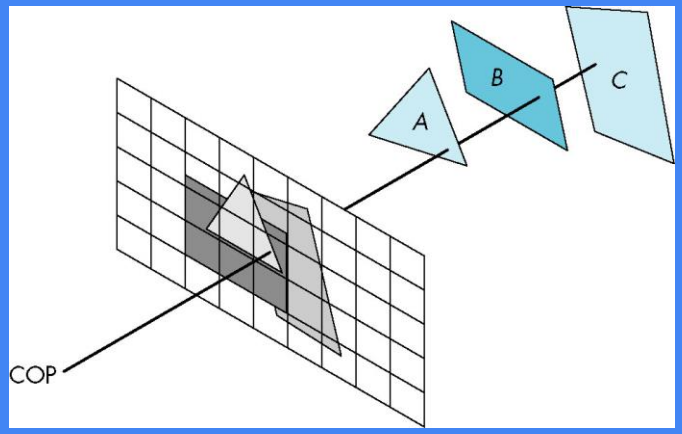
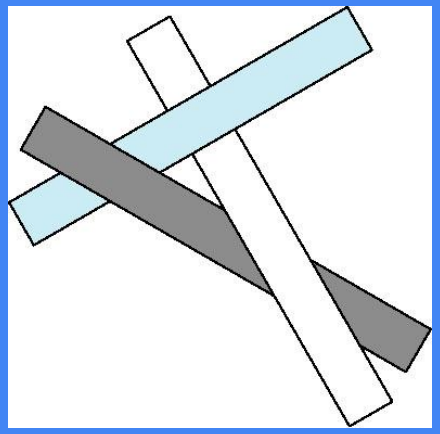
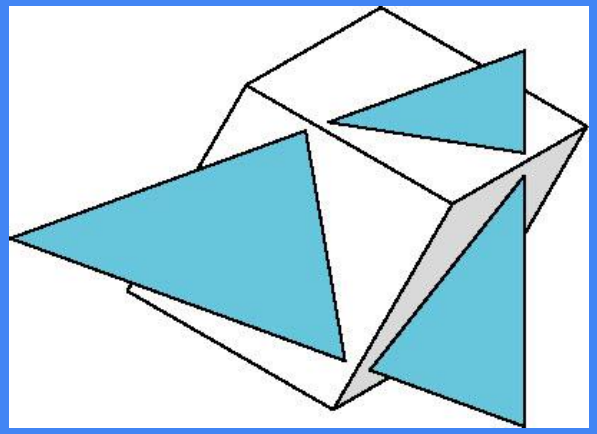
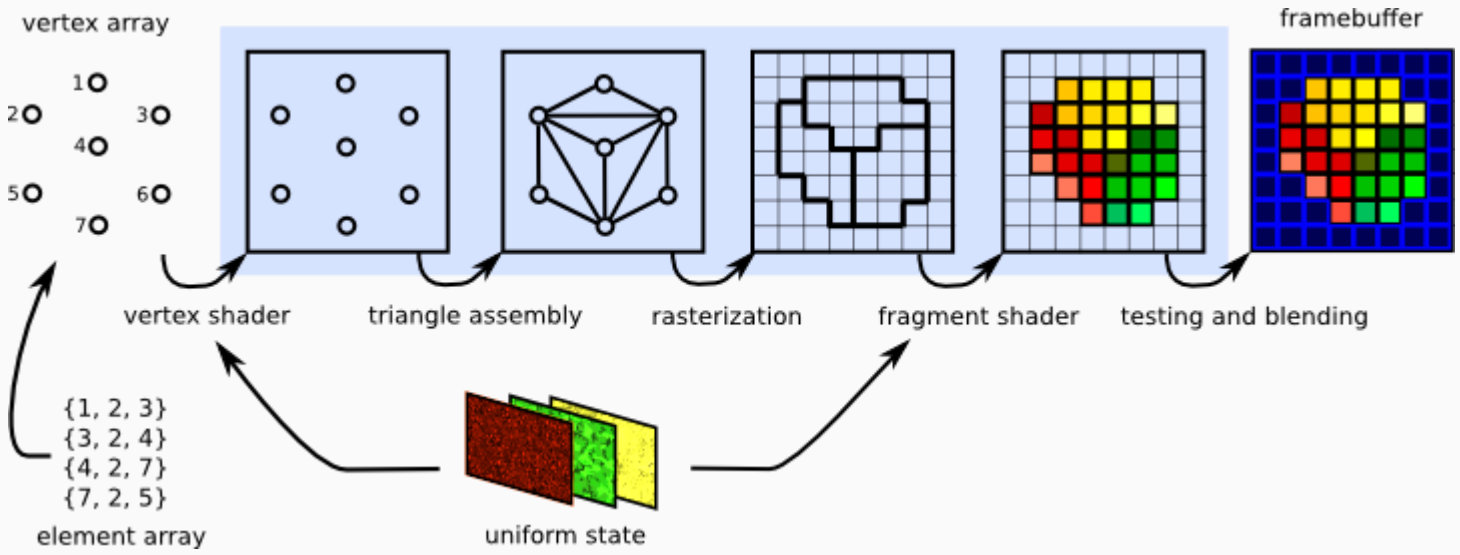
CS113 - ĐỒ HỌA MÁY TÍNH VÀ XỬ LÝ ẢNH

Khử mặt khuất – **Hidden Surface Removal (HSR)**



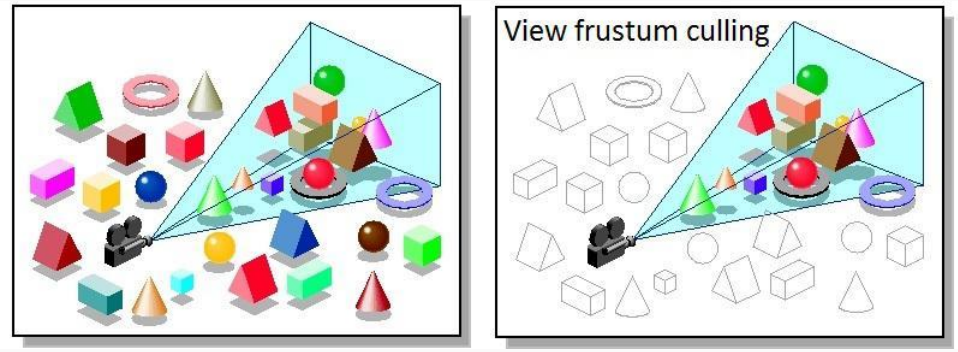





vertex array
framebuffer
1
2
3
4
5
6
7
vertex shader
triangle assembly
rasterization
fragment shader
testing and blending
{1, 2, 3}
{3, 2, 4}
{4, 2, 7}
{7, 2, 5}
element array
uniform state

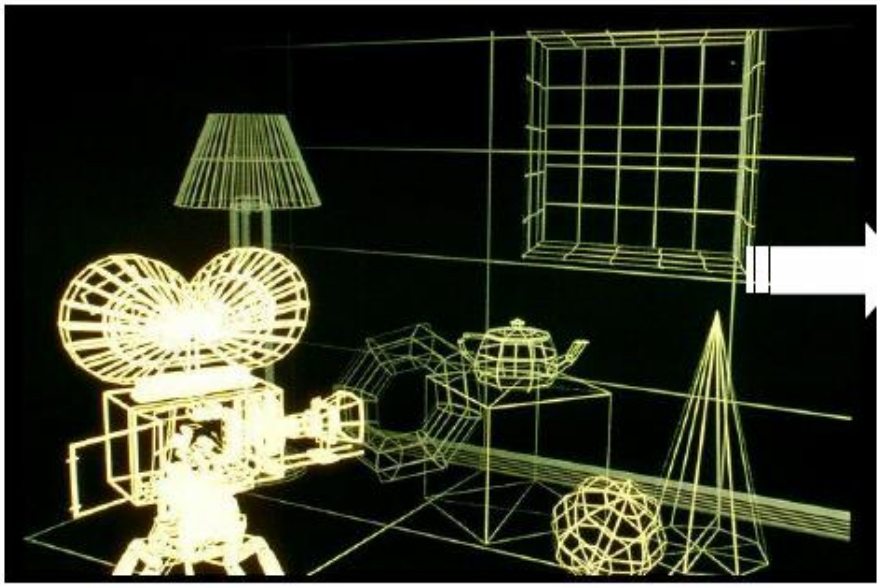
Sự hữu hình của các đối tượng cơ bản

- Chúng ta không muốn phí thời gian để hiển thị những đối tượng không đóng góp vào bức ảnh cuối cùng.
- Một đối tượng có thể không hữu hình vì 3 lý do:
  - Nằm ngoài vùng hiển thị
  - Quay vào trong (*back-facing*)
  - Bị che bởi các đối tượng khác gần người quan sát hơn
- Làm thế nào để loại bỏ chúng một cách hiệu quả?
- Làm thế nào để xác định chúng một cách hiệu quả?



Vấn đề hữu hình

- Loại bỏ các bề mặt hướng ra phía khác so với người quan sát.
- Loại bỏ các bề mặt che bởi các đối tượng gần hơn.

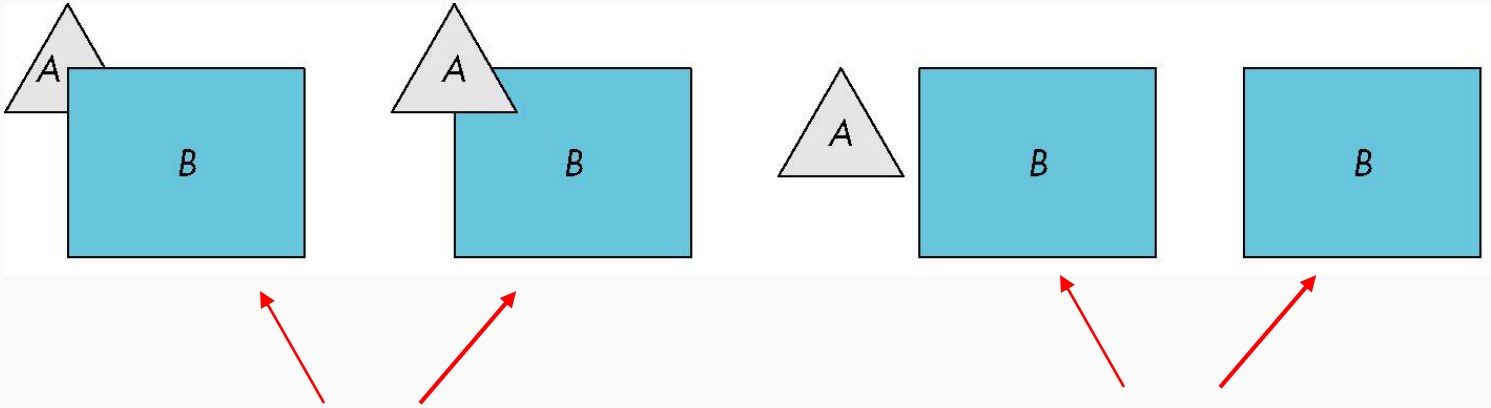


Phân loại thuật toán

- **Không gian đối tượng (object-space)**
  - So sánh các đối tượng, cũng như các thành của chúng với mỗi cái khác để xác định xem các mặt và đường nào sẽ được đánh nhãn là không nhìn thấy được.
  - Thường xử lý cảnh vật theo thứ tự các vật thể
- **Không gian ảnh (image-space)**
  - Tính chất nhìn thấy được của một điểm được quyết định bởi điểm ở vị trí pixel trên mặt phẳng chiếu.
  - Thường xử lý cảnh vật theo thứ tự ảnh
- Hầu hết các thuật toán khử mặt khuất dùng phương pháp không gian ảnh.
- Các thuật toán khử đường khuất hầu hết dùng phương pháp không gian đối tượng.

Hidden Surface Removal

- Object-space approach: use pairwise testing between polygons (objects)



- Worst case complexity $O(n^2)$ for n polygons

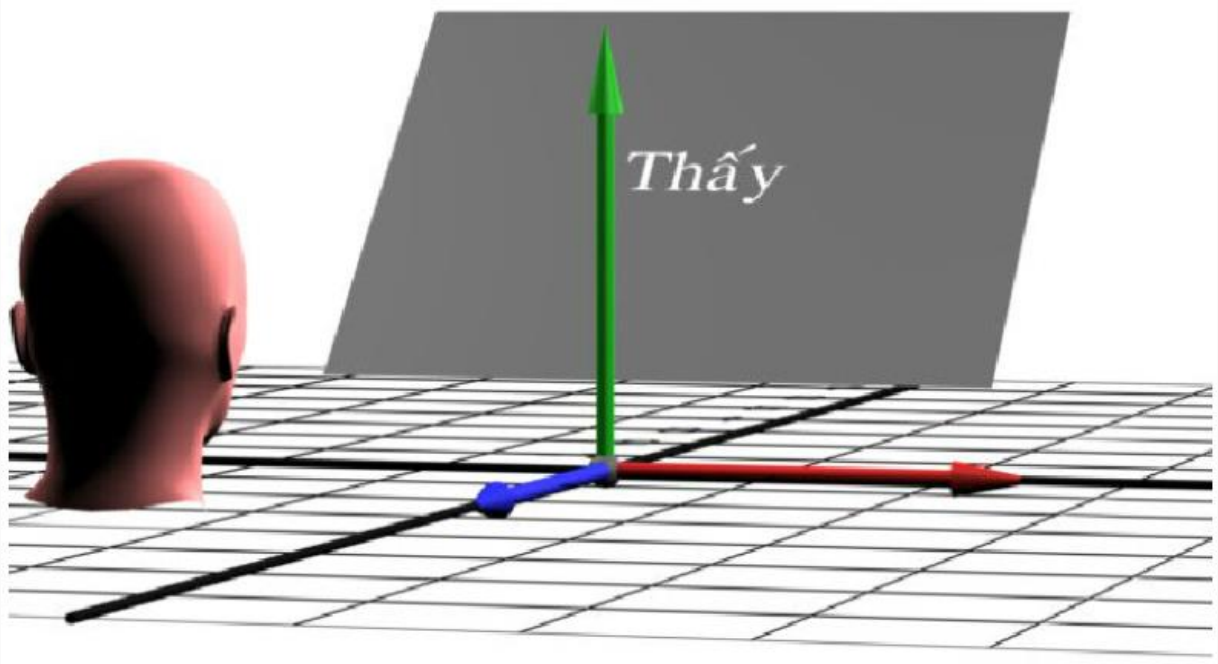
Một số thuật toán

- Thuật toán loại bỏ mặt sau
- Thuật toán Painter
- Thuật toán depth sorting
- Thuật toán scanline
- Thuật toán warnock
- Thuật toán BSP
- Thuật toán Ray casting

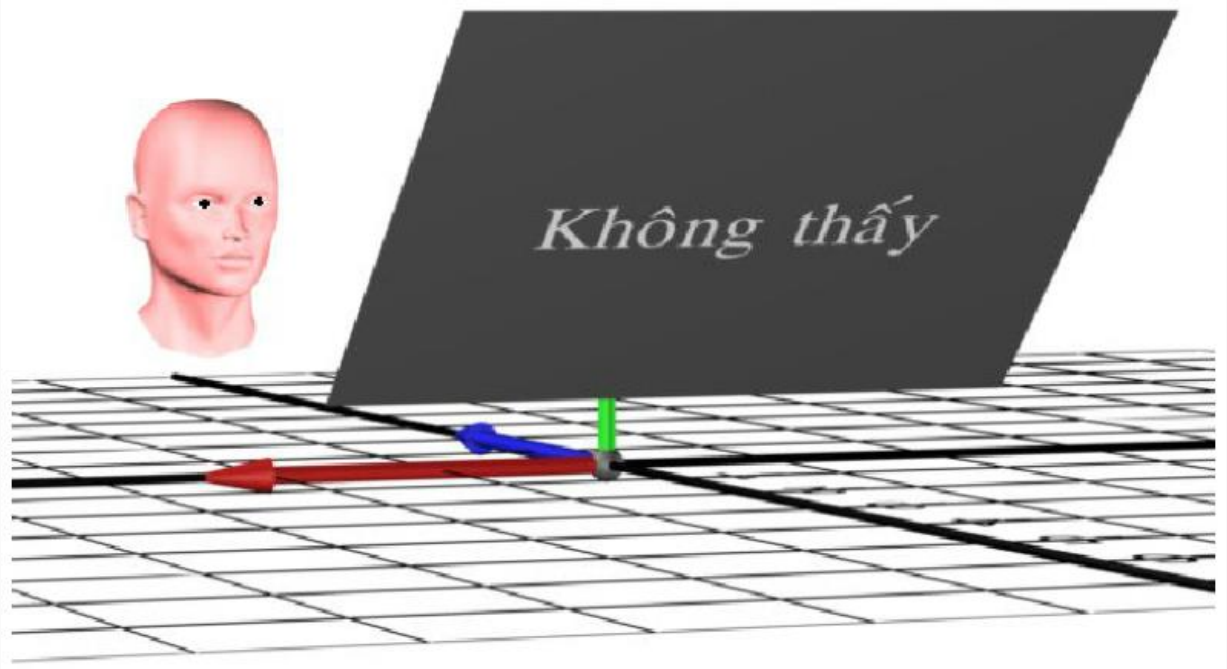
Thuật toán loại bỏ mặt sau

Backface Culling

Mặt trước

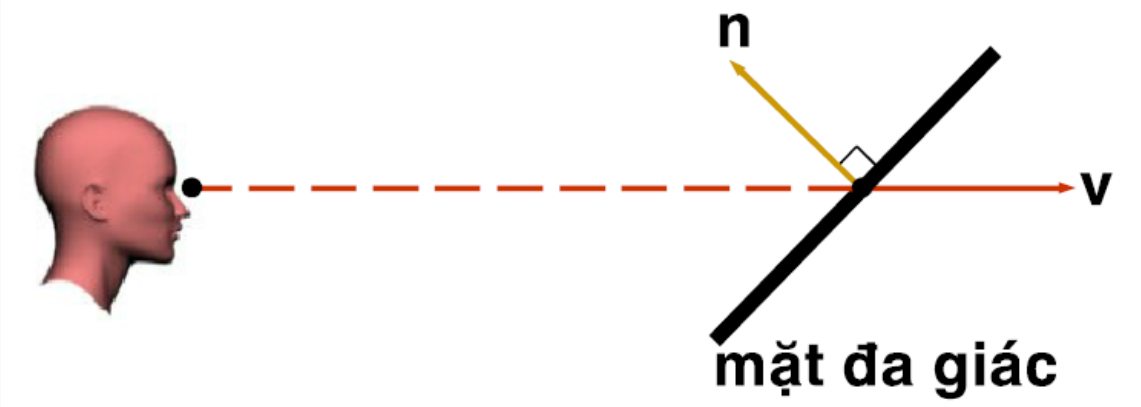


Mặt sau

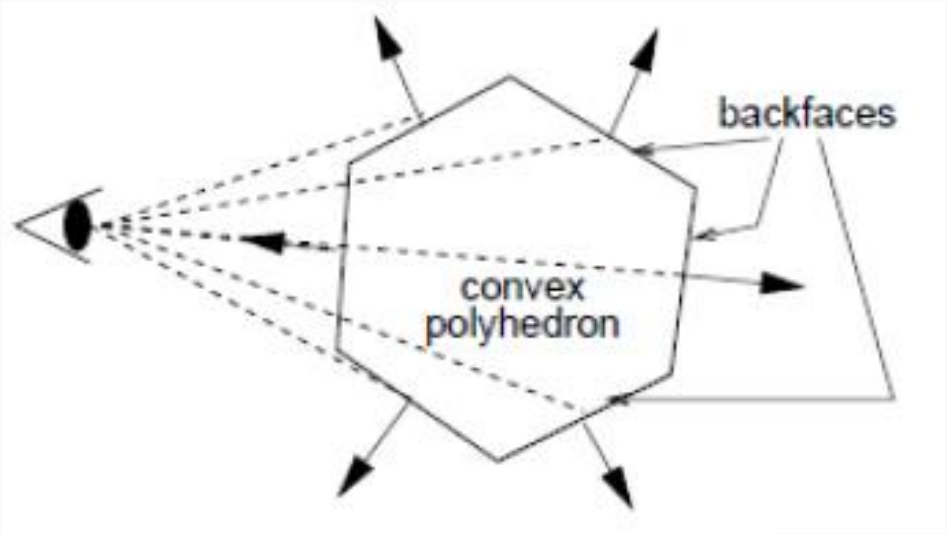


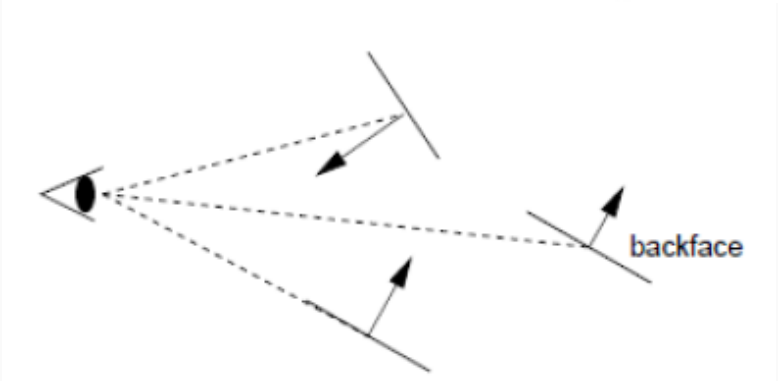
Thuật toán loại bỏ mặt sau

- Duyệt tuần tự các mặt đa giác trong danh sách.
- Xét đa giác f
  - Bước 1: Xác định v
  - Bước 2: Tính v.n
  - Bước 3: Xét giá trị v.n
    - >=0: loại bỏ mặt đa giác f
    - <0 : giữ lại mặt đa giác f

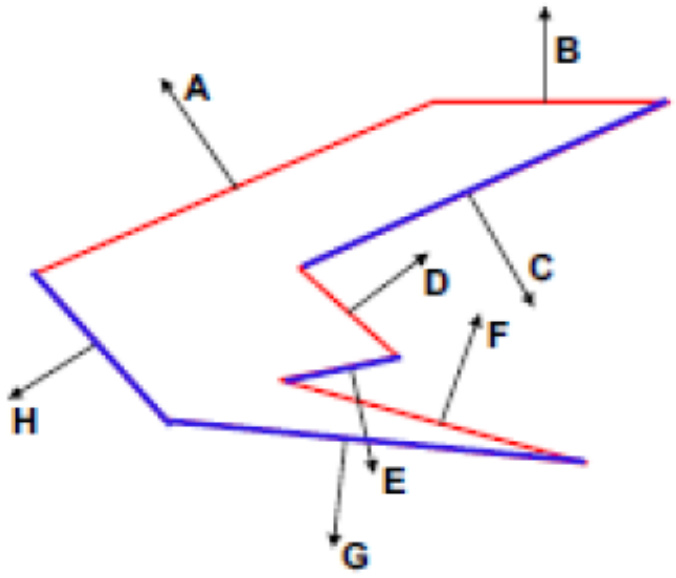


Ví dụ





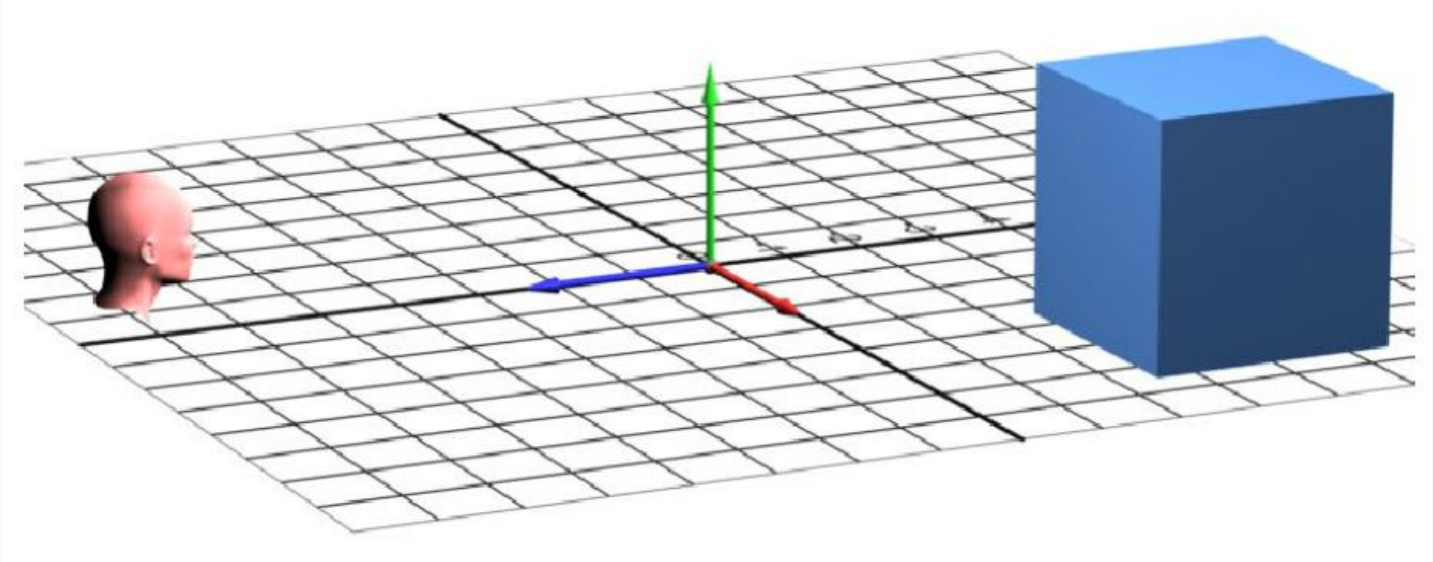
Ví dụ



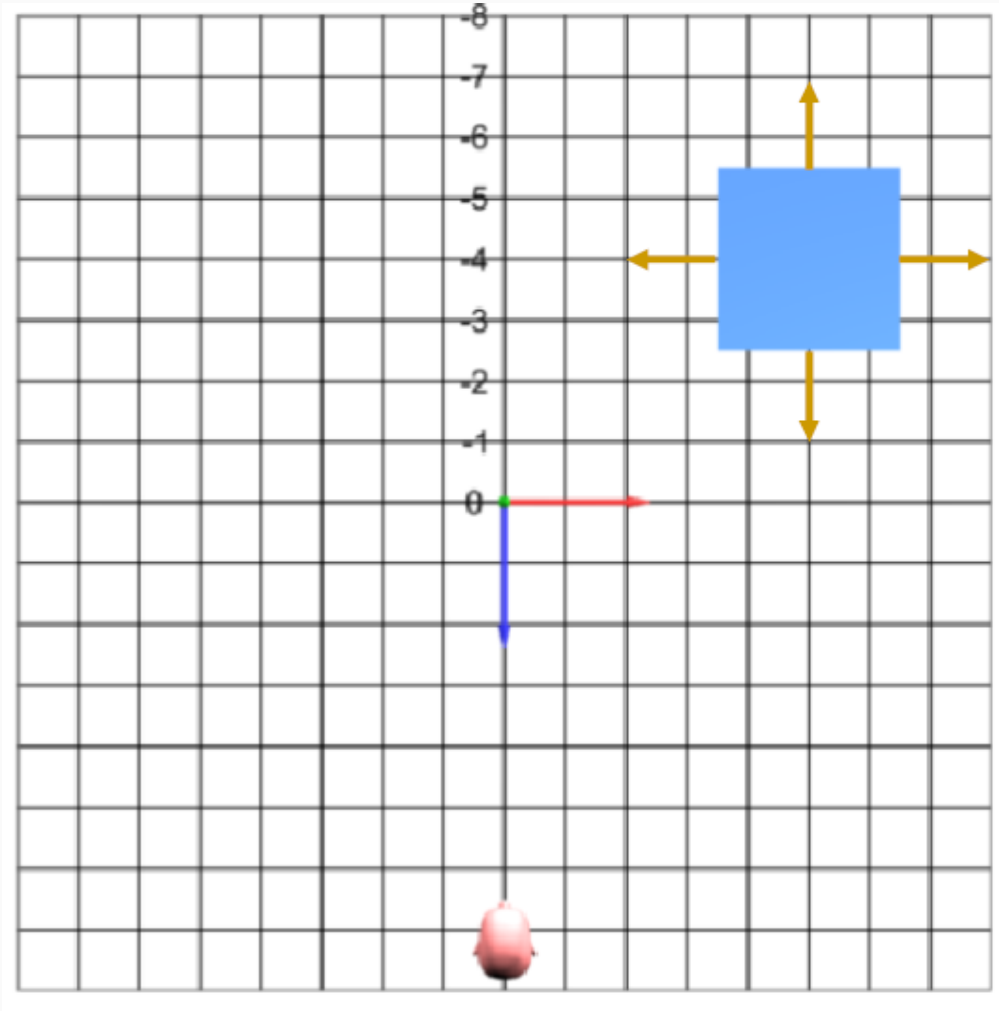
Mặt sau: A, B, D, F

Mặt trước: C, E, G, H

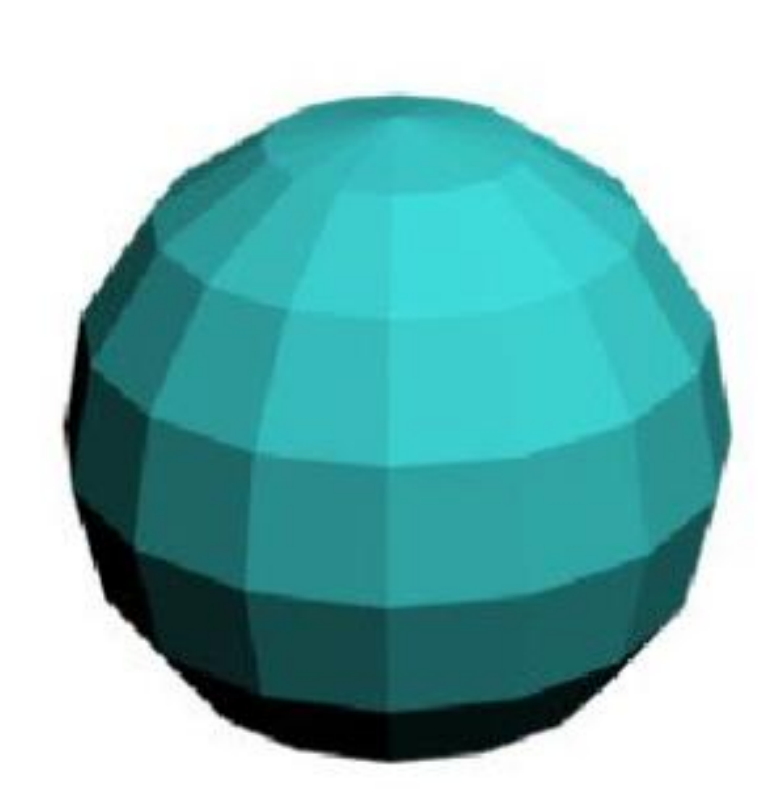
Ví dụ

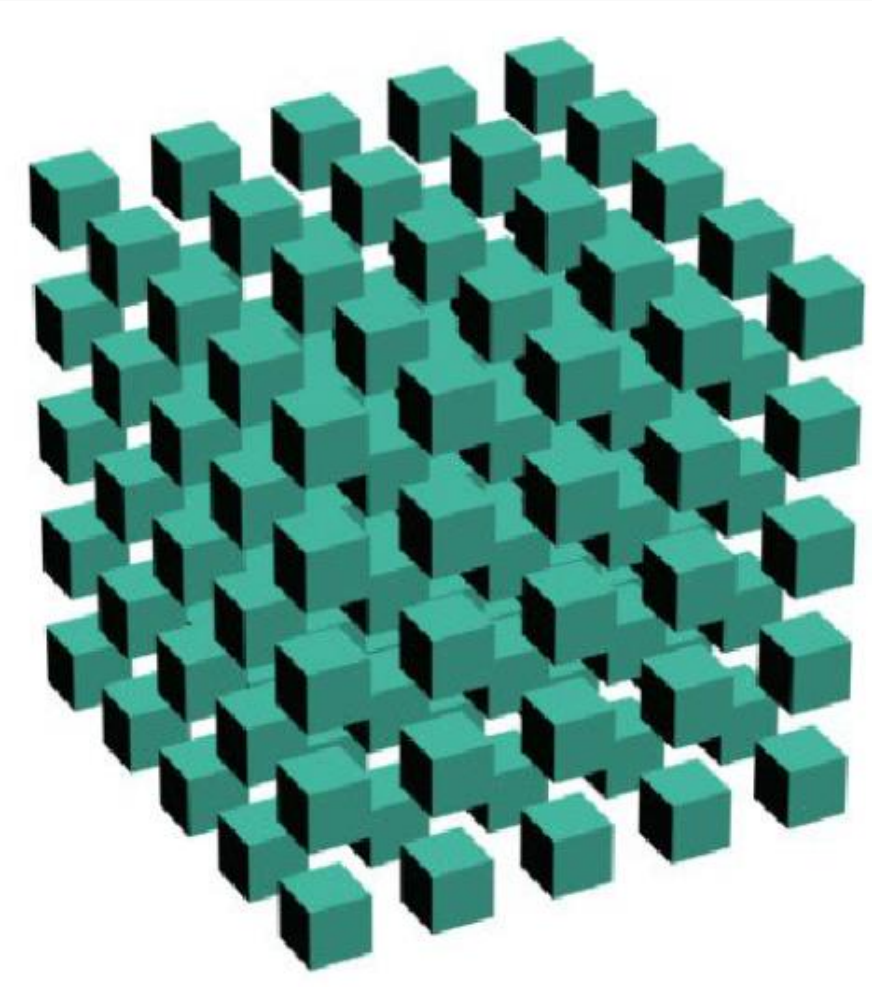


Ví dụ



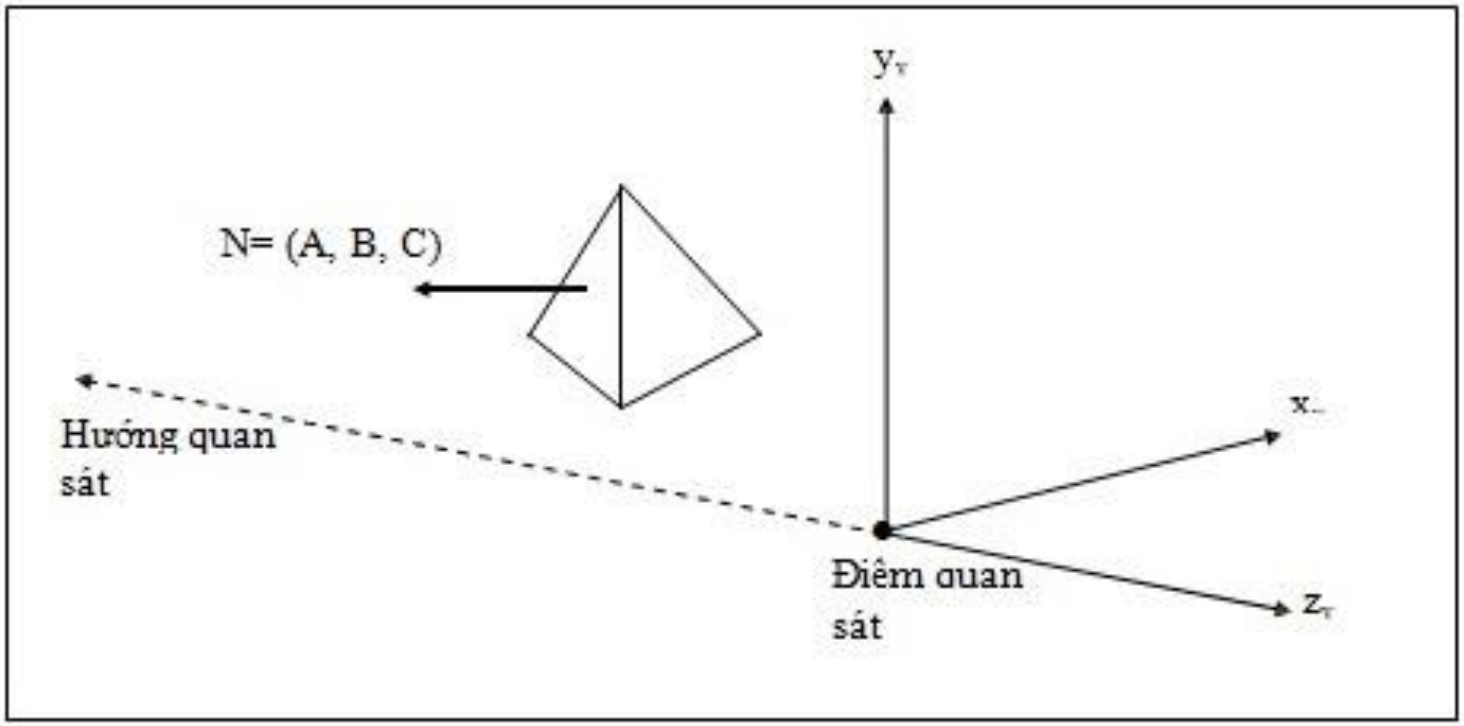
Ví dụ





Lưu ý

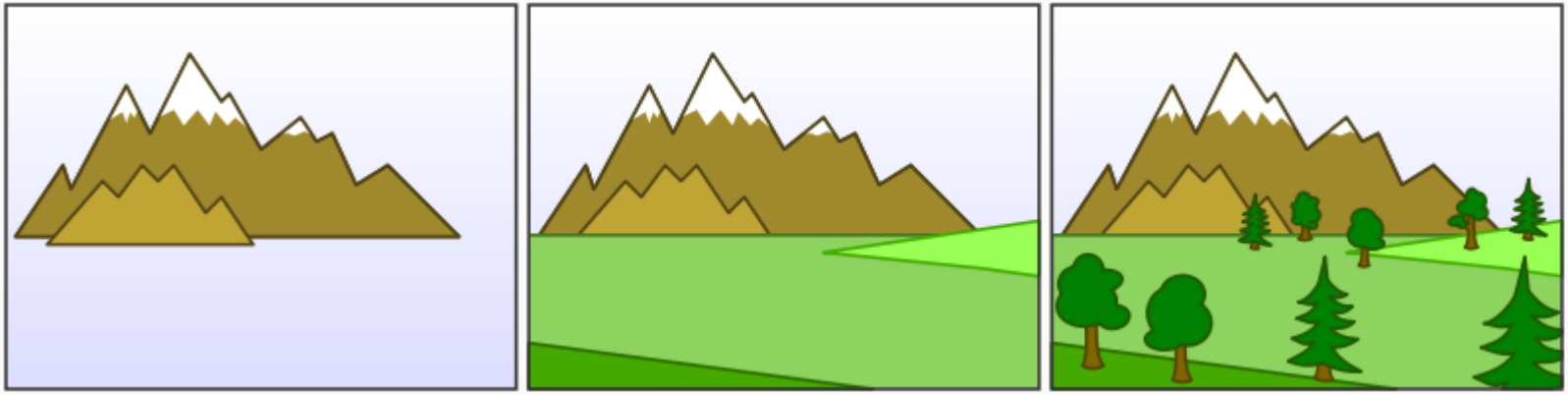
- Đối với một khối đa diện lồi đơn lẽ, như hình kim tự tháp, việc kiểm tra này xác định tất cả các mặt bị che khuất trên đối tượng, bởi vì mỗi mặt thì là hoàn toàn được nhìn thấy hoặc hoàn toàn bị che khuất.



Thuật toán Painter

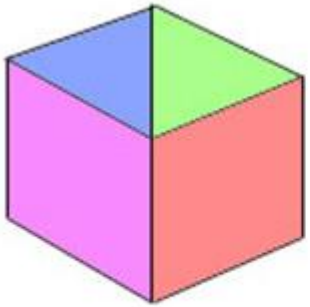
Thuật toán

1. Sắp xếp các đa giác theo thứ tự từ sau ra trước.
2. Vẽ từng đa giác theo thứ tự đã sắp

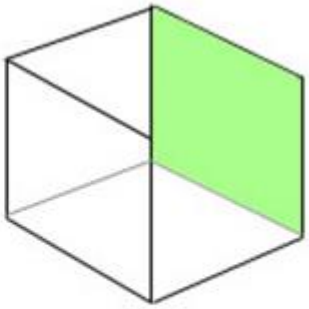


Source: https://en.wikipedia.org/wiki/Painter%27s_algorithm

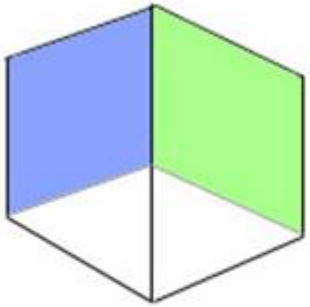
Ví dụ



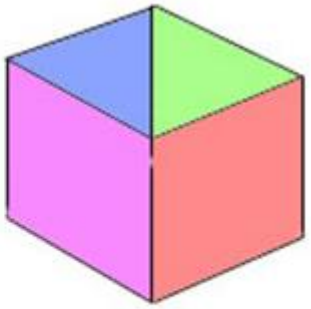
Example: Open Cube.



Paint farthest polygon first.



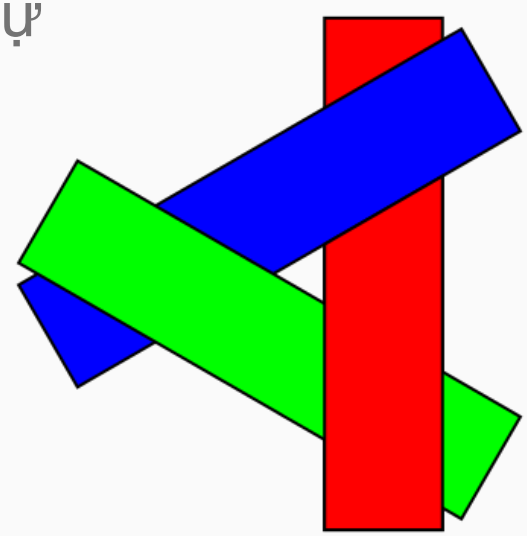
Next polygon in back-to-front order.



Closest polygons last, overwrite existing ones

Nhận xét

- Dễ cài đặt
- Mỗi lần rendering, đòi hỏi phải sắp xếp các đa giác.
- Dựa trên điểm nào để sắp xếp ? (Thông thường là tâm của đa giác)
- Một pixel có thể được vẽ lại (repainted) nhiều lần.
- Tuy nhiên, trong thực tế không phải lúc nào cũng xác định được thứ tự



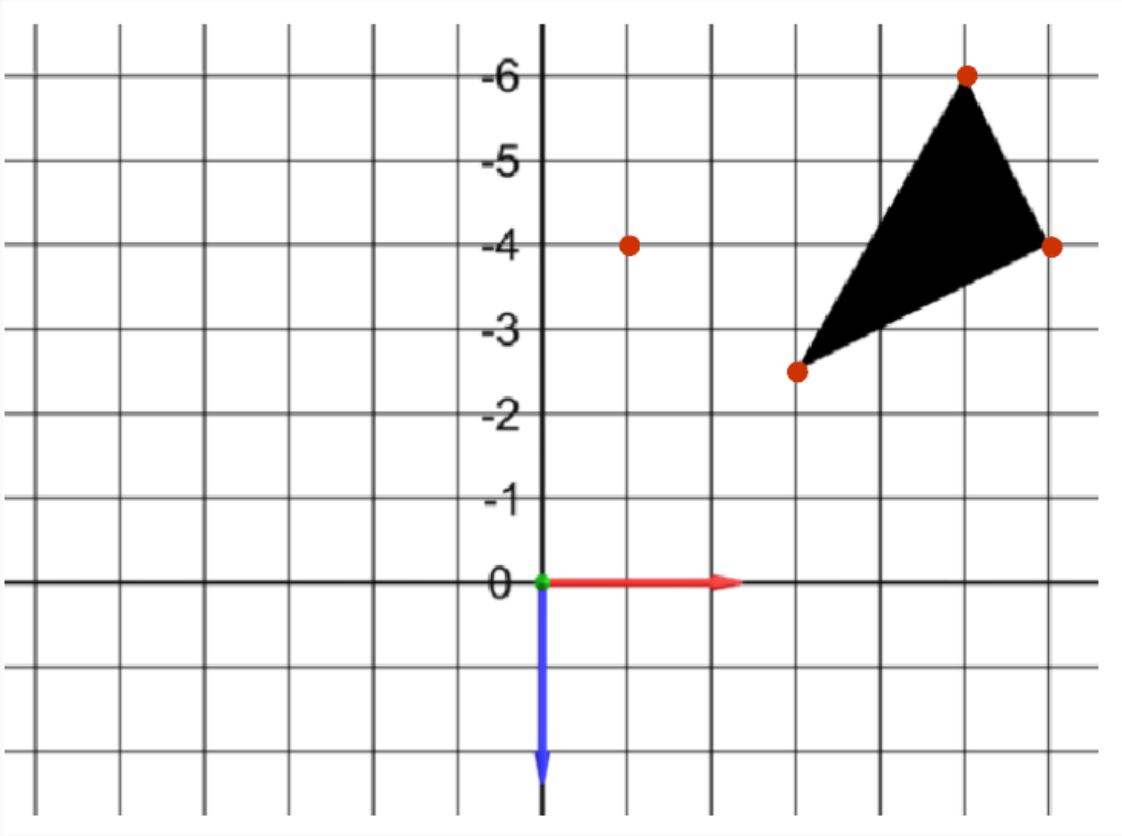
Overlapping polygons can cause the algorithm to fail

Source: https://en.wikipedia.org/wiki/Painter%27s_algorithm

Thuật toán Depth sorting

Khái niệm độ sâu

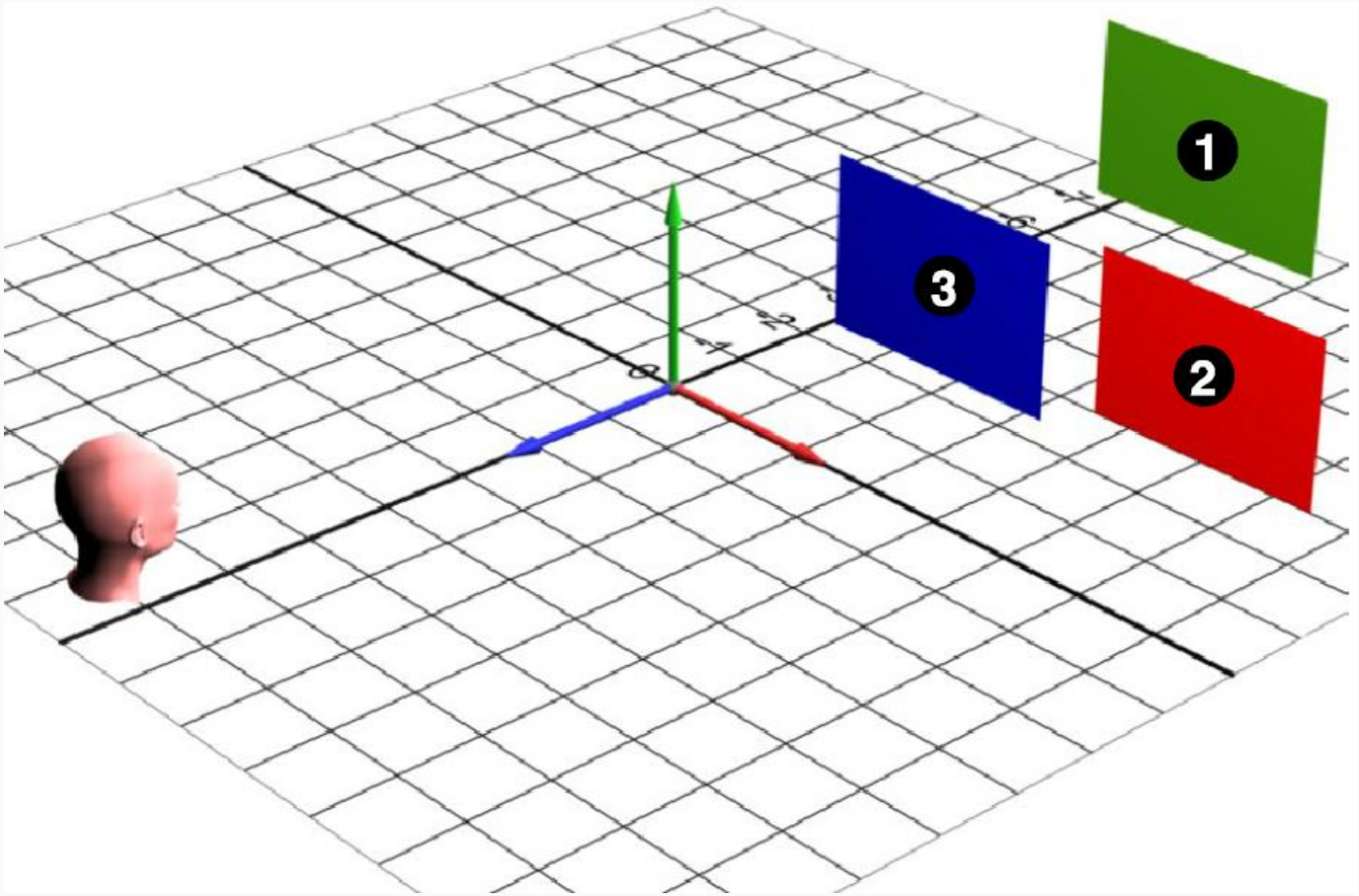
- Điểm: độ sâu p = - p_z
- Đa giác: độ sâu f = min{độ sâu p_0,…., độ sâu p_{n-1}}



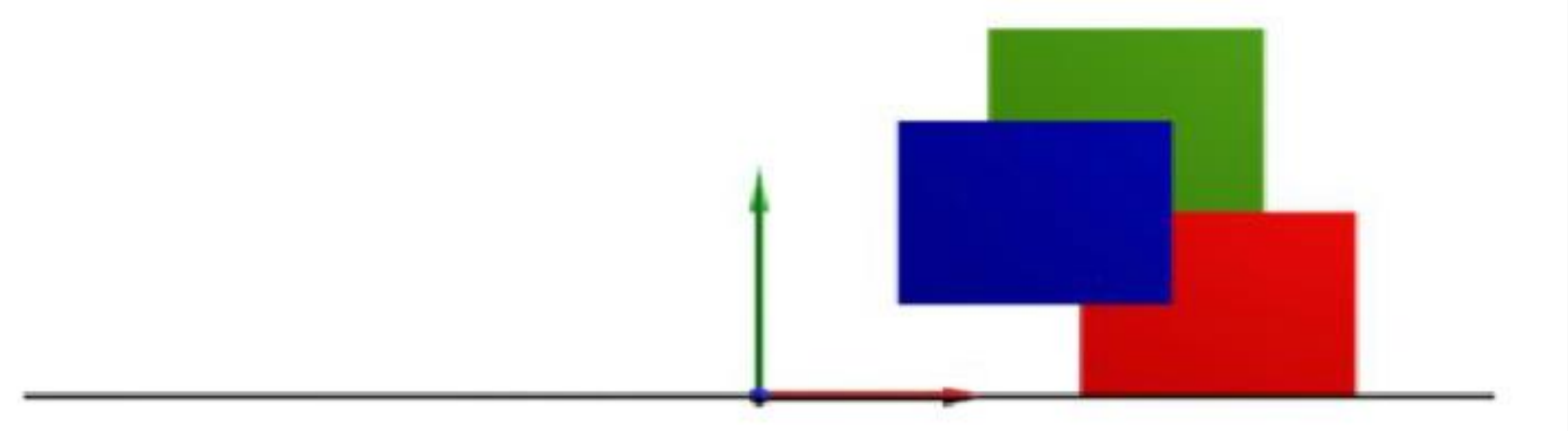
Thuật toán

- Bước 1: Sắp xếp các mặt đa giác trong danh sách theo thứ tự độ sâu tăng dần.
- Bước 2: Vẽ các mặt đa giác theo thứ tự đã sắp xếp

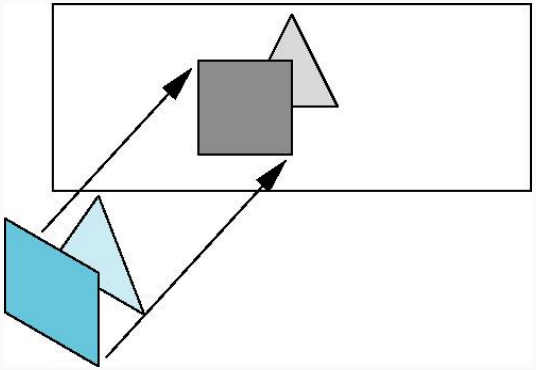
Ví dụ



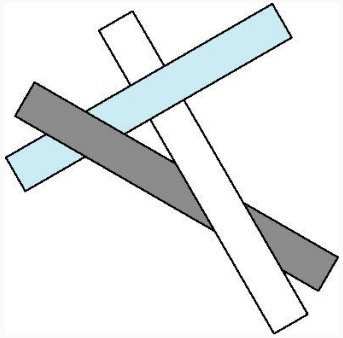
Ví dụ



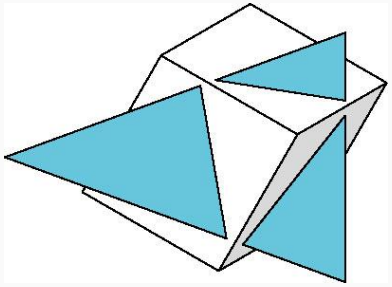
Nhận xét



Easy cases – One polygon is completely behind the other.
Just “paint” over it.



cyclic overlap



penetration

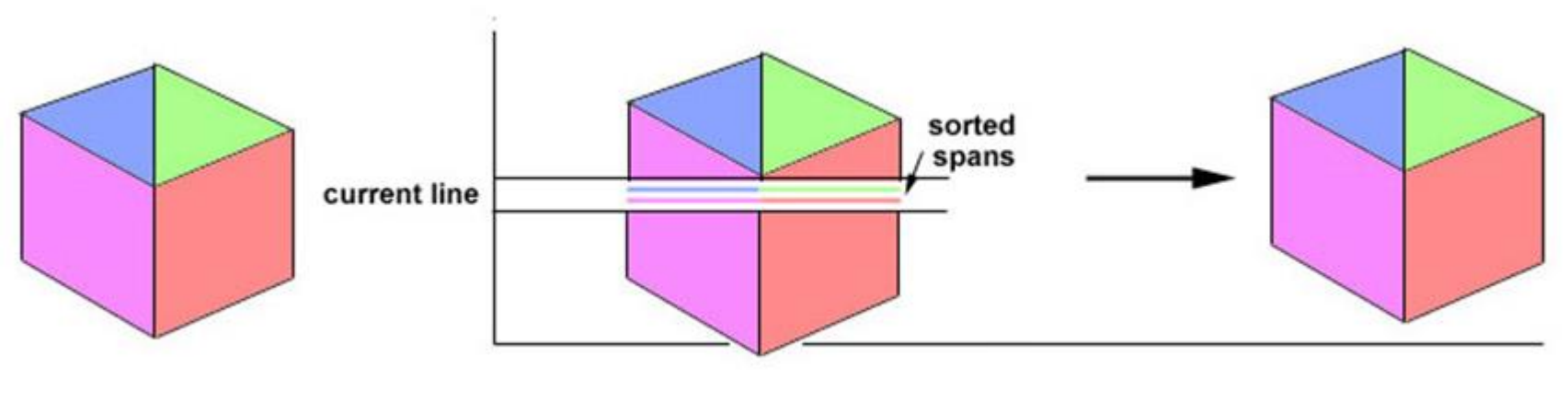
Thuật toán Scanline

Giới thiệu

- Là sự mở rộng của thuật toán scan-line để tô phần bên trong của một đa giác → xử lý với nhiều mặt.
- Khi mỗi đường quét được xử lý:
  - Tất cả các mặt đa giác cắt đường quét đó sẽ được kiểm tra để xác định xem mặt nào nhìn thấy được.
  - Ở mỗi vị trí trên cùng đường quét các tính toán độ sâu được thực hiện cho mỗi mặt để xác định mặt gần mặt phẳng quan sát nhất.
  - Khi mặt mặt nhìn thấy được được xác định, giá trị độ sáng cho vị trí đó được nhập vào vùng đệm làm tươi (refresh buffer).

Thuật toán

1. Vertically scan the y-axis of the final image.
2. Rasterize all polygons at the same time.
3. For each line, sort polygon spans and render.



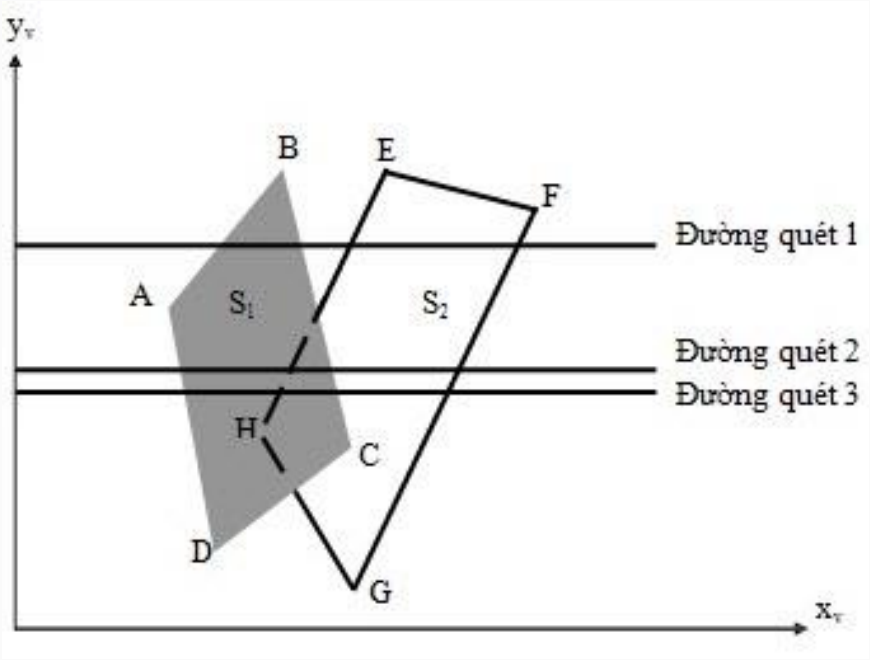
Solves self-overlap problem of Painter's algorithm because pixels are sorted per scan-line instead of per polygon

Thuật toán

- Chúng ta định nghĩa một cờ (flag) cho mỗi mặt, cờ này được đặt là on hay off để chỉ ra mỗi vị trí nằm dọc trên đường quét là nằm trong hay nằm ngoài mặt.
- Các đường quét được xử lý từ trái sang phải. Ở biên bên trái nhất của một mặt, cờ của mặt là on; và ở biên bên phải nhất cờ là off.

Ví dụ

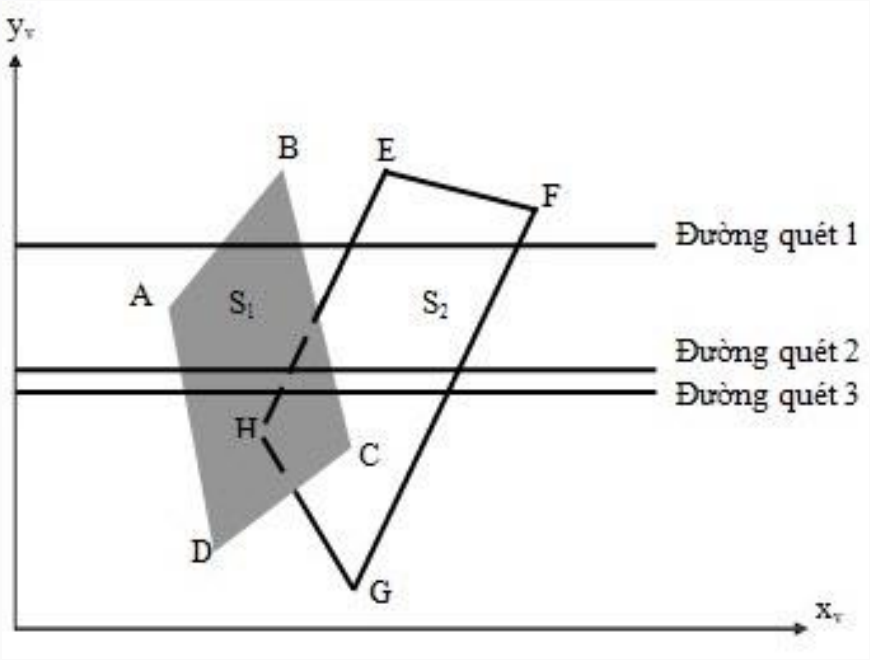
- Dòng quét 1 cắt các cạnh AB, BC, HE, và FG.
- Đối với các vị trí dọc theo đường quét này giữa các cạnh AB và BC, chỉ cờ mặt S1 là on. Do đó, không phép tính độ sâu nào là cần thiết, và thông tin độ sáng của mặt S1 được lấy từ bảng đa giác để nhập vào vùng làm tươi.
- Tương tự, các cạnh HE và FG, chỉ cờ cho mặt S2 là on.



https://voer.edu.vn/m/khu-cac-mat-khuat-va-duong-khuat/3367ec74

Ví dụ

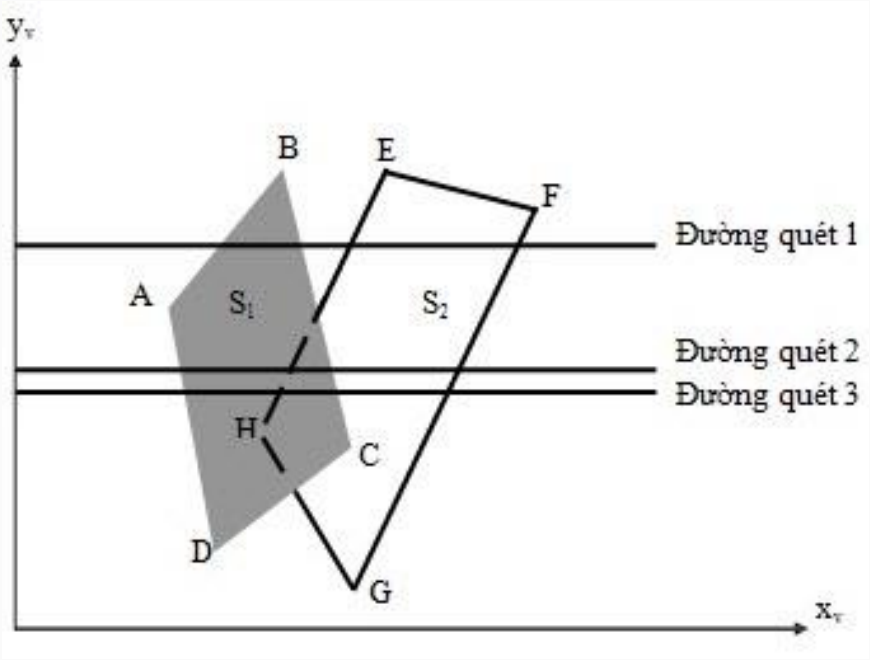
- Dọc theo đường quét 2 từ cạnh DA đến cạnh EH, chỉ cờ của mặt S1 là on.
- Nhưng giữa HE và BC, các cờ cho cả hai mặt là on. Trong đoạn này, các tính toán độ về độ sâu phải được thực hiện bằng cách dùng tham số mặt của các mặt. Trong ví dụ này, độ sâu của mặt S1 được được giả thiết là nhỏ hơn của mặt S2, vì vậy độ sáng của mặt S1 được nạp vào trong vùng đệm làm tươi đến khi biên BC được gặp. Sau đó cờ của mặt S1 trở thành off, và độ sáng của mặt S2 được lưu cho đến cạnh FG được đi qua.



https://voer.edu.vn/m/khu-cac-mat-khuat-va-duong-khuat/3367ec74

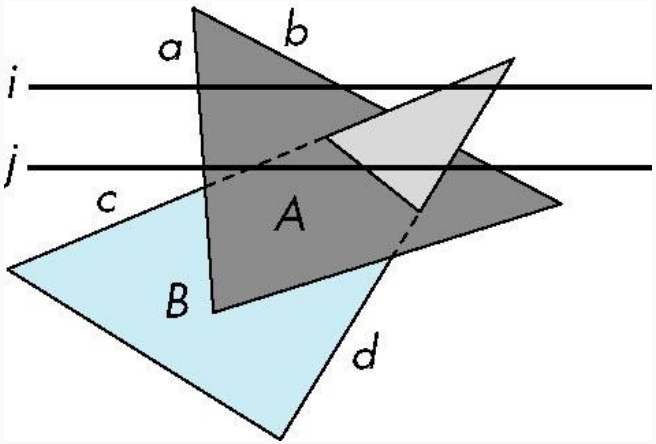
Ví dụ

- Dòng quét 3 có danh sách cạnh giống như của dòng 2.
- Bởi vì không có thay đổi nào xảy ra tại các giao điểm đường, ta không cần tính lại độ sâu giữa các cạnh HE và BC. Hai mặt phải có hướng tương tự như được xác định trên đường quét 2, vì vậy độ sáng cho mặt S1 không cần nhập lại.



https://voer.edu.vn/m/khu-cac-mat-khuat-va-duong-khuat/3367ec74

- Can combine shading and hsr through scan line algorithm

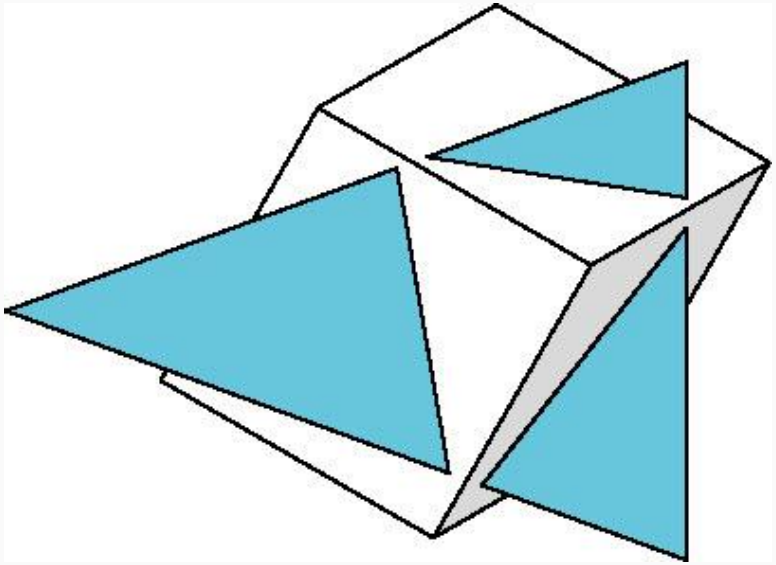


scan line i: no need for depth information, can only be in no or one polygon

scan line j: need depth information only when in more than one polygon

Nhận xét

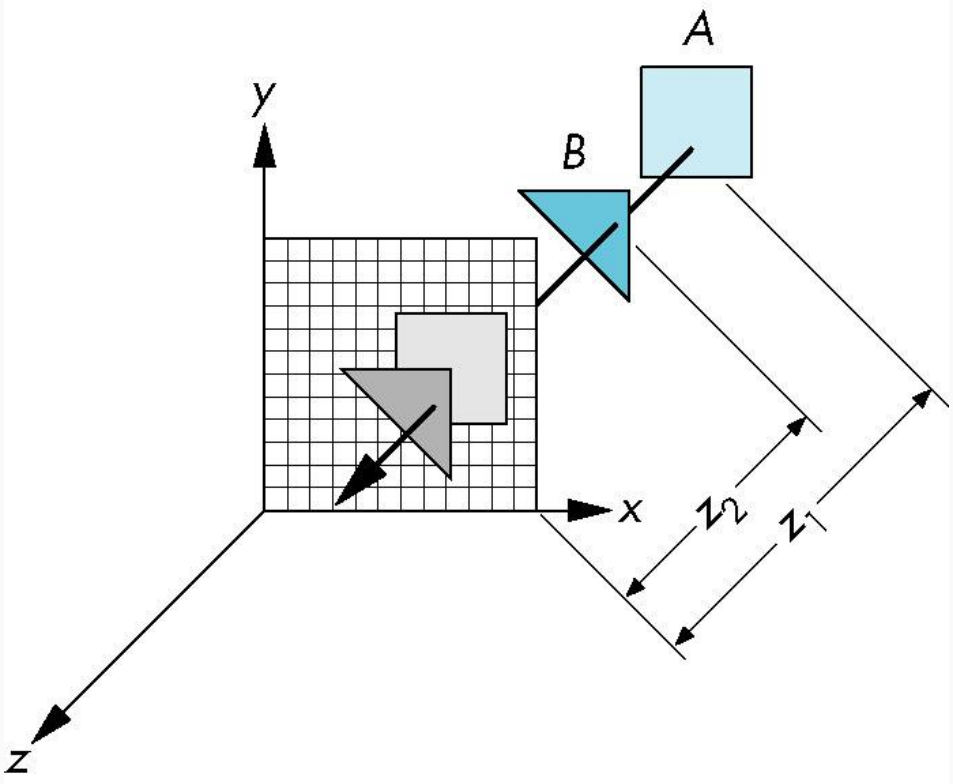
- Cài đặt phức tạp
- Không xử lý trường hợp các đa giác giao cắt nhau



Thuật toán z-buffer

Thuật toán

1. Khởi động z-buffer/ depth buffer
2. Duyệt tuần tự các mặt đa giác: Nếu độ sâu trên mặt đa giác nhỏ hơn độ sâu tương ứng đang lưu trong buffer thì cập nhật lại z-buffer và color-buffer



Thuật toán chi tiết

1. Chuẩn bị 2 buffer (vùng đệm):
   - Depth buffer: lưu độ sâu của mặt (x, y) (mặt được xử lý)

     Depth(x,y) = 0;
   - Refresh buffer: lưu màu của các pixel (của mặt được xét)

     Refresh(x,y) = colorback;
2. Tính độ sâu z cho mỗi vị trí (x, y) trên đa giác.
3. So sánh giá trị độ sâu của pixel thuộc mặt đang xét với giá trị độ sâu đã được lưu trước đo trong depth buffer để kiểm tra khả năng thấy.
   - If (z > depth(x, y)) Then

     Depth(x,y) = z;

     Refresh(x,y) = color(x,y);

Tính độ sâu z

- Pt mặt phẳng: Ax+By=Cz+D=0
- Các giá trị độ sâu cho một vị trí (x, y) được tính từ phương trình của mỗi mặt:

$$z = \frac{-Ax - By - D}{C}$$

- Với mỗi đường quét bất kỳ, các tọa độ x trên cùng đường quét sai khác nhau 1, và các giá trị y giữa hai đường quét cũng sai khác nhau 1. Nếu độ sâu của vị trí (x,y) được xác định là z, khi đó độ sâu z' của vị trí kế tiếp (x+1, y) dọc theo theo đường quét có được như sau:

$$z' = \frac{-A(x+1) - By - D}{C}$$

- Hoặc

$$z' = z - \frac{A}{C}$$

Tỷ số A/C không đổi với mỗi mặt, vì vậy giá trị độ của điểm kế tiếp trên cùng đường quét có được từ giá trị trước đó với một phép trừ.

Tính độ sâu z

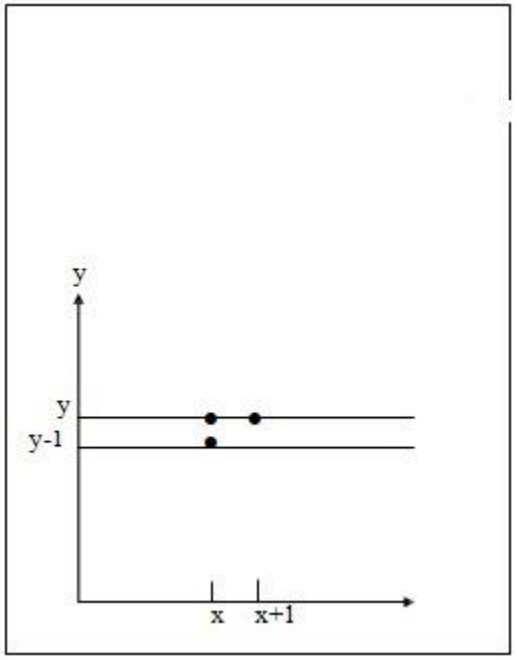
- Chúng ta thu được các giá trị độ sâu giữa các đường quét theo cách tương tự. Một lần nữa giả sử rằng vị trí (x, y) có độ sâu z. Khi đó ở vị trí (x, y-1) trên đường quét ngay bên dưới, giá trị độ sâu được tính từ phương trình mặt phẳng như sau:

$$z'' = \frac{-Ax - B(y-1) - D}{C}$$

- hoặc:

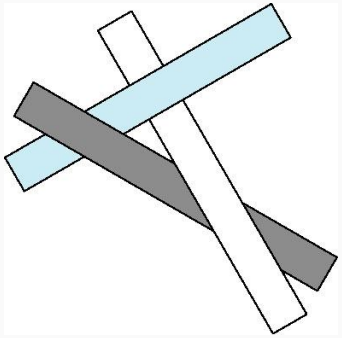
$$z'' = z + \frac{B}{C}$$

ở đây cần một phép cộng hằng B/C với giá trị độ sâu z trước đó.

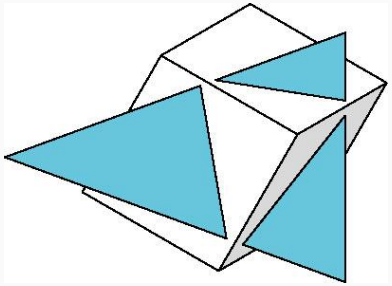


Nhận xét

- Xử lý được vấn đề overlap và penetration
- Đòi hỏi phải có một buffer riêng.
- Dễ cài đặt
- (Không xử lý vấn đề trong suốt (transparency))

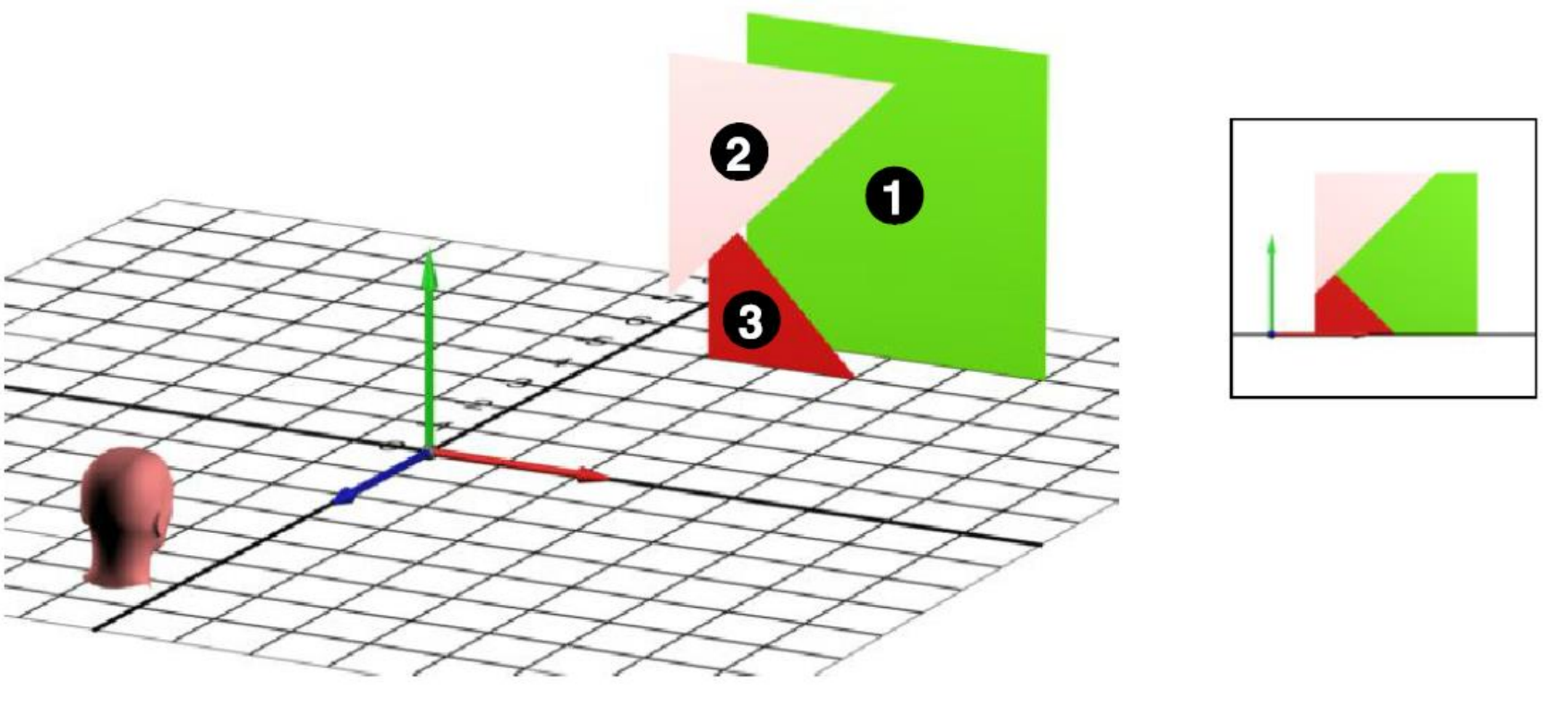


cyclic overlap

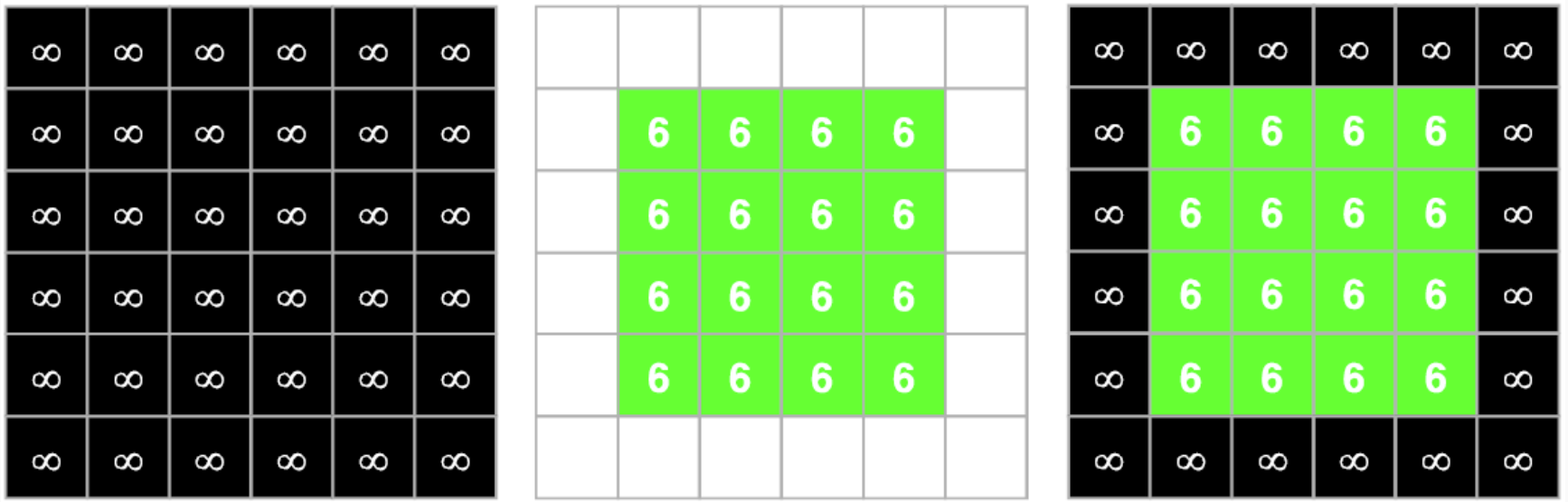


penetration

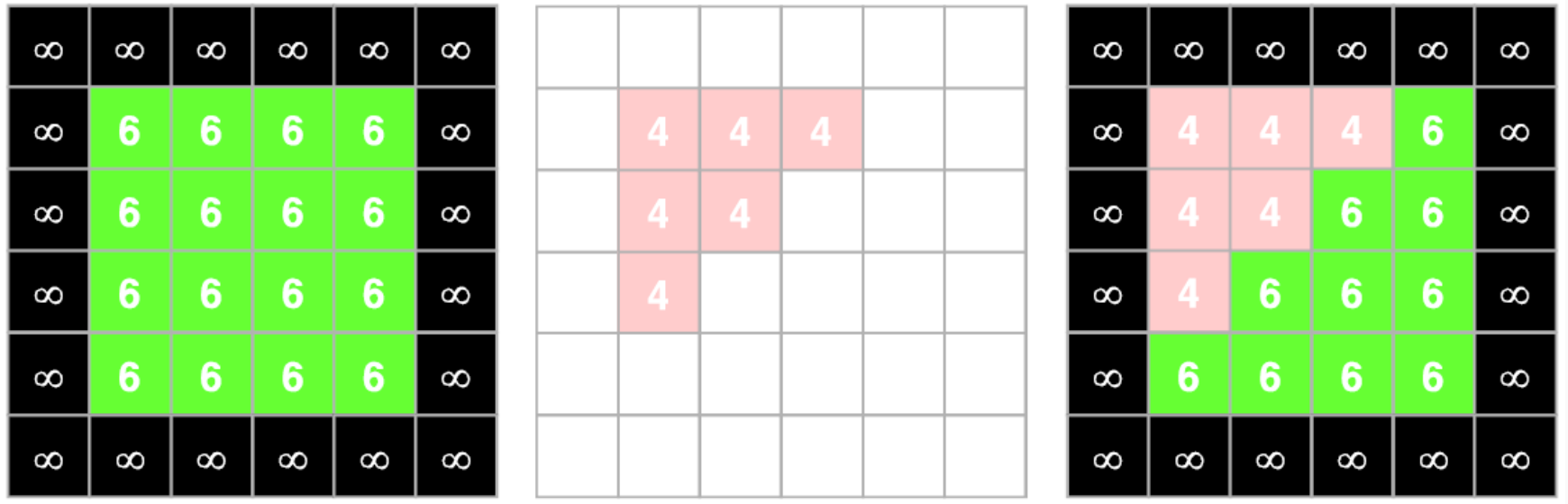
Ví dụ



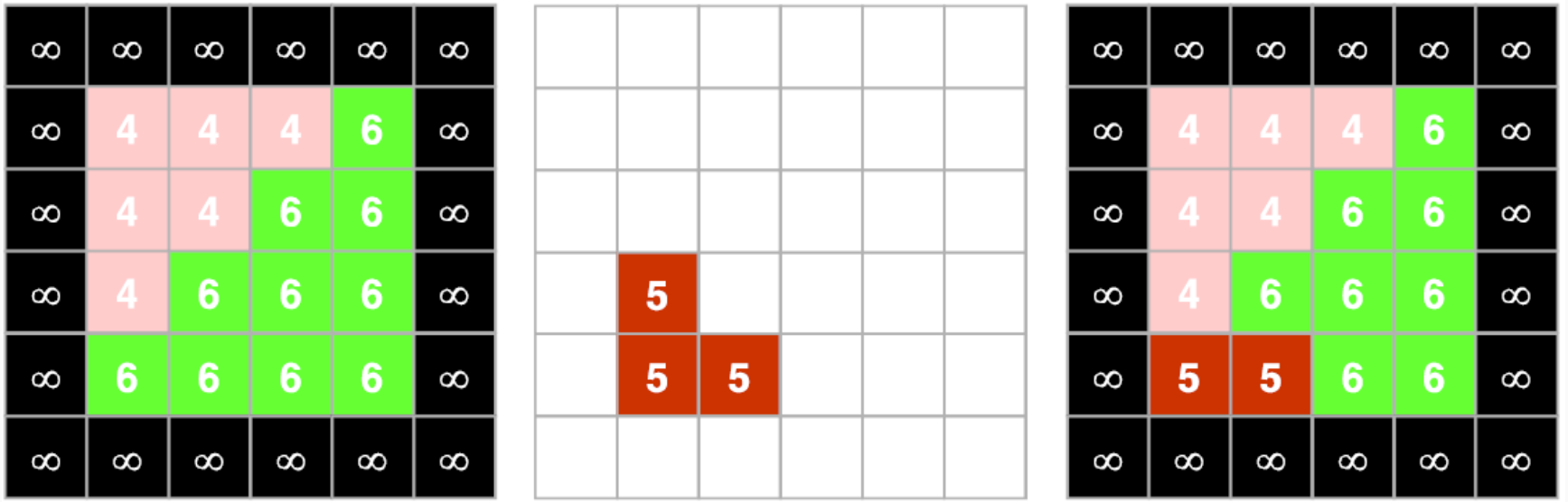
Duyệt đa giác 1



Duyệt đa giác 2



Duyệt đa giác 3



Ưu điểm

- Thích hợp cài đặt trên phần cứng.
- Ta có thể scan-convert các polygon theo thứ tự bất kỳ.
- Mỗi lần ta chỉ phải xét một polygon
- Cho phép tổng hợp nhiều cảnh với nhau hoặc bổ sung các đối tượng mới vào một cảnh phức tạp.
- Có thể áp dụng với các mặt cong, các mặt không có dạng đa giác.

Hạn chế

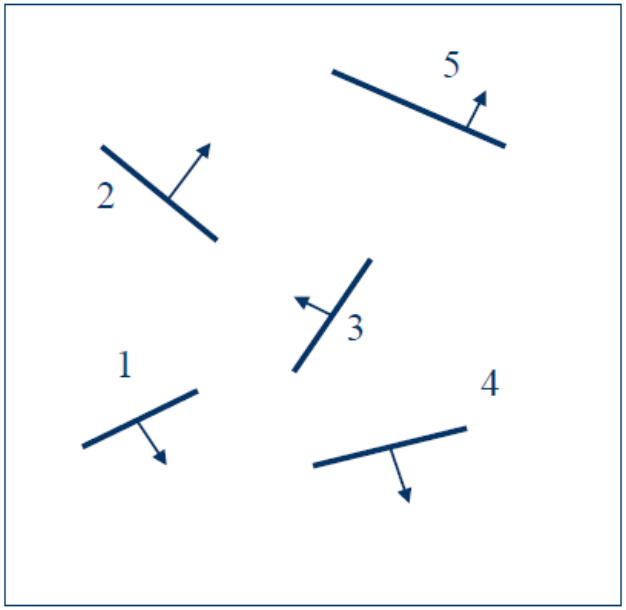
- Đòi hỏi bộ nhớ rất lớn
- Có thể mất chính xác khi chuẩn hoá trong qua trình tính độ sâu.
- Không thực hiện được phép xử lý anti-alias
- Phải scan-convert tất cả các đối tượng.

Thuật toán BSP

(Binary Space Partitioning)

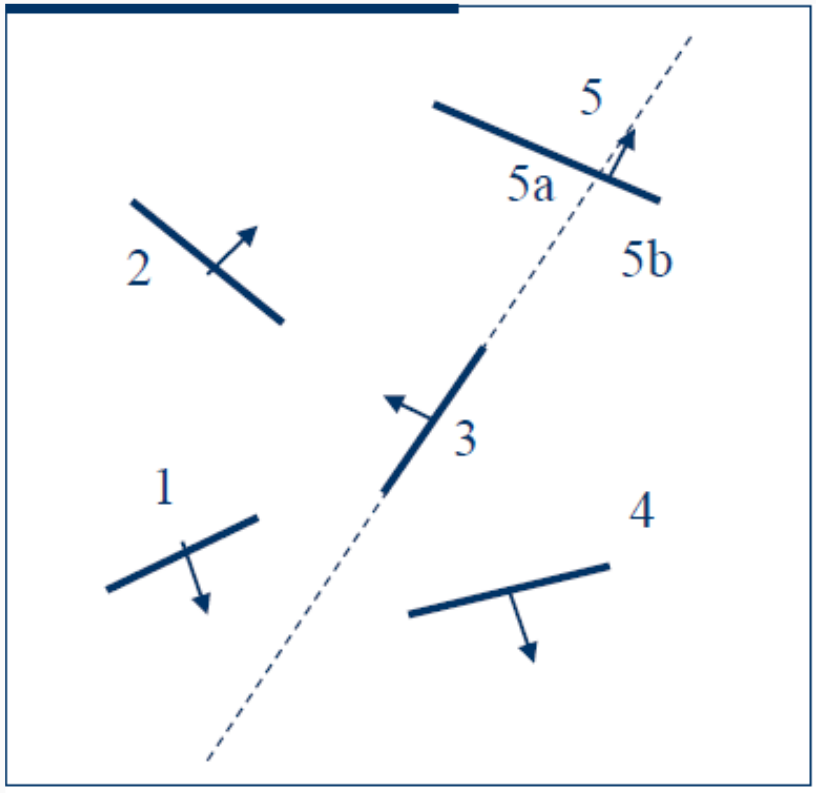
Thuật toán

1. Chuyển danh sách đa giác sang dạng cấu trúc cây nhị phân (cây BSP)
2. Duyệt cây BSP và vẽ các đa giác ra bộ đệm khung theo thứ tự từ sau ra trước



Mặt phẳng phân tách

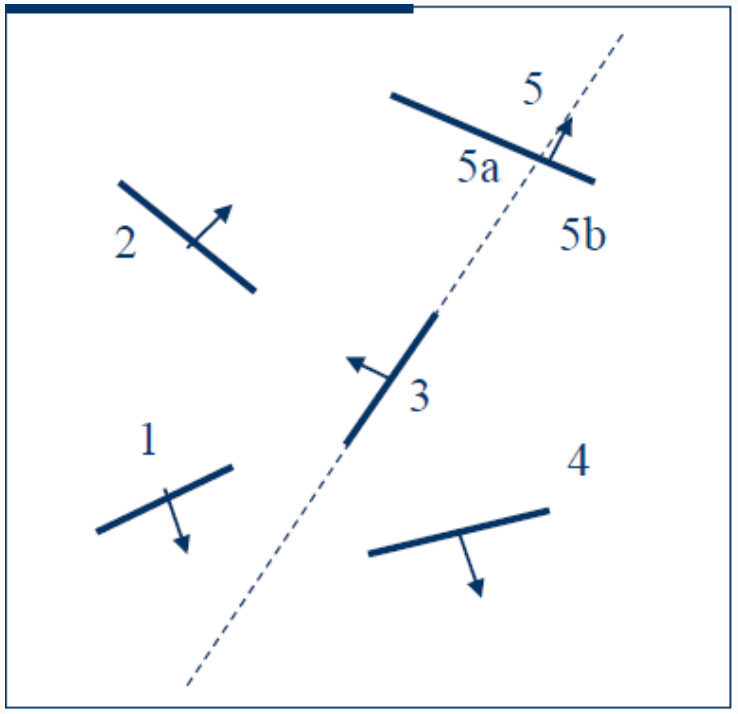
- sao cho không có đa giác nào nằm ở nửa không gian chứa điểm nhìn bị một đa giác nằm ở nửa không gian còn lại che khuất

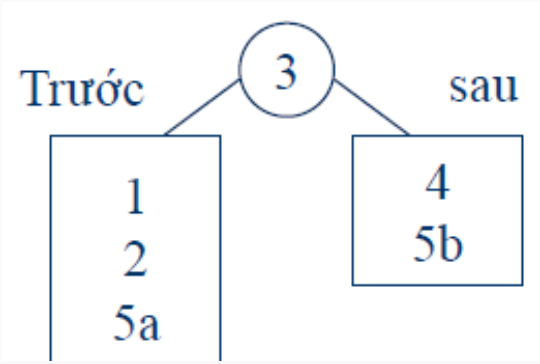


Quá trình phân chia

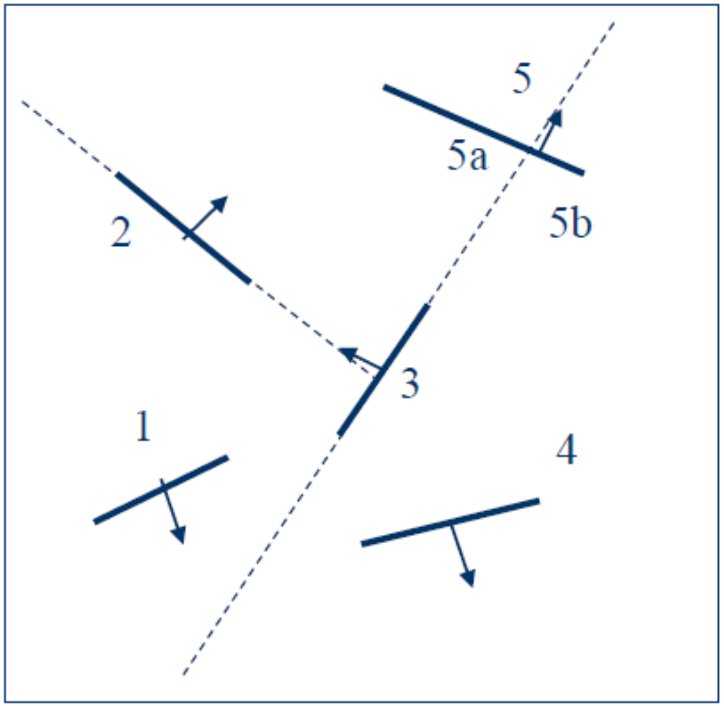
- Chọn đa giác bất kỳ
- Chia cảnh vật ra 2 nửa không gian: trước và sau.
- **Chia những đa giác nằm ở cả hai nửa không gian.**
- Chọn một đa giác ở mỗi nửa – chia đôi cảnh vật tiếp.
- Tiếp tục chia cho đến khi mỗi phần chỉ còn một đa giác.

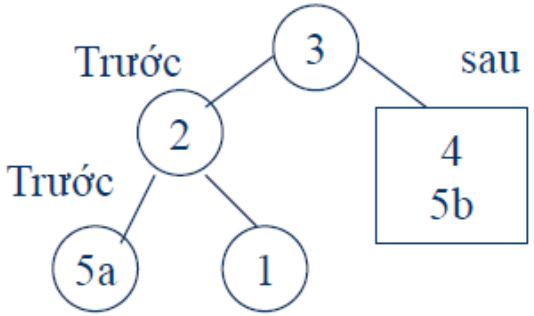
Ví dụ



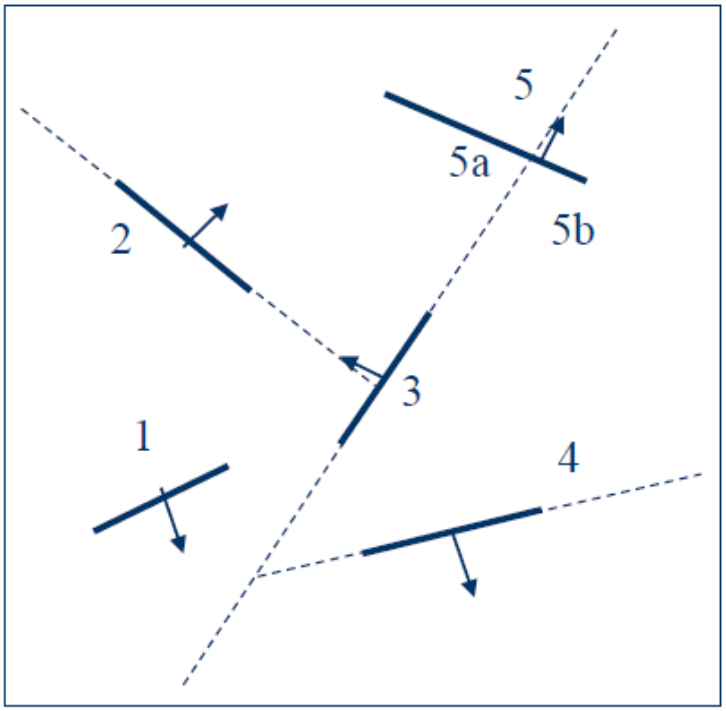


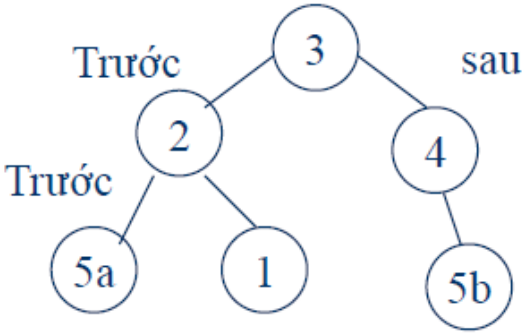
Ví dụ



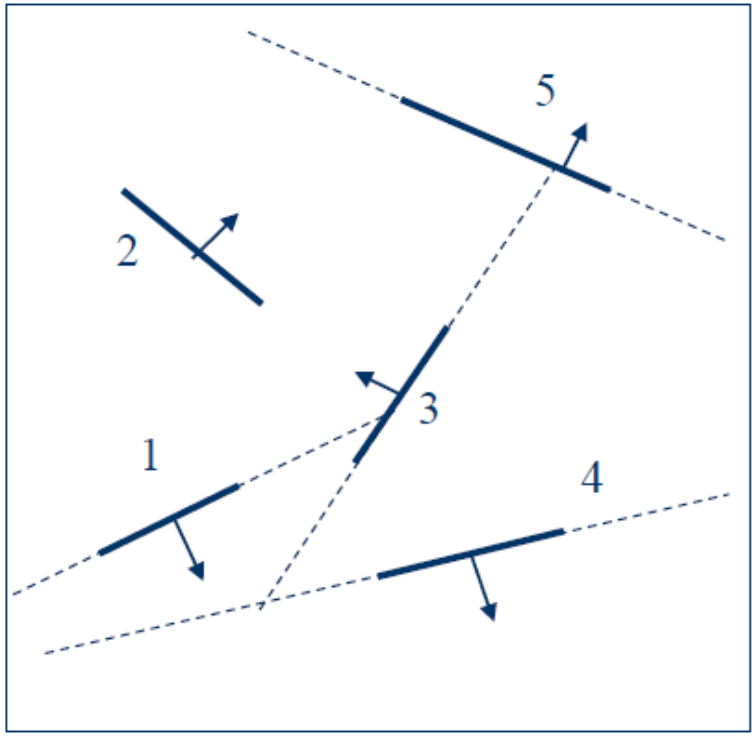


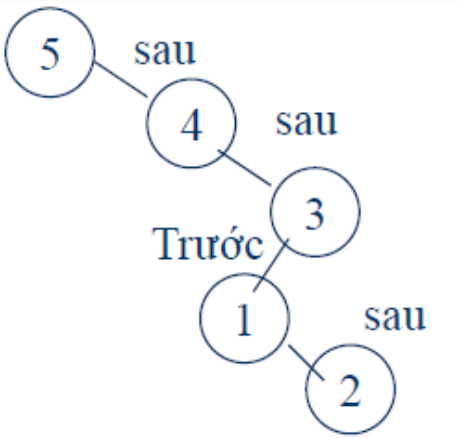
Ví dụ





Ví dụ 2



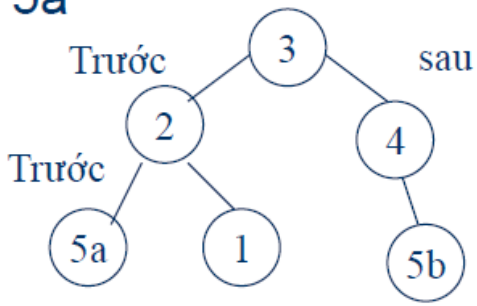


Hiển thị cây BSP

- Cây BSP có thể được duyệt để tạo ra một danh sách ưu tiên cho một góc nhìn bất kỳ
- Ví dụ:

Duyệt cây InOrder(BSP)

5a, 2, 1, 3, 4, 5b → Thứ tự vẽ 5b, 4, 3, 1, 2, 5a



5, 4, 1, 3, 2 →Thứ tự vẽ 2, 3,1, 4, 5