Lecture #14. 직선 이동

2D 게임 프로그래밍

이대현 교수

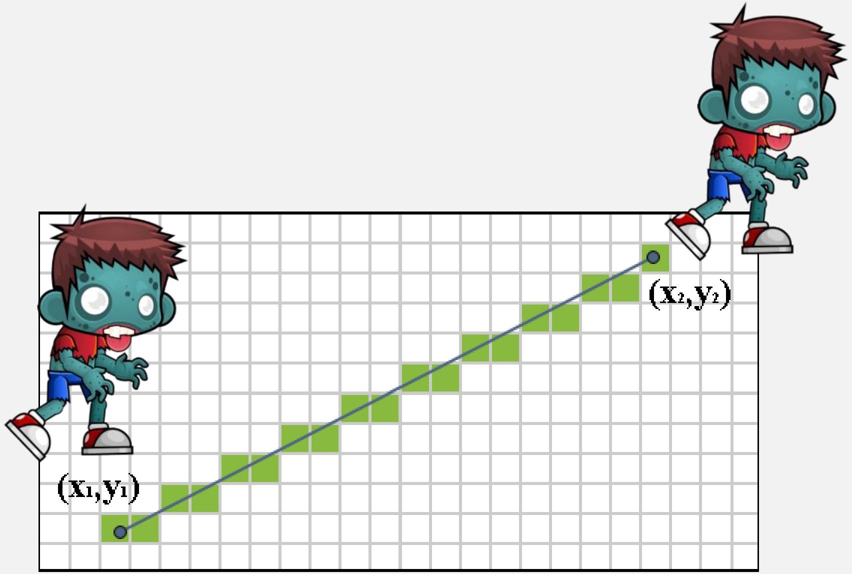
한국공학대학교
TECH UNIVERSITY OF KOREA

학습 내용

- 직선 방정식을 이용한 직선 이동
- Parametric Representaion을 이용한 직선 이동
- 좀비 직선 이동

좀비를 (x1,y1)에서 (x2,y2)까지 이동시키기

1. **물리 공식을 이용. 즉, 속도와 시간을 이용**
   - s = s0 + v*t
2. **선분을 그리는 방법을 이용**
   - 선분 상에 점(캐릭터의 이동 위치)을 찍어보자.
   - 선분을 정확히 그리려면, bresenhem algorithm을 이용해야 한다.



Bresenhem Line Algorithm

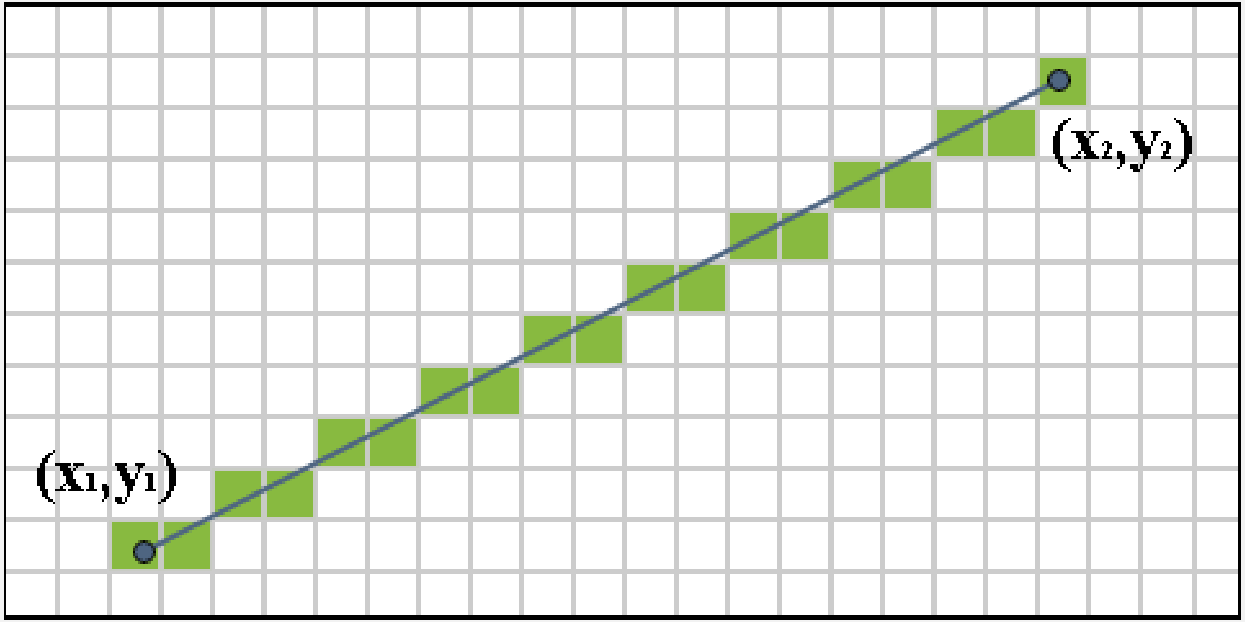
- 컴퓨터 화면 상에 직선(선분)을 그리는 알고리즘.
- 덧셈과 뺄셈만을 이용함으로써, 고속으로 직선을 그릴 수 있음.

The Bresenham Line Algorithm

BRESENHAM'S LINE DRAWING ALGORITHM
(for $|m| < 1.0$)

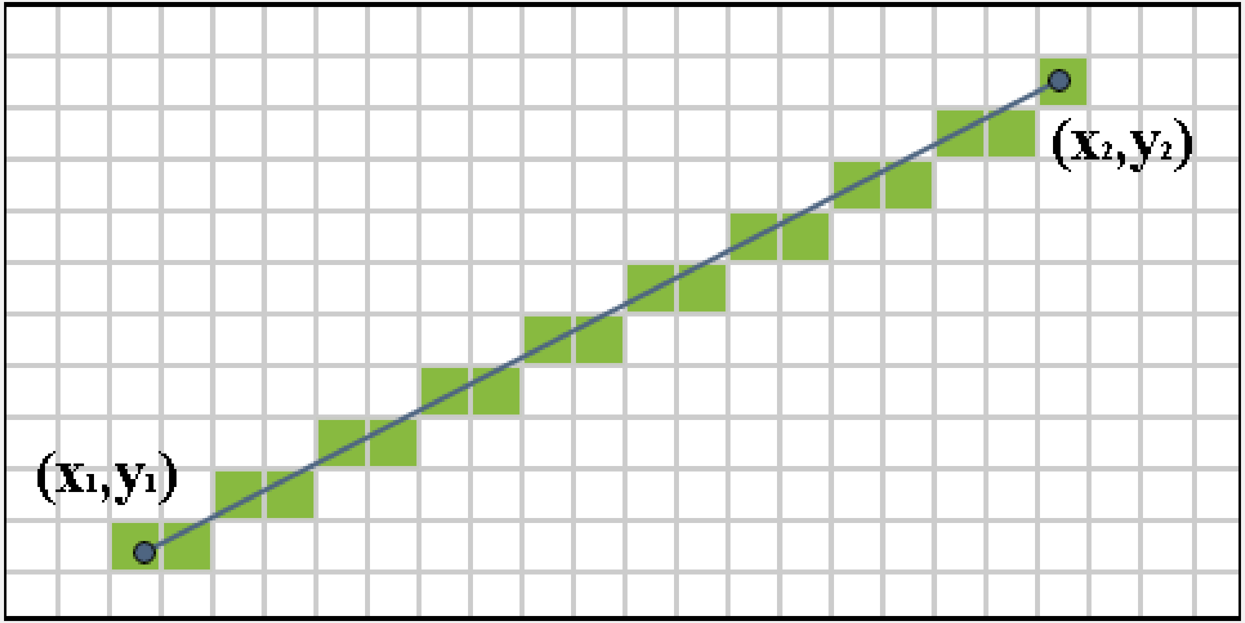
1. Input the two line end-points, storing the left end-point in (x_0, y_0)
2. Plot the point (x_0, y_0)
3. Calculate the constants Δx, Δy, $2\Delta y$, and $(2\Delta y - 2\Delta x)$ and get the first value for the decision parameter as:
$$p_0 = 2\Delta y - \Delta x$$
4. At each x_k along the line, starting at $k = 0$, perform the following test. If $p_k < 0$, the next point to plot is (x_k+1, y_k) and:
$$p_{k+1} = p_k + 2\Delta y$$

(x1,y1)에서 (x2,y2)까지의 선분 상에 어떻게 점을 찍을까?



무식한 방법?

- **선분 상의 점들의 리스트를 작성**
  - [(2,1), (3,1), (4,2),(5,2), ---------------------, (20, 10)]
  - 무식하지만? 장점도 있다? 뭘까?



(x1,y1)에서 (x2,y2)까지의 선분 상에 점 찍기

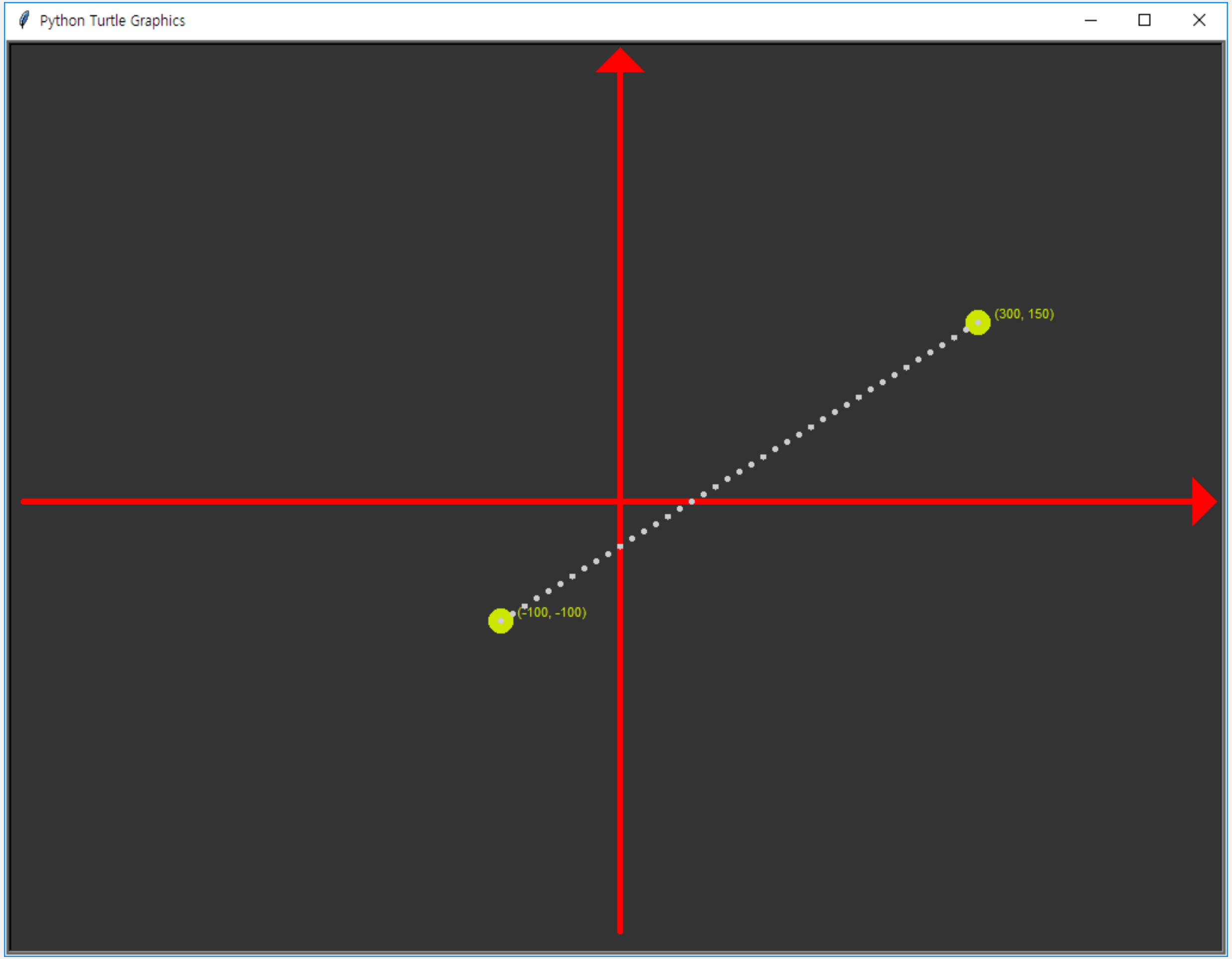
- 직선의 방정식을 구한다.
  - y = ax + b
  - a = (y2-y1)/(x2-x1)
  - b = y1 – x1 * a
- x를 x1부터 x2까지 (일정간격) 변화시켜가면서, y 값을 계산한다.

```
a = (y2-y1)/(x2-x1)
b = y1 - x1 * a

for x in range(x1, x2 + 1, 10):
    y = a * x + b
```

line.py


```python
def draw_line(p1, p2):
    draw_big_point(p1)
    draw_big_point(p2)

    x1, y1 = p1
    x2, y2 = p2

    a = (y2-y1)/(x2-x1)
    b = y1 - x1 * a
    for x in range(x1, x2 + 1, 10):
        y = a * x + b
        draw_point((x, y))

    draw_point(p2)
```

```python
prepare_turtle_canvas()

draw_line((-100,-100),(300,150))

turtle.done()
```

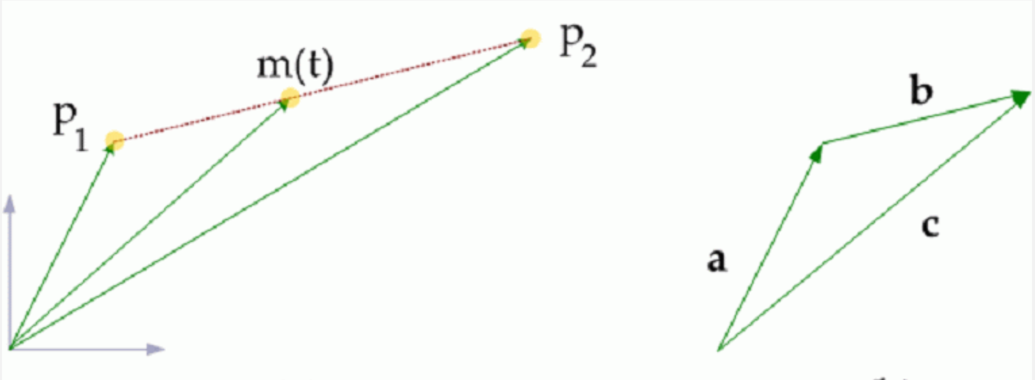

Python Turtle Graphics
(300, 150)
(-100, -100)

문제점?

- y축과 평행인 직선(x=c)을 그릴 수 없음.

- 해결책은?

Parametric Representation of Lines



(a) (b)

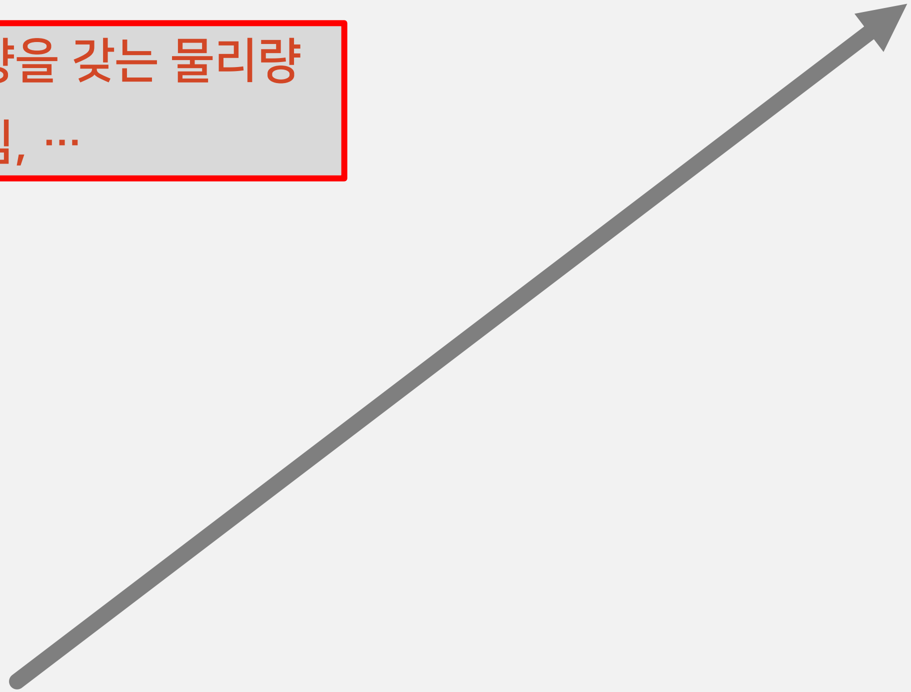
$$m(t) = p_1 + t\,(p_2 - p_1) = (1 - t)\,p_1 + t\,p_2 \quad (0 \leq t \leq 1)$$

$$\begin{aligned} m(t) &= p_1, \text{ at } t = 0 \\ &= p_2, \text{ at } t = 1 \end{aligned}$$

벡터

크기와 방향을 갖는 물리량

예) 속도, 힘, …



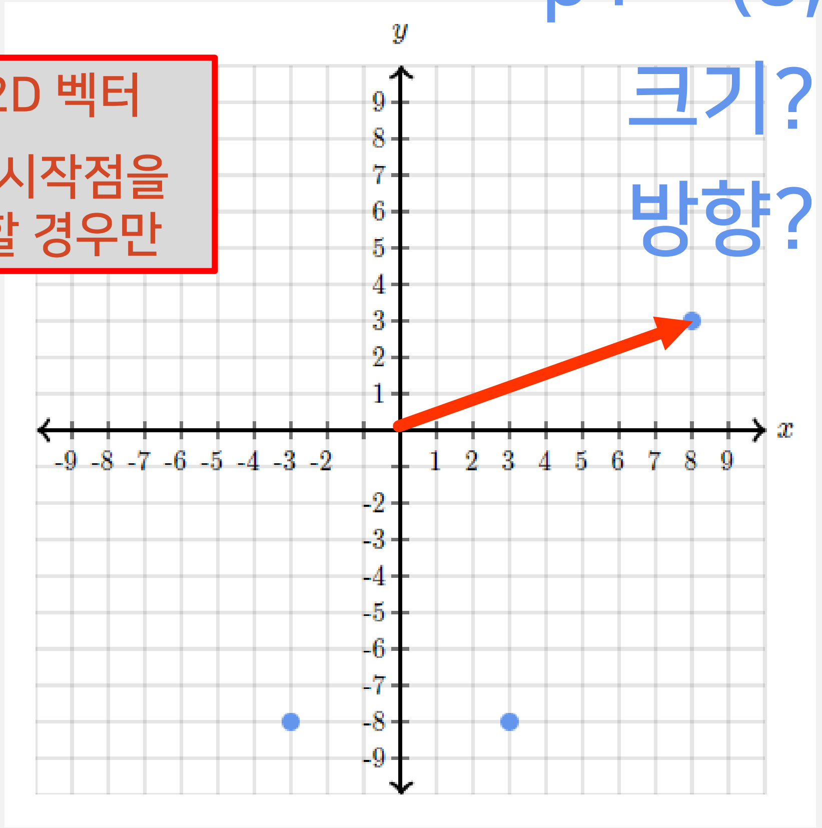
p1 = (8,3)

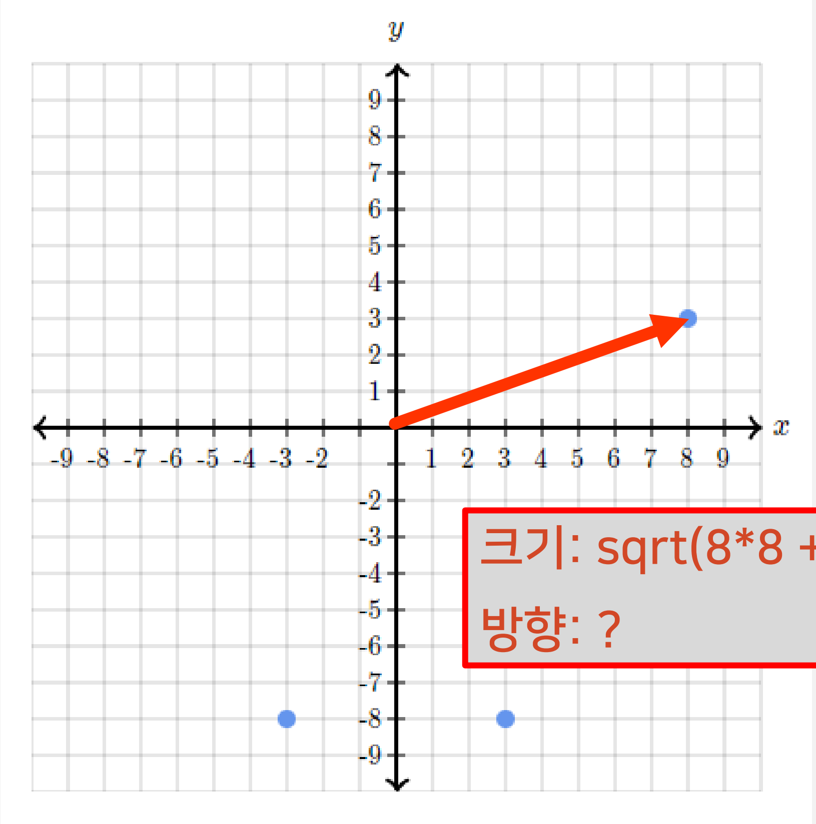
크기?

방향?

2D 점 = 2D 벡터

단, 벡터의 시작점을 원점으로 할 경우만





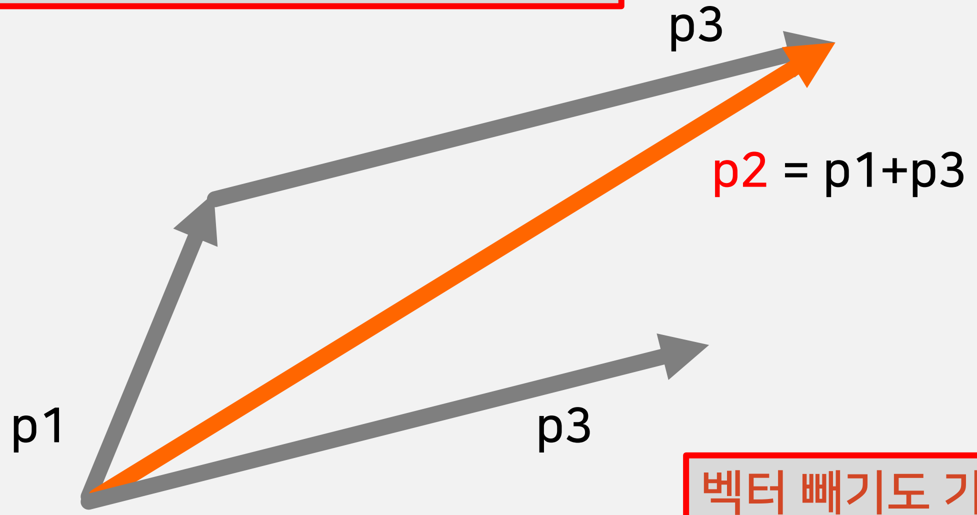
크기: sqrt(8*8 + 3*3)

방향: ?

벡터는 더할 수 있다.? 힘을 더하면?

p1+p3?

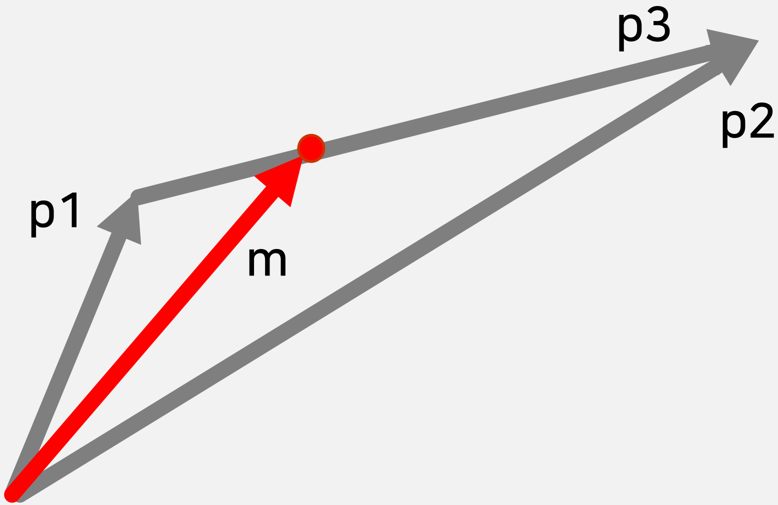
p1끝점과 p3시작점을 일치



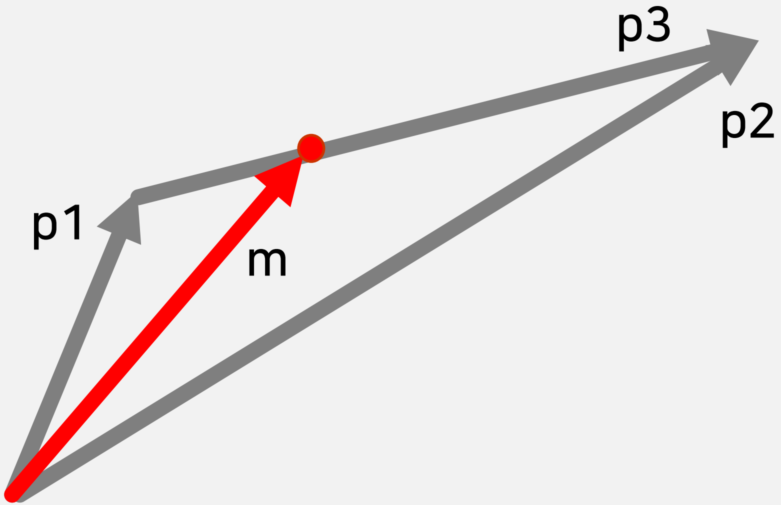
벡터 빼기도 가능함.

p3 = p2 – p1

p3 벡터 위 30% 지점의 점을 m이라고 하면, 벡터 m?



p3 벡터 위 30% 지점의 점을 m이라고 하면, 벡터 m?

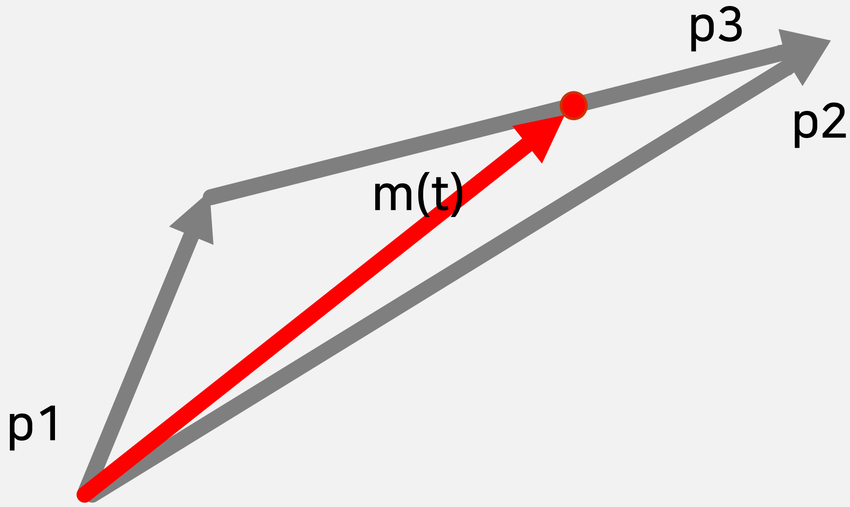


$$m = p1 + 0.3p3 = p1 + 0.3(p2 - p1) = 0.7p1 + 0.3p2$$

p3 벡터 위 t% 지점의 점을 m(t)라고 하면, 벡터 m(t)?

$$m(t) = p1 + t\ p3 = p1 + t(p2 - p1) = (1 - t)p1 + t\ p2$$

t의 범위: 0 <= t <=1

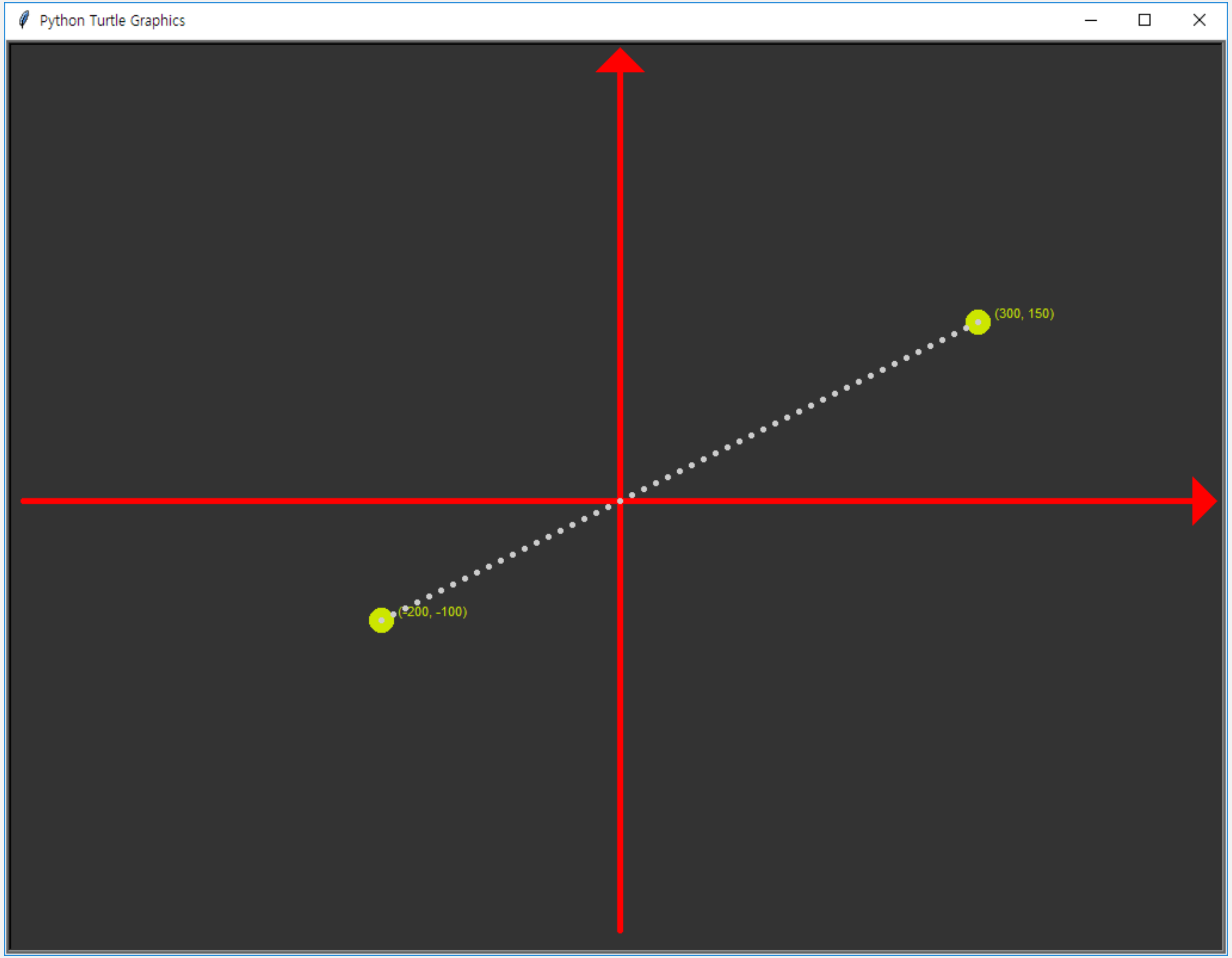


m(t)는 결국, p1과 p2를 1-t:t의 비율로 섞은 것임.

m(t)는 두 점 p1과 p2의 선형조합임.

line.py


```python
def draw_line(p1, p2):
    draw_big_point(p1)
    draw_big_point(p2)

    x1, y1 = p1
    x2, y2 = p2

    for i in range(0, 100+1, 4):
        t = i / 100
        x = (1-t)*x1 + t*x2
        y = (1-t)*y1 + t*y2
        draw_point((x, y))

    draw_point(p2)
```


Python Turtle Graphics
(300, 150)
(-200, -100)

zombie.py – 좀비의 직선 이동

```python
def __init__(self):
    # 중략
    self.t = 0.0
    self.sx, self.sy = self.x, self.y
```

```python
def update(self):
    # 중략
    if self.t < 1.0:
        self.t += 0.01
        self.x = self.sx * (1.0 - self.t) + (self.arrow.x * self.t)
        self.y = self.sy * (1.0 - self.t) + (self.arrow.y * self.t)
    else:
        self.x, self.y = self.arrow.x, self.arrow.y
        self.t = 0.0
        self.arrow.reset_position()
        self.sx, self.sy = self.x, self.y
```

frame time 으로부터 t 값 계산

$$1 : distance = t : FrameTime * v$$

$$distance * t = 1 * FrameTime * v$$

$$t = \frac{FrameTime * v}{distance}$$

시간과 속도 반영

```
def __init__(self):
    # 중략
    self.t = 0.0
    self.sx, self.sy = self.x, self.y
    self.distance = math.sqrt((self.arrow.x - self.x) ** 2 + (self.arrow.y - self.y) ** 2)
```

```
def update(self):
    # 중략
    if self.t < 1.0:
        self.t += RUN_SPEED_PPS * game_framework.frame_time / self.distance
        self.x = self.sx * (1.0 - self.t) + (self.arrow.x * self.t)
        self.y = self.sy * (1.0 - self.t) + (self.arrow.y * self.t)
    else:
        self.x, self.y = self.arrow.x, self.arrow.y
        self.t = 0.0
        self.arrow.reset_position()
        self.sx, self.sy = self.x, self.y
        self.distance = math.sqrt((self.arrow.x - self.x) ** 2 + (self.arrow.y - self.y) ** 2)
```

Parametric Representation

- 직선, 또는 곡선의 (x,y) 좌표를 공통적인 파라미터를 이용하여 표현하는 방법.
- 일반적인 수학적 표현에 비해, 컴퓨터를 이용하여 그리기가 편리함.
- 동일한 곡선에 대해, 파라미터 표현법은 여러 개 있음.

m(t) = (1 - t)*p1 + t * p2, t의 범위: 0 <= t <=1

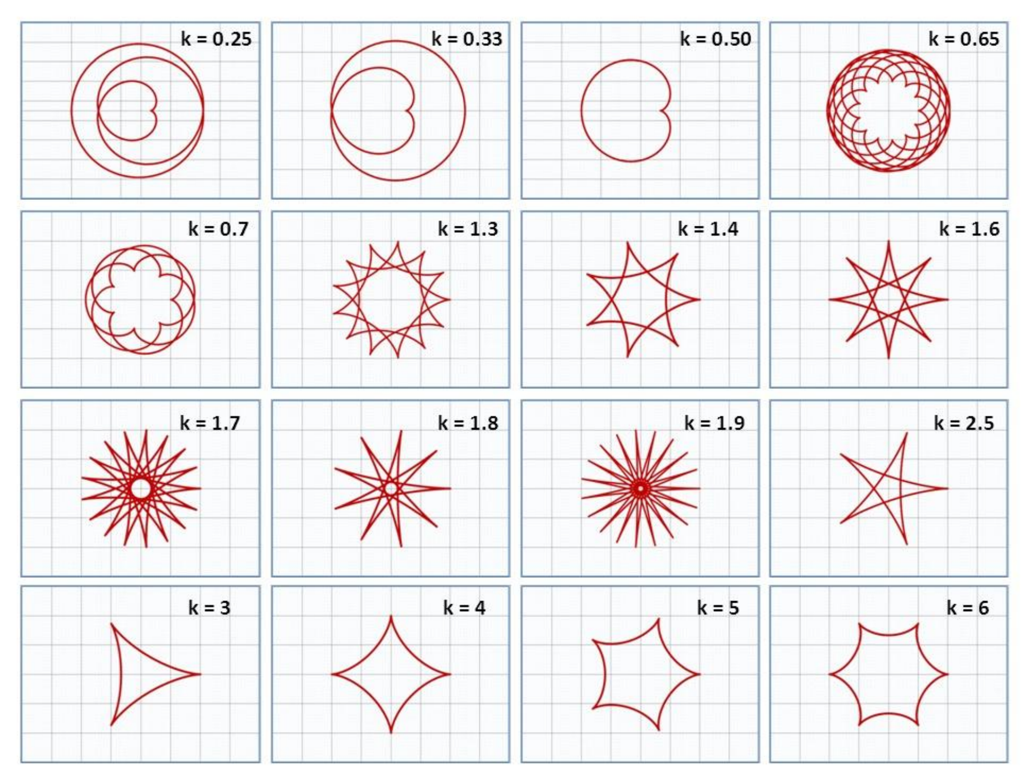
파라미터 t로 표현

	implicit form	parametric form
circle	$x^2 + y^2 - r^2 = 0$	$x(t) = r\frac{1-t^2}{1+t^2} \quad y(t) = r\frac{2t}{1+t^2}$
ellipse	$\frac{x^2}{a^2} + \frac{y^2}{b^2} - 1 = 0$	$x(t) = a\frac{1-t^2}{1+t^2} \quad y(t) = b\frac{2t}{1+t^2}$
hyperbola	$\frac{x^2}{a^2} - \frac{y^2}{b^2} - 1 = 0$	$x(t) = a\frac{1+t^2}{1-t^2} \quad y(t) = b\frac{2t}{1-t^2}$
parabola	$y^2 - 2px = 0$	$x(t) = \frac{t^2}{2p} \quad y(t) = t$

Contour type	Parametric representation
Apple shaped:	$\Gamma^{(a)} = \left\{ \frac{0.5 + 0.4\cos t + 0.1\sin 2t}{1 + 0.7\cos t}(\cos t,\ \sin t) : t \in [0, 2\pi] \right\}$
Circle:	$\Gamma^{(c)} = \{c_0(\cos t,\ \sin t) : t \in [0, 2\pi]\}, c_0$: constant
Drop shaped:	$\Gamma^{(d)} = \left\{ \left(-0.5 + 0.75\sin\frac{t}{2}, -0.75\sin t \right) : t \in [0, 2\pi] \right\}$
Ellipse:	$\Gamma^{(e)} = \{(e_0 \cos t, e_1 \sin t) : t \in [0, 2\pi]\}, e_0, e_1$: constant
Kite shaped:	$\Gamma^{(k)} = \{(\cos t + 1.3\cos^2 t - 0.8, 1.5\sin t) : t \in [0, 2\pi]\}$
Peanut shaped:	$\Gamma^{(p)} = \left\{ \sqrt{\cos^2 t + 0.25\sin^2 t}(\cos t,\ \sin t) : t \in [0, 2\pi] \right\}$
Rounded triangle:	$\Gamma^{(r)} = \{(2 + 0.3\cos 3t)(\cos t,\ \sin t) : t \in [0, 2\pi]\}$

https://www.researchgate.net/figure/Parametric-representation-of-boundary-curves_tbl2_233628915

$$x = [a - b]\cos(t) \ + b\cos\left[t\left(\frac{a}{b} - 1\right)\right]$$
$$y = [a - b]\sin(t) \ - b\sin\left[t\left(\frac{a}{b} - 1\right)\right], k = \frac{a}{b}$$



Source: https://en.wikipedia.org/wiki/Parametric_equation