8. Hidden Surface Elimination

- Identification and Removal of parts of picture that are not visible from a chosen viewing position.
8. Hidden Surface Elimination

Basic idea:

Overwriting

Paint things in sorted order (from far to near)
Overwriting (order is important)
Overwriting (order is important)
Two categories:

- **Object space methods**: deal with object definitions directly. Order the surfaces so they can be drawn in a particular order to provide correct image.

- **Image space methods**: work as part of the projection process to determine relationship between object points on each projector.

**Pixel based**

**Face based**
8. Hidden Surface Elimination

- Sorting and Coherence techniques are used to improve performance
  - Sorting: to facilitate depth comparisons
  - Coherence methods: to take advantages of regularities in a scene

Object coherence, face coherence, edge coherence, area coherence, depth coherence, frame coherence
8.1 Back Face Removal

- For convex objects, sufficient to remove all back faces.
8.1 Back Face Removal

1. Compute outward normal:
   \[ \mathbf{N} = (\mathbf{v}_2 - \mathbf{v}_1) \times (\mathbf{v}_4 - \mathbf{v}_1) \]

2. (a) Projection type = parallel
   a face (polygon) is a back face if \( \mathbf{DOP} \cdot \mathbf{N} > 0 \)

   (b) Projection type = perspective
   a face (polygon) is a back face if \( \mathbf{V} \cdot \mathbf{N} > 0 \)
   (\( \mathbf{V} \) is a vector from COP to any vertex of the face)
8.2 **Z-Buffer (Depth-Buffer) Method**

- Image space method
- Simplest method, **one polygon at a time**
- Requires two arrays: **Intensity** and **Depth**, indexed by pixel coordinates \((x, y)\)

- For each pixel \((x, y)\) of the display screen, *keep the* depth of the object that lies closest to the viewer within the pixel in **Depth**\([x, y]\), and intensity value at the point of the object in **Intensity**\([x, y]\)
8.2 Z-Buffer (Depth-Buffer) Method

Algorithm:
1. For each pixel \((x, y)\) of the screen
   
   \[
   \text{Depth}[x, y] \leftarrow -1.0 \\
   \text{Intensity}[x, y] \leftarrow \text{background intensity (color)}
   \]

2. (Scan Conversion)
   For each polygon in the scene, find all the pixels that lie within the boundary of the polygon when projected onto the view plan
8.2 Z-Buffer (Depth-Buffer) Method

For each of these pixels \((x, y)\)

(2.1) calculate the depth \(z\) of the polygon at \((x, y)\)

(2.2) If \((z > \text{Depth}[x, y])\) then

(a) \(\text{Depth}[x, y] \leftarrow z\)

(b) \(\text{Intensity}[x, y] \leftarrow \) intensity or shading value of the polygon at \((x, y)\)

3. Copy Intensity into the frame (refresh) buffer
8.2 Z-Buffer (Depth-Buffer) Method

Initially,

- View point

Depth buffer : -1
Z-Buffer (Depth-Buffer) Method

\[ d_2 > \text{current value of depth buffer (}-1) \]

Hence, current value of Depth Buffer is replaced with \( d_2 \)

View point

Depth buffer: \( d_2 \)
Z-Buffer (Depth-Buffer) Method

\[ d_3 < \text{current value of Depth Buffer (d2)} \]

Hence, we keep the current value of the Depth Buffer
Z-Buffer (Depth-Buffer) Method

d1 > current value of depth buffer (d2)

Hence, d2 is replaced with d1
8.2 Z-Buffer (Depth-Buffer) Method

Notes:

- Polygons have to be transformed into (normalized) viewing coordinates and clipped against the normalized view volume first.

- Calculation of depth can be done as follows:
  "Record the plane equation of each polygon in the (normalized) viewing coordinate system and then use incremental method to find the depth z"
8.2 Z-Buffer (Depth-Buffer) Method

\[ Ax + By + Cz + D = 0 \]

\[ \rightarrow z = \frac{-D - Ax - By}{C} \]

Hence, depth at \((x + \Delta x, y) = z - \frac{A}{C} (\Delta x)\)

Only one addition/subtraction is required to compute depth at \((x + \Delta x, y)\)
8.2 Z-Buffer (Depth-Buffer) Method

Piercing polygons: can Z-buffer method handle piercing polygons?
8.3 Scan-Line Method

Depth sorting in 3D is too complicated and not efficient, hence, is not recommended.
8.3 Scan-Line Method

- An extension of the 2D scan-conversion algorithm
- Image-space method
- Deals with multiple polygons

General Idea:
1. Create **segments** of polygons by intersecting the polygons with the plane represented by a scan line
2. Sort all segments endpoints by x to identify all the **spans** of the scan line
3. If no segments appear in a span, the background intensity is used for this span

4. If only one segment is contained in a span then the segment is visible and the polygon equation is used to compute the intensity values for all the pixels in this span

5. If several segments extend across the entire span then the segment closest to the viewer, the one with the largest \( z \) value, is found and it’s intensity is used for this span
8.3 Scan-Line Method

Find intersection points of the scan line with all the active edges

Diagram shows a scan line intersecting with several polygons.
8.3 Scan-Line Method

Need 2 tables: Bucket Sorted Edge table (ET) Polygon table (PT)

- $PTR_1$ and $PTR_2$ are pointers to the polygons in PT that share this edge
### 8.3 Scan-Line Method

**Polygon Table**

<table>
<thead>
<tr>
<th>No of vertices</th>
<th>Plane equation</th>
<th>Shading (color) information</th>
<th>To Index</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Table</td>
</tr>
</tbody>
</table>
8.3 Scan-Line Method

- With the following representation,
8.3 Scan-Line Method

Need 2 lists: **Active-edge list** (AEL), **Active polygon list** (APL)

AEL: edges intersecting current scan line
APL: count, polygons overlapping current span

(updated for each new span)
8.3 Scan-Line Method

The algorithm

1. Set \( y \) to the smallest \( y \)-coordinate that has a non-empty bucket in ET

2. Set AEL and APL to empty; count \( \leftarrow 0 \).

3. Repeat the following steps while \( y \leq YMAX \)

\[(3.1)\]
Merge the edges in bucket \( y \) of ET with the edges in AEL in sorted order on \( x \), and let \([x_1, x_2], [x_2, x_3], \ldots, [x_{m-1}, x_m]\) be the corresponding spans on the current scan line.
(3.2)

If AEL is not empty then

for \( i = 1 \) to \( m - 1 \) do

for each edge in AEL whose \( x_{\text{min}} \) equals \( x_i \), if the polygon that contains this edge is already in APL then remove this polygon from APL and decrease \( \text{count} \) by 1; otherwise, put this polygon in APL and increase \( \text{count} \) by 1.
8.3 Scan-Line Method

if \( \text{count} > 0 \) then
    if \( \text{count} = 1 \) then compute the intensity values of the pixels between \( x_i \) and \( x_{i+1} \) using the equation of the polygon in APL
    else (\( \text{count} > 1 \)) compute the intensity values of the pixels between \( x_i \) and \( x_{i+1} \) using the equation of the polygon that is closest to the viewer
else /* \( \text{count} = 0 \) */
    Paint the span with background color
8.3 Scan-Line Method

(3.3) Remove edges in AEL whose $y_{top}$ equals current $y$.

(3.4) For each edge remaining in AEL, replace $x_{min}$ with $x_{min} + 1/m$

(3.5) Increment $y$ by 1 (to the next scan line)
8.4 Algorithm Efficiency
8.5 BSP Tree Method

- Binary Space-Partition (BSP) Trees (Fuchs, Kedem, Naylor)
- Efficient for calculating visibility relationships among 3D polygons (from any viewpoint)
- Based on the following concept
8.5 BSP Tree Method

A polygon on the same side of the plane as the eye point can not be obscured by polygons on the other side.
8.5 BSP Tree Method

Given a set of 3D polygons (with assigned normal directions), a BSP tree can be constructed as follows:

"Choose an arbitrary polygon as the root polygon. Use root polygon to partition environment into 2 half spaces: front and back (relative to polygon normal). Any polygon lying on both sides of root polygon’s plane is split. Then choose an arbitrary polygon on each side to divide the remaining polygons in its half-space in the same fashion. This process is recursively repeated until each region contains at most one polygon."
8.5 BSP Tree Method
8.5 BSP Tree Method
8.5 BSP Tree Method
8.5 BSP Tree Method
8.5 BSP Tree Method
8.5 BSP Tree Method

How to use a BSP tree to calculate visibility?

For a given view point, recursively display polygons of the tree in the following order:

- if the view point is in the root polygon’s front half-face, display polygons in the root’s rear half-space, the root polygon and then polygons in its front half-space.
- if the view point is in the root polygon’s rear half-space, display polygons in reverse order
- if the view point is on the plane that contains the root polygon then either way is okay.
End of Hidden Surface Elimination