

# Visible Surface Detection

When a picture that contains the non-transparent objects and surfaces are viewed, the objects that are behind the objects that are closer cannot be viewed. To obtain a realistic screen image, these hidden surfaces need to be removed. This process of identification and removal of these surfaces is known as Hidden-surface problem.

## Techniques for Efficient Visible-Surface Algorithms

There are two basic approaches used for visible surface detection :

- Object precision algorithm
- Image precision algorithm

In both the algorithms we require to perform a number of potentially costly operations such as determination of projections of objects, whether or not they intersect and where they intersect, closest object in case of intersection and so on. To create and display picture in minimum time we have to perform visible surface algorithms more efficiently.

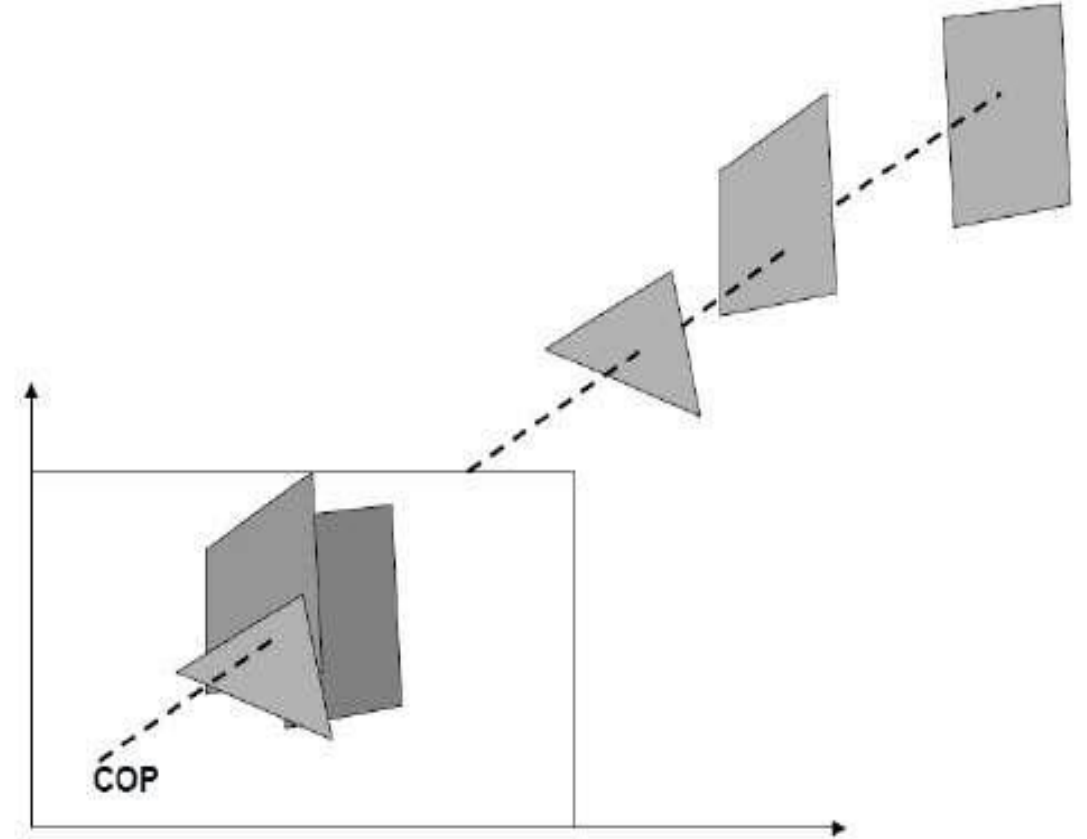
# Z-Buffer Method

It is an image-space approach developed by Cutmull. The Z-depth of each surface is tested for determining the closest surface. Across the surface, one pixel position is processed at a time. The colour that is to be displayed on the frame buffer is determined by the closest (smallest z) surface, by comparing the depth values for a pixel. The closer polygons are override by using two buffers namely frame buffer and depth buffer.

Depth buffer is used to store depth values for (x, y) position, as surfaces are processed ( $0 \leq \text{depth} \leq 1$ ).

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 plane and 1 value for z-coordinates indicates front clipping plane.



## Algorithm

**Step-1** – Buffer values are set as –

Depthbuffer  $(x, y) = 0$

Framebuffer  $(x, y) = \text{background color}$

**Step-2** – Each polygon is processed one at a time.

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

If  $Z > \text{depthbuffer}(x, y)$

Compute surface color,

set depthbuffer  $(x, y) = z$ ,

framebuffer  $(x, y) = \text{surfacecolor}(x, y)$

## Advantages

- Implementation is easy.
- The problems related to speed is reduced once it is implemented in the hardware.
- It processes one object at a time.

## Disadvantages

- Large memory is required.
- It is a time consuming process.

# Scan-Line Method?

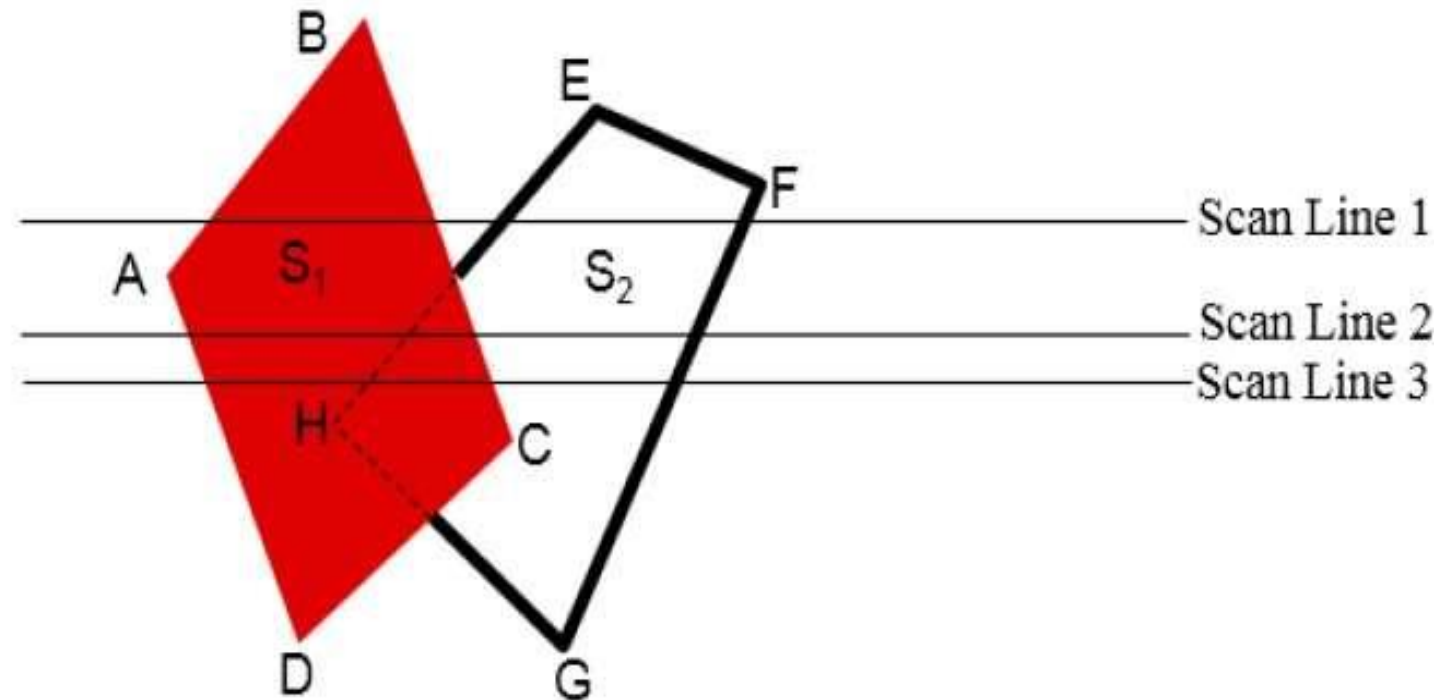
The visible surface is identified by the image-space method. All the polygons intersecting need to be grouped and processed for a particular scan-line before processing the next scan-line. This is done by maintaining two tables' edge table and polygon table.

**The Edge Table** – It contains coordinate endpoints of each line in the scene, the inverse slope of each line, and pointers into the polygon table to connect edges to surfaces.

**The Polygon Table** – It contains the plane coefficients, surface material properties, other surface data, and may be pointers to the edge table.

The search for the surfaces that cross a given scan line can be facilitated by an active list of edges. Only the edges that cross the scan line is stored by the active list. For indicating whether the position along a scan line is inside or outside the surface, a flag is set.

Each of the scan line is processes from left to right. The surface flag is turned on at left intersection and at right intersection it is turned off.

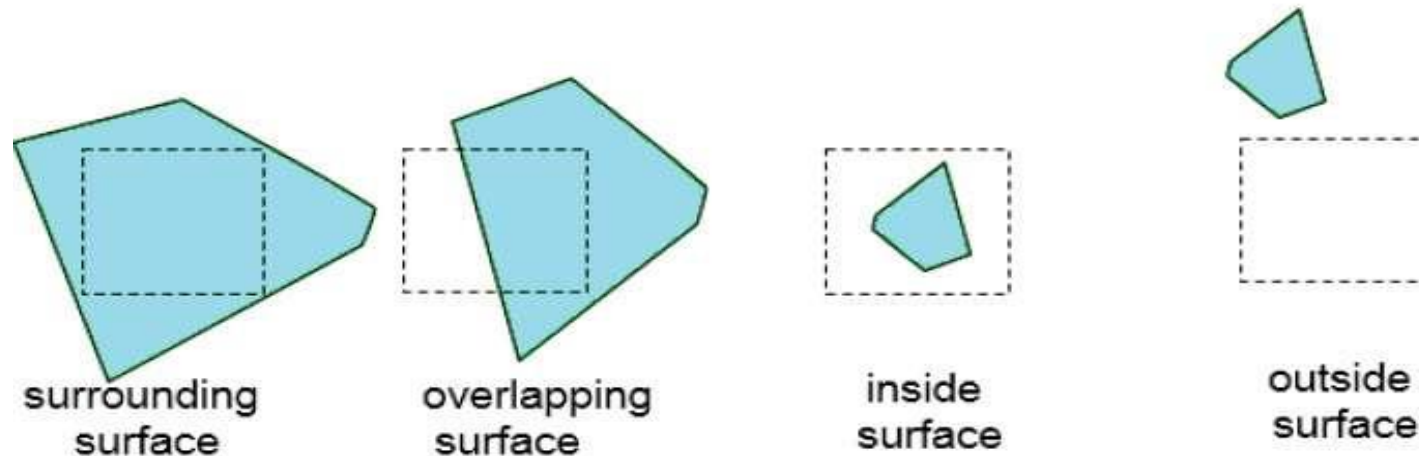


# Area-Subdivision Method

The view areas are located by representing a part or single surface. The total viewing area is divided into smaller and smaller rectangles until each of the small area is projected as a part of single surface.

This process is continues till all the subdivisions are analyzed to a single surface. This is done by dividing the area into four equal parts. With a specified area boundary there are four relationships:

- **Surrounding surface** – One that completely encloses the area.
- **Overlapping surface** – One that is partly inside and partly outside the area.
- **Inside surface** – One that is completely inside the area.
- **Outside surface** – One that is completely outside the area.



The surface visibility within an area is determined can be tested by stating in terms of the above four classification. If any one of the following conditions is true, the subdivisions of the area are not required.

- All surfaces are outside surfaces with respect to the area.
- Only one inside, overlapping or surrounding surface is in the area.
- A surrounding surface obscures all other surfaces within the area boundaries.

# Back-Face Detection

The back faces of the polyhedron are identified on the basis of “inside-outside” tests. A point  $(x, y, z)$  is "inside" a polygon surface with plane parameters  $A, B, C$ , and  $D$  if When an inside point is along the line of sight to the surface, the polygon must be a back face.

This test is simplified by the normal vector  $N$  to the polygon surface having Cartesian components  $(A, B, C)$ . The polygon is back face if,

$$V \cdot N > 0$$

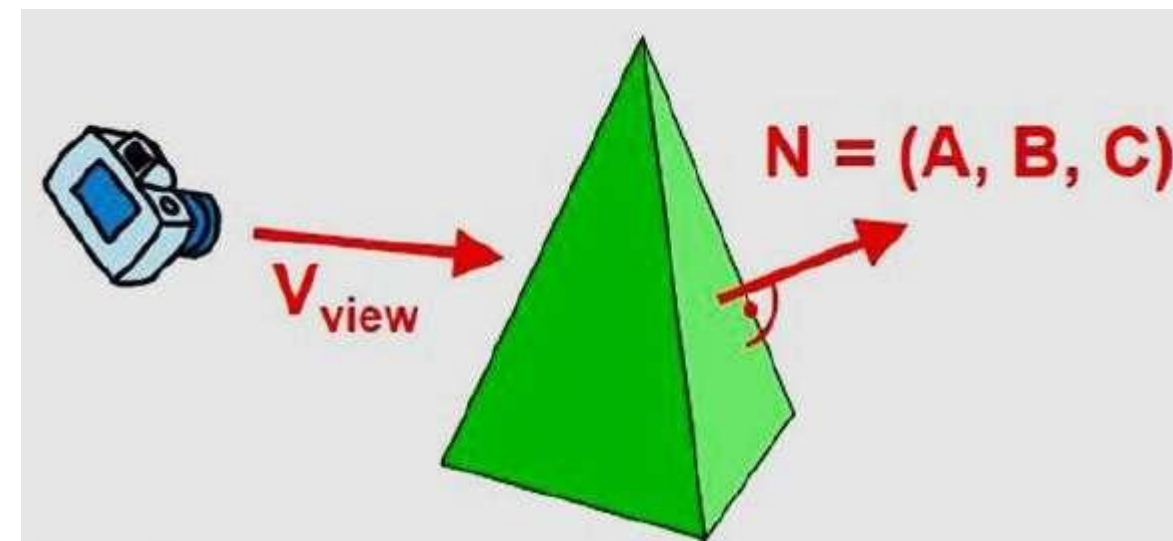
If object descriptions are converted to projection coordinates and your viewing direction is parallel to the viewing  $z$ -axis, then –

$$V = (0, 0, V_z) \text{ and } V \cdot N = V_z C$$

Hence only  $C$  of the component of the normal vector  $N$  is considered.

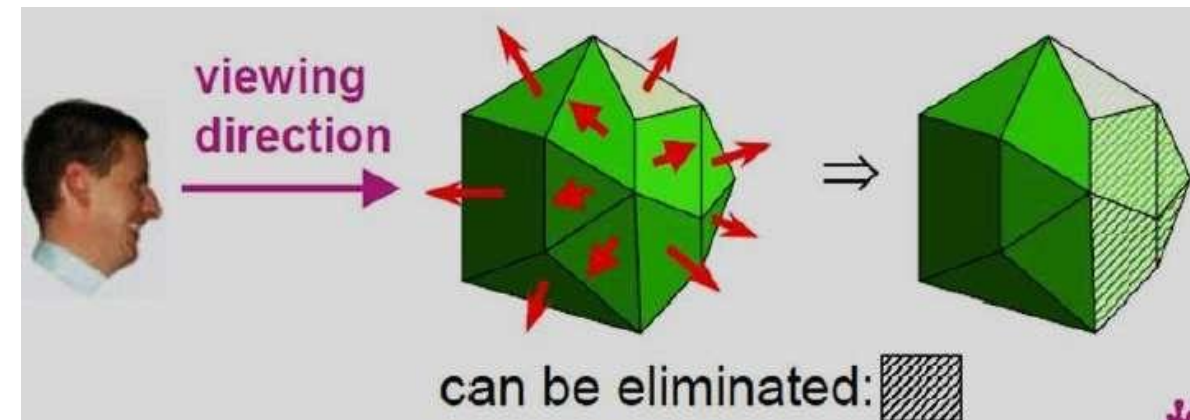
The polygon is back face if  $C < 0$ , for a right-handed viewing system with viewing direction along the negative  $ZV$  axis. Hence, in general any polygon is named as back face if the normal vector has the  $z$  component value.

$$C \leq 0$$



The packages can use similar methods for employing a left-handed viewing system. The polygon vertex coordinates are used for calculating  $A, B, C$  and  $D$ .

Normal vectors are available for back faces that are used for pointing away from the viewing position and identified by  $C \geq 0$ , along the  $Z_v$  axis. All the back faces are easily identified.

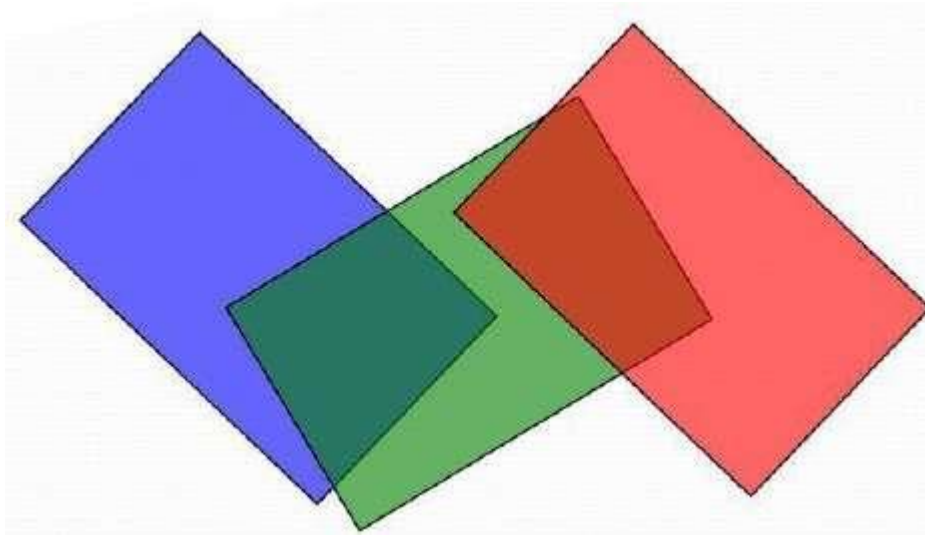




## A-Buffer Method

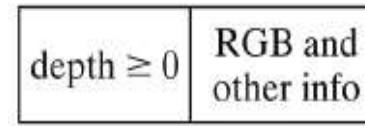
The extension of depth-buffer method is A-Buffer method. For rendering the Renders Everything You Ever Saw (REYES), the visibility detection method developed is A-buffer method.

Transparency is provided by A-buffer method. The accumulated buffer is the key data structure for the A-buffer method.



Each position in the A-buffer has two fields –

- **Depth field** – It stores a positive or negative real number
- **Intensity field** – It stores surface-intensity information or a pointer value



(a)



(b)

If depth  $\geq 0$ , the depth of the single surface corresponding to the pixel area stores the number. The A-buffer includes:

- RGB intensity components
- Opacity Parameter
- Depth
- Percent of area coverage
- Surface identifier

# Fractals

A French/American mathematician Dr Benoit Mandelbrot discovered Fractals. The word fractal was derived from a Latin word *fractus* which means broken. Fractals are very complex pictures generated by a computer from a single formula. They are created using iterations. This means one formula is repeated with slightly different values over and over again, taking into account the results from the previous iteration. Fractals are used in many areas such as –

- **Astronomy** – For analyzing galaxies, rings of Saturn, etc.
- **Biology/Chemistry** – For depicting bacteria cultures, Chemical reactions, human anatomy, molecules, plants,
- **Others** – For depicting clouds, coastline and borderlines, data compression, diffusion, economy, fractal art, fractal music, landscapes, special effect, etc.

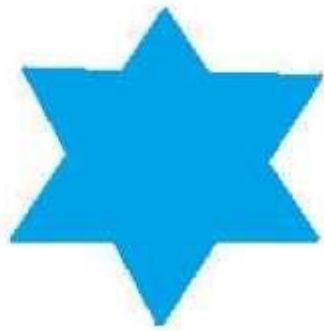
## Generation of Fractals

Fractals can be generated by repeating the same shape over and over again as shown in the following figure. In figure aa shows an equilateral triangle. In figure bb, we can see that the triangle is repeated to create a star-like shape. In figure cc, we can see that the star shape in figure bb is repeated again and again to create a new shape. We can do unlimited number of iteration to create a desired shape. In programming terms, recursion is used to create such shapes.

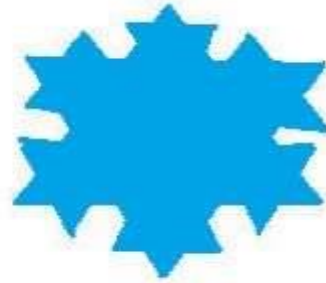




(a) Zeroth Generation



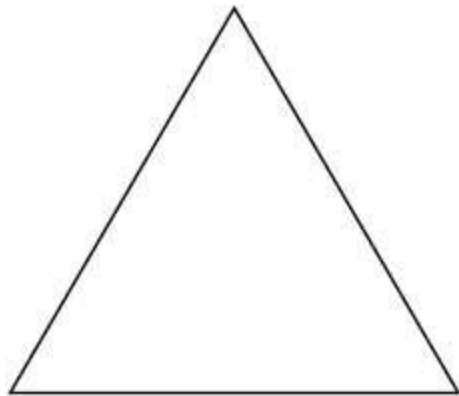
(b) First Generation



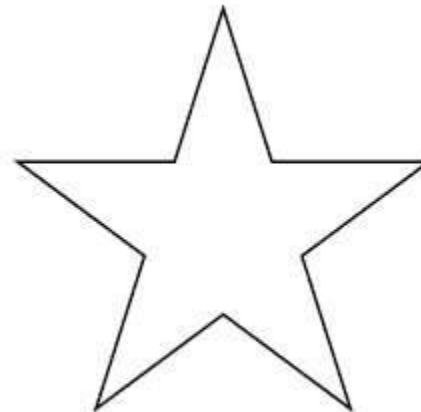
(c) Second Generation

## Geometric Fractals

Geometric fractals deal with shapes found in nature that have non-integer or fractal dimensions. To geometrically construct a deterministic nonrandom self-similar fractal, we start with a given geometric shape, called the **initiator**. Subparts of the initiator are then replaced with a pattern, called the **generator**.


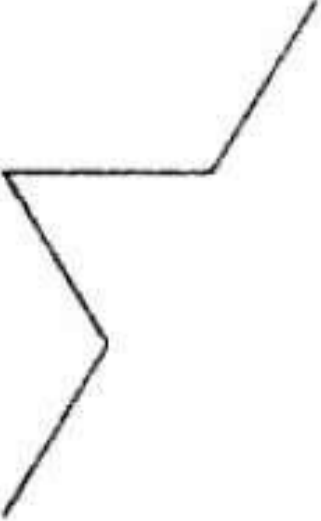



**Initiator**



**Generator**

Also, the length of each line segment in the initiator increases by a factor of  $\frac{4}{3}$  at each step, so that the length of the fractal curve tends to infinity as more detail is added to the curve as shown in the following figure –

Segment Length = 1	Segment Length = $\frac{1}{3}$	Segment Length = $\frac{1}{9}$
		
Length = 1	Length = $\frac{4}{3}$	Length = $\frac{16}{9}$