9. Polygon Rasterization

💡 **Definition of a Polygon**

Polygon Inside/Outside Test

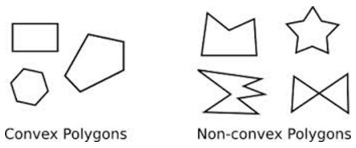
- **Odd/even rule**
- **Winding number**

Polygon Rasterization

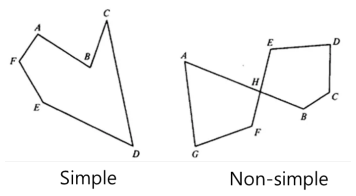
- **Scanline algorithm**
- **Triangle rasterization**

Definition of a Polygon

- (Polygon 정의) **3개 이상**의 꼭짓점이 edge 로 연결된 모양
- (Polygon 분류 1) **Convex (볼록)** vs **non convex (오목)**
  - 다각형 안의 임의의 점 2개를 찾아서 선분으로 연결했을 때,
    선분 전체가 영역 안에 들어가면 convex polygon



- (Polygon 분류 2) **Simple (edge crossing X)** vs **non simple (edge crossing O)**



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

- cf) Degenerate Polygon (수업 시간엔 없다고 가정, 실세계에서는 이런 경우를 다 고려해야 하므로 복잡하다.)
  - (case 1) Collinear: edge 가 동일 선상에 있어서 vertex 가 필요 없는 경우
  - (case2) Duplicated vertices : vertex 가 중복돼서 edge 길이가 0 인 경우

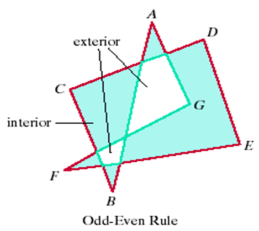
Polygon Inside/Outside Test

💡 Polygon 에서 어디가 inside이고 outside인지 확인하는 방법

(1) Odd/even rule

- Inside 에 있는지 확인하려는 점에서 임의의 방향으로 **ray** 를 쏜다.

💡 **Ray**: 시작은 있는데, 끝은 없는 line (방향 존재)
cf) line segment : 양끝점이 있는 line



- Polygon 의 edge 들과 교차를 확인
- Ray 가 총 몇 번 교차하는 지 판단
  - **홀수 → inside / 짝수 → outside**

(2) Winding number

- Winding number 의 **초기값은 0**
- Inside 에 있는지 확인하려는 점에서 임의의 방향으로 **ray** 를 쏜다.
- Polygon 의 edge 들과 교차를 확인, **교차 방향**에 따른 Winding number 증감
  - **Counter-clockwise (반시계 방향)** 로 교차하면, winding number **+1**
  - **Clockwise (시계 방향)** 로 교차하면, winding number **-1**
- Winding number 에 따라 판단
  - **0 아니면 → inside / 0 → outside**

💡 두 방법의 결과가 다르다.
경우에 따라 다른 방법을 선택해야 한다.

Winding Rule : Counter-clockwise cross vs Clockwise cross

- (판단 방법 1) (정의에 따른 방법)
  - Ray 기준으로 **left half-space, right half-space** 존재
  - **CCW: Right → left / CW: Left → right**
- (판단 방법 2) (수학적 판단 방법)
  - 그림이 주어지지 않고 **edge vector, ray vector** 주어졌을 때 CCW, CW 찾는 방법

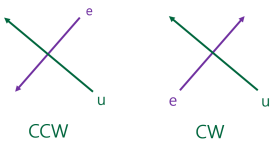
$$\text{Edge direction: } e = (e_x, e_y, 0)$$

$$\text{Ray direction: } u = (u_x, u_y, 0)$$

  - Cross product (외적) 이용
    - u x e 외적 결과인 **z 성분**에 따라 판단

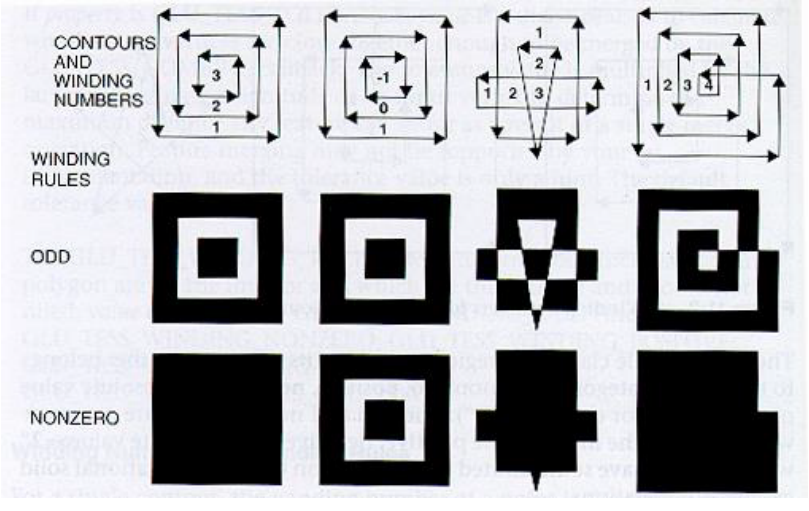
$$u \times e = u_x e_y - u_y e_x$$

    - **CCW: z >0 / CW: otherwise**



💡 Edge 의 방향 벡터 e, Ray 의 방향 벡터 u
⇒ 그림으로 주어진 경우에는, **시각적으로 정의된 e, u 벡터**를 이용해 Winding Rule 에 따라 판단 가능 (추가적인 벡터 계산 필요 없음)

Odd-Even VS Winding



💡 두 방법의 결과가 다르다.
경우에 따라 다른 방법을 선택해야 한다.

Polygon Rasterization

💡 이제 inside, outside 판단하는 법을 알았으니, 이를 활용해 inside 채우는 방법을 알아보자

(1) Scanline algorithm (일반적인 경우)

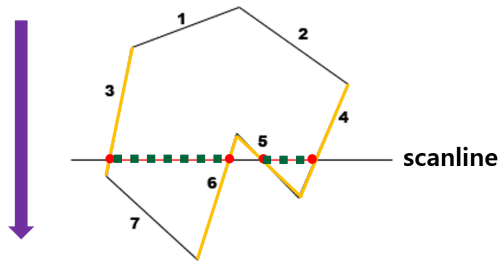
- 가상의 scan line 이 위에서 아래로 내려오며 채워 나간다. (y 축 방향을 말한다)

1. 모든 **edge** 를 y 기준으로 **sorting**

   💡 각 scan line 마다 모든 edge 와 교점을 계산할 필요 없이, **직전과 다음 교점만 비교하면 됨**
   ⇒ scan line 이 만나는 edge 를 효율적으로 찾음
   만약, 3번 edge 를 만나지 않으면 4번 edge 와의 교점을 고려할 필요 없음
   ⇒ 불필요한 교점 계산을 줄임

2. 각 scan line 마다
   ① **Edge**
   : Scan line 과 만나는 **edge 찾기**
   ② **Intersection**
   : Scan line 과 그 edge 가 만나는 **교점 찾기**
   - 해당 교점들을 x 축 기준, 왼쪽에서 오른쪽으로 sorting

   ③ **Inside/outside 판단**
   : 각 intersection point 의 **interval 에 대해 어디가 outside 이고 inside 인지**
   (1) odd/even rule, (2) winding number 이용해서 찾기
   - 교점으로 만들어진 선분 위의 임의의 점에서 ray 를 쏘면 된다.



(2) Triangle rasterization (삼각형에 특화)

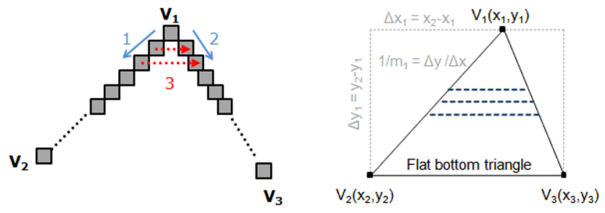
(가정) Flat bottom triangle (y2, y3 가 같아서 평행) 이라고 하자

① edge **v1v2** 에 대해, y 축 방향으로 **pixel 이 하나 증가할 때까지만** Bresenham's Algorithm 적용

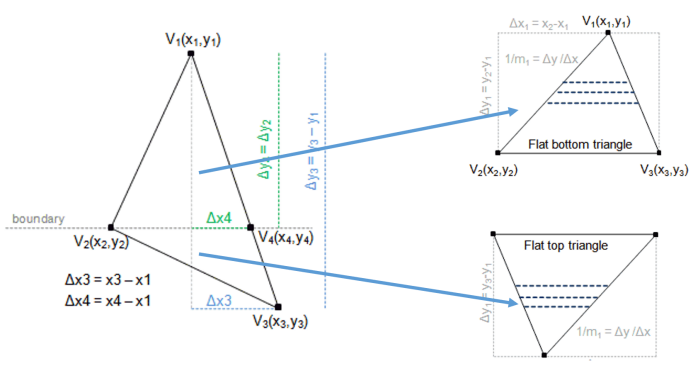
② 다른 edge **v1v3** 에 대해, 위와 똑같이 Bresenham's Algorithm 적용

③ 그 사이를 채운다.

⇒ v2 나 v3 에 도달할 때까지 1, 2, 3 을 반복한다.

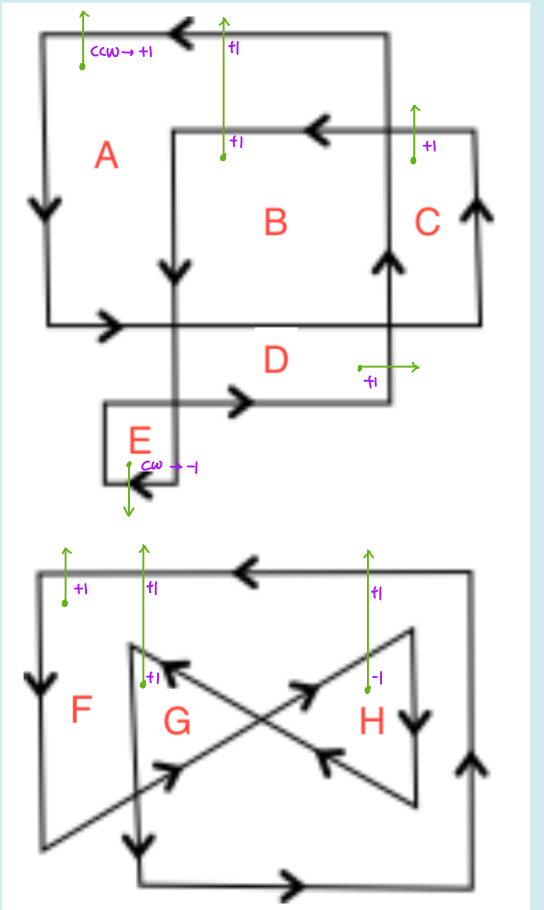


💡 Flat bottom triangle 이 아닌 경우: flat bottom triangle 두 개로 나누면 된다.



QUIZ) Polygon Rasterization

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



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?

winding number 가 0인경우, outside

하나 이상을 선택하세요.

- [ ] 1. A
- [ ] 2. B
- [ ] 3. C
- [ ] 4. D
- [ ] 5. E
- [ ] 6. F
- [ ] 7. G
- [x] 8. H ✔
- [ ] 9. None of the above

Only H has zero for the winding number.

정답 : H