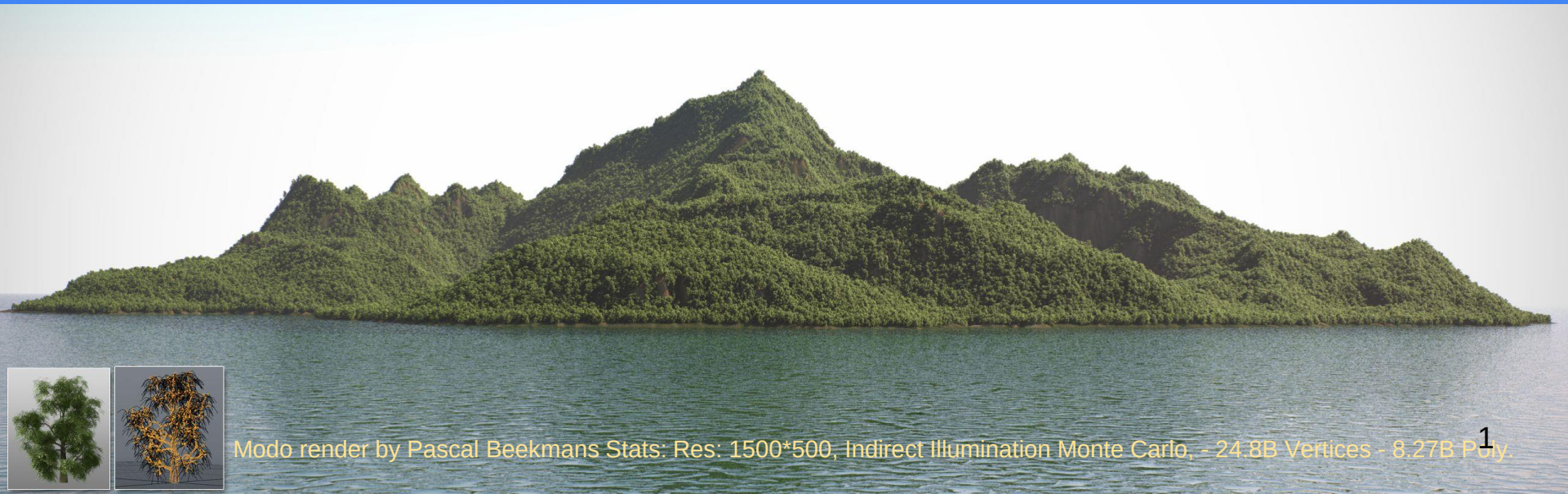
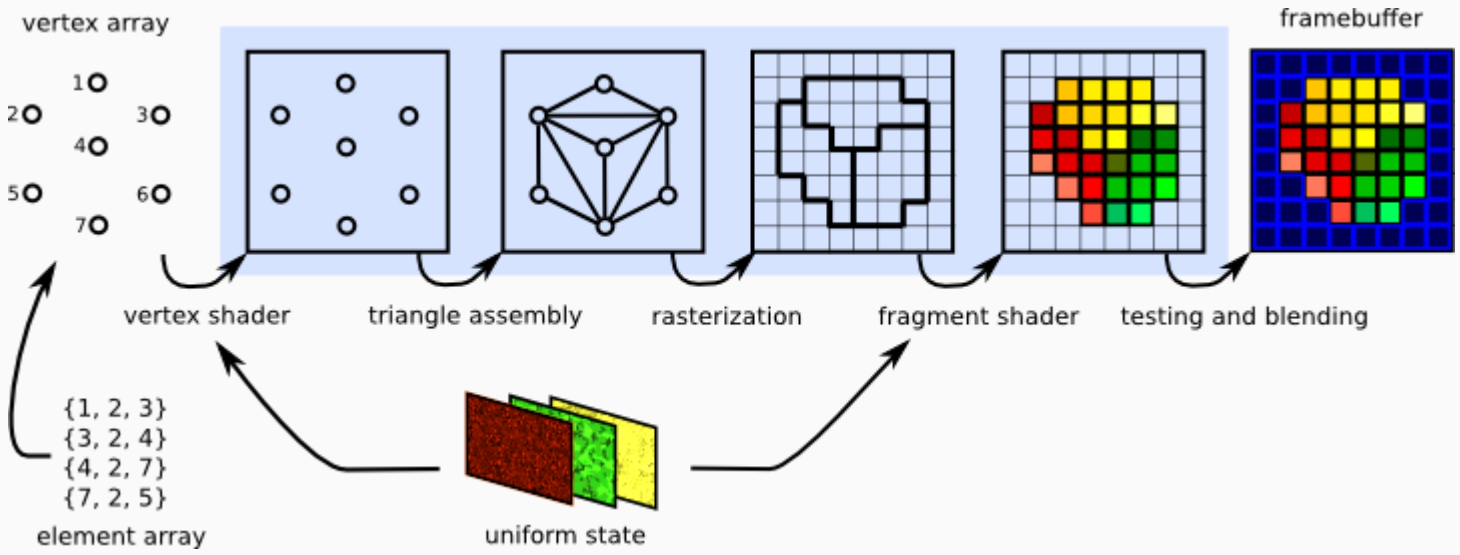
CS113 - ĐỒ HỌA MÁY TÍNH VÀ XỬ LÝ ẢNH

Rasterization – Phần 2 : Region Filling



Quy trình hiển thị

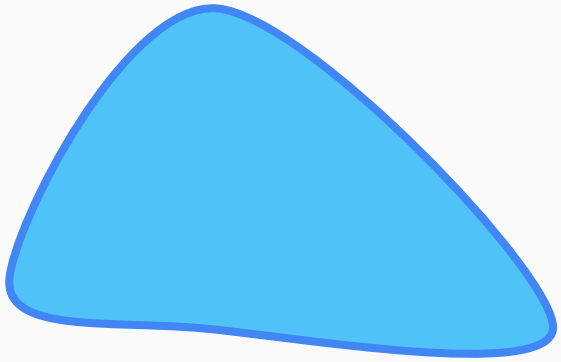


- Trong bước rời rạc hóa (Rasterization), ta cần xác định các điểm trên lưới pixel (pixel-grid) tương ứng với đối tượng cần được hiển thị.

Khái niệm về Vùng tô

Có hai loại vùng:

- Vùng được xác định bởi điểm ảnh – pixel-defined region
- Vùng xác định bởi đa giác – polygonal region



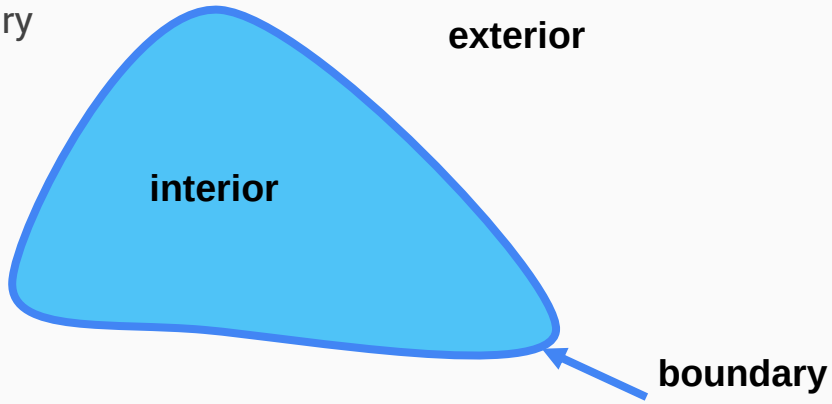
pixel-defined region

polygonal region

Vùng được xác định bởi điểm ảnh

Vùng được định nghĩa bởi màu của pixel, chia làm 3 phần:

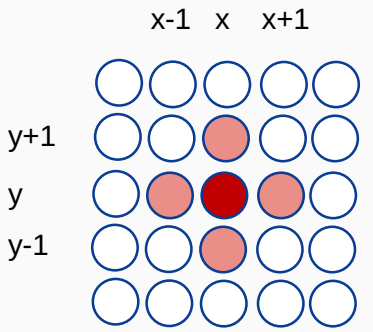
- Vùng trong – interior
- Vùng ngoài – exterior
- Biên (liên tục) - boundary



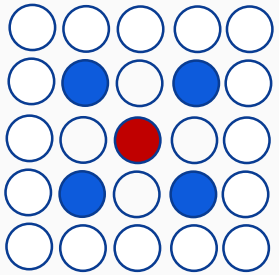
Vùng được xác định bởi điểm ảnh

Khái niệm về lân cận :

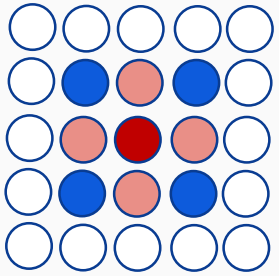
- Lận cận 4 (4- Adjacency) của một pixel p (x,y) là các pixel có tọa độ (kề nhau theo chiều ngang hay chiều dọc):
  - (x+1,y), (x-1,y), (x,y+1), (x,y-1)
  - Ký hiệu: $N_4(p)$.
- Lận cận đường chéo (diagonal Adjacency) của p là các pixel có tọa độ:
  - (x+1,y+1), (x+1,y-1), (x-1,y+1), (x-1,y-1)
  - Ký hiệu: $N_D(p)$.
- Lân cận 8 (8- Adjacency): là tập các pixel lân cận 4 và lân cận đường chéo.
  - Ký hiệu: $N_8(p)$.
  - $N_8(p)=N_4(p) \cup N_D(p)$



$N_4(p)$



$N_D(p)$

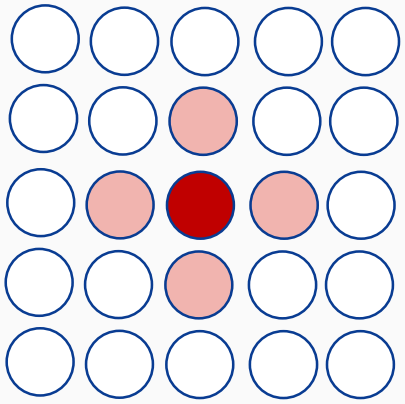


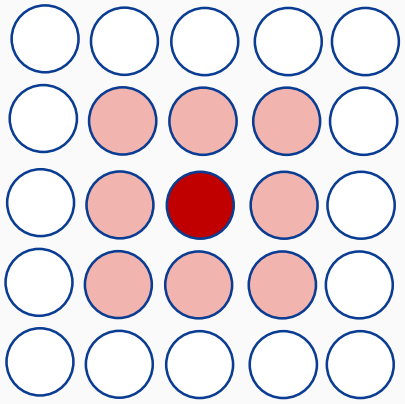
$N_8(p)$

Vùng được xác định bởi điểm ảnh

Khái niệm về liên thông:

- Liên thông 4 (4-connected): Ta nói pixel p liên thông 4 với q nếu q $\in N_4(p)$
- Liên thông 8 (8-connected): Ta nói pixel p liên thông 8 với q nếu q $\in N_8(p)$





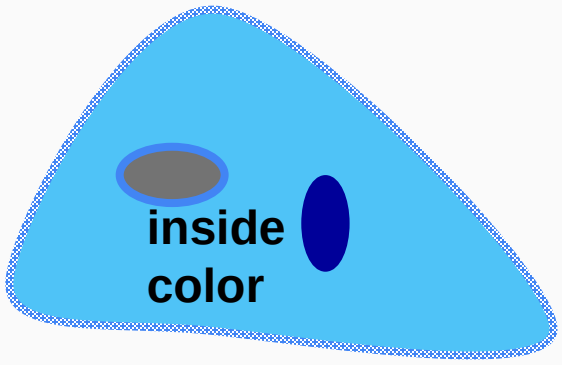
Vùng được xác định bởi điểm ảnh

Interior defined

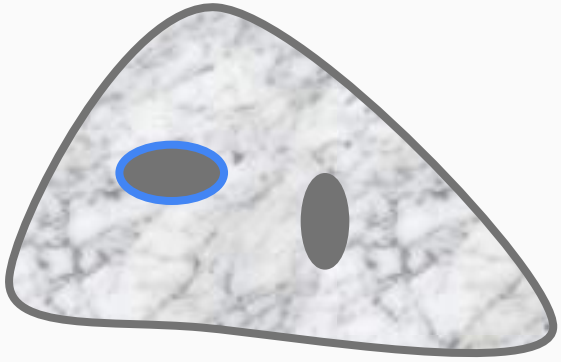
- Tất cả các pixel trong vùng có cùng một màu, gọi là inside-color
- Các pixel trên biên không có màu này
- Có thể có lỗ trong vùng

Boundary defined

- Các pixel thuộc biên có cùng màu – boundary-color
- Các pixel trong vùng không có màu này
- Nếu một số pixel trong vùng có màu boundary-color thì vùng sẽ chứa lỗ



Interior-defined



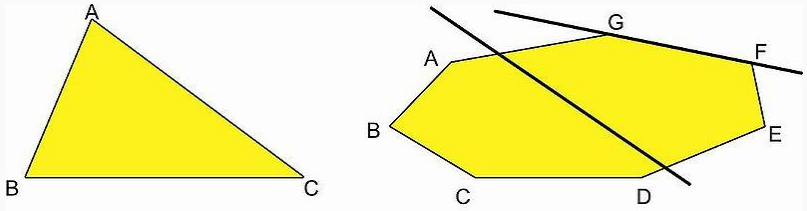
Boundary-defined

Polygonal Region

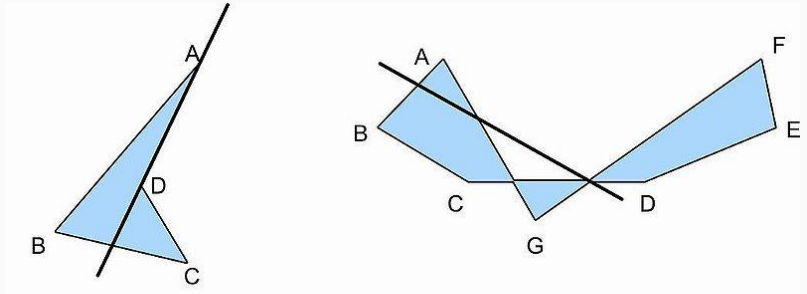
Định nghĩa bằng đa giác: xác định các định các đỉnh $p_i = (x_i, y_i)$

Các loại đa giác:

- Đa giá lồi – Convex polygon: toàn bộ đa giác nằm về một phía của đường thẳng chứa cạnh bất kỳ nào của đa giác
- Đa giác lõm (Concave polygon): đa giác nằm về hai phía của ít nhất một đường thẳng chứa cạnh nào đó.



Đa giác lồi-*Convex polygon*



Đa giác lõm đơn
Simple concave polygon

Đa giác lõm phức
Simple complex polygon

Đa giác lõm-*Concave polygon*

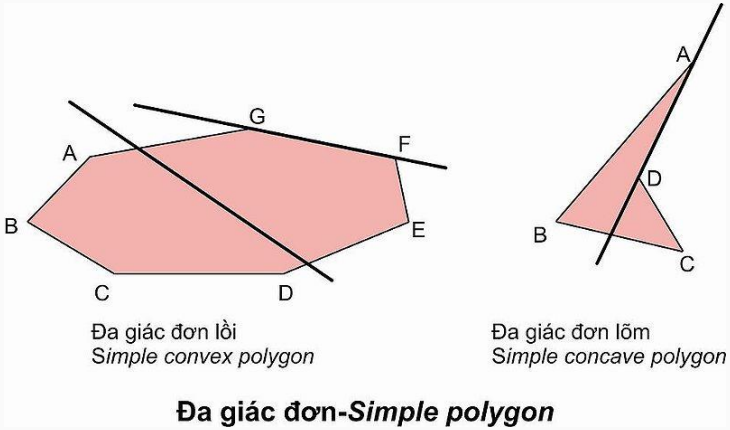
Nguồn: https://vi.wikipedia.org/wiki/Đa_giác

Polygonal Region

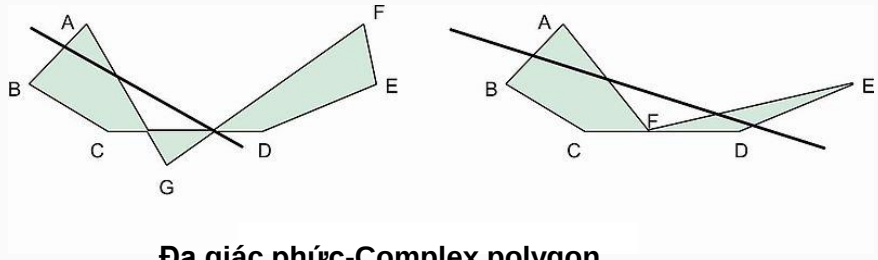
Định nghĩa bằng đa giác: xác định các định các đỉnh $p_i = (x_i, y_i)$

Các loại đa giác:

- Đa giác đơn (simple polygon): đa giác mà các cạnh chỉ có thể cắt nhau tại các đầu mút (đỉnh đa giác), không có hai cạnh không kề nhau cắt nhau.
- Đa giác không đơn (Nonsimple, đa giác phức-Complex polygon): đa giác có hai cạnh không kề nhau cắt nhau, điểm cắt nhau đó không phải là đỉnh của đa giác.



Đa giác đơn-*Simple polygon*



Đa giác phức-Complex polygon

Nguồn: https://vi.wikipedia.org/wiki/Đa_giác

Region Fill Algorithms

- Seed Fill Approaches
  - Boundary Fill
  - Flood Fill
    - Work at the pixel level.
    - Suitable for interactive painting applications
- Scanline Fill Approaches
  - Work at the polygon level
  - Better performance

Thuật toán Boundary-Fill

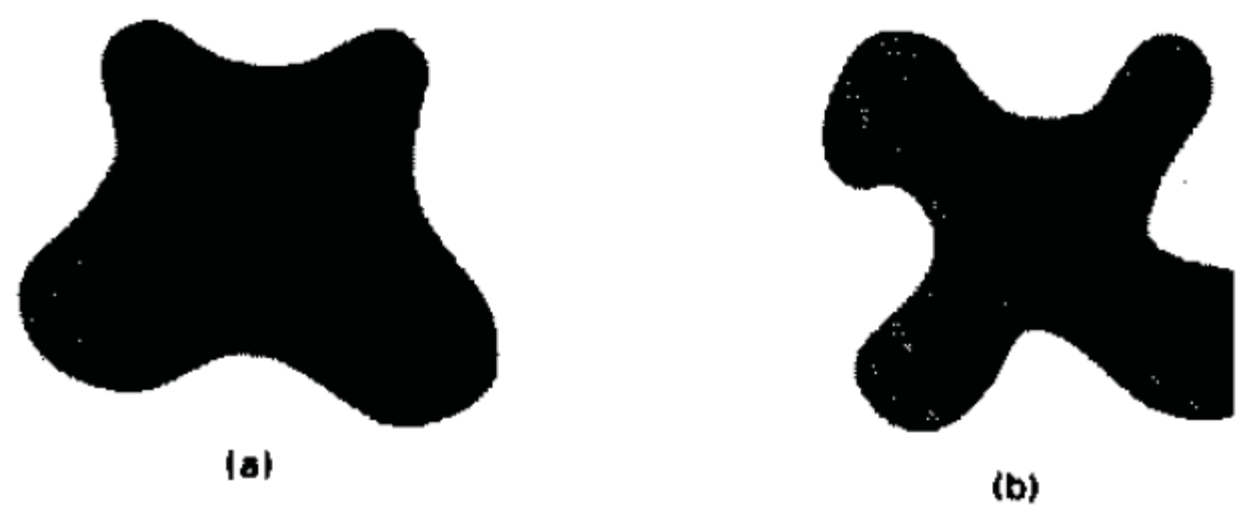
- Recursive algorithm that begins with a starting pixel, called a seed inside the region.
- The algorithm checks to see if this pixel is a boundary pixel or has already been filled.
- If the answer is no, it fills the pixel and makes a recursive call to itself using each and every neighboring pixel as a new seed.
- If the answer is yes, the algorithm simply returns to its caller.

Thuật toán Boundary-Fill

- A boundary-fill procedure accepts as input the coordinates of an interior point (x, y), a fill color, and a boundary color.
- Starting from (x, y), the procedure tests neighboring positions to determine whether they are of the boundary color. If not, they are painted with the fill color, and their neighbors are tested. This process continues until all pixels up to the boundary color for the area have been tested.
- Both inner and outer boundaries can be set up to specify an area.

Thuật toán Boundary-Fill

- Example color boundaries for a boundary-fill procedure:



Thuật toán Boundary-Fill

```
void boundaryFill4 (int x, int y, int fill, int boundary)
{
    int current;
    current = getpixel (x, y);
    if ((current != boundary) && (current != fill))
    {
        setcolor (fill);
        setpixel (x, y);
        boundaryFill4 (x+1, y, fill, boundary);
        boundaryFill4 (x−1, y, fill, boundary);
        boundaryFill4 (x, y+l, fill, boundary);
        boundaryFill4 (x, y−1, fill, boundary);
    }
}
```

Thuật toán Boundary-Fill

- Recursive boundary-fill algorithms may not fill regions correctly if some interior pixels are already displayed in the fill color.
- This occurs because the algorithm checks next pixels both for boundary color and for fill color.
- Encountering a pixel with the fill color can cause a recursive branch to terminate, leaving other interior pixels unfilled.
- To avoid this, we can first change the color of any interior pixels that are initially set to the fill color before applying the boundary-fill procedure.

Flood-Fill Algorithm

Begin with a seed(starting pixel) inside the region.

It checks to see if the pixel has the region’s original color.

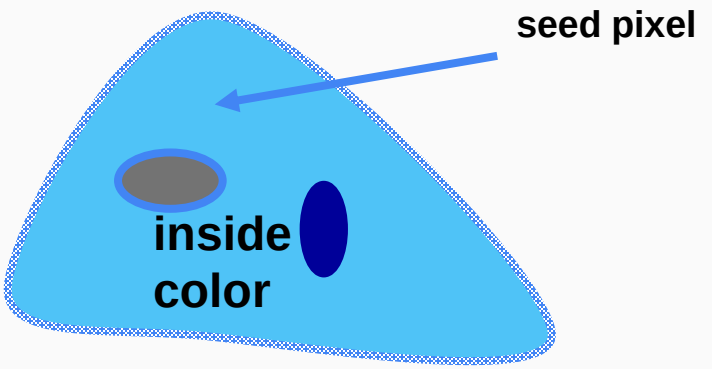
If the answer is yes, it fills the pixel with a new color and uses each of the pixel’s neighbors as anew seed in a recursive call.

If answer is no , it returns to the caller.

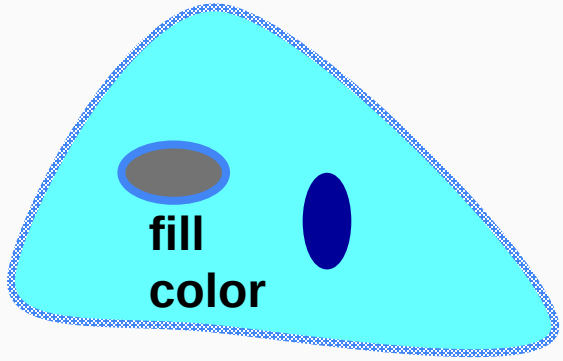
Thuật toán Flood Fill

Đổi màu của tất cả các interior-pixel thành màu tô – fill color.

Quá trình tô màu bắt đầu từ một điểm (seed pixel) thuộc phía trong vùng tô và lan truyền khắp vùng tô => Flood-Fill



Interior-defined



Recursive Flood-Fill

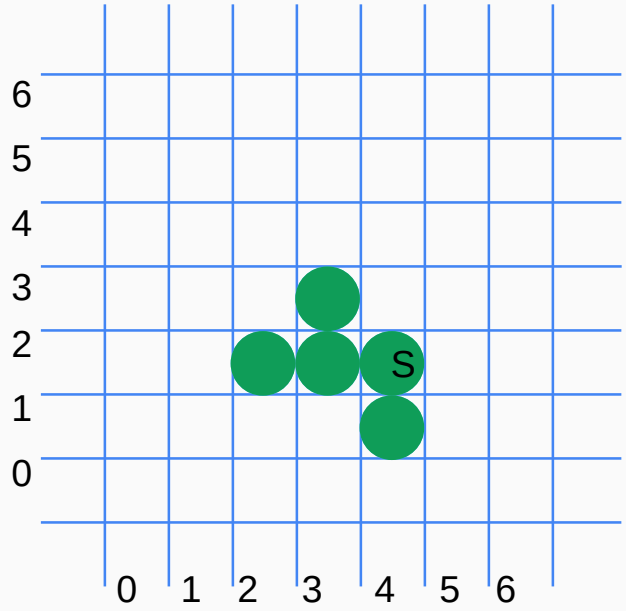
Recursive Flood-Fill Algorithm (cont)

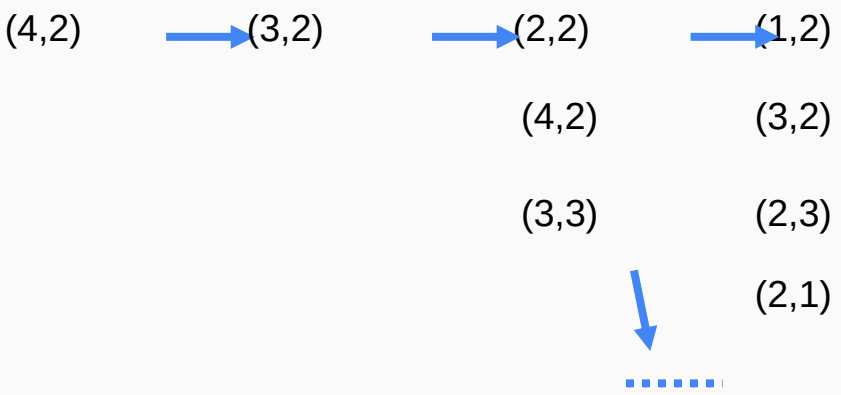
Thuật toán

Nếu pixel tại (x,y) thuộc vùng trong – màu của pixel đó là inside-color thì

- Đổi màu của nó thành fill-color
- Áp dụng quá trình trên cho 4 điểm lân cận nó (4-connected).

Ngược lại, không làm gì.





Recursive Flood-Fill Program

```
void FloodFill(int x, int y, int inside_color, int
fill_color)
{
   if (getpixel(x,y) == inside_color)
        {
              putpixel(x,y,fill_color);
              FloodFill(x-1,y, inside_color, fill_color);
              FloodFill(x+1,y, inside_color, fill_color);
              FloodFill(x,y+1, inside_color, fill_color);
              FloodFill(x,y-1, inside_color, fill_color);
        }
}
```

Thuật toán cải tiến – Dùng stack

Cho vào *stack* run chứa *seed pixel*

while *stack* not empty {

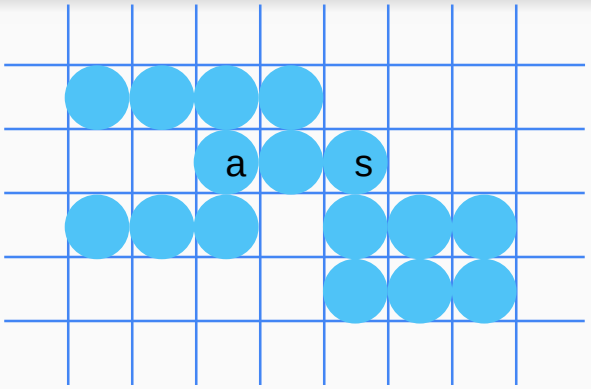
begin = pop();

Tô run bắt đầu từ *begin*

Cho vào *stack* các run ở bên trên

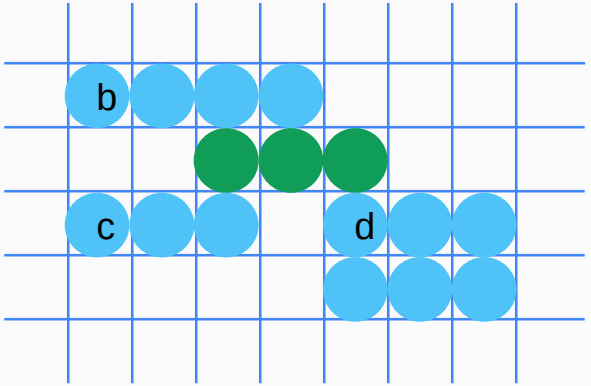
Cho vào *stack* các run ở bên dưới

}



Stack:

a

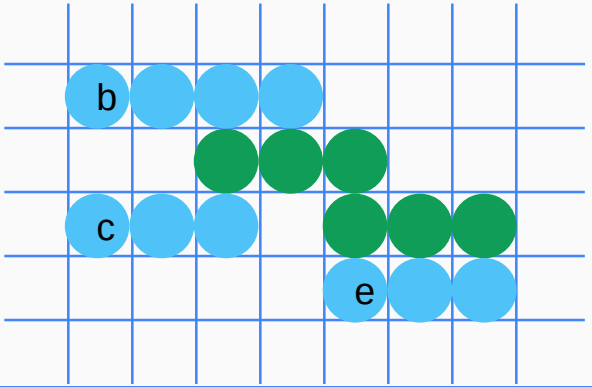


Stack:

b

c

d



Stack:

b

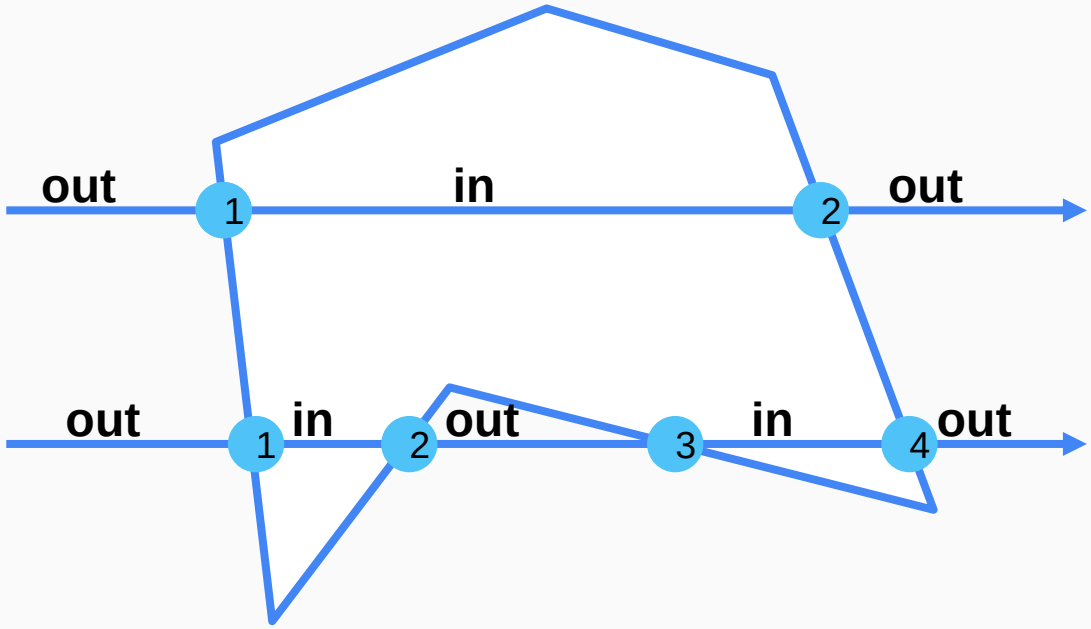
c

e

Polygonal Region – Scanline Algorithm

Scanline

- Đường thẳng nằm ngang
- Số giao điểm của scanline và đa giác là số chẵn (tổng quát)
- Các pixel nằm giữa các cặp giao điểm lẽ-chẵn nằm trong đa giác

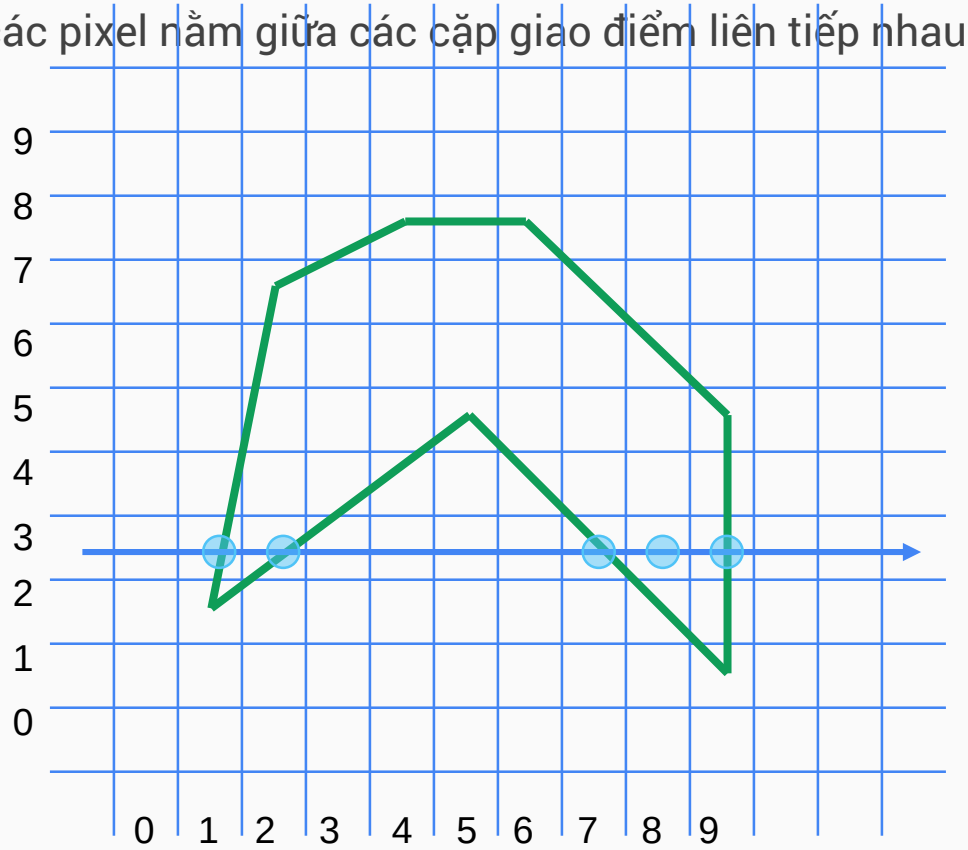


Thuật toán Scanline tổng quát

for each scanline {

- Tìm giao điểm của scanline với các cạnh của đa giác
- Sắp xếp các giao điểm theo thứ tự tăng dần theo x
- Tô các pixel nằm giữa các cặp giao điểm liên tiếp nhau

}

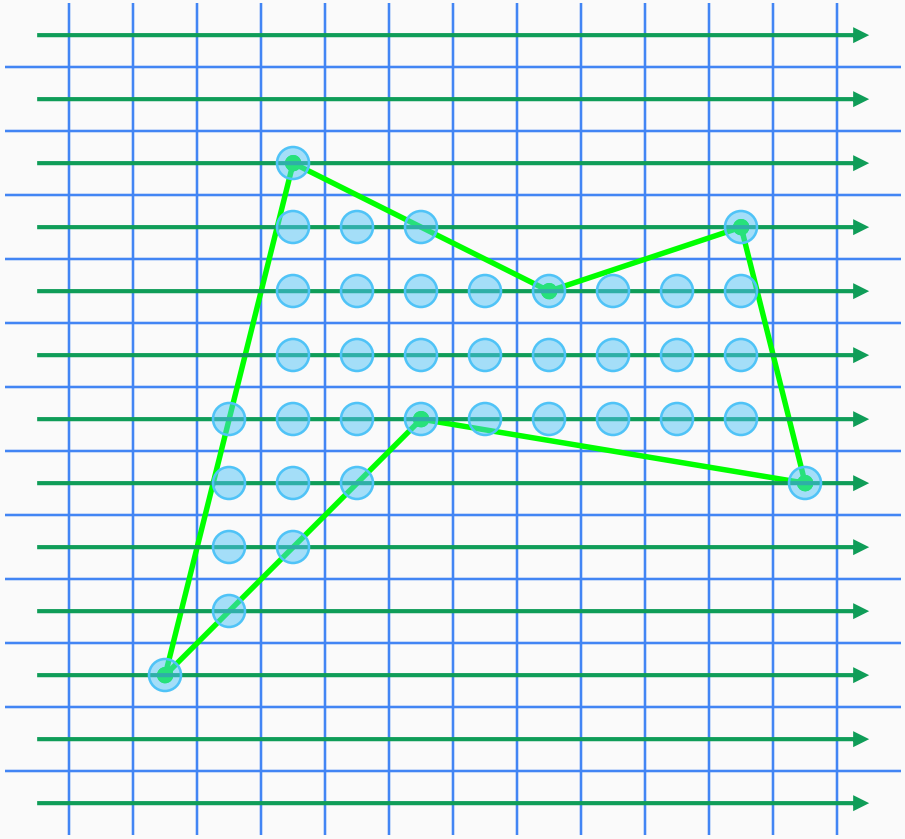


Tại dòng scanline y = 3:

Các hoành độ giao điểm sau khi làm tròn là 1, 2, 7, 9

Do đó, 2 run [1,2] và [7,9] được tô

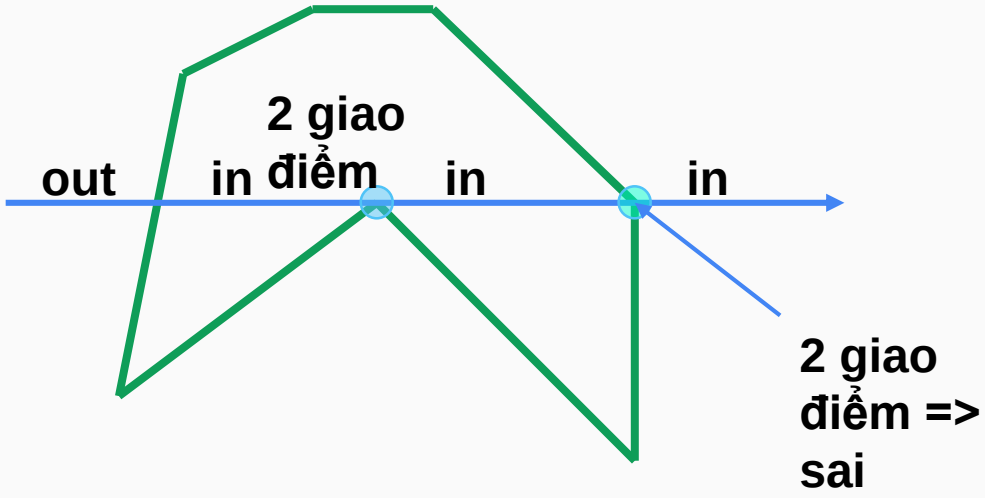
Demo



Các trường hợp đặc biệt

Các cạnh nằm ngang không xét đến vì chúng sẽ được tô khi xét 2 cạnh kề với nó

Khi scanline đi qua đỉnh của đa giác, nó sẽ giao với 2 cạnh. Trong trường hợp đỉnh không là cực trị, số giao điểm của scanline với đa giác là số lẽ.



Các trường hợp đặc biệt (cont)

y-extrema vertices:

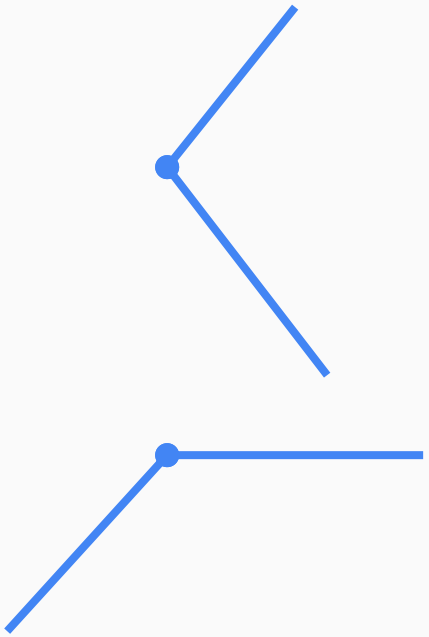
minimum

maximum

y-monotonic:

minimum với 1 cạnh

maximum với cạnh còn lại

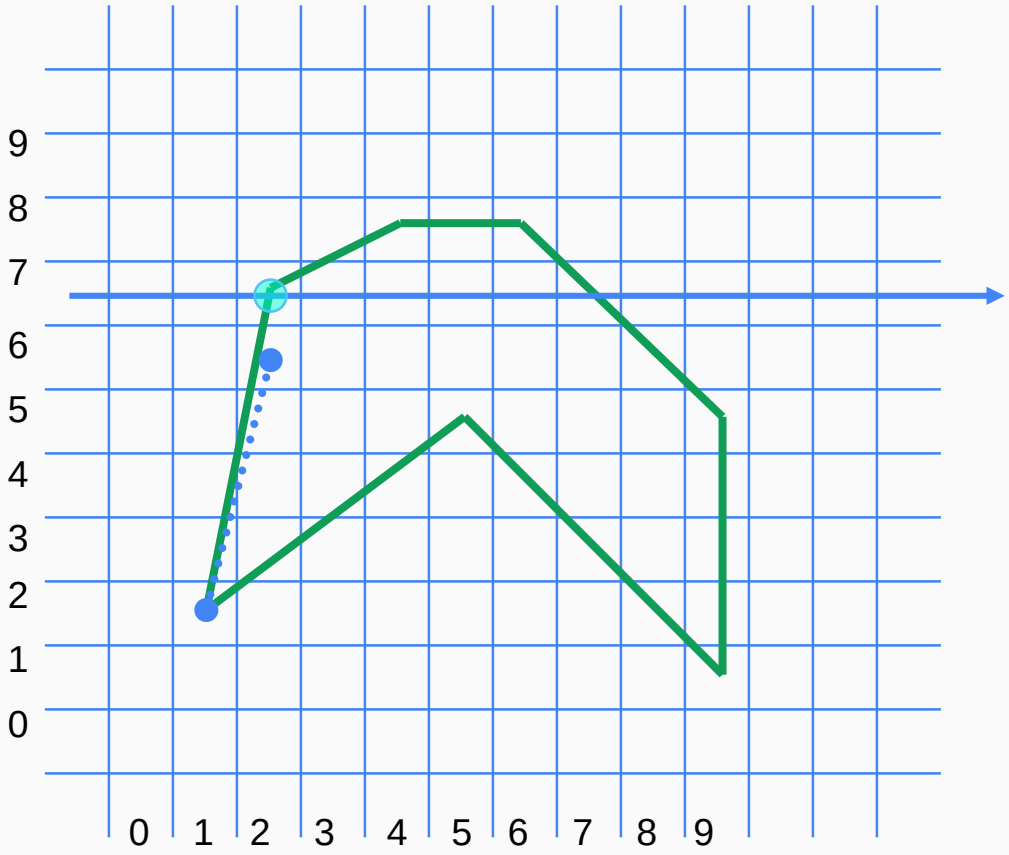


Xử lí

Trước quá trình tô màu, kiểm tra các đỉnh.

Nếu đỉnh không phải là cực trị, xét cạnh phía dưới.

Giảm tung độ trên y_upper xuống một đơn vị



Danh sách đỉnh đa giác trước khi cải tiến:

(6,8), (9,5), (9,1), (5,5), (1,2), (2,7), (4,8)

Sau khi cải tiến, danh sách các cạnh của đa giác như sau - một cạnh bị xóa và 2 cạnh được rút gọn:

e_1 = (6,8) to (9,5)

e_2 = (9,4) to (9,1)

e_3 = (9,1) to (5,5)

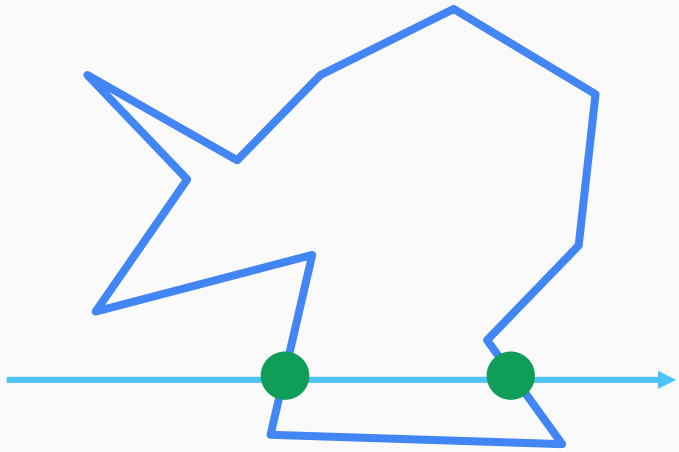
e_4 = (5,5) to 1,2)

e_5 = (1,2) to (2,6)

e_6 = (2,7) to (4,8)

Hạn chế của thuật toán

- Để xác định giao điểm của đường scanline và cạnh của đa giác, chúng ta phải duyệt tất cả các cạnh của đa giác.
- Khi số cạnh của đa giác khác lớn, chúng ta phải mất rất nhiều thời gian để duyệt hết các cạnh, trong khi số cạnh mà đường scanline cắt thì rất ít.



Số giao điểm là 2, trong khi số cạnh là 12

Cải tiến tốc độ thuật toán

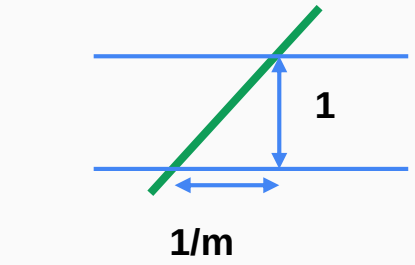
Nhận xét:

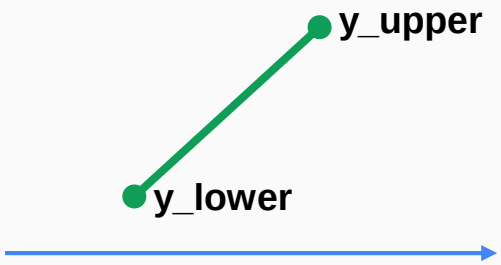
- Khi dòng quét tăng một đơn vị theo y thì hoành độ giao điểm thay đổi theo 1/m

→ Công thức tính giao điểm đơn giản

- Giả sử rằng 1 cạnh của đa giác có tung độ bị chặn bởi [y_lower, y_upper] thì khi tung độ của dòng quét không thuộc đoạn này, chúng không cắt cạnh đó

→ Giảm số lượng tính toán, không nhất thiết phải tính giao điểm với tất cả các cạnh





Active Edge List (AEL)

Để gia tăng tốc độ tính toán, chúng ta xây dựng và duy trì một danh sách xác định tọa độ giao điểm của đa giác và đường scanline ở mỗi bước (AEL).

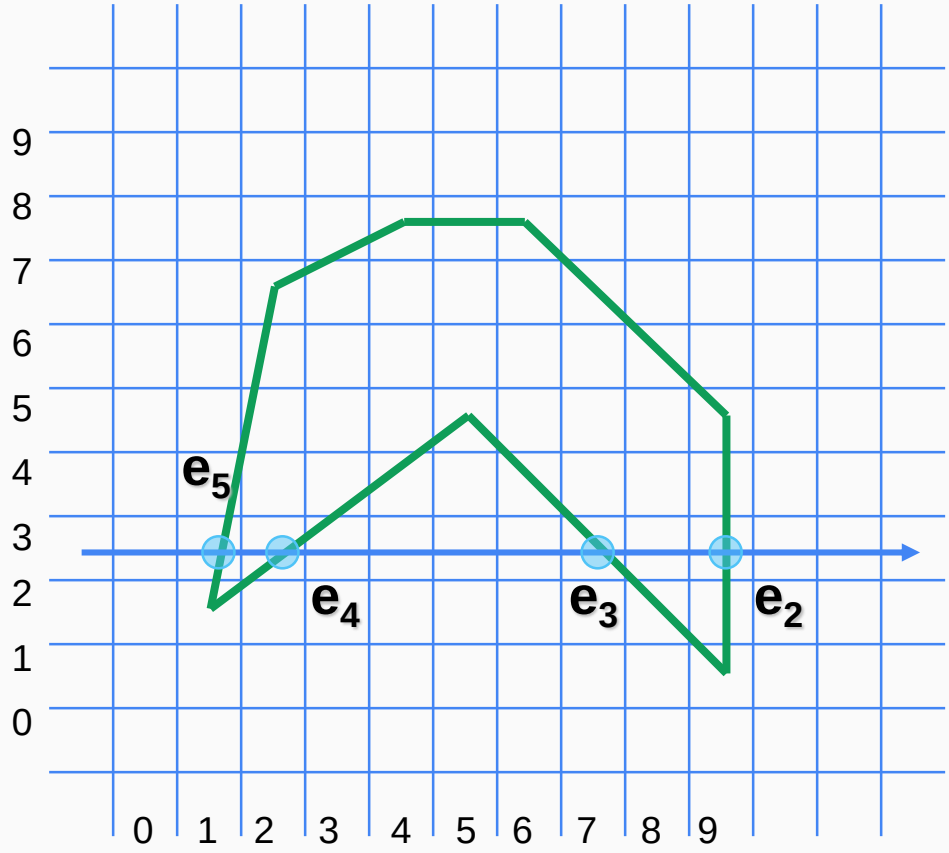
Danh sách này cho phép tính toán giao điểm một cách nhanh chóng bằng cách lưu các thông tin các cạnh mà đường scanline cắt.

Để thuận lợi tính toán, một cạnh có các thông tin sau:

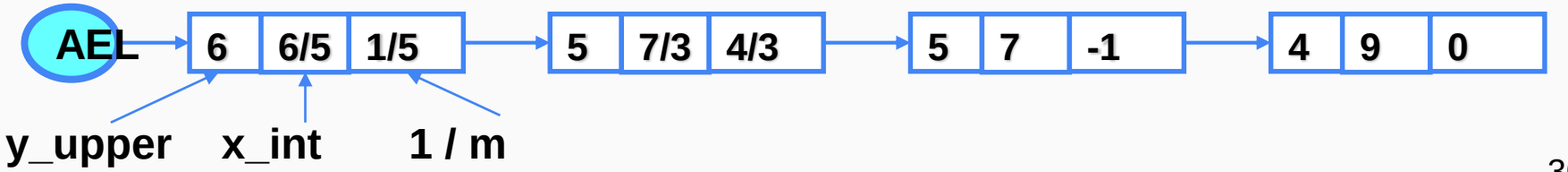
- Tung độ cao nhất y_upper của cạnh (sau khi rút gọn).
- Hoành độ giao điểm x_intersection với đường scanline hiện hành.
- Nghịch đảo hệ số góc 1/m : reciprocal_slope. Chú ý, 1/m được tính trước khi cạnh được rút gọn, do đó bảo đảm tính chính xác của giao điểm.

y_upper	x_int	recip_slope

Ví dụ về AEL



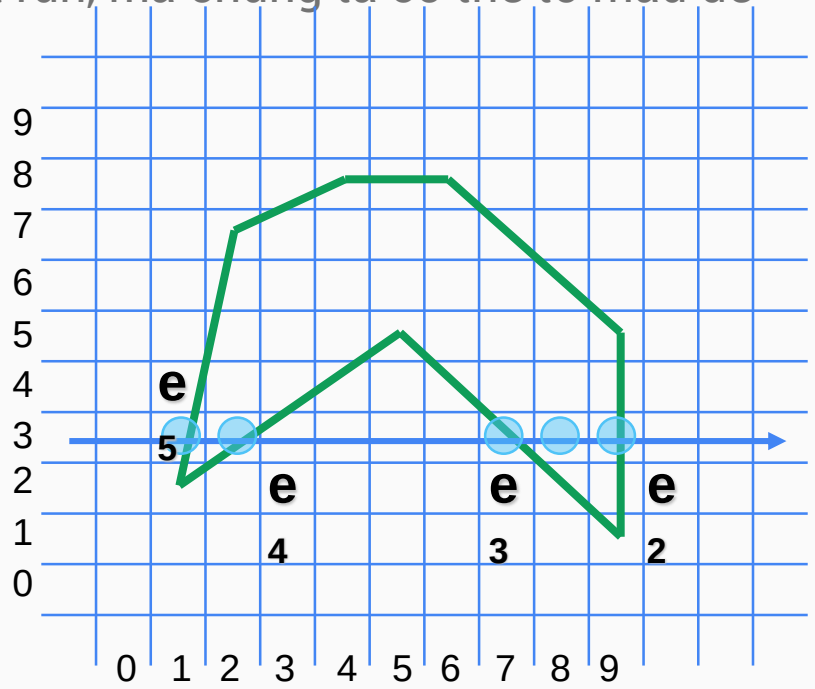
Tại dòng scanline y = 3:



Sử dụng AEL để tô màu tại một dòng scanline

- Tại dòng scanline hiện hành y, AEL lưu trữ giao điểm của scanline và cạnh đa giác.
- Để tô màu, chúng ta sắp xếp các cạnh theo chiều tăng dần của hoành độ giao điểm x_int.
- Mỗi cặp giá trị của x_int xác định một run, mà chúng ta có thể tô màu dễ dàng

```
tmp = AEL;
while (tmp != NULL)
{
   x1 = tmp.x_int;
   tmp = tmp->next;
   x2 = tmp.x_int;
   tmp = tmp->next;
   for(x = x1; x <= x2; x++)
        putpixel(x,y,color);
}
```

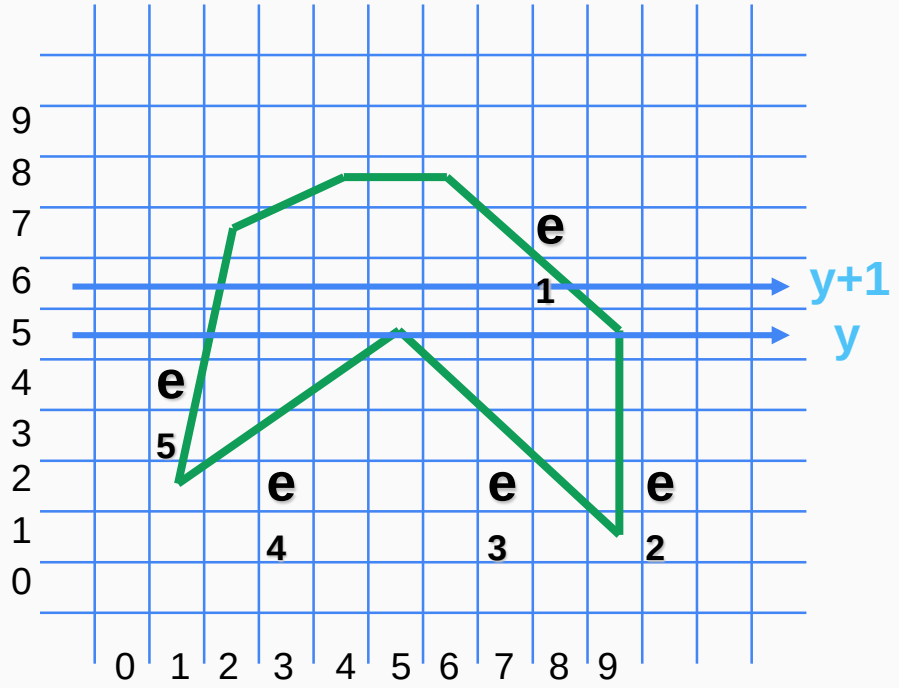


Cập nhật AEL khi dòng scanline di chuyển

Sau khi tô màu tại dòng scanline hiện hành y, AEL phải được cập nhật tại scanline y+1:

1. Bằng cách so sánh y và y_upper của các cạnh trong AEL, ta xác định "dòng scanline mới nằm phía trên cạnh nào đó trong AEL" : xóa cạnh có y vượt quá y_upper.

2. Giá trị của hoành độ giao điểm thay đổi theo dòng scanline. Khi dòng scanline tăng lên 1 thì x_int thay đổi là 1/m : cập nhật tất cả các cạnh với x_int = x_int + recip_slope



Tại y : ael = {e_5, e_4, e_3, e_1}

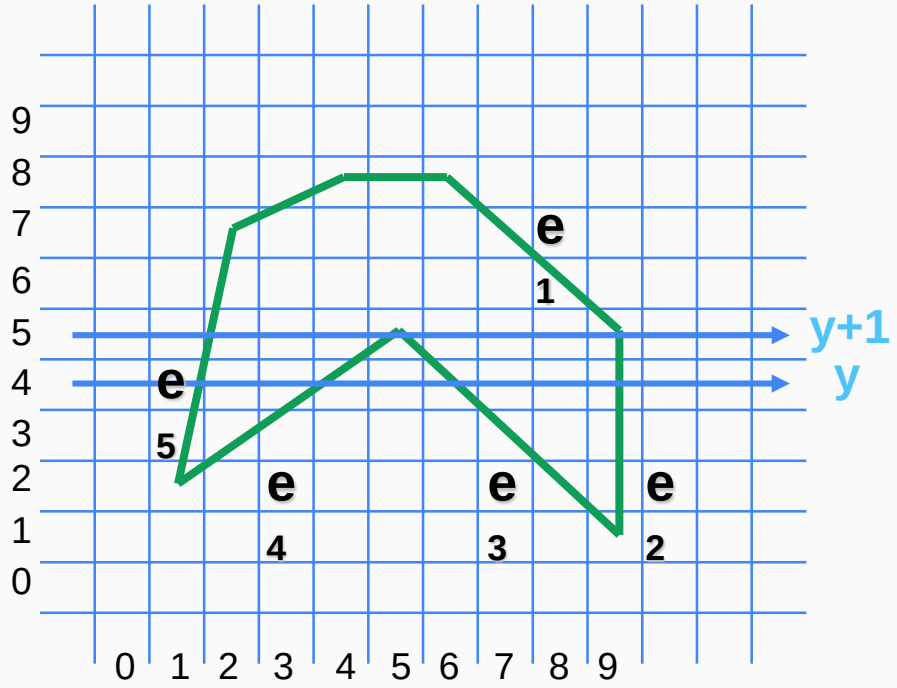
Tại y+1 : ael = {e_5, e_1}

Cập nhật AEL khi dòng scanline di chuyển (cont)

Sau khi tô màu tại dòng scanline hiện hành y, AEL phải được cập nhật tại scanline y+1:

3. Khi y+1 bằng với y_lower của một cạnh thì nó phải được chèn vào AEL (giá trị của cạnh này sẽ trình bày sau trong Edge Table).

4. Thứ tự của hoành độ giao điểm có thể đảo ngược khi 2 cạnh giao nhau (đa giác tự cắt) : AEL phải được sắp xếp lại.



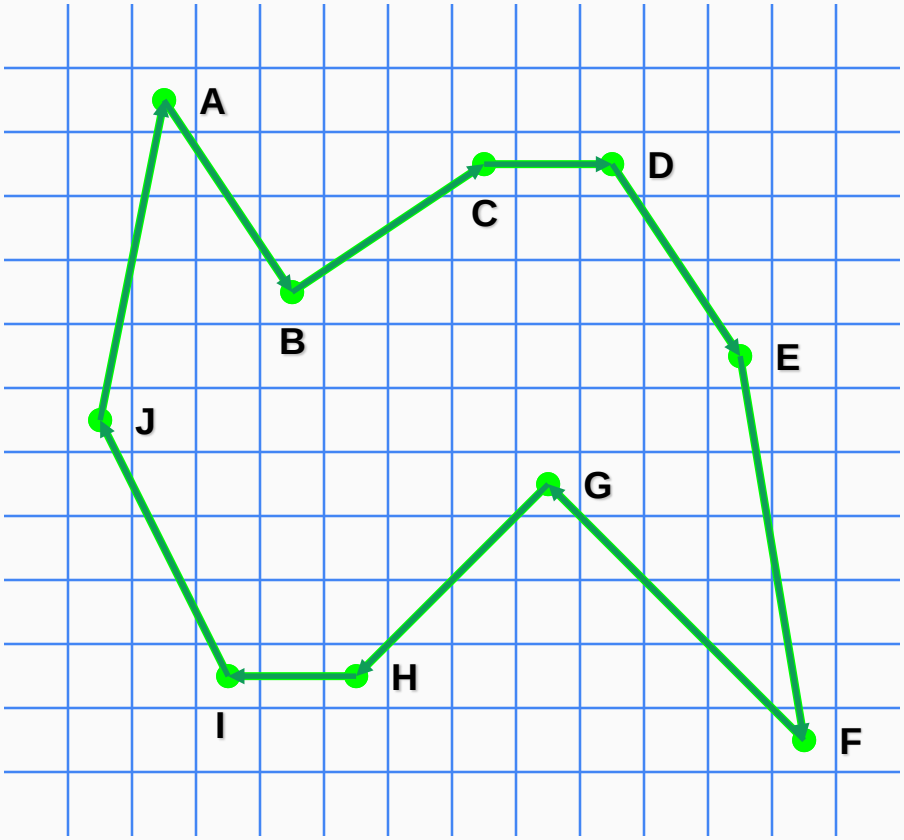
Tại y : ael = {e_5, e_4, e_3, e_2}

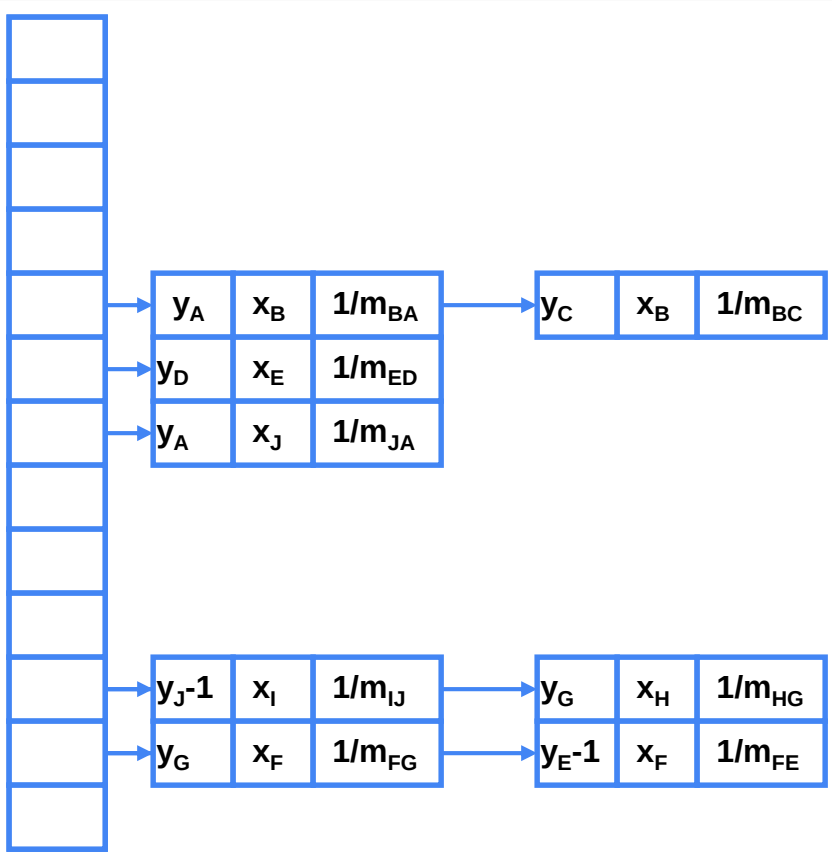
Tại y+1 : ael = {e_5, e_4, e_3, e_1}

EdgeTable

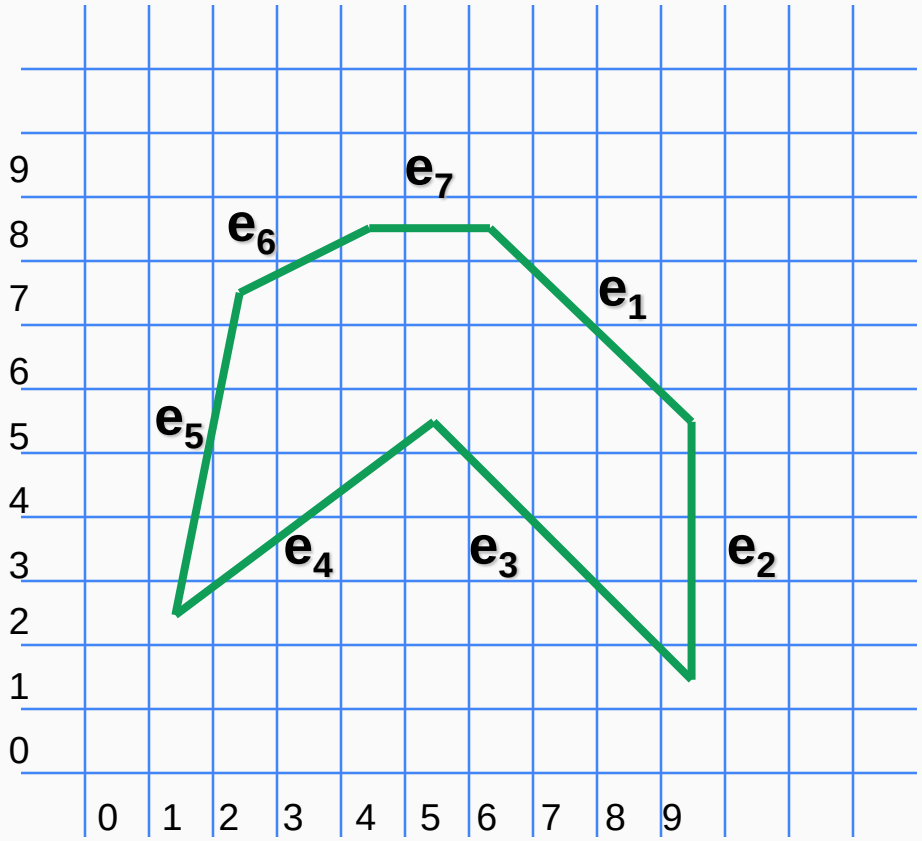
- Để xác định cạnh nào được chèn vào AEL, chúng ta phải xét từng đỉnh của đa giác. Tuy nhiên, cấu trúc EdgeTable được tạo ra để lưu trữ thông tin các cạnh trước khi quá trình tô màu xảy ra, bảo đảm yêu cầu cập nhật nhanh chóng AEL:
- Mỗi cạnh được xác định y_upper, recip_slope thông qua đỉnh đa giác, và x_int là hoành độ đỉnh dưới của cạnh.
- EdgeTable là một mảng các danh sách các cạnh (như danh sách AEL). EdgeTable[y] chứa danh sách các cạnh có y_lower = y

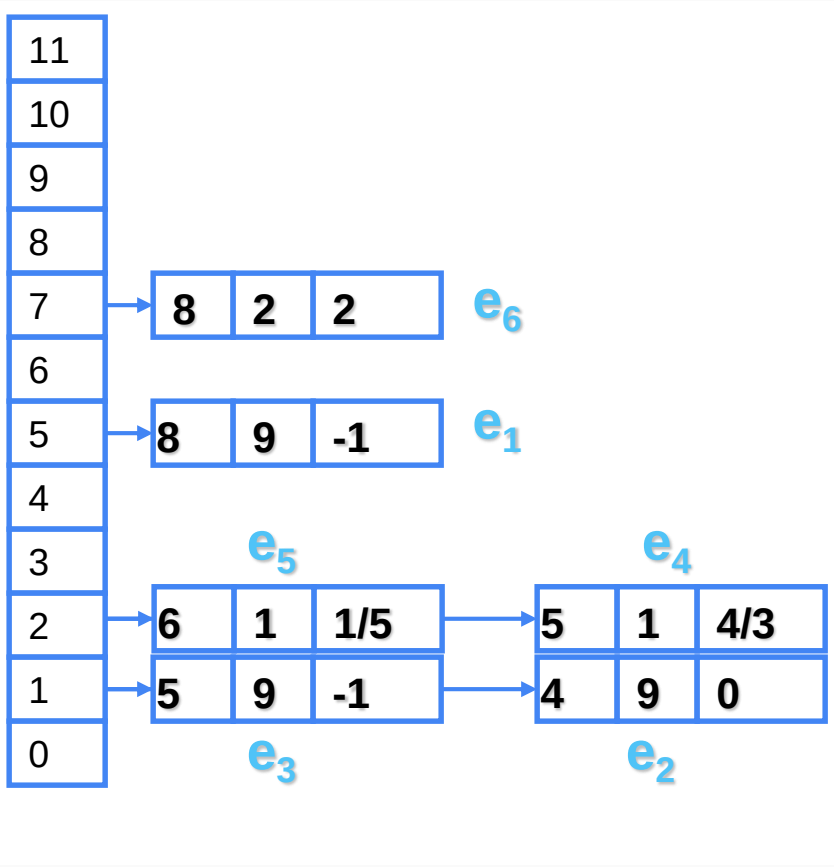
Building EdgeTable





Ví dụ EdgeTable

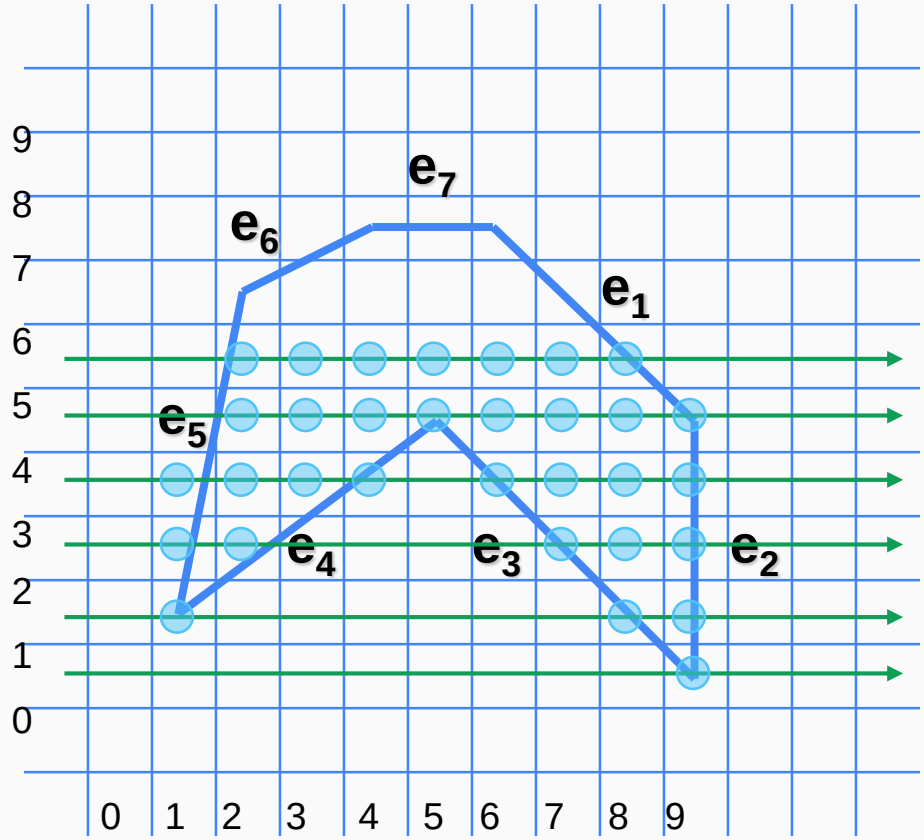


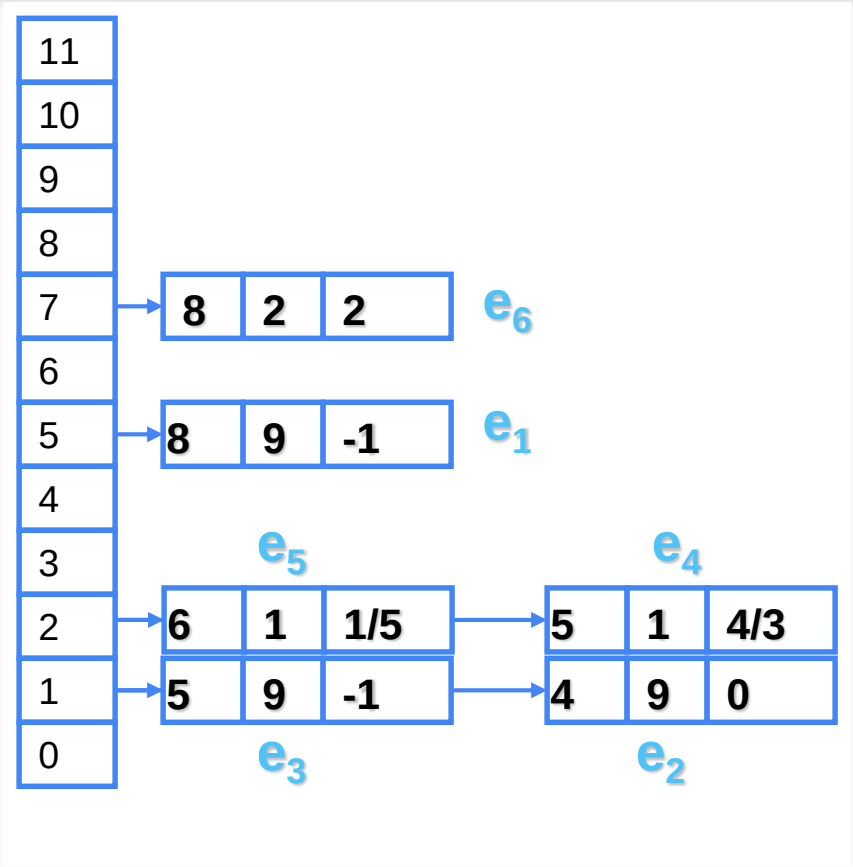


Dùng EdgeTable để cập nhật AEL

- Sau khi tạo EdgeTable, AEL dễ dàng được cập nhật thông qua các cạnh có sẵn trong EdgeTable tại dòng scanline y:
- Chèn các cạnh tại EdgeTable[y] vào AEL : nghĩa là dòng scanline bắt đầu cắt các cạnh có y_lower = y
- Giá trị ban đầu của x_int là hoành độ của đỉnh dưới nên chính là hoành độ giao điểm ban đầu.

Ví dụ


9
8
7
6
5
4
3
2
1
0
0 1 2 3 4 5 6 7 8 9
e7
e6
e1
e5
e4
e3
e2


11
10
9
8
7
6
5
4
3
2
1
0
8 2 2
e6
8 9 -1
e1
e5
e4
6 1 1/5
5 1 4/3
5 9 -1
4 9 0
e3
e2


AEL
6 9/5 1/5
8 8 -1
5 5 -1
8 9 -1

Thuật toán cải tiến

```
Xây dựng EdgeTable;
AEL = NULL;
for( y = y_min; y <= y_max; y++)
{
        Chèn tất cả các cạnh trong EdgeTable[y] vào AEL;
        if (AEL != NULL)
        {
                Sắp xếp AEL theo chiều tăng dần của x_int;
                Tô màu các run trong AEL;
                Xóa các cạnh trong AEL có y_upper = y;
                Cập nhật giá trị x_int trong các cạnh của AEL;
        }
}
```

Tài liệu tham khảo

- Slide này được biên soạn được tham khảo từ một số tài liệu sau:
  - http://jcsites.juniata.edu/faculty/rhodes/graphics/verts2frags2.htm
  - Slide: Computer Graphics 4: Bresenham Line Drawing Algorithm, Circle Drawing & Polygon Filling By:Kanwarjeet Singh
  - http://www.math.hcmuns.edu.vn/~hvthao/dhmt/areafilling
  - https://www.slideshare.net/wahab13/polygon-fill
  - https://www.slideshare.net/kumar_vic/fill-areaalgorithms
  - https://www.slideshare.net/kumar_vic/comp175-04regionfilling
  - http://slideplayer.com/slide/5155621/
  - https://www.scratchapixel.com/lessons/3d-basic-rendering/rasterization-practical-implementation/rasterization-stage
  - https://soict.hust.edu.vn/~trungtt/uploads/slides/CG04_Base_Algorithm.pdf