VISIBILITY ALGORITHM

In computer graphics, visibility algorithms are crucial for determining which objects or parts of objects in a 3D scene are visible from a particular viewpoint. These algorithms help optimize rendering by ensuring that only the visible objects are processed, thus saving computational resources. There are several visibility algorithms, each with its own strengths and weaknesses. Some of the key visibility algorithms are

Painter's Algorithm

- The Painter's Algorithm is one of the simplest visibility algorithms. It sorts all objects in the scene based on their distance from the viewer and renders them from back to front. This approach works well for scenes with non-overlapping objects but fails when objects intersect.
- It is computationally straightforward but suffers from the "sorting problem," where determining the correct order for all objects can be expensive.

Painter Algorithm

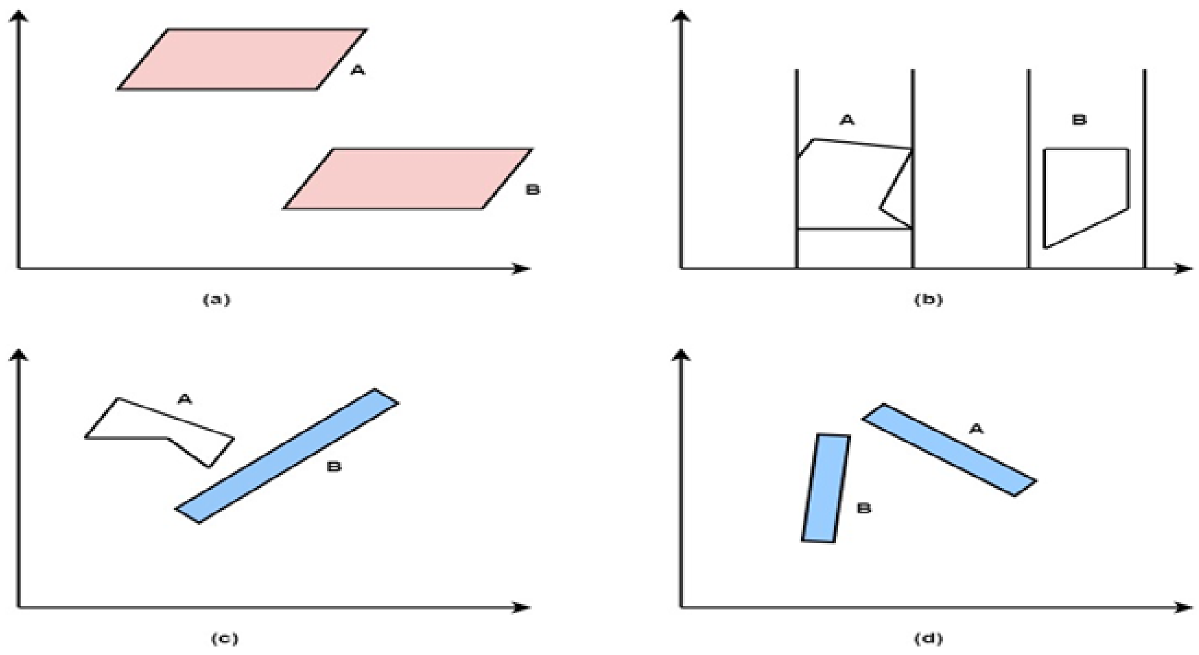
Step1: Start Algorithm

Step2: Sort all polygons by z value keep the largest value of z first.

Step3: Scan converts polygons in this order.
Test is applied

1. Does A is behind and non-overlapping B in the dimension of Z as shown in fig (a)
2. Does A is behind B in z and no overlapping in x or y as shown in fig (b)
3. If A is behind B in Z and totally outside B with respect to view plane as shown in fig (c)
4. If A is behind B in Z and B is totally inside A with respect to view plane as shown in fig (d)

The success of any test with single overlapping polygon allows F to be painted.



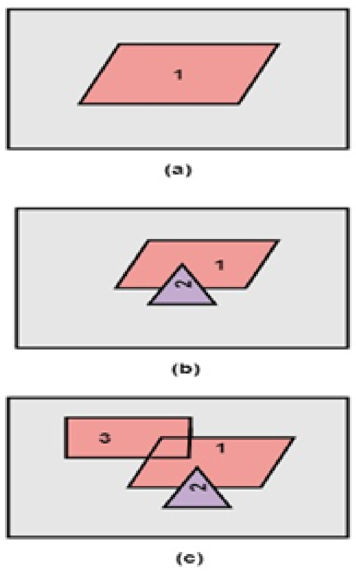


Figure showing addition of surface one by one and then painting done using painter algorithm

Z-Buffer (Depth Buffer):

- The Z-Buffer algorithm is a widely used and efficient technique for handling visibility. It uses a buffer, known as the Z-buffer or depth buffer, to store the depth value (distance from the viewer) for each pixel on the screen.
- During rendering, for each pixel, the Z-buffer is checked to determine if the current object is closer than what has already been rendered at that pixel. If it is closer, the pixel's color is updated, and the new depth value is written to the Z-buffer.

- The Z-Buffer algorithm works well for scenes with complex geometry and overlapping objects. However, it can be memory-intensive for high-resolution screens due to the need to store depth values for every pixel.

Depth Buffer Z−Buffer Method

This method is developed by Cutmull. It is an image-space approach. The basic idea is to test the Z-depth of each surface to determine the closest visible surface.

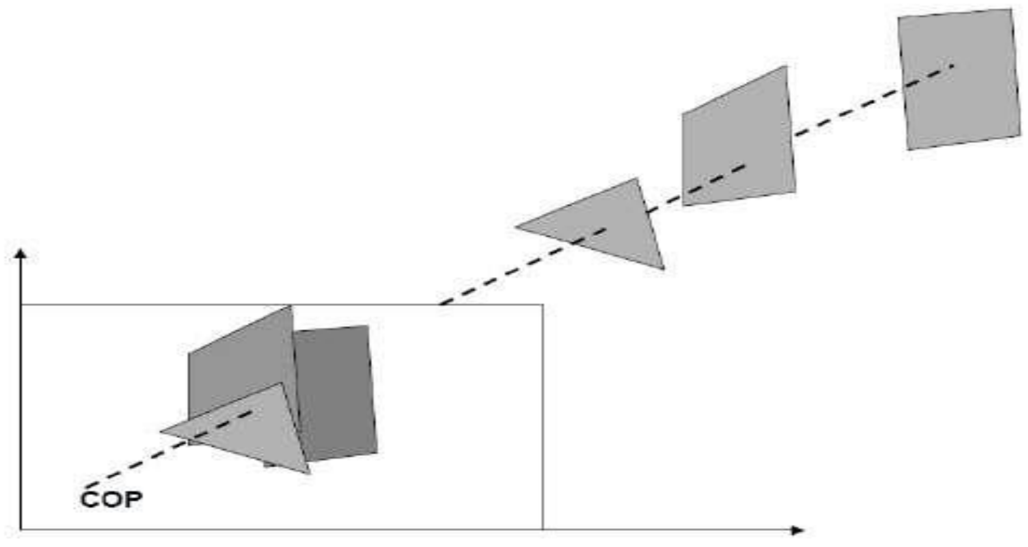
In this method each surface is processed separately one pixel position at a time across the surface. The depth values for a pixel are compared and the closest smallest z surface determines the color to be displayed in the frame buffer.

It is applied very efficiently on surfaces of polygon. Surfaces can be processed in any order. To override the closer polygons from the far ones, two buffers named **frame buffer** and **depth buffer,** are used.

Depth buffer is used to store depth values for x,y position, as surfaces are processed 0≤depth≤10≤h≤1.

The **frame buffer** is used to store the intensity value of color value at each position x,y

The z-coordinates are usually normalized to the range [0, 1]. The 0 value for z-coordinate indicates back clipping pane and 1 value for z-coordinates indicates Front clipping pane



Algorithm

Step-1 − Set the buffer values −

Depthbuffer x,y = 0

Framebuffer x,y = background color

Step-2 − Process each polygon Oneatatime

For each projected x,y pixel position of a polygon, calculate depth z.

If Z > depthbuffer x,y

Compute surface color,

set depthbuffer x,y = z,

framebuffer x,y = surfacecolor x,y

Advantages

- It is easy to implement.
- It reduces the speed problem if implemented in hardware.
- It processes one object at a time.

Disadvantages

- It requires large memory.
- It is time consuming process.