Computer Graphics

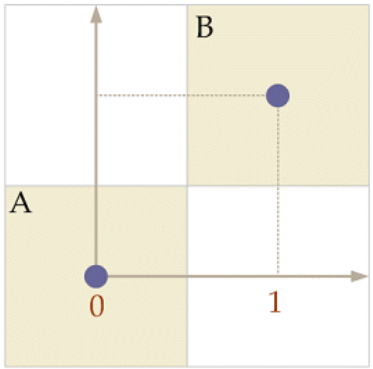
Prof. Jibum Kim

Department of Computer Science & Engineering

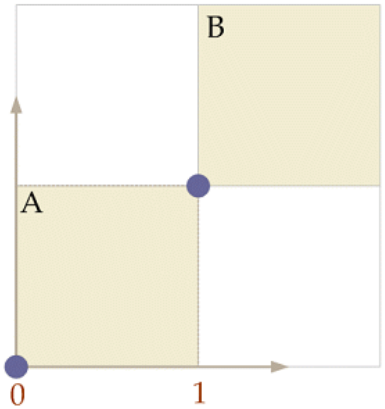
Incheon National University

- **화소 좌표**

- 각 pixel은 일정한 크기의 영역을 가지고 있다. 그렇다면 그 영역 중 어디를 기준으로 해당 pixel의 기준을 설정해야 하나?
- 화면 왼쪽 아래 4개의 pixel에 대해서 두 가지 방법을 생각해보자
- (a) Pixel의 center가 그 Pixel을 대표한다.
- (b) Pixel의 좌 하단 (bottom left)이 그 pixel을 대표 한다



(a)



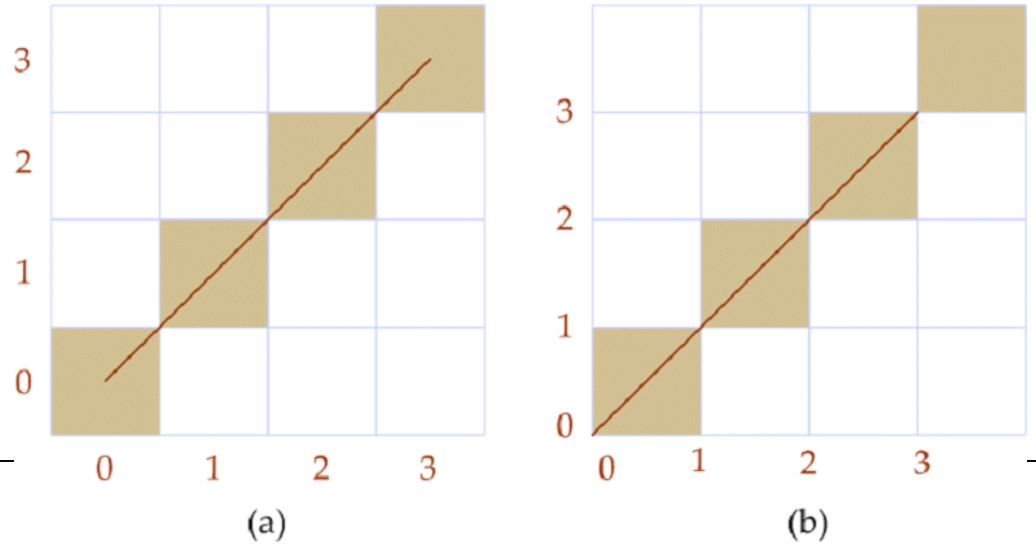
(b)

- 두 가지 방법에 의해 (0,0), (3,3)을 연결하는 red line을 pixel로 rasterize한 결과

(a) pixel 중앙에 coordinate 할당: 실제 선분보다 한 pixel 큰 선분이 된다

(b) pixel 왼쪽 하단에 coordinate 할당: 역시, 실제보다 한 pixel 큰 선분이 된다=> 주어진 선분의 외부에 할당한 pixel 제거

(a) 방법은 양쪽에 다 걸쳐있어서 특정 pixel만 제거하기 어렵다

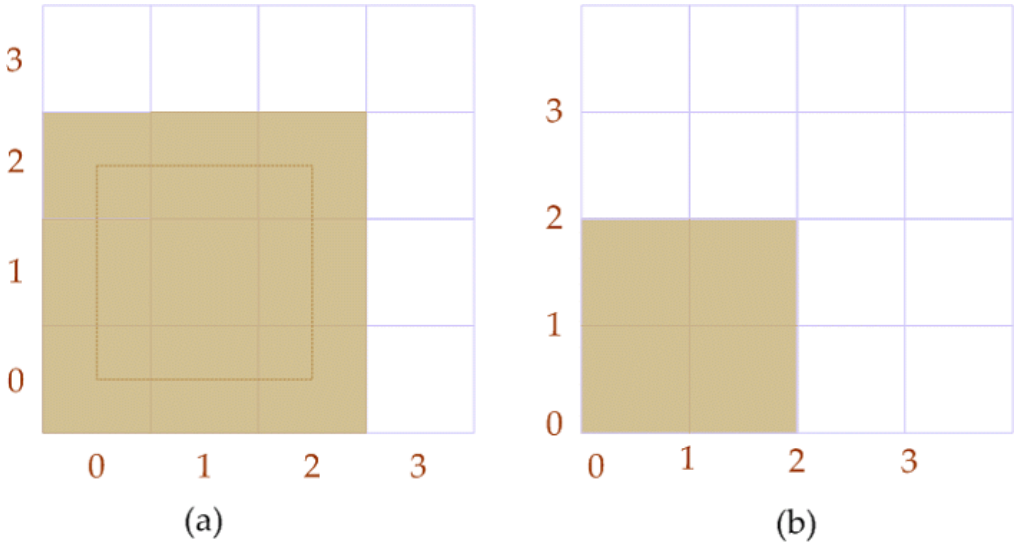


- 두 가지 method에 의해 (0,0), (2,0), (2,2), (0,2), 4 개의 vertex로 정의된 길이 2인 사각형을 rasterize한 결과

(a) pixel 중앙에 coordinate 할당: 실제보다 큰 사각형이 된다

(b) Pixel 왼쪽 하단에 coordinate 할당: 실제보다 큰 사각형이 된다

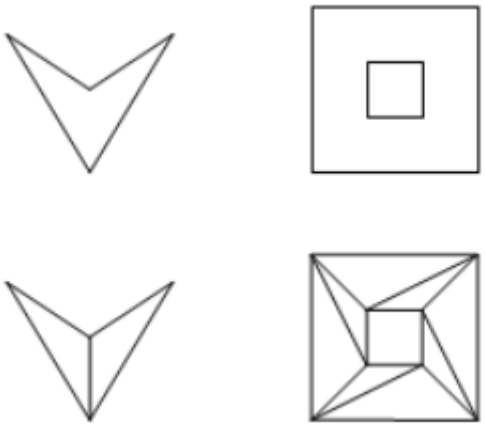
단, 범위가 벗어난 부분 제거 시 길이 2인 사각형 그릴 수 있음



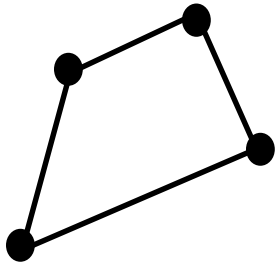
- OpenGL, DirectX 등 대부분의 그래픽스 관련 프로그램에서는 pixel의 좌하단 (bottom left)를 기준으로 pixel을 나타낸다
- 단, 선분길이 및 면적 조정을 위해 마지막 pixel은 제외시킴

- **Tessellation and Convex polygon rasterization**

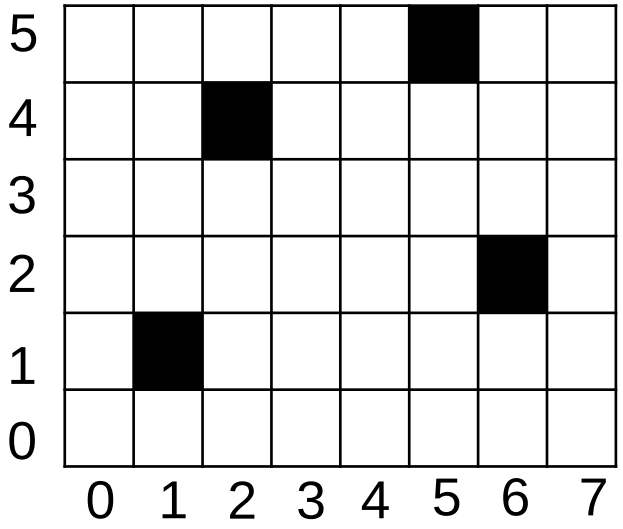
- Polygon rasterization시에 convex polygon이 아닌 경우에는 polygon을 분해 (decompose)해서 convex polygon으로 나누면 된다
- 이를 통해 자동적으로 polygon을 decompose해서 잘게 나뉘어진 삼각형들을 생성하는 (automatic triangulation)을 tessellation을 이용할 수 있다
- Tessellation: manage datasets of polygons presenting objects and divide them into suitable structures for rendering
- 최신 OpenGL 버전 (4.1부터), GLU library에서는 tessellation 제공

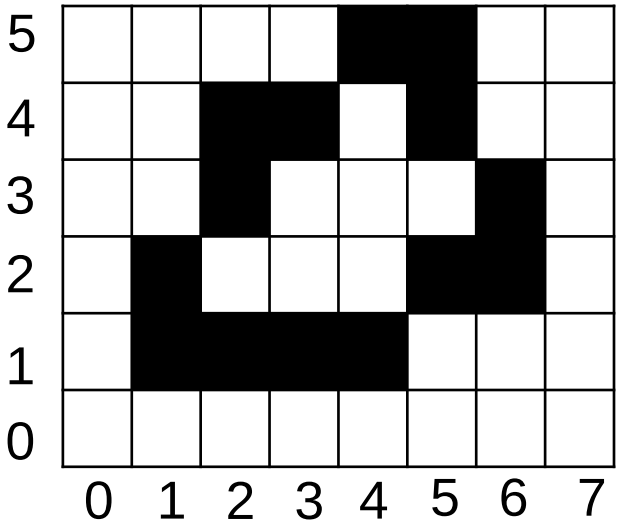


- **Old style (convex) polygon rasterization**
- **1. Rasterize edges first, Line rasterization (예: 브레스넘 알고리즘)**

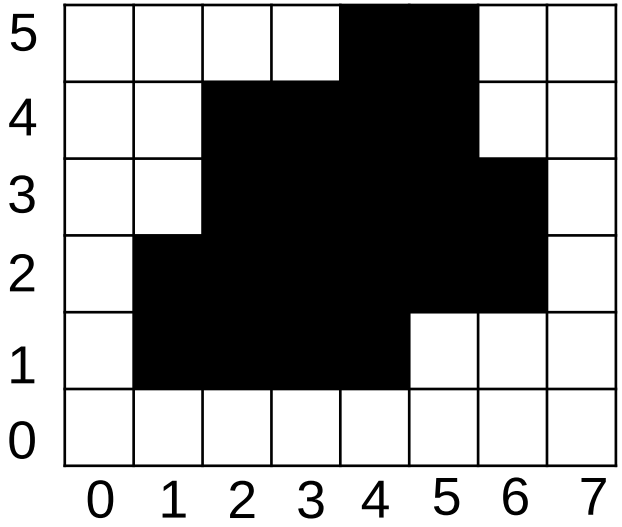


Polygon





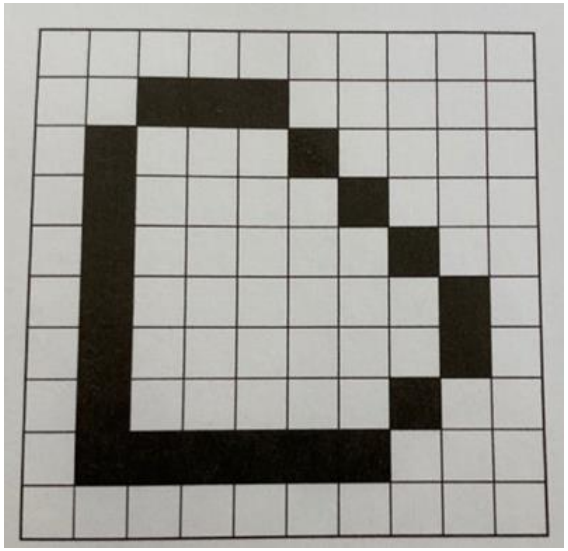
- **2. Polygon filling**
- **Edge 내부 (polygon 내부)에 있는 pixel들만 채운다**
- **이와 같이 polygon filling시에는 어떠한 pixel이 polygon 내부에 있고 외부에 있는지 판단하는 것이 중요하다**



- 이외에 여러 가지 polygon rasterization 방법이 있다
- 가장 널리 사용되는 방법은 뒤에 배울 scanline fill 알고리즘 이다

- **Flood-fill algorithm**

- Polygon rasterization시에 앞에서와 같이 polygon을 이루는 vertex의 위치가 정해지고 polygon의 edge들을 Bresenham 알고리즘을 이용해 edge들이 rasterization이 끝났다고 하자
- 배경 (background)이 white 이고 물체 (foreground)의 색이 black이라고 하면 Edge에 의해 표현된 polygon은 아래와 같다



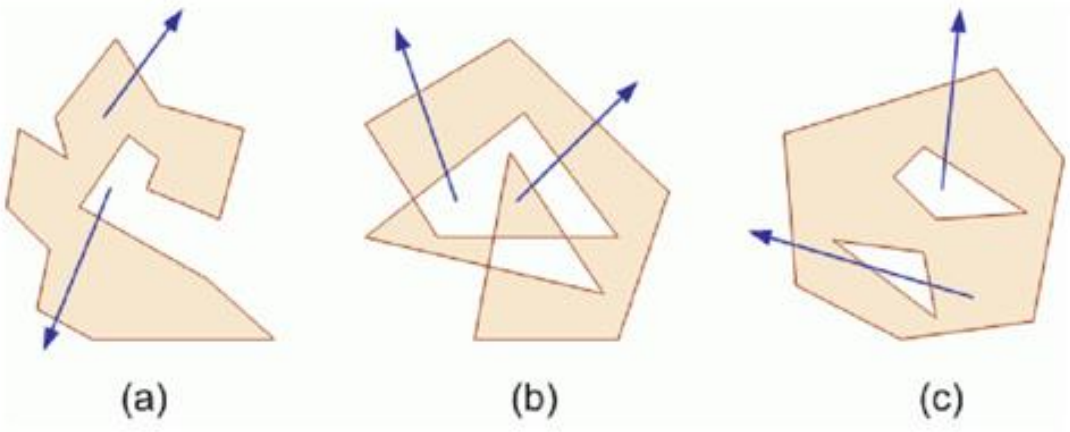
- 그 이후에 polygon 내부의 최소 시작점 (x, y) (이를 seed point라 함)를 찾을 수 있다면, 그 seed point로 부터 이웃을 recursive하게 반복하면서 물체색으로 색을 칠하면 된다. 이를 flood-fill 알고리즘이라 한다

```
function floodFill(x, y)
{
   if (readPixel (x, y) == WHITE)
  {
     writePixel(x, y, BLACK);
     floodFill(x-1, y);
     floodFill(x+1, y);
     floodFill(x, y-1);
     floodFill(x, y+1);
   }
}
```

- Flood fill 알고리즘 코드 예
- 실행후 마우크 왼쪽 클릭
- https://www.dropbox.com/s/uxdxvlgo96sbroj/floodfill.txt?dl=0

- **Inside-outside testing (odd-even) test**

- **Inside-outside testing (odd-even) test**
- **Suppose that p is a point inside a polygon.**
- **Any ray emanating from p and going off to infinity must cross an odd number of edges**
- **Any ray emanating from a point outside the polygon and entering the polygon crosses an even number of edges before reaching infinity**



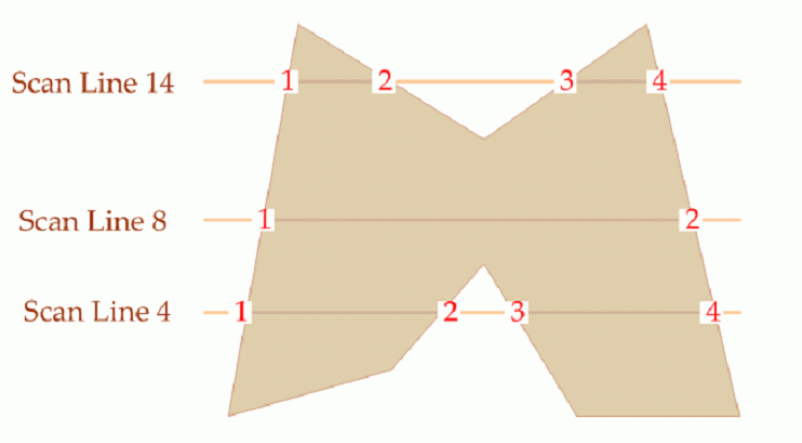
- **Scan line fill algorithm**

- polygon 내부의 색을 칠하는 또다른 방법은 주사선 채움 알고리즘 (scan line fill algorithm)을 이용하는 것이다
- Scanline: one row of pixels
- Scan line filling 알고리즘은 polygon을 rasterization 하는 방법으로 앞에서 배운 odd-even test를 사용 한다
- 화면 아래부터 row-by-row로 주사선 (scan line)과 다각형의 edge들과의 교차점을 계산한다. 각 주사선에서 홀수 번째의 교차점부터 짝수 째의 교차점 직전 구간에 있는 화소들을 모두 칠한다. 이를 통해 다각형 내부만 골라서 칠함

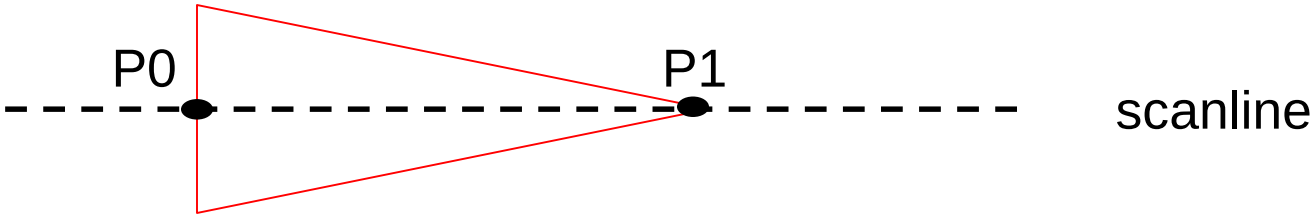
- **Scanline fill algorithm의 Pseudo-code**

```
for (each scanline L)
{
   Find intersections of L with all edges of P
   Sort the intersections by increasing x-value
   Fill pixels runs between all pairs of intersections
}
```

- **예: scanline 4는 polygon의 edge와 4번 intersection 생김**
- **교차점을 왼쪽부터 1, 2, 3, 4라고 하면 1-2 칠함, 3-4 칠함**



- **Scanline fill 알고리즘의 예외 처리 (singularity 문제)**
- **어떤 scanline이 동시에 polygon의 여러 edge와 교차할 수 있다. P1은 scanline과 2번 교차한다**
- **만일 2번 교차한 걸로 하면 어떤 문제가 생기나?**



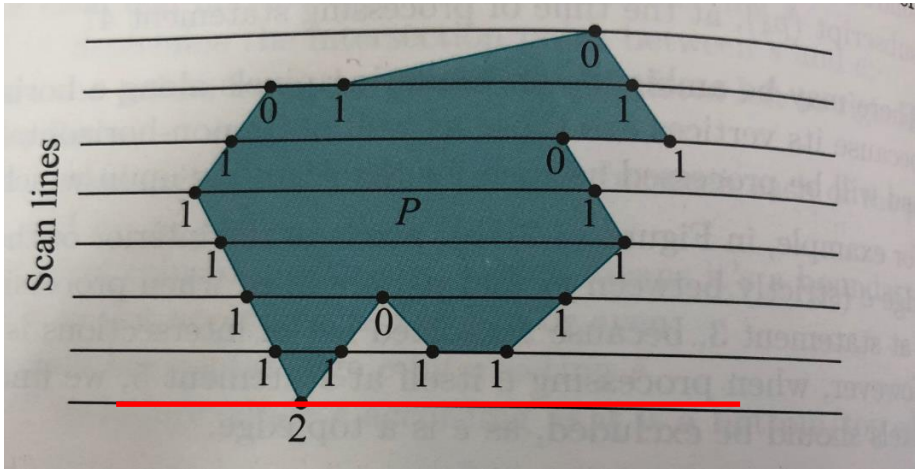
- **1번교차 – 2번 교차까지 칠해짐 , 3번 교차부터 칠해짐 -**

- **Scanline filling 알고리즘의 예외 처리**
- **앞의 scanline filling 알고리즘의 수정 (추가)**

Find the intersections of the scanline with all edges of P
Discard intersections with horizontal edges and with the upper endpoint of any edge

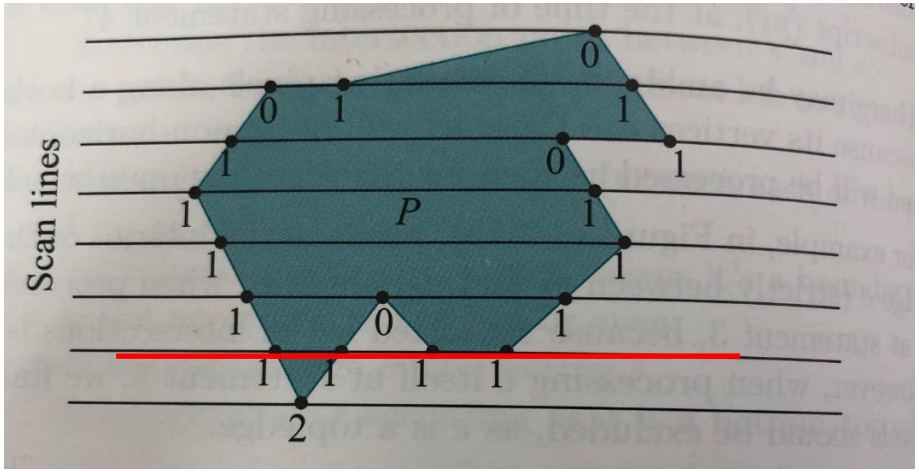
```
for (each scanline L)
{
   Find intersections of L with all edges of P
   Discard intersections with horizontal edges and with the  upper endpoint of any edge
   Sort the intersections by increasing x-value
   Fill pixels runs between all pairs of intersections
}
```

- 예: scanline과 polygon의 edge와의 교차 횟수 (교차점 개수) 표시



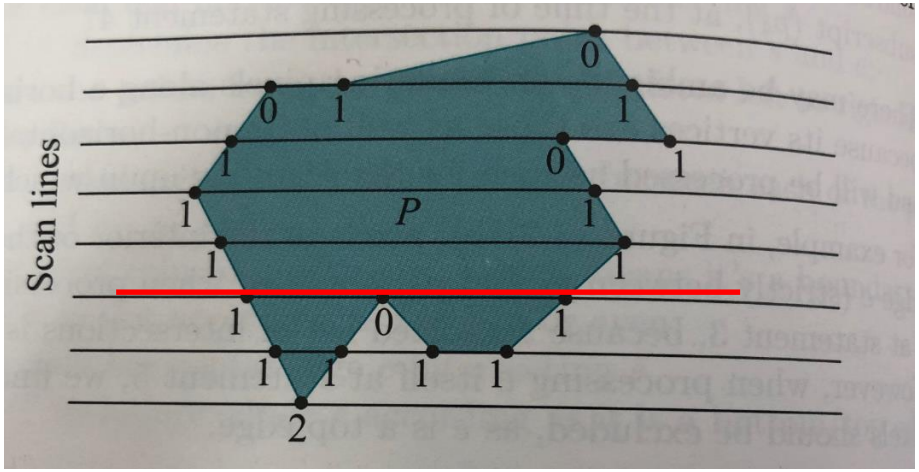
- 1. 제일 아래 scanline: polygon edge와 2번 교차, 교차점 p1
- 즉 p1 하나 칠해짐, output: {P1}

- 예: scanline과 polygon의 edge와의 교차 횟수 (교차점 개수) 표시



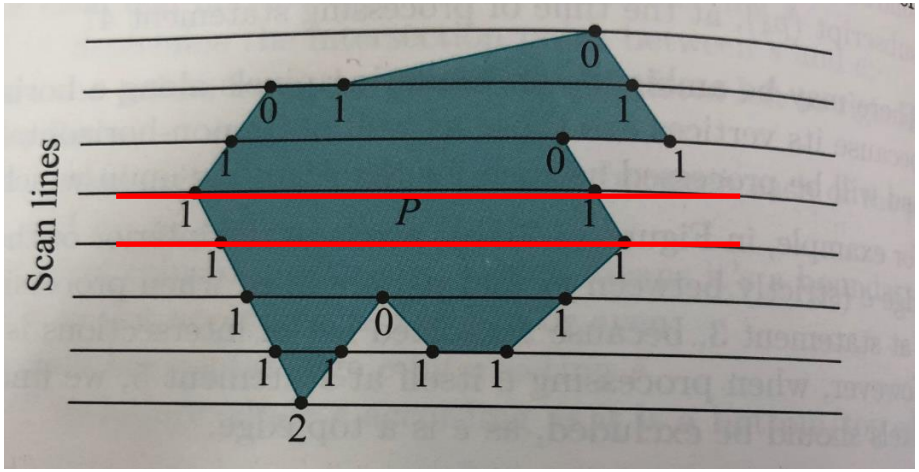
- 2. 2번째 scanline과의 교차점을 왼쪽 부터 p1,p2,p3,p4
- output: {p1, p2, p3, p4}
- p1-p2까지 칠해짐, p3-p4까지 칠해짐

- 예: scanline과 polygon의 edge와의 교차 횟수 (교차점 개수) 표시



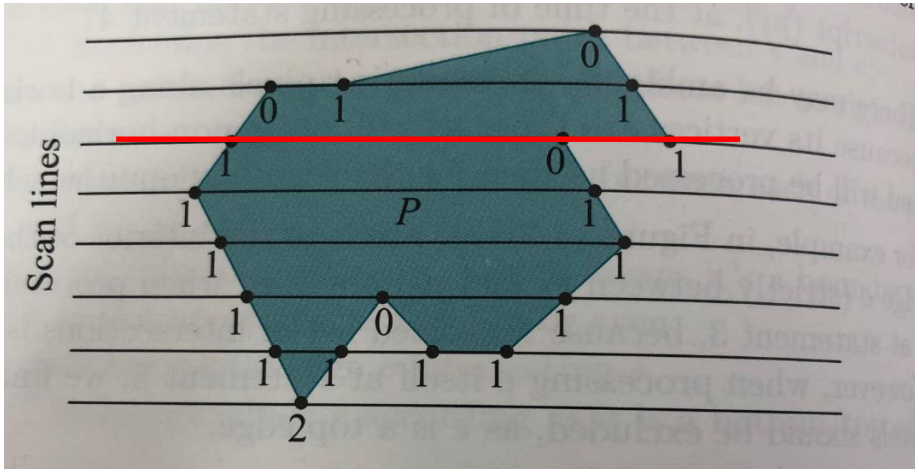
- 3. 3번째 scanline과의 교차점을 왼쪽 부터 p1,p2,p3
- p2는 0 intersections (why?), output{p1, p3}
- p1부터 p3까지 칠해짐

- 예: scanline과 polygon의 edge와의 교차 횟수 (교차점 개수) 표시



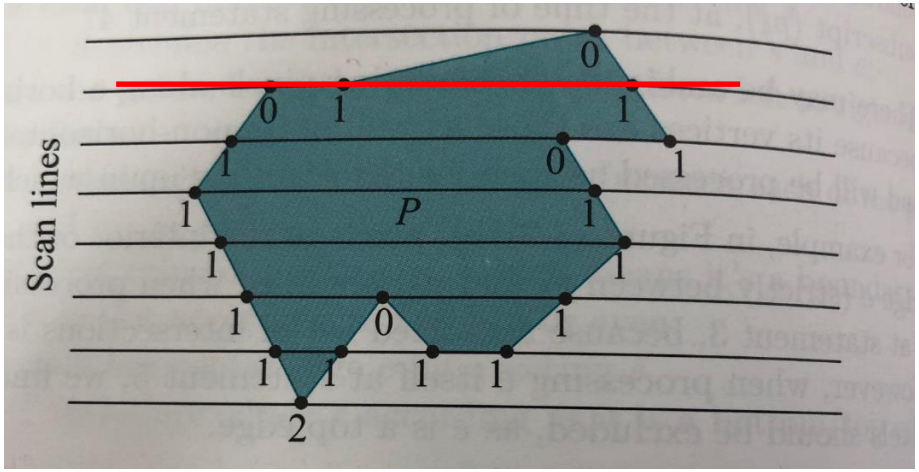
- 4. 4번째 5번째 scanline과의 교차점을 왼쪽 부터 p1, p2
- output: {p1, p2}
- P1부터 p2까지 칠해짐

- 예: scanline과 polygon의 edge와의 교차 횟수 (교차점 개수) 표시



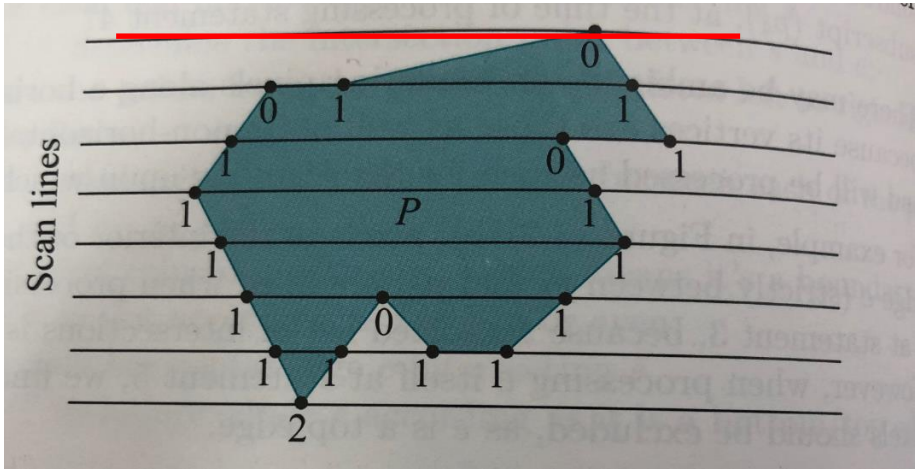
- 5. 6번째 scanline과의 교차점을 왼쪽 부터 p1,p2,p3
- P2는 0 intersections, output: {P1, P3}
- P1부터 P3까지 칠해짐

- 예: scanline과 polygon의 edge와의 교차 횟수 (교차점 개수) 표시



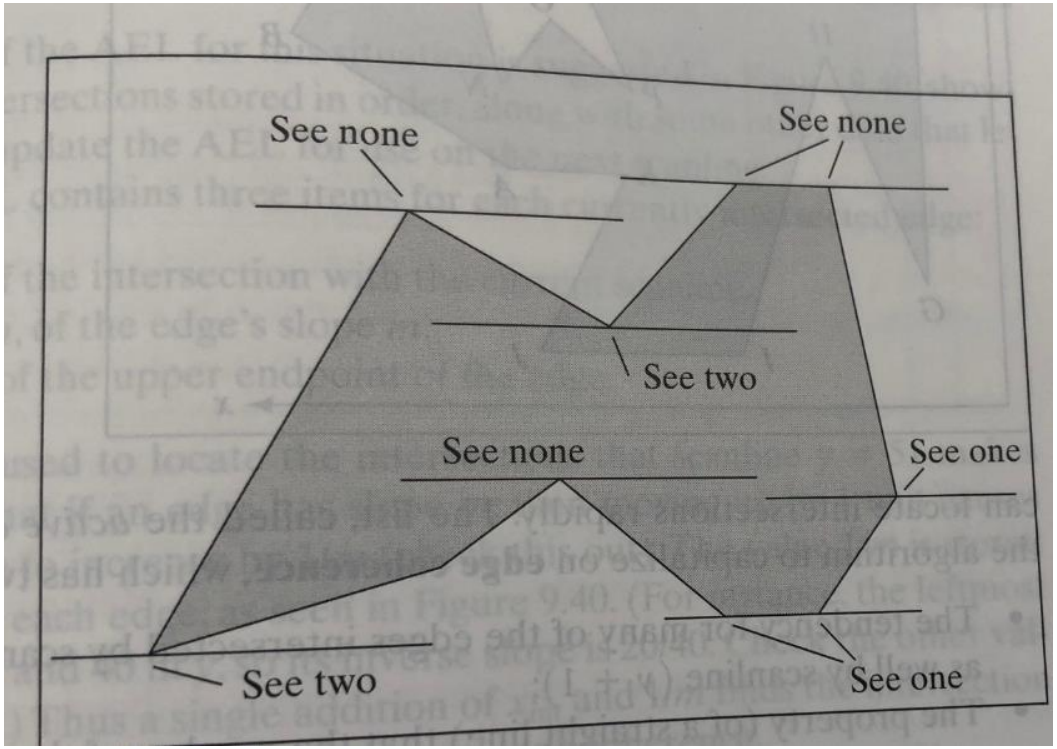
- 6. 7번째 scanline과의 교차점을 왼쪽 부터 p1,p2,p3
- P1은 0 intersections, output: {p2, p3}
- P2부터 P3까지 칠해짐

- 예: scanline과 polygon의 edge와의 교차 횟수 (교차점 개수) 표시



- 7. 8번째 scanline과의 교차점을 왼쪽 부터 p1
- P1은 0 intersections, output: none
- 칠해지지 않음

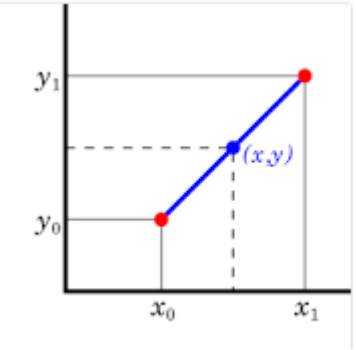
- 다음은 polygon의 끝점들에서 scanline과 polygon의 edge들과의 교차 횟수를 표시한 것이다. 맞는지 확인해 보자



- **Scanfill 알고리즘 OpenGL 코드 github**
- **실행후 OpenGL window에서 마우스 오른쪽 클릭후 scanfill 마우스 왼쪽 클릭**
- **https://www.dropbox.com/s/t9wwjp9m8alga3n/scanfill.txt?dl=0**

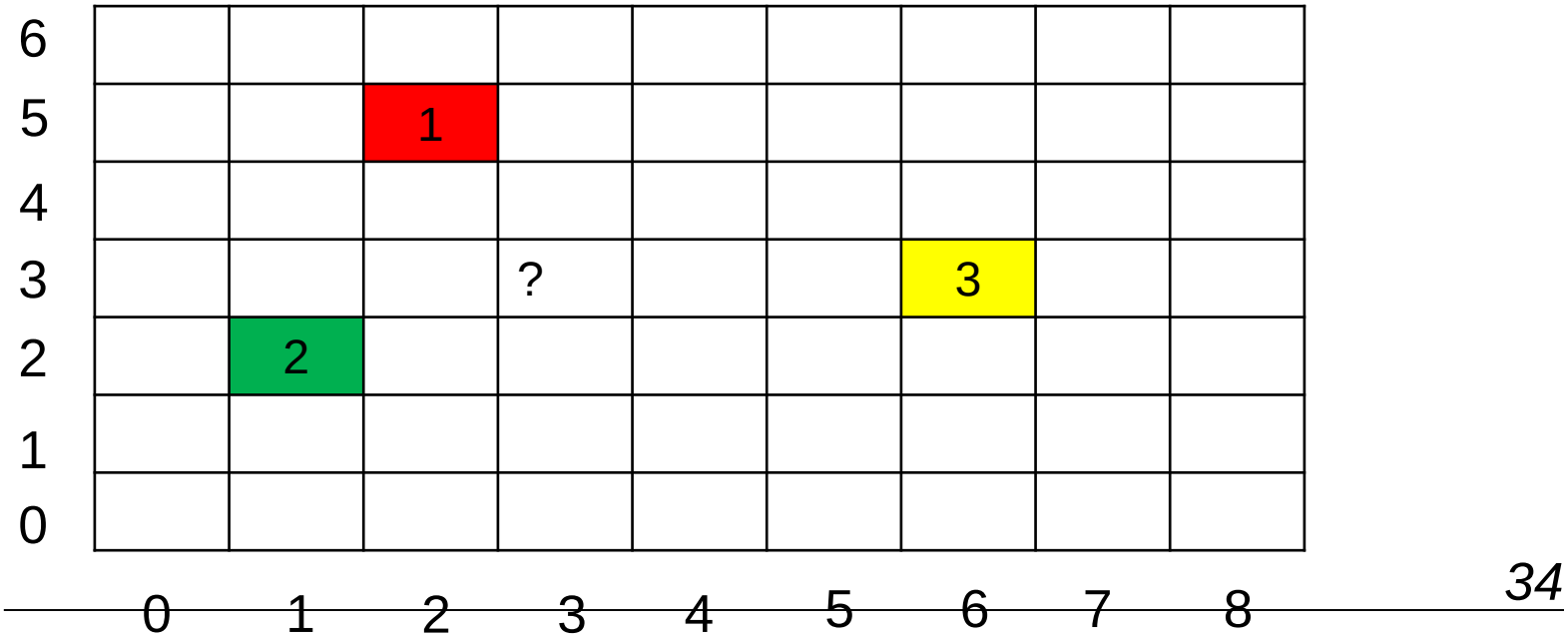
- **보간법 (interpolation)**

- 보간법 (interpolation)이란 원래 의미로 간격을 메꾼다는 의미로 어떤 데이터 들이 주어져 있을때 그 사이 데이터 값을 예측하기 위한 방법들을 의미한다
- 예: (x_0, y_o), (x_1, y_1)값이 주어짐
- (x, y)값 예측



- 이러한 보간법은 그래픽스에서 색깔 보간에 쓰일 수 있다

- Rasterization시 Color interpolation의 예
- 아래 Polygon, 즉, 삼각형을 이루는 세 개의 vertex 색이 다음과 같이 주어져 있을 때 아래 삼각형 내부의 픽셀 ?에 해당하는 색을 예측해 보자
- V1: (R,G,B)=(1, 0, 0), Viewport 좌표 (2, 5)
- V2: (R,G,B)=(0, 1, 0), Viewport 좌표 (1, 2)
- V3: (R,G,B)=(1, 1, 0), Viewport 좌표 (6, 3)
- ?, Viewport 좌표 (3, 3), (R, G, B)=(?,?,?)

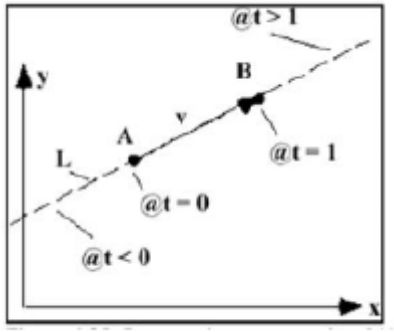


- 색깔 보간 (후에는 법선 벡터 보간)시에는 새로운 좌표인 무게 중심 좌표 (barycentric coordinates) 라는 개념을 사용한다

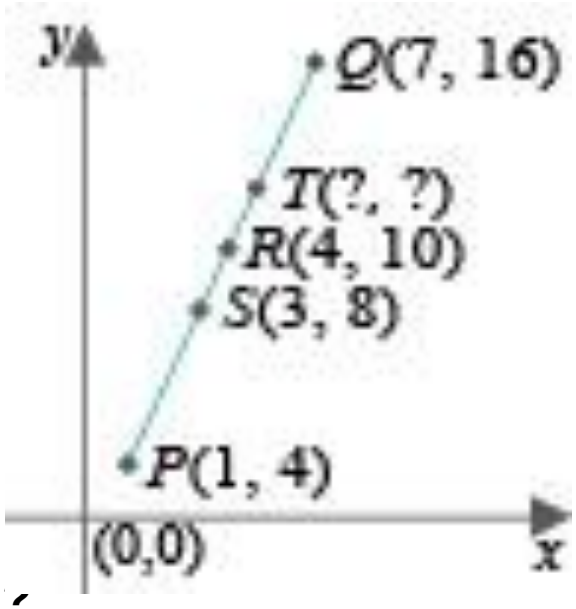
- 보간법에 대해서 알아보기 위하여 먼저 무게 중심 좌표
- 에 대하여 공부해 보자

- **Barycentric coordinates (1D)**

- Revisit parametric representation of a line segment (선분) using a parameter t
- x(t)=A+*v*t, *v*=B-A, $0 \leq t \leq 1$
- x(t)=A+(B-A)t, $0 \leq t \leq 1$
- x(t)=(1-t)A+tB, $0 \leq t \leq 1$
- 여기서 t를 선분 내에서의 가중치 (weight)라고도 표현한다



- 두 점 P(1, 4)와 Q(7, 16)으로 이루어진 선분에 대해서 생각해 보자
- 이 선분 사이의 임의의 점 x 는 다음과 같은 식으로 표현 가능한가?
- $x(t)=(1-t)P+tQ,\ 0 \le t \le 1$



- 예: t=1이면 어느 점인가? Q
- 예: t=0이면 어느 점인가? P
- 예: t=0.5이면 어느 점인가? R
- 예: t=1/3 이면 어느 점인가? S

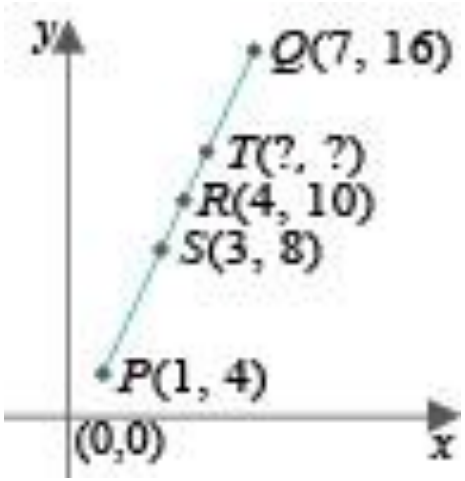
- Q) $0 \le t < 0.5$이면 오른쪽 선분에서 어디를 의미?

- 두 점 P와 Q로 이루어진 선분이 있을 때 이 두 점 사이에 있는 선분 위의 모든 점은 다음과 같이 표현 가능하다
- x= α *P + β*Q
- (단, $\alpha + \beta = 1$, $0 \leq \alpha, \beta \leq 1$)

- 이때 (α, β)를 무게 중심 좌표, barycentric coordinates라고 한다
- 점 x가 P에 가까워 질 수록 α는 1에 가까워 짐
- 점 x가 Q에 가까워 질 수록 β는 1에 가까워 짐

- P와 Q로 이루어 지는 선분위의 점 x가 아래 식과 같이 표현 가능
- x= α *P + β*Q (단, $\alpha + \beta = 1$, $0 \leq \alpha, \beta \leq 1$)

- 이를 색깔 보간에 이용해 보자. P와 Q의 색이 주어져 있고 선분 위의
- 점 x의 색을 보간하고 싶다면 이 α와 β를 P와 Q의 색이 섞이는 비율
- (가중치)로 이해할 수 있다
- P: α=1, β=0, P의 색만 사용
- Q: α=0, β=1, Q의 색만 사용
- S: α=2/3, β =1/3, P의 색 2/3 + Q의 색 1/3 섞음

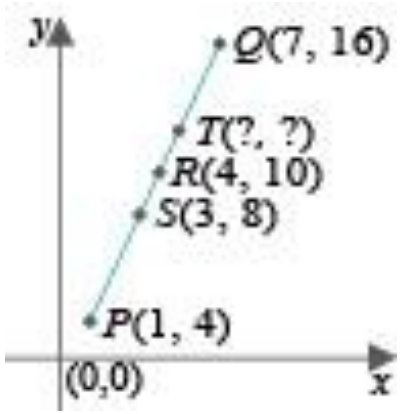


- 1D에서의 선분 사이의 특정 위치에서의 색 보간
- P의 색 (R1, G1, B1), Q의 색 (R2, G2, B2)

- X의 무게 중심 좌표가 (α, β)라면
- X= α *P + β*Q (단, $\alpha + \beta = 1$, $0 \leq \alpha, \beta \leq 1$)
- X의 색 보간: α*(R1, G1, B1) + β*(R2, G2, B2)

- 예: P(1, 4)의 색이 (R, G, B)=(1.0, 1.0, 0.0)이고
- Q(7, 16)의 색이 (R, G, B)=(1.0, 0.0, 0.0)이다
- S의 무게 중심 좌표: (α, β)=(2/3, 1/3) 일때 S에서의 색을 보간해 보자

- S의 색 = 2/3*(1.0, 1.0, 0.0) + 1/3*(1.0, 0.0, 0.0)=(1.0, 2/3, 0.0)



- 그렇다면 선분 위의 특정 위치에서의 무게 중심 좌표는 어떻게 구할까?
- 무게 중심 좌표를 구하는 방법은 여러 가지가 있다
- 예:두 점 P(1,4)와 Q(7,16)이 주어져 있다. 이 선분 위의 점 S(3, 8)을 무게 중심 좌표로 나타내 보자
- S= α *P + β*Q (단, $\alpha + \beta = 1$, $0 \leq \alpha, \beta \leq 1$)

- α=2/3, β=1/3