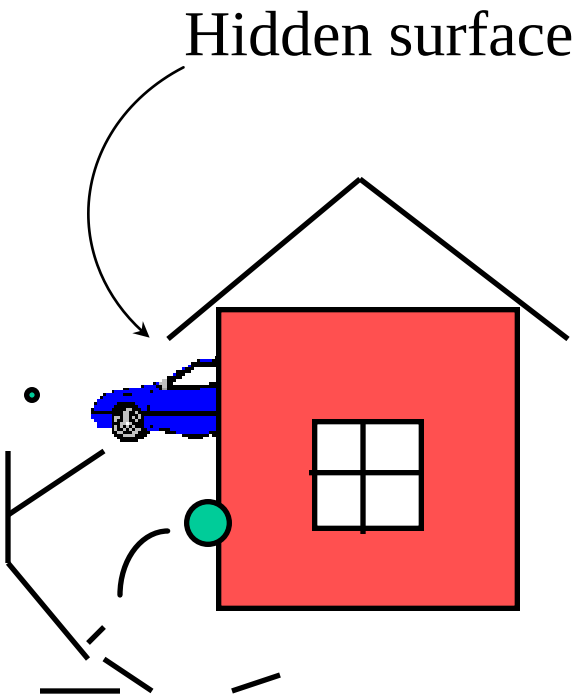
Hidden Surface Elimination

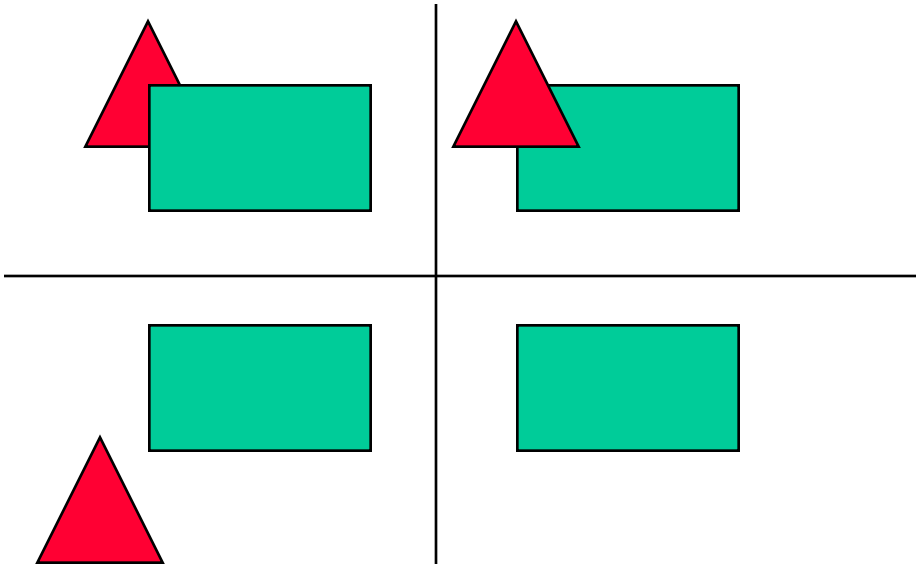
What are hidden surfaces?

When we view a picture containing non transparent objects and surfaces, then we can't see those objects from view which are behind from the objects closer to eye. We must remove these hidden surfaces to get realistic screen image. The identification & removal of these surfaces is called the **Hidden- surface problem.**

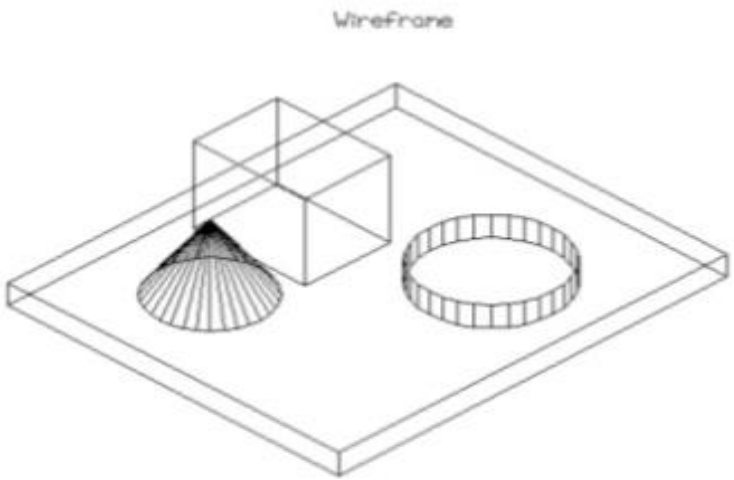
Hidden surface removal / Visible surface determination



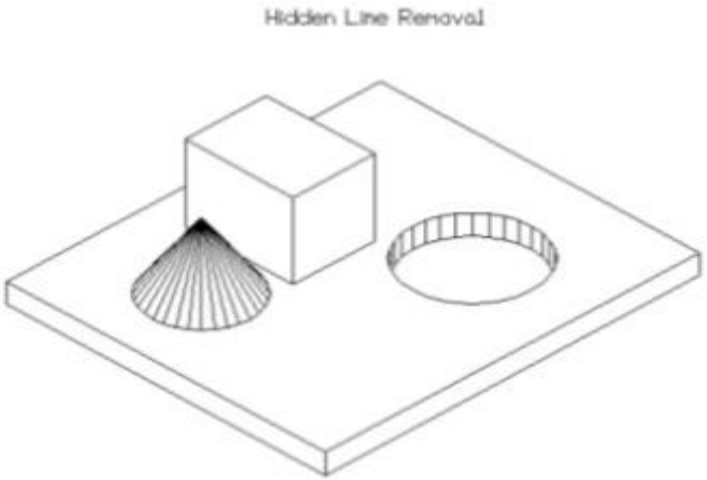
We do not wish to see things which are hidden behind other objects.



No Lines Removed



Hidden Lines Removed



Visible Surface Detection

- Visible surface detection or hidden surface removal.
- Realistic scenes: closer objects occludes the others.
- Classification:
  - Object space methods
  - Image space methods

Object Space Methods

- Algorithms to determine which parts of the shapes are to be rendered in 3D coordinates.
- Methods based on comparison of objects for their 3D positions and dimensions with respect to a viewing position.
- Efficient for small number of objects but difficult to implement.
- Depth sorting, area subdivision methods.

Image Space Methods

- Based on the pixels to be drawn on 2D. Try to determine which object should contribute to that pixel.
- Running time complexity is the number of pixels times number of objects.
- Space complexity is two times the number of pixels:
  - One array of pixels for the frame buffer
  - One array of pixels for the depth buffer
- Coherence properties of surfaces can be used.
- Depth-buffer and ray casting methods.

Z-Buffer Method

- Also known as ***depth*-Buffer** method.
- Proposed by Catmull in 1974
- Easy to implement
- Z-buffer is like a frame buffer, contain depths

Z-Buffer Method

It is an image space approach

- 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.
- Applied very efficiently on polygon surfaces
- Surfaces are processed in any order

Refresh (Frame) Buffer

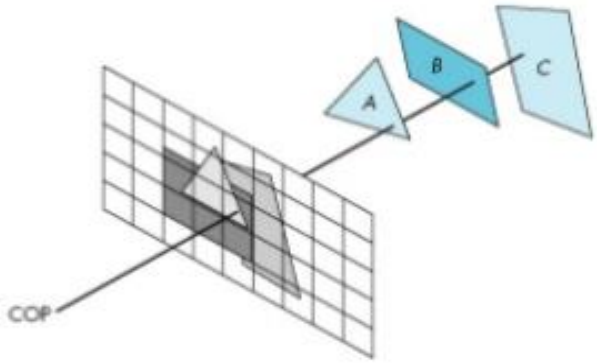
I(X,Y)	…	
: :		

Depth (Z) Buffer

Z(X,Y)	…	
: :		

Visibility

- How do we ensure that closer polygons overwrite further ones in general?



Z-Buffer Method

- Two buffers are used
  - Frame Buffer
  - Depth Buffer
- The z-coordinates (depth values) are usually normalized to the range [0,1]

Z-buffer Algorithm

```
Initialize all d[I,j]=1.0(max depth),c[I,j]=background color
For(each polygon)
For(each pixel in polygon's projection)
{
Find depth-z of polygon at (x,y) corresponding to pixel (I,j);
If z<d[I,j]
C[I,j]=color;          d[I,j]=z ;
End
}
```

Coherence Property

Calculating depth values efficiently

- We know the depth values at the vertices. How can we calculate the depth at any other point on the surface of the polygon.
- Using the polygon surface equation:

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

- Since . equation of plane is : $Ax+By+Cz+D=0$

Scan-Line Method

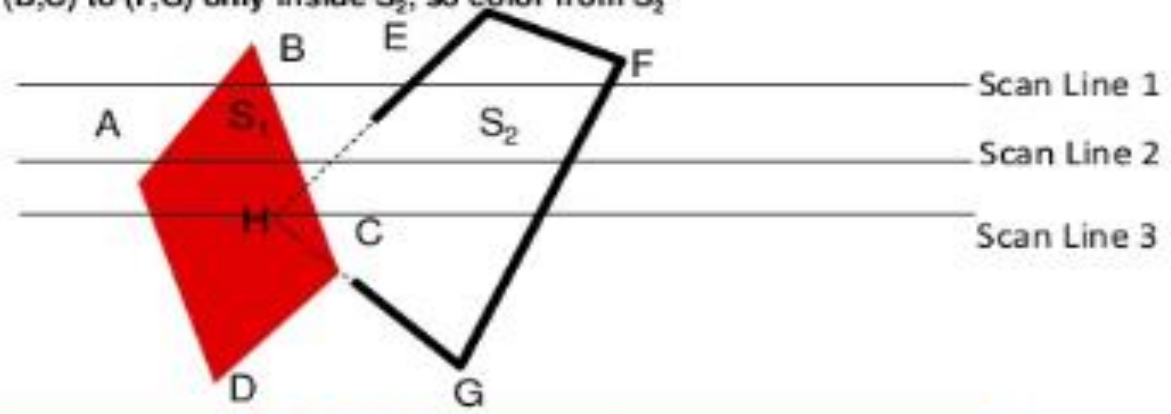
- Unlike z-buffer or A-buffer, scan-line method is also used to detect visible surface.
- In this algorithm we maintain depth information for each scan line.
- We do not compare the Z value where portion of surfaces are not overlapped but in case of overlapping surface we need to compare the Z value same as Z buffer and A buffer algorithm.

In this algorithm we have to maintain following things:

- Build table of edges of all polygons in scene.
- Maintain active-edge-table as we visit each scan-line in scene. AET now contains edges for all polygons at that scan line. Just Like Scan line algorithm (Polygon Fill).
- Must maintain flag for each surface to determine whether pixel on scan-line is inside that surface.

Scan-Line Method Basic Example

- Scan Line 1:
  - (A,B) to (B,C) only inside S_1, so color from S_1 (Flag1 will on and 2 will off)
  - (E,H) to (F,G) only inside S_2, so color from S_2 (Flag 2 will on 1 will off)
- Scan Line 3:
  - **(A,D) to (E,H) only inside S_1, so color from S_1**
  - **(E,H) to (B,C) inside S_1 and S_2, so compute & test depth. In this example we color from S_1**
  - **(B,C) to (F,G) only inside S_2, so color from S_2**



For Scan Line 1

Span	Flag of Surface s1	Flag of Surface s2	Intensity for the span
AB to BC	ON	OFF	Intensity of S1
BC to EH	OFF	OFF	Back Ground Intensity
EH to FG	OFF	ON	Intensity of S2

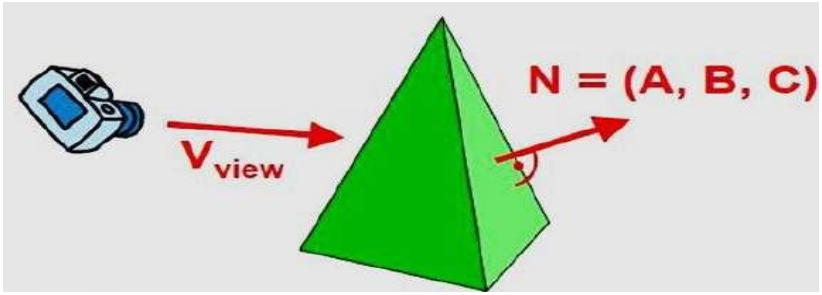
For Scan Line 3

Span	Flag of Surface s1	Flag of Surface s2	Intensity for the span
AD to EH	ON	OFF	Intensity of S1
EH to BC	ON	ON	Depth Calculations to decide Intensity
BC to FG	OFF	ON	Intensity of S2

Back-Face Detection

- A fast and simple object-space method for identifying the back faces of a polyhedron is based on the "inside-outside" tests. A point x,y,z is "inside" a polygon surface with plane parameters A, B, C, and D. When an inside point is along the line of sight to the surface, the polygon must be a back face we are inside that face and cannot see the front of it from our viewing position.
- We can simplify this test by considering the normal vector **N** to a polygon surface, which has Cartesian components A,B,C .
- In general, if V is a vector in the viewing direction from the eye or"camera"or"camera" position, then this polygon is a back face if
- **V.N > 0**

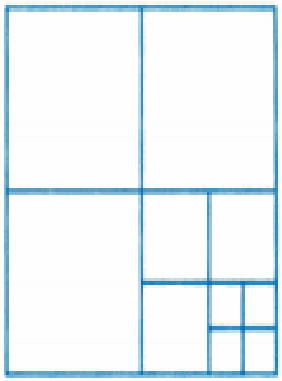
- Furthermore, if object descriptions are converted to projection coordinates and your viewing direction is parallel to the viewing z-axis, then −
- $V = (0, 0, V_z)$ and $V.N = V_Z C$
- So that we only need to consider the sign of C the component of the normal vector **N**.
- In a right-handed viewing system with viewing direction along the negative Z_V axis, the polygon is a back face if $C < 0$. Also, we cannot see any face whose normal has z component $C = 0$, since your viewing direction is towards that polygon. Thus, in general, we can label any polygon as a back face if its normal vector has a z component value −
- **C <= 0**



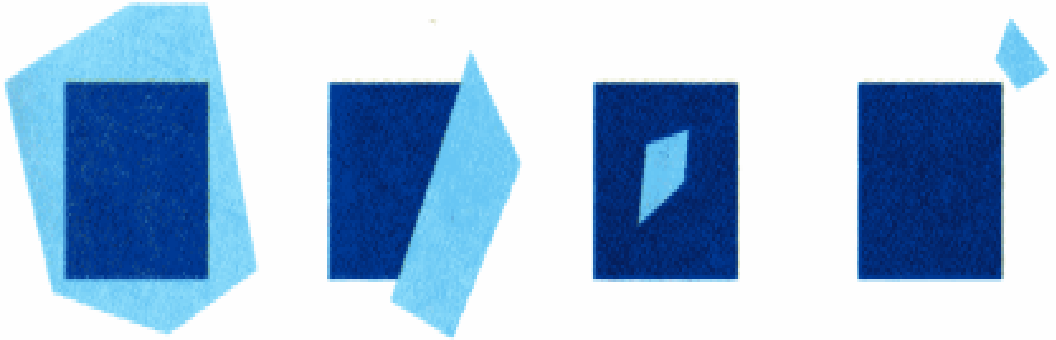
Area Subdivision(Warnock) Algorithm

The area-subdivision method takes advantage of area coherence in a scene by locating those view areas that represent part of a single surface.

The total viewing area is successively divided into smaller and smaller rectangles until each small area is simple, ie. it is a single pixel, or is covered wholly by a part of a single visible surface or no surface at all.



Dividing a square area into equal-sized quadrants at each step.



Possible relationships between polygon surfaces and a rectangular area.

Area Subdivision Algorithm

Consider an area of the projected image

If it is easy to decide which polygons are visible in the area, display
Else the area is subdivided into smaller areas and the decision is made recursively

Divide and Conquer

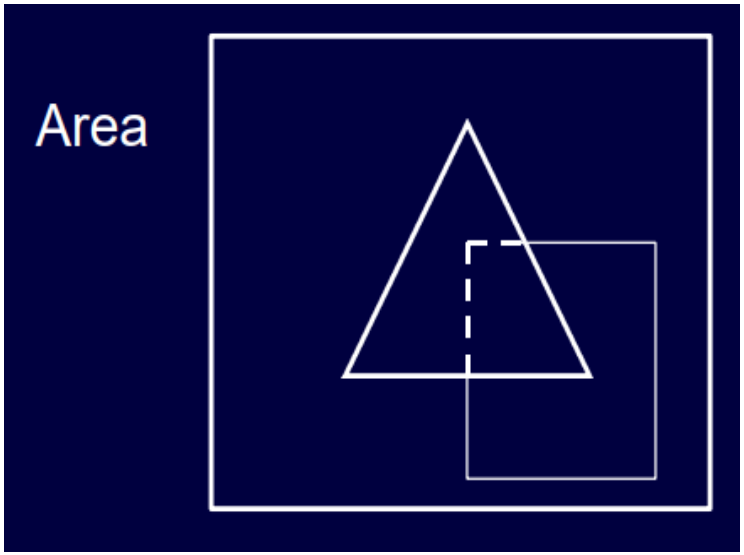
No Subdivision is required if

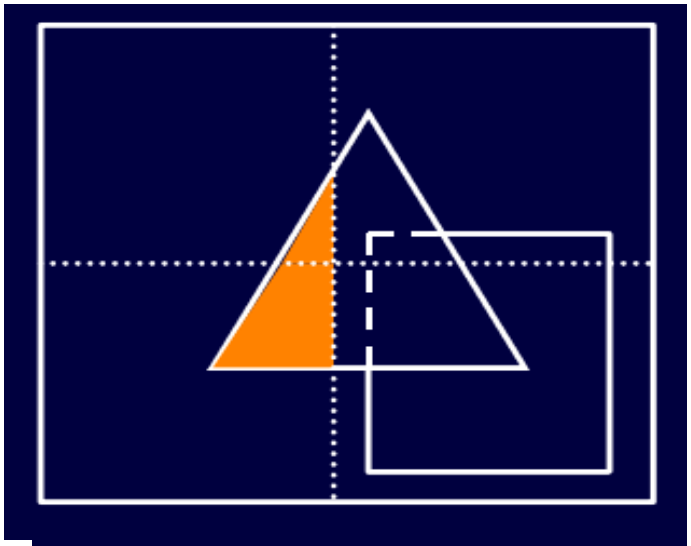
1. All the polygons are disjoint: background color in the area.

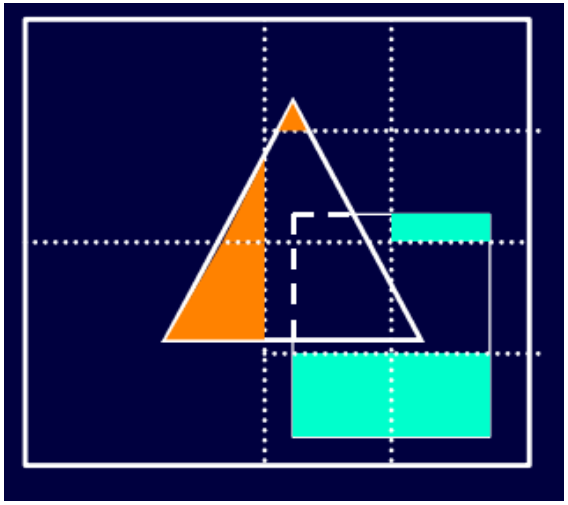
2. Only one intersecting or only one contained polygon: The area is filled first by background color, then the polygon part contained in the area.

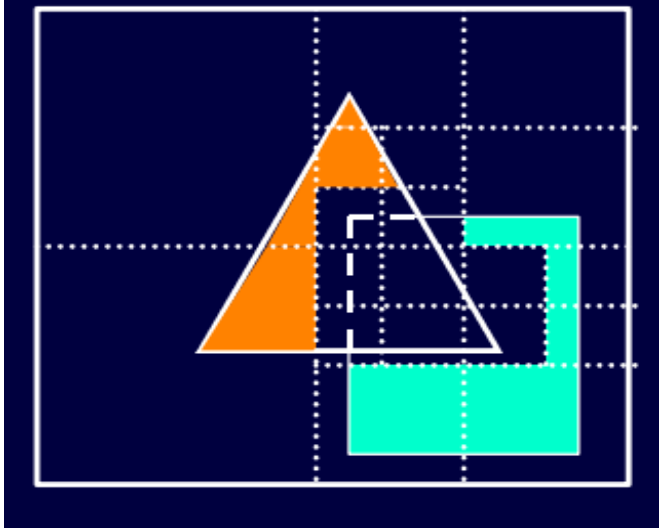
3. Only one surrounding polygon (no contained and intersecting polygons): The area is filled with the color of the surrounding polygon.

4. More than one polygon is intersecting, contained in, or surrounding the area, with surrounding polygon in front: Fill the area with the color of the surrounding polygon.


Area







- In Extreme case , it may happen that the size of area reduces to a size of single pixel and still overlapped by more than 1 polygons.
- In this case , depth calculations are performed to determine the nearest polygon , so that the pixel intensity is set to the intensity of it.