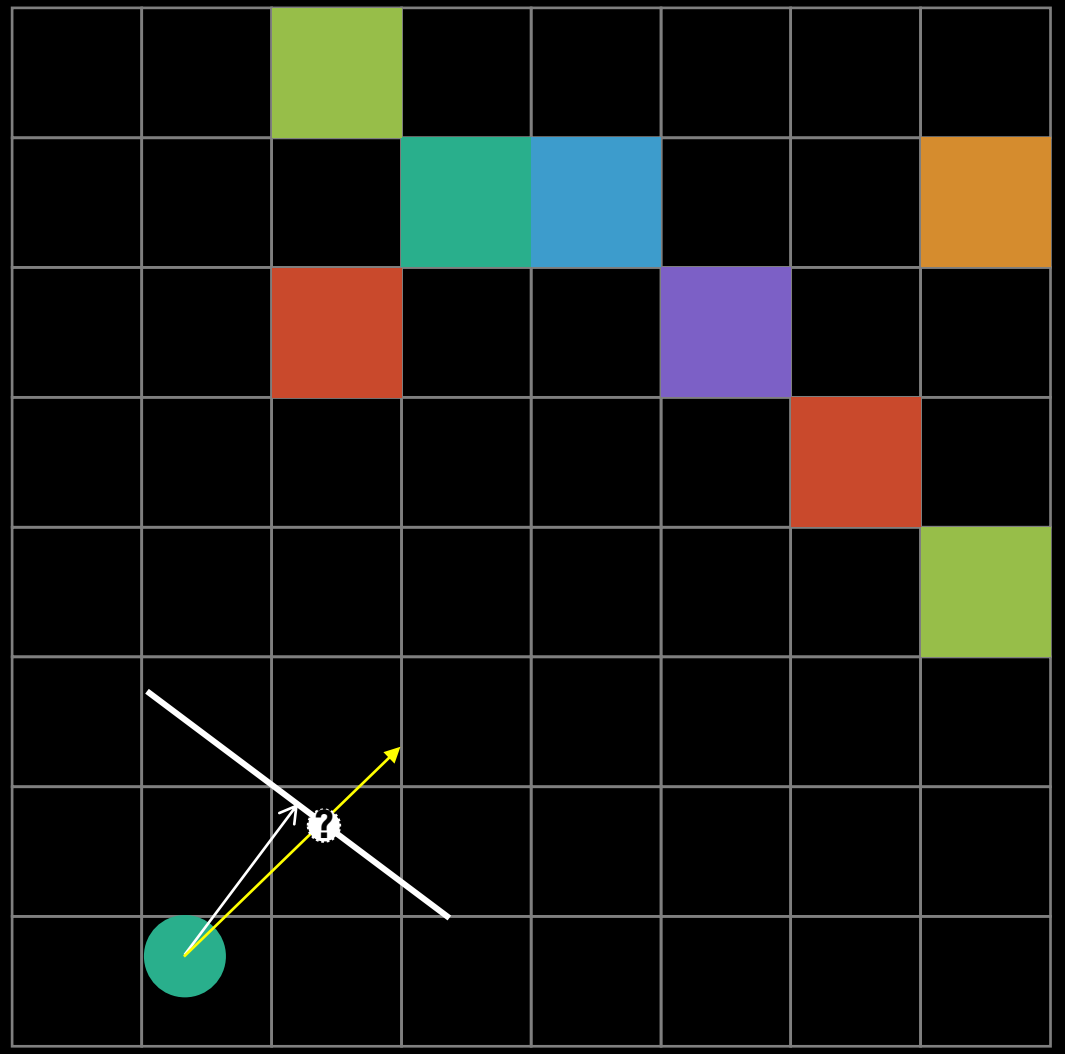
DDA알고리즘

DDA알고리즘을 직관적으로 이해하기



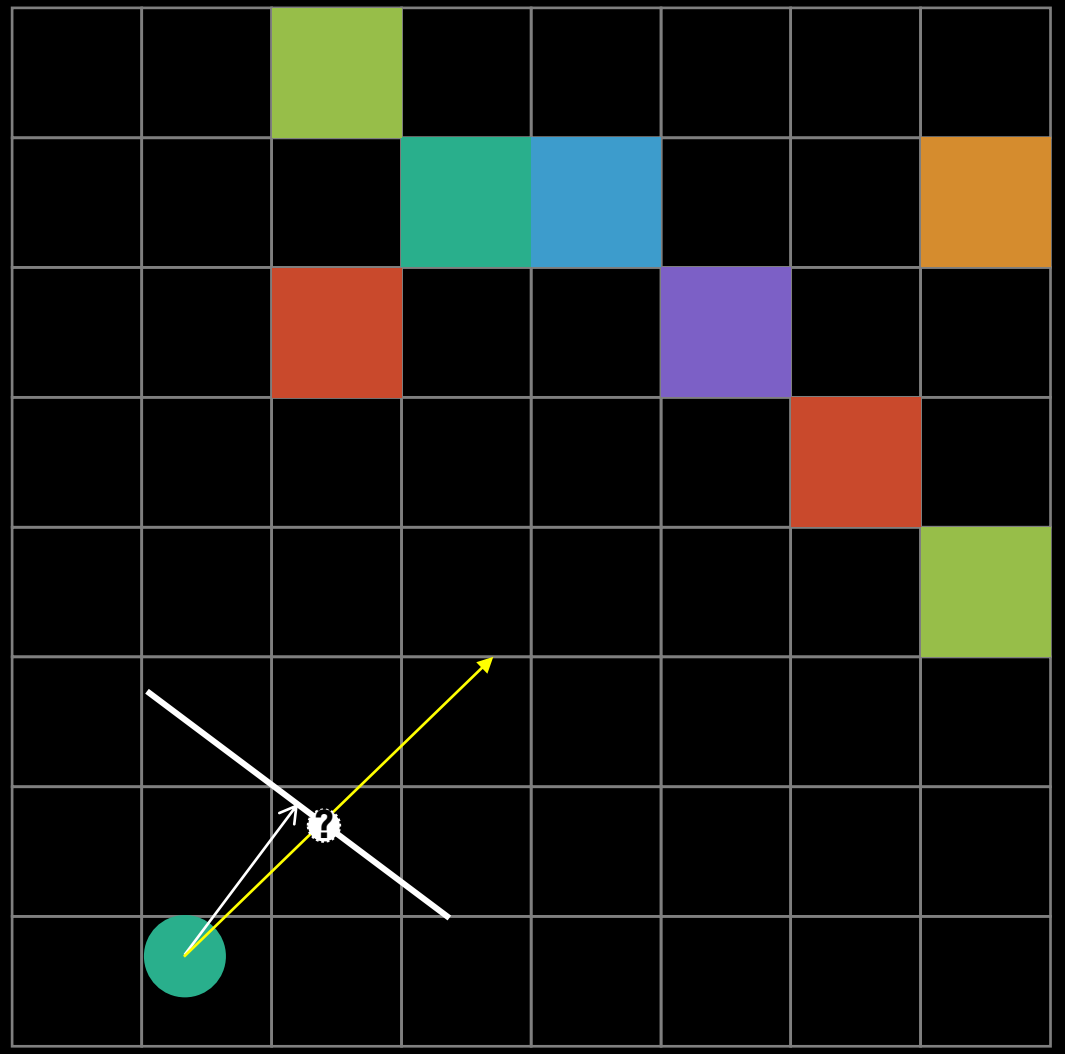
DDA알고리즘

DDA알고리즘을 직관적으로 이해하기



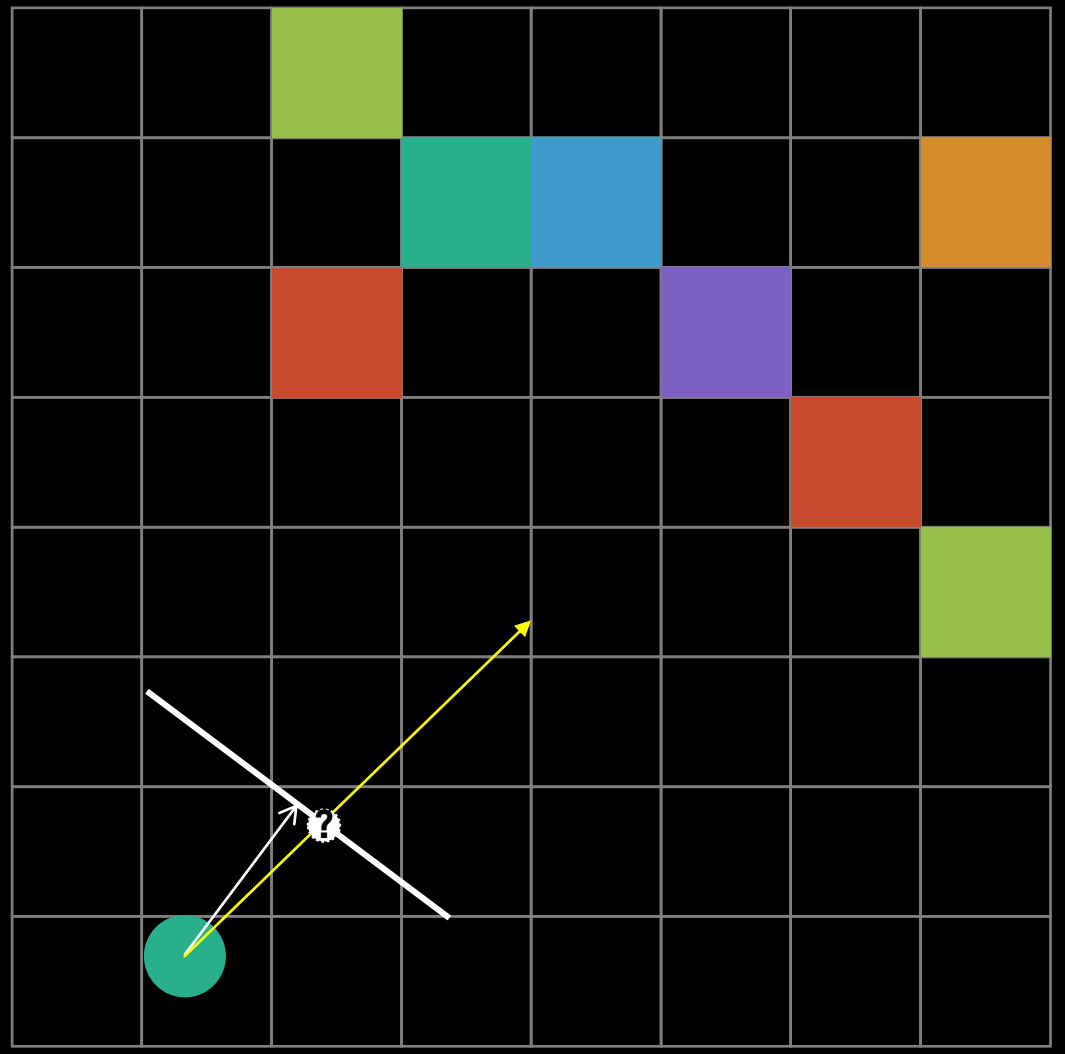
DDA알고리즘

DDA알고리즘을 직관적으로 이해하기



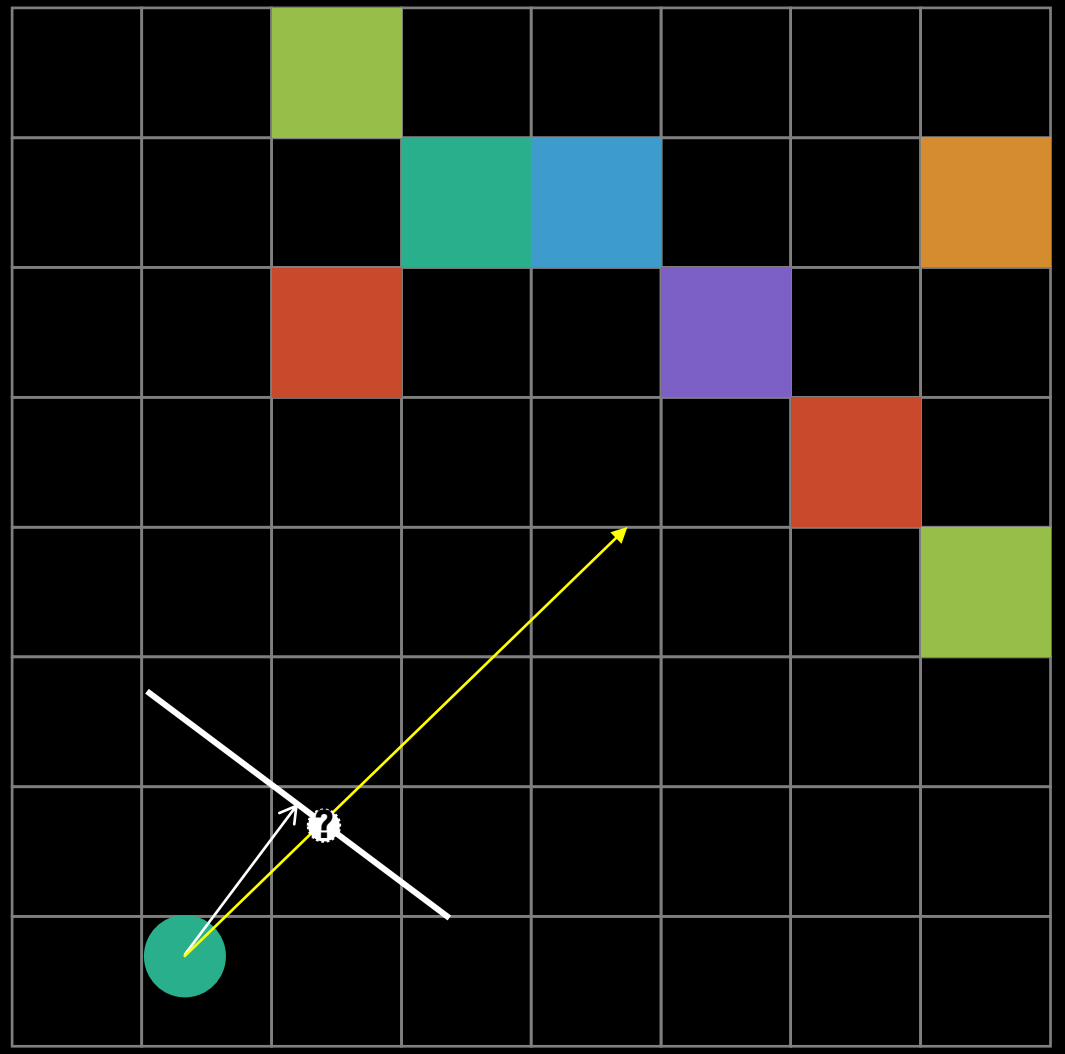
DDA알고리즘

DDA알고리즘을 직관적으로 이해하기



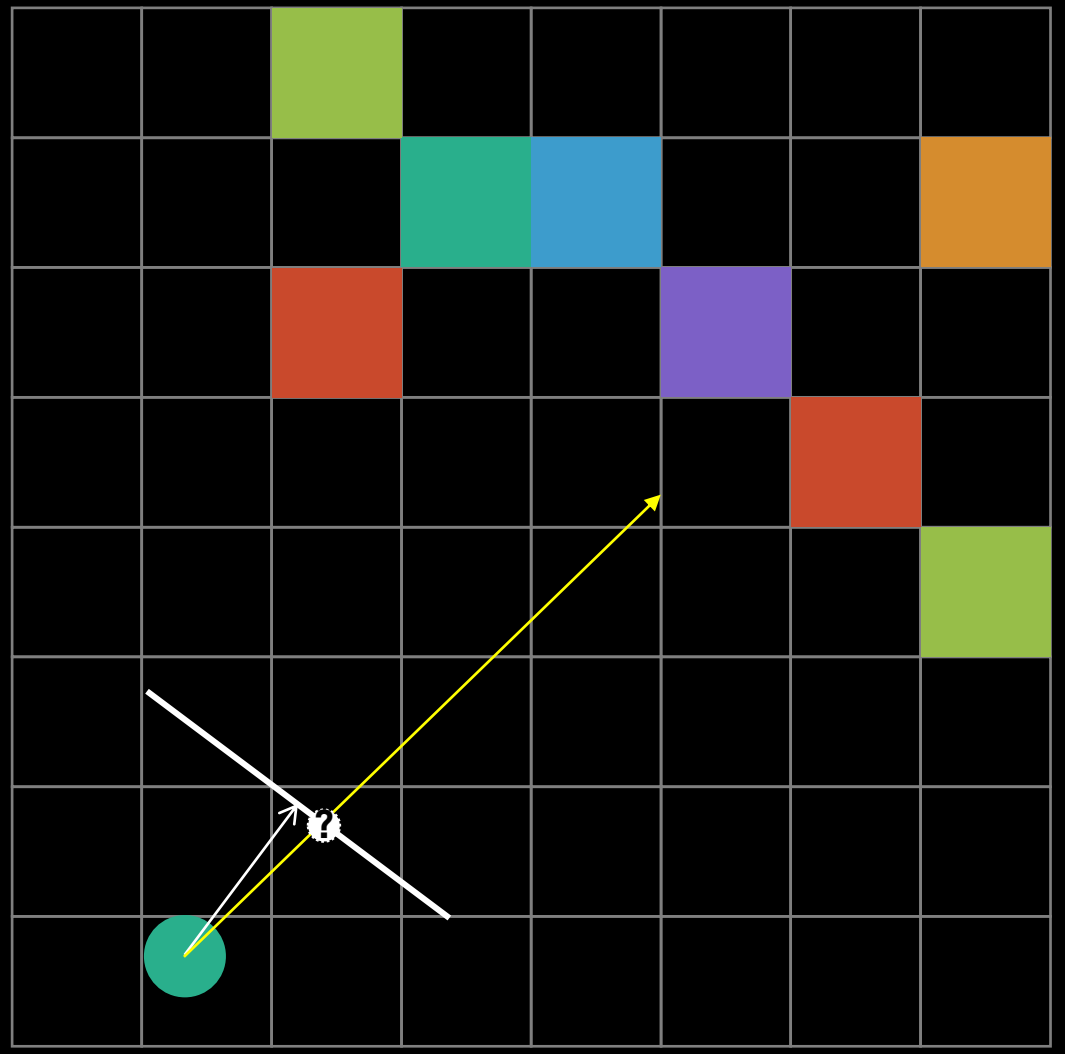
DDA알고리즘

DDA알고리즘을 직관적으로 이해하기



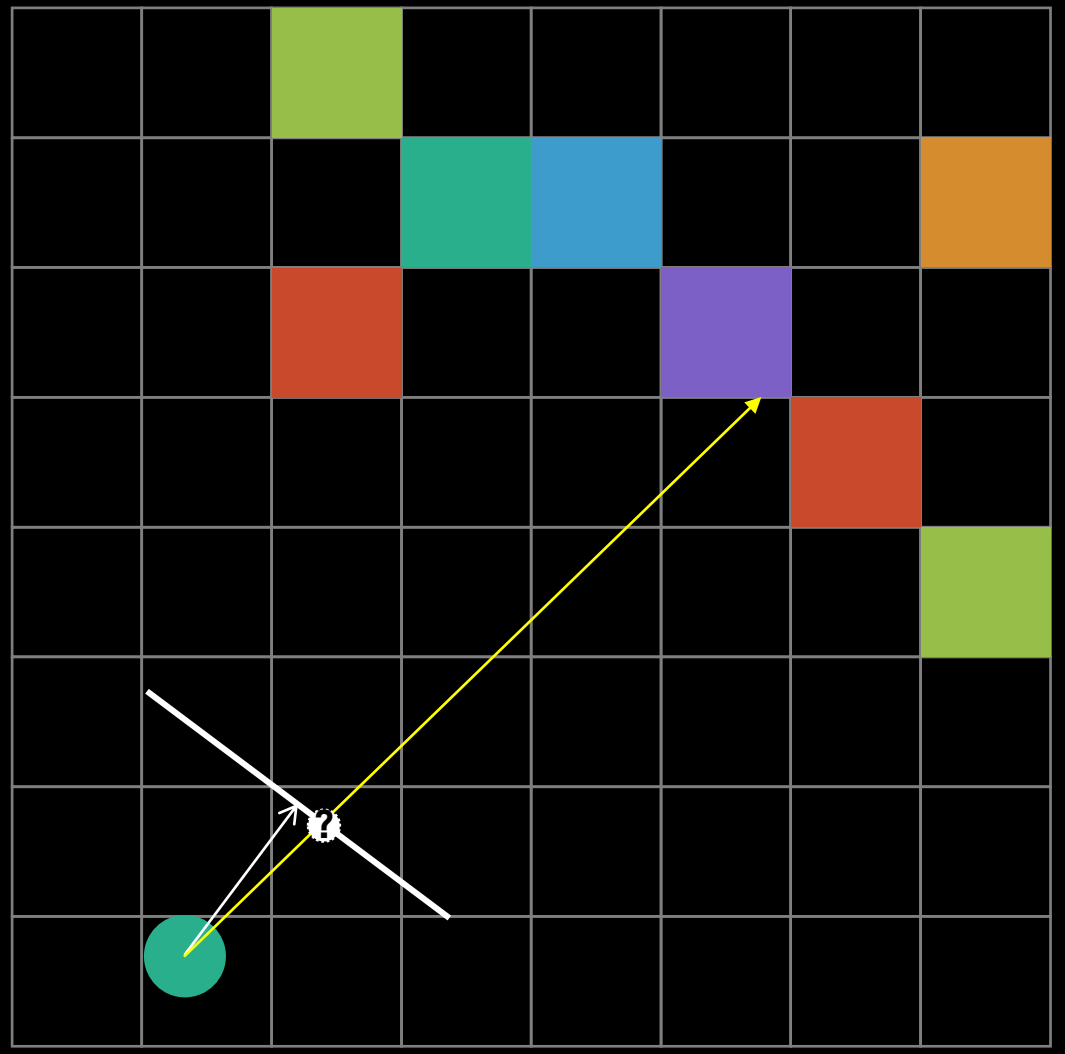
DDA알고리즘

DDA알고리즘을 직관적으로 이해하기



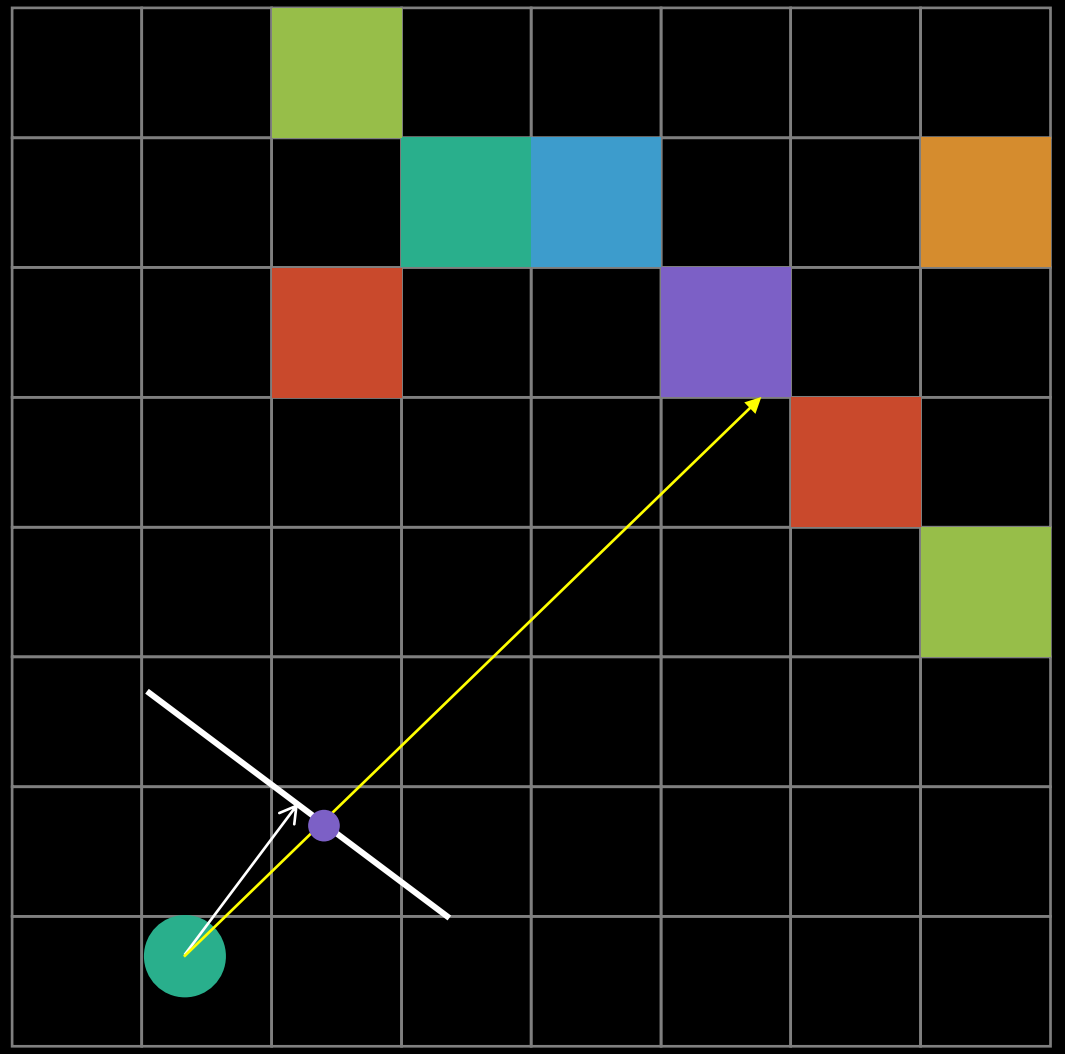
DDA알고리즘

DDA알고리즘을 직관적으로 이해하기



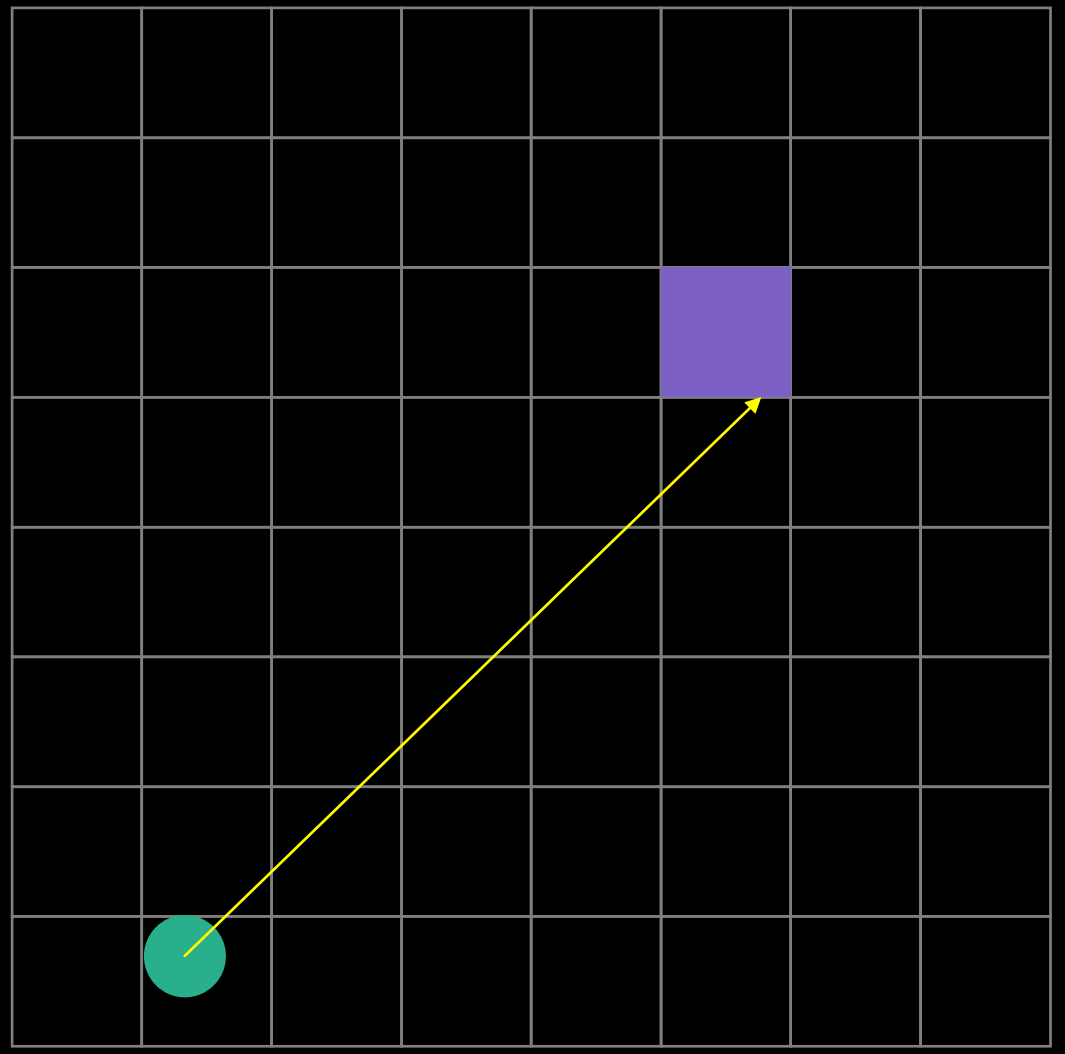
DDA알고리즘

DDA알고리즘을 직관적으로 이해하기



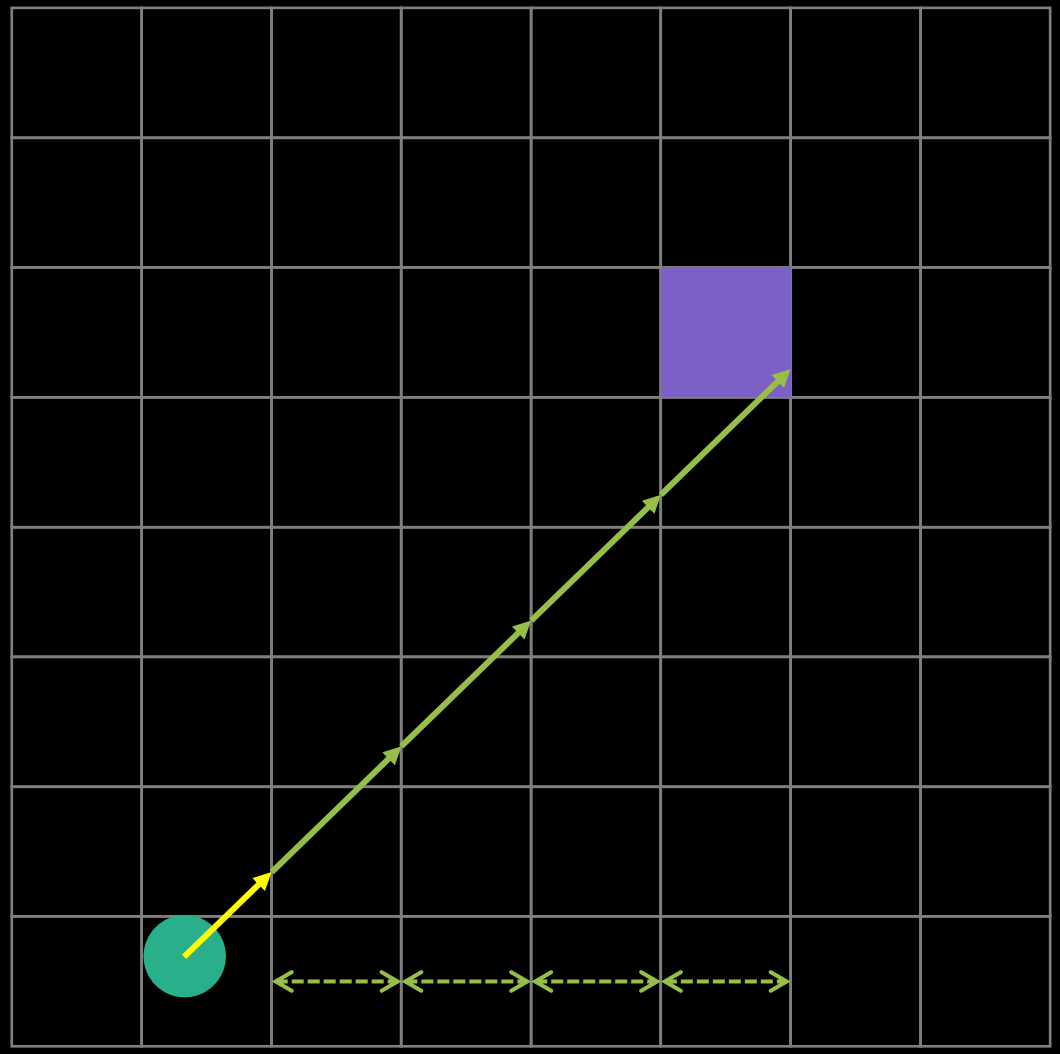
DDA알고리즘

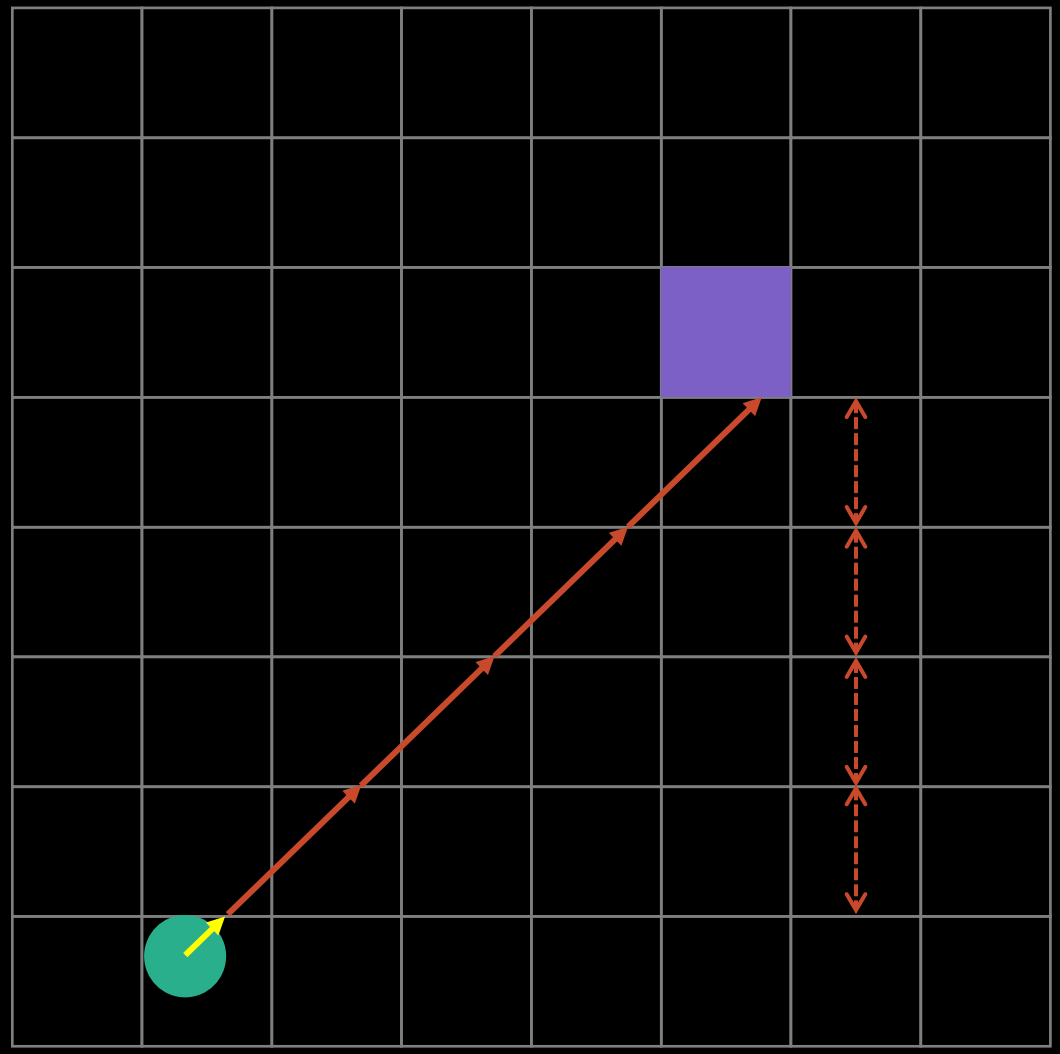
DDA알고리즘 자세히 보기



DDA알고리즘

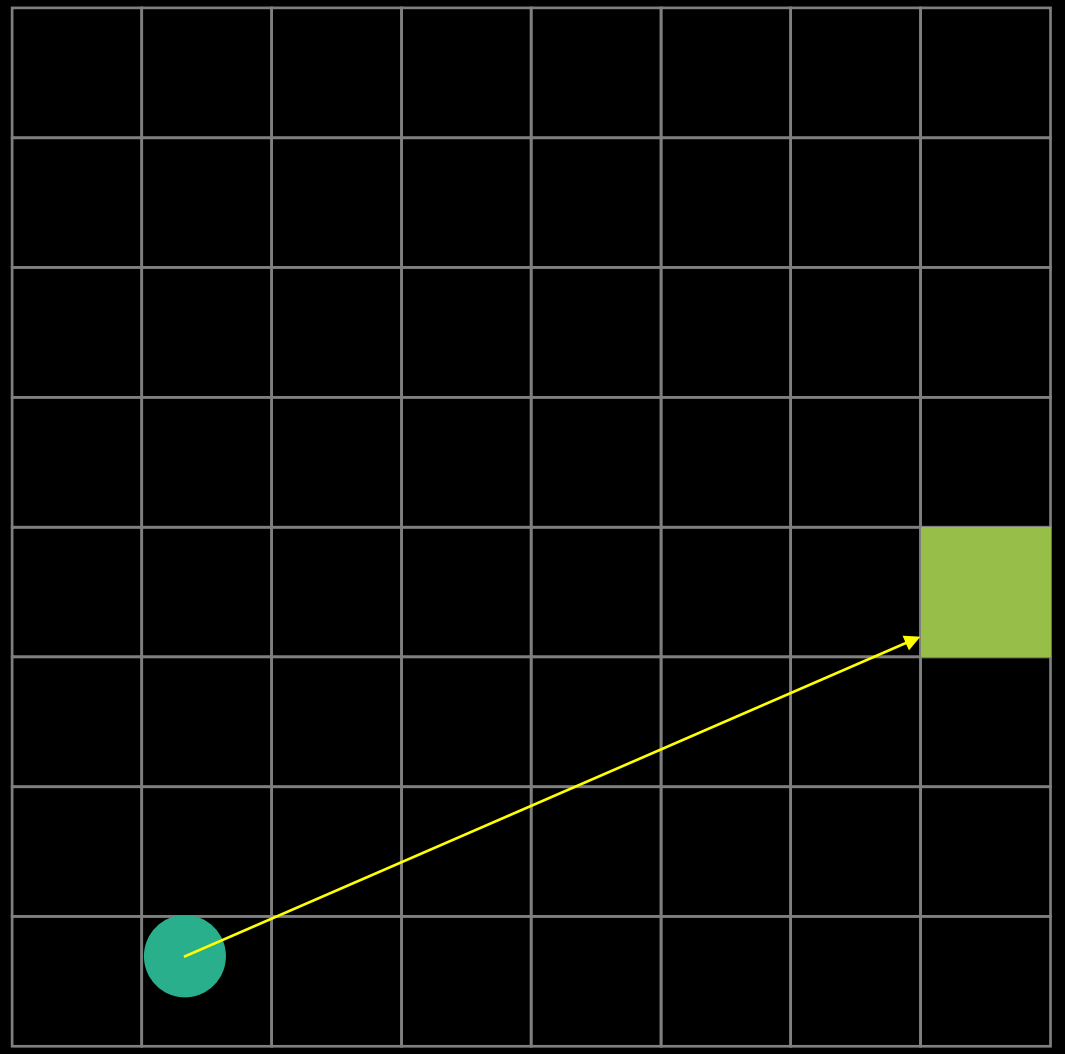
DDA알고리즘 자세히 보기





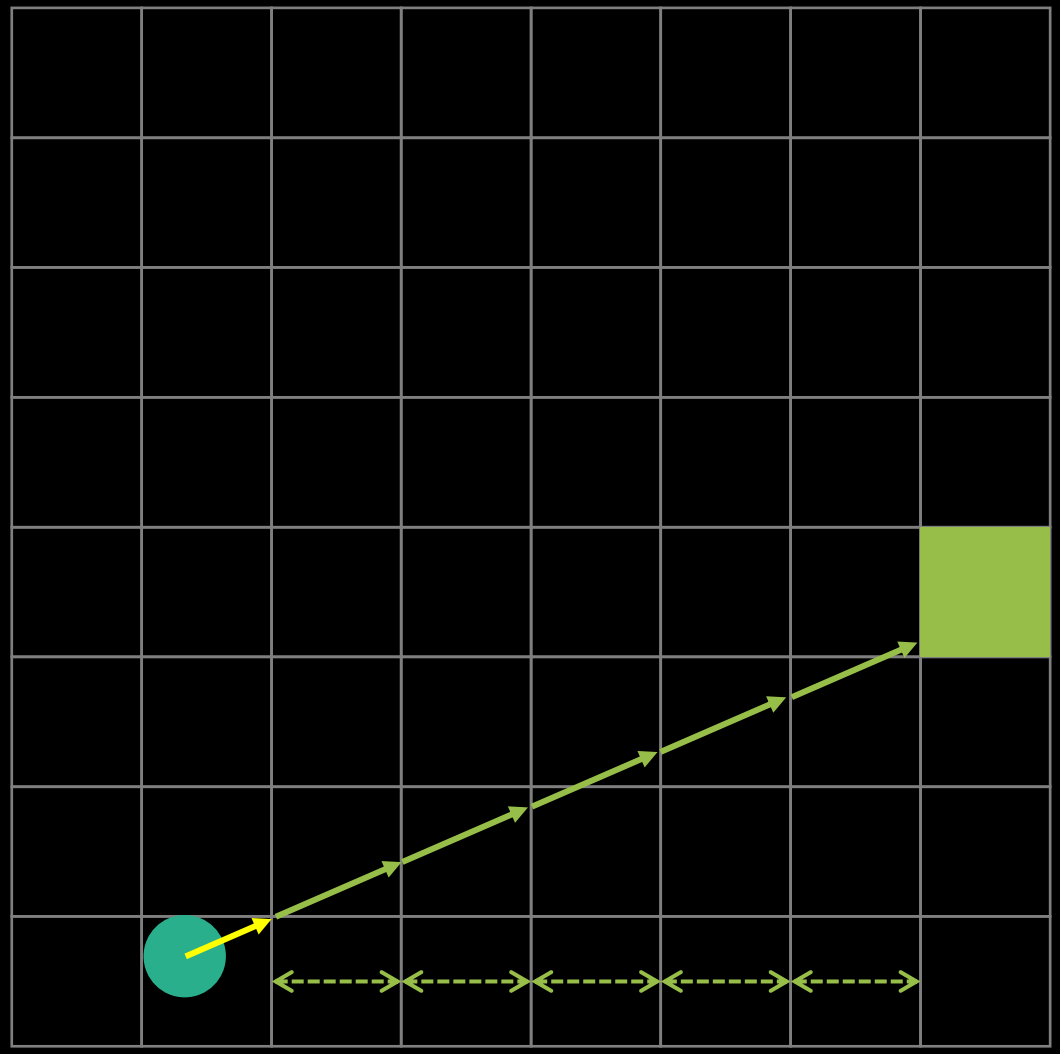
DDA알고리즘

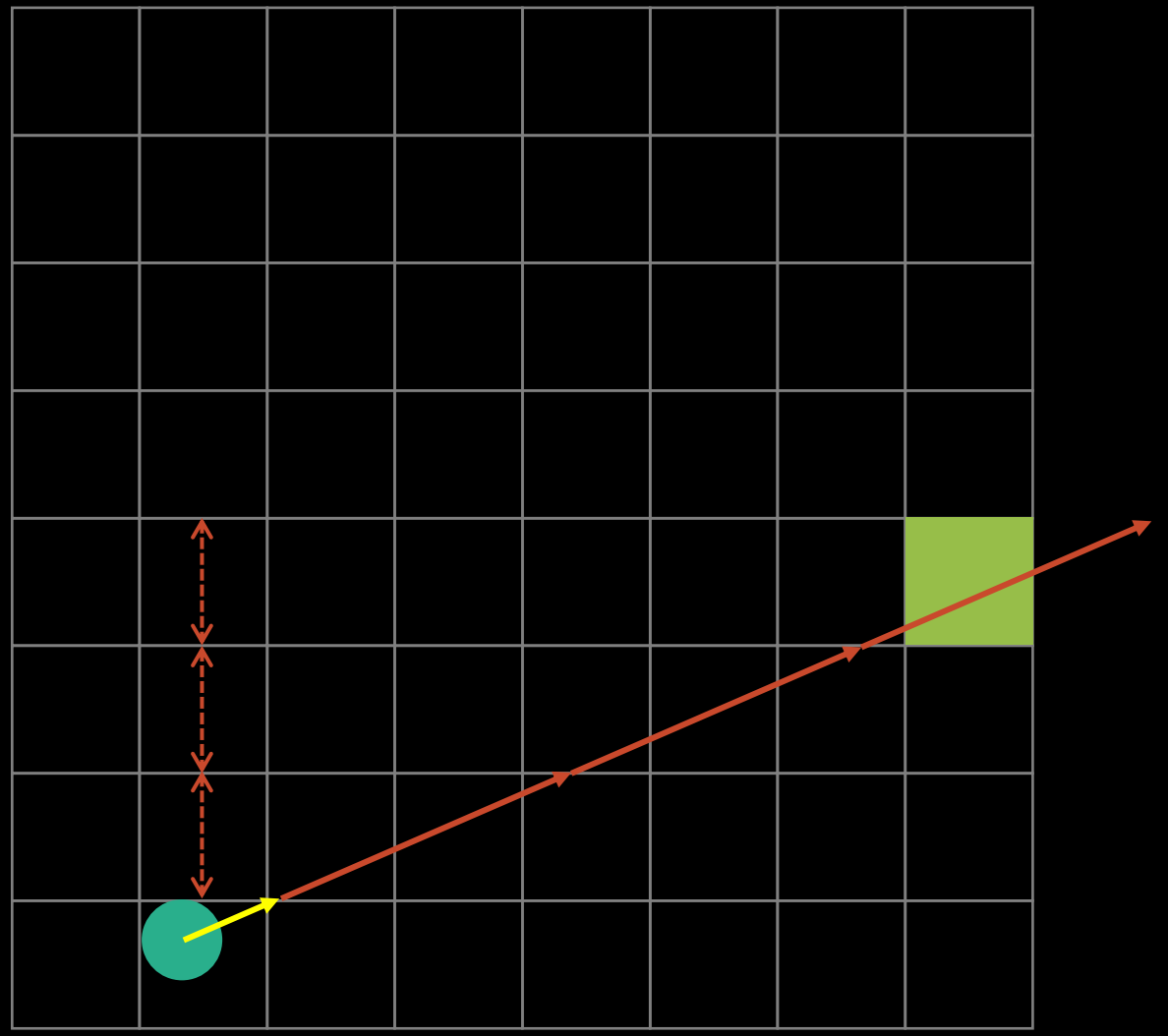
DDA알고리즘 자세히 보기



DDA알고리즘

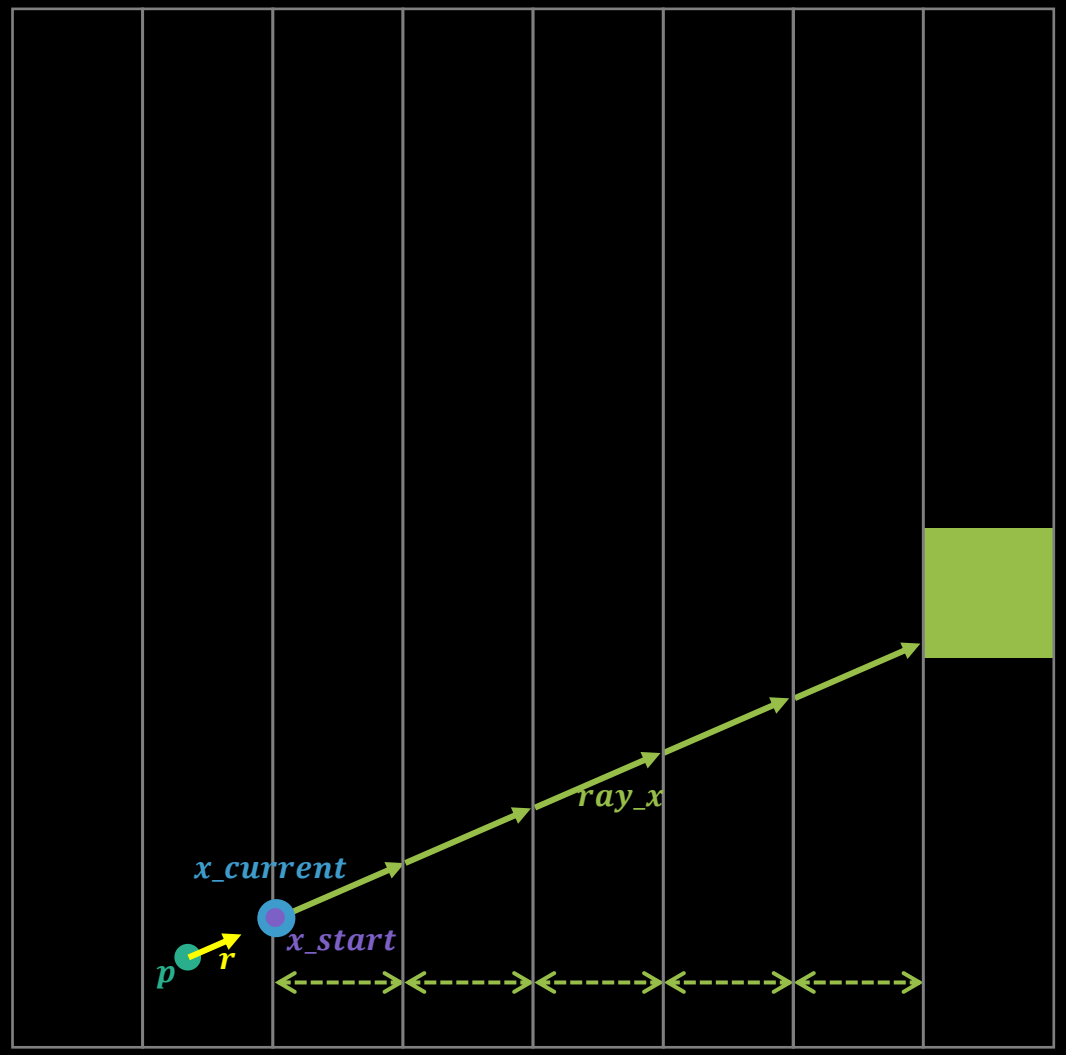
DDA알고리즘 자세히 보기

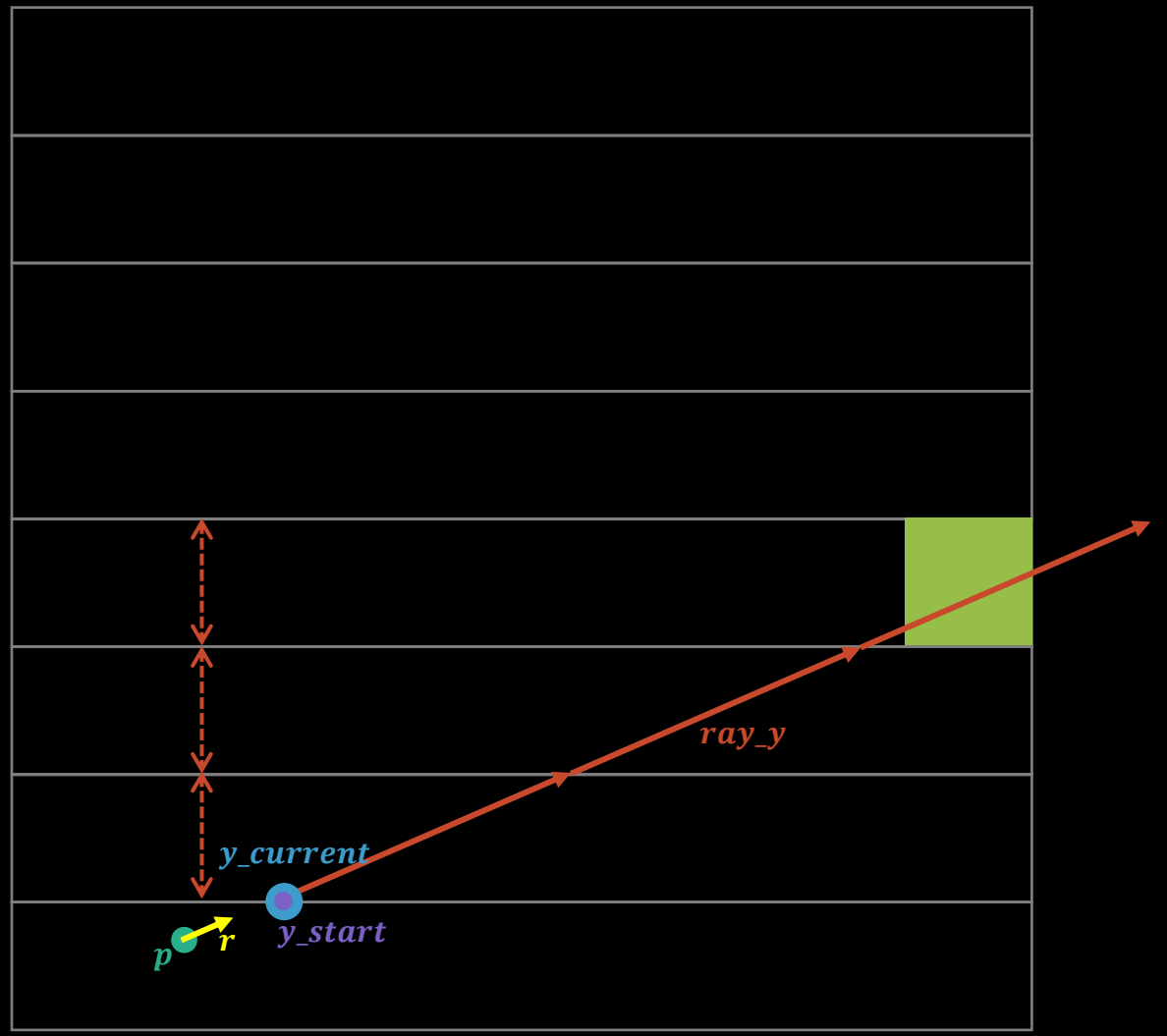




DDA알고리즘

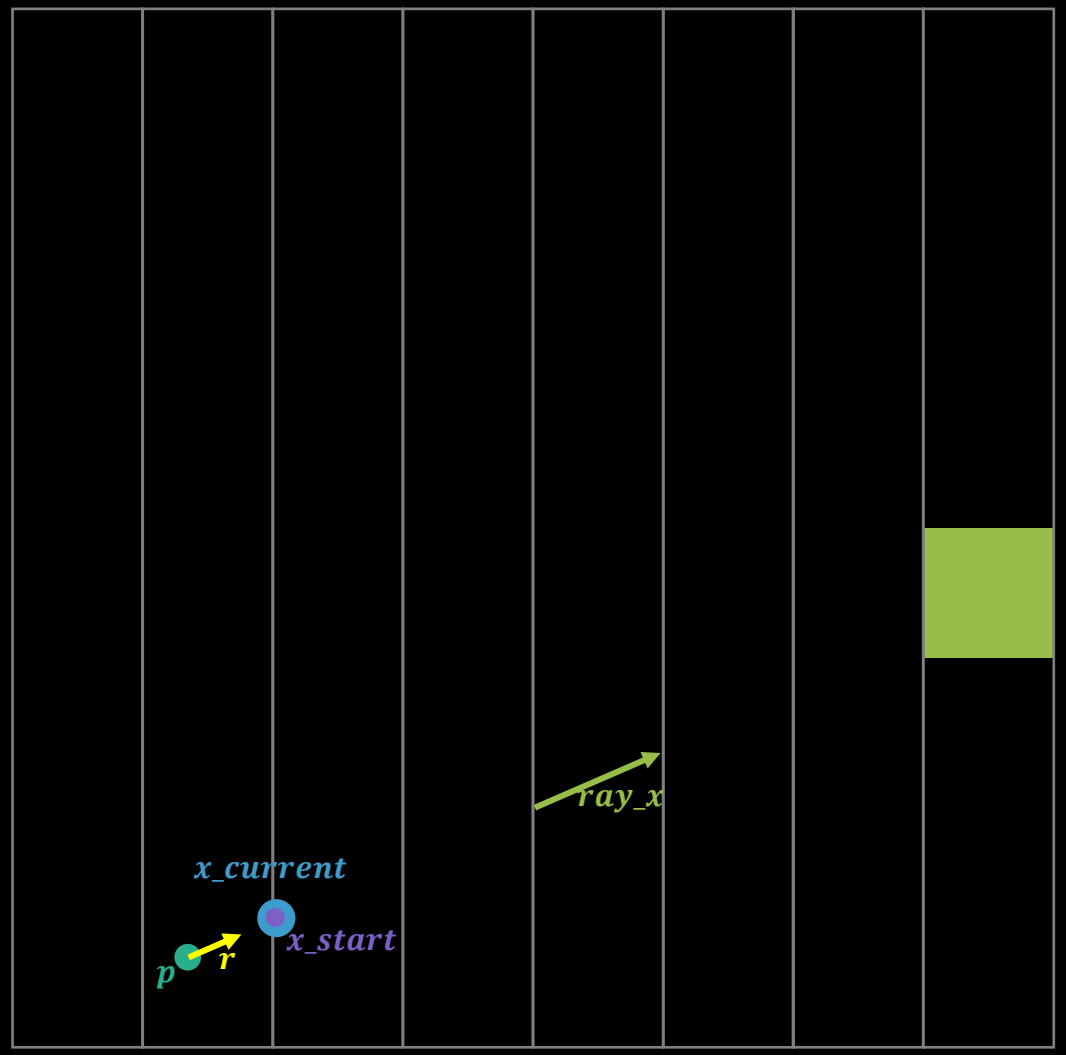
DDA알고리즘 구현하기





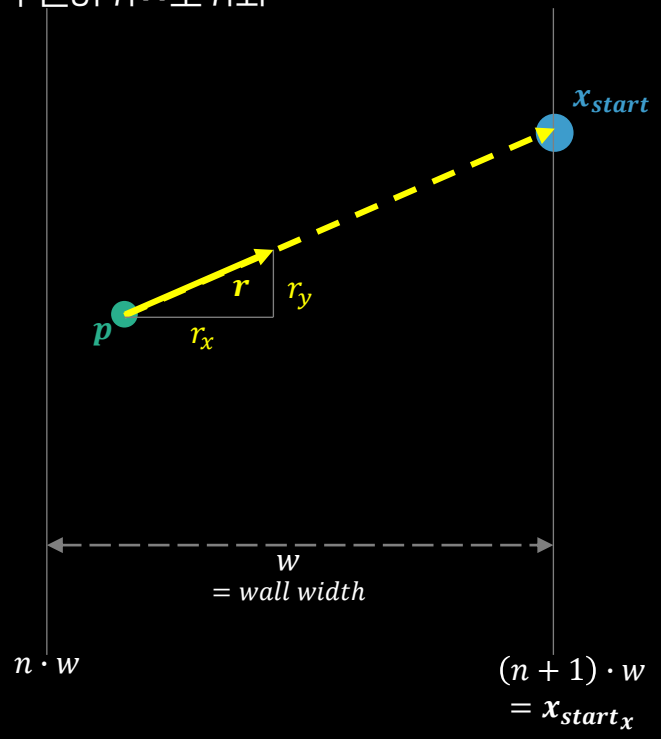
DDA알고리즘

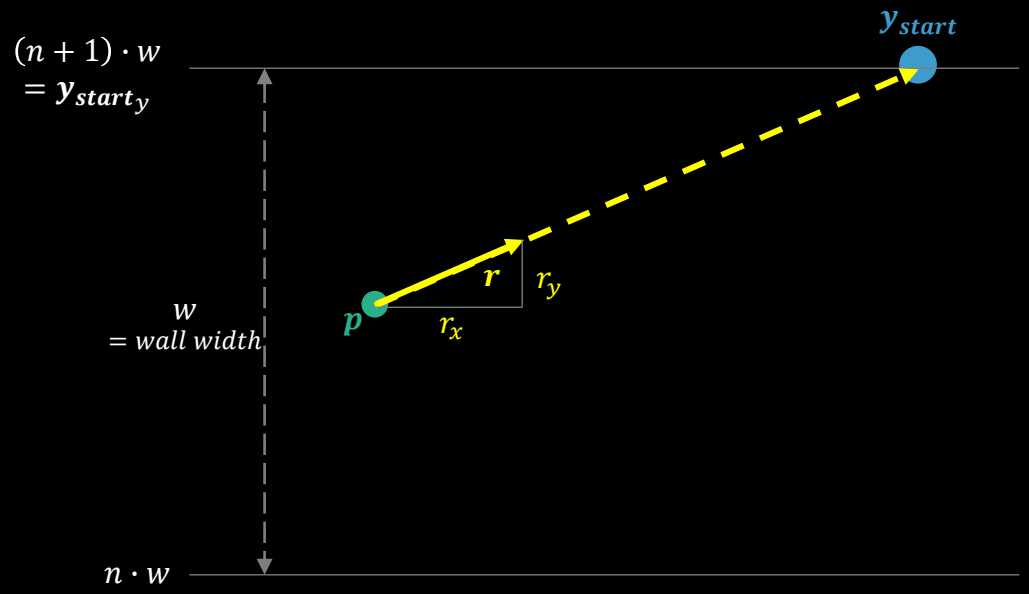
DDA알고리즘 구현하기::초기화



DDA알고리즘

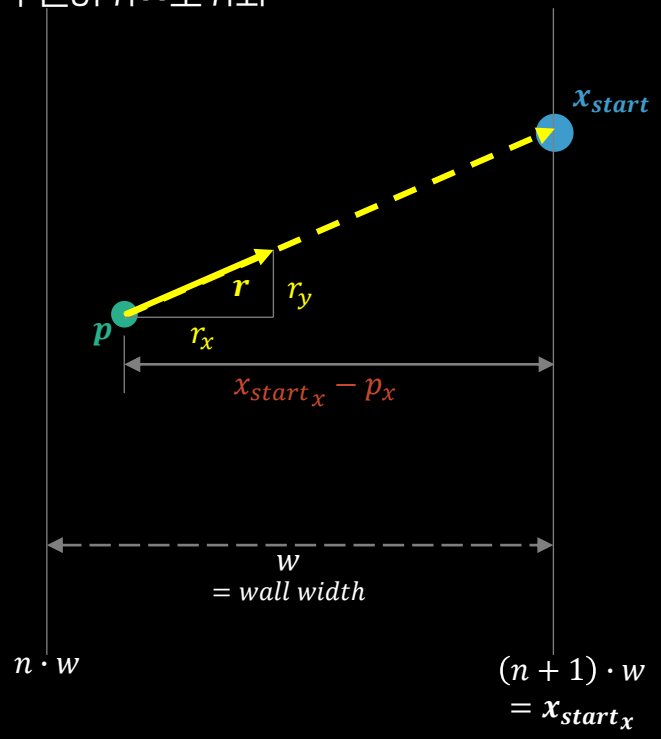
DDA알고리즘 구현하기::초기화



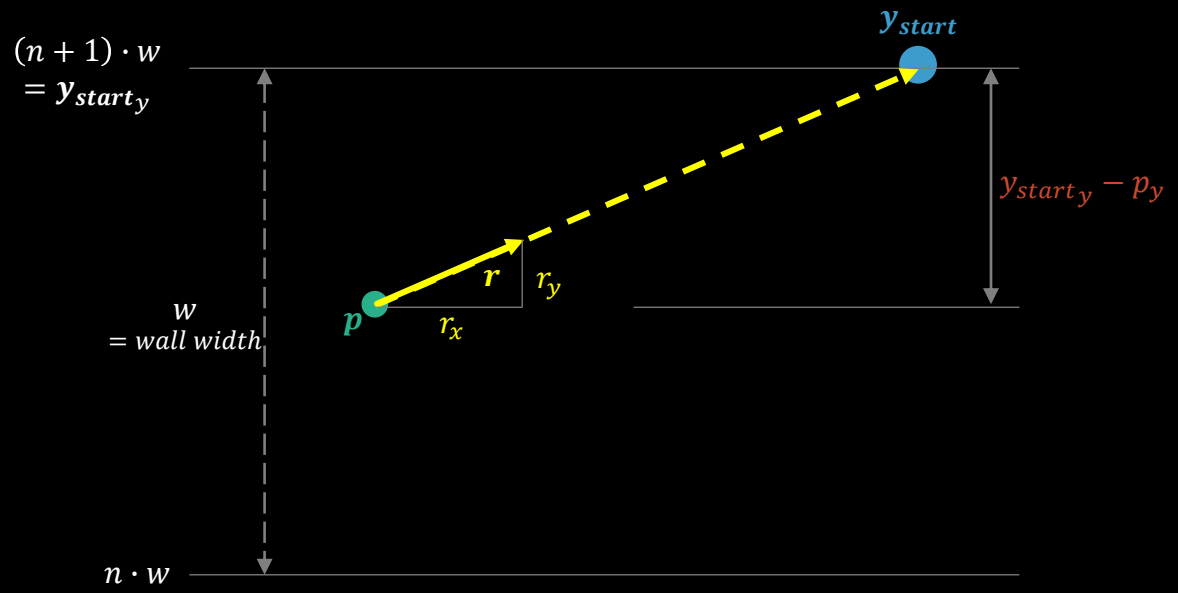


DDA알고리즘

DDA알고리즘 구현하기::초기화



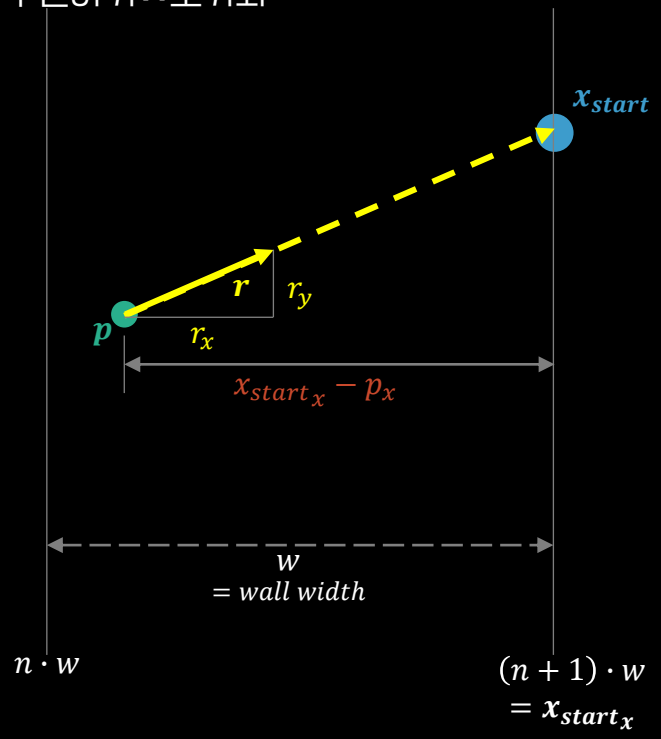
$n = (int)p_x \div w$

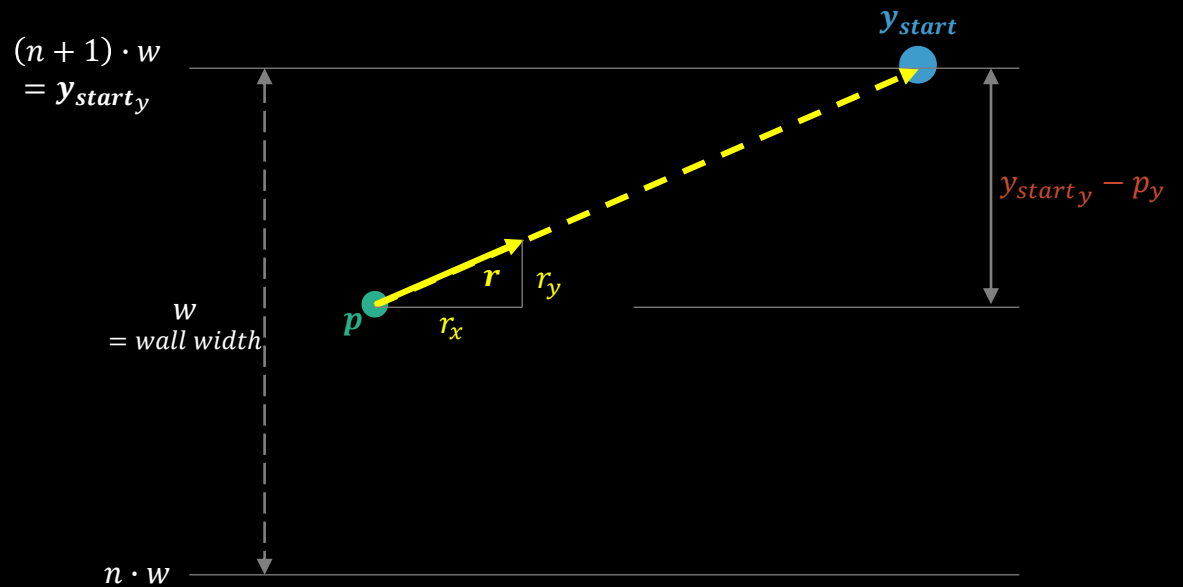


$n = (int)p_y \div w$

DDA알고리즘

DDA알고리즘 구현하기::초기화





$n = (int)p_x \div w$

$if\ (r.x > 0)$

$x_{start_x} = (n+1) * w$

$else$

$x_{start_x} = n * w$

$n = (int)p_y \div w$

$if\ (r.y > 0)$

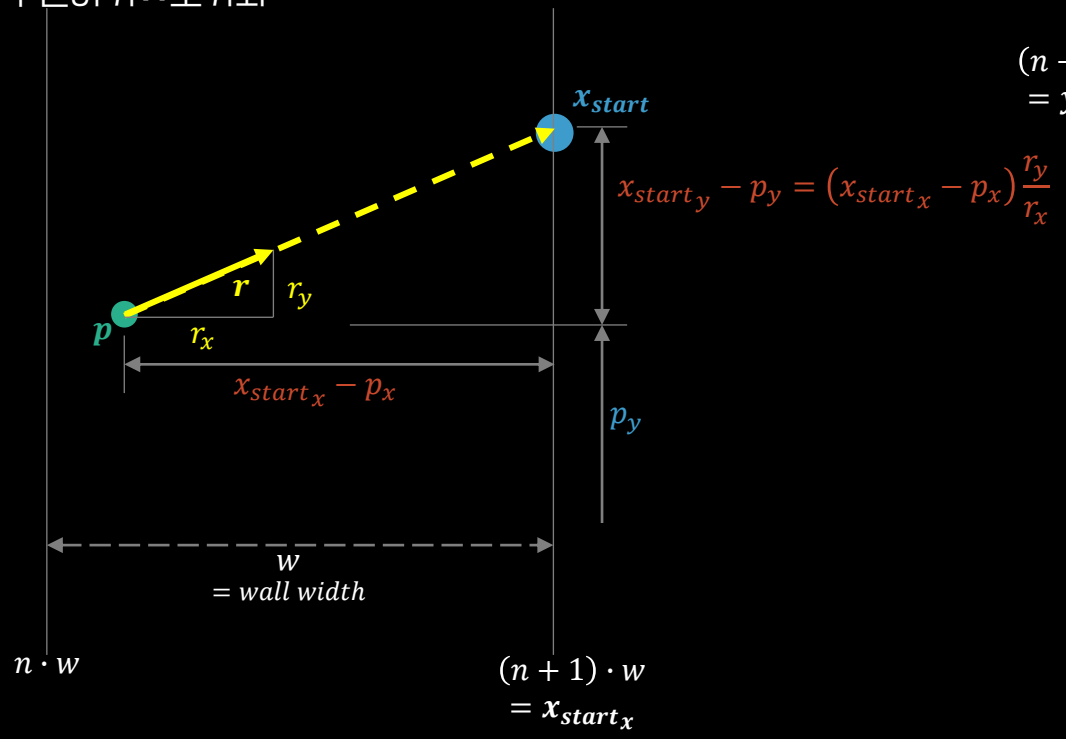
$y_{start_y} = (n+1) * w$

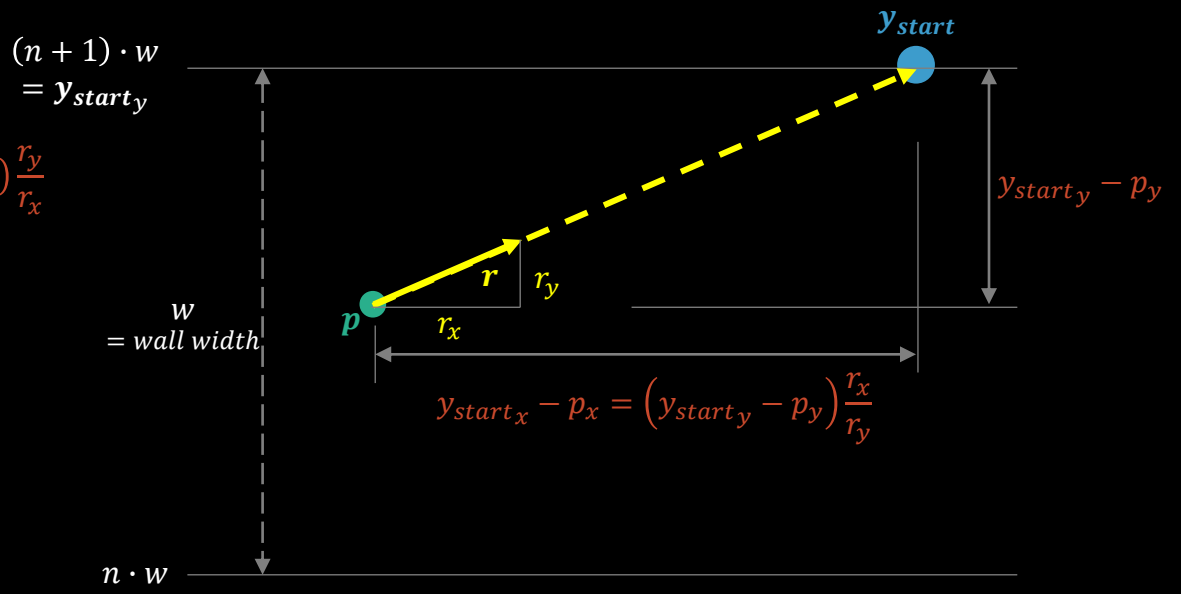
$else$

$y_{start_y} = n * w$

DDA알고리즘

DDA알고리즘 구현하기::초기화





$n = (int)p_x \div w$

$if\ (r.x > 0)$

$\quad x_{start_x} = (n+1) * w$

$else$

$\quad x_{start_x} = n * w$

$x_{start_y} = (x_{start_x} - p_x)\frac{r_y}{r_x} + p_y$

$n = (int)p_y \div w$

$if\ (r.y > 0)$

$\quad y_{start_y} = (n+1) * w$

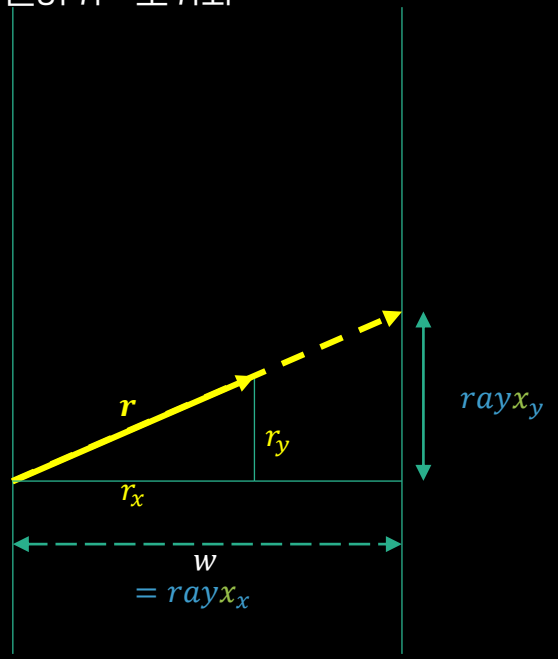
$else$

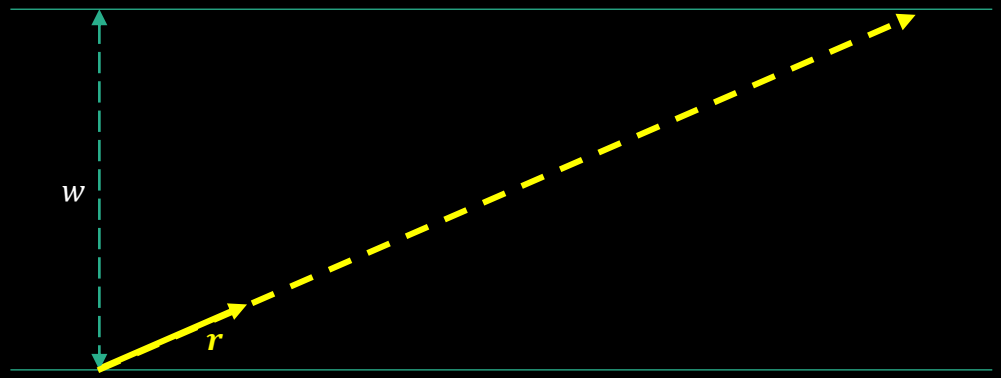
$\quad y_{start_y} = n * w$

$y_{start_x} = (y_{start_y} - p_y)\frac{r_x}{r_y} + p_x$

DDA알고리즘

DDA알고리즘 구현하기::초기화





$$rayx_x = w$$

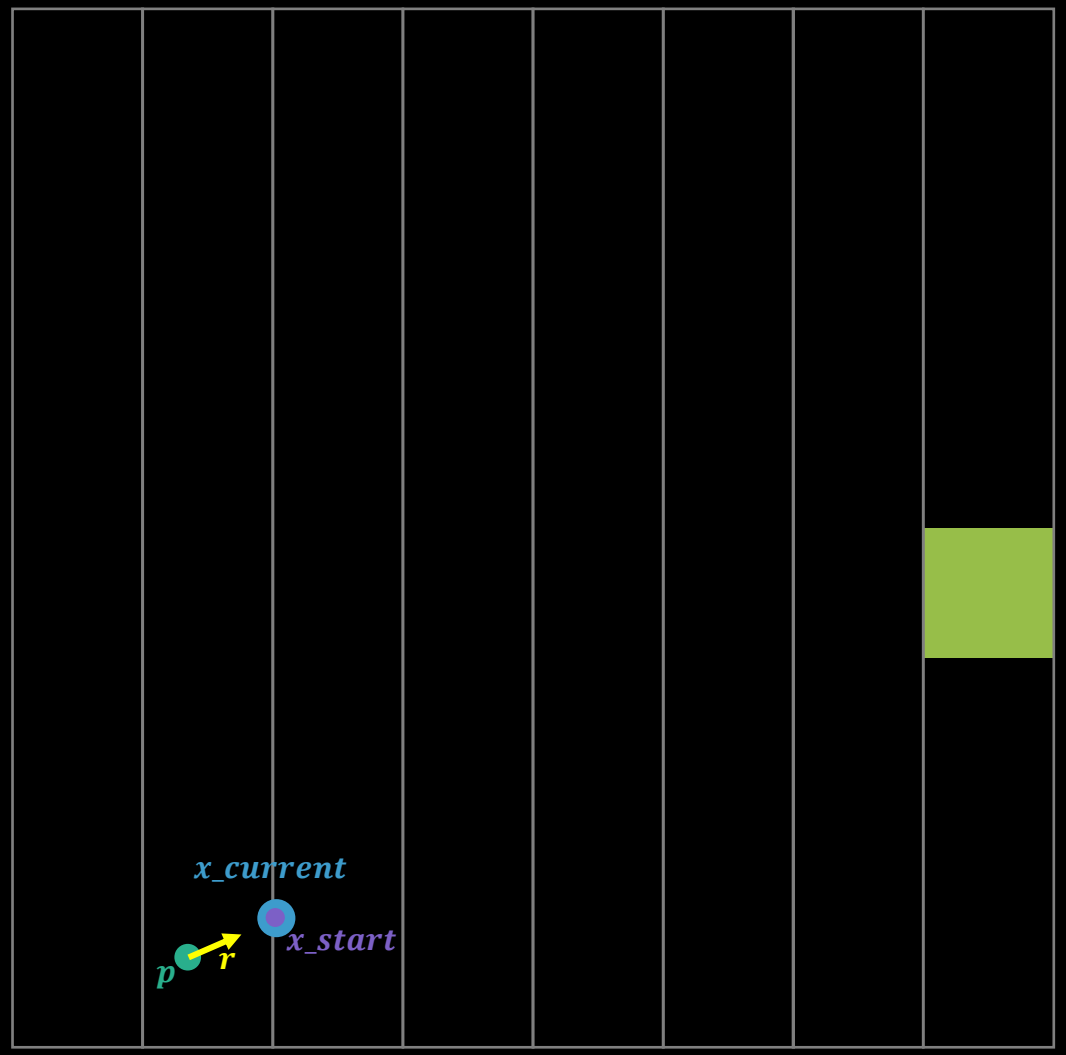
$$rayx_y = w\frac{r_y}{r_x}$$

$$rayx_y = w$$

$$rayx_x = w\frac{r_x}{r_y}$$

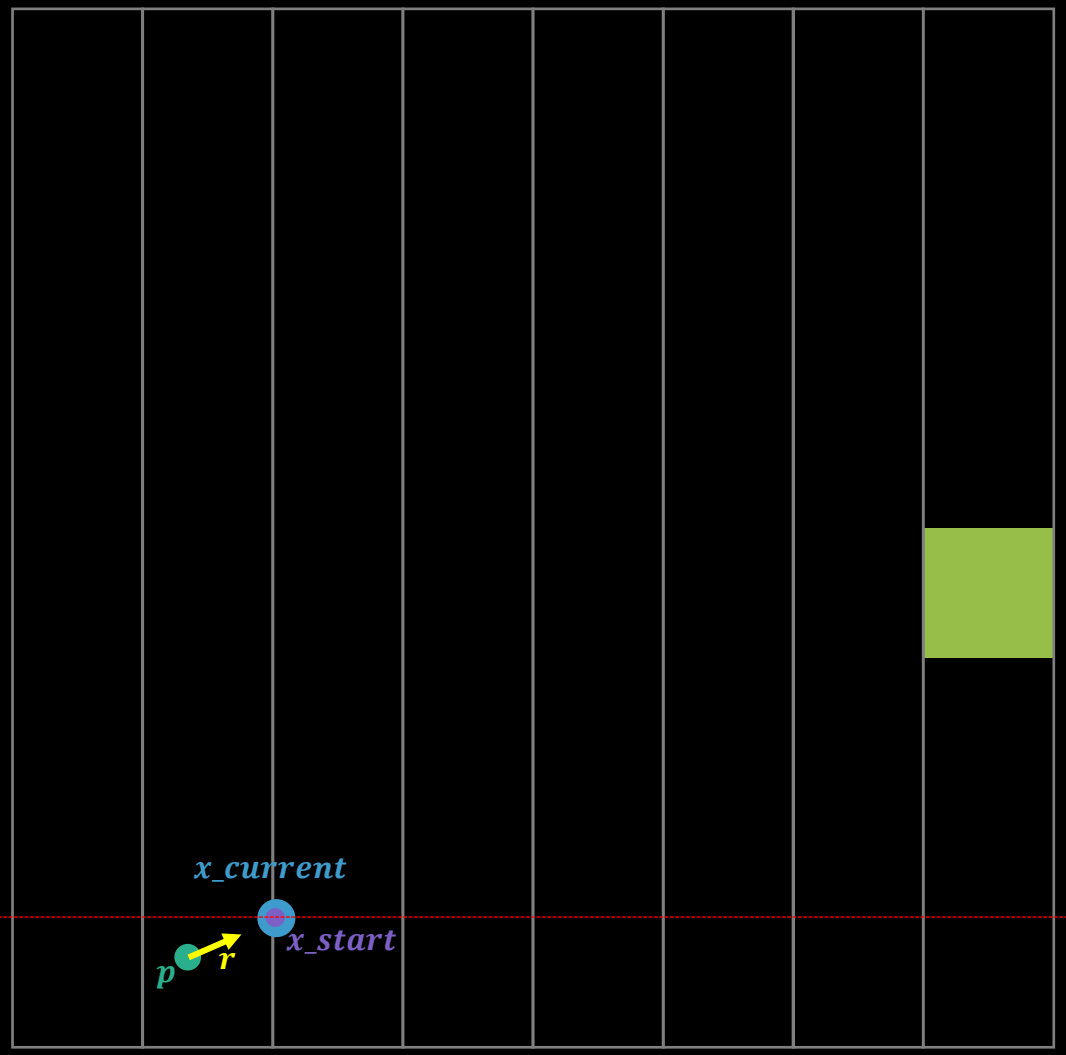
DDA알고리즘

DDA알고리즘 구현하기::시작



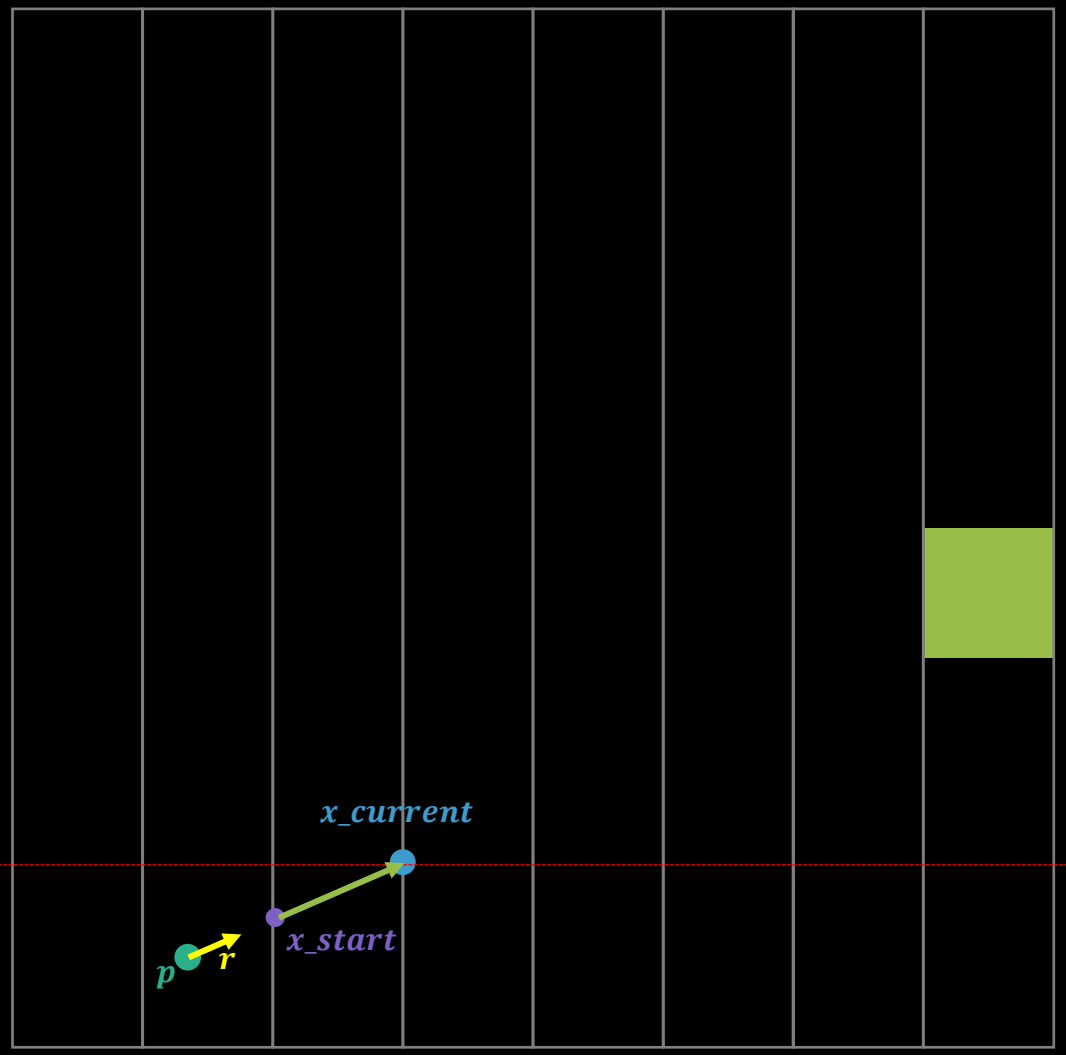
DDA알고리즘

DDA알고리즘 구현하기::반복구간



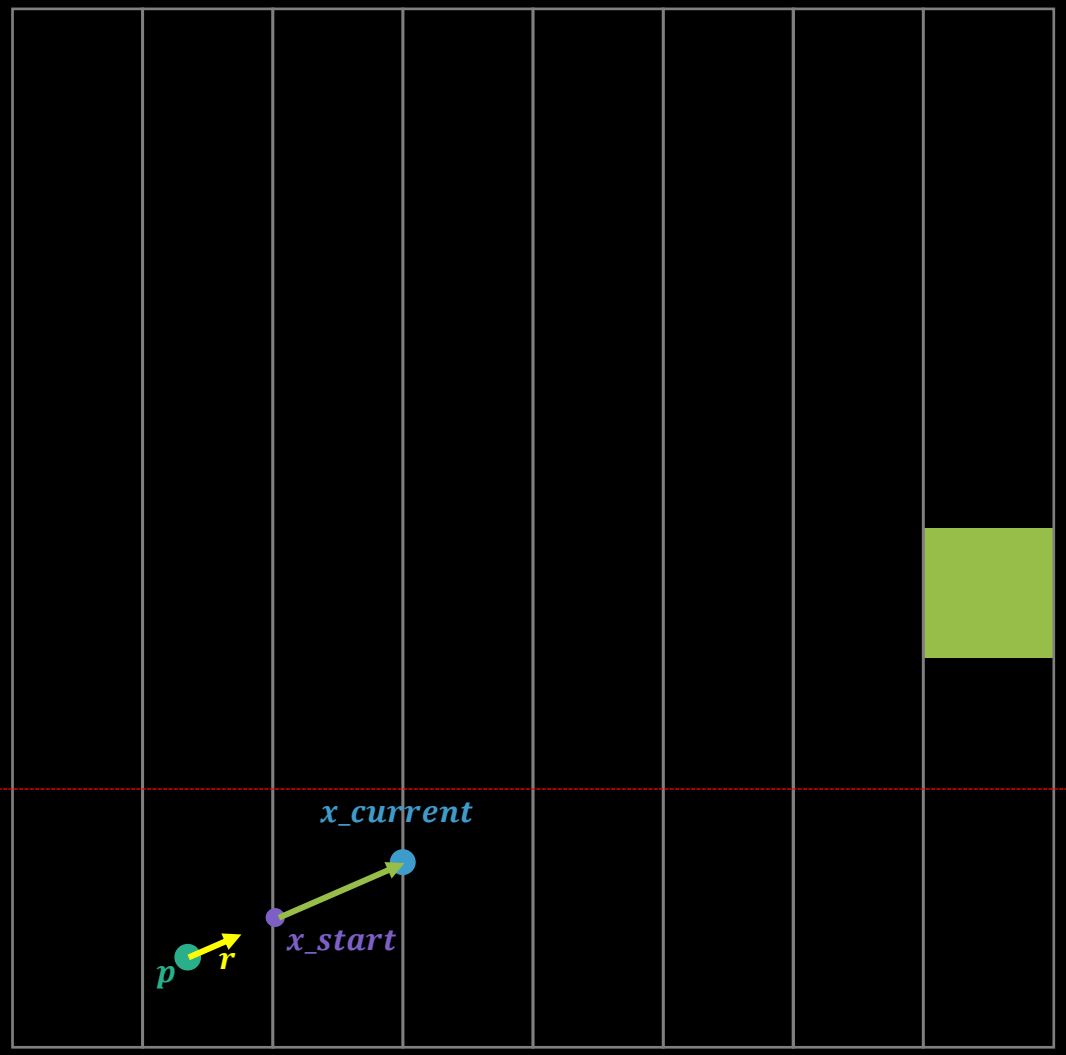
DDA알고리즘

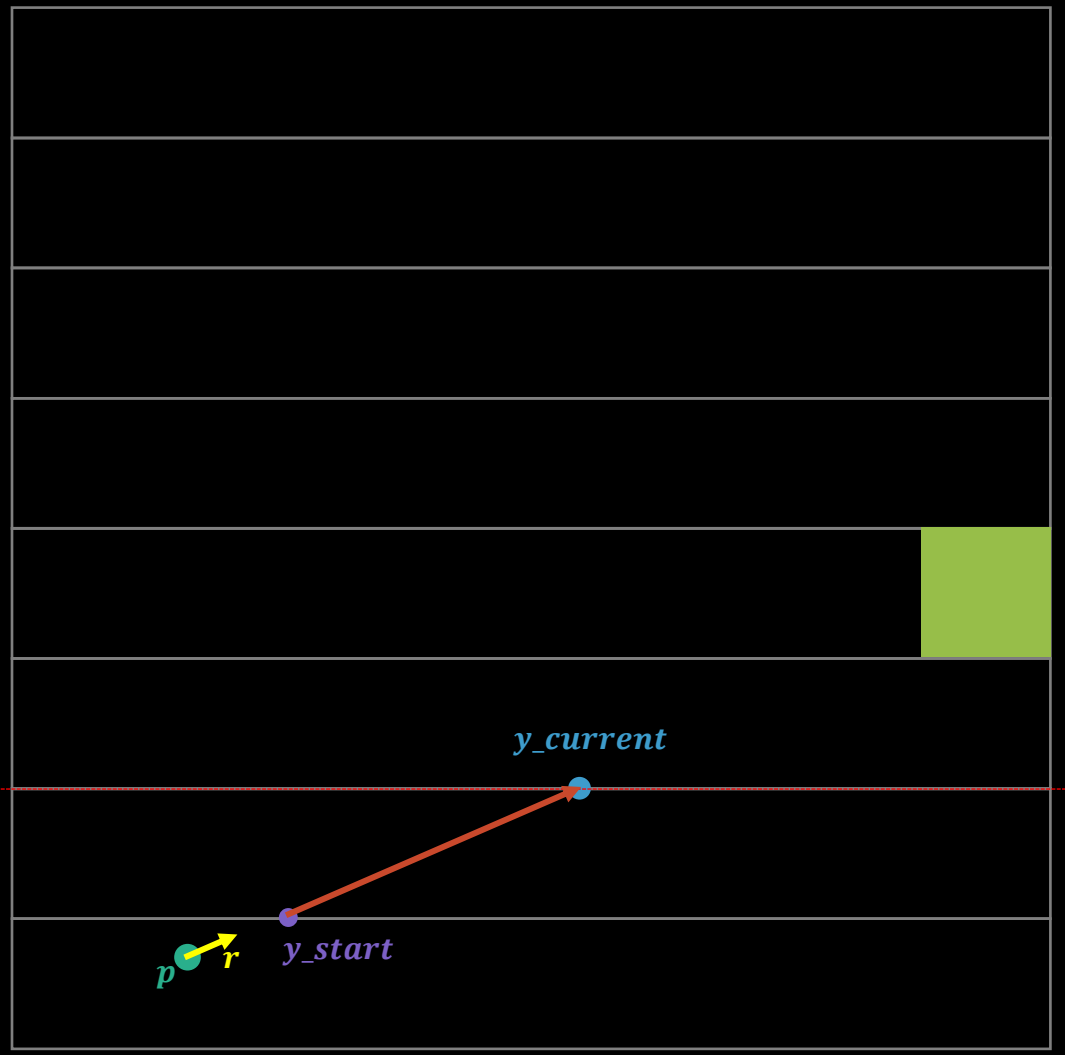
DDA알고리즘 구현하기::반복구간



DDA알고리즘

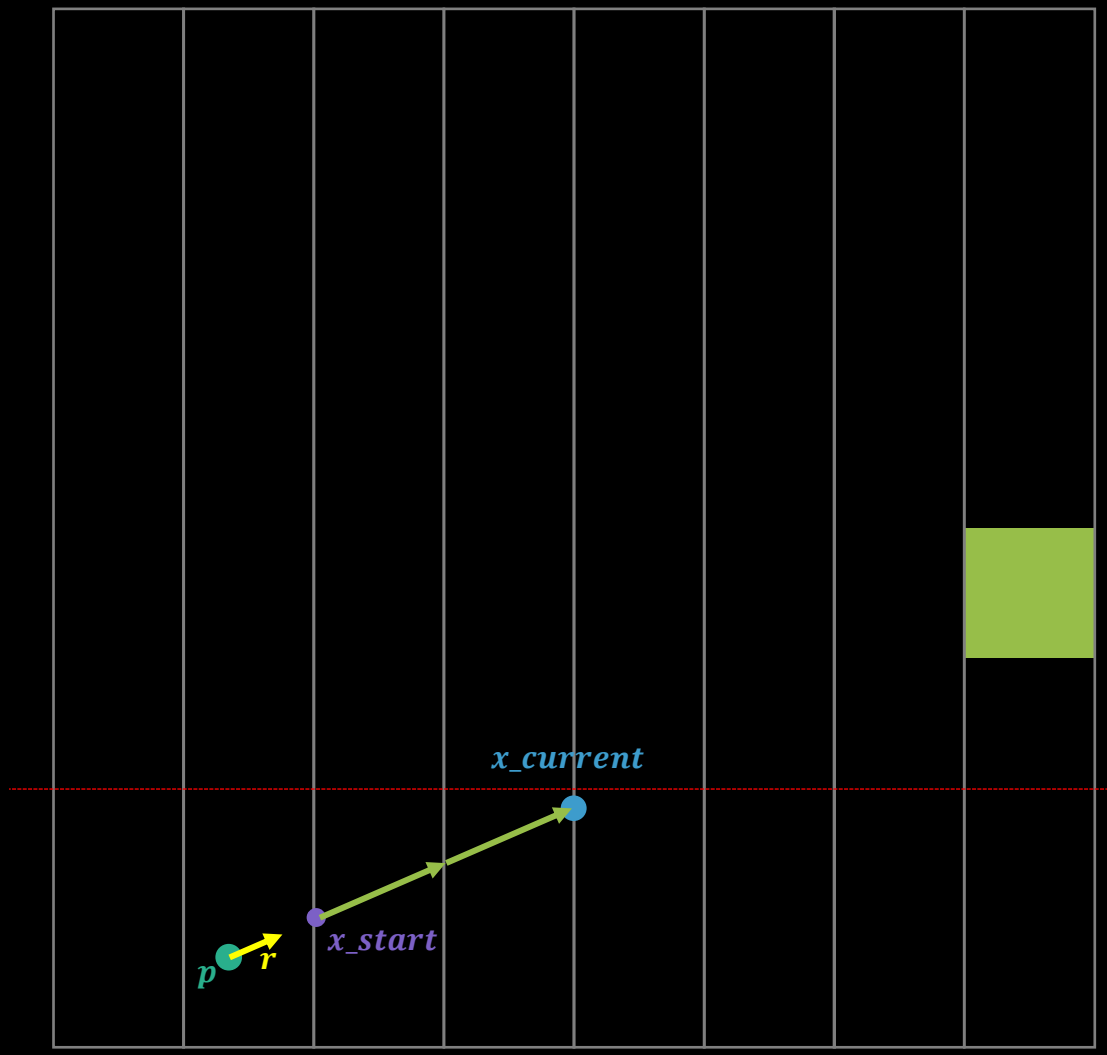
DDA알고리즘 구현하기::반복구간

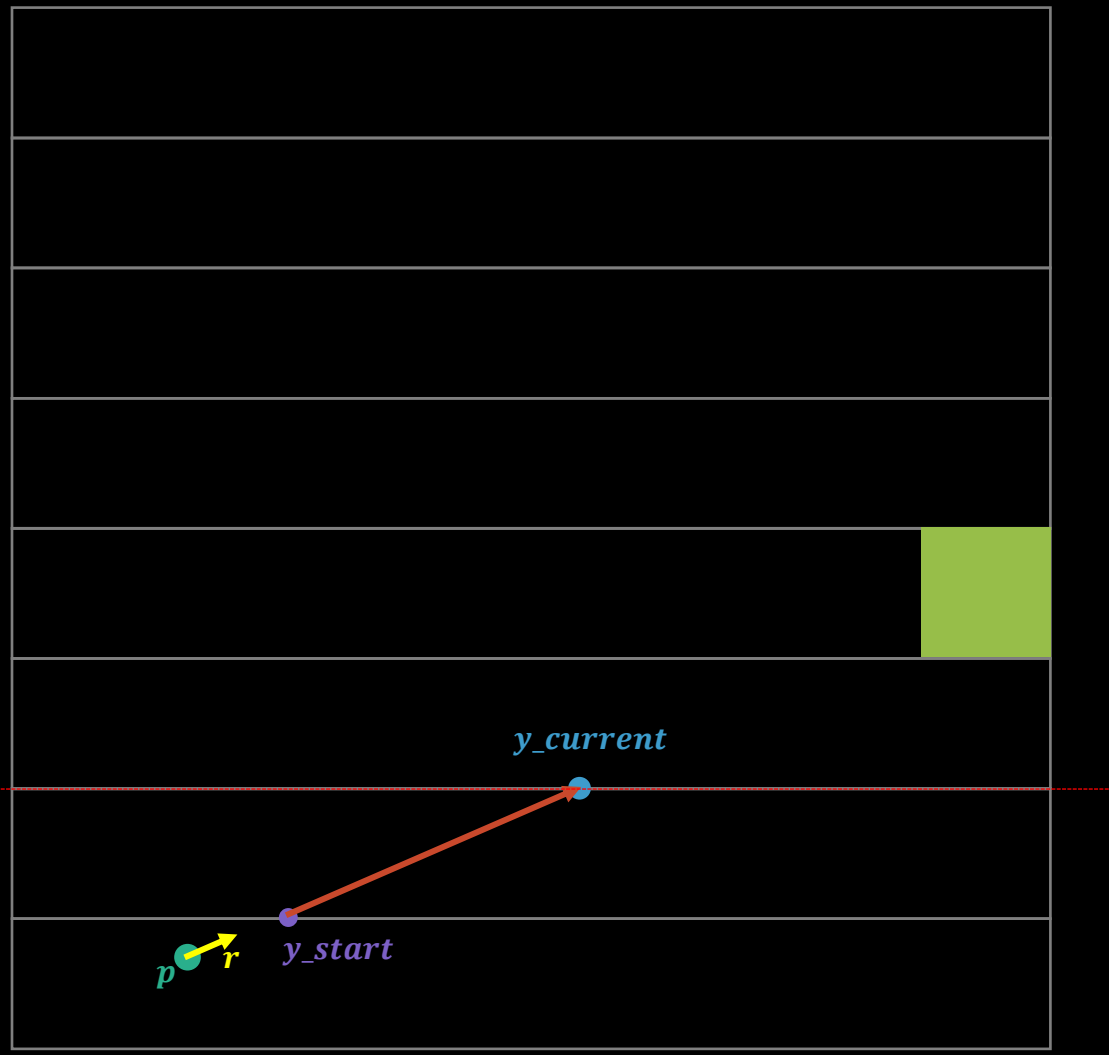




DDA알고리즘

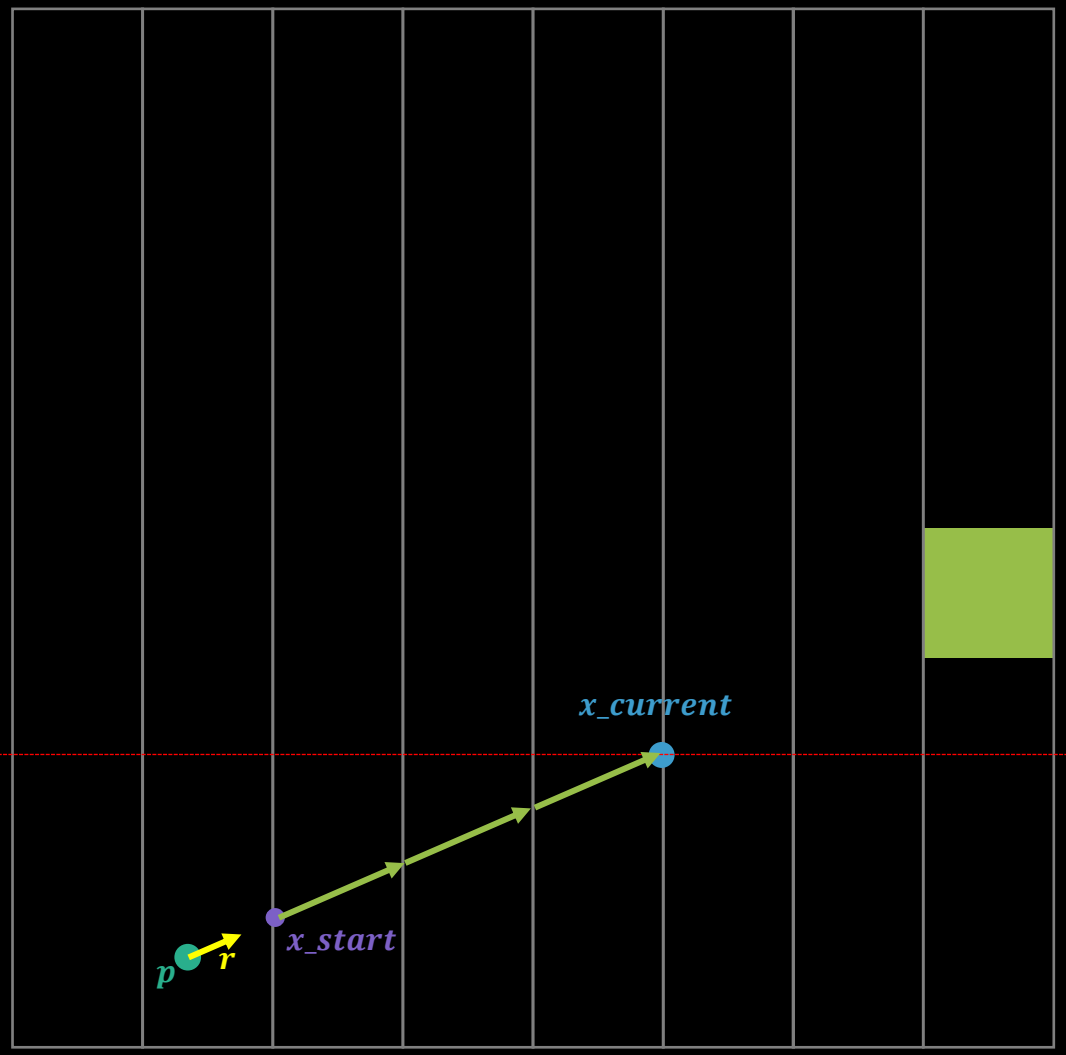
DDA알고리즘 구현하기::반복구간

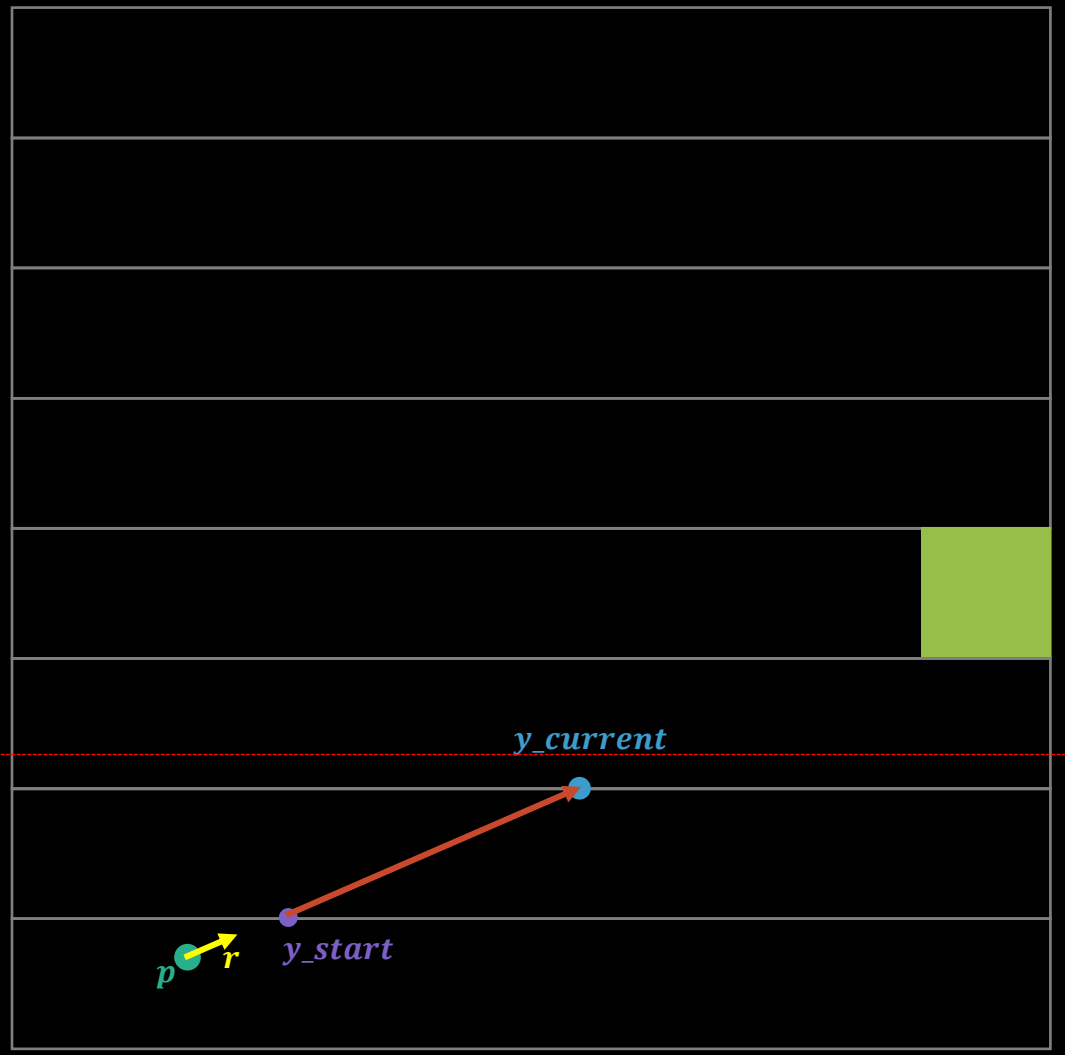




DDA알고리즘

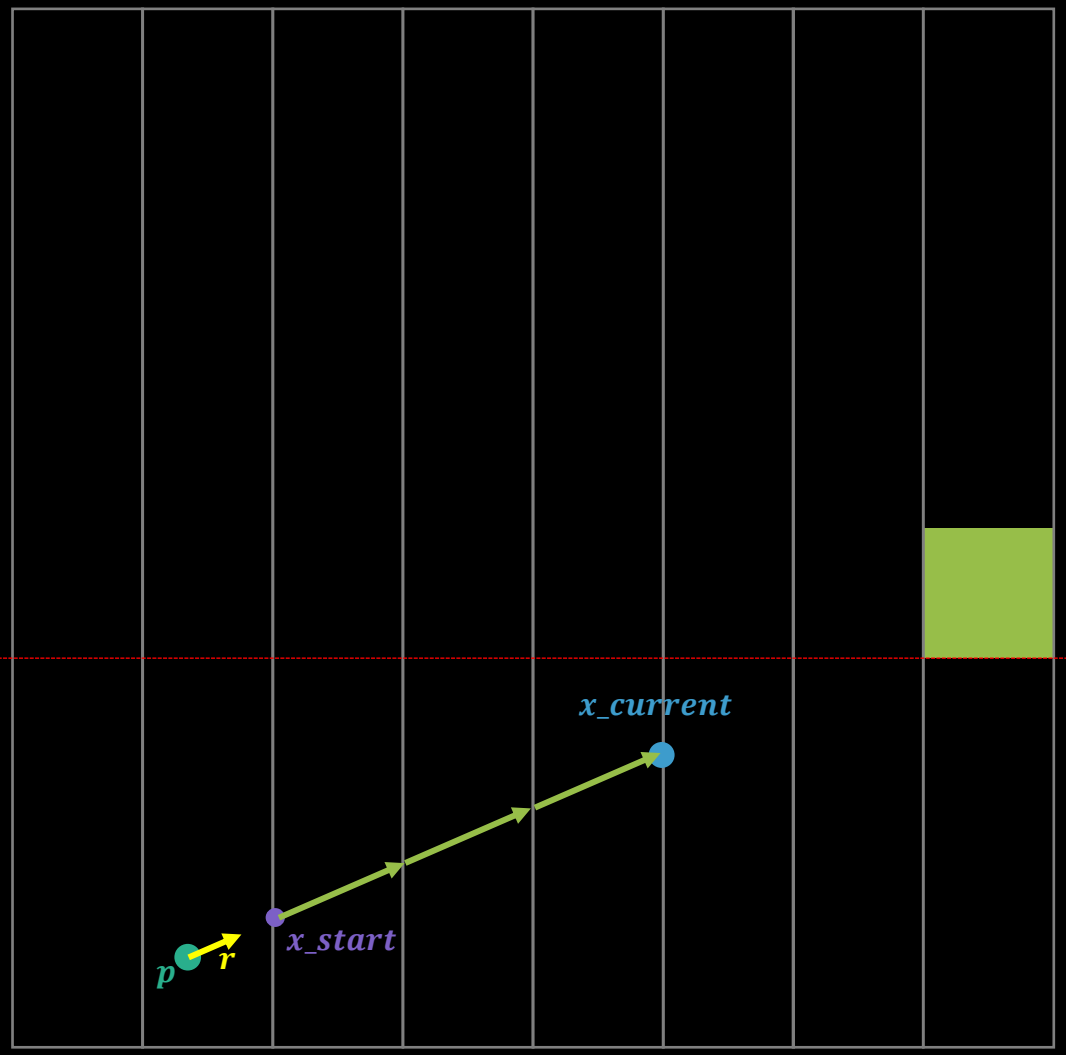
DDA알고리즘 구현하기::반복구간

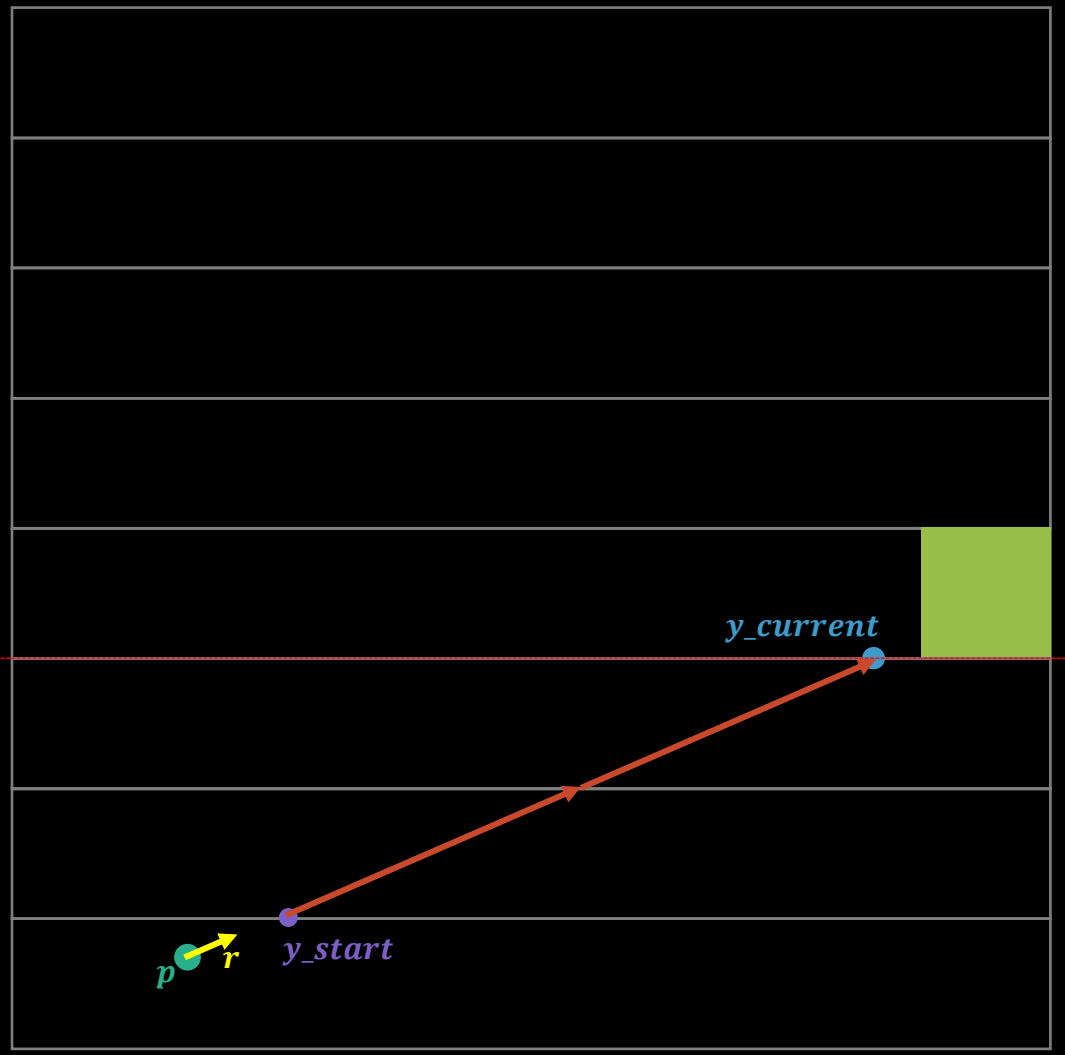




DDA알고리즘

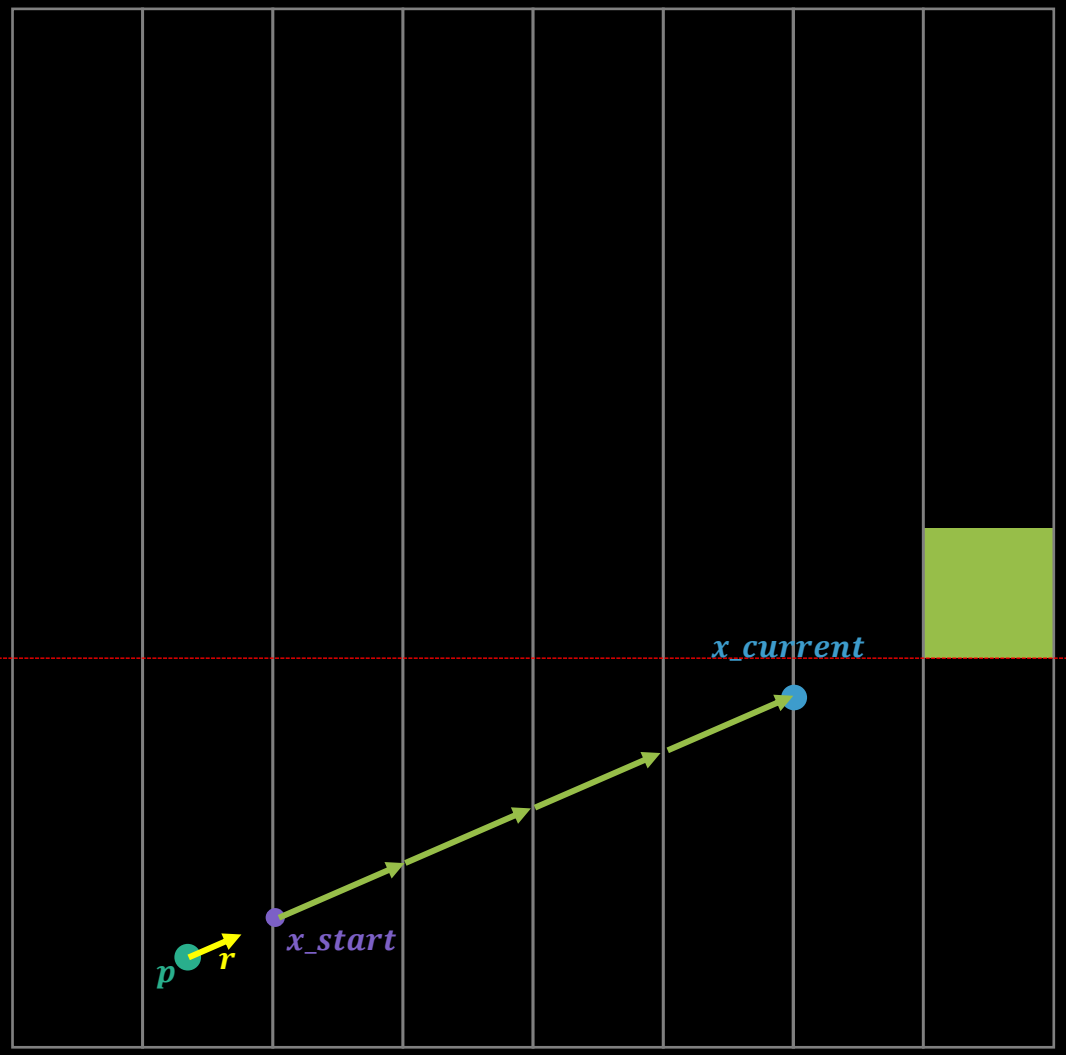
DDA알고리즘 구현하기::반복구간

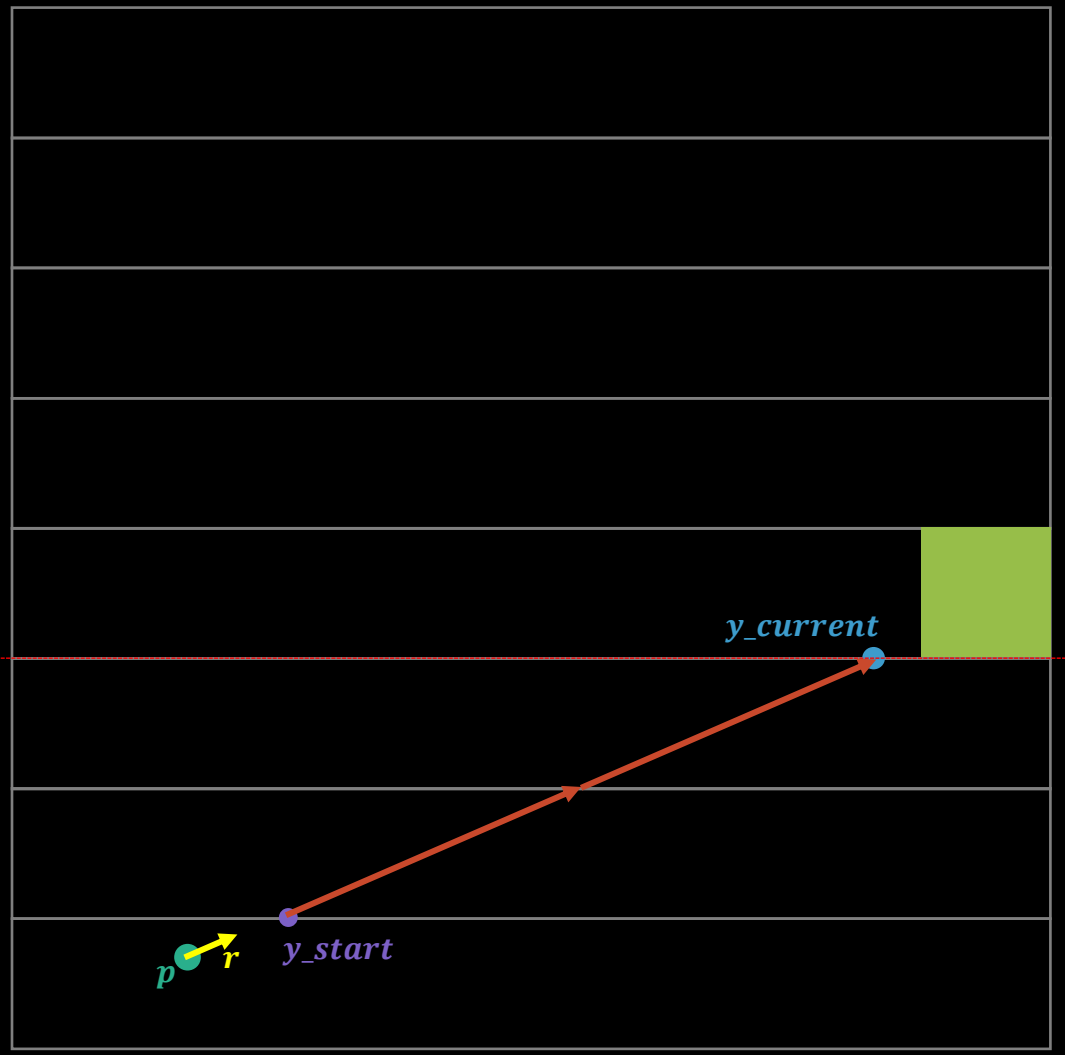




DDA알고리즘

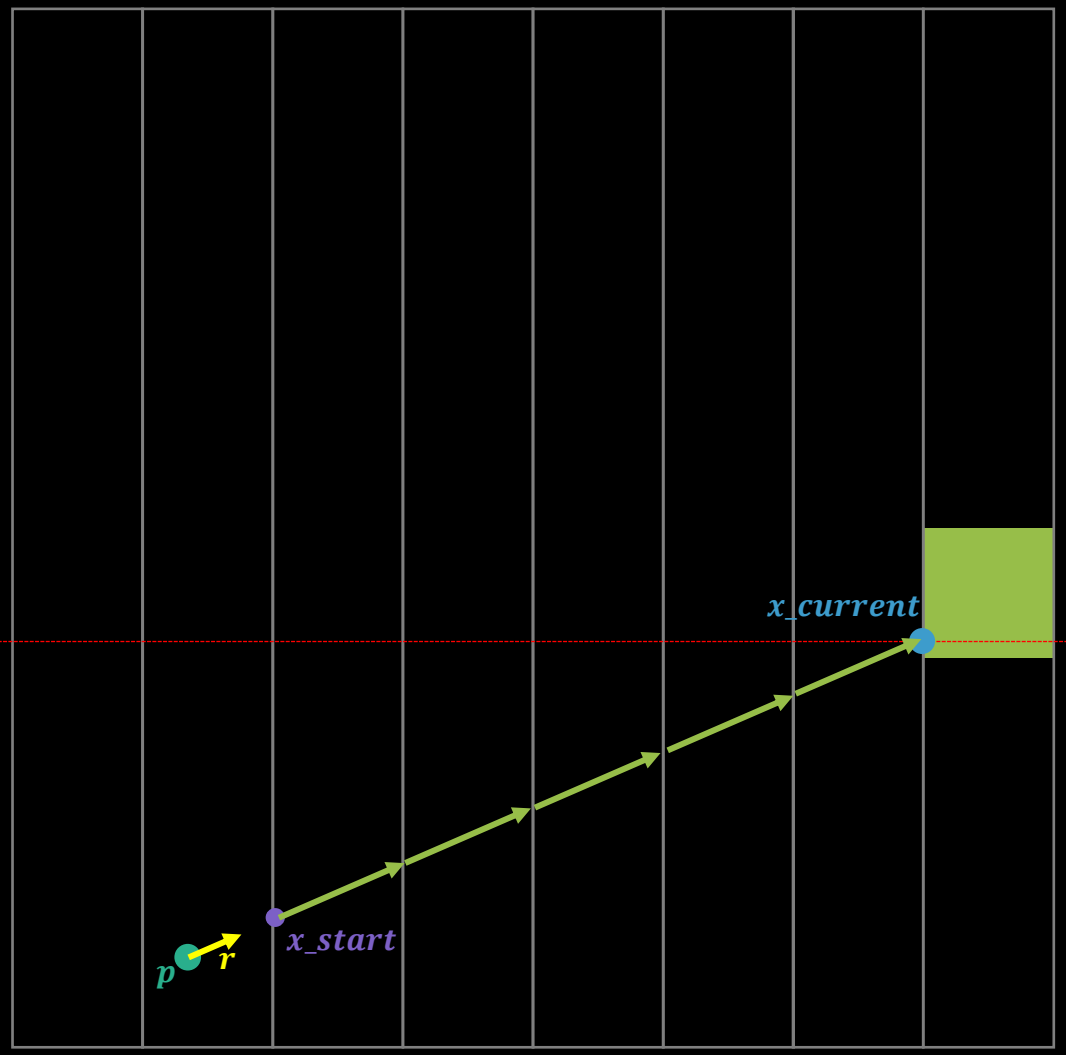
DDA알고리즘 구현하기::반복구간





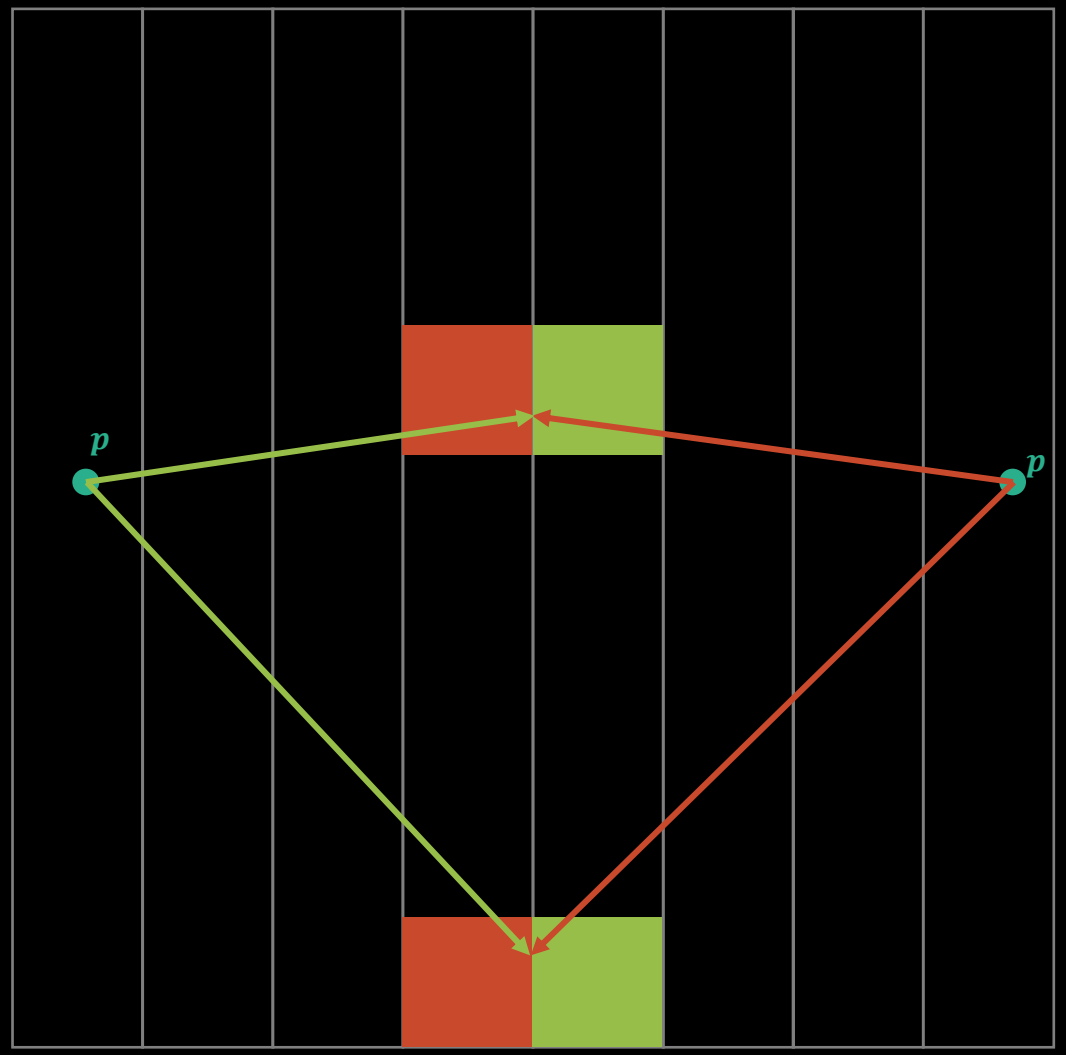
DDA알고리즘

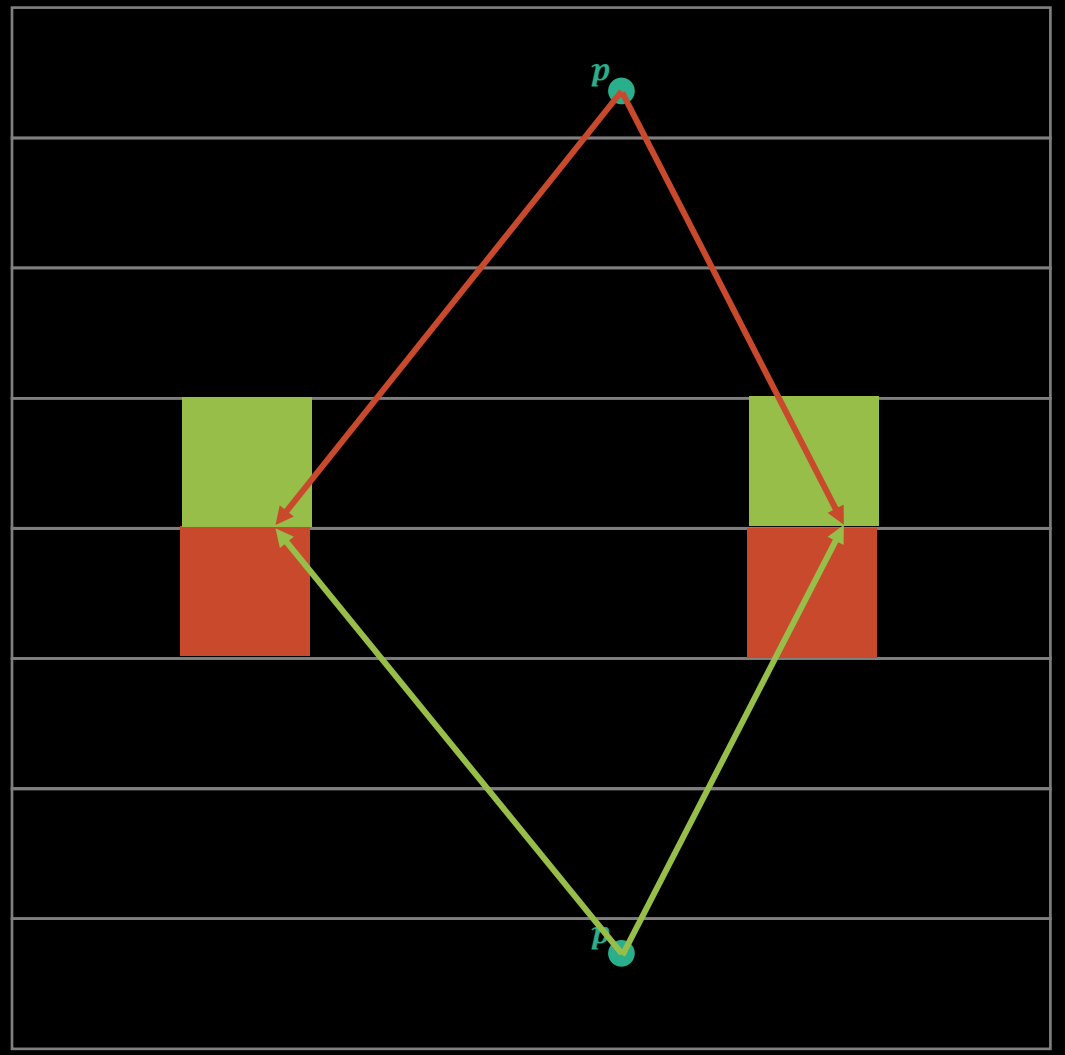
DDA알고리즘 구현하기::마무리



DDA알고리즘

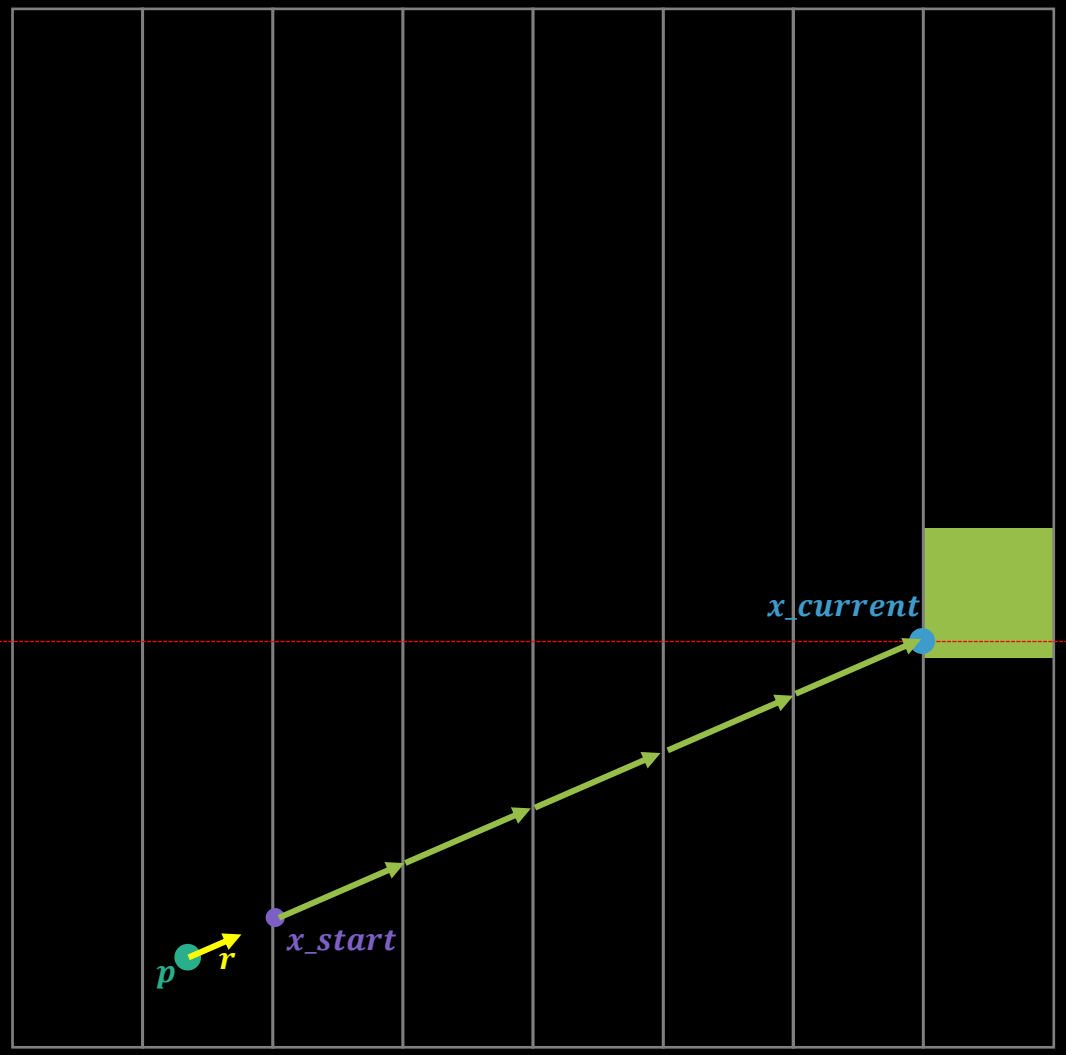
DDA알고리즘 구현하기::광선 진행방향에 주의하기





DDA알고리즘

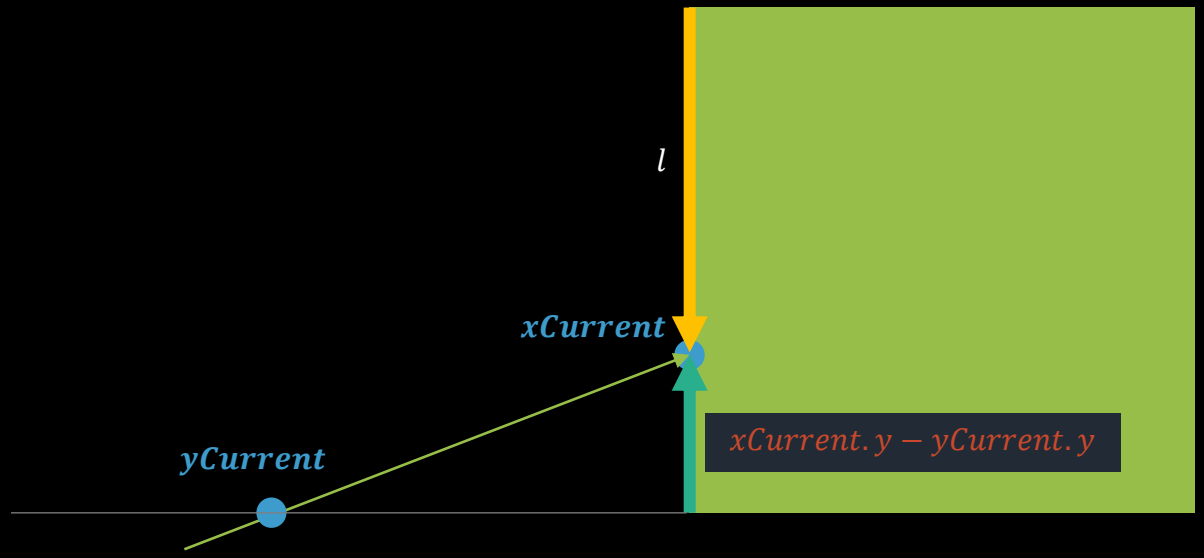
DDA알고리즘 구현하기::거리 구하기

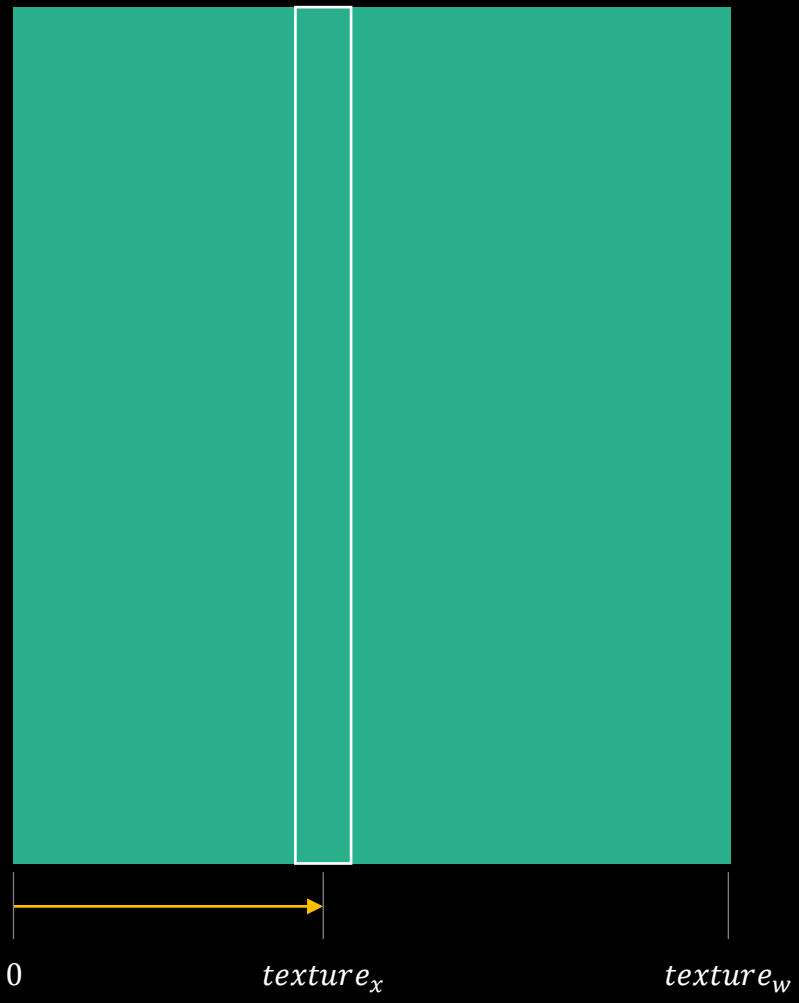


$$distance = \sqrt{(\boldsymbol{current} - \boldsymbol{p})^2}$$

DDA알고리즘

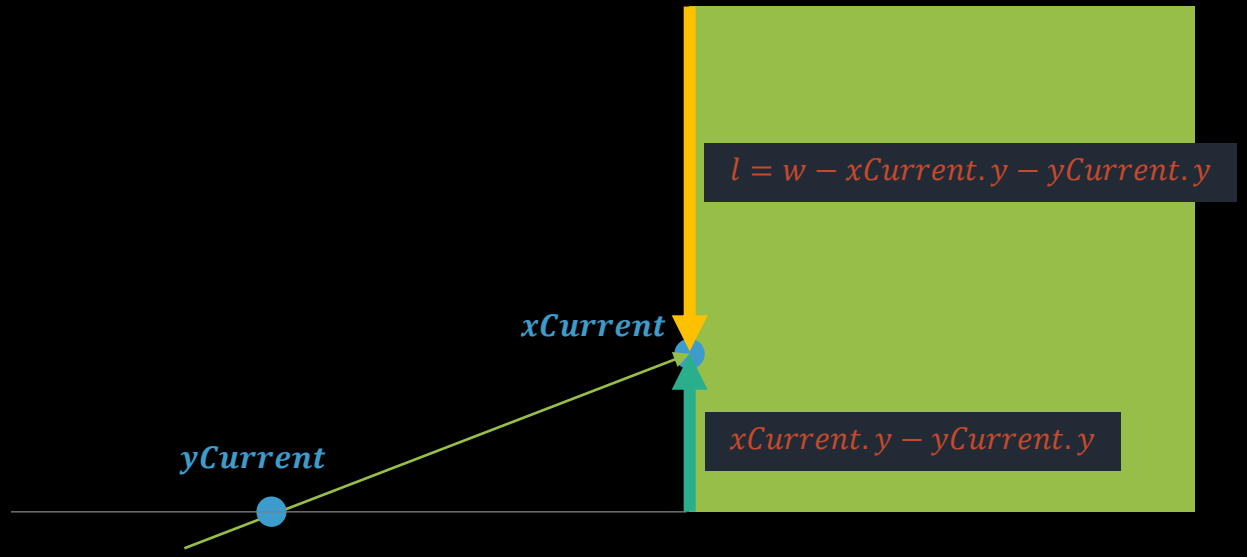
텍스처의 x방향 인덱스 구하기

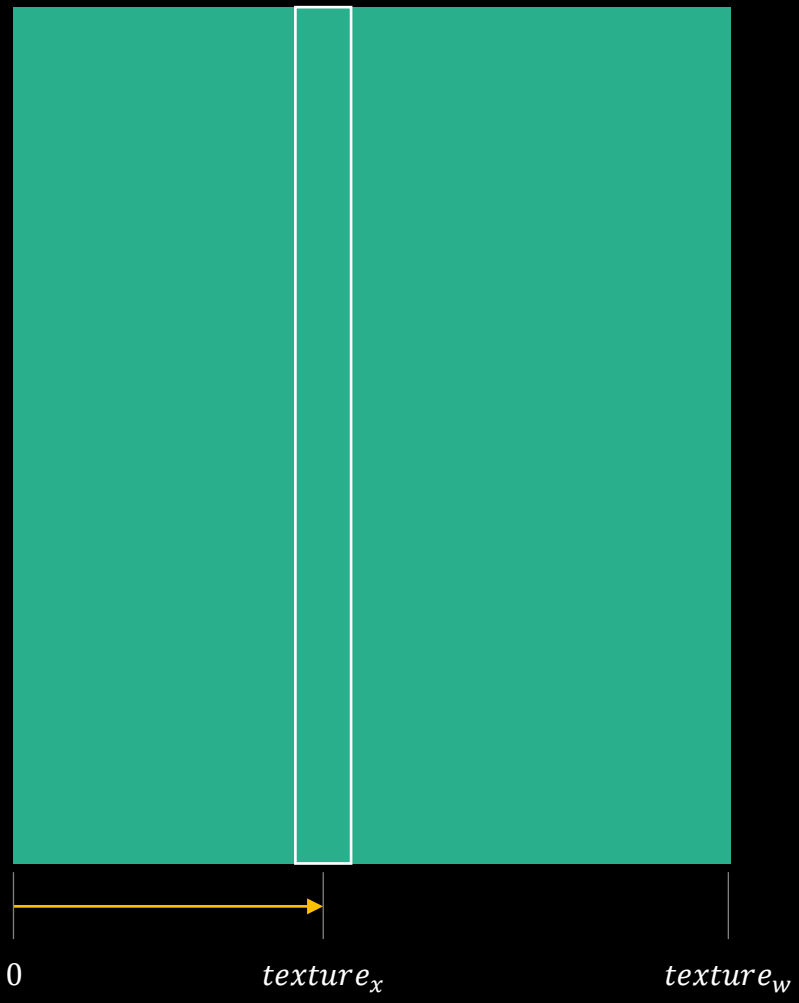




DDA알고리즘

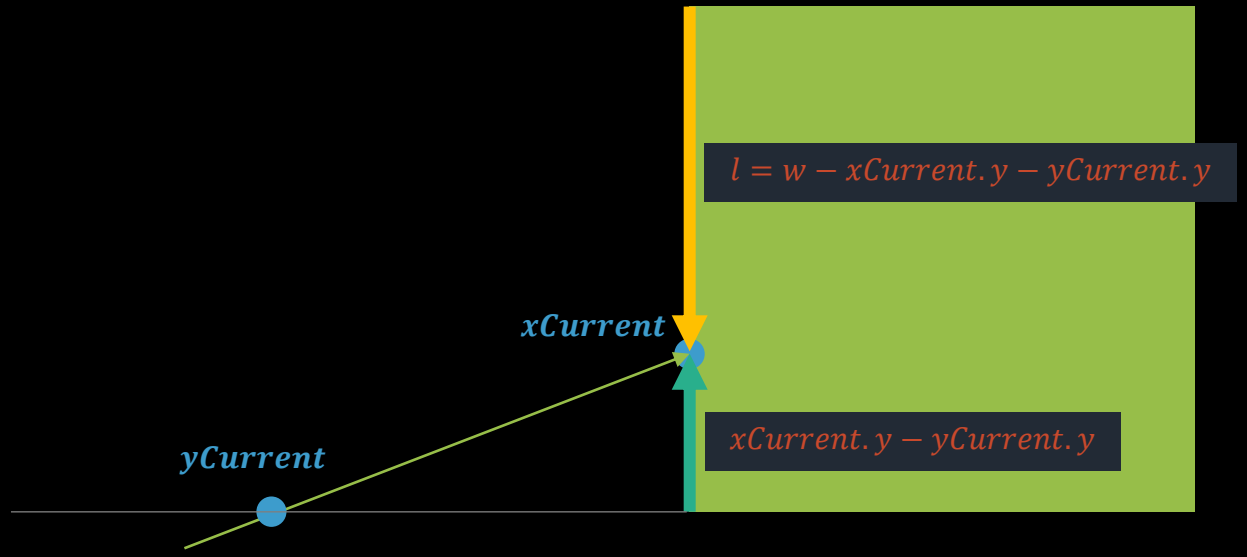
텍스처의 x방향 인덱스 구하기

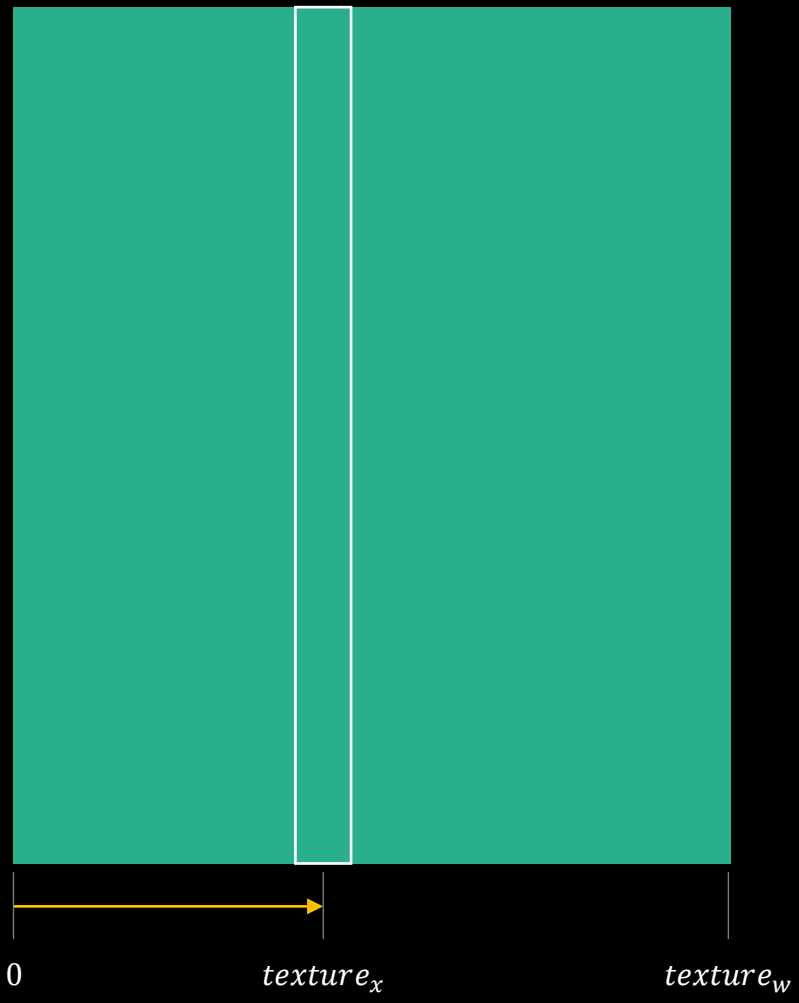




레이캐스팅::DDA알고리즘

텍스처의 x방향 인덱스 구하기





$$\frac{texture_x}{texture_w} = \frac{l}{w}$$

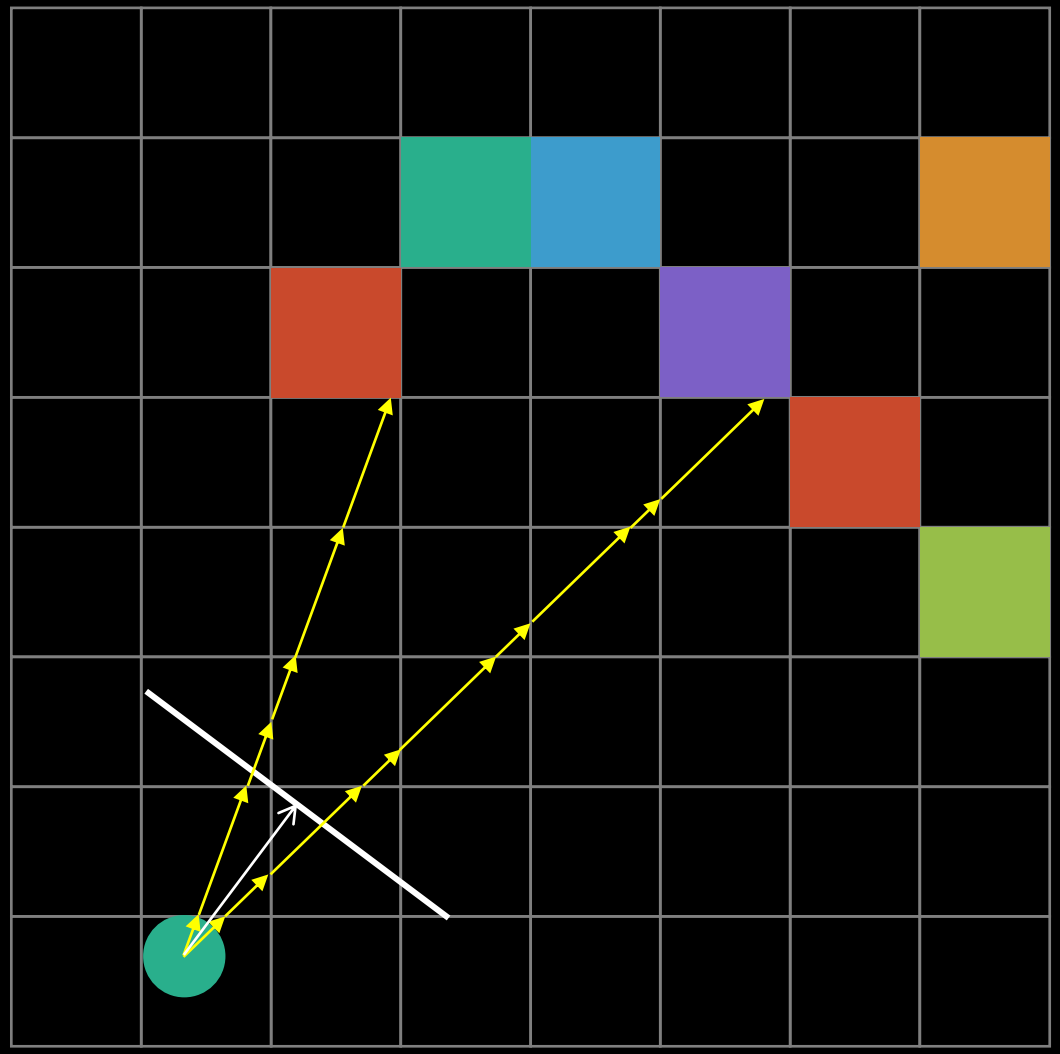
$$texture_x = l * texture_w \,/\, w$$

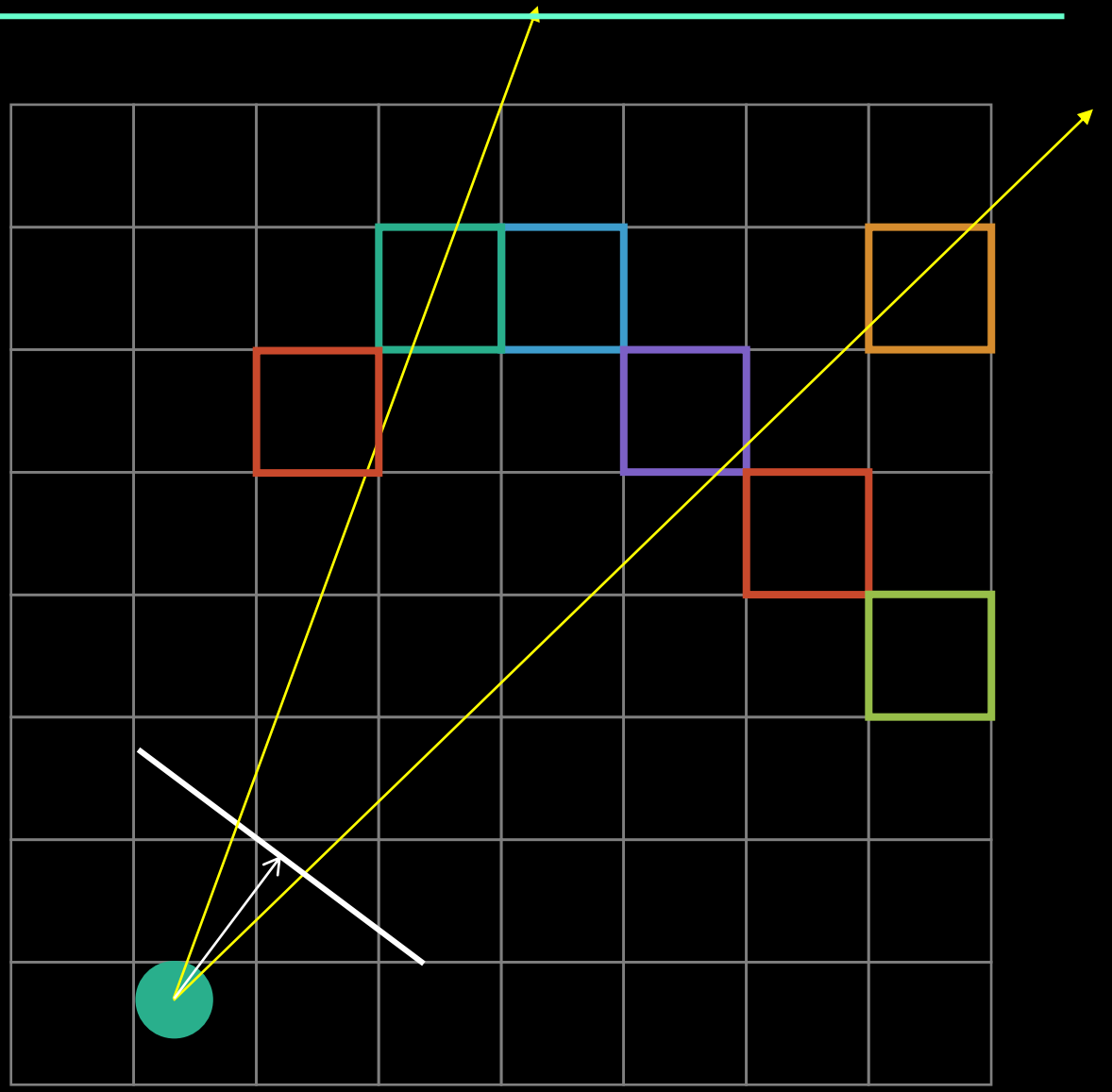
minckim의 레이캐스팅 알고리즘

#minckim #야매 #사파 #사문난적 주의

minckim의 레이캐스팅 알고리즘

DDA와 비교

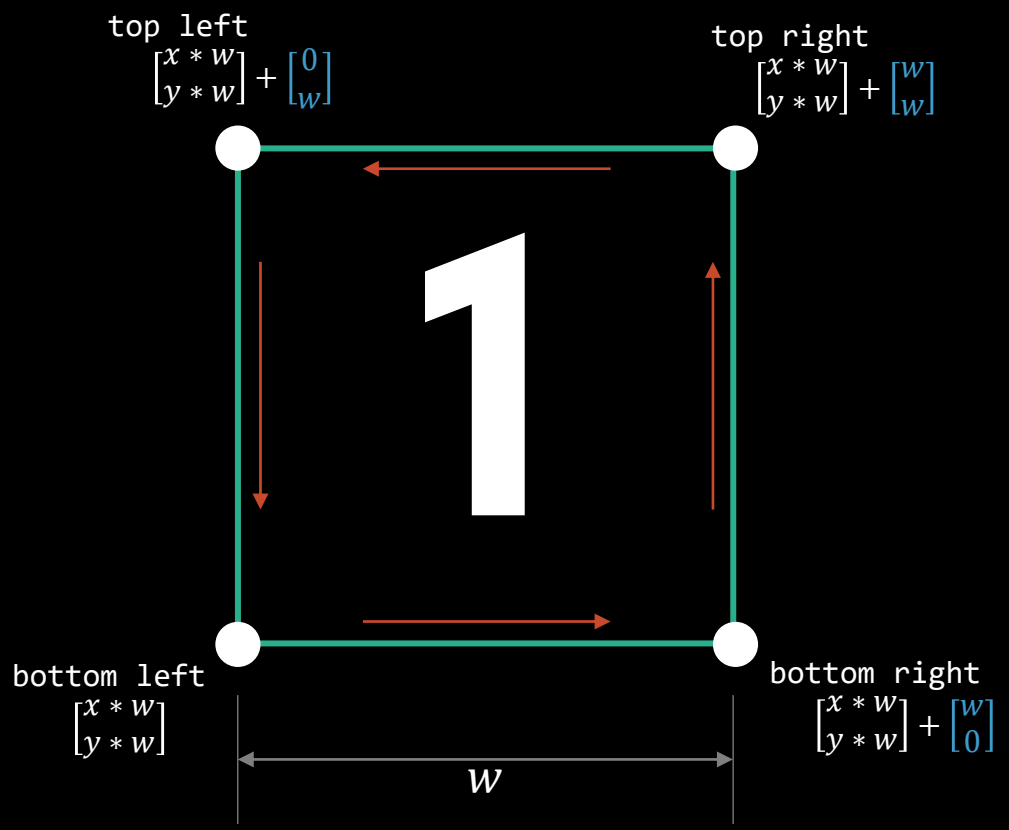




minckim의 레이캐스팅 알고리즘

벽면 요소 만들기

```
typedef struct      s_entity{
    t_vec           a;
    t_vec           b;
    t_img           *texture;
}                   t_entity;
```

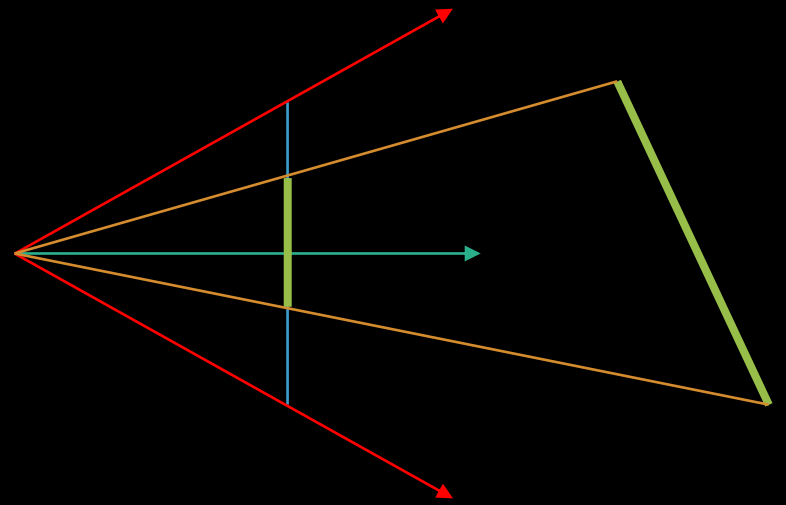



```
t_entity    *make_wall_from_a_point(int x, int y, t_entity *start, t_img *tex)
{
    t_vec   btm_l;
    t_vec   btm_r;
    t_vec   top_l;
    t_vec   top_r;

    btm_l = vec_new(x * WALL_W          , y * WALL_W);
    btm_r = vec_new(x * WALL_W + WALL_W , y * WALL_W);
    top_l = vec_new(x * WALL_W          , y * WALL_W + WALL_W);
    top_r = vec_new(x * WALL_W + WALL_W , y * WALL_W + WALL_W);
    start[0] = entity_new(btm_r, top_r, tex + EAST);
    start[1] = entity_new(top_r, top_l, tex + NORTH);
    start[2] = entity_new(top_l, btm_l, tex + WEST);
    start[3] = entity_new(btm_l, btm_r, tex + SOUTH);
    return start + 4;
}
t_entity    *make_wall(char **map, t_img *texture)
{
    t_entity    *entity;
    t_entity    *entity_tmp;
    int         x;
    int         y;

    entity = malloc(sizeof(t_entity) * (get_entity_num(map, '1') * 4 + 1));
    entity_tmp = entity;
    y = -1;
    while (map[++y])
    {
        x = -1;
        while (map[y][++x])
            if (map[y][x] == '1')
            {
                entity = make_wall_from_a_point(x, y, entity, texture);
            }
    }
    entity->texture = 0;
    return entity_tmp;
}
```

minckim의 레이캐스팅 알고리즘

레이캐스팅 함수::최적화 하기

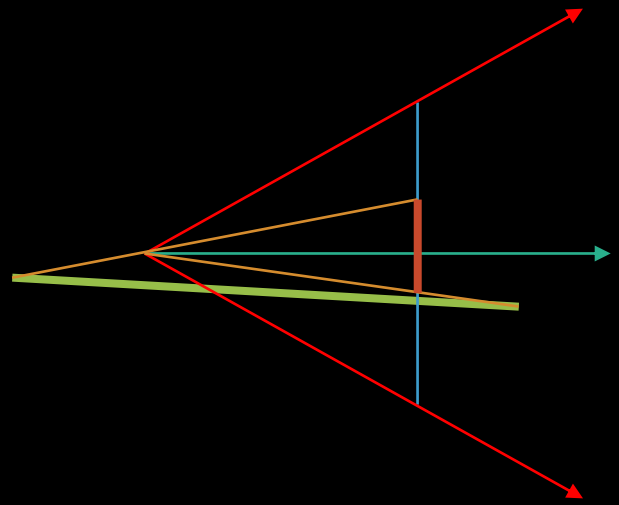
```
void    draw_wall(t_screen *screen, t_entity *wall)
{
    t_vec   range;
    int     min;
    int     max;
    t_vec   hit_info;
    t_ray   *ray;

    while (wall->texture)
    {
        range = get_width_range(screen, wall);
        min = (int)range.x;
        max = (int)range.y;
        while (min < max)
        {
            ray = screen->ray + min;
            hit_info = ray_x_face(ray, wall, &screen->origin);
            if (0 < hit_info.x && 0 <= hit_info.y && hit_info.y <= 1)
            {
                if (hit_info.x < ray->distance)
                {
                    ray->distance = hit_info.x;
                    draw_vertical(screen, wall, &hit_info, min);
                }
            }
            min++;
        }
        wall++;
    }
}
```



minckim의 레이캐스팅 알고리즘

최적화 하기::예외처리

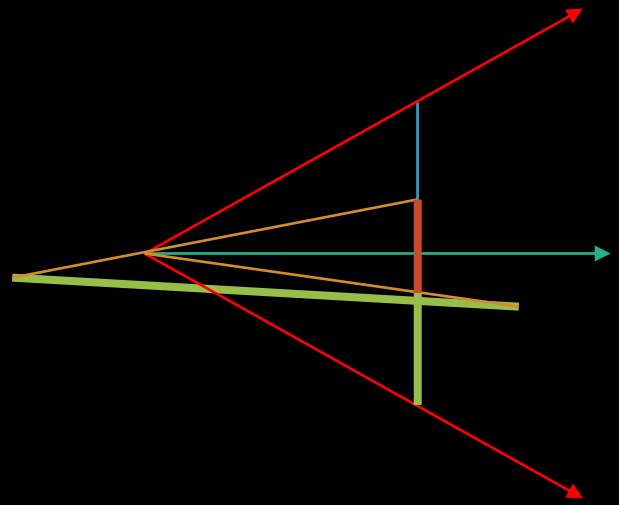
```
void    draw_wall(t_screen *screen, t_entity *wall)
{
    t_vec   range;
    int     min;
    int     max;
    t_vec   hit_info;
    t_ray   *ray;

    while (wall->texture)
    {
        range = get_width_range(screen, wall);
        min = (int)range.x;
        max = (int)range.y;
        while (min < max)
        {
            ray = screen->ray + min;
            hit_info = ray_x_face(ray, wall, &screen->origin);
            if (0 < hit_info.x && 0 <= hit_info.y && hit_info.y <= 1)
            {
                if (hit_info.x < ray->distance)
                {
                    ray->distance = hit_info.x;
                    draw_vertical(screen, wall, &hit_info, min);
                }
            }
            min++;
        }
        wall++;
    }
}
```



minckim의 레이캐스팅 알고리즘

최적화 하기::예외처리

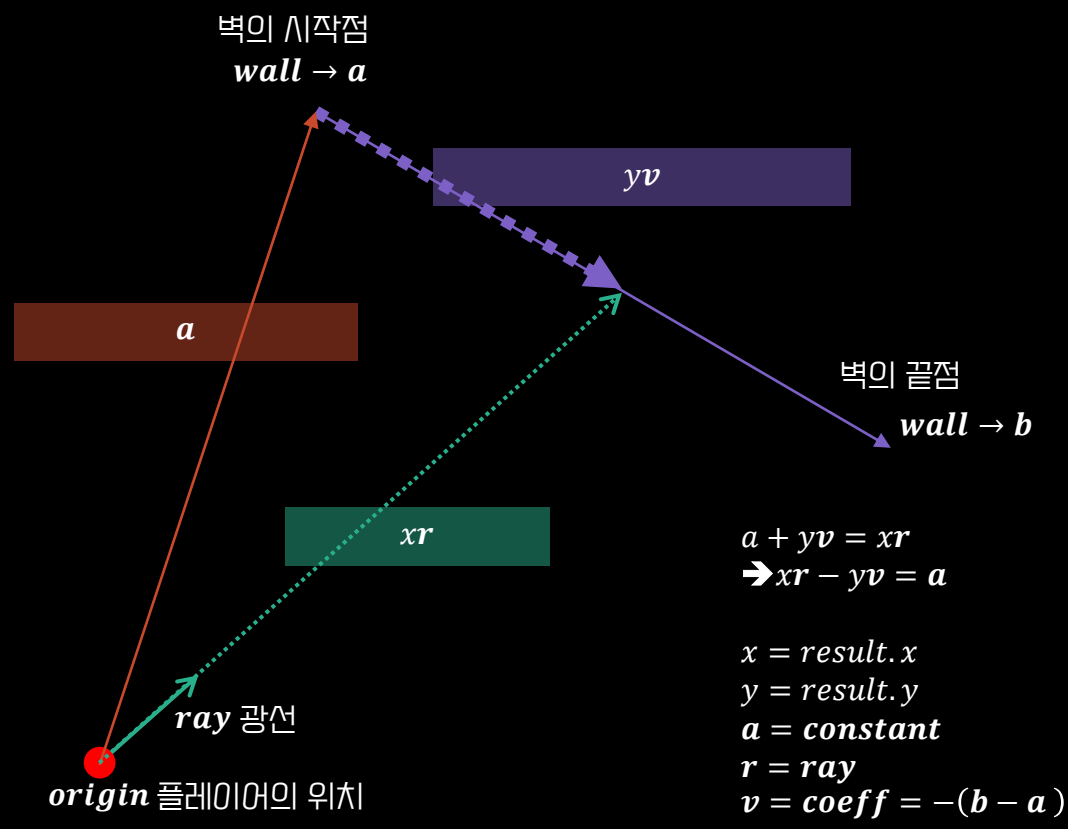
```
void    draw_wall(t_screen *screen, t_entity *wall)
{
    t_vec   range;
    int     min;
    int     max;
    t_vec   hit_info;
    t_ray   *ray;

    while (wall->texture)
    {
        range = get_width_range(screen, wall);
        min = (int)range.x;
        max = (int)range.y;
        while (min < max)
        {
            ray = screen->ray + min;
            hit_info = ray_x_face(ray, wall, &screen->origin);
            if (0 < hit_info.x && 0 <= hit_info.y && hit_info.y <= 1)
            {
                if (hit_info.x < ray->distance)
                {
                    ray->distance = hit_info.x;
                    draw_vertical(screen, wall, &hit_info, min);
                }
            }
            min++;
        }
        wall++;
    }
}
```



minckim의 레이캐스팅 알고리즘

벽과의 거리 계산 및 텍스처 인덱스 구하기

```c
void    draw_wall(t_screen *screen, t_entity *wall)
{
    t_vec   range;
    int     min;
    int     max;
    t_vec   hit_info;
    t_ray   *ray;

    while (wall->texture)
    {
        range = get_width_range(screen, wall);
        min = (int)range.x;
        max = (int)range.y;
        while (min < max)
        {
            ray = screen->ray + min;
            hit_info = ray_x_face(ray, wall, &screen->origin);
            if (0 < hit_info.x && 0 <= hit_info.y && hit_info.y <= 1)
            {
                if (hit_info.x < ray->distance)
                {
                    ray->distance = hit_info.x;
                    draw_vertical(screen, wall, &hit_info, min);
                }
            }
            min++;
        }
        wall++;
    }
}
```

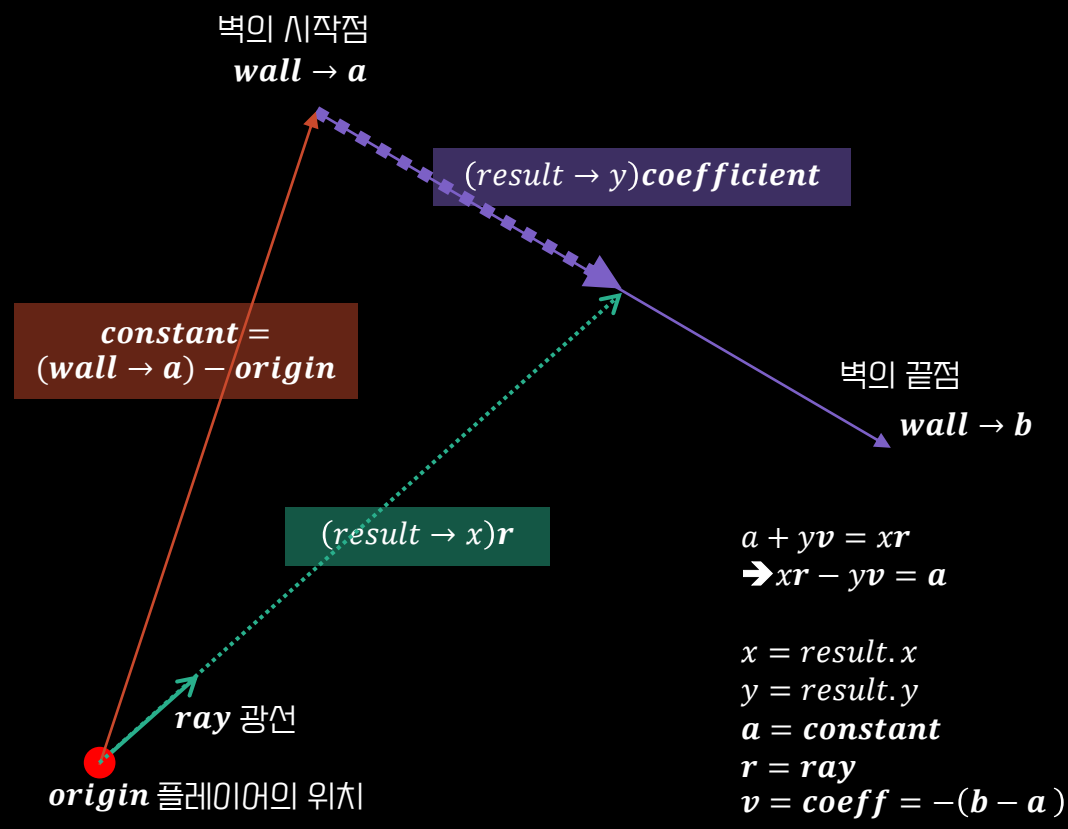



```c
t_vec   ray_x_face(t_ray *ray, t_entity *wall, t_vec *origin)
{
    t_vec   coeff;
    t_vec   constant;

    coeff = vec_sub(wall->a, wall->b);
    constant = vec_sub(wall->a, *origin);
    return (equation_solver(&ray->dir, &coeff, &constant));
}
```

minckim의 레이캐스팅 알고리즘

벽과의 거리 계산 및 텍스처 인덱스 구하기

```
void    draw_wall(t_screen *screen, t_entity *wall)
{
    t_vec   range;
    int     min;
    int     max;
    t_vec   hit_info;
    t_ray   *ray;

    while (wall->texture)
    {
        range = get_width_range(screen, wall);
        min = (int)range.x;
        max = (int)range.y;
        while (min < max)
        {
            ray = screen->ray + min;
            hit_info = ray_x_face(ray, wall, &screen->origin);
            if (0 < hit_info.x && 0 <= hit_info.y && hit_info.y <= 1)
            {
                if (hit_info.x < ray->distance)
                {
                    ray->distance = hit_info.x;
                    draw_vertical(screen, wall, &hit_info, min);
                }
            }
            min++;
        }
        wall++;
    }
}
```

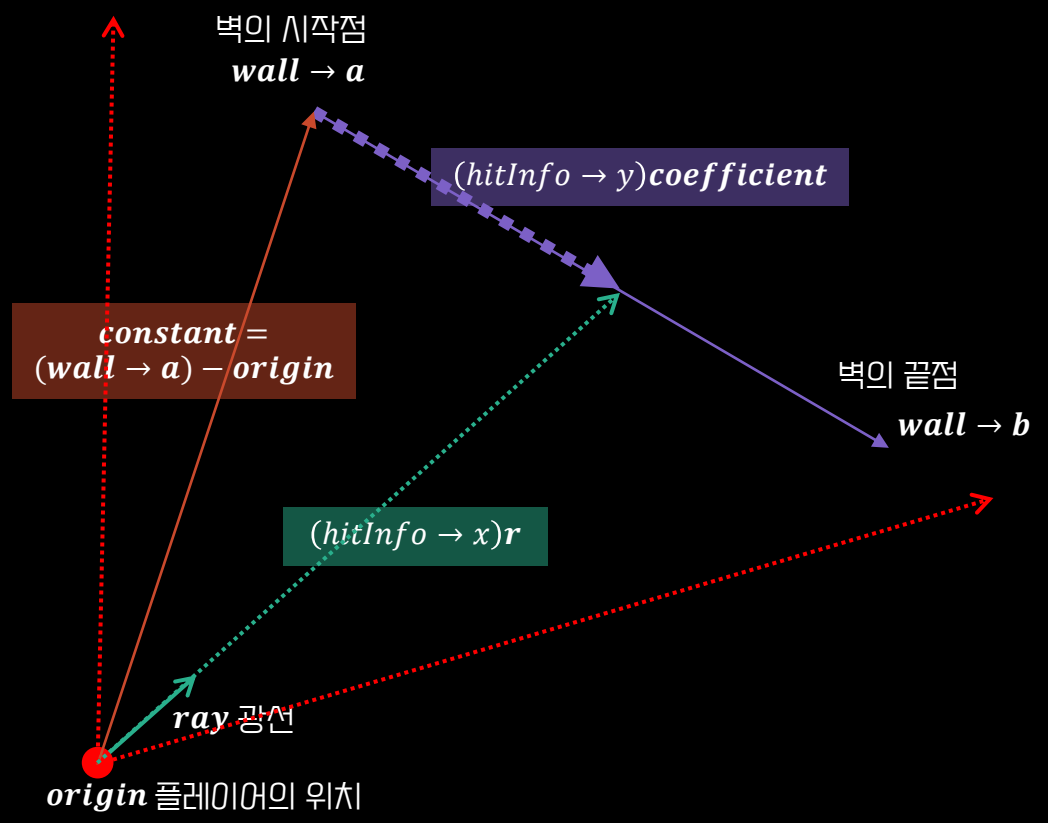



```
t_vec   ray_x_face(t_ray *ray, t_entity *wall, t_vec *origin)
{
    t_vec   coeff;
    t_vec   constant;

    coeff = vec_sub(wall->a, wall->b);
    constant = vec_sub(wall->a, *origin);
    return (equation_solver(&ray->dir, &coeff, &constant));
}
```

minckim의 레이캐스팅 알고리즘

벽과의 거리 계산 및 텍스처 인덱스 구하기:: 광선이 빗나간 경우

```
void    draw_wall(t_screen *screen, t_entity *wall)
{
    t_vec   range;
    int     min;
    int     max;
    t_vec   hit_info;
    t_ray   *ray;

    while (wall->texture)
    {
        range = get_width_range(screen, wall);
        min = (int)range.x;
        max = (int)range.y;
        while (min < max)
        {
            ray = screen->ray + min;
            hit_info = ray_x_face(ray, wall, &screen->origin);
            if (0 < hit_info.x && 0 <= hit_info.y && hit_info.y <= 1)
            {
                if (hit_info.x < ray->distance)
                {
                    ray->distance = hit_info.x;
                    draw_vertical(screen, wall, &hit_info, min);
                }
            }
            min++;
        }
        wall++;
    }
}
```

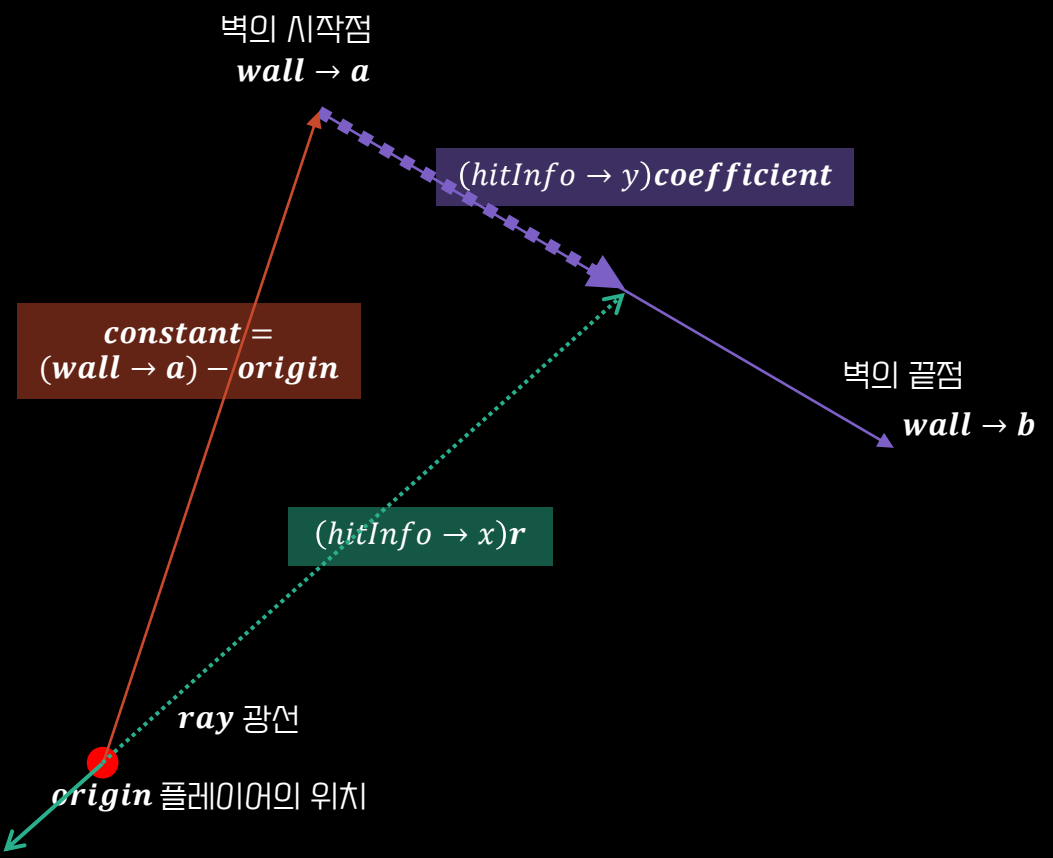



```
t_vec   ray_x_face(t_ray *ray, t_entity *wall, t_vec *origin)
{
    t_vec   coeff;
    t_vec   constant;

    coeff = vec_sub(wall->a, wall->b);
    constant = vec_sub(wall->a, *origin);
    return (equation_solver(&ray->dir, &coeff, &constant));
}
```

minckim의 레이캐스팅 알고리즘

벽과의 거리 계산 및 텍스처 인덱스 구하기:: 광선이 빗나간 경우

```
void    draw_wall(t_screen *screen, t_entity *wall)
{
    t_vec   range;
    int     min;
    int     max;
    t_vec   hit_info;
    t_ray   *ray;

    while (wall->texture)
    {
        range = get_width_range(screen, wall);
        min = (int)range.x;
        max = (int)range.y;
        while (min < max)
        {
            ray = screen->ray + min;
            hit_info = ray_x_face(ray, wall, &screen->origin);
            if (0 < hit_info.x && 0 <= hit_info.y && hit_info.y <= 1)
            {
                if (hit_info.x < ray->distance)
                {
                    ray->distance = hit_info.x;
                    draw_vertical(screen, wall, &hit_info, min);
                }
            }
            min++;
        }
        wall++;
    }
}
```

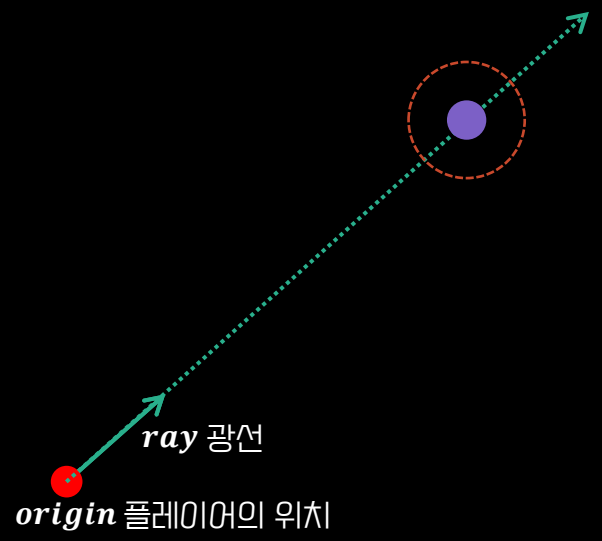



```
t_vec   ray_x_face(t_ray *ray, t_entity *wall, t_vec *origin)
{
    t_vec   coeff;
    t_vec   constant;

    coeff = vec_sub(wall->a, wall->b);
    constant = vec_sub(wall->a, *origin);
    return (equation_solver(&ray->dir, &coeff, &constant));
}
```

minckim의 레이캐스팅 알고리즘

색과 거리 갱신

```
void    draw_wall(t_screen *screen, t_entity *wall)
{
    t_vec   range;
    int     min;
    int     max;
    t_vec   hit_info;
    t_ray   *ray;

    while (wall->texture)
    {
        range = get_width_range(screen, wall);
        min = (int)range.x;
        max = (int)range.y;
        while (min < max)
        {
            ray = screen->ray + min;
            hit_info = ray_x_face(ray, wall, &screen->origin);
            if (0 < hit_info.x && 0 <= hit_info.y && hit_info.y <= 1)
            {
                if (hit_info.x < ray->distance)
                {
                    ray->distance = hit_info.x;
                    draw_vertical(screen, wall, &hit_info, min);
                }
            }
            min++;
        }
        wall++;
    }
}
```

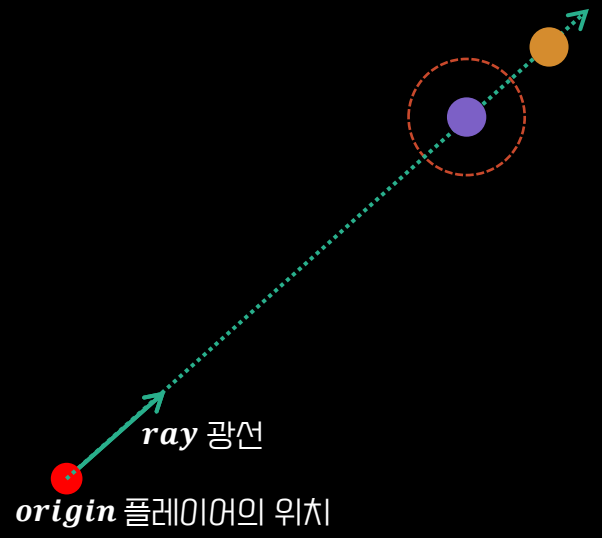



```
t_vec   ray_x_face(t_ray *ray, t_entity *wall, t_vec *origin)
{
    t_vec   coeff;
    t_vec   constant;

    coeff = vec_sub(wall->a, wall->b);
    constant = vec_sub(wall->a, *origin);
    return (equation_solver(&ray->dir, &coeff, &constant));
}
```

minckim의 레이캐스팅 알고리즘

색과 거리 갱신

```
void    draw_wall(t_screen *screen, t_entity *wall)
{
    t_vec   range;
    int     min;
    int     max;
    t_vec   hit_info;
    t_ray   *ray;

    while (wall->texture)
    {
        range = get_width_range(screen, wall);
        min = (int)range.x;
        max = (int)range.y;
        while (min < max)
        {
            ray = screen->ray + min;
            hit_info = ray_x_face(ray, wall, &screen->origin);
            if (0 < hit_info.x && 0 <= hit_info.y && hit_info.y <= 1)
            {
                if (hit_info.x < ray->distance)
                {
                    ray->distance = hit_info.x;
                    draw_vertical(screen, wall, &hit_info, min);
                }
            }
            min++;
        }
        wall++;
    }
}
```

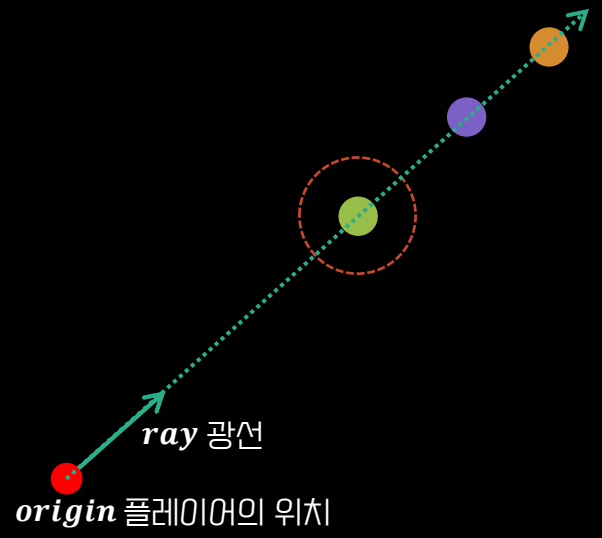



```
t_vec   ray_x_face(t_ray *ray, t_entity *wall, t_vec *origin)
{
    t_vec   coeff;
    t_vec   constant;

    coeff = vec_sub(wall->a, wall->b);
    constant = vec_sub(wall->a, *origin);
    return (equation_solver(&ray->dir, &coeff, &constant));
}
```

minckim의 레이캐스팅 알고리즘

색과 거리 갱신

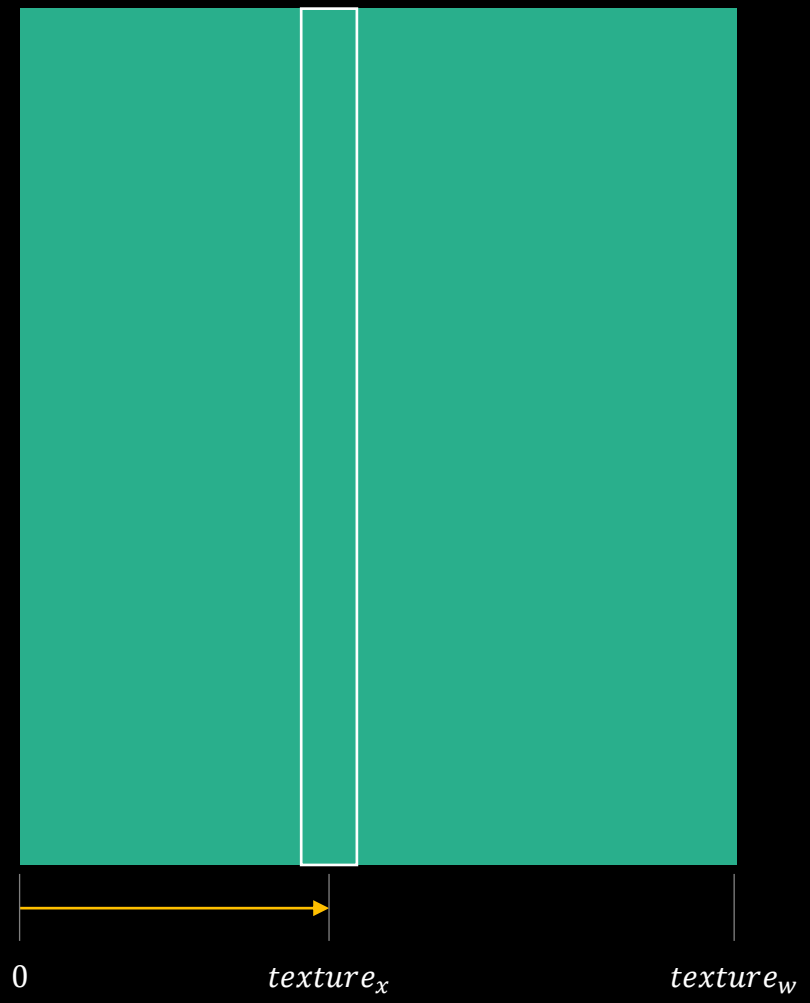
```
void    draw_wall(t_screen *screen, t_entity *wall)
{
    t_vec   range;
    int     min;
    int     max;
    t_vec   hit_info;
    t_ray   *ray;

    while (wall->texture)
    {
        range = get_width_range(screen, wall);
        min = (int)range.x;
        max = (int)range.y;
        while (min < max)
        {
            ray = screen->ray + min;
            hit_info = ray_x_face(ray, wall, &screen->origin);
            if (0 < hit_info.x && 0 <= hit_info.y && hit_info.y <= 1)
            {
                if (hit_info.x < ray->distance)
                {
                    ray->distance = hit_info.x;
                    draw_vertical(screen, wall, &hit_info, min);
                }
            }
            min++;
        }
        wall++;
    }
}
```



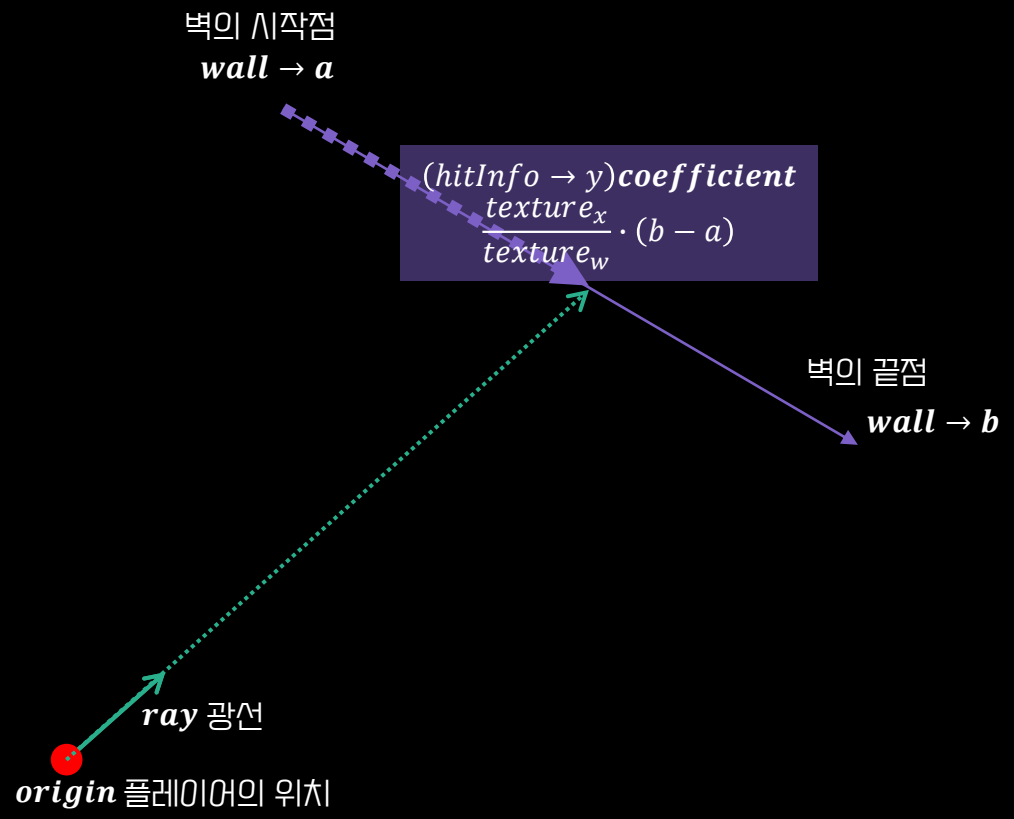

```
t_vec   ray_x_face(t_ray *ray, t_entity *wall, t_vec *origin)
{
    t_vec   coeff;
    t_vec   constant;

    coeff = vec_sub(wall->a, wall->b);
    constant = vec_sub(wall->a, *origin);
    return (equation_solver(&ray->dir, &coeff, &constant));
}
```

minckim의 레이캐스팅 알고리즘

텍스처 인덱스 계산



$$texture_x = hitInfo \to y \cdot texture_w$$



텍스처 그리기

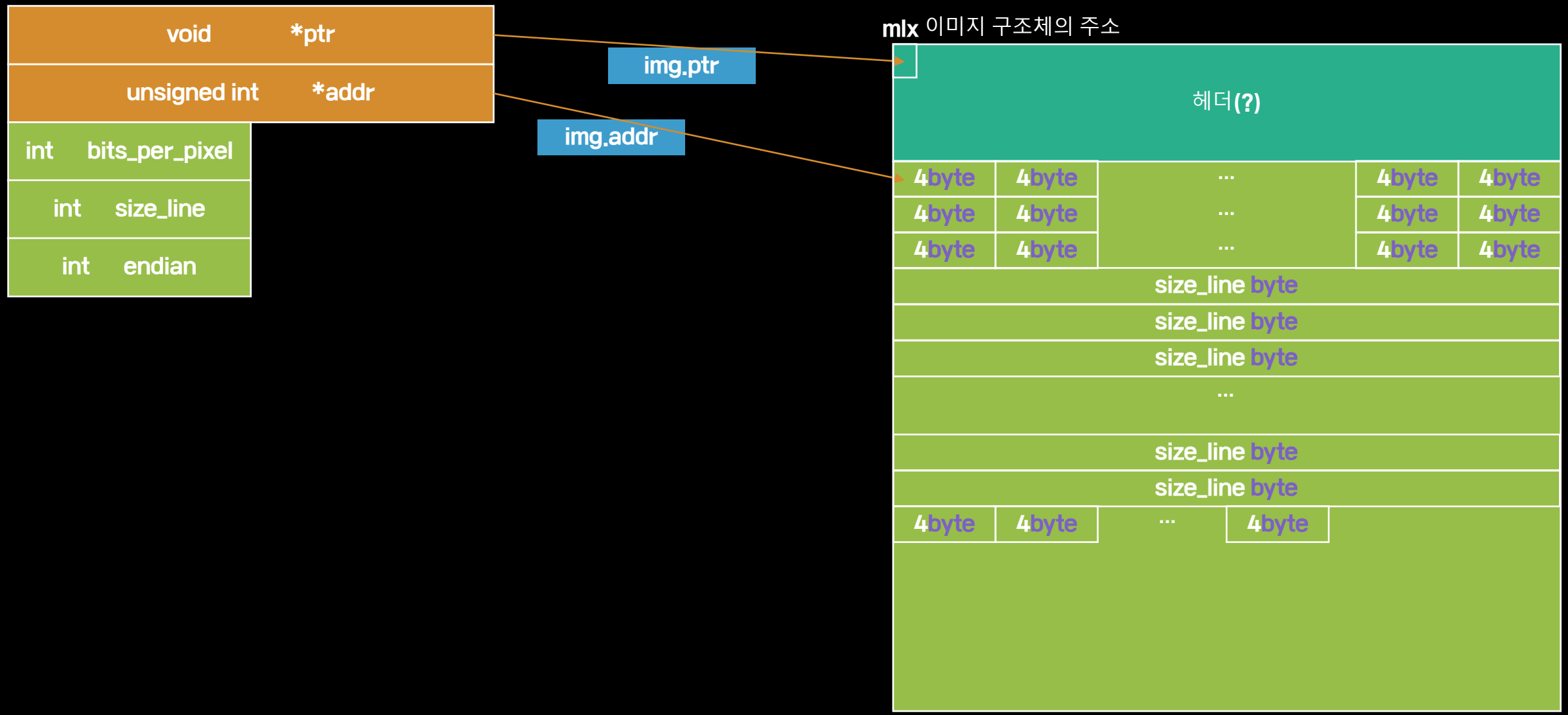
벽의 텍스처를 화면에 그리기

텍스처 그리기

mlx 이미지 다루기

```
img.ptr = mlx_new_image(mlx, w, h);
```

```
img.addr = (unsigned int*)mlx_get_data_addr(img.ptr, &img.bits_per_pixel), &img.size_line, &img.endian);
```

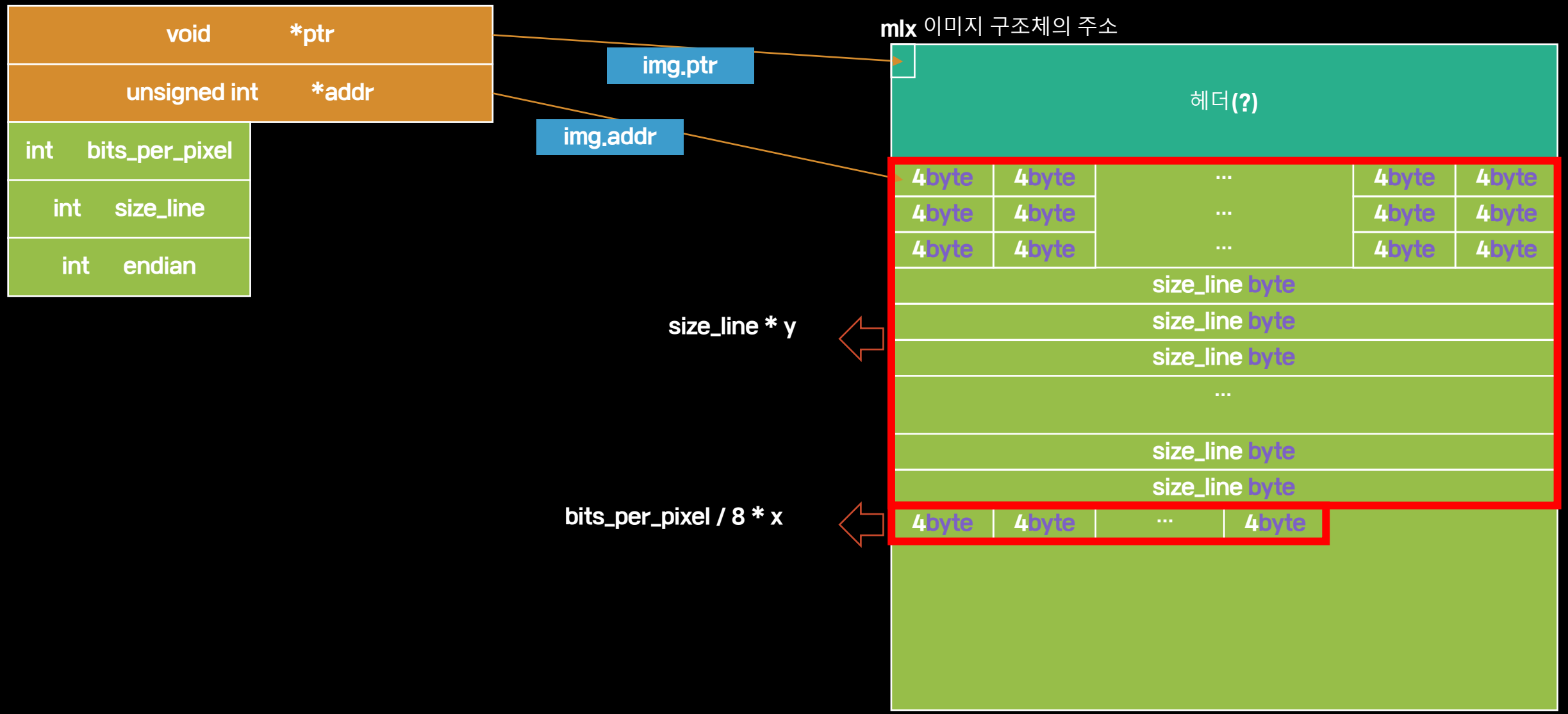


텍스처 그리기

mlx 이미지 다루기

```
img.ptr = mlx_new_image(mlx, w, h);
```

```
img.addr = (unsigned int*)mlx_get_data_addr(img.ptr, &img.bits_per_pixel), &img.size_line, &img.endian);
```



텍스처 그리기

mlx 이미지 다루기

```
img.ptr = mlx_new_image(mlx, w, h);
```

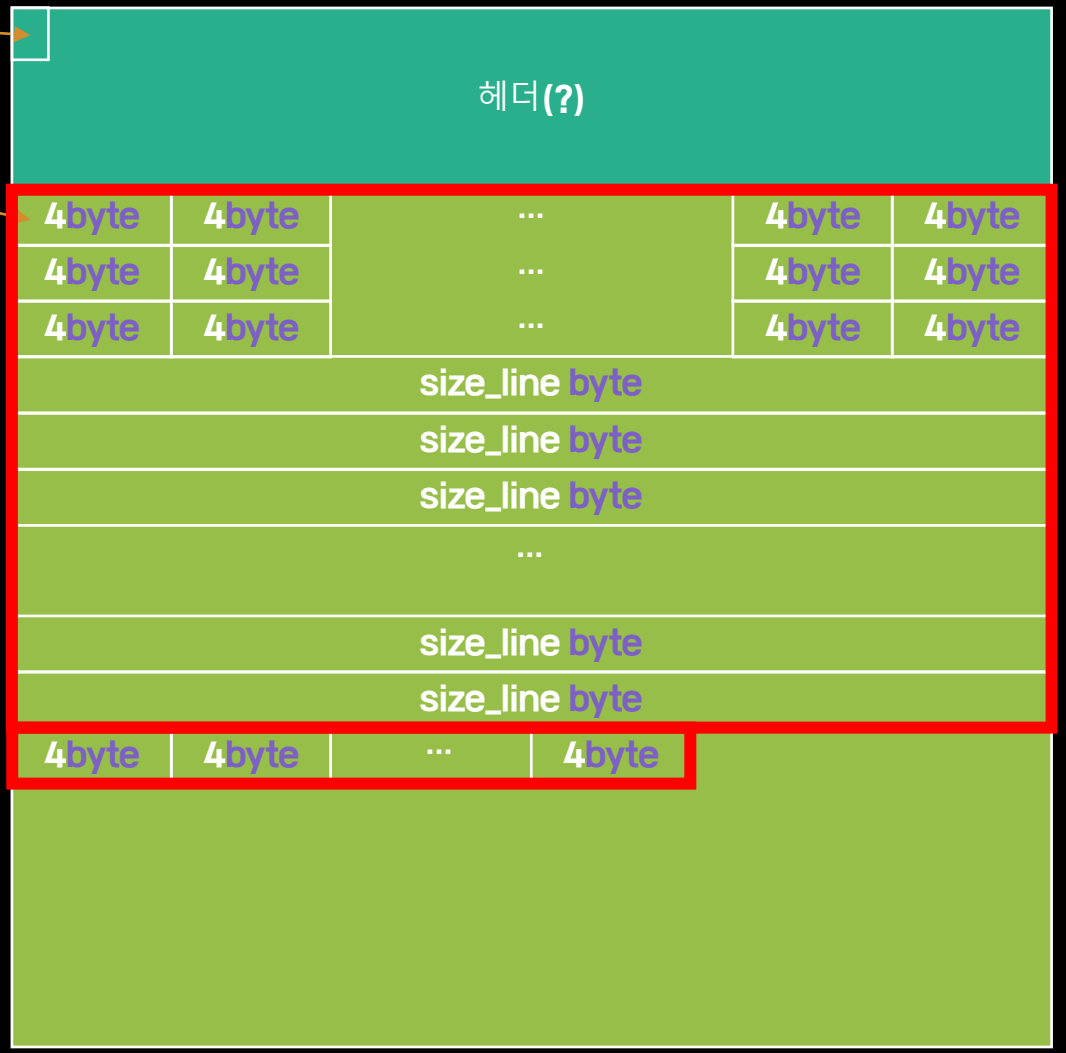
```
img.addr = (unsigned int*)mlx_get_data_addr(img.ptr, &img.bits_per_pixel), &img.size_line, &img.endian);
```

void	*ptr
unsigned int	*addr
int	bits_per_pixel
int	size_line
int	endian

img.ptr

img.addr

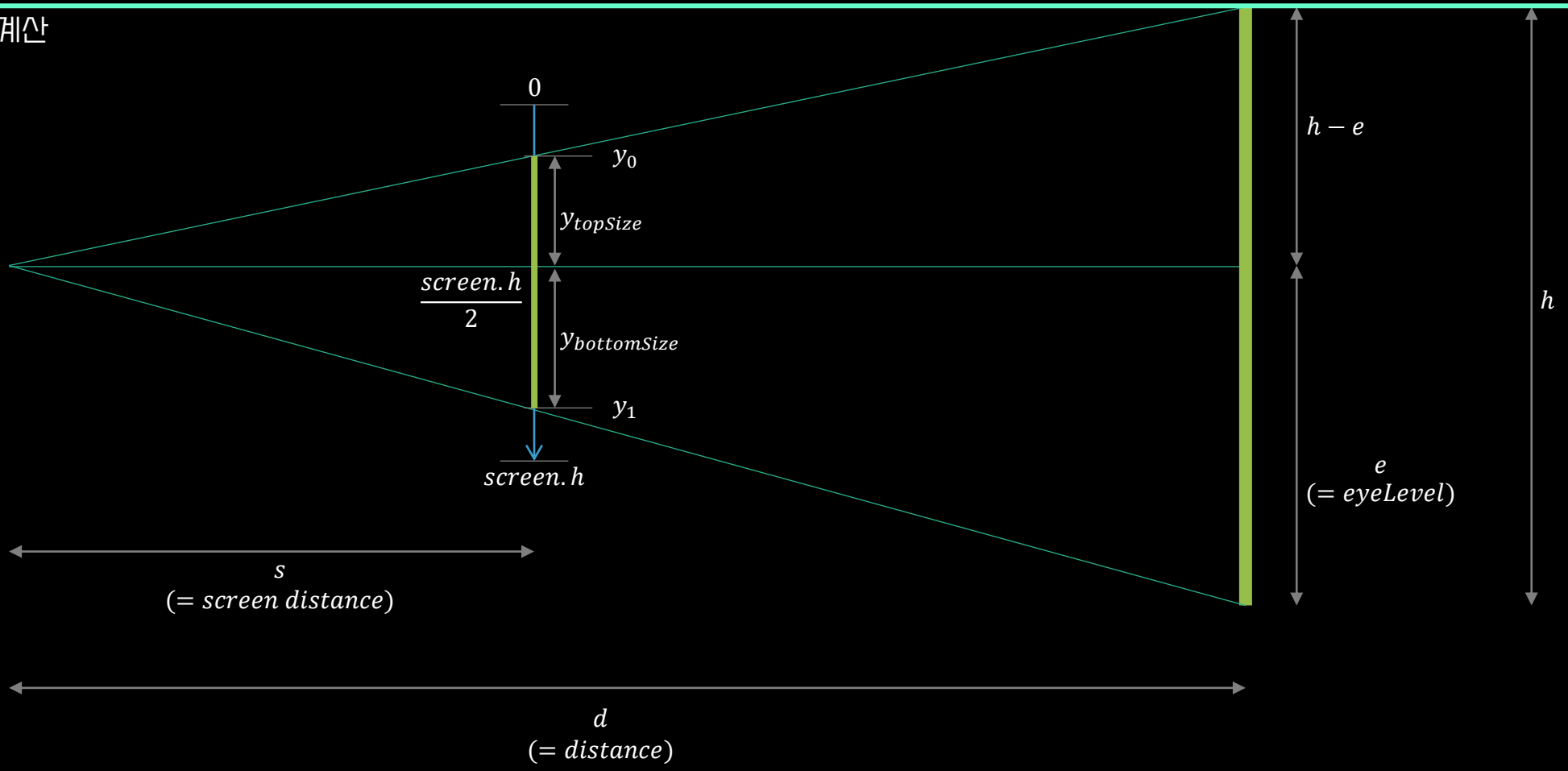
mlx 이미지 구조체의 주소




```
void        img_put_color(t_img *img, int x, int y, unsigned int color)
{
    unsigned int        *point;

    point = (unsigned int*)((char*)img->addr + img->size_line * y \
            + img->bits_per_pixel / 8 * x);
    *point = color;
}

unsigned int    img_pick_color(t_img *img, int x, int y)
{
    unsigned int    *point;

    point = (unsigned int*)((char*)img->addr + img->size_line * y \
            + img->bits_per_pixel / 8 * x);
    return *point;
}
```

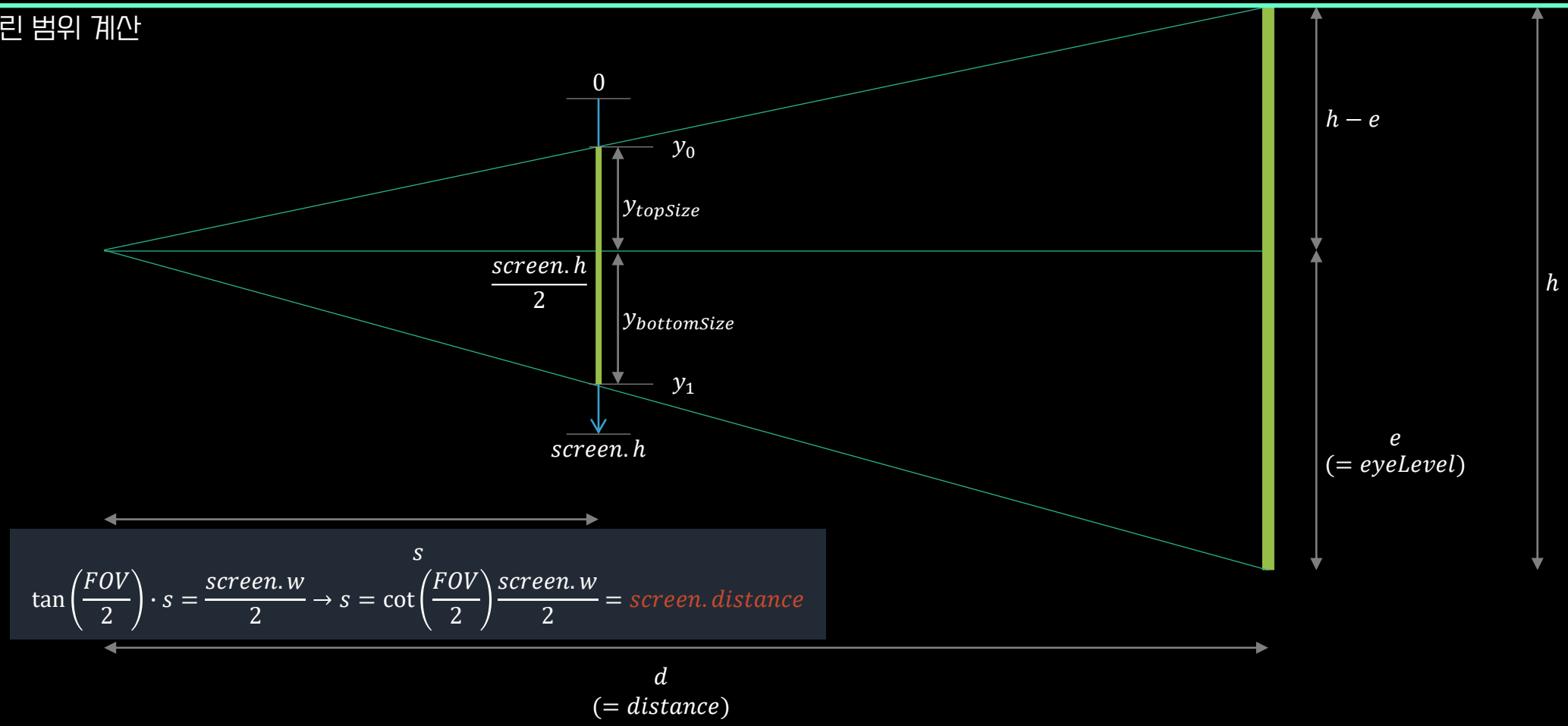
텍스처 그리기

스크린 범위 계산



텍스처 그리기

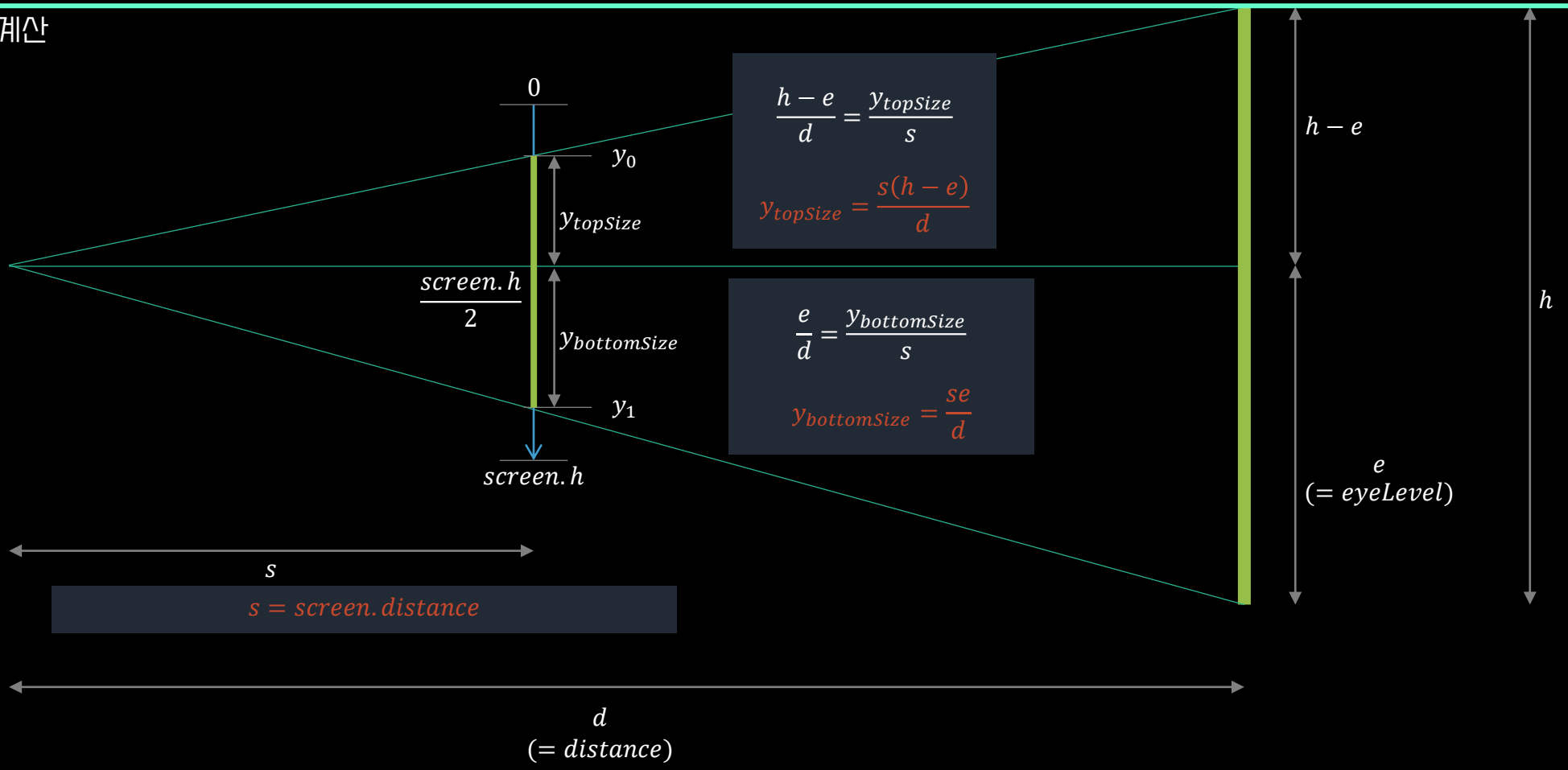
스크린 범위 계산



$$\tan\left(\frac{FOV}{2}\right) \cdot s = \frac{screen.w}{2} \rightarrow s = \cot\left(\frac{FOV}{2}\right)\frac{screen.w}{2} = screen.distance$$

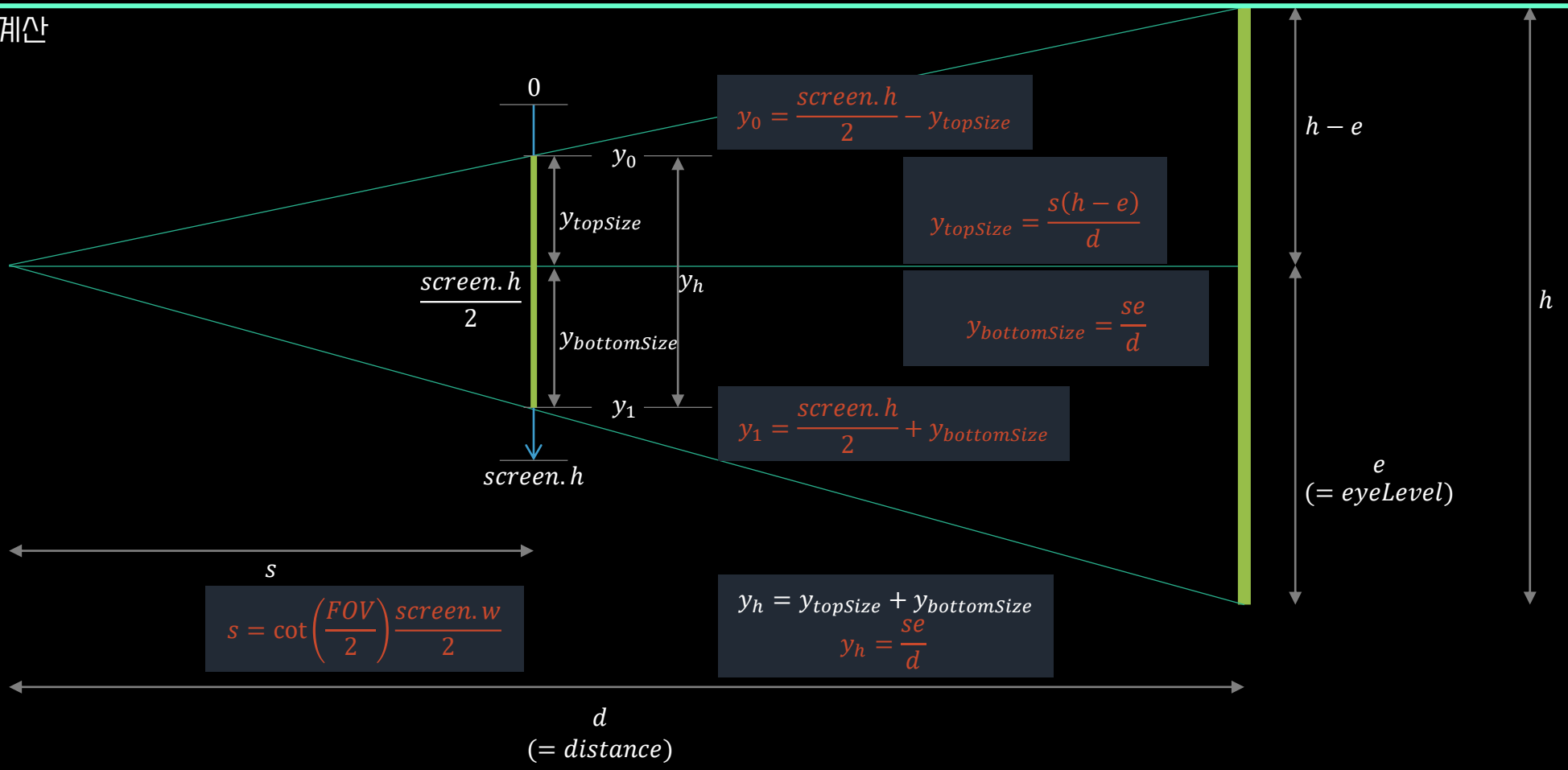
텍스처 그리기

스크린 범위 계산



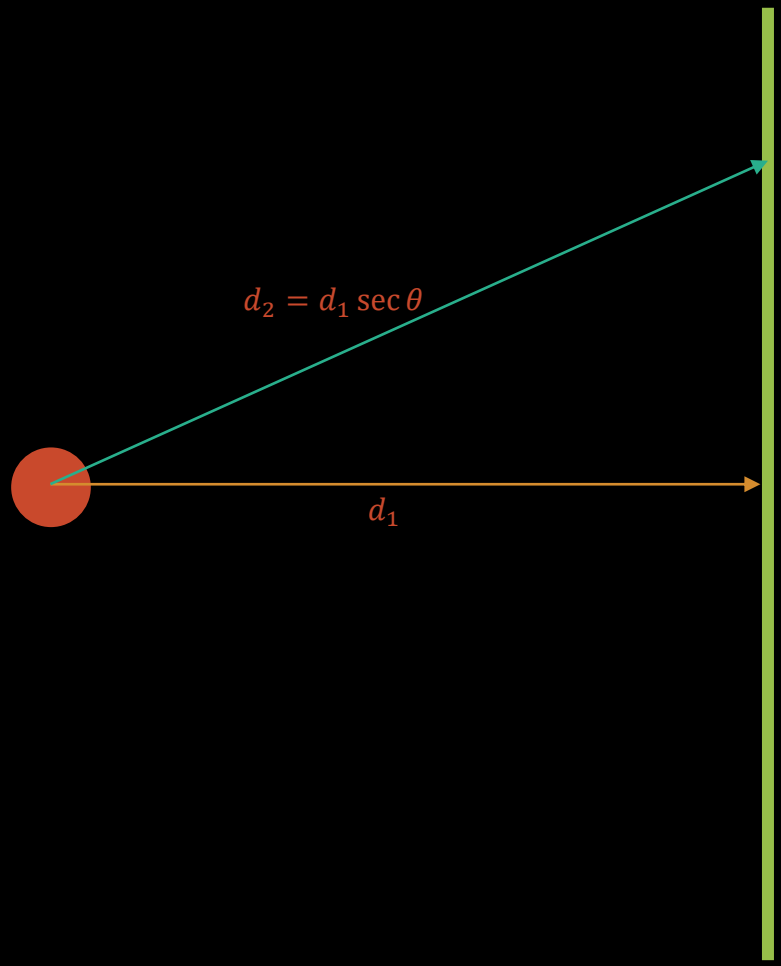
텍스처 그리기

스크린 범위 계산



텍스처 그리기

어안효과 보정

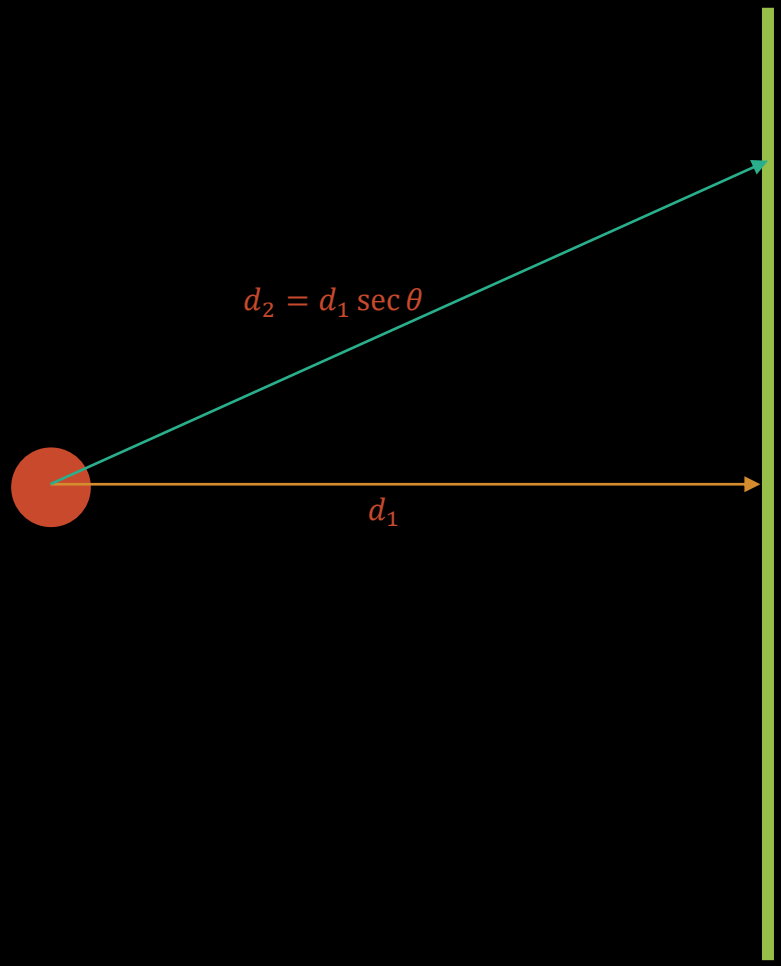


$$y_h = y_{topSize} + y_{bottomSize}$$

$$y_h = \frac{se}{d}$$

텍스처 그리기

어안효과 보정



$$y_h = y_{topSize} + y_{bottomSize}$$

$$y_h = \frac{se}{d}$$

$$y_{h2} = \frac{se}{d_2} = y_{h1} \cos\ \theta$$

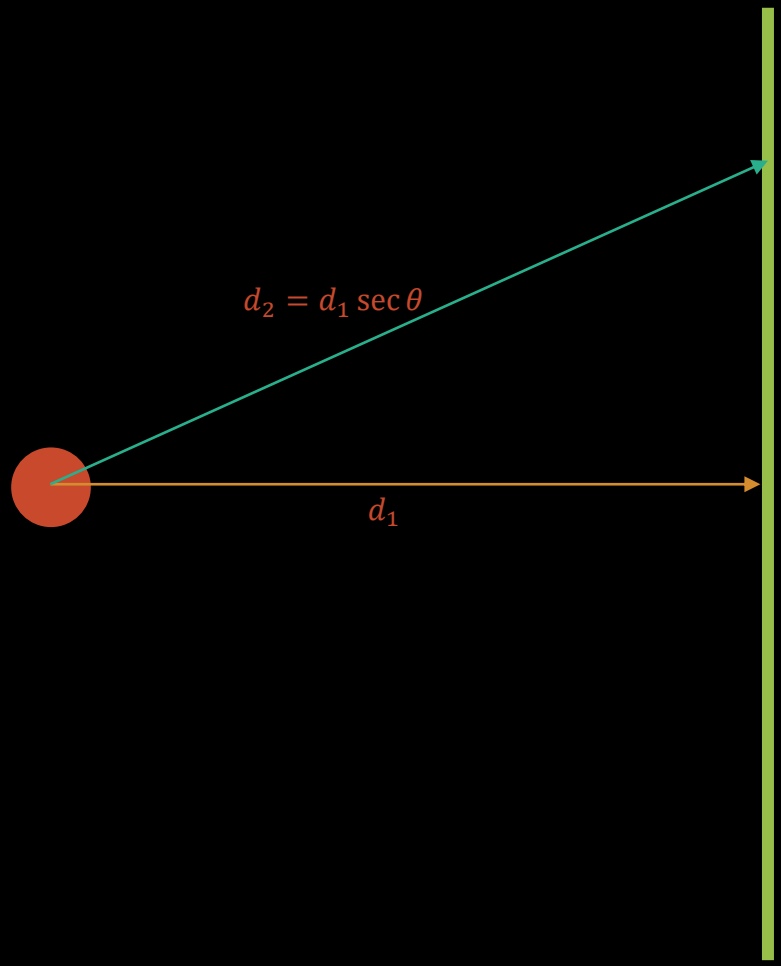
$$y_{h1} = \frac{se}{d_1}$$

텍스처 그리기

어안효과 보정

텍스처 그리기

어안효과 보정



$$y_h = y_{topSize} + y_{bottomSize}$$

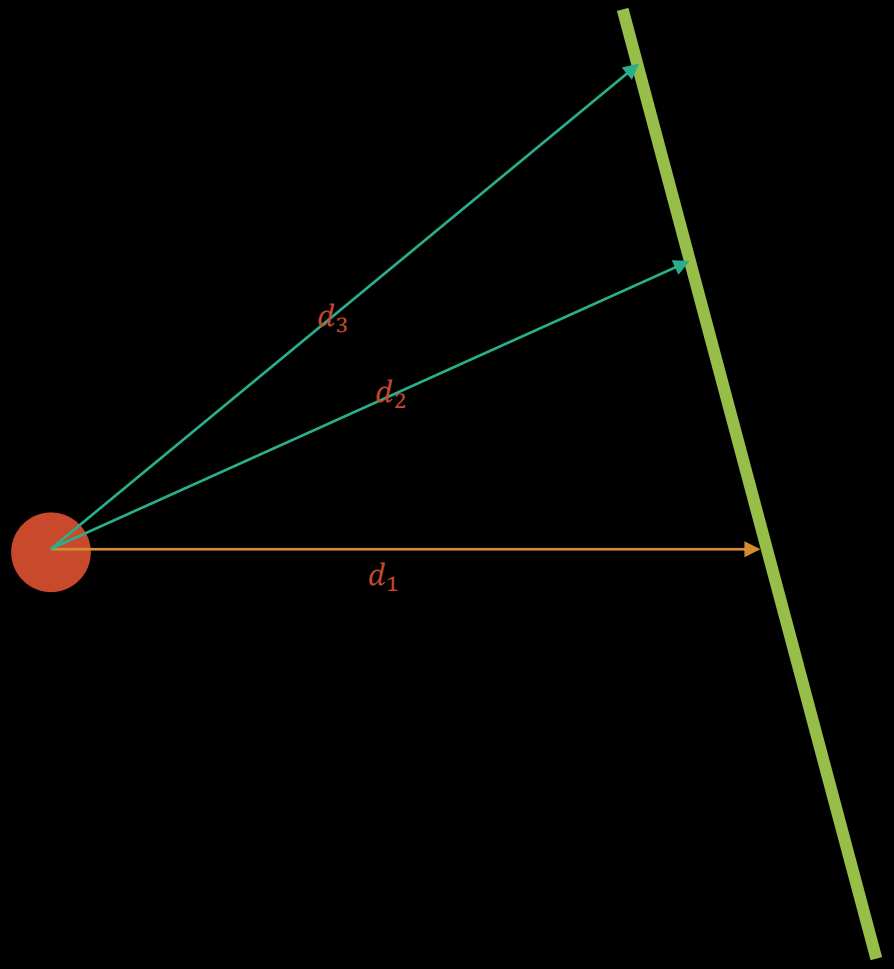
$$y_h = \frac{se}{d}$$

$$y_{h2} = \frac{se}{d_2} = y_{h1} \cancel{\cos\theta}$$

$$y_{h1} = \frac{se}{d_1}$$

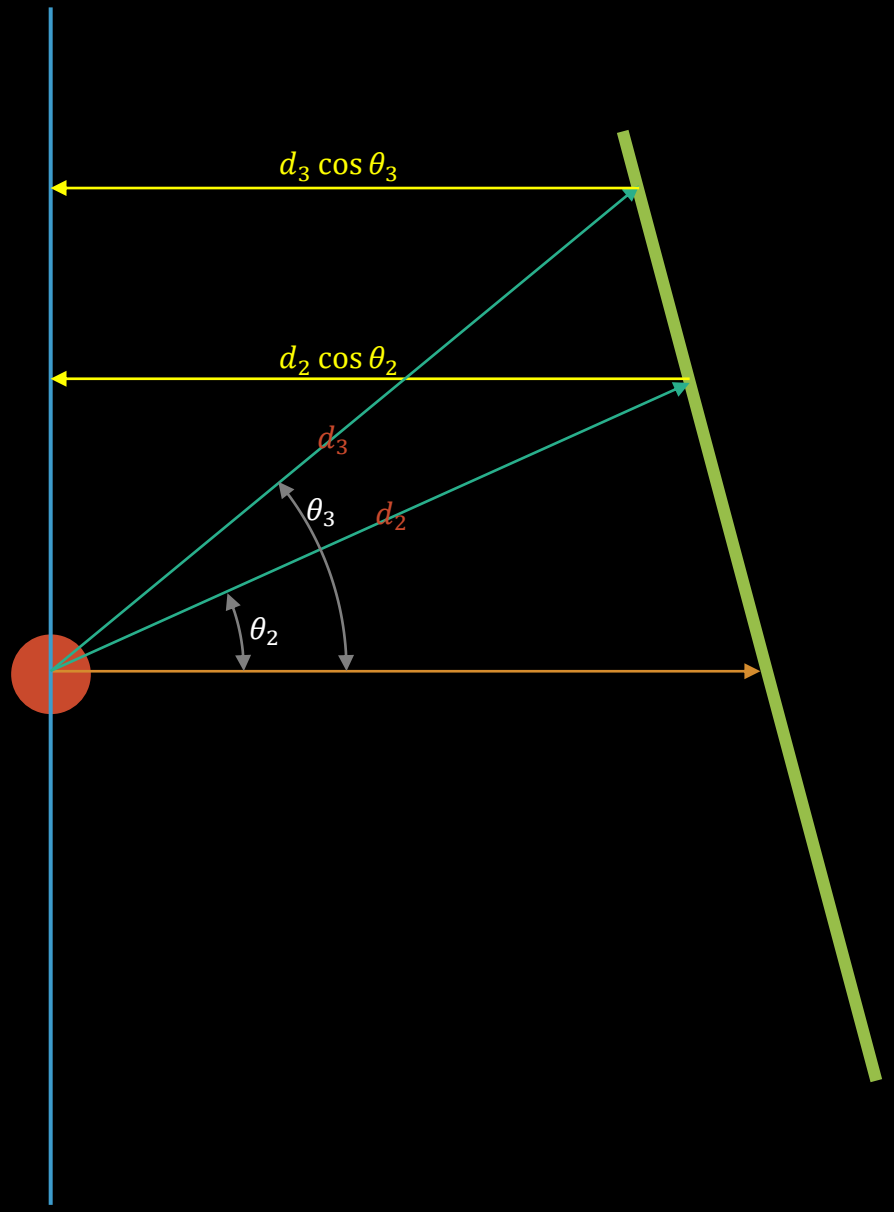
텍스처 그리기

어안효과 보정



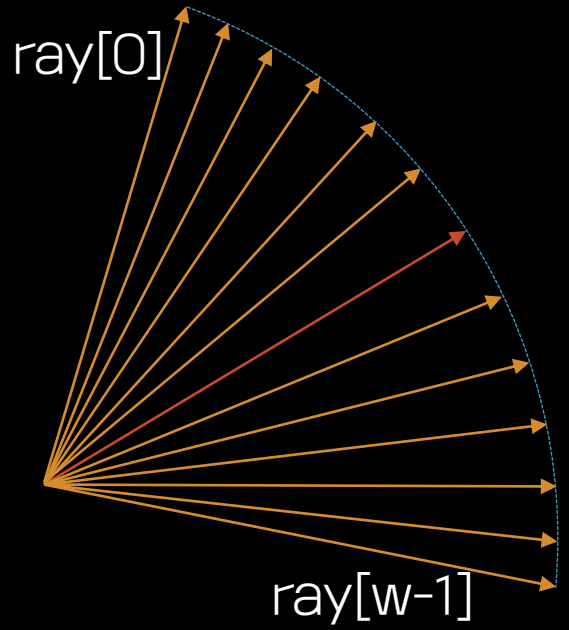
텍스처 그리기

어안효과 보정

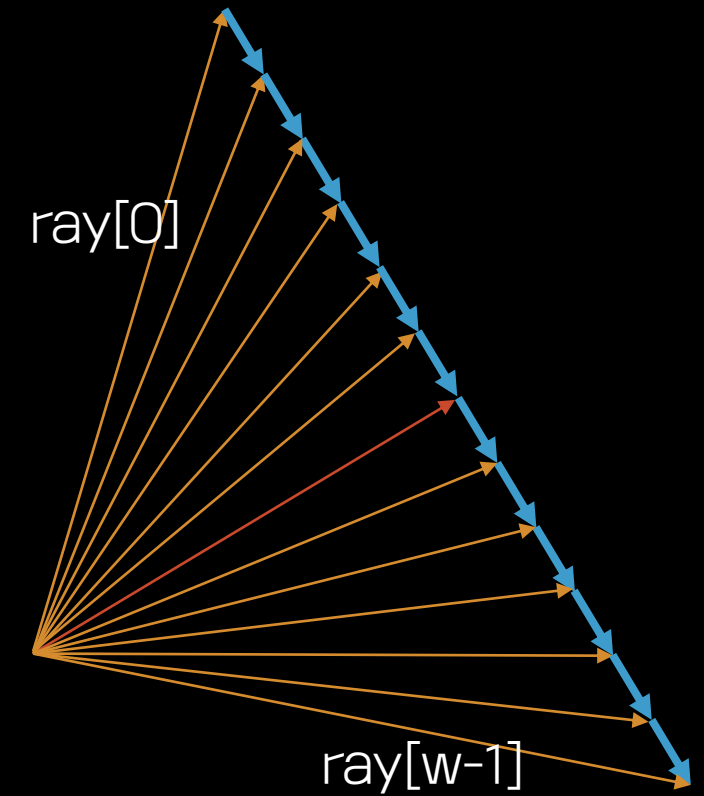


텍스처 그리기

어안효과 보정

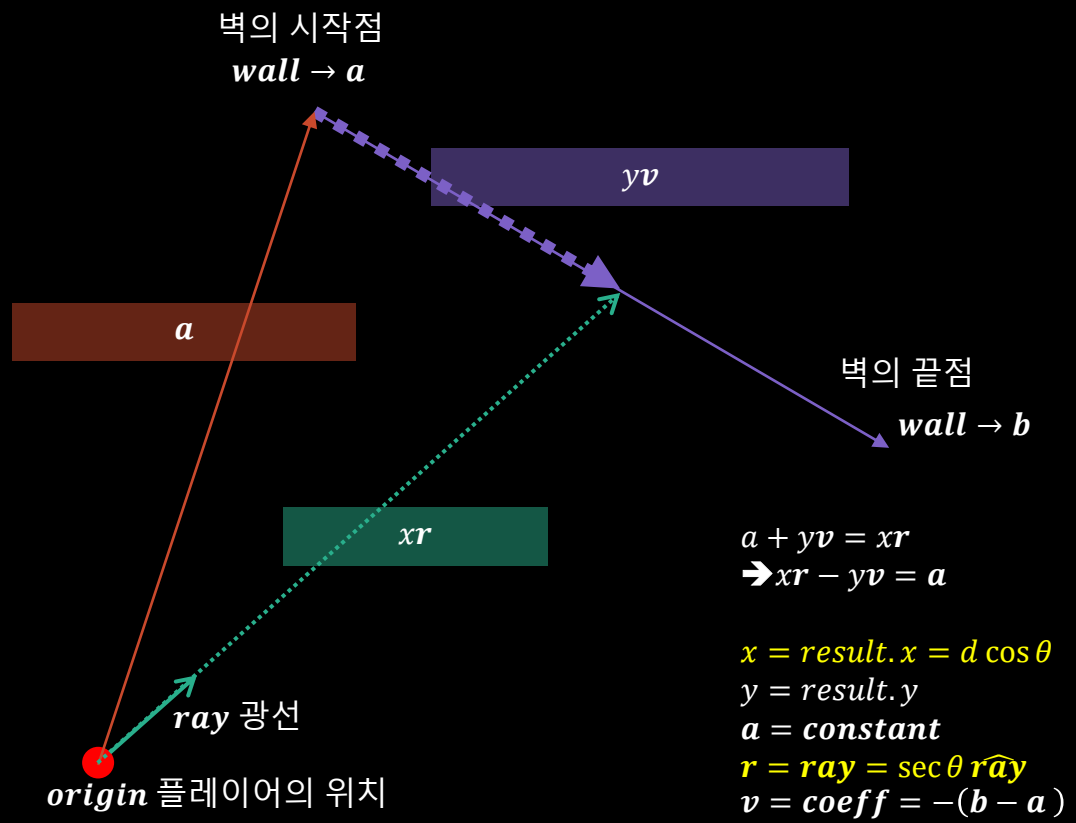


VS



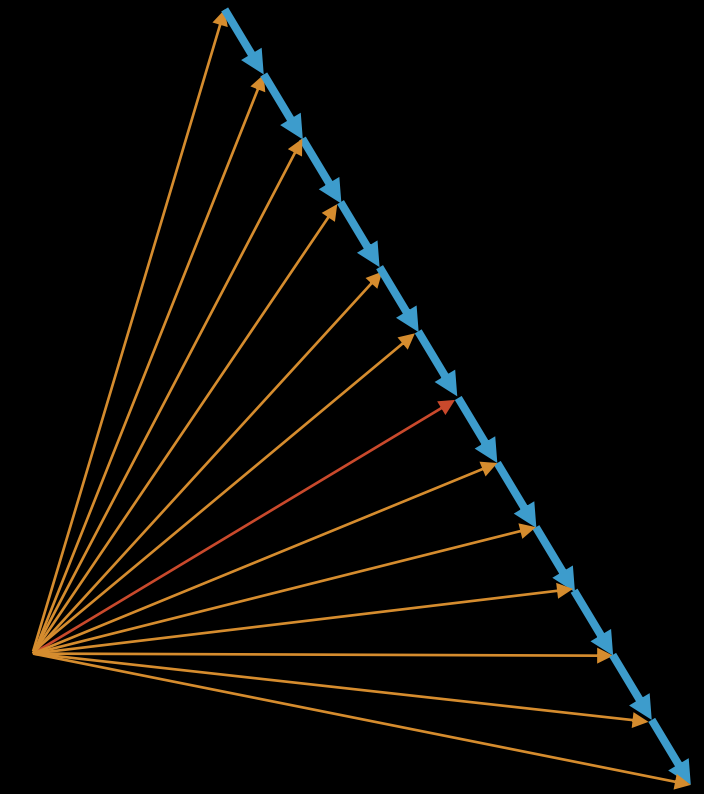
텍스처 그리기

어안효과 보정



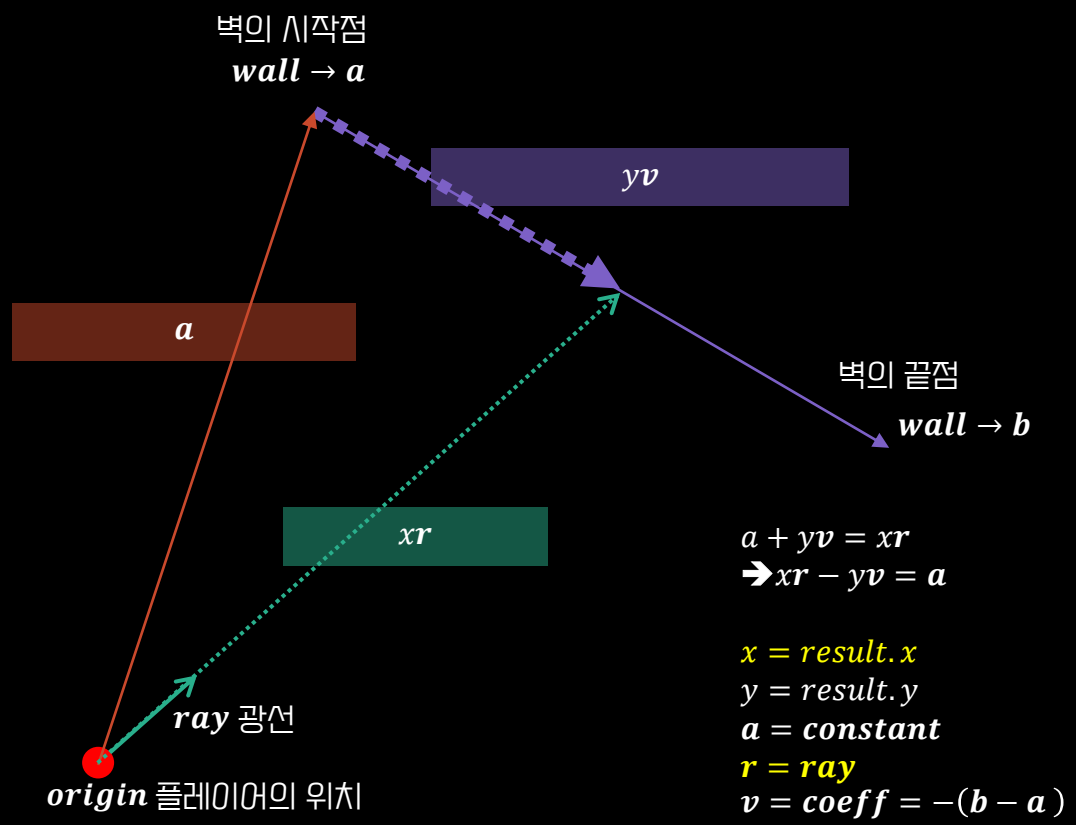
$a + yv = xr$
➔$xr - yv = a$

$x = result.x = d\cos\theta$
$y = result.y$
$a = constant$
$r = ray = \sec\theta\, \widehat{ray}$
$v = coeff = -(b - a)$



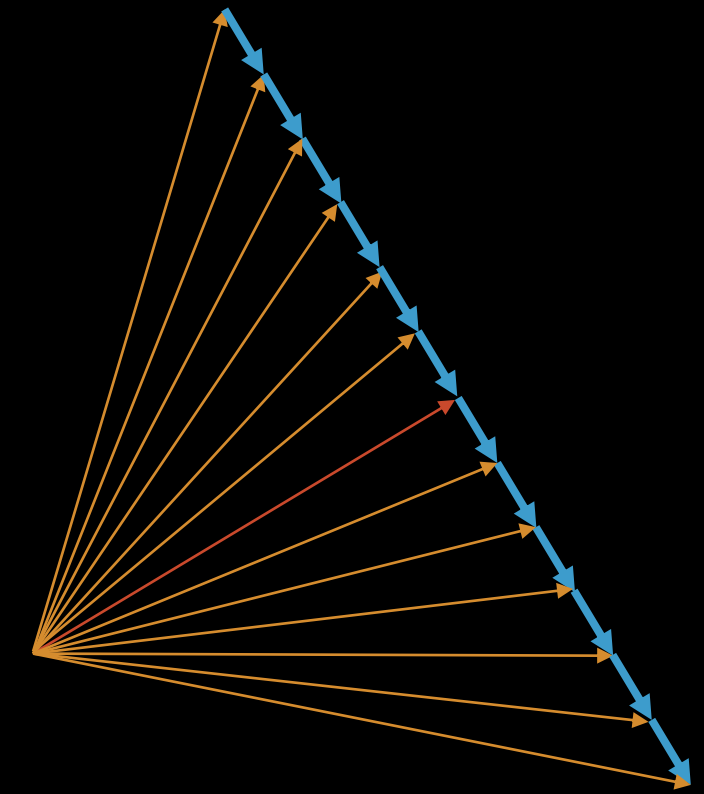
텍스처 그리기

어안효과 보정



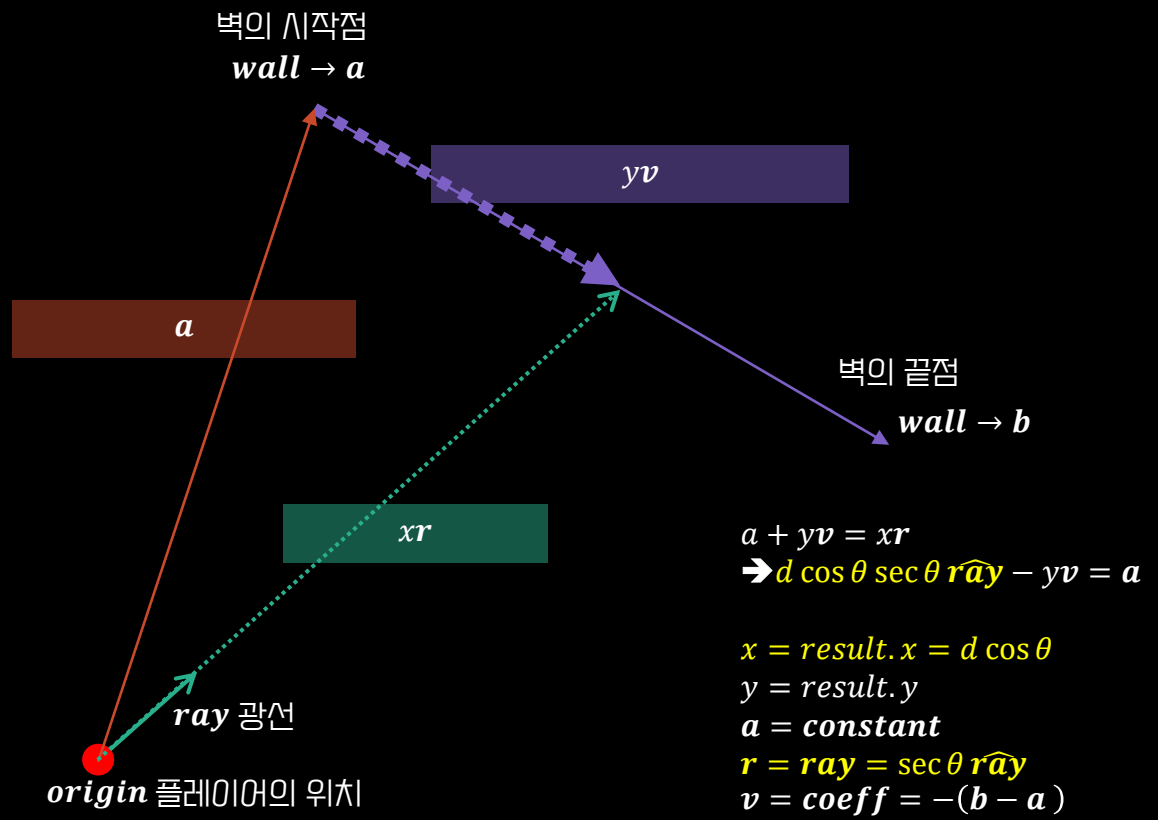
$a + yv = xr$
➔$xr - yv = a$

$x = result.x$
$y = result.y$
$a = constant$
$r = ray$
$v = coeff = -(b - a)$



텍스처 그리기

어안효과 보정



$a + yv = xr$

➔ $d \cos\theta \sec\theta \widehat{ray} - yv = a$

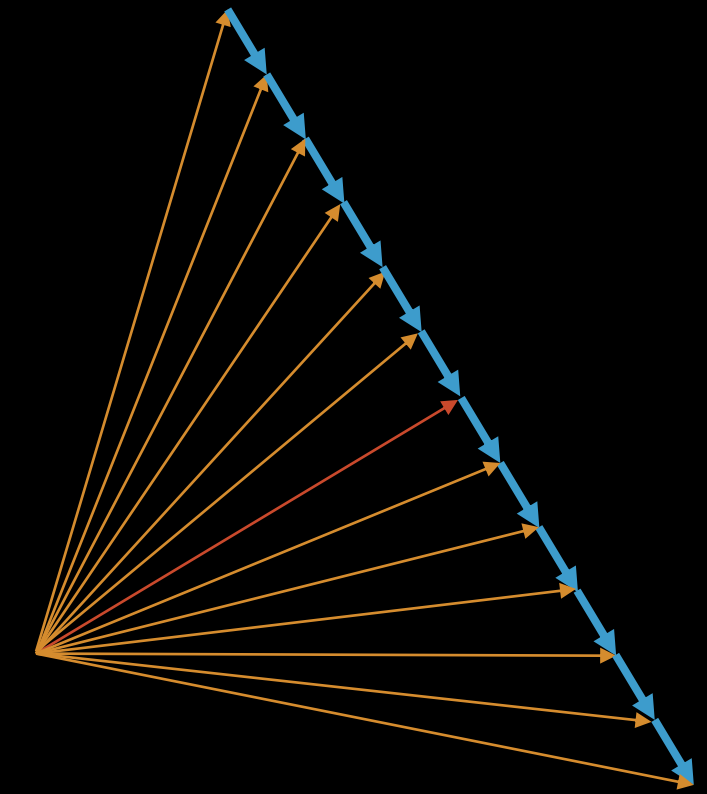
$x = result.x = d \cos\theta$

$y = result.y$

$a = constant$

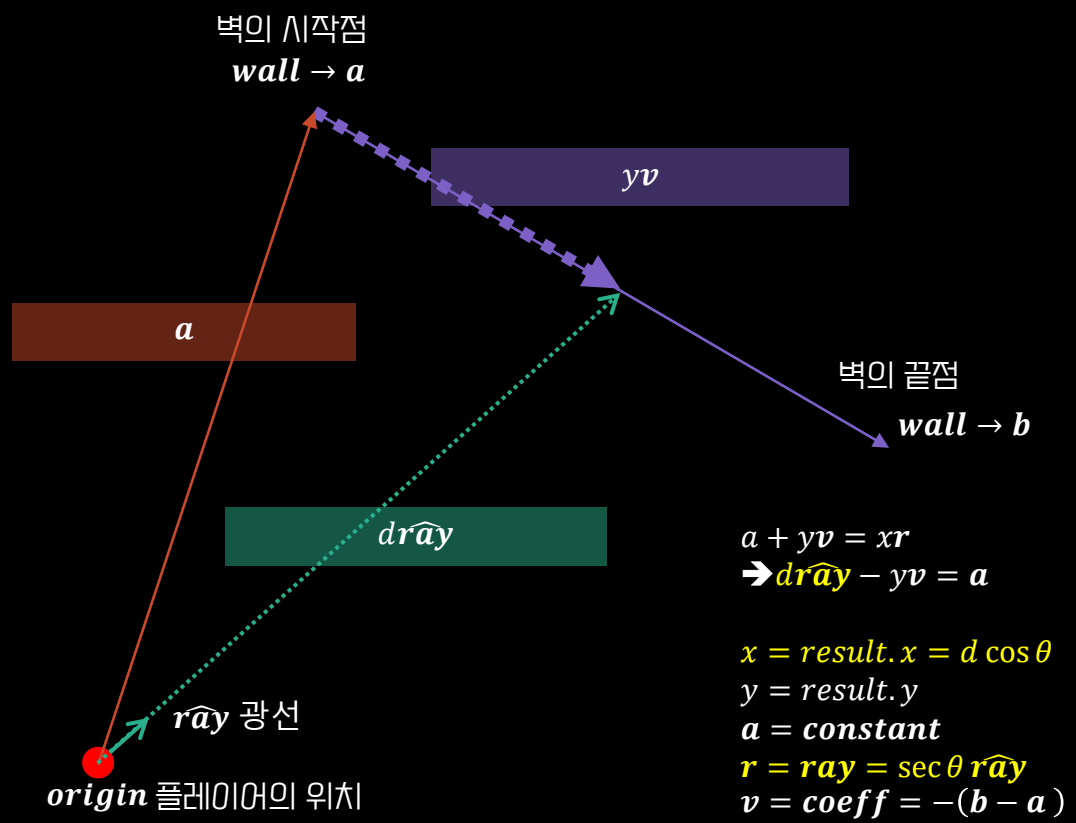
$r = ray = \sec\theta \widehat{ray}$

$v = coeff = -(b - a)$



텍스처 그리기

어안효과 보정



$a + yv = xr$

➔ $d\widehat{ray} - yv = a$

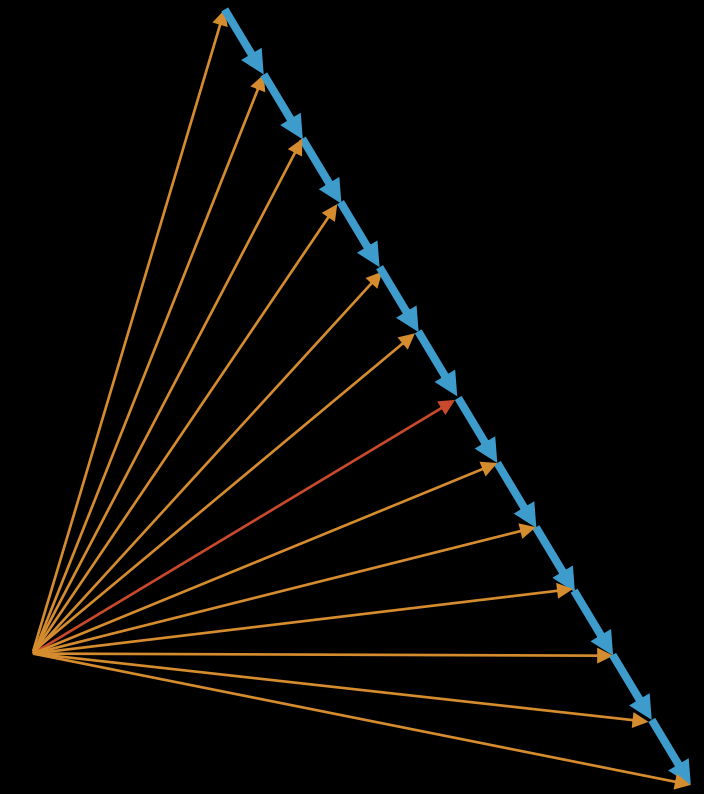
$x = result.x = d\cos\theta$

$y = result.y$

$a = constant$

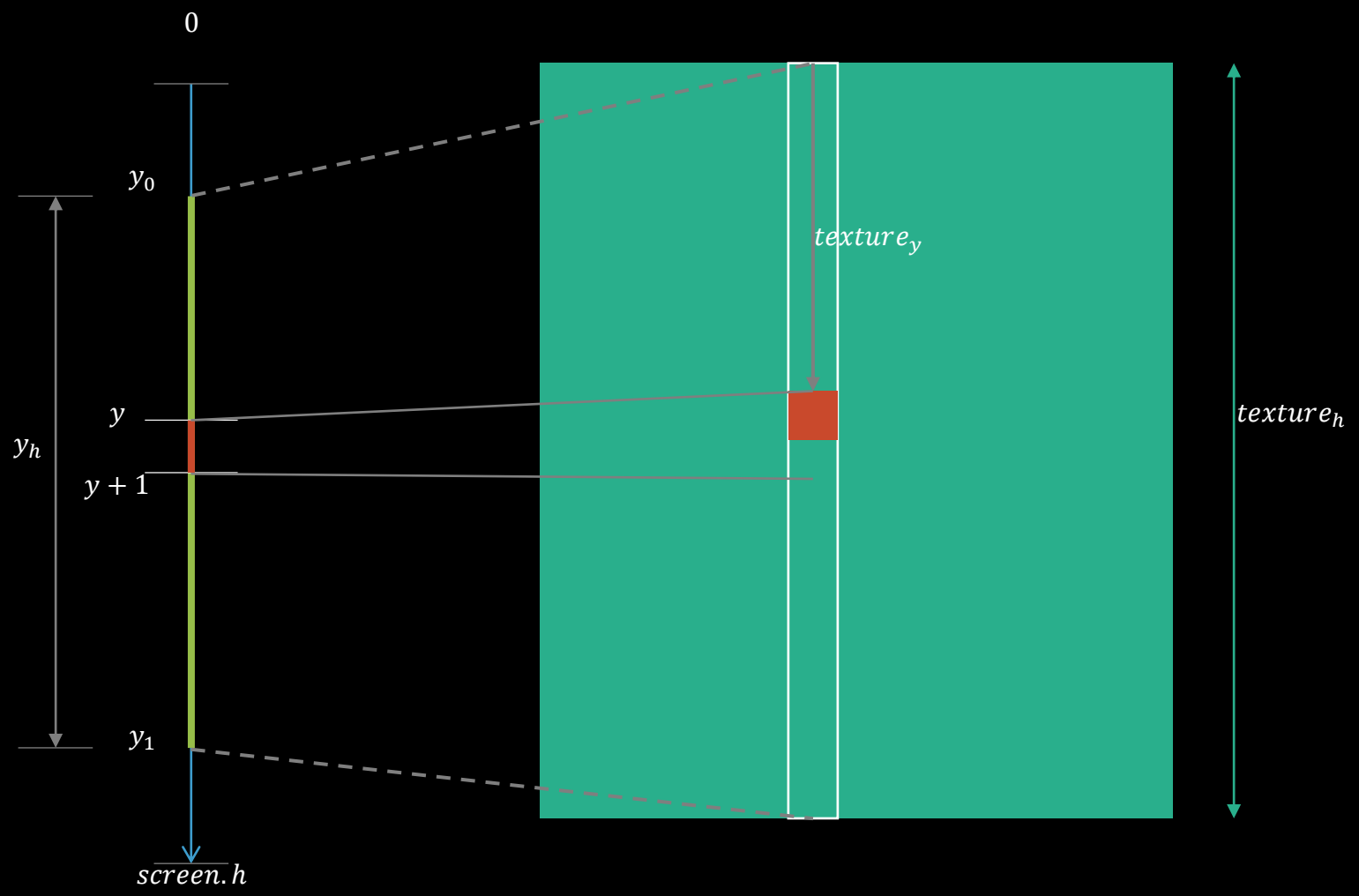
$r = ray = \sec\theta\,\widehat{ray}$

$v = coeff = -(b - a)$



텍스처 그리기

텍스처 범위 계산



$$y_1 = \frac{screen.h}{2} + y_{bottomSize}$$

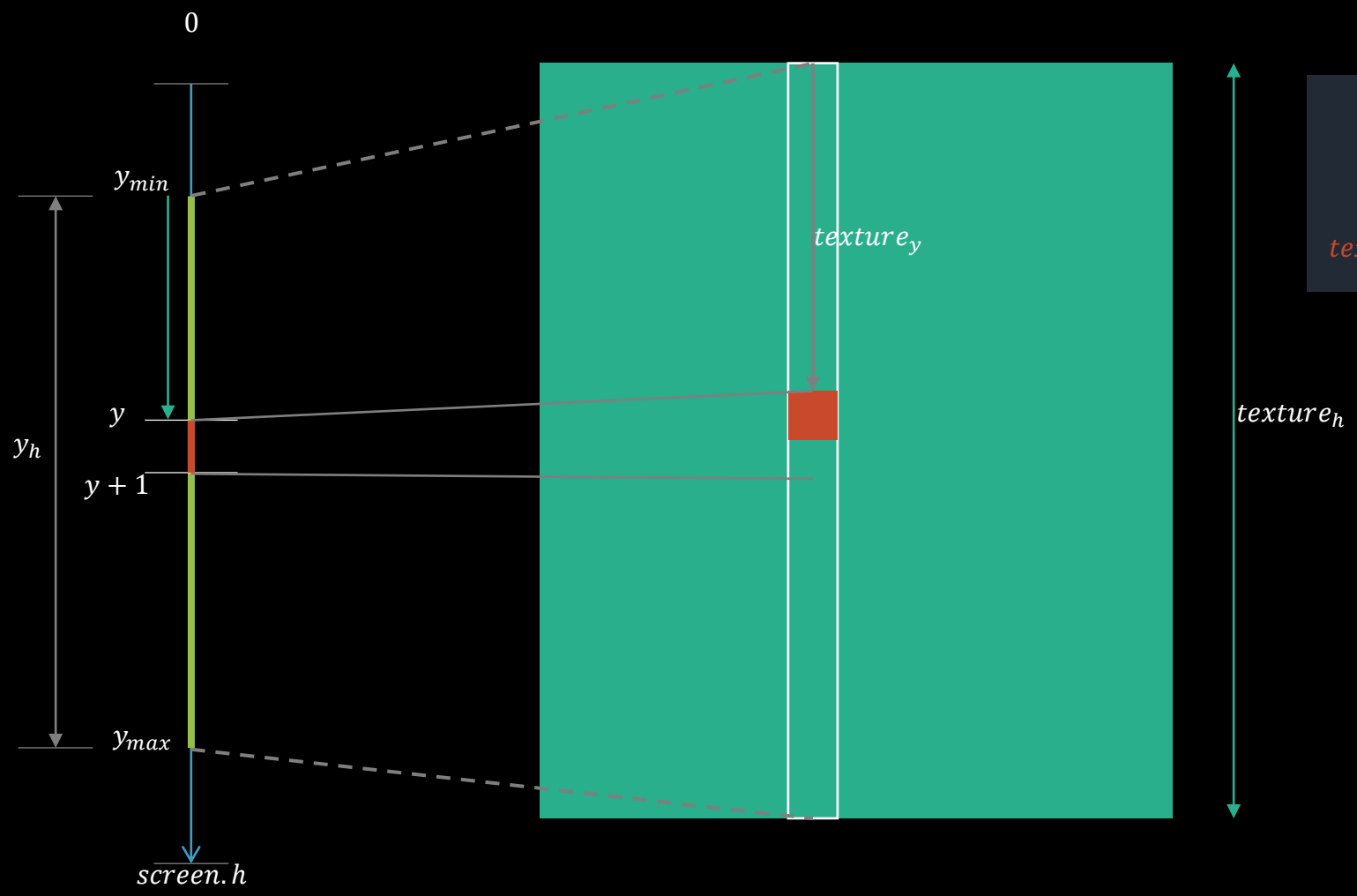
$$y_0 = \frac{screen.h}{2} - y_{topSize}$$

$$y_{bottomSize} = \frac{se}{d}$$

$$y_{topSize} = \frac{s(h-e)}{d}$$

텍스처 그리기

텍스처 범위 계산



$$\frac{y - y_0}{y_h} = \frac{texture_y}{texture_h}$$

$$texture_y = \frac{y - y_0}{y_h} texture_h$$

$$y_1 = \frac{screen.h}{2} + y_{bottomSize}$$

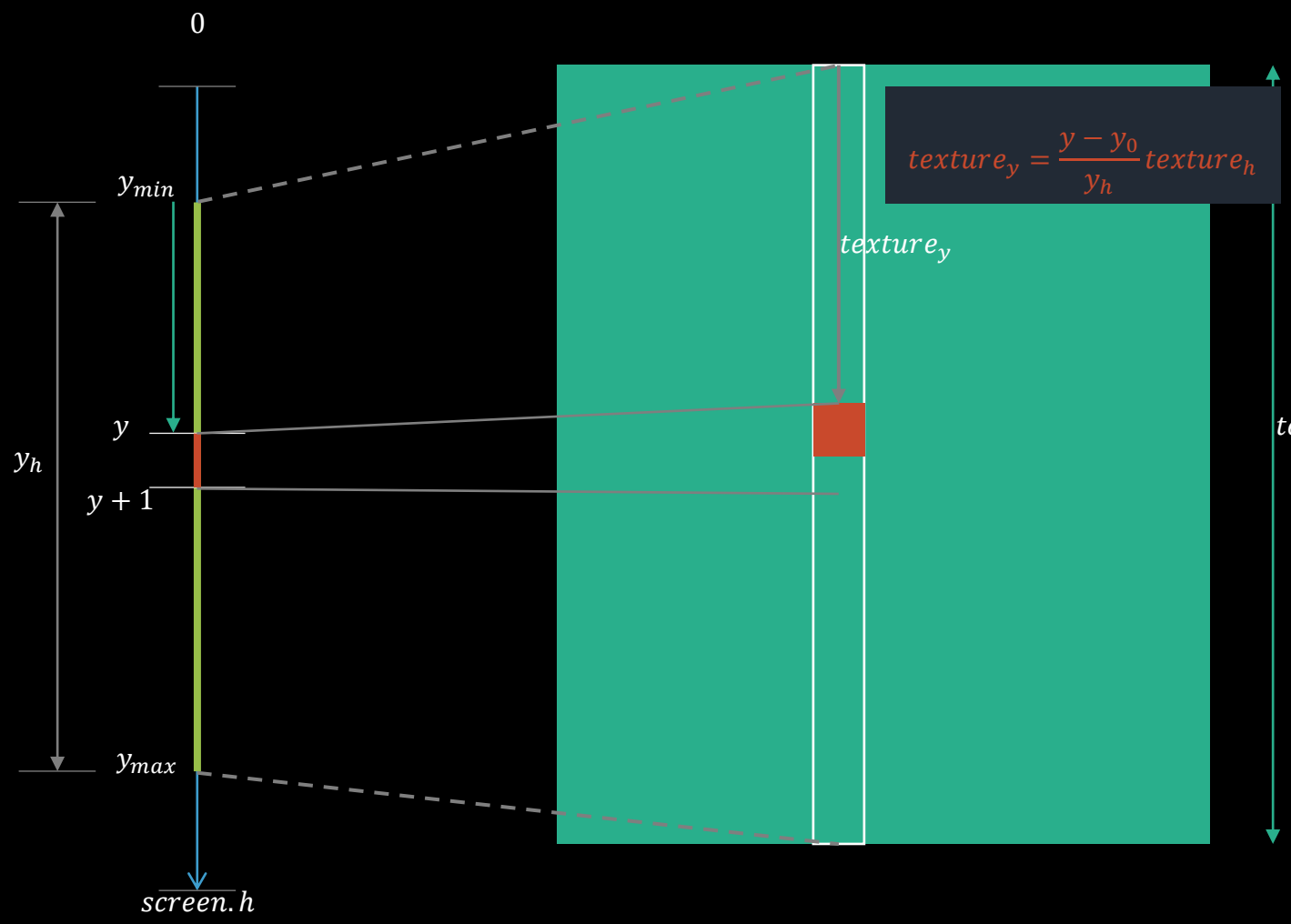
$$y_0 = \frac{screen.h}{2} - y_{topSize}$$

$$y_{bottomSize} = \frac{se}{d}$$

$$y_{topSize} = \frac{s(h - e)}{d}$$

텍스처 그리기

텍스처 범위 계산




```
while (y < y_h)
    y = y_0
    texture_y = (texture_h / y_h) y
    color = pickColor(texturePtr, texture_x, texture_y)
    putColor(screenImgPtr, x, y, color)
    y++
```

$texture_x = hitInfo \rightarrow y \cdot texture_w$

$s = screen.distance$

$y_1 = \frac{screen.h}{2} + y_{bottomSize}$

$y_0 = \frac{screen.h}{2} - y_{topSize}$

$y_{bottomSize} = \frac{se}{d}$

$y_{topSize} = \frac{s(h-e)}{d}$

텍스처 그리기

텍스처 범위 계산

```
void    draw_vertical(t_screen *screen, t_entity *wall, t_vec *hit_info, int x)
{
    int             screen_y0;
    int             screen_y1;
    int             screen_y_h;
    int             texture_x;
    int             texture_y;
    unsigned int    color;

    texture_x = hit_info->y * wall->texture->w;
    screen_y0 = -(WALL_H - EYE_LEVEL) / hit_info->x * screen->distance + screen->h / 2;
    screen_y1 = (EYE_LEVEL) / hit_info->x * screen->distance + screen->h / 2;
    screen_y_h = screen_y1 - screen_y0;
    screen_y0 = screen_y0 < 0 ? 0 : screen_y0;
    while (screen_y0 < screen_y1 && screen_y0 < screen->h)
    {
        texture_y = wall->texture->h - (screen_y1 - screen_y0) * wall->texture->h / (float)screen_y_h;
        if ((color = img_pick_color(wall->texture, texture_x, texture_y)) \
         != 0xff000000)
        {
            *(screen->pixel[x][screen_y0].color) = color;
            screen->pixel[x][screen_y0].distance = hit_info->x;
        }
        screen_y0++;
    }
}
```

$while\ (y < y_h)$

$y = y_0$

$texture_y = (y * texture_h) \,/\, y_h$

$color = pickColor(texturePtr, texture_x, texture_y)$

$putColor(screenImgPtr, x, y, color)$

$y++$

$texture_x = hitInfo \rightarrow y \cdot texture_w$

$$texture_y = \frac{y - y_0}{y_h} texture_h$$

$s = screen.distance$

$$y_{topSize} = \frac{s(h - e)}{d}$$

$$y_{bottomSize} = \frac{se}{d}$$

$$y_0 = \frac{screen.h}{2} - y_{topSize}$$

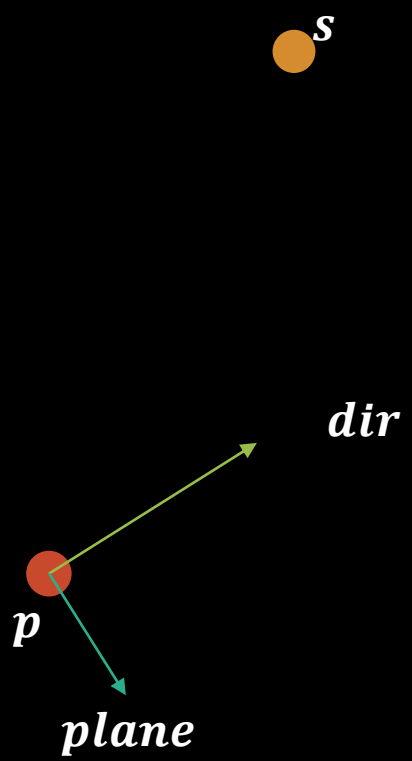
$$y_1 = \frac{screen.h}{2} + y_{bottomSize}$$

스프라이트 그리기

~~사파 야매~~ 알고리즘

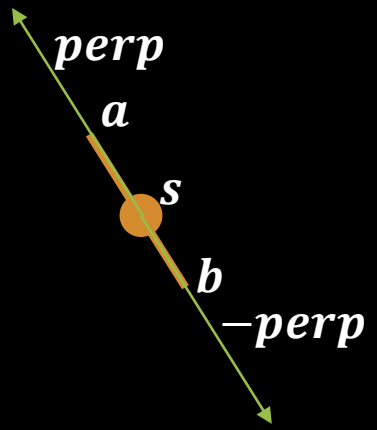
스프라이트 그리기

사파의 방법



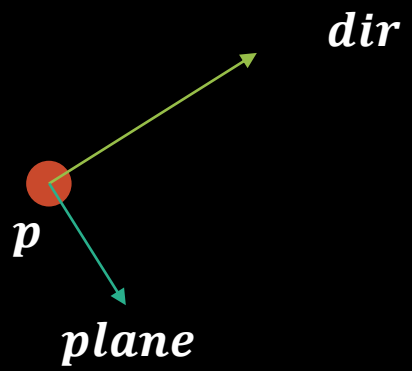
스프라이트 그리기

사파의 방법



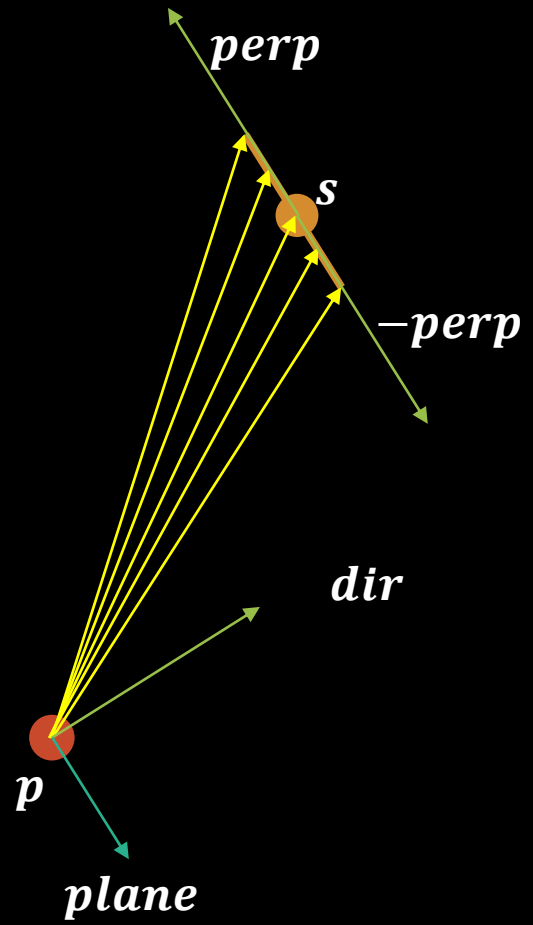
$$a = s + spriteSize \cdot perp$$
$$b = s - spriteSize \cdot perp$$

$$perp.x = dir.y$$
$$perp.y = dir.x$$



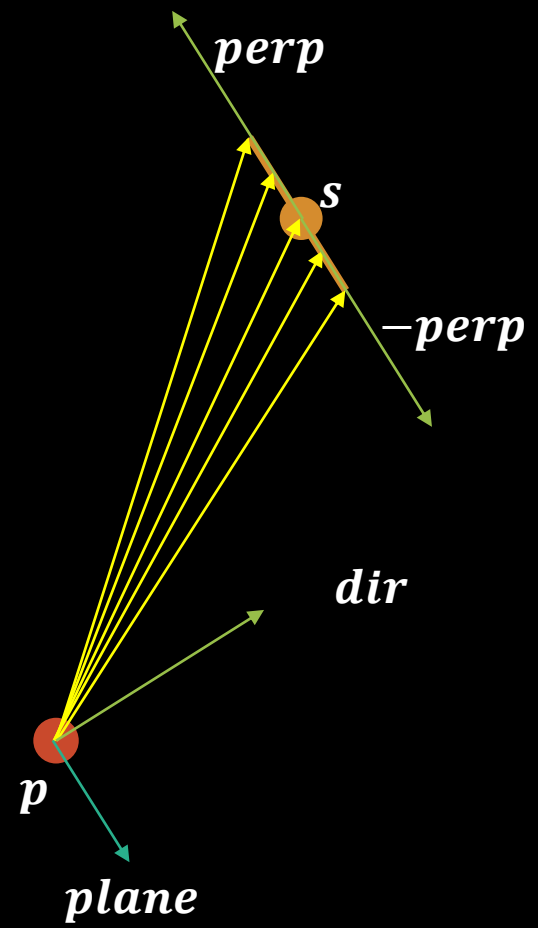
스프라이트 그리기

사파의 방법



스프라이트 그리기

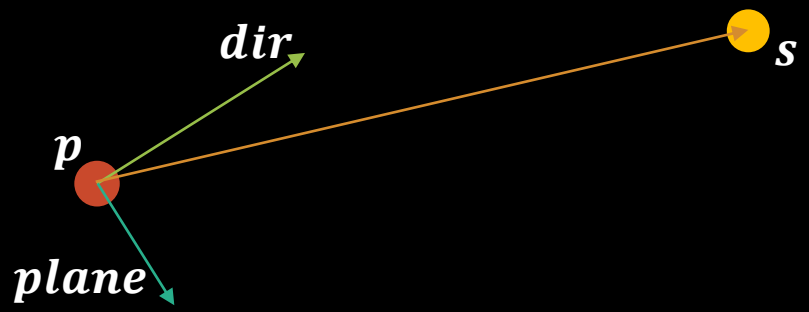
사파의 방법




```
void    draw_sprite(t_screen *screen, t_entity *sprite)
{
    t_entity    sprite_tmp;
    t_vec       perp;
    t_vec       range;
    int         min;
    int         max;
    t_vec       hit_info;
    t_ray       *ray;

    sprite_tmp.texture = sprite->texture;
    perp.x = -screen->dir.y * WALL_W / 2;
    perp.y = screen->dir.x * WALL_W / 2;
    while (sprite->texture)
    {
        sprite_tmp.a = vec_add(sprite->a, perp);
        sprite_tmp.b = vec_sub(sprite->a, perp);
        range = get_width_range(screen, &sprite_tmp);
        min = (int)range.x;
        max = (int)range.y;
        while (min < max)
        {
            ray = screen->ray + min;
            hit_info = ray_x_face(ray, &sprite_tmp, &screen->origin);
            if (0 < hit_info.x && 0 <= hit_info.y && hit_info.y <= 1)
            {
                if (hit_info.x < ray->distance)
                {
                    ray->distance = hit_info.x;
                    draw_vertical(screen, &sprite_tmp, &hit_info, min);
                }
            }
            min++;
        }
        sprite++;
    }
}
```

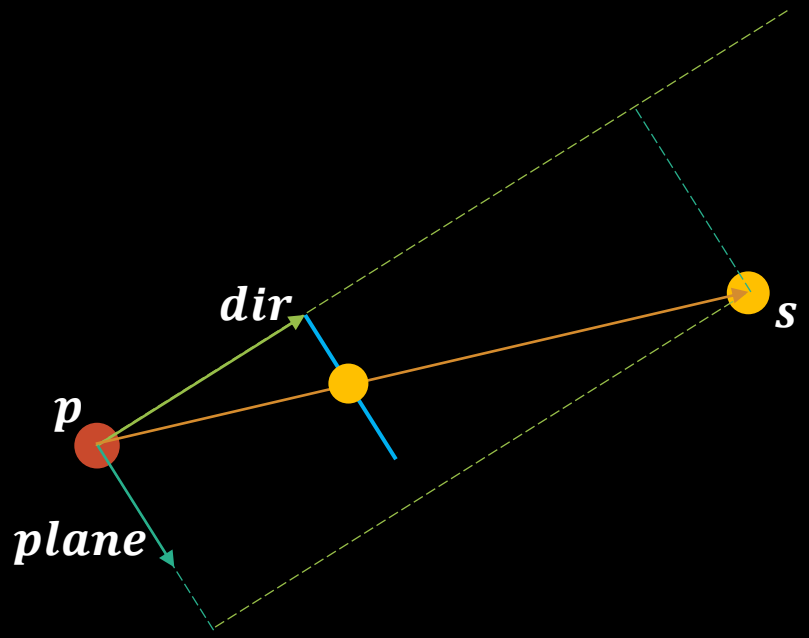
스프라이트 그리기

정파의 방법



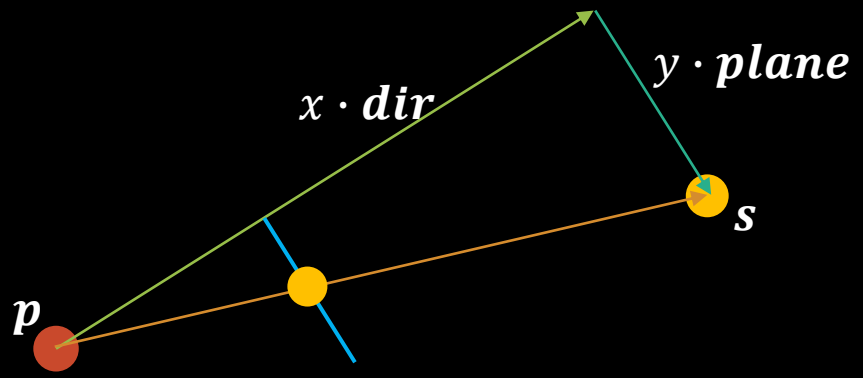
스프라이트 그리기

정파의 방법



스프라이트 그리기

정파의 방법



$$x\boldsymbol{dir} + y\boldsymbol{plane} = \boldsymbol{s} - \boldsymbol{p}$$

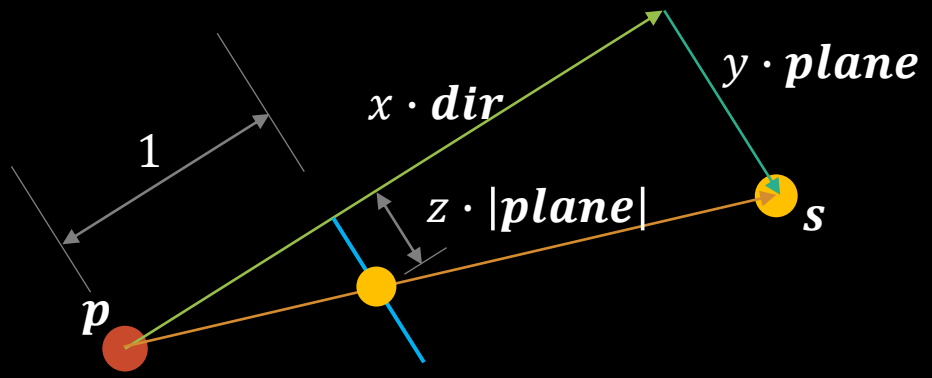
$$[\boldsymbol{dir} \quad \boldsymbol{plane}] \begin{bmatrix} x \\ y \end{bmatrix} = [\boldsymbol{s} - \boldsymbol{p}]$$

$$\begin{bmatrix} dir_x & plane_x \\ dir_y & plane_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} s_x - p_x \\ s_y - p_y \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} dir_x & plane_x \\ dir_y & plane_y \end{bmatrix}^{-1} \begin{bmatrix} s_x - p_x \\ s_y - p_y \end{bmatrix}$$

스프라이트 그리기

정파의 방법



$$x\boldsymbol{dir} + y\boldsymbol{plane} = \boldsymbol{s} - \boldsymbol{p}$$

$$[\boldsymbol{dir} \quad \boldsymbol{plane}] \begin{bmatrix} x \\ y \end{bmatrix} = [\boldsymbol{s} - \boldsymbol{p}]$$

$$\begin{bmatrix} dir_x & plane_x \\ dir_y & plane_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} s_x - p_x \\ s_y - p_y \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} dir_x & plane_x \\ dir_y & plane_y \end{bmatrix}^{-1} \begin{bmatrix} s_x - p_x \\ s_y - p_y \end{bmatrix}$$

$$\frac{1}{z|\boldsymbol{plane}|} = \frac{x|\boldsymbol{dir}|}{y|\boldsymbol{plane}|}$$

$$z = \frac{y}{x}$$

스프라이트 그리기

정파의 방법