

## Determining the Visible Surface

“Given a set of objects in 3-D and a Viewing specification, to determine the portions of lines or surfaces that are visible from the view point”

- Ray Casting Algorithm
- Depth (Z) Buffer Algorithm
- Scanline Algorithm
- BSP trees
- Warnock’s Area Subdivision Algorithm.

## Image Space Algorithms

These typically solve the visible surface problem for **image pixel**, as follows:

```
for each pixel in image
do
    determine object closest to the view that is pierced
    by a projector through the center of the pixel.
    Illuminate pixel with the appropriate color.
end
```

Complexity:  $O(np)$ , n: number of objects, p: number of pixels in image.

## Object Space Algorithms

```
for each object in the environment
do
    determine parts of the object unobstructed by itself or
    any other object.
    Illuminate pixel with the appropriate color.
end
```

Complexity:  $O(n^2)$ , n: number of objects;

## Algorithm Classification

- Ray Casting: Image space algorithm, but visibility tests are at object precision.
- Z Buffer: Image space algorithm.
- BSP Tree: Object space algorithm: Object space algorithm, depth ordering determined in object space, followed by scan conversion.
- Warnock’s Area Subdivision: Image space algorithm.

## The Ray Casting Algorithm

- For visibility determination, only the primary ray is traced.
- Must use space subdivision structures to minimize intersection computations.
- Octrees, BSP trees, uniform grids are typically used.
- Refer to slides on global illumination, for details.

## Z (Depth) Buffer Algorithm (Catmull '74)

- A buffer of **Z or depth** values is maintained, one for each pixel, in addition to the frame buffer.
- During scan conversion, if primitive's depth is **less (closer)** than the existing value in the Z buffer, replace it and corresponding color (intensity) in the frame buffer.
- **Each primitive is scanconverted once**(visible or not), independent of any other primitive, and **in any order**.

## Z-buffer Algorithm

```
For each object primitive
{
  For each pixel in projection
  {
    Z = primitive's depth value at pixel(x,y).
    if Z < Z_BUF[x][y]
    {
      Write_ZBUF (x, y, Z)
      Write_Pixel (x, y, pixel_color)
    }
  }
}
```

## Calculating Z: Scanline Coherence

$$Ax + By + Cz + D = 0$$
$$z(x, y) = \frac{-Ax - By - D}{C}$$

Let  $z(x, y) = z_c$

$$z(x + \Delta x, y) = \frac{-A(x + \Delta x, y) - By - D}{C}$$
$$= z_c - \frac{A}{C}(\Delta x)$$
$$= z_c - \frac{A}{C}$$

## Calculating Z: Edge Coherence

Can similarly exploit edge coherence along Y (from scanline to scanline).

- Note that from the scanline algorithm, x coordinate varies as

$$x' = x - \frac{1}{m}$$

- Thus

$$\begin{aligned} A(x) + By + Cz + D &= 0 \\ A(x - 1/m) + B(y + 1) + Cz' + D &= 0 \end{aligned}$$

- Solve for z':

$$z' = z - \frac{\frac{A}{m} + B}{C}$$

## Z Buffer Algorithm: Notes

- Must have sufficient memory to have a Z buffer, no longer an issue today.
- If there is insufficient memory, may use just a scanline Z buffer, but lose coherence.
- Z values are usually from 32 bits wide or more.
- Easy to implement in hardware.
- Advantage decreases as the polygons reduce in size (or have fewer and fewer pixels in their projection)

## Scan-Line Algorithms

### Features

- Developed by Wylie, Romney, Evans, Erdahl.
- Extension of the polygon filling algorithm, exploiting scanline, edge and depth coherence
- **Differences:**
  - ⇒ need to deal with multiple polygons,
  - ⇒ incorporate shading and depth calculations.

## Data Structures

### 1. Edge Table:

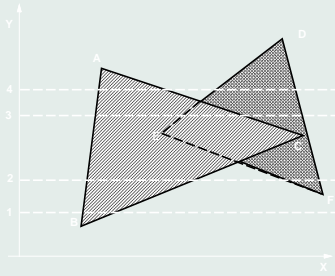
- ⇒ x coordinate of the edge with smaller y.
- ⇒ Max y coordinate of the edge.
- ⇒  $\Delta x$  (inverse slope) of the edge.
- ⇒ Polygon identifier.

### 2. Active Edge Table

### 3. Polygon Table:

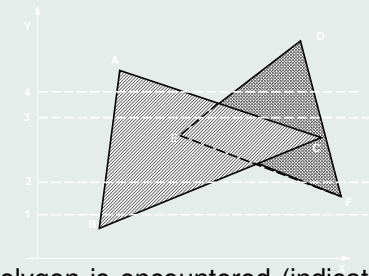
- ⇒ Coefficients of the plane equation.
- ⇒ Shading information for polygon.
- ⇒ Boolean flag, signifying if scan is in/out of polygon.

## Scanline Algorithm



- The scan conversion algorithm closely follows the polygon filling algorithm from scanline to scanline, exploiting scanline and edge coherence.
- The edge table and active edge table permit efficient scan conversion.
- If the scanline intersects only 1 polygon, then there is no change from the filling algorithm (scanline 1).

## Scanline Algorithm(contd)



- If a second polygon is encountered (indicated by the boolean flag set to 1 and an edge from a second polygon is encountered), then depth calculations are performed and the closer polygon's span is scan converted. New shading calculations are also triggered.
- When the scan 'leaves' a polygon, control goes back to the polygon which still has its boolean flag set to 'in' (scanline 3 in figure).
- Assumes non-penetrating polygons. If polygons penetrate, then they will need to be fragmented.

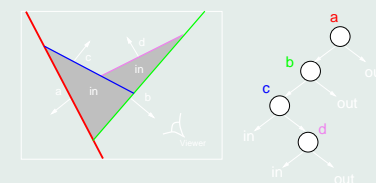
## Scan-line Z buffer

- Allocate a Z-buffer and frame buffer for 1 scanline.
- Depth calculations are performed between all points on the span each polygon on the scanline.
- Permits inter-penetrating polygons.

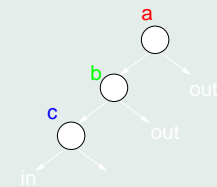
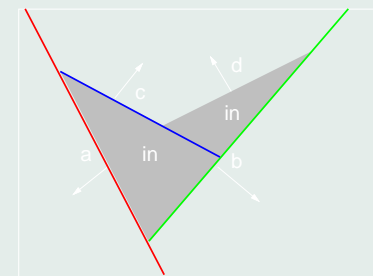
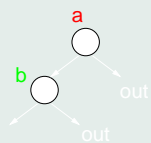
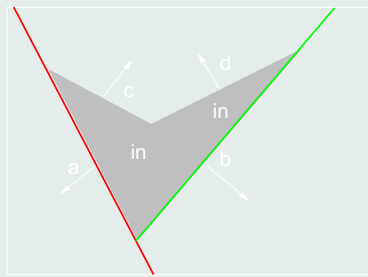
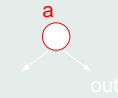
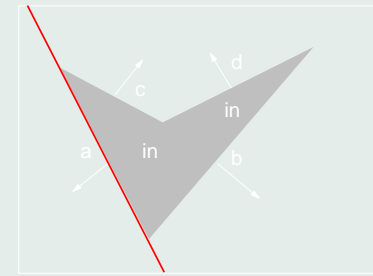
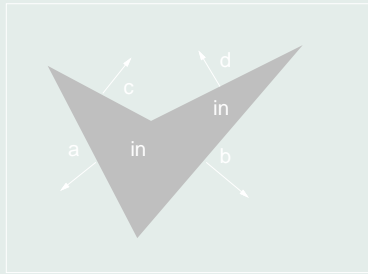
## Binary Space Partitioning Trees

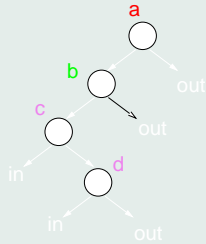
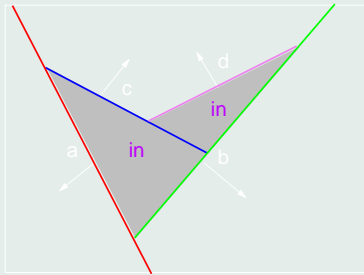
### Features:

- Continuous Space Representation.
- Represents geometry with hyperplanes.
- Convex decomposition of space by recursive subdivision.
- Internal nodes contain discontinuities, leaf nodes represent convex regions.

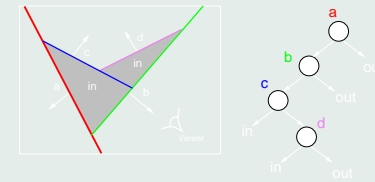


# Representing Polyhedra: An example



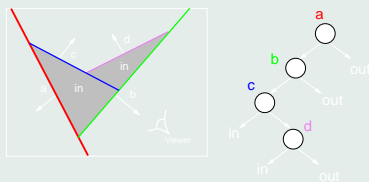


## Determining Visible Surface using the BSP Tree



- Paint faces **back to front**, as faces in front cannot obscure those that are behind.
- Classification is performed recursively, resulting in generating a priority order of faces, for a given viewer position.
- The BSP tree is independent of viewer location, so long as the objects do not move.

### Traversal Order for Viewer:



a+, a, a-  
a+, a, (b-, b, b+)  
a+, a, (c+, c, c-) b, b+  
a+, a, (d+, d, d-), c, c-, b, b+  
a+, a, d+, d, d-, c, c-, b, b+

## Area Subdivision Algorithms

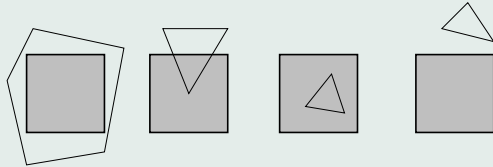
### Idea

- **Divide and conquer** strategy of spatial partitioning the projection plane.
- If it is “easy” to display the polygons in the area, display them, else **subdivide** and apply the same logic to the sub-areas.

## Warnock's Algorithm

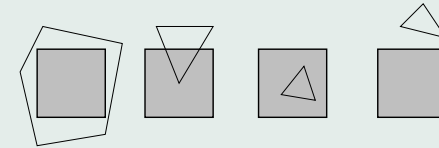
Subdivision of the original area into 4 equal areas. Analyze the polygon relationship to the area.

- **Surrounding polygons** completely contain the area
- **Intersecting polygons** intersect the area
- **Contained polygons** are completely inside the area
- **Disjoint polygons** are completely outside the area.



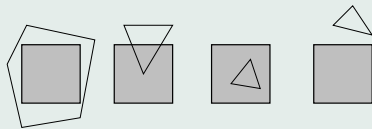
## Warnock's Algorithm

Consider the polygons that project onto this area (intersecting, surrounding, contained, or disjoint)



- All polygons are disjoint - paint background color over the area.
- Only 1 intersecting or 1 contained polygon - fill area with background color, then the part of polygon inside area is scanconverted.
- Single surrounding polygon, no intersecting or contained polygon - Area is filled with color of surrounding polygon.
- More than 1 polygon is intersecting, contained in, or surrounding the area - determine if a single surrounding polygon is in front of all others. If yes, fill area with color of surrounding polygon. The depth (z coordinates) of the planes of the polygons are examined at the 4 corners of the area.

## Warnock's Algorithm: Subdivision



- If none of the 4 cases resolves the problem, the area is subdivided into 4 equal areas and each area's polygons are examined to resolve the closest visible surface.
- Once the resolution of the display is reached, subdivision stops; the closest polygon visible at the center of the pixel is taken as the color of the pixel (for anti-aliasing, multiple subpixel areas are examined and averaged (with or without weighting)).