Bài 14
XÓA BỀ MẶT ẨN

Trịnh Thành Trung
trungtt@soict.hust.edu.vn

TỔNG QUAN

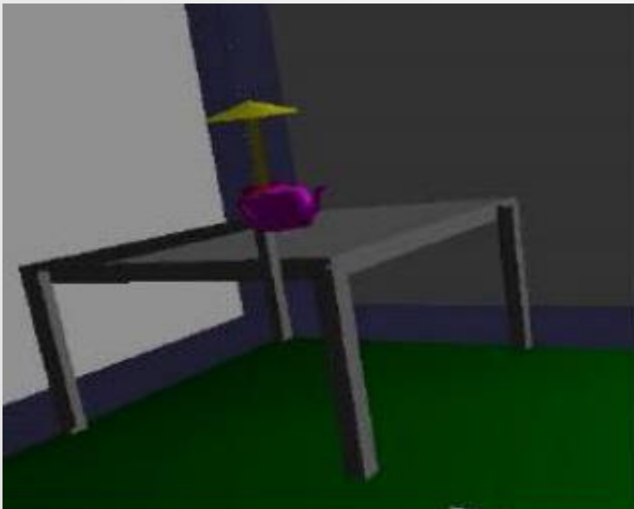
Xóa bề mặt ẩn

- Tại sao cần phải xóa bề mặt ẩn (Hidden surface removal)
  - Các kỹ thuật render đòi hỏi sự tính toán chính xác của khả năng nhìn thấy của đối tượng
  - Khi nhiều đa giác cùng hiển thị trên không gian hiển thị, chỉ có đa giác gần nhất là có thể nhìn thấy được (xóa các bề mặt khác bị ẩn)

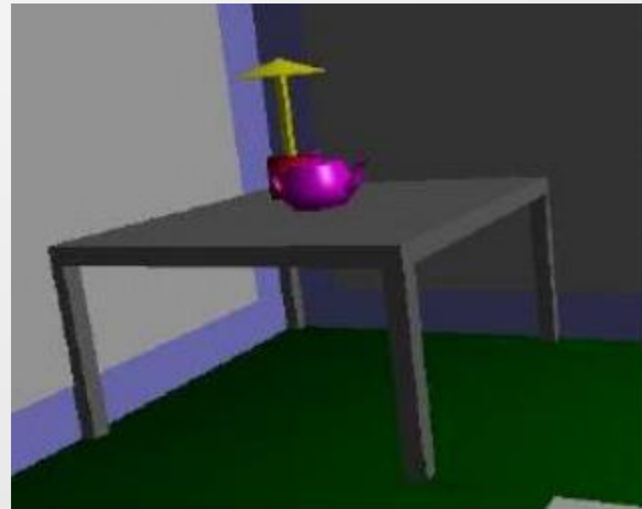
Xóa bề mặt ẩn

- **Ví dụ**

  – Hiển thị sai



  – Hiển thị đúng



Xóa bề mặt ẩn

- Tại sao cần phải xóa bề mặt ẩn
  - Chúng ta không muốn lãng phí các tài nguyên của máy tính để hiển thị các thực thể cơ sở mà không được hiển thị trên ảnh kết quả cuối cùng
    - Ví dụ: Đổ bóng

Xóa bề mặt ẩn

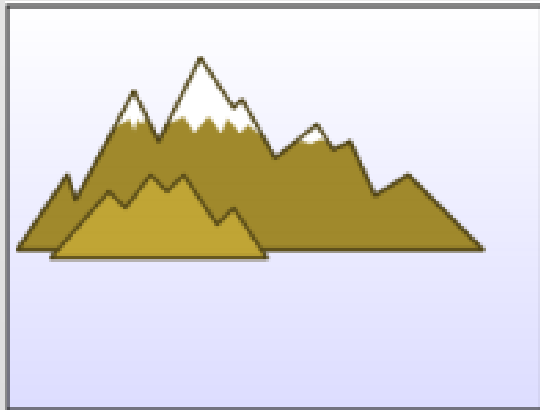
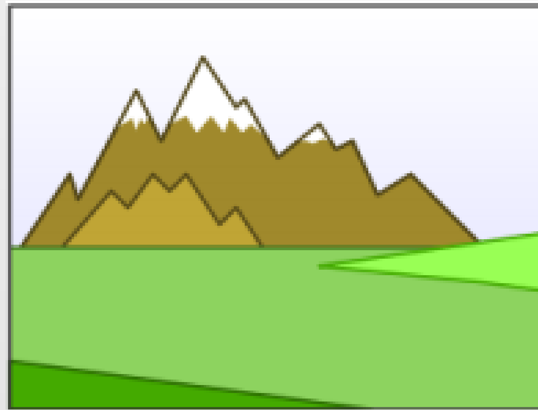
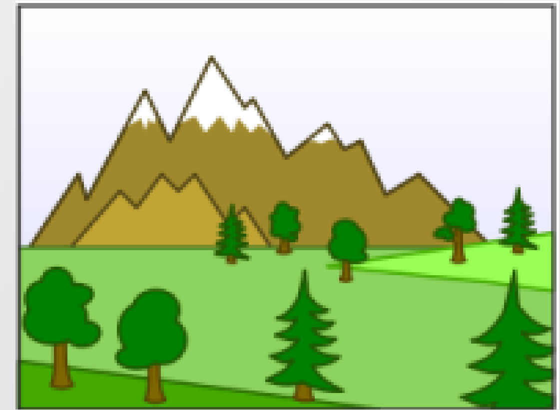
- **Các thuật toán xóa bề mặt ẩn**
  - Painter's algorithm
  - Z-buffer
  - BSP tree
  - Portal culling
  - Một số các thuật toán khác
    - Back face culling

2

THUẬT TOÁN NGƯỜI HỌA SỸ

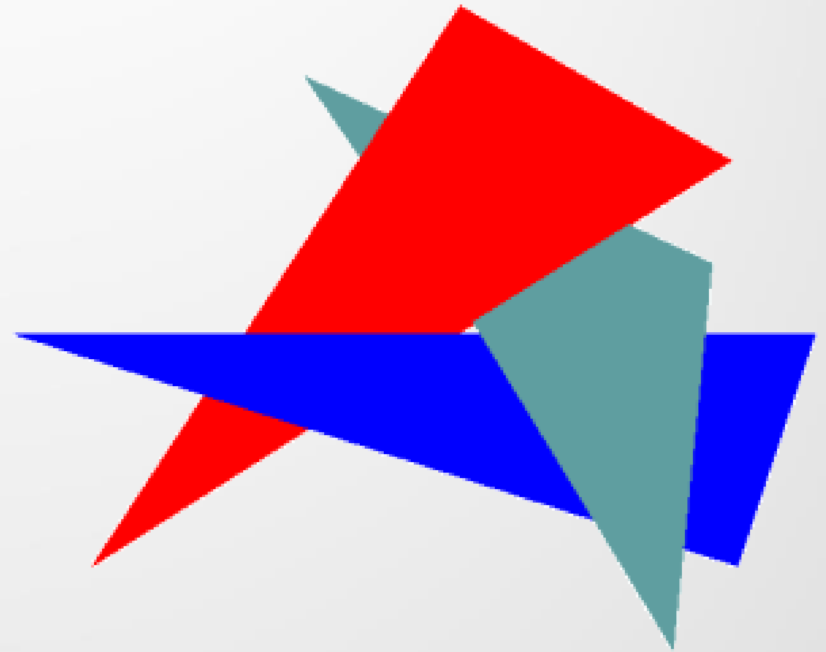
Painter algorithm

- Vẽ các bề mặt lần lượt từ sau ra trước. Những đa giác gần hơn sẽ được *vẽ* đè lên các đa giác ở xa
- Cần phải xác định thứ tự xa gần của các đối tượng.

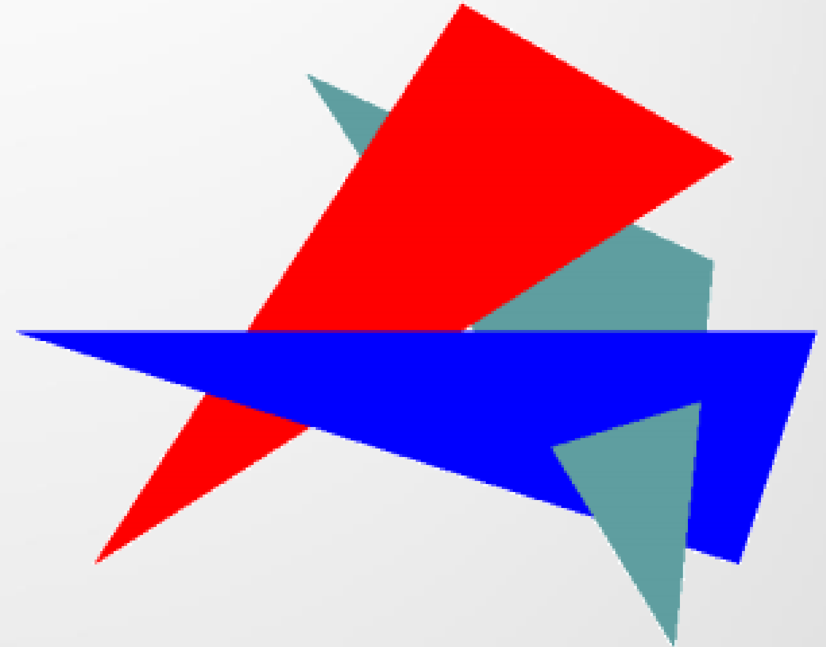
Painter algorithm

- **Một số vấn đề gặp phải**
  - Cần xác định thứ tự của các đối tượng trước khi vẽ
  - Không phải lúc nào cũng có thể vẽ được



Painter algorithm

- **Một số vấn đề gặp phải**
  - Một ví dụ khác về trường hợp không áp dụng được thuật toán
  - Trong cả hai trường hợp, chúng ta cần phải chia nhỏ các tam giác ra để có thể sắp xếp thứ tự



BỘ ĐỆM CHIỀU SÂU Z-BUFFER

Z-buffer

- Z-buffer là phương pháp dựa trên xử lý ảnh áp dụng trong bước rời rạc hóa (rasterization)
- Là phương pháp tiêu chuẩn được áp dụng trong hầu hết các thư viện đồ họa
- Dễ dàng thực thi đối với các phần cứng đồ họa
- Phát minh bởi Wolfgang Straßer năm 1974

Z-buffer

Ý tưởng chính

•Đối với mỗi đa giác đầu vào:

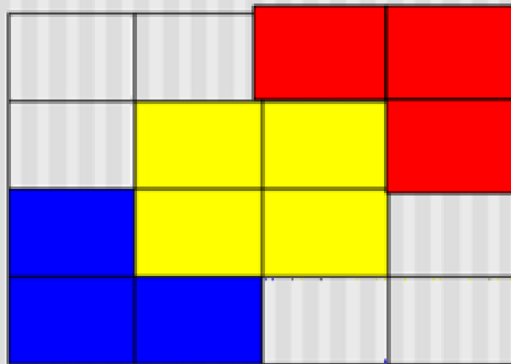
-Đối với từng pixel bên trong đa giác, tính giá trị z tương ứng bằng phương pháp nội suy

-So sánh giá trị độ sâu (depth value) với giá trị gần nhất của đa giác khác (z lớn nhất) đã tính được

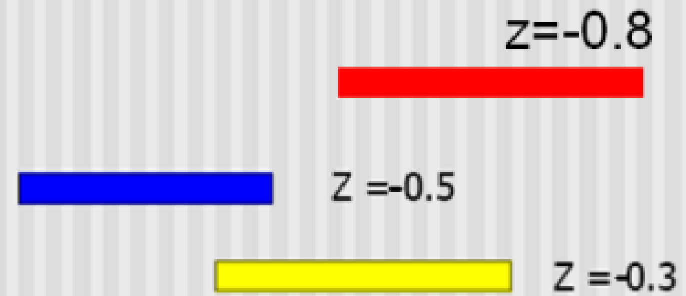
-Vẽ pixel đó với màu của đa giác nếu pixel đó gần hơn

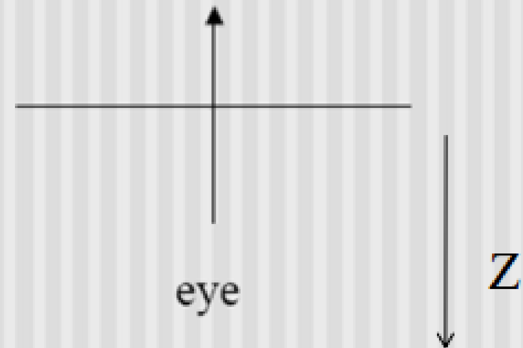
Z-buffer

Z buffer example



Correct Final image





Top View

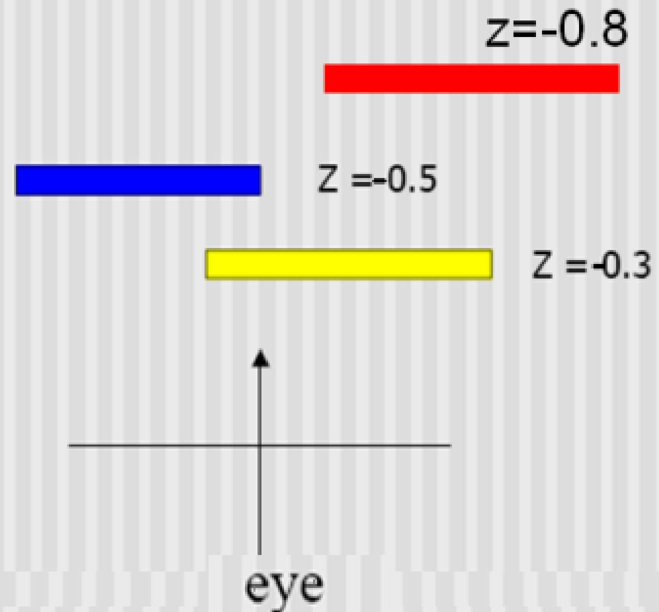
Z-buffer

- Bước 1: Khởi tạo depth buffer

$$\begin{bmatrix} -1.0 & -1.0 & -1.0 & -1.0 \\ -1.0 & -1.0 & -1.0 & -1.0 \\ -1.0 & -1.0 & -1.0 & -1.0 \\ -1.0 & -1.0 & -1.0 & -1.0 \end{bmatrix}$$

Z-buffer

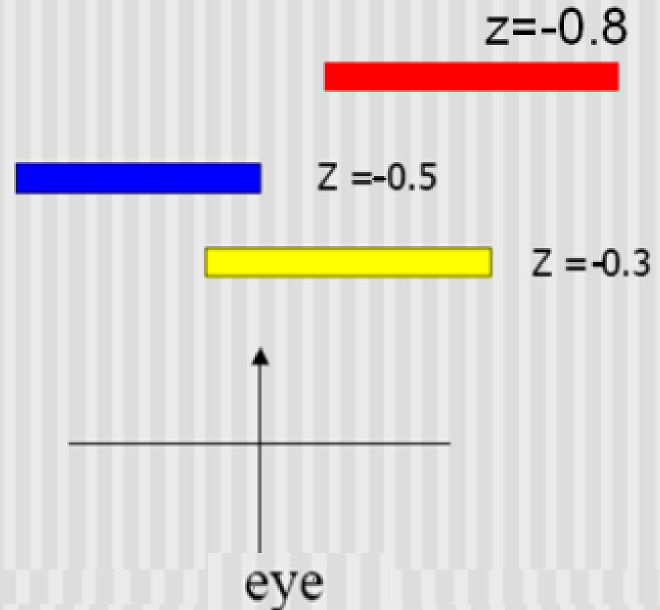
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-0.5	-0.5	-1.0	-1.0
-0.5	-0.5	-1.0	-1.0



- Bước 2: Vẽ đa giác màu xanh (Thứ tự vẽ không ảnh hưởng đến kết quả cuối cùng)

Z-buffer

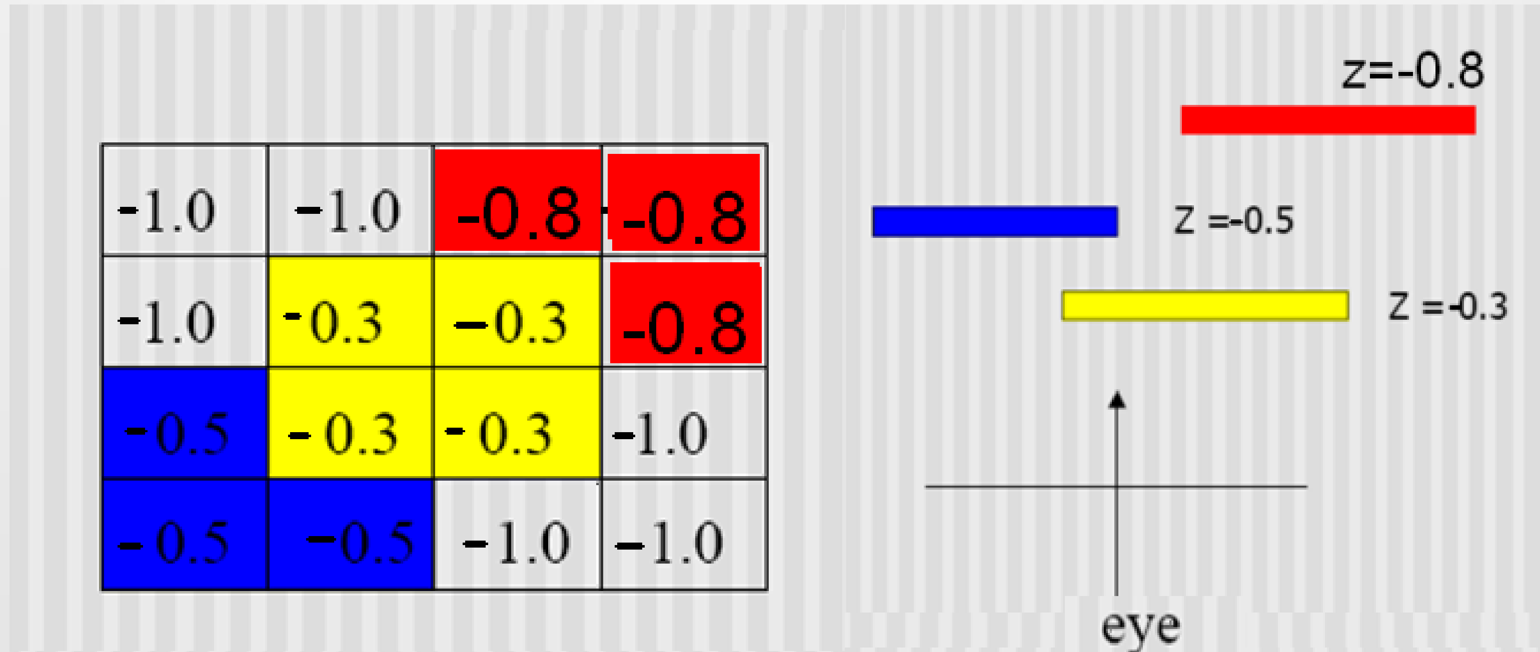
-1.0	-1.0	-1.0	-1.0
-1.0	-0.3	-0.3	-1.0
-0.5	-0.3	-0.3	-1.0
-0.5	-0.5	-1.0	-1.0



- Bước 3: Vẽ đa giác màu vàng

• Nếu giá trị độ sâu lớn hơn giá trị tương ứng của z-buffer, pixel đó được tô màu và giá trị của pixel đó trong z-buffer được cập nhật lại

Z-buffer



- Bước 4: Vẽ đa giác màu đỏ
- Tương tự, nếu giá trị độ sâu lớn hơn giá trị tương ứng của z-buffer, pixel đó được tô màu và giá trị của pixel đó trong z-buffer được cập nhật lại

Z-buffer

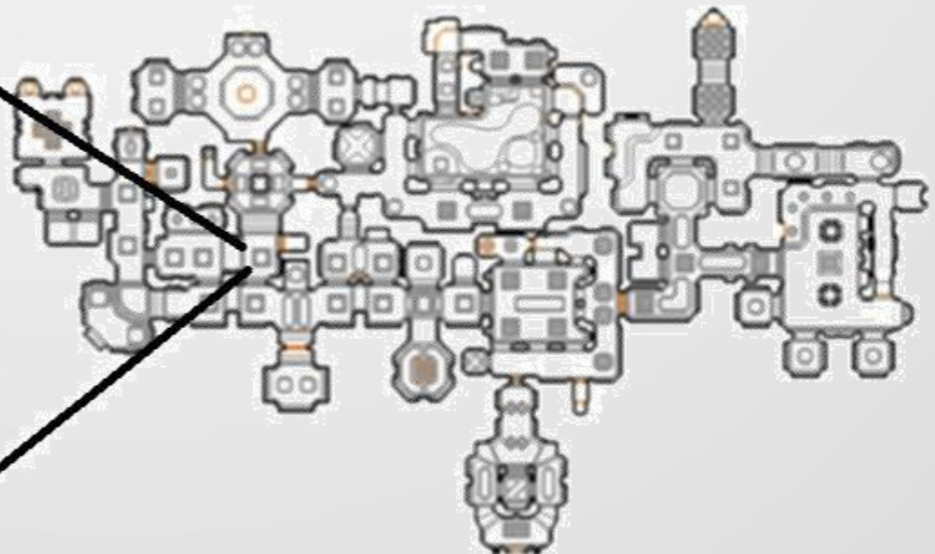
- Advantages
  - Dễ dàng được thực thi bởi các phần cứng đồ họa (Bộ nhớ sử dụng cho z-buffer không còn quá tốn kém)
  - Có thể sử dụng với đa dạng các đối tượng đồ họa, không chỉ là các đa giác
  - Không cần phải sắp xếp các đối tượng
  - Không cần phải tính toán giao điểm giữa đối tượng với đối tượng

Z-buffer

- Disadvantages
  - Tốn thời gian vẽ các đối tượng ẩn
  - Lỗi độ chính xác (z-precision) trong khử răng cưa

- Xét ví dụ sau

Có quá nhiều đa giác phía sau bức tường

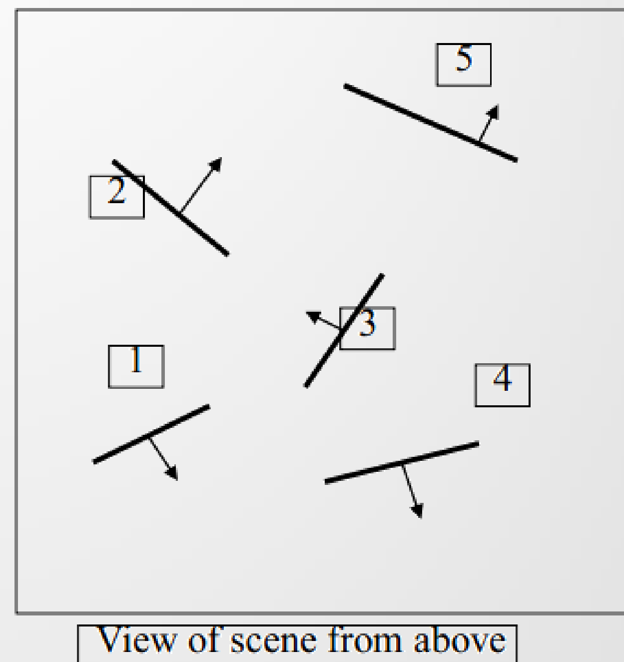


3

CÂY PHÂN VÙNG KHÔNG GIAN NHỊ PHÂN BSP

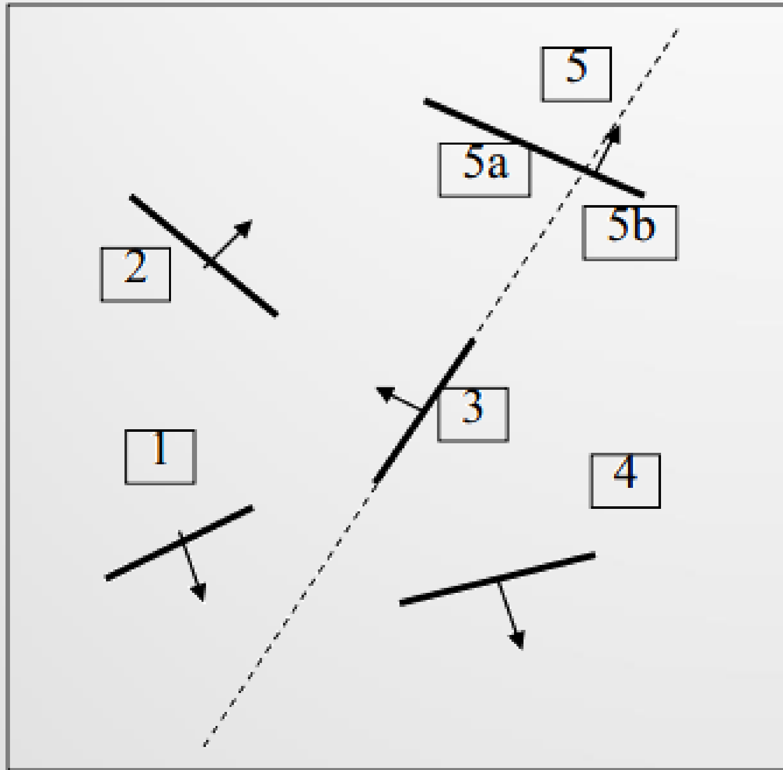
BSP Tree

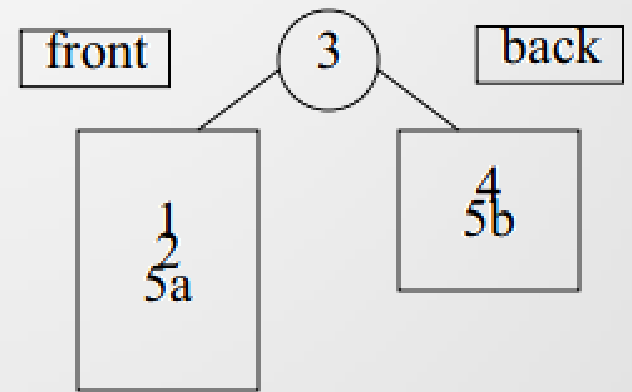
- BSP
(Binary Space Partitioning)



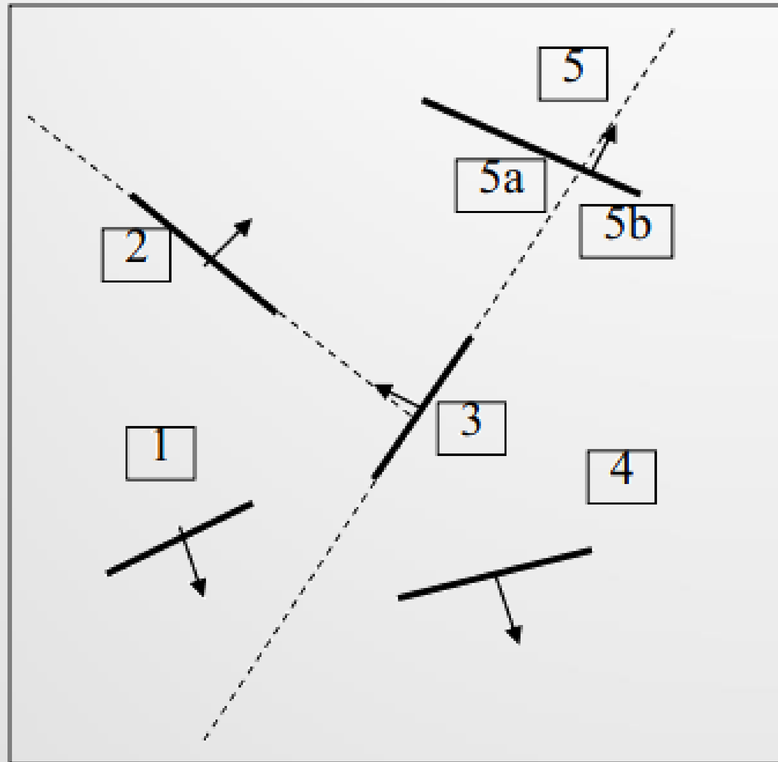
- Ý tưởng
  - Chọn một đa giác tùy ý
  - Chia scene ra làm 2 nửa: Phía trước và phía sau (theo normal vector)
  - Chia đôi bất kỳ đa giác nào nằm trên cả hai nửa
  - Chọn một đa giác ở mỗi bên, thực hiện lại việc chia
  - Thực hiện đệ quy việc chia mỗi nửa cho đến khi mỗi node chỉ chứa 1 đa giác

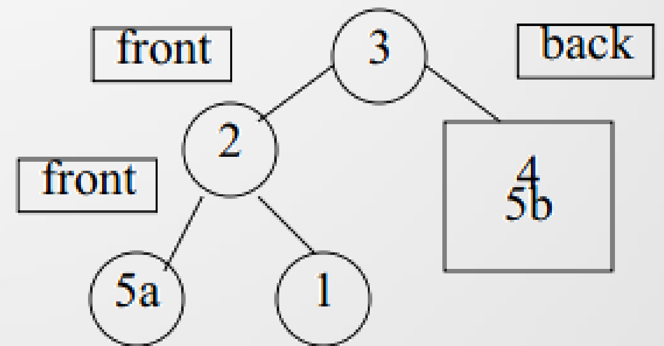
BSP Tree



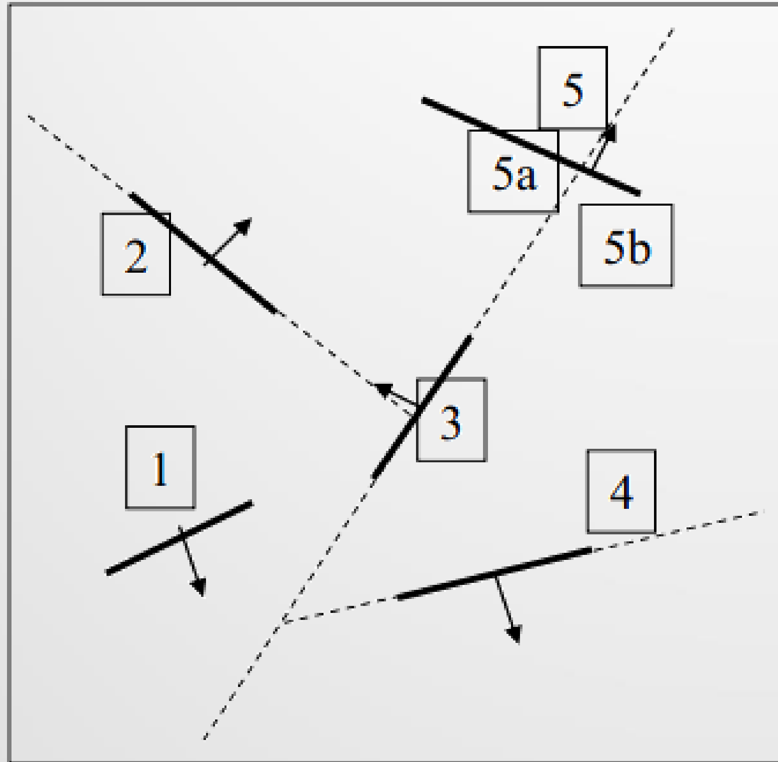


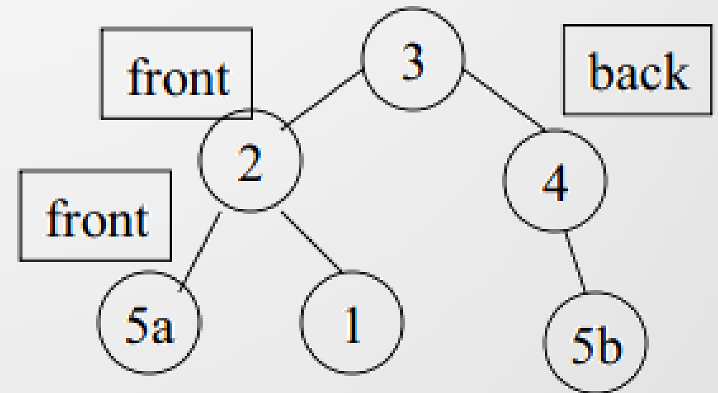
BSP Tree





BSP Tree





Hiển thị cây BSP

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

- Từ sau ra trước (Back-to-front): Tương tự thuật toán painter
- Từ trước đến sau (Front-to-back): Có hiệu quả hơn

Hiển thị cây BSP

Từ sau ra trước

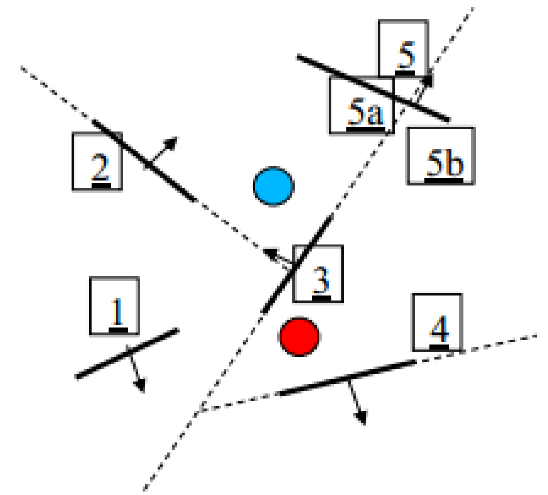
- Bắt đầu ở đa giác gốc
    - Nếu người xem là ở phía trước nửa không gian, vẽ hình đa giác đằng sau đa giác gốc trước, sau đó đến đa giác gốc, sau đó là các đa giác ở phía trước.
    - Nếu người xem là ở phía sau nửa không gian, vẽ hình đa giác đằng sau đa giác gốc trước, sau đó đến đa giác gốc, sau đó là các đa giác ở phía sau
    - Nếu đa giác là trên cạnh, có thể vẽ thế nào cũng được
    - Đệ quy xuống các node ở dưới cây.
- Luôn luôn vẽ ở phía đối nghịch của người nhìn trước

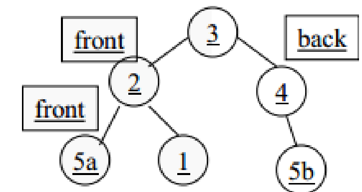
Hiển thị cây BSP

```
traverse_tree(bsp_tree* tree, point eye)
{
  location = tree->find_location(eye);

  if(tree->empty())
    return;

  if(location > 0)        // if eye in front of location
  {
    traverse_tree(tree->back, eye);
    display(tree->polygon_list);
    traverse_tree(tree->front, eye);
  }
  else if(location < 0) // eye behind location
  {
    traverse_tree(tree->front, eye);
    display(tree->polygon_list);
    traverse_tree(tree->back, eye);
  }
  else                    // eye coincidental with partition hyperplane
  {
    traverse_tree(tree->front, eye);
    traverse_tree(tree->back, eye);
  }
}
```





Hiển thị cây BSP

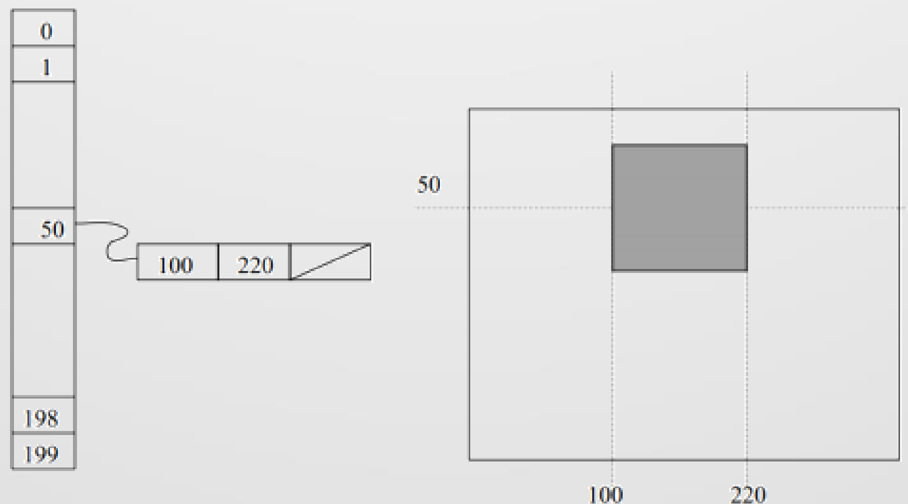
Từ trước đến sau

- Render theo thuật toán từ sau ra trước khiến máy tính phải vẽ lại rất nhiều
- Thuật toán vẽ từ trước đến sau cho hiệu qua cao hơn nhiều
  - Xác định vùng nào đã được vẽ
  - Bỏ qua tất cả các vùng mà đã được vẽ trên màn hình

Hiển thị cây BSP

Từ trước đến sau

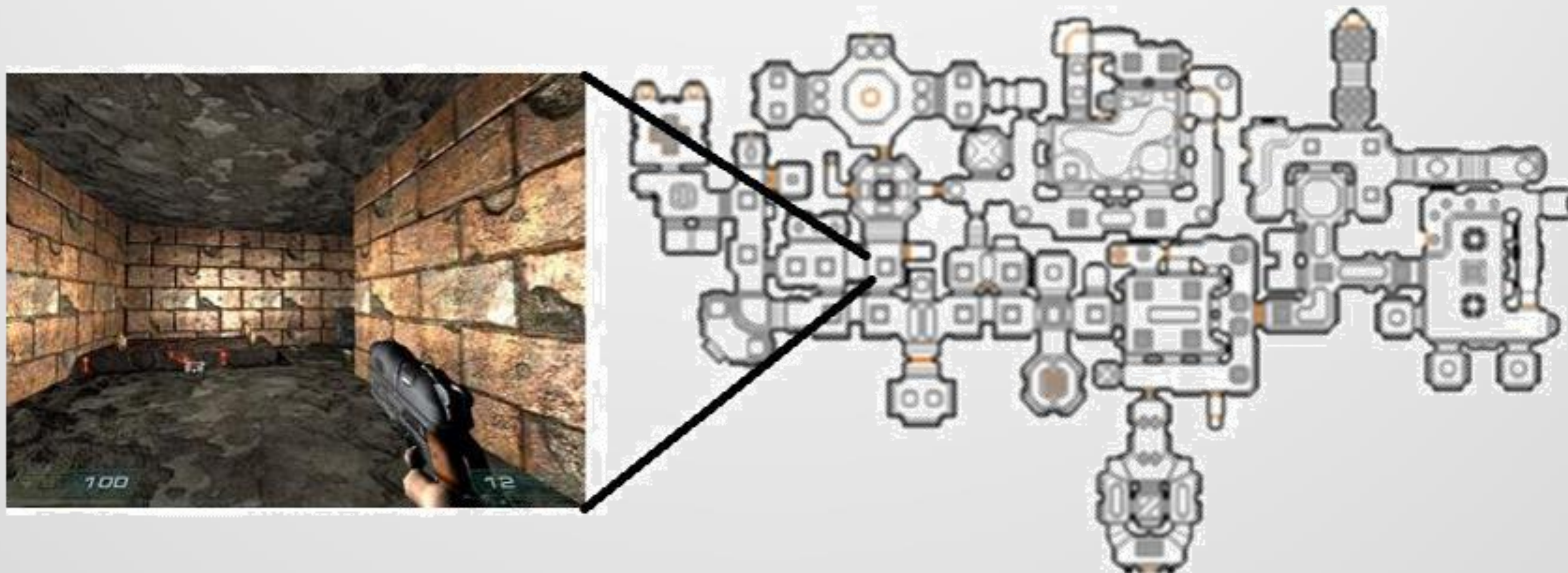
- Để lưu trữ các dữ liệu đã được vẽ, ta sử dụng một Active Edge Table (AET)
- Lưu trữ các pixel đã được hiển thị theo mỗi đường scan line



BSP Tree

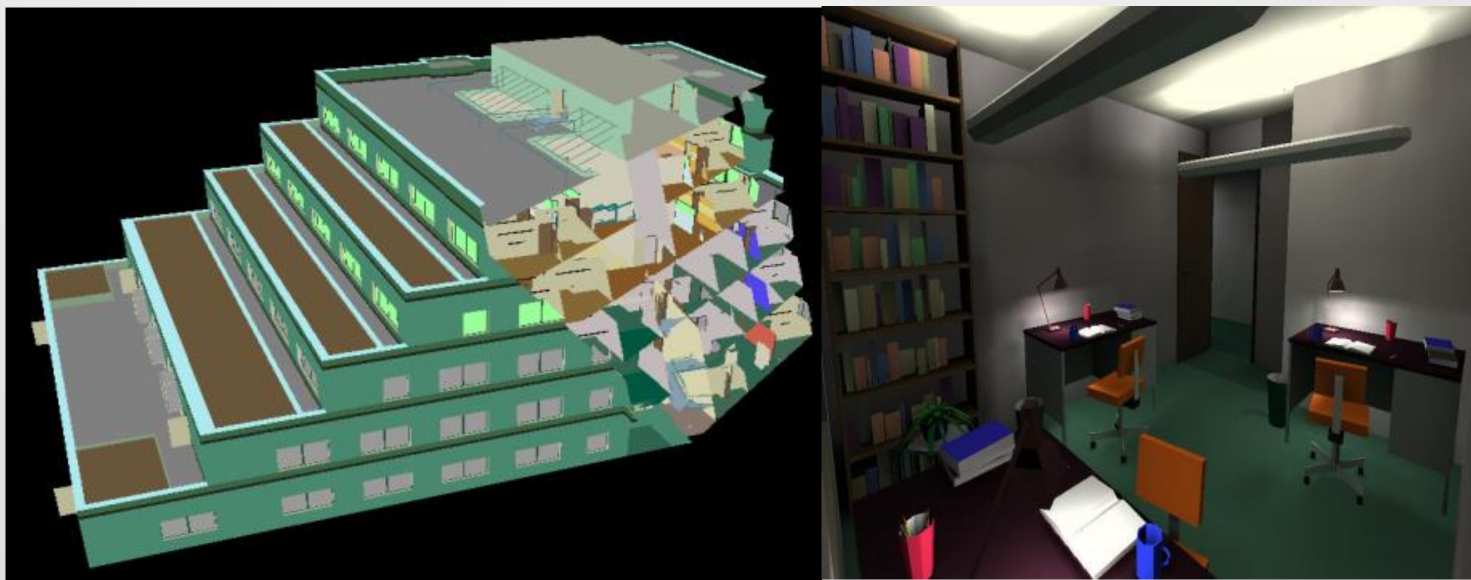
- Đòi hỏi rất nhiều tính toán trước khi vẽ
  - Cần xây dựng một cây cân bằng
  - Tính toán các đa giác giao nhau có thể rất tốn kém tài nguyên
- Dễ dàng tính khả năng hiển thị một khi cây BSP đã được tạo

➢ Hiệu quả đối với các khung cảnh tĩnh mà các đối tượng không thay đổi thường xuyên



BSP Tree

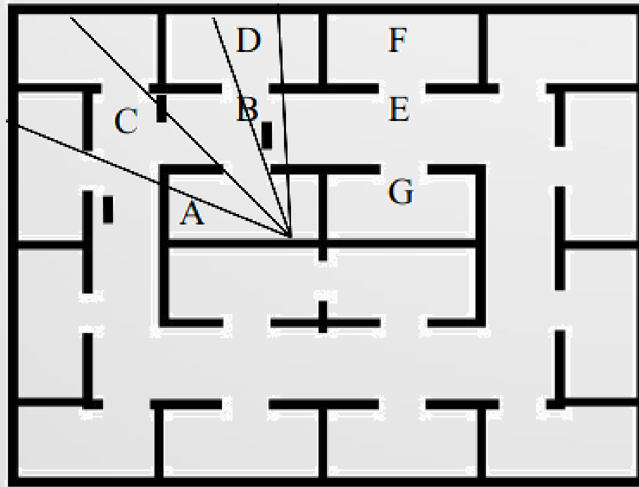
- Thường được kết hợp với z-buffer
- Render các đối tượng tĩnh trước (trước đến sau) với z-buffer
- Sau đó vẽ các đối tượng động (nhân vật, cửa...)

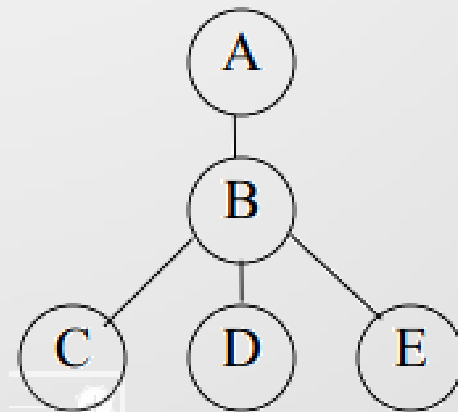


PORTAL CULLING

Portal Culling

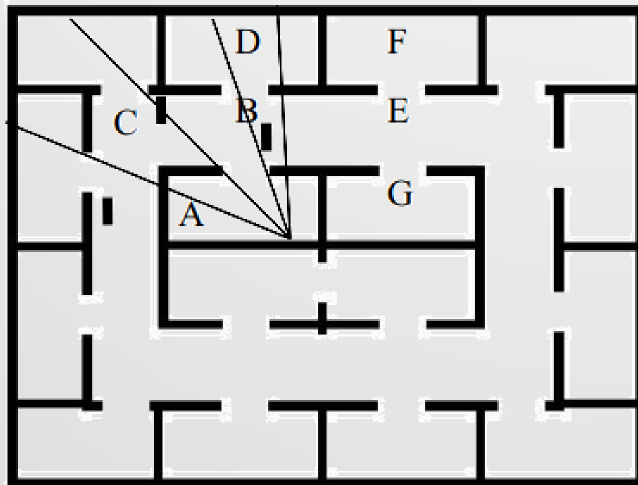
- Xây dựng khung cảnh dưới dạng đồ thị (scene graph)
  - Node: Cells (hoặc rooms)
  - Edge: Portal (hoặc doors)

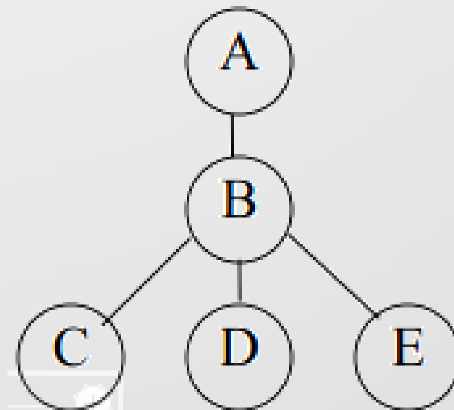




Portal Culling

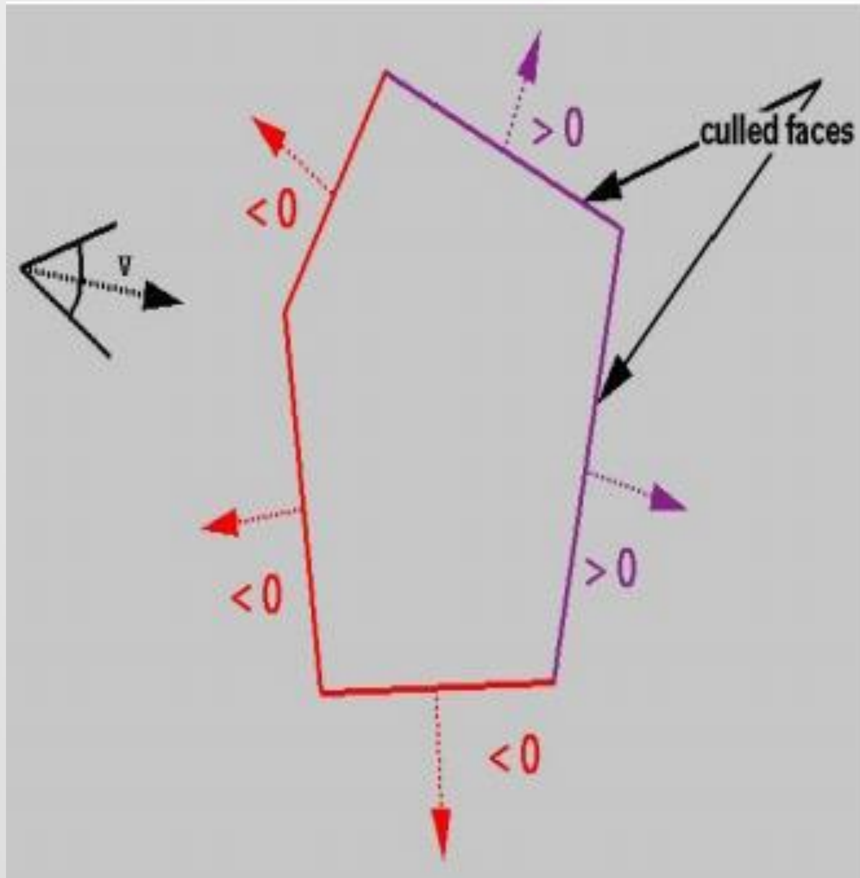
1. Render the room
2. Nếu có thể thấy được portal đến room tiếp theo thì render room tiếp theo trong vùng của portal
3. Lặp lại các bước trên scene graph





BACKFACE CULLING

Back Face Culling



- Không vẽ các đa giác quay mặt hiển hiện thị theo hướng khác góc nhìn
- Kiểm tra thành phần z của normal vector của bề mặt, nếu âm thì bỏ qua (normal vector quay theo hướng ngược lại với góc nhìn)
- Hoặc nếu N.V > 0 thì nghĩa là chúng ta đang nhìn mặt sau của đa giác do đó đa giác sẽ không nhìn thấy được

Phân loại

- Object space technique
  - Được áp dụng trước khi các đỉnh được map vào pixel
  - Ví dụ: Thuật toán painter, Cây BSP, Portal culling
- Image space technique
  - Được áp dụng khi các đỉnh được rời rạc hóa
  - Ví dụ: Z-buffering

Phân loại

- Z-buffer là kỹ thuật rất dễ dàng để thực thi đối với các phần cứng đồ họa, do đó đây là kỹ thuật cơ bản để xóa các bề mặt ẩn
- Thông thường chúng ta cần kết hợp với các kỹ thuật object-based, đặc biệt khi có quá nhiều đa giác. Ví dụ cây BSP hoặc Portal culling