<u>**Introduction**</u>
Visible surface determination

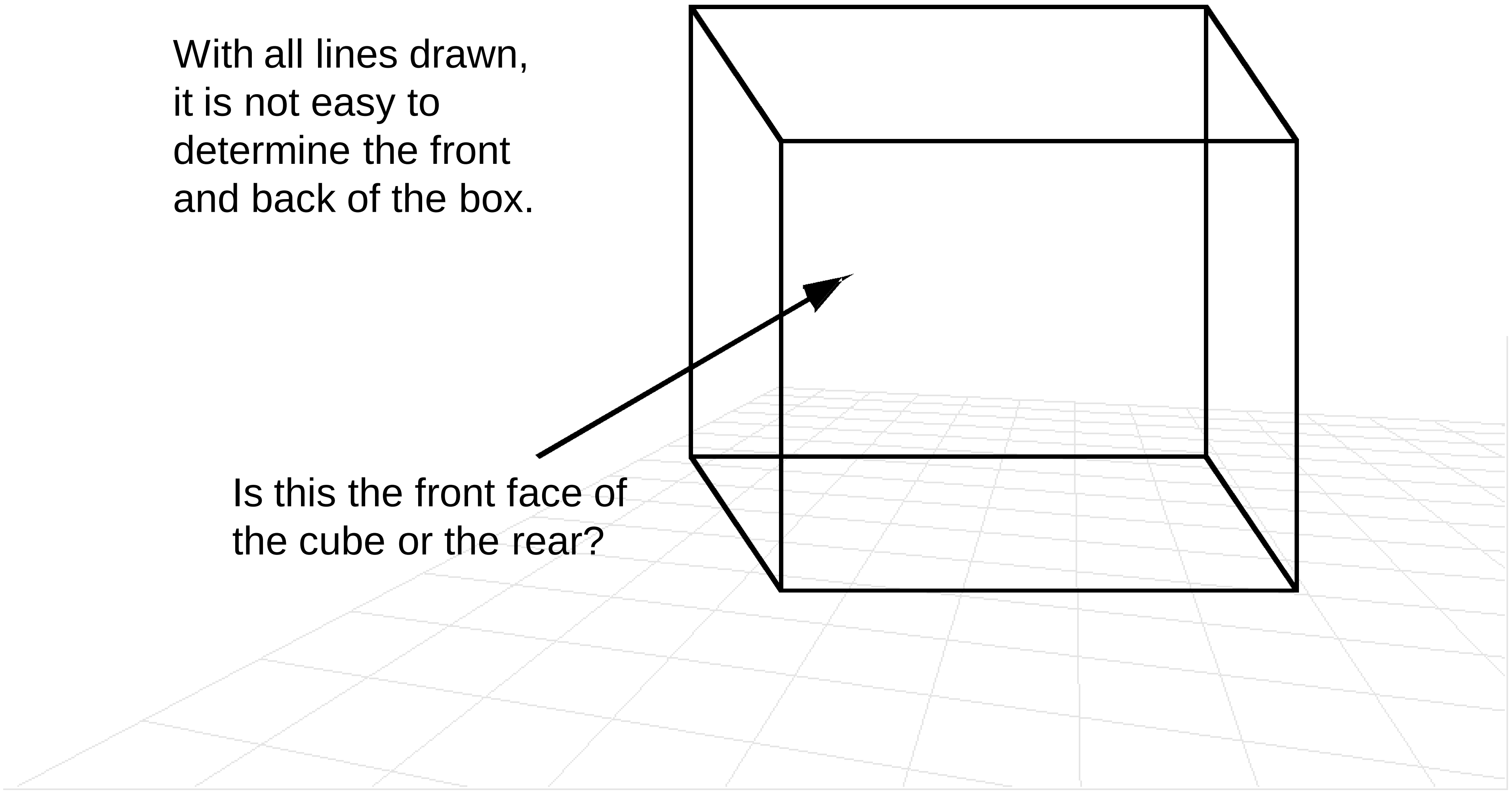
In 3D computer graphics, **hidden surface determination** (also known as **hidden surface removal** (**HSR**), **occlusion culling** (**OC**) or **visible surface determination** (**VSD**)) is the process used to determine which surfaces and parts of surfaces are not visible from a certain viewpoint.

Introduction to Hidden Surface

- Given a set of 3D objects and a viewing specification, we wish to determine which lines or surfaces are visible, so that we do not unnecessary calculate and draw surfces, which will not ultimately be seen by the viewer, or which might confuse the viewer.
- The various algorithms are referred to as visible-surface detection methods.
- Sometimes these methods are also referred to as hidden-surface elimination methods, although there can be differences between identifying visible surfaces and eliminating hidden surfaces.
- we may not want to actually eliminate the hidden surfaces, but rather to display them with dashed boundaries to retain information about their shape.

Simple example…

With all lines drawn, it is not easy to determine the front and back of the box.



Is this the front face of the cube or the rear?

Approaches

- There are 2 fundamental approaches to the problem.
  - Object space
  - Image space

Object space

- An object-space method compare objects and parts of objects to each other within the scene definition to determine which surfaces, as a whole, we should label as visible.

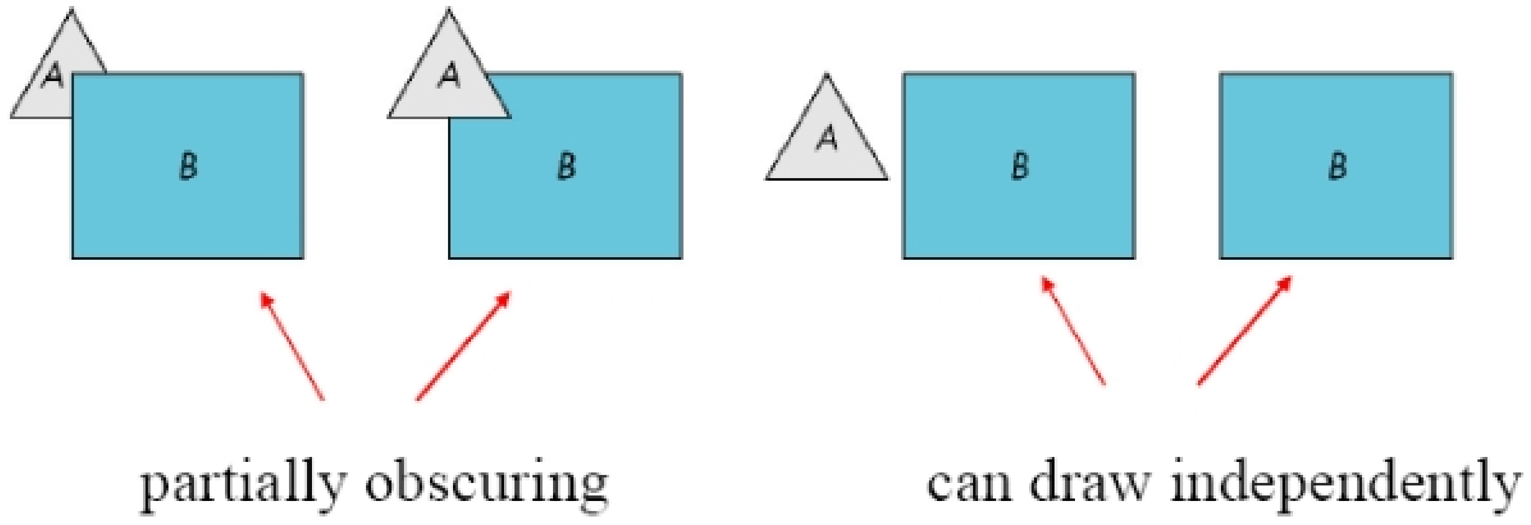
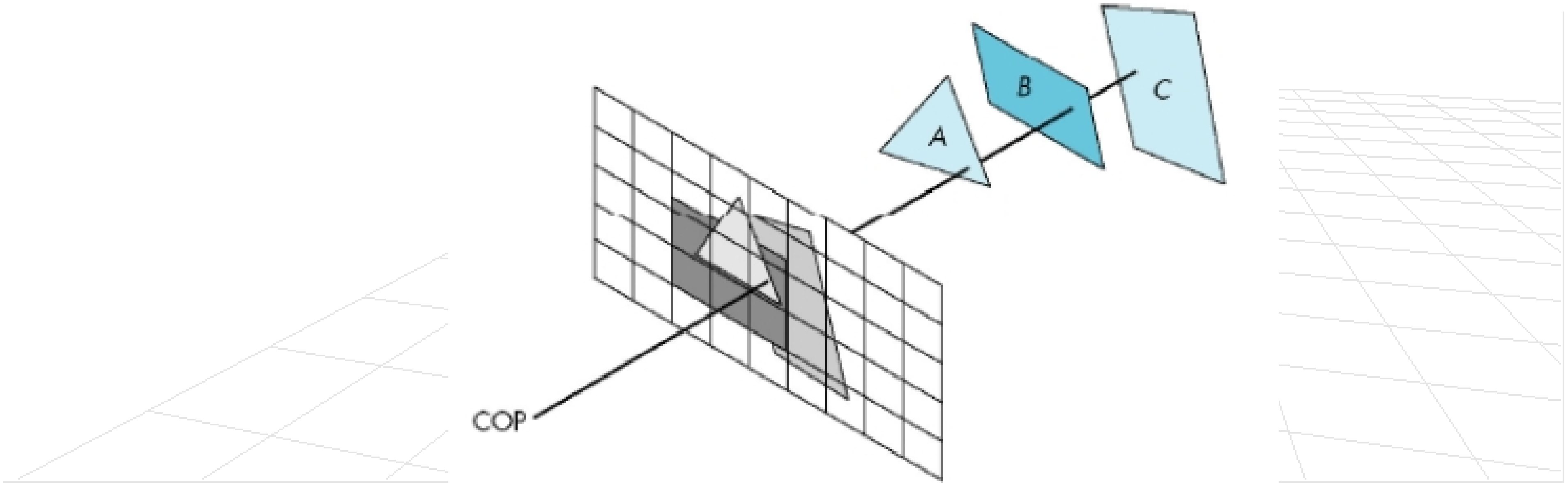


Image space

- In an image space algorithm, visibility is decided point by point at each pixel position on the projection plane.
- Pseudo code…

```
– for each pixel in the frame buffer
    • determine which polygon is closest to the
      viewer at that pixel location
    • colour the pixel with the colour of that
      polygon at that location
```



Algorithms

- Object space
  - Back-face culling
- Image space
  - z-buffer algorithm
  - Scan line algorithm
  - Area subdivision algorithm

Z-buffer algorithm

- It is an image-space approach to detecting visible surfaces. it is also called depth-buffer method.
- It compares surface depths at each pixel position on the projection plane.
- Each surface of a scene is processed separately, one point at a time across the surface.
- The method is usually applied to scenes containing only polygon surfaces, because depth values can be computed very quickly **and** the method is easy to implement. But the method can be applied to nonplanar surfaces also.

Figure shows three surfaces at varying distances along the orthographic projection line from position (x,y) in a view plane.. Surface s1, is closest at this position, so its surface intensity value at **(x,** y) is saved.

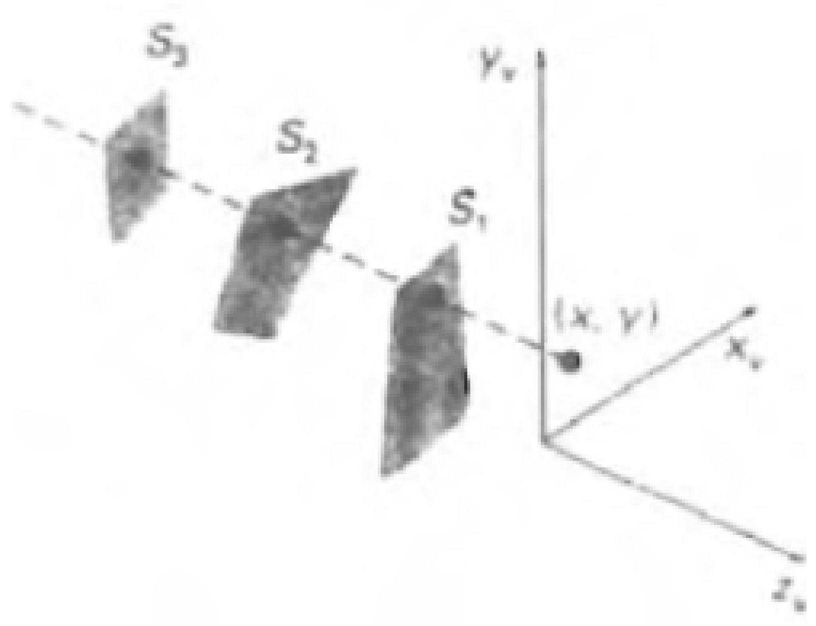


Figure 13-4
At view-plane position (x, y), surface S_1 has the smallest depth from the view plane and so is visible at that position.

1. Initialize the depth buffer and refresh buffer so that for all buffer positions (x, y),

$$\text{depth}(x, y) = 0, \qquad \text{refresh}(x, y) = I_{\text{backgnd}}$$

2. For each position on each polygon surface, compare depth values to previously stored values in the depth buffer to determine visibility.

   - Calculate the depth z for each (x, y) position on the polygon.
   - If $z > \text{depth}(x, y)$, then set

$$\text{depth}(x, y) = z, \qquad \text{refresh}(x, y) = I_{\text{surf}}(x,y)$$

where I_{backgnd} is the value for the background intensity, and $I_{\text{surf}}(x,y)$ is the projected intensity value for the surface at pixel position (x,y). After all surfaces have been processed, the depth buffer contains depth values for the visible surfaces and the refresh buffer contains the corresponding intensity values for those surfaces.

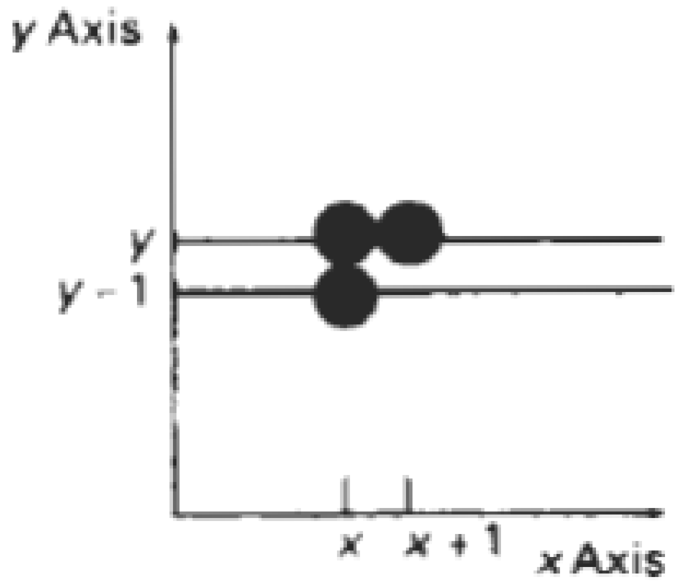


Figure : **From** position (x, **y)** on a scan line, the next position across the line has coordinates ***(X*** **+** 1, **y),** and the position immediately below on the next line has coordinates ***(x,*** y - **1).**

Depth values for a surface position ***(x, y)*** are calculated from the plane equation for each surface:

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

For any scan line in figure adjacent horizontal positions across the line differ by 1, and a vertical y value on an adjacent scan line differs by 1. If the depth of position ***(x,*** y) has been determined to be z, then the depth ***z'*** of the next position (x **+** 1, y) along the scan line is obtained from as

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

or

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

The ratio ***-A/C*** is constant for each surface, so succeeding depth values across a scan line are obtained from preceding values with a single addition.

On each scan line, calculate the depth on a left edge of the polygon that intersects that scan line in Fig. below. Depth values can be calculated by previous equation.

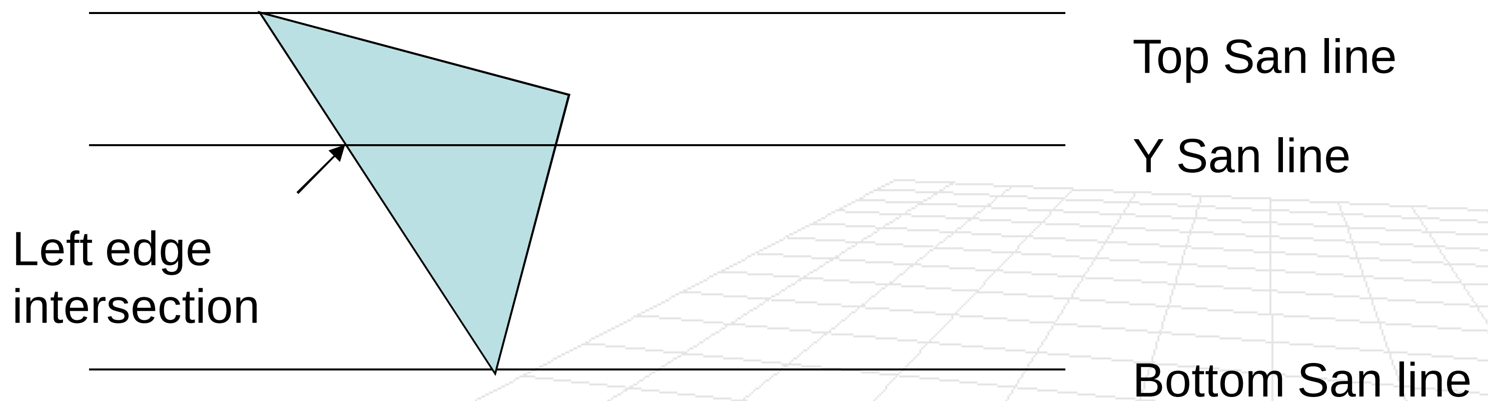


Fig. 13.6

We first determine the y-coordinate extents of each polygon, and process the surface from the topmost scan line to the bottom scan line, as shown in Fig. 13-6.

Starting at a top vertex, we can recursively calculate x positions down a left edge of the polygon as $x' = x - 1/m$, where m is the slope of the edge (Fig. 13-7).

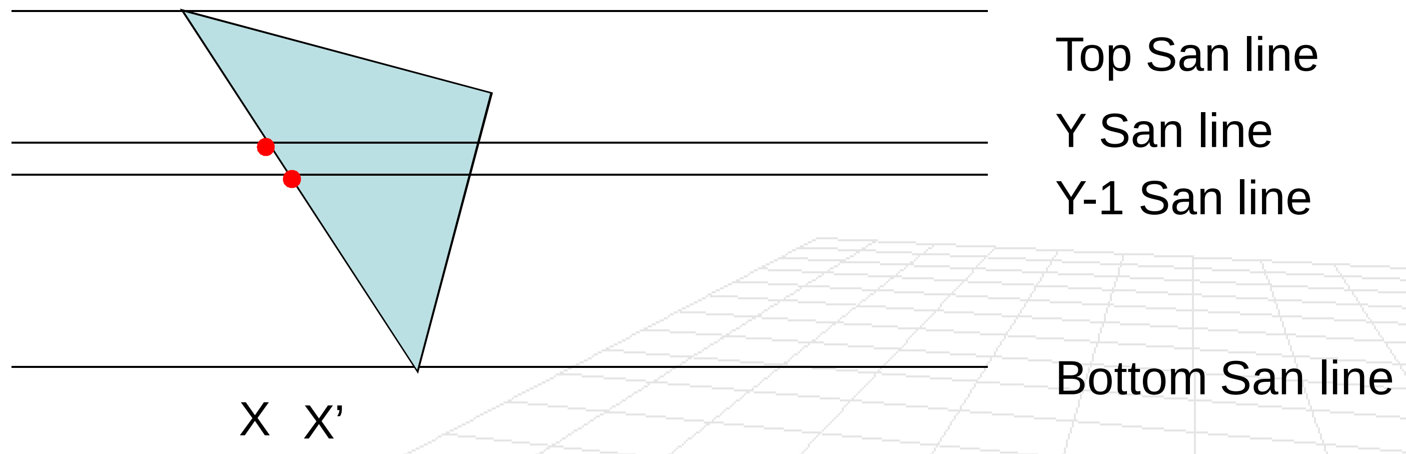


Figure 13-7: Intersection positions on successive scan Lines along a left polygon edge.

Depth values down the edge are then obtained recursively as

$$z' = z + \frac{A/m + B}{C} \qquad (13\text{-}7)$$

If we are processing down a vertical edge, the slope is infinite and the recursive calculations reduce to

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

Application

A hidden surface determination algorithm is a solution to the visibility problem, which was one of the first major problems in the field of 3D computer graphics. The process of hidden surface determination is sometimes called **hiding**, and such an algorithm is sometimes called a **hider**. The analogue for line rendering is hidden line removal. Hidden surface determination is necessary to render an image correctly, so that one cannot look through walls in virtual reality.

Scope of Research

The **irregular Z-buffer** is an algorithm designed to solve the visibility problem in real-time 3-d computer graphics. It is related to the classical Z-buffer in that it maintains a depth value for each image sample and uses these to determine which geometric elements of a scene are visible. The key difference, however, between the classical Z-buffer and the irregular Z-buffer is that the latter allows arbitrary placement of image samples in the image plane, whereas the former requires samples to be arranged in a regular grid.