2023-2 Computer Graphics PI 4차시

# Clipping & Rasterization

AM11:00에 시작됩니다.

입장 후 채팅창에 학번/이름 작성 부탁드립니다.

# 2023-2 Computer Graphics PI 4차시

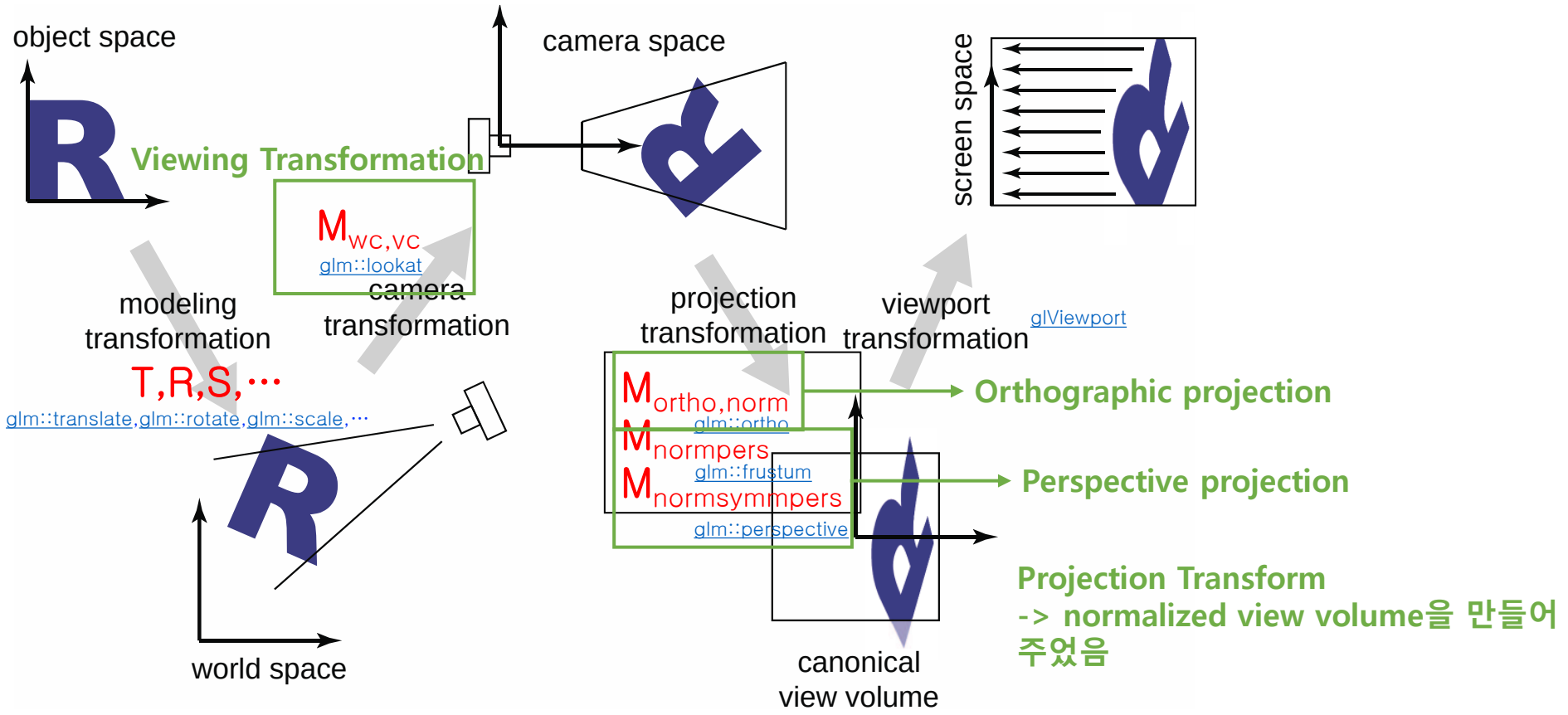
- **Clipping**
  - Sutherland-Hodgman Algorithm

- **Line Rasterization**
  - (1) DDA
  - (2) Brensenham's algorithm

- **Polygon Inside/Outside Test**
  - (1) Odd/Even rule
  - (2) Winding number

- **Polygon Rasterization**
  - (1) Scanline algorithm
  - (2) Triangle rasterization

# Graphics Pipeline




저번 시간에 배운 부분

# Clipping

# Clipping

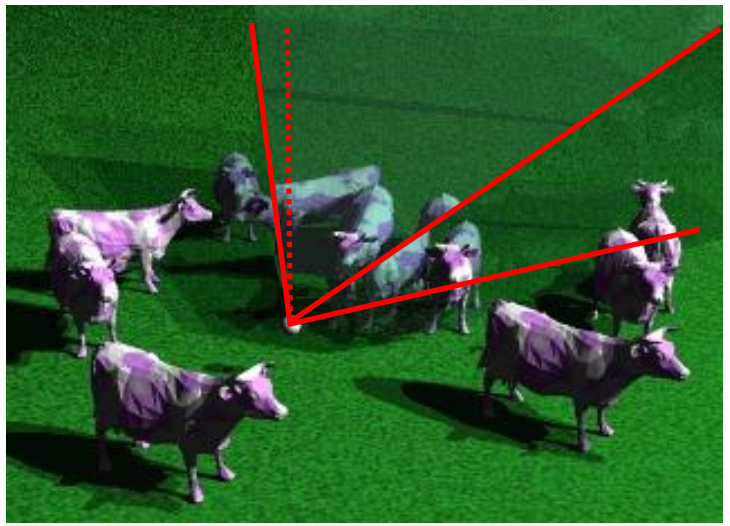
- Clipping: viewing volume 밖으로 벗어난 geometric primitive(점,선,면)을 제거

- **왜 clipping을 하는가?**
  - pixel의 color를 계산하기 위한 과정인 lighting, texturing은 매우 많은 계산이 필요
  - clipping을 한 이후 color 값이 필요한 곳들만 계산해 낭비될 수 있는 계산을 절약
  - = optimization

- **3가지 경우**
  1. trivial acceptance: viewing volume 안에 완벽하게 들어오는 경우
     - clipping이 필요 없음
  2. trivial rejection: viewing volume 밖으로 완전히 벗어난 경우
     - 더 이상 신경 쓸 필요가 없어짐 -> 남은 projection, color 계산 x
  3. crossing clip plane: 일부는 vv 안에 일부는 vv 밖에 있는 경우
     - vv 안에 들어온 부분은 keep, vv 밖 부분은 날려야 함
     - 가장 어렵고 우리가 중점적으로 신경 쓸 부분

# Clipping

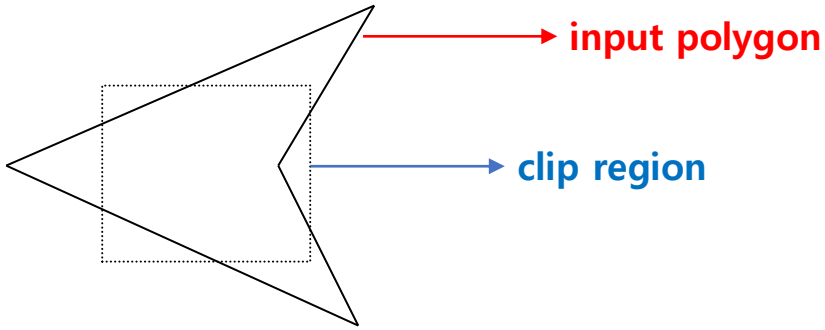
- **primitives의 관점**
  1. 점(points)
     - 3번째 경우 불가능
     - trivial accept/reject 경우만 존재
  2. 선(lines)
     - clip plane과 만나는 부분에서 잘라내 vv 안쪽 부분만 keep
  3. 면(polygon): vertex로 정의
     - clipping을 하더라도 input의 vertex 순서를 유지해야함
     - = connectivity 유지



courtesy of L. McMillan

# Sutherland-Hodgman Clip Algorithm

- 2차원의 경우
- input polygon: clip할 polygon
- clip region: clip되는 영역
  - 3차원에서는 clip volume: normalized view volume




- input polygon이 clip region 밖에 있는 부분을 잘라내고 싶음
- 그 과정에서 vertex ordering은 유지 되어야 함

# Sutherland-Hodgman Clip Algorithm

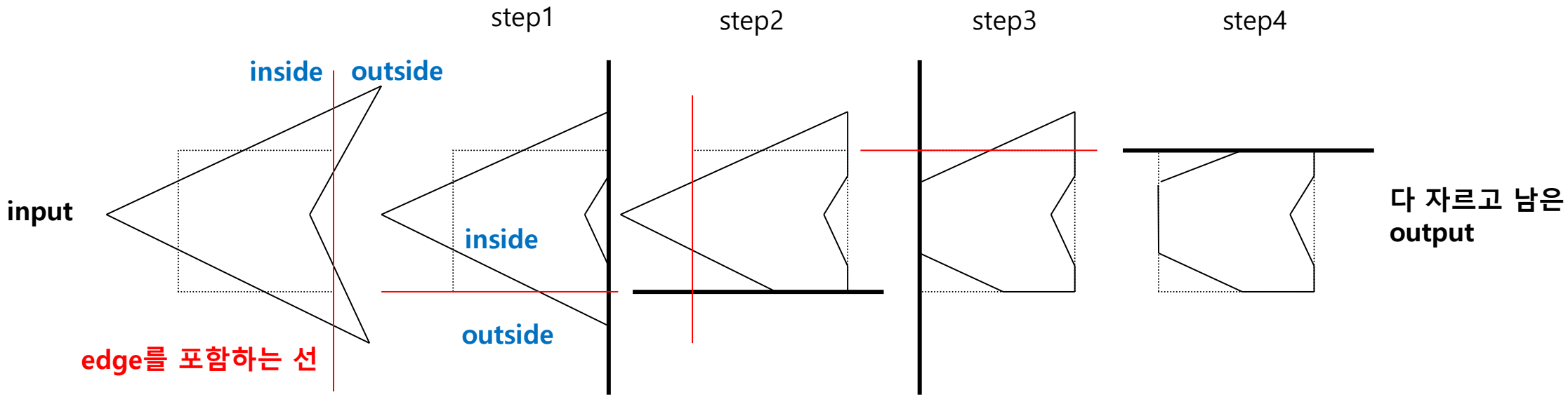
과정

**STEP1.**

1. clip region의 하나의 edge(윤곽선) 선택
2. edge를 포함하는 line을 생각하자
   - clip region을 포함하는 쪽이 inside, 포함하지 않는 쪽이 outside가 됨
   - inside 쪽은 keep, outside 쪽은 잘라버림

**STEP2~4.**

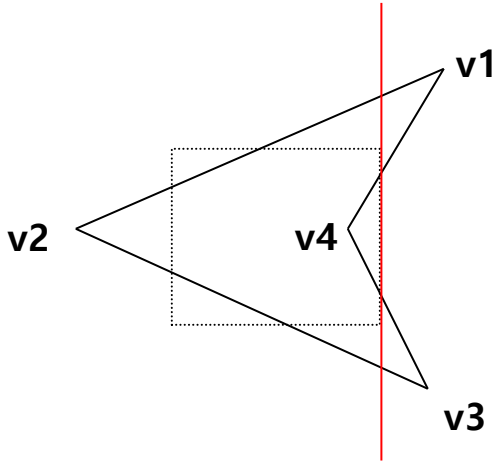
- 다음 clip edge 선택 후 step1 반복 (돌려서 다 자를 때까지)



# Sutherland-Hodgman Clip Algorithm

구현

- STEP 1~4는 모두 유사한 step 1개의 반복
- STEP1에서 vertex 1~4를 순회하면서 어디가 inside/outside인지에 따라 자를 것인지 여부 결정
- 순회할 때 4가지 케이스에 따라 적절한 행동을 취함

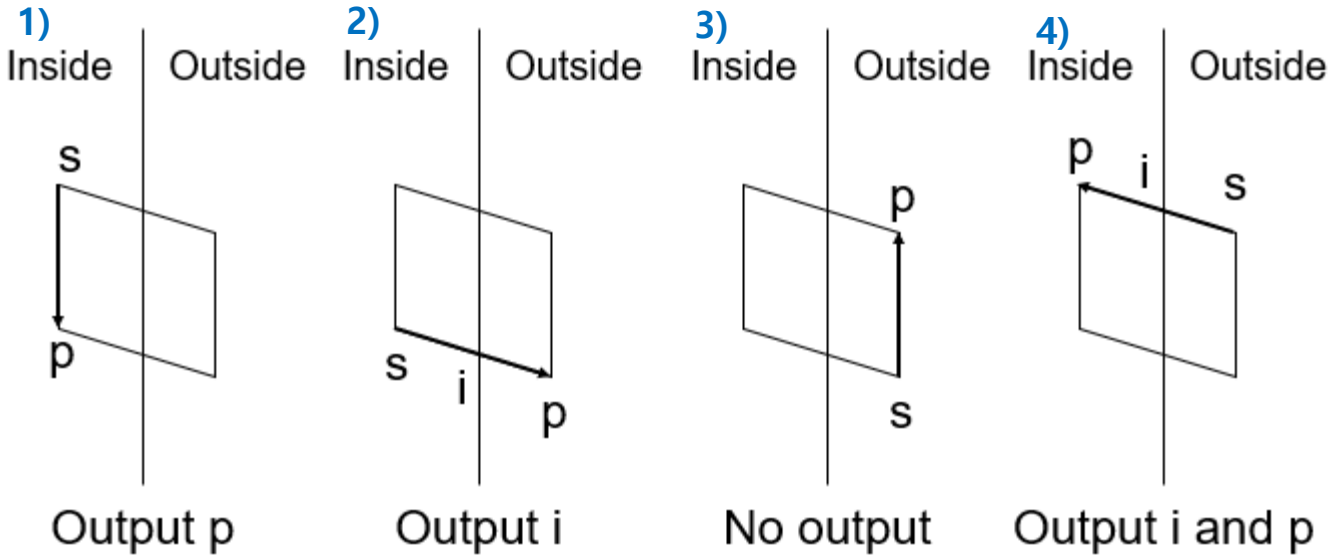



1) 시작점: inside -> 도착점: inside
   - 도착점 p만 출력

2) inside -> outside
   - clip line과 polygon edge이 만나는점(intersection) i를 구해서 출력

3) outside -> outside
   - 잘려나가므로 아무것도 출력하지 않음

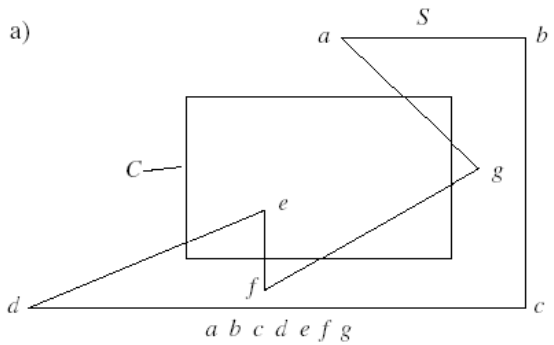
4) outside -> inside
   - case 2의 반대 버전
   - intersection i를 먼저 출력하고, 도착점 p도 출력



# Sutherland-Hodgman Clip Algorithm

예시

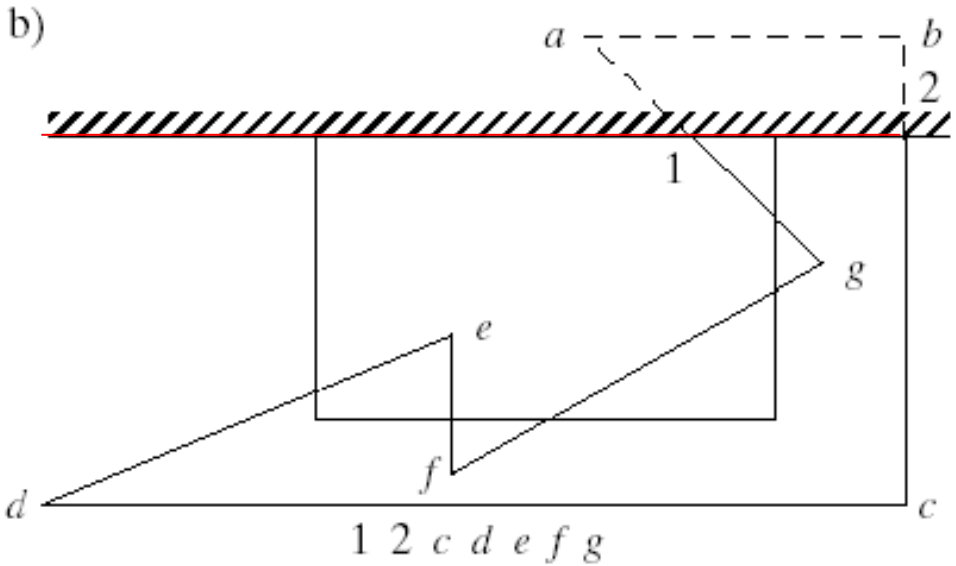
- input: vertex가 7개인 polygon
- clip region: 직사각형




**STEP1.**

1. clip region의 edge 선택
2. input polygon을 돌며 4가지 case 중 무엇이냐에 따라서 적절하게 출력

- polygon의 점 g에서 시계방향으로 순회
- a->g: inside->outside - intersection i 출력 => 1
- a->b: outside->outside => skip
- b->c: outside->inside => intersection 2+도착 vertex c
- c->d->e->f->g: inside->inside: 각 구간의 도착 vertex 출력
  => d,e,f,g
- 여기까지가 첫번째 iteration => 1 2 c d e f g : vertex 순서

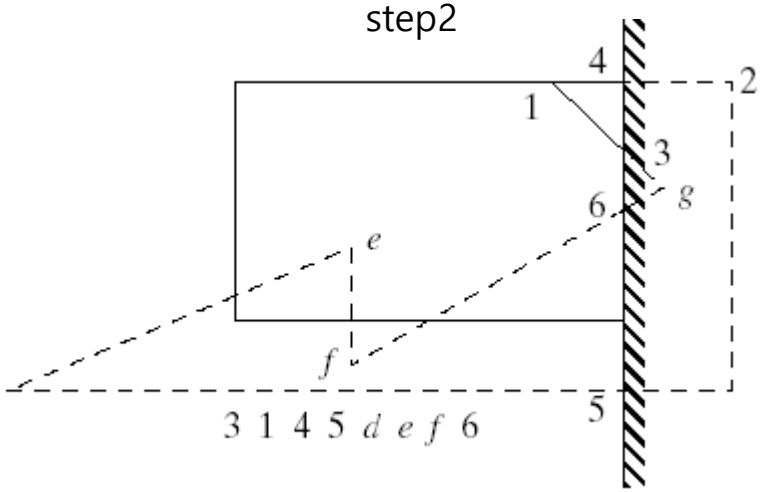


# Sutherland-Hodgman Clip Algorithm

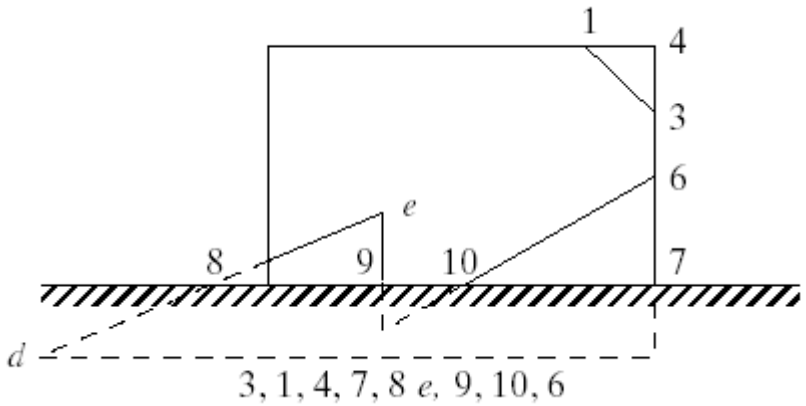
STEP 2~4

- 다른 clip region edge를 선택하고 STEP1과 비슷한 방식으로 진행

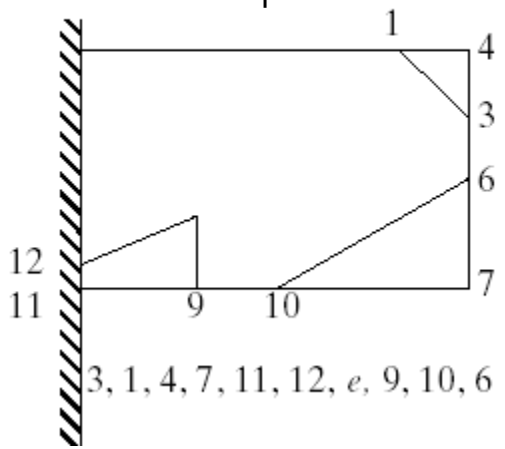
step2




step3




step4




- 최종결과물: 3,1,4,7,12,e,9,10,6
  = 삼각형 2개 + 사각형 1개
- (3,6), (9,10) 선은 실제 OpenGL 등에서는 날려버리나 수업시간에는 포함한 것을 결과물로 생각

# Sutherland-Hodgman Clip Algorithm

3차원의 경우

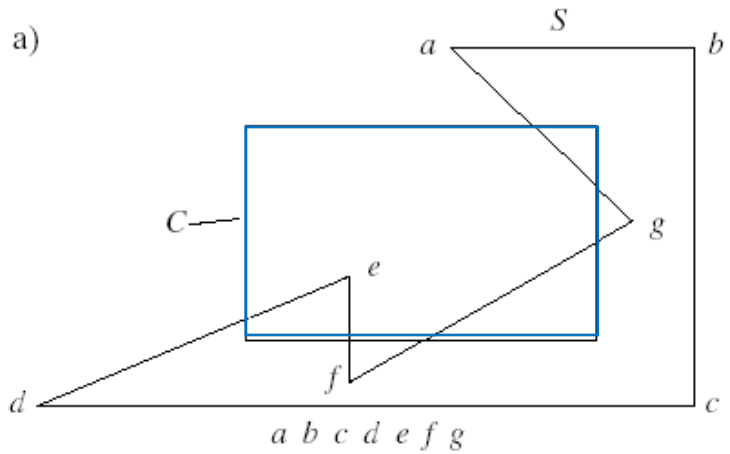
- viewing pipeline에서는 viewing volume이 사각형이 아니라 정육면체 형태
  - line 대신 clip plane을 품는 평면을 가지고 자름
  - clip volume 포함하는 쪽의 평면이 inside, 반대쪽이 outside

알고리즘

- 바깥쪽 loop은 clip region의 edge를 선택해서 도는 것
- 안쪽 loop은 polygon 주변을 도는 것
- loop을 돌며 4가지 case (inside, outside) 에 따라 출력

**For each edge/plane ($c_i$, $c_j$) in clip region/volume**
**For each successive edge (s, p) in input polygon**
**1.** If both inside ($c_i$, $c_j$): Output new **p**
**2.** If both outside: Output nothing
**3.** If **s** inside, **p** outside: Output intersection of (**s**, **p**) with ($c_i$, $c_j$)
**4.** If **s** outside, **p** inside: Output intersection of (**s**, **p**) with ($c_i$, $c_j$), then **p**

- 알고리즘 실제로 구현하려면 2가지 해결 필요



# Sutherland-Hodgman Clip Algorithm

**해결해야 하는 2가지**

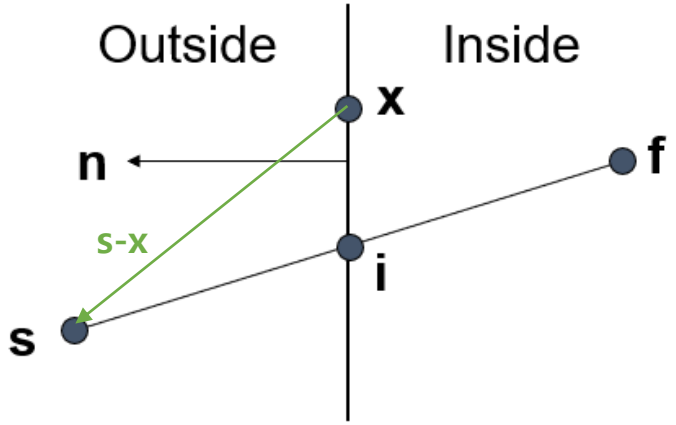
1) 시작, 도착 vertex가 어떤 case에 해당하는지 어떻게 알 것인가?
2) inside->outside, outside->inside 경우 i를 어떻게 구할 것인가?

**1) Inside-outside testing**

- 시작, 도착 vertex 어떤 case에 해당하는지 알 수 있는 방법
- -> vertex가 clip plane의 안쪽(inside)에 있는지 바깥쪽(outside)에 있는지 알고 싶음
- 가정) outward pointing normal = plane의 수직 방향 벡터 n은 clip region의 바깥쪽을 가르킴

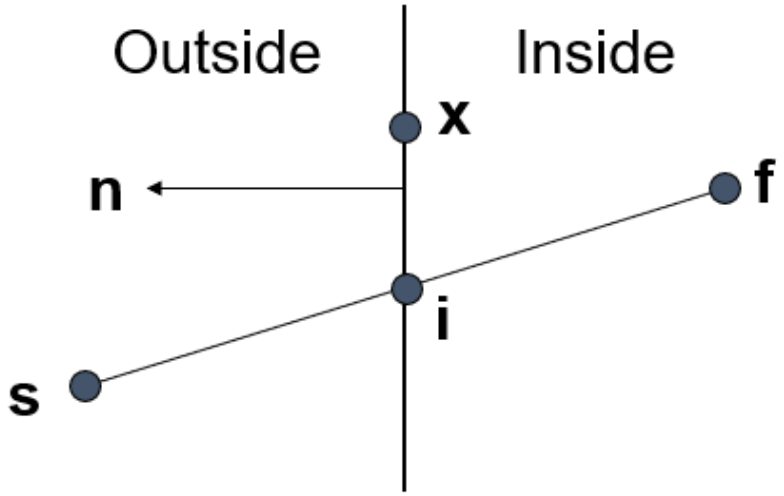
방법

1. plane 상의 임의의 점 x를 잡음
2. inside/outside 여부를 확인하고 싶은 점 s로 s-x 벡터를 구함



# Sutherland-Hodgman Clip Algorithm

3. $n \bullet (s - x)$ 계산
   - $n \bullet (s - x) > 0$ : s가 outside에 있음
     - s가 outside에 있다면, s와 n이 같은 방향에 놓여있고, n벡터와 (s-x)의 끼인각<90이기 때문
   - $n \bullet (s - x) < 0$ : s가 inside에 있음
     - s가 inside에 있다면, s와 n이 반대 방향에 놓여있고, n벡터와 (s-x)의 끼인각>90이기 때문
   - $n \bullet (s - x) = 0$ : s가 clip plane 위에 있음
     - s가 clip plane 위에 있다면 n벡터와 (s-x)의 끼인각=90이기 때문




$n \bullet (s - x) > 0$ -> s: outside
$n \bullet (i - x) = 0$ -> i: clip plane 위
$n \bullet (f - x) < 0$ -> f: inside

# Sutherland-Hodgman Clip Algorithm

**2) Find Intersection points**

- inside->outside, outside->inside 경우 intersection point i를 구하는 방법
- 선분(edge)와 평면이 만나는 좌표를 찾아야함

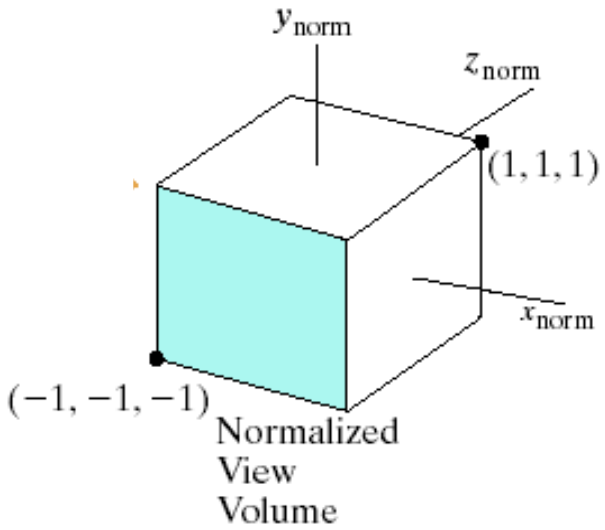
방법

1. 매개변수 t를 사용해 x1와 x2를 연결하는 선분을 수식으로 표현

- $\mathbf{x}(t) = \mathbf{x}_1 + (\mathbf{x}_2 - \mathbf{x}_1)t,\ 0 \le t \le 1$

2. 평면의 방정식 정의

- 일반적인 평면의 방정식의 형태: ax+by+cz+d=0
- view volume은 정육면체 형태이므로 clip plane은 x,y,z축에 수직
- $x_s=\pm1$, $y_s=\pm1$, $z_s=\pm1$



# Sutherland-Hodgman Clip Algorithm

3. 직선과 평면의 intersection을 구해야 함

- 평면의 방정식: x=a, 점 $\mathbf{x}_1$을 $(\mathbf{x}_1, \boldsymbol{y}_1, \boldsymbol{z}_1)$, 점 $\boldsymbol{x}_2$을 $(\mathbf{x}_2, \boldsymbol{y}_2, \boldsymbol{z}_2)$라고 해보자

$$\begin{matrix} x_1 & x_2 - x_1 & \\ y_1 + y_2 - y_1 & \mathrm{t} = & y(t) \\ z_1 & z_2 - z_1 & \end{matrix} \quad \begin{matrix} \\ \\ z(t) \end{matrix}$$

$x_1 + (x_2 - x_1)\mathrm{t} = a \qquad \mathrm{t} = \frac{a - x_1}{x_2 - x_1}$

t를 식에 다시 대입하면 intersection 점 $x_i$를 구할 수 있음

$$\mathbf{x}_i = \left(a, y_1 + \frac{(y_2 - y_1)}{(x_2 - x_1)}(a - x_1), z_1 + \frac{(z_2 - z_1)}{(x_2 - x_1)}(a - x_1)\right)$$

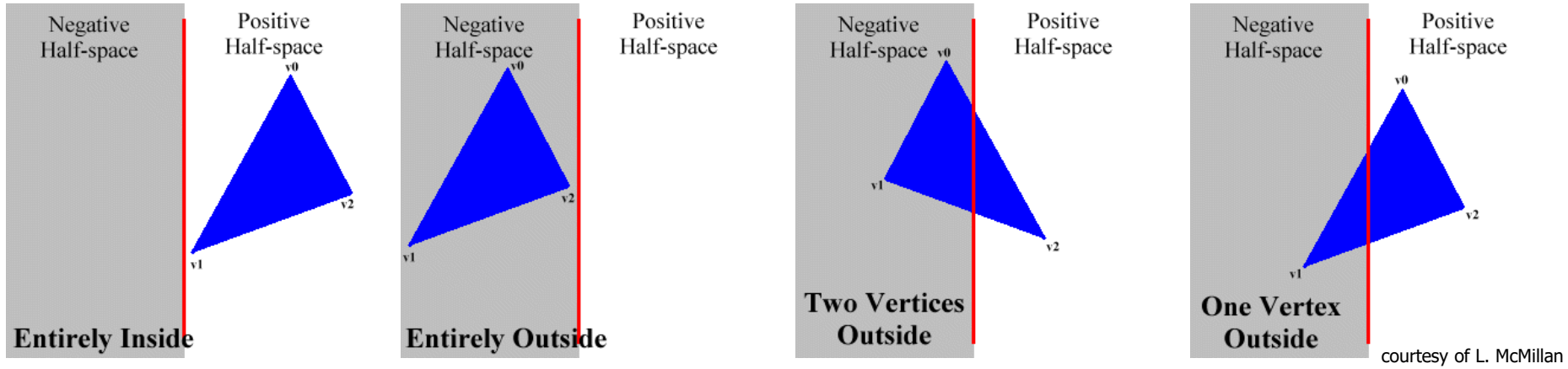
# Sutherland-Hodgman: Triangle

- input polygon이 삼각형인 경우
  - 앞에서는 일반적인 polygon에 대한 알고리즘을 배움
  - input polygon이 삼각형이면 앞의 알고리즘 압축시켜 짧게 구현 가능

방법

1. inside-outside testing으로 inside 점 개수 count

- 삼각형은 vertex가 3개이므로 outside/inside인 점의 개수가 몇 개인지 세보면 4가지 경우 존재



# Sutherland-Hodgman: Triangle

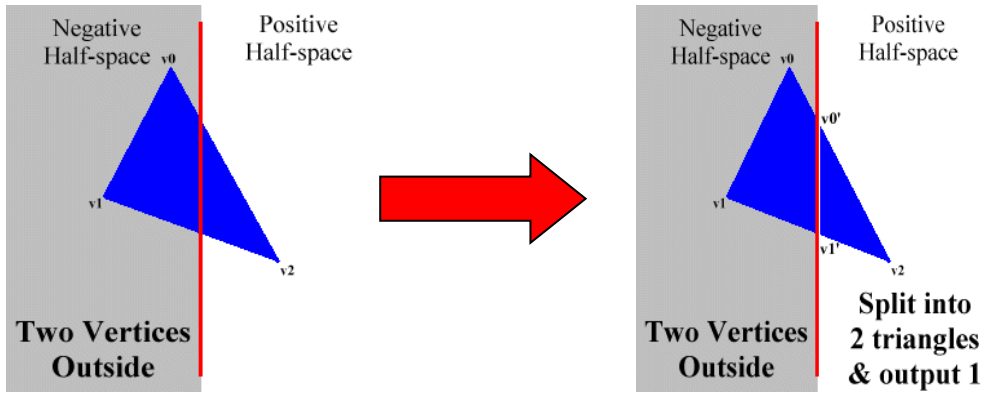
2.

1) inside vertex 0개: trivially reject이므로 다 날림
2) inside vertex 3개: trivially accepted 상태이므로 다 keep

3) inside vertex 1개

바깥쪽 점들과 연결해서 만나는 점 v0',v1'을 구하고

$\Delta v'_0 v'_1 v_2$을 새로운 output으로 출력




$\Delta v'_0 v'_1 v_2$

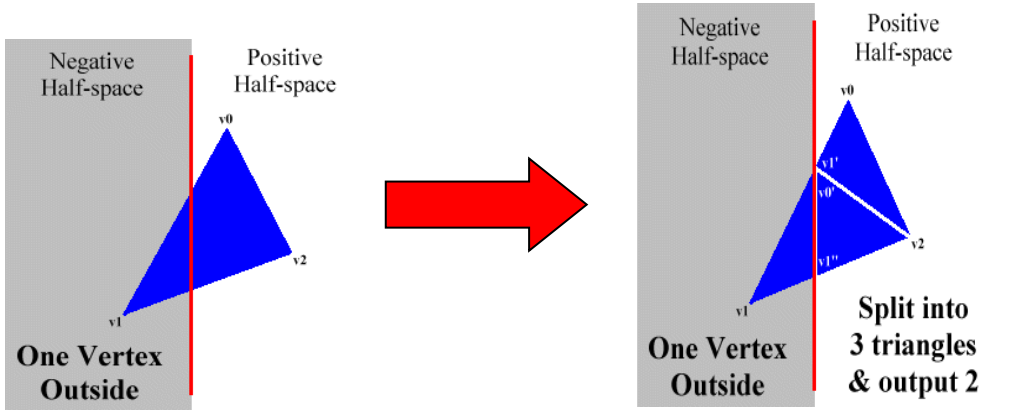
4) inside vertex 2개

바깥쪽 점들과 연결해서 만나는 점 v1',v1"을 구하면

사각형 v0v1'v1"v2 가 나옴

-> output도 삼각형으로 만들어주기 위해 사각형을 쪼개줌

-> $\Delta v_0 v'_1 v_2$, $\Delta v'_1 v"_1 v_2$ 가 만들어짐



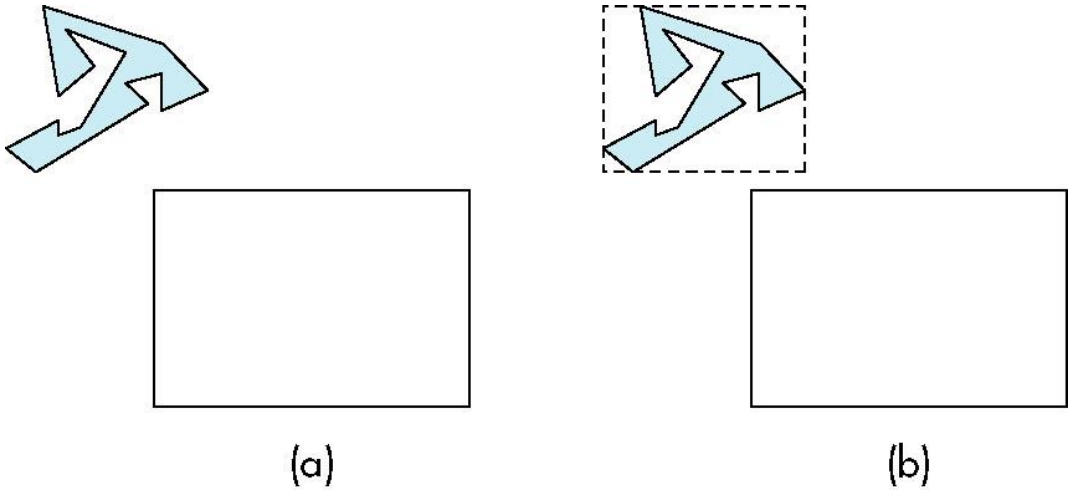

$\Delta v_0 v'_1 v_2$

$\Delta v'_1 v"_1 v_2$

# Clipping Accerleration

clipping Accerleration

- bounding area/volume에 먼저 clipping을 적용해 test 하는 것
- bounding area: 복잡한 input polygon을 감싸는 간단한 shape(box, sphere)
- 사용하는 이유
  - trivial reject case인 경우 -> 복잡한 polygon에 대해 test 하려면 시간을 많이 낭비하게 됨
  - bounding area에 먼저 clipping을 적용해 trivial reject한 경우면 날림
  - 간단한 shape이므로 test가 훨씬 간단하고 시간 절약 가능



# Quiz- clipping

Consider a specialized version of Sutherland-Hodgman clipping for a triangle T with three vertices $v_1$, $v_2$, $v_3$ against a clipping plane $P_1$.

- $P_1$: z=0 (+z is outside of the clip region)
- T: $v_1$(0, 0, 1), $v_2$(0, 0, -1), $v_3$(-1, 0, -1)

문제 1

정답

총 1.00 점에서 1.00 점 할당

문제 표시

Find the normal vector of the plane $P_1$ that points at the positive z-axis.

하나를 선택하세요.

- a. (1, 0, 0)
- b. (0, 1, 0)
- c. (0, 0, -1)
- d. (0, 0, 1) ✔

0x+0y+1z=0. The normal vector consists of the coefficients.

정답 : (0, 0, 1)

# Quiz- clipping

문제 2
정답
총 1.00 점에서 1.00 점 할당
문제 표시

Find respectively whether each vertex $v_1$, $v_2$, $v_3$ is inside or outside of $P_1$.

하나를 선택하세요.

- a. inside, outside, inside
- b. outside, outside, outside
- c. outside, inside, inside ✔
- d. outside, outside, inside

Pick any point on z=0, say x=(0, 0, 0)

$$(v_1 - x) \cdot (0,0,1) = 1 > 0 \rightarrow \text{outside}$$
$$(v_2 - x) \cdot (0,0,1) = -1 < 0 \rightarrow \text{inside}$$
$$(v_3 - x) \cdot (0,0,1) = -1 < 0 \rightarrow \text{inside}$$

정답 : outside, inside, inside

# Quiz- clipping

문제 4
정답
총 1.00 점에서 1.00 점 할당
문제 표시

Find the clipped triangle(s) of T as a result of the Sutherland Hodgman algorithm with respect to the clip plane $P_1$.

하나를 선택하세요.

a. $\triangle(v_1, v_2, v_3)$

b. $\triangle(v_3', v_3, v_2), \triangle(v_1', v_3', v_2)$

✔

c. $\triangle(v_3', v_3, v_2), \triangle(v_1, v_3', v_2)$

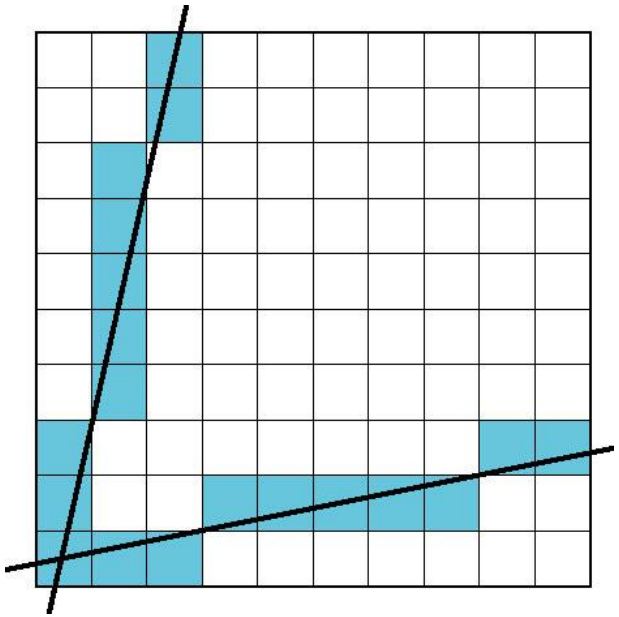
d. Empty

정답 : $\triangle(v_3', v_3, v_2), \triangle(v_1', v_3', v_2)$

# Line Rasterization

# Rasterization

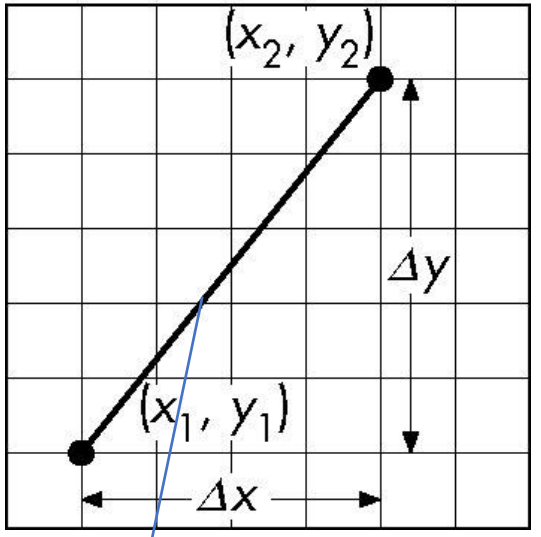
- rasterization : 2D 를 pixel 로 바꿔주는 것
- line segment rasterization, polygon rasterization 을 살펴보자



# Line Rasterization

- (가정1) **시작점 끝점 좌표가 integer라고 가정**
  integer가 아니면 반올림or내림 등의 방법으로 정수로 만들면 된다.

- (가정2) **line과 가장 근접한 좌표를 찾는 과정까지만 배우고,**
  **해당 좌표를 어떻게 display 할 지는 배우지 않는다.**
  (x,y)가 주어지면 해당 pixel 을 켜는 setPixel(x,y) 라는 함수가 있다고만 배운다.
  그 함수의 내부는 배우지 않는다. 그 구현 방법은 매우 다양하기 때문.




$$y = mx + h$$

$$m = \Delta y/\Delta x = (y_2 - y_1)/(x_2 - x_1)$$

# Line Rasterization – (1) DDA 이해

- x를 시작점부터 하나씩 증가시키면서, y를 증가시켜야 할지 아닌지를 결정

```
m = (y2-y1)/(x2-x1);
```
기울기 구하고

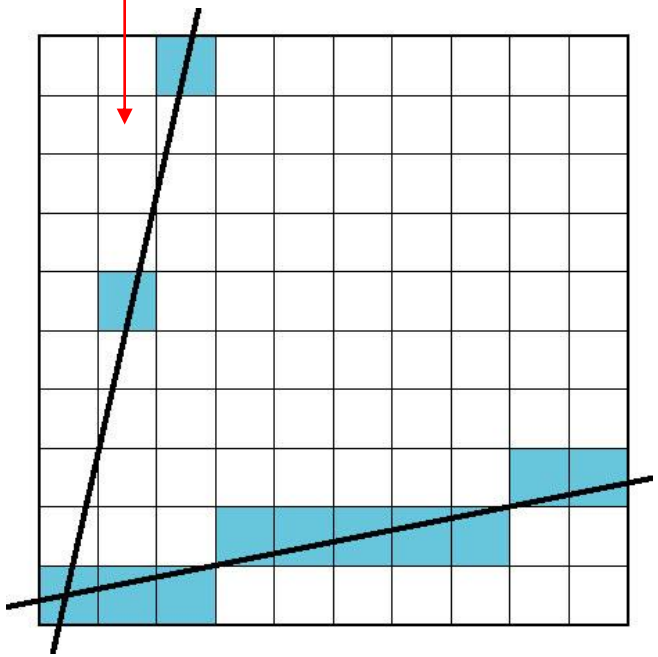
시작점 x 가 x2가 될 때까지 1씩 증가

```
for(x=x1, y=y1; x<=x2; x++)
{
   setPixel(x, round(y));
   y+=m;
}
```

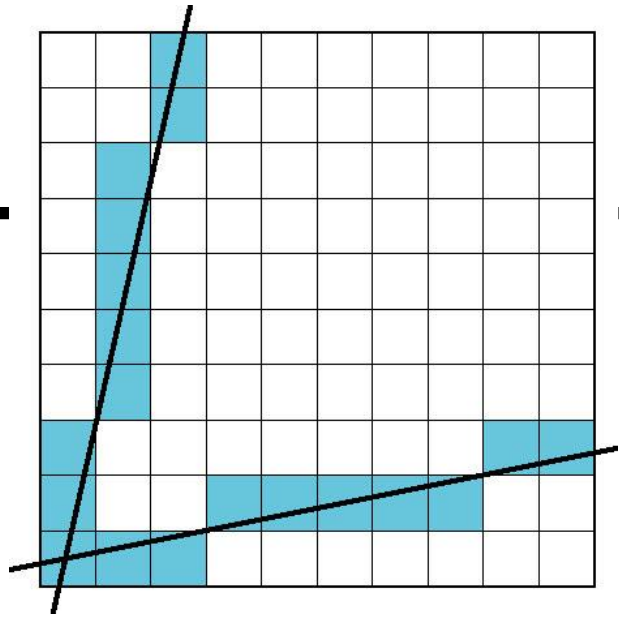
setPixel() 에 들어가는 x, y 는 픽셀 값이므로 정수여야 한다.
y 는 기울기를 더한 값이라 정수가 아닐 수 있다.
따라서 round() 해줘야 한다.

Δx 가 1 이므로
m = Δy/Δx = Δy
즉, y 는 기울기만큼 증가한다

Problems for steep lines
: 기울기가 1보다 작으면 잘 그려지는데
기울기가 1보다 크면 문제 발생
: line 처럼 보이지 않는다.



# Line Rasterization – (1) DDA 결론



- **( 1 ≥ |m| ≥ 0 인 경우 )** 앞의 알고리즘 그대로 이용

```
m = (y2-y1)/(x2-x1);

for(x=x1, y=y1; x<=x2; x++) {
  setPixel(x, round(y));
  y+=m;
}
```

- **( |m| > 1 인 경우 )** x와 y의 역할을 swap

(코딩 방법1) y 가 1 증가하면, x 가 1/m 증가

```
m = (y2-y1)/(x2-x1);
for(x=x1, y=y1; y<=y2; y++) {
  setPixel(round(x), y);
  x+=1/m;
}
```

Δy 가 1 이므로
m = Δy/Δx = 1/ Δx
Δx = 1/m
즉, x 는 1/m 만큼 증가한다

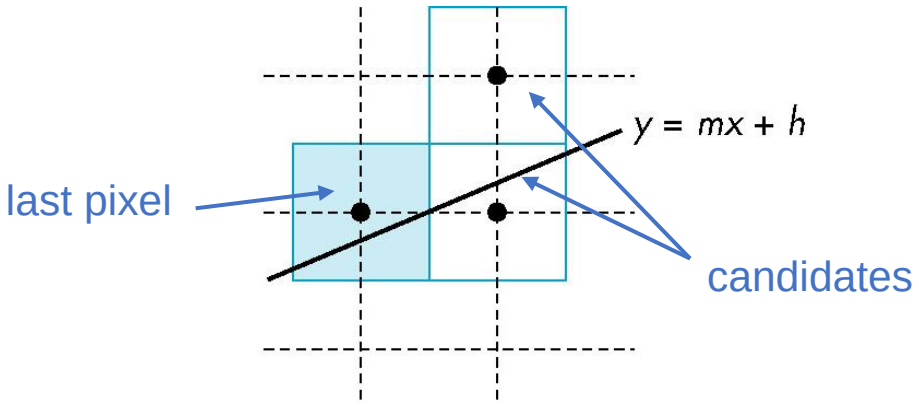
(코딩방법2) 코드로 x, y swap

```
swap x with y;
m = (y2-y1)/(x2-x1);
for(x=x1, y=y1; x<=x2; x++) {
  setPixel(x, round(y));
  y+=m;
}
swap x with y;
```

# Line Rasterization – (2) Bresenham's Algorithm 이해

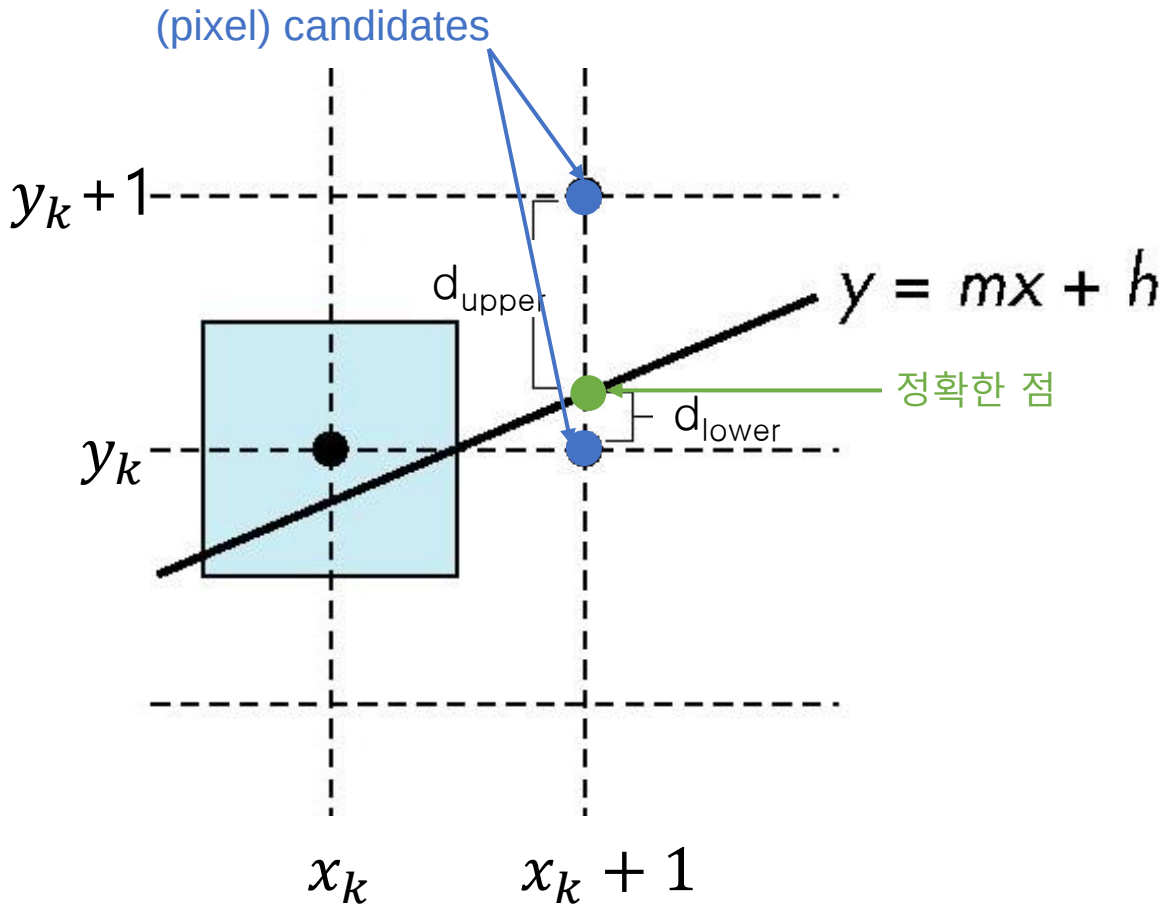
- DDA 코드를 최적화한 방법
- DDA 는 input 도 output 도 integer 를 원하는데, 중간과정에서 floating point 연산 존재
  floating point 연산을 없애서 최적화하자

- **( 1 ≥ |m| ≥ 0 인 경우 )** 만 살펴보고 **( |m| > 1 인 경우 )** 는 앞과 동일하게 symmetry 이용해서 처리하면 됨
- 기울기가 1보다 작기 때문에 현재 픽셀을 찾았으면 그 다음 픽셀의 후보는 오직 2개이다.



# Line Rasterization – (2) Bresenham's Algorithm 이해

- 2개의 후보 중에 어떤 것을 선택할 지 알려주는 것이 Decision Variable $p_k$




$d_{lower}$, $d_{upper}$는 정수가 아니다. (1보다 작음)
$\Delta x$ 를 곱해주면 $p_k$가 정수가 된다.

$$p_k = \Delta x\,(d_{lower} - d_{upper})$$

$p_k$ is an integer
$p_k < 0$ use lower pixel
$p_k > 0$ use upper pixel

POINT1. $p_k$가 정수
POINT2. $p_{k+1}$ 은 $p_k$로부터 쉽게 구할 수 있다

# Line Rasterization – (2) Bresenham's Algorithm 결론

- $p_0$ 의 유도과정과 증명은 생략 (교수님 수업 및 강의자료 참고)

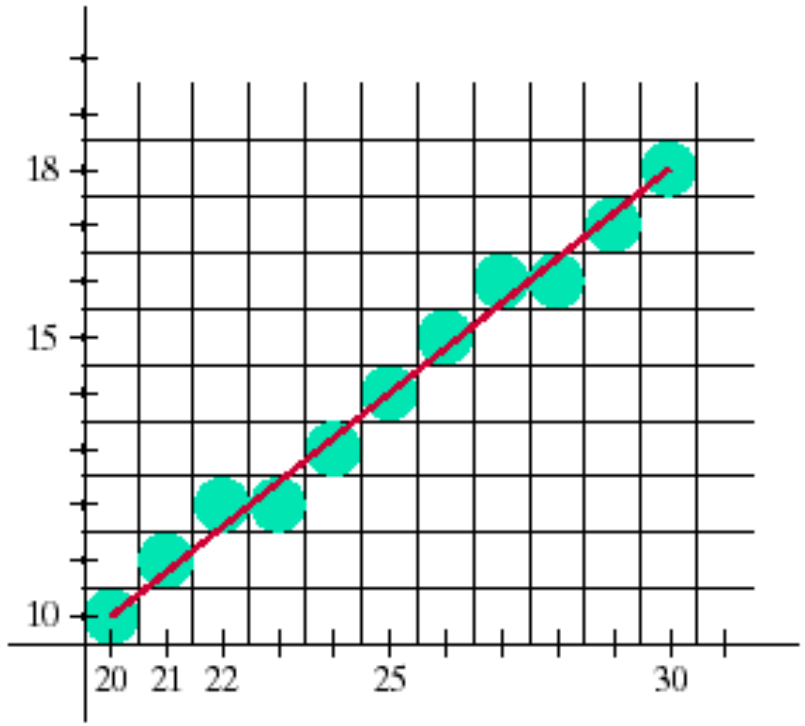
- **STEP1. 기울기 확인**
  - ( $1 \geq |m| \geq 0$ 인 경우 ) 그대로
  - ( $|m| > 1$ 인 경우 ) 앞뒤에서 x, y swap 필요

- **STEP2. 초기값 계산**
  - setPixel $(x_0, y_0)$
  - $P_0$ 계산. $P_0 = 2\Delta y - \Delta x$

- **STEP3. Decision Variable $p_k$ 계산하며 그 다음 픽셀 결정. STEP3 iteration.**
  - $P_k < 0$ 이면,
    - $(x_k+1, y_k)$ 선택 (즉 y 그대로 유지)
    - $P_{k+1} = P_k + 2\Delta y$
  - $P_k \geq 0$ 이면,
    - $(x_k+1, y_k+1)$ 선택 (즉 y 1 증가)
    - $P_{k+1} = P_k + 2\Delta y - 2\Delta x$

# Line Rasterization – (2) Bresenham's Algorithm 예제

$$p_0 = 2\Delta y - \Delta x \qquad \Delta x = 10, \qquad \Delta y = 8$$
$$= 6$$




| $k$ | $p_k$ | $(x_{k+1}, y_{k+1})$ |
|---|---|---|
| 0 | 6 | (21, 11) |
| 1 | 2 | (22, 12) |
| 2 | −2 | (23, 12) |
| 3 | 14 | (24, 13) |
| 4 | 10 | (25, 14) |

| $k$ | $p_k$ | $(x_{k+1}, y_{k+1})$ |
|---|---|---|
| 5 | 6 | (26, 15) |
| 6 | 2 | (27, 16) |
| 7 | −2 | (28, 16) |
| 8 | 14 | (29, 17) |
| 9 | 10 | (30, 18) |

# QUIZ – Line Rasterization

정보

문제 표시

Given a line segment $\overline{P_1P_2}$ connecting two points $P_1(5,5)$ and $P_2(10,8)$, rasterize it using Brensenham's algorithm. The first rasterized point $(x_0, y_0)$ is the same as $P_1$.

문제 1

정답

총 1.00 점에서 1.00 점 할당

문제 표시

Find $(x_1, y_1)$.

하나를 선택하세요.

- a. (6, 5)
- b. (6, 6) ✔
- c. (6, 7)
- d. (6, 8)

답이 맞습니다.

$P_0=1 > 0$

정답 : (6, 6)

문제 2

정답

총 1.00 점에서 1.00 점 할당

문제 표시

Find $(x_2, y_2)$.

하나를 선택하세요.

- a. (7, 8)
- b. (6, 7)
- c. (7, 6) ✔
- d. (7, 7)

답이 맞습니다.

$P_1=-3 < 0$

정답 : (7, 6)

문제 3

정답

총 1.00 점에서 1.00 점 할당

문제 표시

Find $(x_3, y_3)$.

하나를 선택하세요.

- a. (8, 8)
- b. (7, 7)
- c. (8, 7) ✔
- d. (8, 9)

답이 맞습니다.

$P_2=3 > 0$

정답 : (8, 7)

문제 4

정답

총 1.00 점에서 1.00 점 할당

문제 표시

Find $(x_4, y_4)$.

하나를 선택하세요.

- a. (9, 8)
- b. (9, 7) ✔
- c. (9, 9)
- d. (8, 8)

답이 맞습니다.

$P_3=-1 < 0$

정답 : (9, 7)

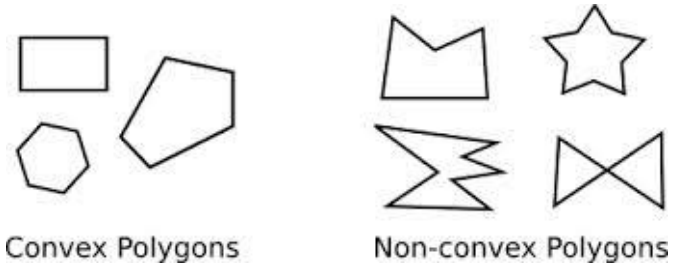
# Polygon Inside/Outside Test

# Polygon 정의

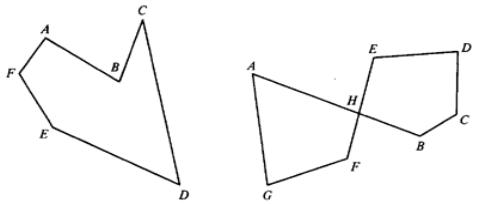
- **(Polygon 정의)** 3개 이상의 꼭짓점이 edge로 연결된 모양

- **(Polygon 분류1) convex (볼록) VS non-convex (오목)**

: 다각형 안의 임의의 점 2개를 찾아서 선분으로 연결했을 때
선분 전체가 영역 안에 들어가면 convex polygon




- **(Polygon 분류2) simple (edge crossing X) VS non-simple (edge crossing O)**




Simple Non-simple

⇒ Non-convex, non-simple 일 때도 inside 를 찾아서 칠해야 한다.
따라서 polygon rasterization 이 어렵다.

- **Cf) Degenerate Polygon** → 우리 수업에서는 degenerate 한 경우가 없다고 가정한다. 실세계에서는 이런 경우를 다 고려해야 하므로 복잡하다.

Case1) Collinear : edge가 동일 선상에 있어서 vertex 가 필요 없는 경우

Case2) Duplicated Vertices : vertex 가 중복돼서 edge 길이가 0 인 경우

# Polygon Inside/Outside Test – (1) odd/even rule

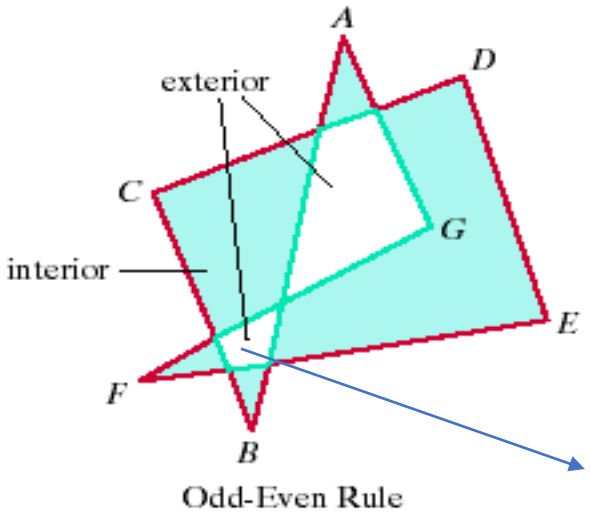
- **(1) odd/even rule**

  임의의 한 점에서 아무 방향으로나 ray 를 쏜다.

  그게 edge와 몇 번 만나는 지 판단.

  홀수 내부, 짝수 외부

Ray란? 시작은 있는데 끝은 없는 line.
cf) line segment : 양끝점이 있는 line.



Odd-Even Rule

두 방법의 결과가 다르다.
경우에 따라 다른 방법을 선택해야 한다.

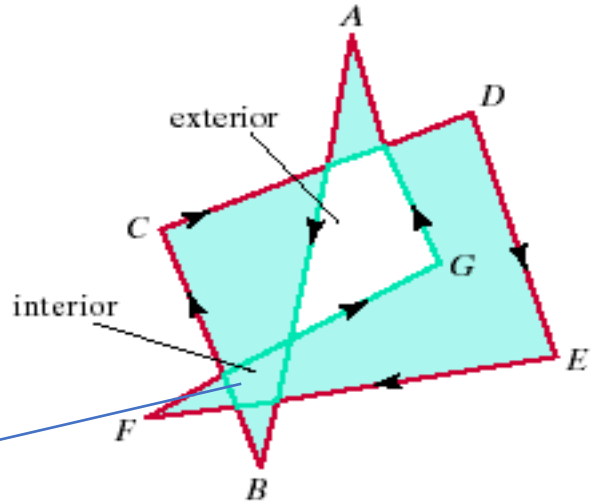
# Polygon Inside/Outside Test – (2) winding number

- **(2) winding number**

  winding number 의 초기값은 0.

  임의의 한 점에서 아무 방향으로나 ray 를 쏜다.
  counter-clockwise (반시계 방향) 으로 cross 되면 winding number +1
  clockwise (시계 방향) 으로 cross 되면 winding number -1

  winding number 0 이면 외부, 0 아니면 내부.



Nonzero Winding-Number Rule

두 방법의 결과가 다르다.
경우에 따라 다른 방법을 선택해야 한다.

# Polygon Inside/Outside Test – (2) winding number

**winding rule : counter-clockwise cross VS clockwise cross**

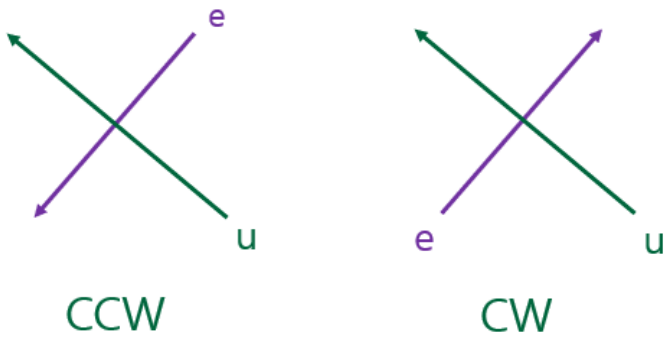
Ray 의 방향벡터 u
Edge의 방향벡터 e
⇒ 애초에 수학적으로 edge 를 찾은 것. 따라서 이미 알고 있음.

- **(판단방법1)** (사실 이게 정의)

  ray 기준에서 left half-space, right half-space 존재

  right -> left : CCW

  left -> right : CW




- **(판단방법2)** (사실 이게 (수학적) 판단방법)

  그림이 주어지지 않고 edge vector, ray vector 주어졌을 때 CCW, CW 찾는 방법

  cross product (외적) 이용

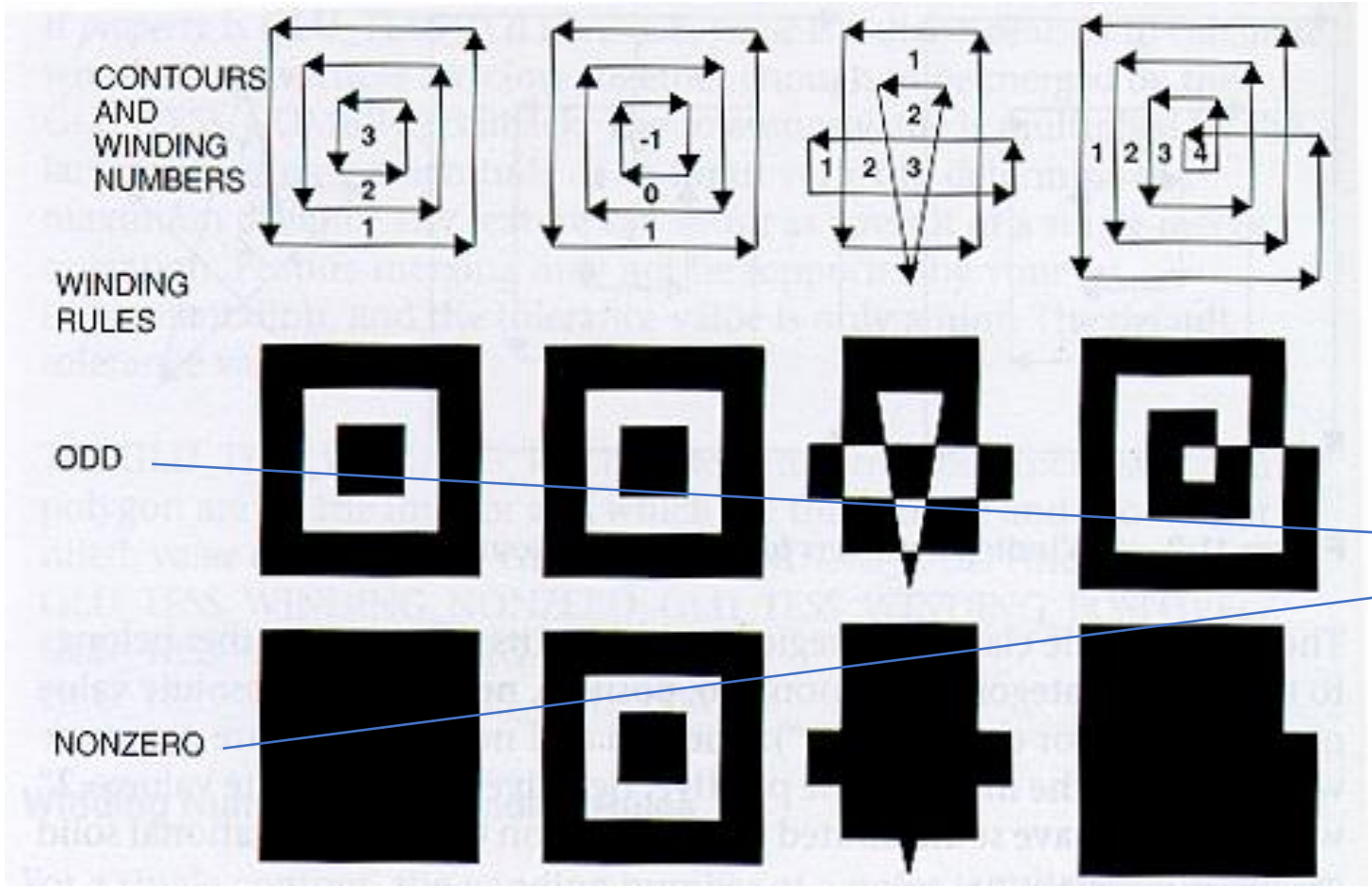
  Edge direction: $e = (e_x, e_y, 0)$

  Ray direction: $u = (u_x, u_y, 0)$

  (z component of $u \times e$) = $u_x e_y - u_y e_x > 0$ : CCW

  otherwise : CW

# Polygon Inside/Outside Test – (1) (2) 방법 비교

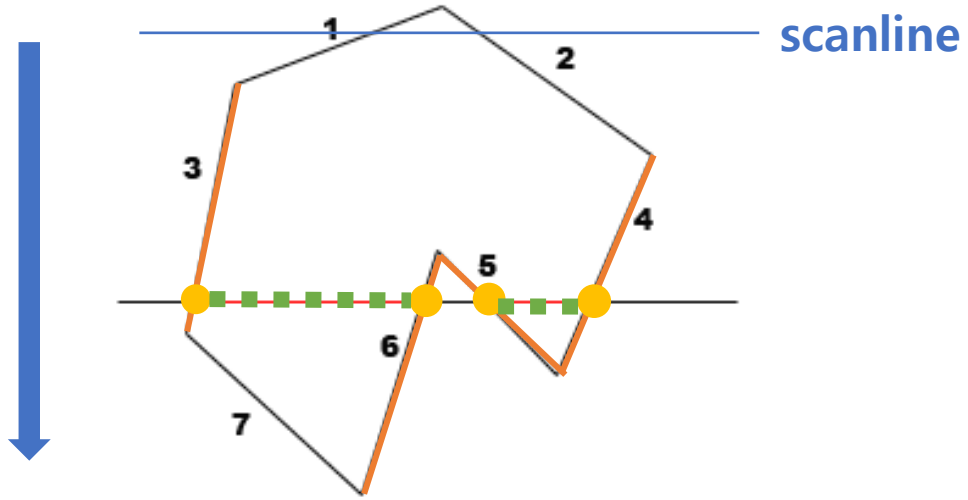



두 방법의 결과가 다르다.
경우에 따라 다른 방법을 선택해야 한다.

# Polygon Rasterization

# Polygon Rasterization – (1) Scanline Algorithm (일반적인 경우)

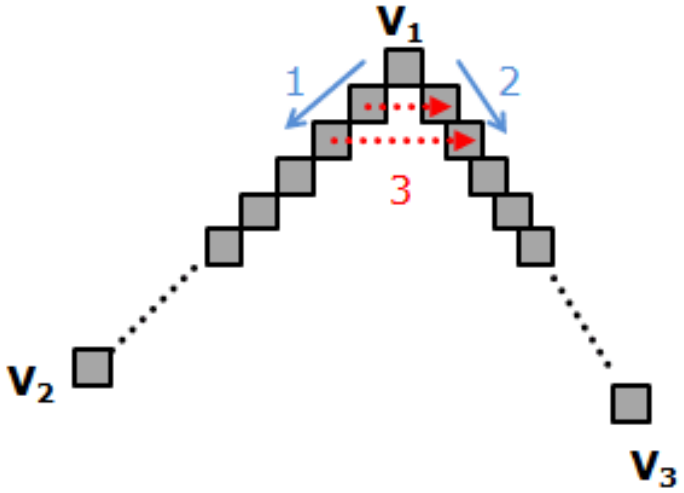
- **가상의 scan line 이 위에서 아래로 내려오며 채워 나간다**
  (y축 방향을 말한다)

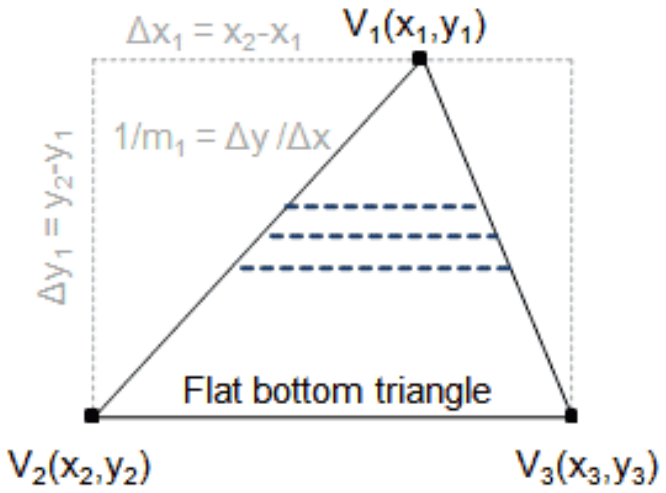



- **STEP1. 모든 edge를 y 기준으로 sorting**
- **STEP2. 각 scan line 마다**
  - **1) [edge]**
    scan line 과 만나는 edge 찾기
  - **2) [intersection]**
    scan line 과 그 edge 가 만나는 교점 찾기. 해당 교점들이 왼쪽에서 오른쪽으로 sorting 된 순서 찾기.
  - **3) [inside/outside 판단]**
    각 intersection point 의 interval 에 대해
    어디가 밖이고 안인지 (1) odd/even rule, (2) winding number 이용해서 찾기.
    교점으로 만들어진 선분 위의 임의의 점에서 ray 를 쏘면 된다.

# Polygon Rasterization – (2) Triangle Rasterization (삼각형에 특화)

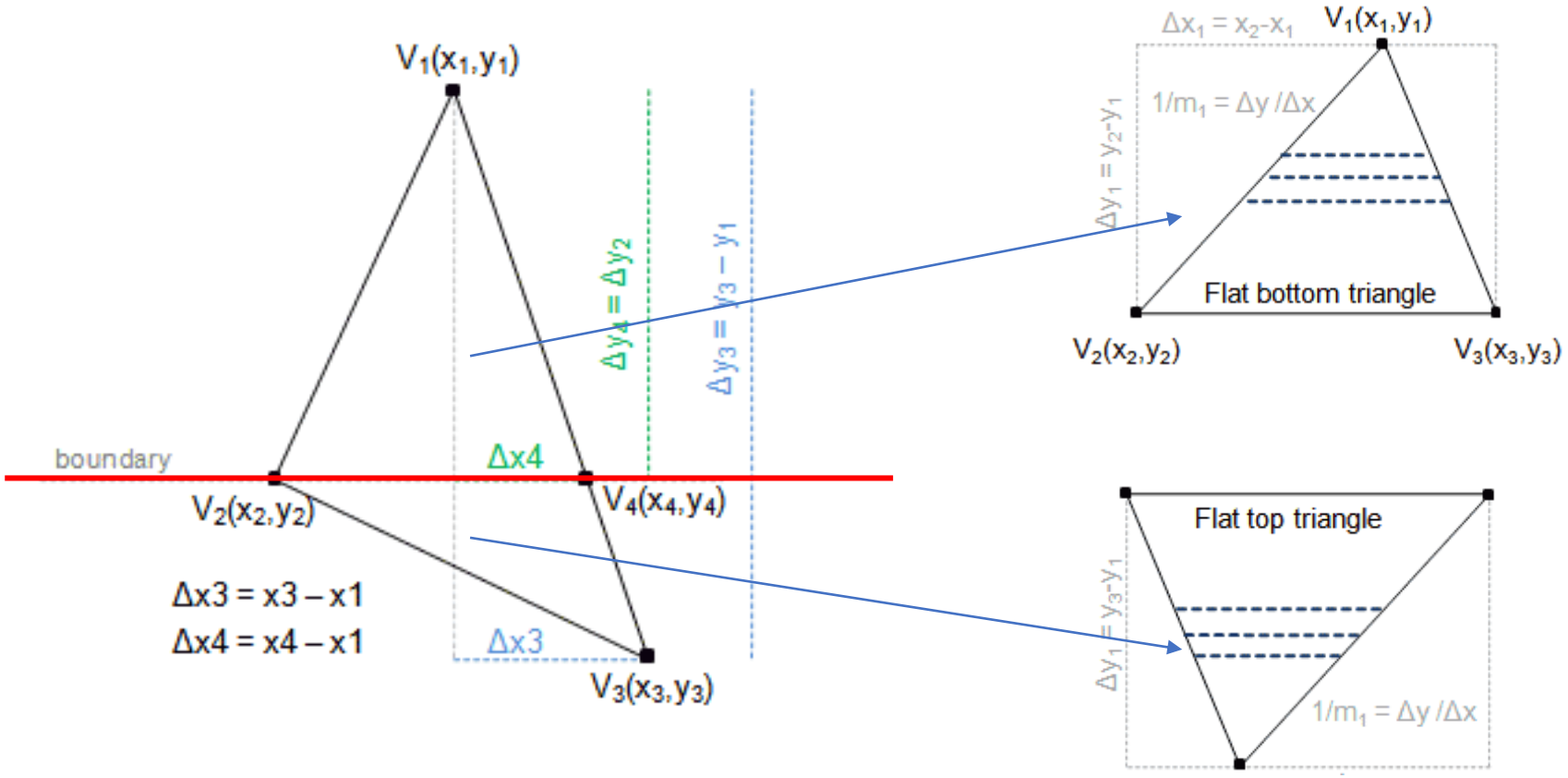
- 1. edge v1v2 에 대해 y 축 방향으로 pixel 이 하나 증가할 때 까지만 Bresenham's Algorithm 적용.
- 2. 다른 edge인 v1v3에 대해 마찬가지로 Bresenham's Algorithm 적용.
- 3. 그 사이를 채운다.
- v2나 v3에 도달할 때까지 1,2,3을 반복한다.



# Polygon Rasterization – (2) Triangle Rasterization (삼각형에 특화)

- Flat bottom triangle 이 아닌 경우 : flat bottom triangle 두개로 나누면 된다.

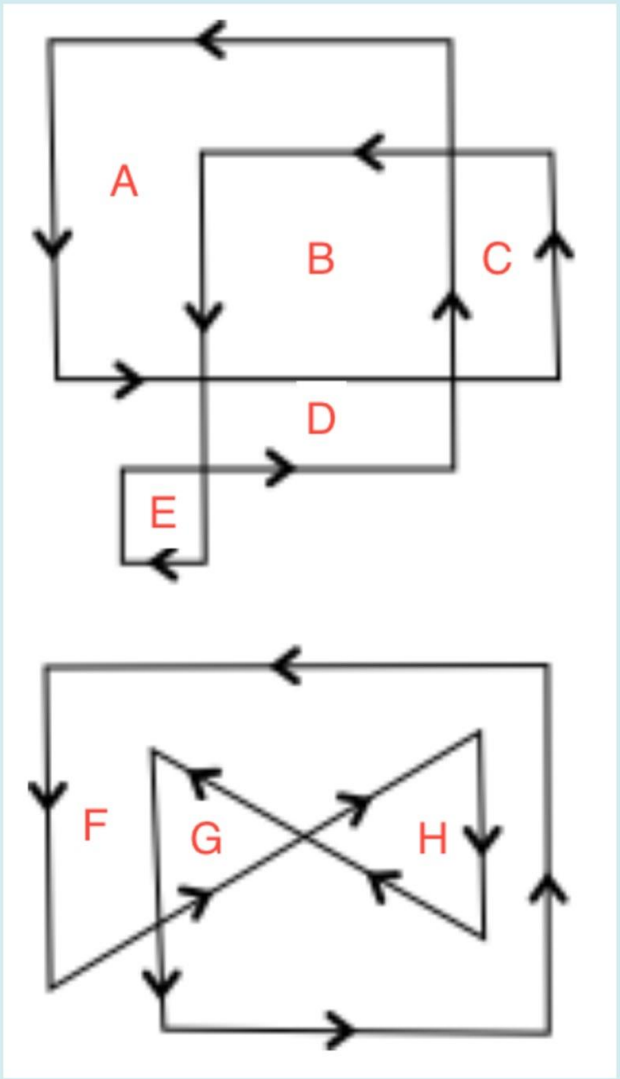


# QUIZ – Polygon Rasterization

정보

문제 표시

Given two non-simple polygons with edge order as shown below,




문제 1

정답

총 8.00 점에서 8.00 점 할당

문제 표시

Find the winding number of the region A~H

| | |
|---|---|
| A | 1 ✔ |
| B | 2 ✔ |
| C | 1 ✔ |
| D | 1 ✔ |
| E | -1 ✔ |
| F | 1 ✔ |
| G | 2 ✔ |
| H | 0 ✔ |

정답 : A → 1, B → 2, C → 1, D → 1, E → -1, F → 1, G → 2, H → 0

Which of A~H is classified as ***outside*** of the polygon according to the non-zero winding number rule?

하나 이상을 선택하세요.

Only H has zero for the winding number.

정답 : H