

# 컴퓨터 그래픽스

## 제4장 2차원 그래픽스의 기본 요소

2017년 2학기

# 4장 학습 내용

- 2차원 그래픽스 기본 요소
  - 점
  - 선
  - 원
  - 영역 채우기
  - 앨리어싱 효과

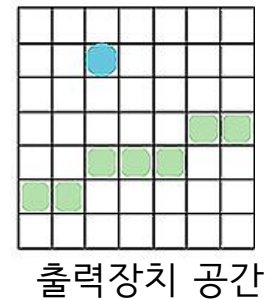
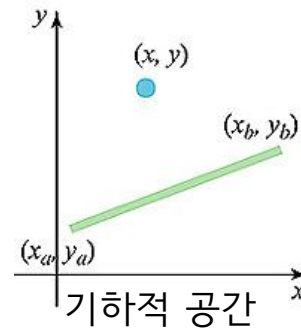
# 점과 선의 정의 및 속성

- 2차원 그래픽스의 기본적인 출력 요소

- 점, 선, 다각형, 원, 타원, 곡선, 문자 등
- 점과 선은 모든 2차원 그래픽스 객체 표현의 기본 요소

- 점 (Point)

- 래스터 방식의 출력장치에서의 기본 요소
  - 점의 속성: 크기, 명암, 색상, 모양 등
- 기하공간에서의 점: 좌표  $(x, y)$



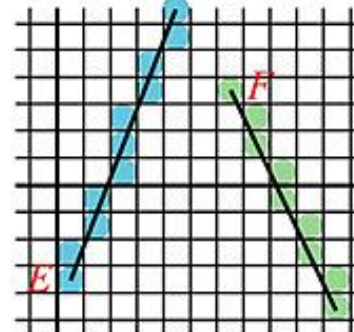
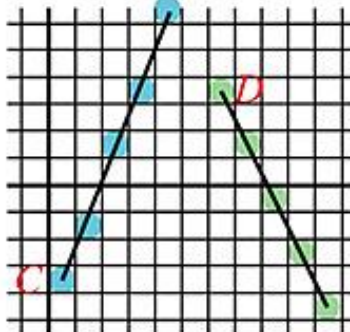
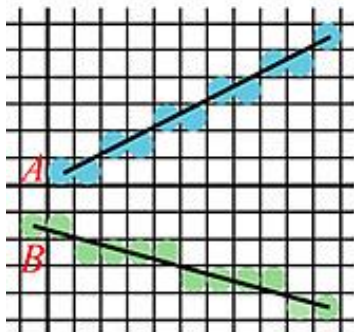
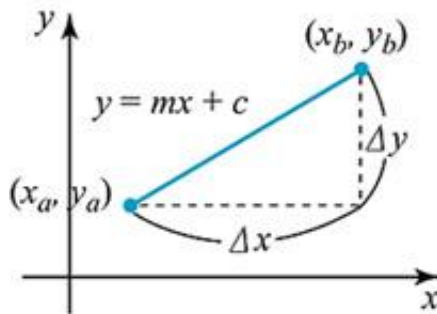
- 선 (Line)

- 시작점  $(x_a, y_a)$ 과 끝점  $(x_b, y_b)$  또는 시작점 좌표  $(x_a, y_a)$ 와 증가 값  $(\Delta x, \Delta y)$ 의 상대좌표로 정의
  - 선의 속성: 유형, 굵기, 색상, 선 끝 모양 등
- 직선 방정식을 이용하여 선의 좌표 값 구하기
  - $y = mx + c$        $m$ : 기울기  $c$ : y축 절편
  - 두 끝점  $(x_1, y_1)$   $(x_2, y_2)$ 를 사용하여  $m$ 과  $c$ 를 구한다
    - $m = (y_2 - y_1) / (x_2 - x_1)$
    - $c = y_1 - mx_1$

# 선 그리기: DDA 알고리즘


## • DDA (Digital Differential Analyzer) 알고리즘

- 선의 양끝 좌표로부터 래스터 출력 장치로 변환하는 가장 기본적인 알고리즘
- 선의 공식을  $y = mx + c$ 의 형태로 계산
- $0 \leq m \leq 1$  인 경우,  $x$ 를 1씩 증가시키면  $y$ 값은  $m$ 만큼 증가
- $m > 1$ 인 경우, 그 반대로  $y$ 를 매번 1씩 증가시키면  $x$ 값이  $\frac{1}{m}$ 만큼씩 증가
- 기울기  $m$ 이 음수인 경우,  $x$  증가에 따라  $y$ 값이 증가 대신 감소



# 선 그리기: DDA 알고리즘

- 초기화를 한다.
  - $\Delta x = x_b - x_a$  ,  $\Delta y = y_b - y_a$  ,
  - $m = \frac{\Delta y}{\Delta x}$
  - $x_1 = x_a$  ,  $y_1 = y_a$
- 기울기에  $m$ 의 값에 따라 다음계산을 수행한다.
  - 기울기  $|m| \leq 1$ 인 경우, 매번  $k+1$ 번 째 점에서 ( $1 \leq k \leq \Delta x$ )

$x_{k+1} = x_k + 1$  

$y_{k+1} = y_k + m$

$y_{k+1}$ 의 래스터 좌표 = Round( $y_{k+1}$ )
  - 기울기  $|m| > 1$ 인 경우, 매번  $k+1$ 번 째 점에서 ( $1 \leq k \leq \Delta y$ )

$y_{k+1} = y_k + 1$

$x_{k+1} = x_k + \frac{1}{m}$

$x_{k+1}$ 의 래스터 좌표 = Round( $x_{k+1}$ )

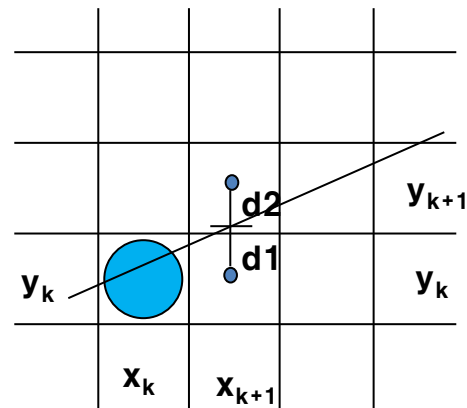
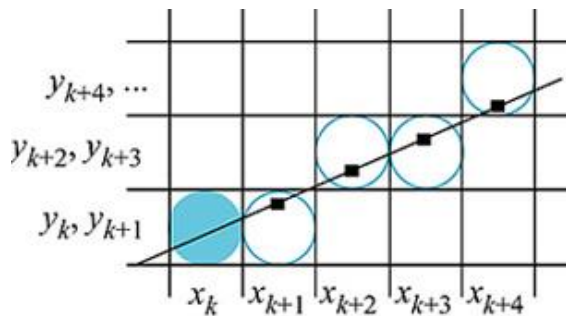
# 선 그리기: DDA 알고리즘

- DDA 알고리즘의 특징
  - 곱하기가 없이 소수점(Floating-point) 더하기 연산만을 반복
  - 부동 소수 연산 사용, 정수연산에 비해서는 상대적으로 **속도가 떨어진다.**
  - 반올림 연산 함수의 실행시간이 걸린다.
  - 매번 정수좌표를 구할 때마다 **오차가 축적**

# 선 그리기: 브레즌햄 알고리즘

- Bresenham 알고리즘

- 선을 구성하고 있는 어느 한 점의 다음 점은 반드시 오른쪽 점 또는 오른쪽 바로 위의 점이 된다.
- 가능한 두 점 중, 실 선과 두 개의 가능한 점의 차이 (아래 그림에서 d1과 d2)가 더 작은 점을 선택하여 선을 나타내는 알고리즘
- 소수점 계산 없이 정수의 더하기 연산과 이동 연산만으로 처리되므로 속도가 빠르다.



# 선 그리기: 브레즌햄 알고리즘

- 알고리즘 초기화

- 기울기가 1보다 작은 경우 ( $|m| < 1$ ):

- $y = mx + c, m = \frac{\Delta y}{\Delta x}$ 
      - 시작점:  $(x_a, y_a),$
      - 가능한 두 점:  $(x_a+1, y_a), (x_a+1, y_a+1)$
  
      - 일반적인 k번째 점:  $(x_k, y_k),$
      - 가능한 두 점:  $(x_k+1, y_k), (x_k+1, y_k+1)$



# 선 그리기: 브레즌햄 알고리즘

## - 다음 점 $x_{k+1}$ 에서

$$y = mx_{k+1} + c$$

$$d_1 = y - y_k = m(x_k + 1) + c - y_k$$

$$d_2 = (y_k + 1) - y = (y_k + 1) - (m(x_k + 1) + c)$$

$$\begin{aligned} d_1 - d_2 &= \{m(x_k + 1) + c - y_k\} - \{y_k + 1 - (m(x_k + 1) + c)\} \\ &= 2m(x_k + 1) - 2y_k + 2c - 1 \quad (d_1 - d_2: \text{두 거리 사이의 차이}) \end{aligned}$$

## - 양변에 $\Delta x$ 를 곱한다

## - $(d_1 - d_2) \Delta x$ 를 $p_k$ (판단매개변수)라고 하면 ( $m = \Delta y / \Delta x$ )

$$p_k = (d_1 - d_2) \Delta x$$

$$p_k = 2 \Delta y (x_k + 1) + \Delta x(-2y_k + 2c - 1) = 2 \Delta y x_k - 2 \Delta x y_k + 2 \Delta y + \Delta x (2c - 1)$$

$p_k$  에  $p_{k+1}$ 를 대입하면,

$$p_{k+1} = 2 \Delta y \cdot x_{k+1} - 2 \Delta x y_{k+1} + 2 \Delta y + \Delta x (2c - 1)$$

$$p_{k+1} - p_k = 2 \Delta y (x_{k+1} - x_k) - 2 \Delta x (y_{k+1} - y_k)$$

$$p_{k+1} = p_k + 2 \Delta y - 2 \Delta x (y_{k+1} - y_k)$$

# 선 그리기: 브레즌햄 알고리즘

-  $p_k$ 의 부호에 따라

$$p_k \geq 0 \rightarrow d_1 - d_2 \geq 0 \rightarrow d_1 \geq d_2 \rightarrow y_{k+1} = y_k + 1$$

$$p_k < 0 \rightarrow d_1 - d_2 < 0 \rightarrow d_1 < d_2 \rightarrow y_{k+1} = y_k$$

$$p_k \geq 0 \rightarrow p_{k+1} = p_k + 2(\Delta y - \Delta x) \quad (y_{k+1} - y_k == 1)$$

$$p_k < 0 \rightarrow p_{k+1} = p_k + 2\Delta y \quad (y_{k+1} - y_k == 0)$$

따라서  $p_k < 0$  이면  $\rightarrow$  다음 점은  $(x_k+1, y_k)$   
 $0 \leq p_k$  이면  $\rightarrow$  다음 점은  $(x_k+1, y_k+1)$

- 시작점:  $(x_a, y_a)$ 일 때 첫 번째 매개변수 값은,

$$\begin{aligned} p_1 &= 2\Delta y x_k - 2\Delta x y_k + 2\Delta y + \Delta x(2c - 1) \\ &= 2\Delta y x_a - 2\Delta x y_a + 2\Delta y + \Delta x(2c - 1) \\ &= 2\Delta y x_a - 2\Delta x(mx_a + c) + 2\Delta y + \Delta x(2c - 1) \\ &= 2\Delta y x_a - 2(\Delta y x_a + \Delta x c) + \Delta y + \Delta x(2c - 1) \\ &= 2\Delta y - \Delta x \end{aligned}$$

# 선 그리기: 브레즌햄 알고리즘

- 브레즌햄 선 그리기 알고리즘 정리:
  - 기울기가 0과 1 사이인 경우에 적용
  - 초기값을 구한다.
    - 시작점의 좌표:  $(x_1, y_1)$
    - $C1 = 2\Delta y$   $C2 = 2(\Delta y - \Delta x)$
    - $p_1 = 2\Delta y - \Delta x$
  - 판별식  $p_k$ 값에 따라 다음 점의 위치를 구한다.
    - $p_k < 0 \rightarrow$  다음 점:  $(x_k + 1, y_k)$   $p_{k+1} = p_k + C1$
    - $p_k \geq 0 \rightarrow$  다음 점:  $(x_k + 1, y_k + 1)$ ,  $p_{k+1} = p_k + C2$

# 원 그리기

- 원이나 타원 등 곡선은 매개변수 형태의 함수로 표현된다.

- $y = f(x)$  또는  $x = g(\theta)$ ,  $y = h(\theta)$
- 점들을 선분으로 연결하여 곡선의 모양을 근사적으로 그린다.
- 원의 공식:  $x^2 + y^2 = r^2$

- 직교 좌표계에서  $(x, y)$  를 함수 형태로 표현하면

$$y = \pm \sqrt{r^2 - x^2}$$

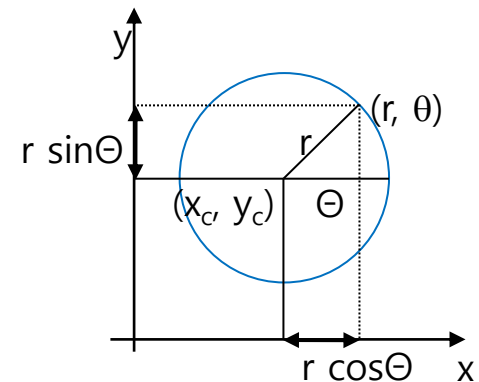
- 극 좌표계 (Polar Coordinate)에서 매개변수 함수로 표현하면

- $x = r \cos \theta$ ,  $y = r \sin \theta$

- 원의 중심이  $(x_c, y_c)$  일 때는,

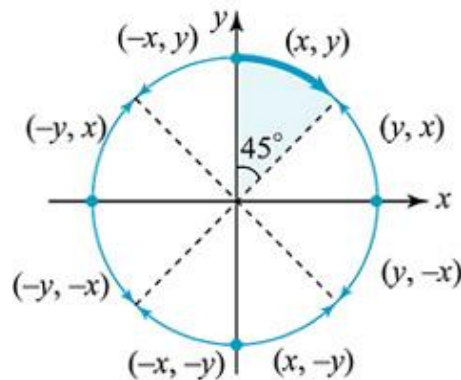
- 원의 공식은  $(x - x_c)^2 + (y - y_c)^2 = r^2$

- 극좌표식은  $x = x_c + r \cos \theta$ ,  
 $y = y_c + r \sin \theta$



# 원 그리기: 브레즌햄 알고리즘

- Bresenham 알고리즘
  - 제곱근이나 삼각함수 등의 계산이 없이 정수 연산만으로 처리
  - 각도가  $45 \leq \theta \leq 90^\circ$ 인 부분에 대하여 계산
  - x방향으로 1만큼 증가  $\rightarrow$  y축에서는 같은 점 또는 1감소된 점:  
k번째 점  $(x_k, y_k) \rightarrow$  k+1번째 점  $(x_k+1, y_k)$  또는  $(x_k+1, y_k-1)$



# 원 그리기: 브레즌햄 알고리즘

-  $x_k < y_k$  인 동안 반복

$$x^2 + y^2 = r^2$$

$$y^2 = r^2 - x^2 \rightarrow y_{k+1}^2 = r^2 - (x_k + 1)^2$$

$$d_1 = y_k^2 - y^2$$

$$d_2 = y^2 - (y_k - 1)^2$$

$$p_k = d_1 - d_2 = (y_k^2 - y^2) - \{y^2 - (y_k - 1)^2\}$$

$$p_{k+1} = (y_{k+1}^2 - y^2) - \{y^2 - (y_{k+1} - 1)^2\}$$

$$p_{k+1} - p_k = 2 y_{k+1}^2 - 2 y_k^2 - 2 y_{k+1} + 2 y_k + 4(x_k + 1) + 2$$

$$p_k < 0 \rightarrow d_1 - d_2 < 0$$

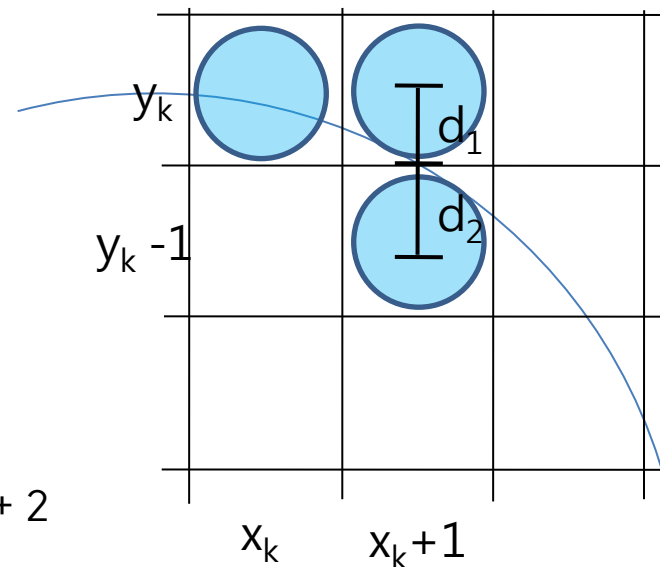
다음 점은  $y(x_k + 1, y_k)$

$$p_k \geq 0 \rightarrow d_1 - d_2 \geq 0$$

다음 점은  $(x_k + 1, y_k - 1)$

$$p_1 = (y_1^2 - y^2) - \{y^2 - (y_1 - 1)^2\} = 3 - 2r$$

$$\text{초기화: } x_1 = x_c, \quad y_1 = y_c + r, \quad p_1 = 3 - 2r$$

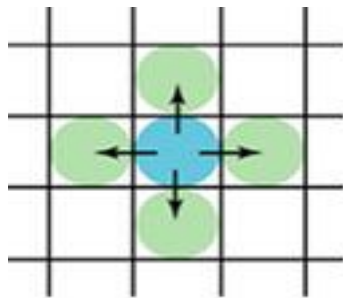


$$p_{k+1} = p_k + 4x_k + 6$$

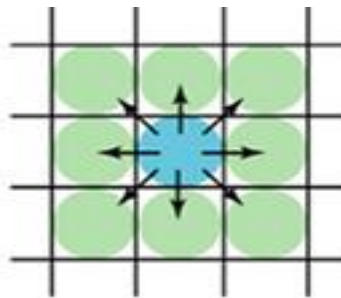
$$p_{k+1} = p_k + 4(x_k - y_k) + 10$$

# 영역 및 다각형 채우기

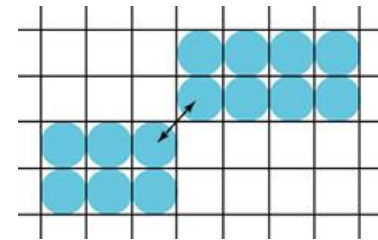
- 2차원 그래픽스에서 영역
  - 모든 그림들은 픽셀들로 구성되고,
    - 선이나 도형이 서로 만나서 영역이 생성된다.
- 영역의 특성
  - 영역: 같은 색상 값을 갖는 이웃한 픽셀들의 집합
  - 이웃한 픽셀간의 연결 방식 (픽셀의 연결 방식)



4방향 연결



8방향 연결



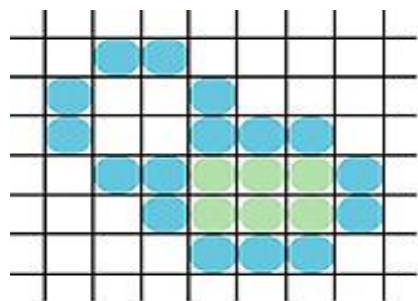
영역 연결방식의 예:

4방향 연결 - 2개의 영역

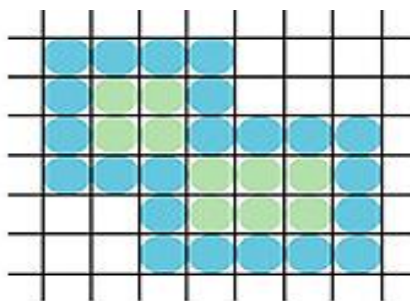
8방향 연결 - 1개의 영역

# 영역 및 다각형 채우기

- 래스터 출력에서 영역의 경계 픽셀과 내부 픽셀은 연결방식을 다르게,
  - 경계 8방향 연결  $\Rightarrow$  내부는 반드시 4방향 연결 채우기
  - 경계 4방향 연결  $\Rightarrow$  일반적으로 내부는 8방향 연결 채우기
- 일반적인 래스터 방식의 출력장치
  - Bresenham 선 그리기 알고리즘은 8방향연결 방식
  - 영역 채우기 알고리즘은 내부 영역을 4방향연결 방식으로 채우기



8방향 연결 방식의 경계  
4방향 연결 방식의 내부



4방향 연결 방식의 경계  
8방향 연결 방식의 내부



# 영역 및 다각형 채우기

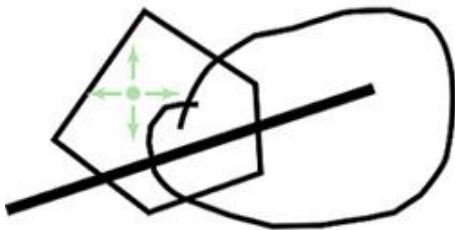
- 영역 채우기 알고리즘

- 시드 채우기 방식

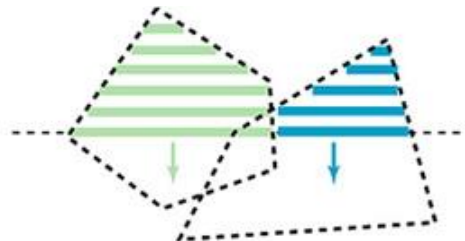
- 그림이 래스터 버퍼에 그려진 후 이미지에서 영역의 채우기를 실행
    - 영역 내부의 한 픽셀이 시드로 주어지고 이 픽셀에서부터 채워나간다
    - 주로 페인팅 소프트웨어나 대화식 이미지 처리 프로그램 (사용자가 원하는 영역을 클릭하면 그 점을 시드로 하여 채우기를 실행)에서 사용

- 다각형 주사변환 방식

- 매 주사선 별로 다각형의 내부 구간을 판단하여 해당 픽셀을 칠한다.
    - 주사선채우기(Scan-line Fill)라고도 한다.
    - 주로 벡터방식의 그리기 소프트웨어에서 사용 (채우기를 하는 도형의 벡터 데이터를 가지고 있다)



시드 채우기 방식

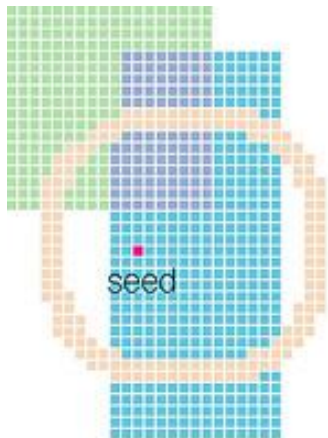


다각형 주사변환 방식

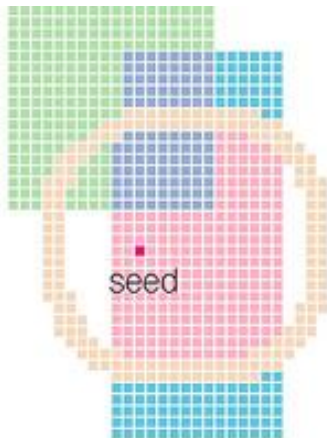
# 영역 및 다각형 채우기

- 시드 채우기 (Seed fill) 방식

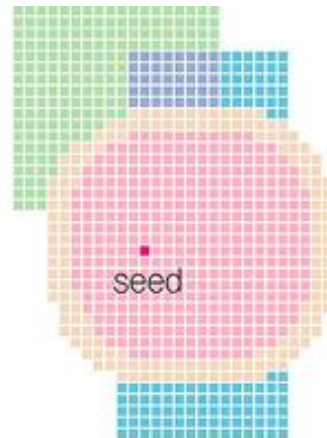
- 다각형 내부의 한 점 (x, y)가 seed로 주어진다.
- 이 점을 중심으로 이웃 픽셀이 영역의 내부에 있는지를 판단하여 영역 채우기를 한다.
- 내부 영역에 대한 판단
  - Interior-defined: 같은 값을 가지고, 연결된 픽셀들을 내부 영역으로 판단 → 범람 채우기 (Flood Fill)
  - Boundary-defined: 경계의 안쪽에 위치하는 픽셀들을 내부 영역으로 판단 → 경계 채우기 (Boundary Fill)



주어진 시드



범람 채우기  
(Flood-fill)



경계 채우기  
(Boundary-fill)

# 영역 및 다각형 채우기

## - 알고리즘 진행 방법

- 내부의 한 점 시드(seed)를 스택에 저장한다
- Seed pixel을 중심으로 4방향 또는 8방향의 이웃 픽셀에 대해 내부의 점인지를 확인
- 재귀적 함수 (Recursive) 사용하여 이웃한 픽셀들을 검사해나간다.

# 영역 및 다각형 채우기

- 범람 채우기 알고리즘

```
void flood_fill (int x, int y)
{
    if (read_pixel (x, y) == bgColor)
    {
        write_pixel (x, y, fillColor);
        flood_fill (x+1, y);
        flood_fill (x-1, y);
        flood_fill (x, y+1);
        flood_fill (x, y-1);
    }
}
```

```
// 시드 (x, y) 에서 시작
// 현재 픽셀이 배경색 'bgColor'이면,
// 채우기 색 'fillColor'로 칠한다.
// 오른쪽으로 반복
// 왼쪽으로 반복
// 아래로 반복
// 위로 반복
```

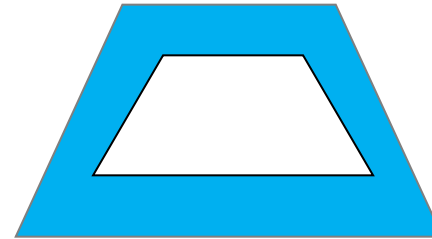
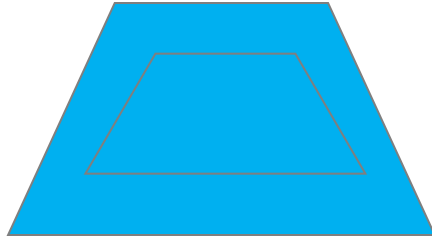
- 경계 채우기 알고리즘

```
void boundary_fill(int x, int y)
{
    current = read_pixel(x, y);
    if ((current != bdColor) && (current != fillColor)) // 경계 및 채울 색인지 확인
    {
        write_pixel (x, y, fillColor);
        boundary_fill (x+1, y);
        boundary_fill (x-1, y);
        boundary_fill (x, y+1);
        boundary_fill (x, y-1);
    }
}
```

```
// 시드 (x, y) 에서 시작
// 내부를 채우기 색 'fillColor'로 칠한다.
// 오른쪽으로 반복
// 왼쪽으로 반복
// 아래로 반복
// 위로 반복
```

# 다각형 내부 판단 규칙

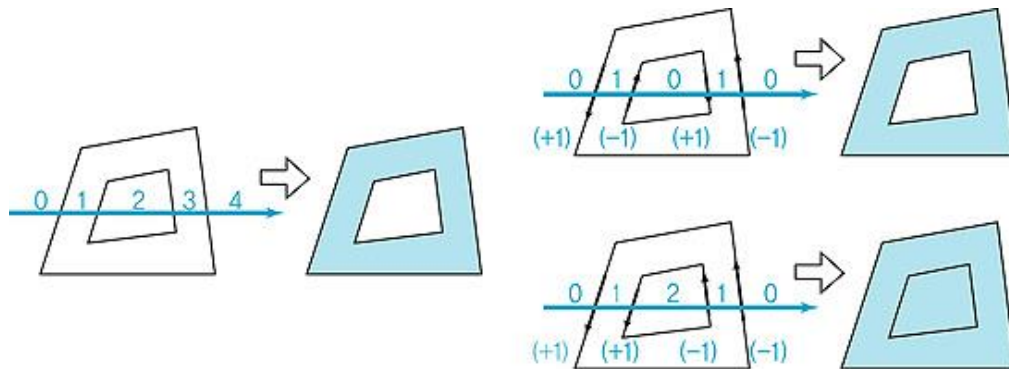
- 여러 개의 다각형으로 구성된 복잡한 도형이 주어지면 내부 영역을 다르게 판단할 수 있다.



- 판단 규칙
  - 홀짝 규칙 (Even-Odd rule)
    - 매 주사선별로 x값을 증가하면서
      - 다각형의 에지가 홀수 번째 교차하면 내부 구간이 시작
      - 짝수 번째 교차하면 외부 구간이 시작된다.
    - 알고리즘이 간단하다.
    - 서로 다른 두 개의 다각형이 겹쳐있을 때 그 겹친 부분은 항상 외부 영역으로 판단

# 다각형 내부 판단 규칙

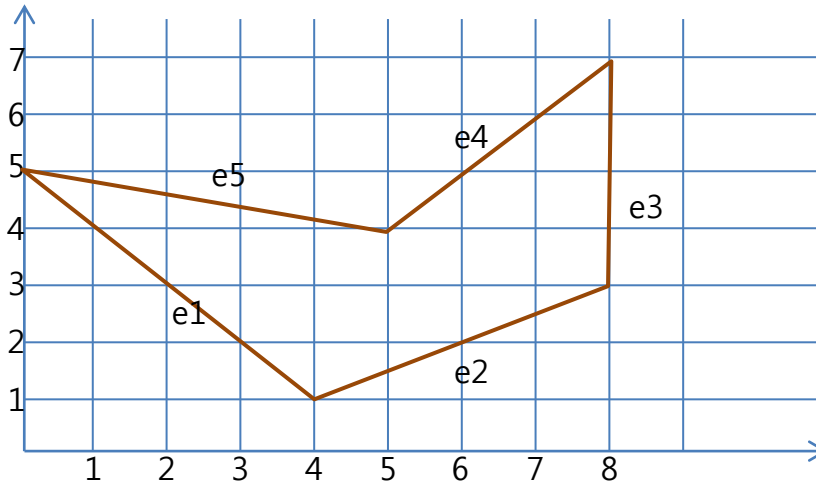
- 접기회수 규칙 (Non-Zero Winding Rule): 에지의 방향을 고려
  - 다각형에서 각 에지의 벡터 방향은 꼭짓점이 주어진 순서에 따라 정해진다
  - 각 주사선에서 아래쪽 방향의 모서리와 교차: 1 증가
  - 각 주사선에서 위쪽 방향의 모서리와 교차: 1 감소
  - 합이 0이면 → 외부
  - 합이 0이 아니면 → 내부
  - 다각형 모서리의 방향에 따라 내부와 외부영역을 지정해줄 수가 있다.
  - 홀짝 규칙보다 약간 복잡, 도형 설계에서 자유롭게 내부와 외부 영역 지정 가능, 정교한 드로잉 소프트웨어에서 많이 사용



# 영역 및 다각형 채우기

- 다각형 주사 변환 방식 (Polygon scan-conversion)
  - 매 주사선마다 교차되는 edge(에지, 모서리)들의 목록을 유지, 갱신하여 영역을 설정한다.
  - 가장 대표적인 방법: **Y-X 다각형 주사선 알고리즘**
  - Edge list
    - **에지 목록** - EL (Edge List): 다각형의 전체 edge의 목록
      - 시작점의 y 좌표값 (더 작은 y 값) 순서로 다각형의 전체 에지를 정렬하여 전체 에지의 EL 구성
      - 매 주사선에서 교차하는 에지를 EL에서 꺼내어 AEL로 옮겨 관리
    - **활성화된 에지 목록** - AEL (Active Edge List): 각 주사선과 교차하여 활성화 된 edge 목록
      - 해당 주사선과 각 에지와의 교차점의 x값을 구한 후 2개 씩 짝을 만들어 이들 사이를 채운다.
    - 그리기가 완료된 AEL내의 에지를 찾아서 제거 (AEL의 에지 중 아래쪽 점의 y 좌표가 주사선의 y좌표보다 작게 되면 EL에서 제거)

# 영역 및 다각형 채우기



e1 (0, 5) (4, **1**)  
 e2 (4, **1**) (8, 3)  
 e3 (8, **3**) (8, 7)  
 e4 (8, 7) (5, **4**)  
 e5 (0, 5) (5, **4**)

EL = {e1, e2, e3, e4, e5} -> 시작점의 y값 (작은 값)에 따라 정렬: {e2, e1, e3, e5, e4}

Y=1: AEL = {e2, e1}



Y=2: AEL = {e2, e1}

Y=3: AEL = {e2, e2, e3}

Y=4: AEL = {e1, e3, e5, e4}

Y=5: AEL = {e1, e3, e5, e4}

Y=6: AEL = {e3, e4}

Y=7: AEL = {e3, e4}

Y=8: AEL = { }



# 영역 및 다각형 채우기

- Y-X 다각형 주사선 알고리즘의 특징

- Y-X 알고리즘 : Y값 순서로 전체 에지 정렬, 교차점은 X좌표값 순서로 정렬
- 효율성 : 에지의 목록에 대한 부분적인 일관성(Coherence)으로 발생

- Y-X 다각형 주사선 알고리즘

- 1) 초기화를 한다.

각 에지들을 Y좌표의 최소값 순서로 정렬하여 Edge List(EL)를 구성한다.

- 2) 매 주사선  $y_k$ 에서 다음을 수행한다.

- a) AEL을 갱신한다.

AEL에서  $y_b < y_k$  인 에지를 삭제하고, // 완료된 에지 삭제

EL에서  $y_a = y_k$  인 에지를 AEL로 이동한다. // 새로운 에지 삽입

단, AEL 과 EL에 더 이상의 에지가 없으면 종료한다.

- b) AEL에서 각 에지의 교차점을 계산한다.

- c) 교차점 x값을 정렬한 후 각 쌍을 결정하여 그 사이를 채운다

# Antialiasing

- 래스터 출력의 문제점

- 앨리어싱 효과

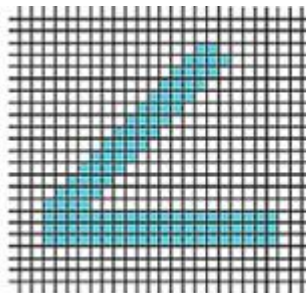
- 계단 현상 (jaggies, aliasing)
    - 모양이 들쭉 날쭉하고 선이 움직일 때 위치가 바뀐다.
    - 작은 물체가 깜빡 깜빡 한다. (blinking)

- 앨리어싱이 생기는 이유

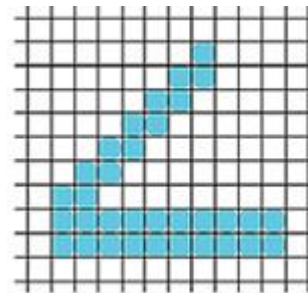
- 저해상도의 출력장치에서 두드러진다.
    - 아날로그 방식의 그림을 디지털 화 하는데 샘플링 오차가 발생



(a) 원래 그림



(b) 고해상도 출력



(c) 저해상도 출력

# Antialiasing

- 안티 앨리어싱(Antialiasing)

- 컬러 또는 회색조(Gray) 출력 장치에서 경계가 부드럽게 보이도록 하는 기법
- 물체의 경계 픽셀에서 물체와 배경의 색상을 혼합해서 그린다.
- 선 그리기, 다각형 채우기, 문자 생성 등에 적용이 가능
- 해상도를 높인다. → 물리적 해상도의 한계
- 안티 앨리어싱 기법
  - 수퍼 샘플링 (Super sampling) 기법
  - 영역 샘플링 (Area sampling) 기법

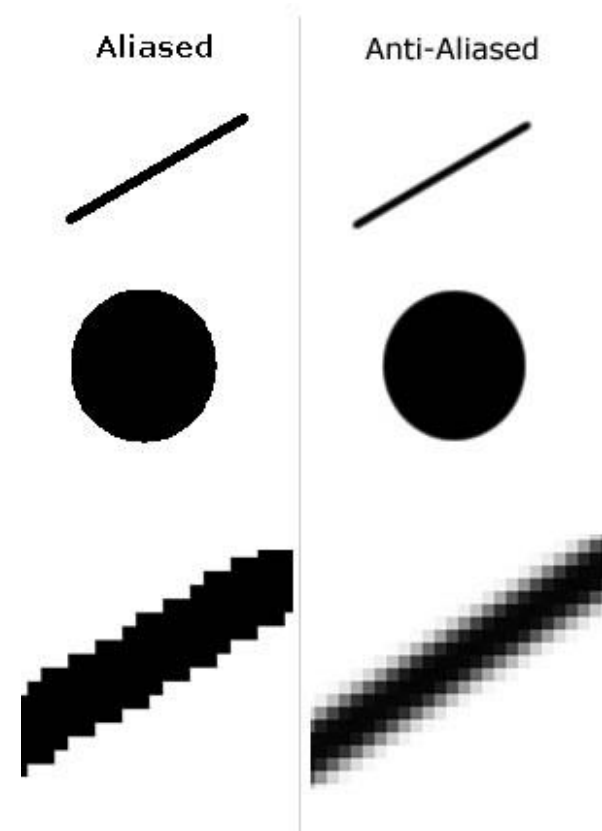
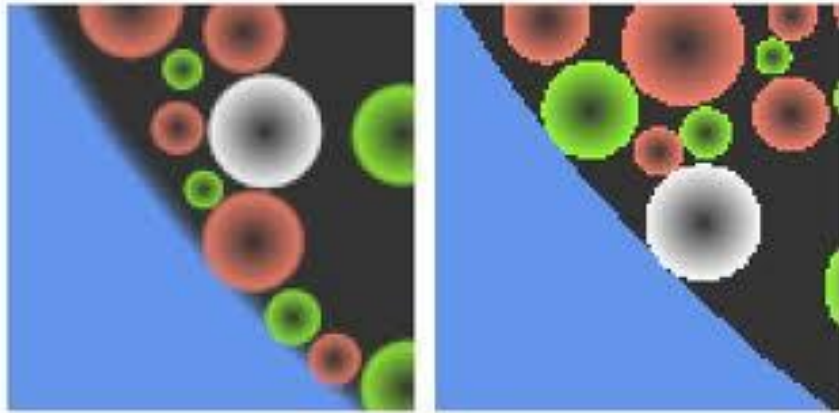
**ABCDE**  
**ABCDE**

앨리어싱 효과

안티앨리어싱 효과



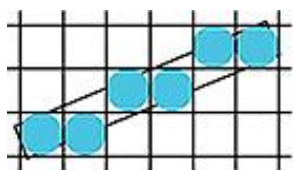
# *Antialiasing*



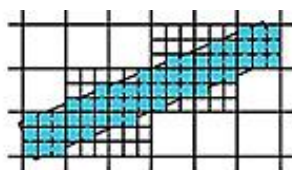
# Antialiasing: 슈퍼 샘플링 기법

- Super sampling 기법

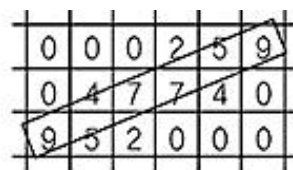
- 출력 장치의 해상도보다 고해상도에서 그림을 자세히 표현할 수 있도록 하나의 픽셀 영역을 여러 개로 분할하는 기법.
- 원래의 해상도로 환원할 때 픽셀의 명암값을 계산하여 보여준다.
  - 픽셀의 영역에 포함되는 고해상도 픽셀의 개수에 비례하여 명암값을 계산



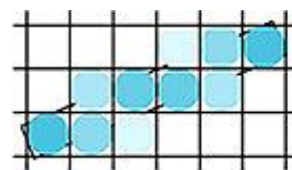
(a) 원래 해상도



(b) 슈퍼샘플링

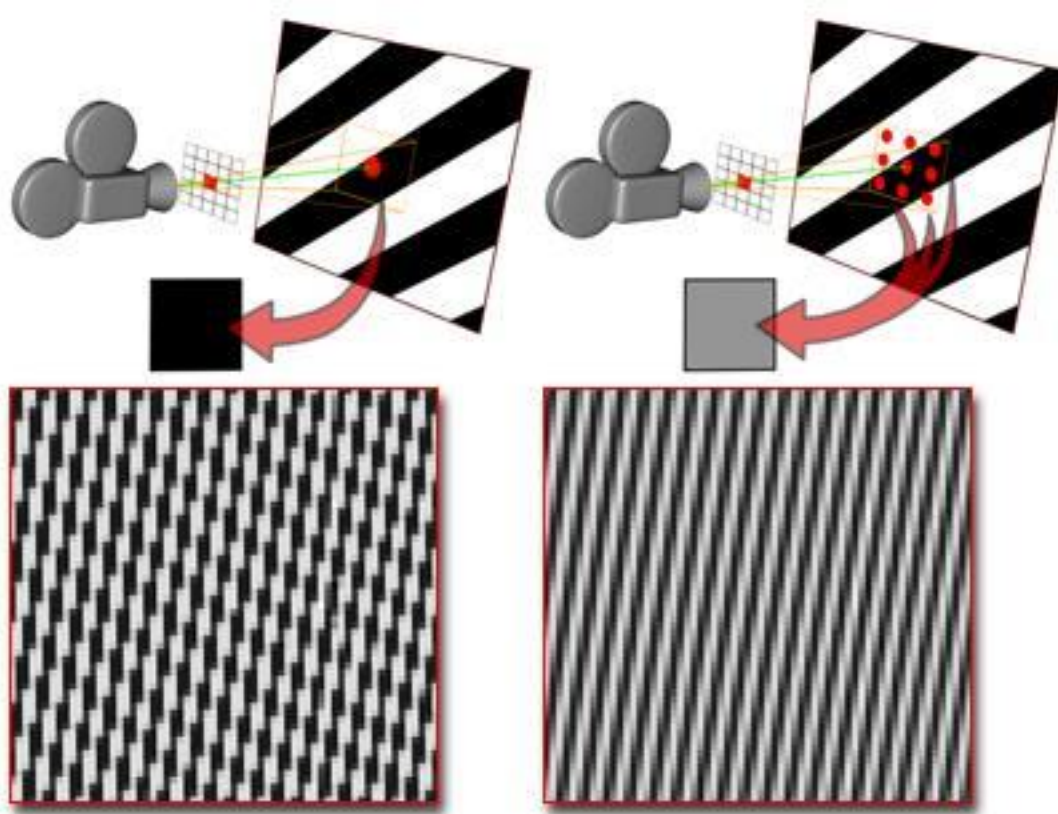


(c) 각 픽셀의 명암 값



(d) 출력 결과

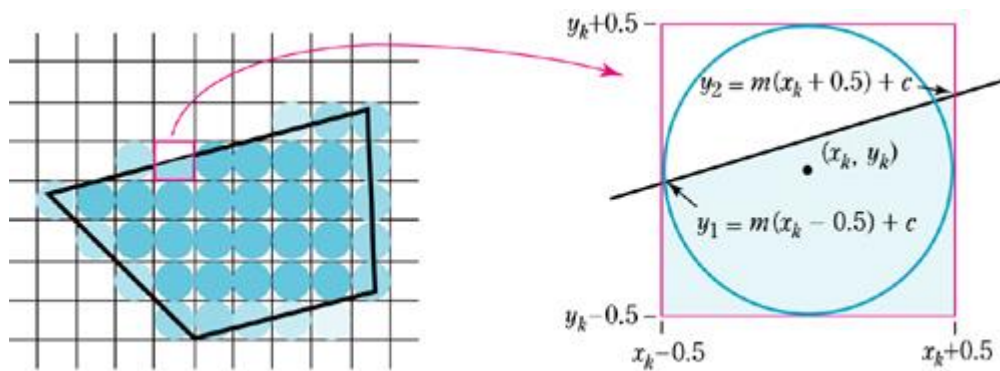
# *Antialiasing: 수퍼 샘플링 기법*



# Antialiasing: 영역 샘플링 기법

- Area sampling 기법

- 다각형 영역의 경계를 부드럽게 한다.
- 선이나 다각형의 테두리에 걸치는 픽셀이 내부영역에 얼마나 포함되는지 면적을 계산하여 비율 값을 사용하여 명암을 결정한다.
- 각 점을 등분하여 경계 안쪽에 있는 영역의 퍼센트 만큼 명암을 결정



- $y_1 = m(x_k - 0.5) + c$
- $y_2 = m(x_k + 0.5) + c$
- 면적 =  $\{ (y_1 - (y_k - 0.5)) + (y_2 - (y_k - 0.5)) \} / 2$   
=  $(y_1 + y_2) / 2 - y_k + 0.5$   
=  $mx_k + c - y_k + 0.5$

# *Antialiasing: 영역 샘플링 기법*

