Visible Surface Determination: Intro

- VSD deals with determining what is visible in a scene
- 2 general approaches are used:

  1. Object precision
     - Older approach
     - Designed for vector displays
     - Determines visibility of each object
     - Algorithm:
       ```
       for (each object in image) {
         determine parts of object not obstructed
         draw visible parts
       }
       ```
     - Performed at precision with which objects are defined
     - Compares every object to every other object, so approach $\in \Theta(n^2)$
     - Tends to be more costly than image precision, even when $n << p$, due to expensive operations

  2. Image precision
     - Designed for raster displays
     - Originally had much smaller address space than vector display
     - Display unaffected by number of objects
     - Determines visibility at each pixel
     - Algorithm:
       ```
       for (each pixel in image) {
         cast projector through pixel
         determine closest object pierced by projector
         color pixel with color of closest object
       }
       ```
     - Usually performed at resolution of display device
     - If $n =$ number of objects, $p = $ number of pixels, approach $\in \Theta(np)$
Visible Surface Determination: Intro (2)

- Major difference between approaches is effect of change in resolution
  - Image precision computed wrt display resolution
    * If this changes, must recompute visible surfaces
  - Object precision computed wrt object resolution
    * Results displayed at display resolution
    * If display resolution changes, must only repeat the display aspects
Visible Surface Determination: Single-valued Functions

- Special case of VSD
- Deals with display of function that computes a single value from 2 variables
  - E.g., \( y = f(x, z) \)

- Because occurs so frequently, handled as special case
- Represent function as \( n \times m \) array \( Y \)
  1. \( m \) corresponds to \( x \) axis
  2. \( n \) corresponds to \( z \) axis
  3. Elements represent \( f(x, z) \)
- Display by
  1. Drawing poly-lines for each row
     - These have constant \( z \)
  2. Then connect points on column-by-column basis
     - These have constant \( x \)
- Need to suppress drawing of non-visible segments
- Since each poly-line of constant \( z \) does not intersect any of the others, can draw rows front to back
- At any point in the process, will have a *silhouette boundary*
  - The visible border of the surface drawn so far
- When draw a new row, only display those parts that fall above or below this silhouette
Visible Surface Determination: Single-valued Functions - Horizon Line Algorithm

1. First step traverses lines of constant $z$
   
   - Records max and min $y$ values (at object resolution) for each $x$ value of display
   - These values stored in arrays $YMAX$ and $YMIN$
     - This set of values establishes the silhouette
     - Note that these values are at display resolution, while those in array $Y$ are at object resolution
   - Values where $YMIN[x_i] \leq y \leq YMAX[x_i]$ are not recorded (and hence not displayed)
   - Must consider visibility of segments between points
     - If 2 adjacent points visible wrt same array, entire segment visible
       - Update arrays with new $y$ values
       * Since resolution in $x$ and $y$ generally not the same, array values must be interpolated
     - If 2 adjacent points not visible, entire segment not visible
     - If either
       - $f(x_i, z) < YMIN[x_i]$ and $f(x_{i+1}, z) > YMAX[x_{i+1}]$, (or vice-versa), or
       - $f(x_i, z) < YMIN[x_i]$ and $YMIN[x_{i+1}] < f(x_{i+1}, z) < YMAX[x_{i+1}]$, (or vice-versa), or
       - $f(x_i, z) > YMAX[x_i]$ and $YMIN[x_{i+1}] < f(x_{i+1}, z) < YMAX[x_{i+1}]$, (or vice-versa)
       - Only part of segment visible
       - Intersect segment with contents of appropriate array to determine new array values
Visible Surface Determination: Single-valued Functions - Horizon Line Algorithm (2)

• Aliasing a problem when array resolution $\leq$ resolution of objects
  – Visible parts may be dropped, non-visible parts may be displayed

  ![Aliasing problems re Resolution](image)

• Object precision approach uses *silhouette polygon* instead of arrays
  – As process each new line, visible parts replace segments of current silhouette

2. Second step creates patch by following lines of constant $x$ in same manner as above
  (a) Start with line closest to view point
  (b) Process lines to both sides of this initial line, moving toward the edges

![Polylines of Constant X](image)
Visible Surface Determination: Single-valued Functions - Horizon Line Algorithm (3)

- For accurate results, cannot process 2 steps independently and overlay results

1. Process a single polyline of constant $z$
2. Process segments of constant $x$ connecting it to next polyline of constant $z$
3. Use a single set of silhouette arrays

- Horizon Line Algorithm not adequate for general VSD
Visible Surface Determination: General Case - Efficiency and Coherence

- General case computationally expensive:
  1. Computes intersection of projectors and objects
  2. Determines which intersection is closest to viewer

- Want to maximize efficiency of these calculations
  - Minimize number of times they are executed

- Efficiency achieved using *coherence*
  - Coherence expresses fact that object properties tend to vary minimally across a face
  - Discontinuities indicate important features of image (edges, etc.)

- Types of coherence:
  1. Object: If objects are completely separate, only need to compare object to object
     - Not component to component
  2. Face: Surface properties change slowly across a face
  3. Edge: Edge properties change slowly across an edge
  4. Implied edge: Edge formed by intersection of faces
     - It’s properties change slowly
  5. Scan line: Image changes little between scan lines
  6. Area: Adjacent pixels tend to have the same properties
  7. Depth: Adjacent parts of objects tend to be close in depth
  8. Frame: Images adjacent in time tend to change slowly
Visible Surface Determination: VSD and Projections

- VSD requires depth information
- Must be performed before projection
- VSD requires comparison of objects pierced by a given projector
  - Simple for parallel projections: have same $x$ and $y$ values
  - For perspective projections, 2 points lie on same projector if
    \[
    \frac{x_1}{z_1} = \frac{x_2}{z_2}, \quad \frac{y_1}{z_1} = \frac{y_2}{z_2}
    \]

\[\text{FP}\]

\[p_2(x_2, z_2)\]

\[p_1(x_1, y_1, z_1)\]

* Requires 4 divisions for initial 2 points, and 2 additional for every other point compared to $p_1$
- Comparisons usually performed after normalizing transformations
- This is another argument for converting canonical perspective view volume into the parallel canonical view volume
Visible Surface Determination: Overview of VSD Strategies

1. Extents/bounding volumes

- Extent is 2D, bounding volume 3D
  
  Extents and Bounding Volumes

- Form (circle, upright rectangle, ...) is chosen to facilitate comparisons
- Basic strategy is
  (a) See if extents overlap
  (b) If not, objects do not obscure each other
  (c) If so, do additional testing (e.g., subdivide extents and recurse)

Extent Cases

- Simplest case uses upright rectangle *bounding box*
  - Requires comparing only $x, y$ values
  - Called *min-max testing*
Visible Surface Determination: Overview of VSD Strategies (2)

- Shape of extent affects efficiency of VSD
  - Weyforst, Hooper, Greenberg formula:
    \[ T = bB + oO \]
  - where
    (a) \( T \) is cost of computing intersection test
    (b) \( b \) is number of times bounding volume is tested for intersection
    (c) \( B \) is cost of intersection test
    (d) \( o \) is number of times object is tested for intersection
    (e) \( O \) is cost of intersection test
    (f) \( o \leq b \)
    - The value of \( o \) varies directly with the tightness of the bounding volume
    - Usually the value of \( B \) varies inversely with tightness

- Orientation also affects the appropriateness of the bounding volume
Visible Surface Determination: Overview of VSD Strategies (3)

2. Back Face Culling

- Decreases processing by 1/2

- To determine back face,
  - Compute dot product of face normal with projector
    * If > 0, is back-facing
  - If test after projection, is back-facing if \( z \) component of normal < 0
    * Can be determined by sign of \( c \) in equation of surface \((ax + by + cz + d = 0)\)

- Eliminating back faces reduces number of polygons considered in object-precision-based VSD

- Object precision VSD \( \in \Theta(n) \)
  - Can improve by creating a table of visible faces indexed by quadrant of viewpoint
Visible Surface Determination: Overview of VSD Strategies (4)

3. Spatial partitioning
   - Divide and conquer approach
   - Breaks space into regular grid
   - Flag grid elements as occupied or unoccupied
   - Compare objects based on grid elements they occupy

4. Hierarchies
   - Complex objects modeled as hierarchy of parts

   - If parent objects do not overlap, do not need to check overlap of children
   - If overlap occurs, move down hierarchy
Visible Surface Determination: Visible Lines - Robert’s Algorithm

- Object precision algorithm
- Applied only to edges of convex polyhedra
- Uses back-face culling to eliminate shared edges of back-facing polygons

Removal of Edges Shared by Back-facing Polygons

- Remaining edges compared with other polyhedra using extents
- Polygon can obscure entire edge, one end, or middle
  - Result is 0, 1, or 2 segments
- Visibility determined by computing visibility of parametric intersection of projector and edge
  - Consider line \( p(t) \) from \( p_0 \) to \( p_1 \), where
    1. \( p(t) = s + dt \),
    2. \( s = p_0(x_0, y_0, z_0) \), and
    3. \( d = p_1 - p_0 \)
  - Consider a ray from the eye to some point on this line:
    \[ q(\alpha, t) = p(t) + g\alpha = s + dt + g\alpha \]
    * This is the parametric equation of a plane
Each polyhedron is stored as a set of faces, each represented as $ax + by + cz + d = 0$.

The plane equations are such that the dot product of a point on the line interior to the polyhedron will always be positive, while one exterior will produce a negative result.

Note that a point on a surface has a dot product equal to zero.

The algorithm works by:

1. Finding the values for $t, \alpha$ for which $h = s \cdot n + td \cdot n + \alpha g \cdot n = 0$ wrt each surface of a polyhedron.
2. Sorting the $t$ values.
3. Identifying the largest $t_{min}$ and smallest $t_{max}$.

* These correspond to the points on the line that delimit the hidden portion.
Visible Surface Determination: Visible Lines - Appel’s Algorithm

- Object precision algorithm
- Based on *quantitative invisibility* of points:
  - The number of front-facing polygons that obscure a point
- As line passes behind edge of polygon, $QI$ increments or decrements
- $QI$ changes only when crossing a *contour edge*
  - One that separates a front from a back-facing polygon, or
  - One from a front-facing polygon that is not shared

*Appel’s Algorithm*
Visible Surface Determination: Visible Lines - Appel’s Algorithm (2)

- Line passes behind contour edge if the edge intersects the triangle formed by the eye and the endpoints of the line
  - Intersection determined by clipping line against this plane
  - $\Delta QI = \text{sign of cross product of edge and contour line}$

- Polygons must be drawn in consistent manner in order to perform these calculations

- Algorithm:

  Select arbitrary vertex as seed vertex
  Determine $QI$ of vertex
  for (each edge sharing vertex) {
    follow edge, adjusting $QI$ as edge crosses behind contour lines
    if ($QI = 0$)
      draw
    if (encounter new vertex) {
      if (contour line intersects new vertex)
        for (each exiting edge)
          Calculate $QI$ against front-facing polygons that share vertex
          recursively follow exiting edges
    }
  }
Visible Surface Determination: Hidden Line Treatment

- Sometimes want to indicate obscured lines
  - E.g., display as dotted, etc.
- Can display using algorithms already discussed by simply changing drawing mode when significant edge encountered
- *Haloed* line is one that has area of suppression around it
  - Lines it obscures are drawn normally except in the haloed area

\[\text{Haloing}\]

```
All Lines Visible

Hidden Lines Haloed

Hidden Lines Removed
```
Visible Surface Determination: Depth Sort/Painter’s Algorithm

- Object precision algorithm
- Based on object extents
- Principle is to draw objects from back to front
  - Objects may be drawn unnecessarily
    * Far objects may be overwritten by nearer ones
- Algorithm:
  1. Sort polygons based on smallest $z$ coordinate
  2. Resolve conflicts of objects with overlapping $z$ extents
  3. Draw polygons in order of increasing $z$ coordinates
- Real problem is step 2

![Overlap Variations]

- No simple way to determine which polygon should be drawn first
Visible Surface Determination: Depth Sort/Painter’s Algorithm (2)

• Let $P$ and $Q$ be 2 polygons, where $P$ precedes $Q$ wrt $z$ coordinate, but extents overlap
  – $P$ drawn only if it does not obscure $Q$

1. Test following in succession until one is passed (or all fail)
   (a) Is the intersection of the $x$ extents empty?
   (b) Is the intersection of the $y$ extents empty?
   (c) Is $P$ entirely on the far side of $Q$’s plane wrt the viewpoint?
   (d) Is $Q$ entirely on the near side of $P$’s plane wrt the viewpoint?

2. If $P$ passes tests wrt $Q$, draw $P$ first
3. If tests not passed, assume $P$ obscures $Q$
   (a) Repeat tests 3 and 4 with roles of $P$ and $Q$ reversed
   (b) If $Q$ passes test wrt $P$, draw $Q$ first, then $P
Visible Surface Determination: Depth Sort/Painter’s Algorithm (3)

4. If neither case conclusive, split a polygon into several

Algorithm falters with more than 2 mutually-overlapping polygons

- Can determine order for any pair, but not for entire set
- Results in infinite loop

To remedy

1. Mark polygon when moved to end of list
2. When 5 tests fail, only repeat steps 3 and 4 with polygon order reversed if $Q$ was not marked previously
3. If $Q$ marked, split instead
Visible Surface Determination: Z Buffer Algorithm

- Image precision algorithm, but deals with objects
- Compatible with pipeline architecture
- Uses additional storage: *z-buffer*
  - Has same resolution as frame buffer
- Z-buffer initialized to infinity (back of clip volume)
- Frame buffer initialized to background color
- Algorithm:

```plaintext
for (each object)
  for (each pixel of object)
    if (z(pixel) > z(Z-buffer))
      ignore
    else {
      write frame buffer with pixel color
      z(Z-buffer) <- z(pixel)
    }
```

- Assumes dealing with orthographic clip volume
  - With perspective projection, cannot compare *z* values of pixels
  - Must see if pixels are on same projectors
Visible Surface Determination: Z Buffer Algorithm (3)

- Can make more efficient using scan-line coherence
  - Consider object in plane $Ax + By + Cz + D = 0$

- where
  - $(x_1, y_1, z_1)$ and $(x_2, y_2, z_2)$ are 2 points in the polygon
  - $\Delta x = x_2 - x_1$, $\Delta y = y_2 - y_1$, $\Delta z = z_2 - z_1$
  - Then $A\Delta x + B\Delta y + C\Delta z = 0$
  - For a given scan line,
    - $\Delta y = 0$, $\Delta x = 1$
  - So
    $$\Delta z = -\frac{A}{C} \Delta x$$
    which is constant for a planar polygon

- If plane equation not known or if surface non-planar, use bilinear interpolation
  1. $z_a = z_1 - (z_1 - z_2) \frac{y_1 - y_s}{y_1 - y_2}$
  2. $z_b = z_1 - (z_1 - z_3) \frac{y_1 - y_s}{y_1 - y_3}$
  3. $z_p = z_b - (z_b - z_a) \frac{x_b - x_s}{x_b - x_a}$

- Efficiency tends to be independent of number of polygons
  - As number of polygons increases, size of each tends to decrease
  - Pairs compared tends to remain constant
Visible Surface Determination: Z Buffer Algorithm (4)

• Problems

1. If z distances are large, and objects differ in z by very small amounts, may map to same z value in buffer
2. Merging of points differential in perspective cases
   – Distant points compress more than near ones
3. May result in *stitching*
   – Shared edges of polygons are generated from both sets of end points
   – Differences based on endpoint locations may result in some pixels shaded by one polygon, others by the other polygon

* Remedy: Insert spurious endpoints to insure scan conversion of shared edge starts from same points

• Z-buffer can be used in several ways

1. Can retain z-buffer
   – Facilitates insertion of new objects without entire visible surface recalculation
2. Can be written to selectively
   – Facilitates
     (a) Addition and removal of overlays
     (b) Blending
     (c) Picking
Visible Surface Determination: Schumacher’s Algorithm

- Example of list-priority algorithm
  - Applicable to situations where objects are static but viewpoint varies
  - Perform preprocessing prior to display

- Algorithm:
  - Divides faces into regions of space
  - Based on planes that cleanly divide polygons into 2 distinct groups

Schumacher’s Algorithm

- Those in near space may obscure those in nether space
- Can represent space partition as binary tree
  * Interior nodes represent planes
  * Leaves represent regions of space
Visible Surface Determination: Schumacher’s Algorithm (2)

- Faces assigned priorities

* Lower values have higher priority
- If 2 faces project to same point, one with higher priority is drawn
- Pixel will display color of highest priority face from highest priority region
- Can also display regions in increasing order, overwriting as needed
Visible Surface Determination: Binary Space Partitioning

• Extension of Schumacher’s Algorithm

1. Arbitrary polygon selected as root
2. Space divided into 2 regions (based on polygon normal):
   (a) Space in front
   (b) Space behind
3. Polygons behind placed on one branch of tree
4. Those in front on the other
5. Objects intersected by plane are split
6. Recursively repeat for each polygon
7. To display, perform in-order traversal
   – If view point is in front of root face, start with back branch
   – Otherwise, start with front branch

BSD Algorithm
Visible Surface Determination: Binary Space Partitioning (2)

- Works correctly regardless of polygon chosen as root
- Best case is root that results in fewest splits of descendents
- Intersection and sorting performed at object precision
- Display performed at image precision
- May result in more splits than depth sort
- Code:

```c
typedef struct {
    polygon root;
    BSP_tree *backChild, *frontChild;
} BSP_tree;

BSP_tree *BSP_makeTree (polygon *polyList)
{
    polygon root;
    polygon *backList, *frontList;
    polygon p, backPart, frontPart;

    if (polyList == NULL)
        return NULL;
    else {
        root = BSP_selectAndRemovePoly(polyList);
        backList = NULL;
        frontList = NULL;
        for (each remaining polygon p in polyList) {
            if (polygon p in front of root)
                BSL_addToList(p, &frontList);
            else if (polygon p in back of root)
                BSL_addToList(p, &backList);
            else {
                BSL_splitPoly(p, root, &frontPart, &backPart);
                BSL_addToList(frontPart, &frontList);
                BSL_addToList(backPart, &backList);
            }
        }
        return BSP_combineTree (BSP_makeTree(frontList), root, BSP_makeTree(backList));
    }
}

void BSP_displayTree (BSP_tree *tree)
{
    if (tree != NULL) {
        if (viewer is in front of tree->root) {
            BSP_displayTree(tree->backChild);
            displayPolygon(tree->root);
            BSP_displayTree(tree->frontChild);
        }
        else
            BSP_displayTree(tree->frontChild);
            displayPolygon(tree->root);
            BSP_displayTree(tree->backChild);
    }}
```
Visible Surface Determination: Binary Space Partitioning (3)

• List priority algorithms can be traversed front-to-back to eliminate unnecessary shading

• May introduce artifacts wrt new edges created by intersections
Visible Surface Determination: Scan Line Algorithms

- Image precision algorithms
- Process one scan line at a time
- Uses edge and scan line coherence
- Requires several data structures:
  1. Edge table
  2. Polygon table
  3. Active edge table
Visible Surface Determination: Scan Line Algorithm for Planar Surfaces

- Image precision
- Processes one scan line at a time
- Based on scan line coherence, edge coherence
- Uses following data structures:

1. Edge table
   - Holds 1 entry for each edge of each polygon
   - Stores
     (a) $x$ coordinate of end with smaller $y$ coordinate
     (b) $y$ coordinate of opposite end ($y_{\text{max}}$)
     (c) $\Delta x$ ($1/m$) from scan line to scan line
     (d) Id of polygon
   - Essentially a hash table
     * Entries indexed by smaller $y$ values
     * In a bucket, entries ordered by smaller $x$ value

2. Polygon table
   - Holds 1 entry per polygon
   - Stores
     (a) Coefficients of plane equation of polygon
     (b) Color
     (c) IN-OUT flag
       * Initialized to $FALSE$

3. Active edge table
   - Holds 1 entry for each edge that intersects current scan line
   - Edges ordered by $x$ intersection value
Visible Surface Determination: Scan Line Algorithm for Planar Surfaces

### Scan Line Algorithm

- **p1(2, 5, 2)**
- **p2(10, 8, -16)**
- **p3(12, 14, -10)**
- **p4(6, 10, 0)**
- **p5(6, 5, 3)**
- **p6(10, 5, 3)**
- **p7(8, 8, 3)**

### Edge Table:

<table>
<thead>
<tr>
<th>bucket</th>
<th>xmin</th>
<th>ymax</th>
<th>1/m</th>
<th>poly</th>
<th>edge</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>2</td>
<td>10</td>
<td>4/5</td>
<td>quad</td>
<td>14</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>8</td>
<td>8/3</td>
<td>quad</td>
<td>12</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>8</td>
<td>2/3</td>
<td>tri</td>
<td>57</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>8</td>
<td>-2/3</td>
<td>tri</td>
<td>76</td>
</tr>
<tr>
<td>8</td>
<td>10</td>
<td>14</td>
<td>1/3</td>
<td>quad</td>
<td>23</td>
</tr>
<tr>
<td>10</td>
<td>6</td>
<td>14</td>
<td>3/2</td>
<td>quad</td>
<td>43</td>
</tr>
</tbody>
</table>

### Poly Table:

<table>
<thead>
<tr>
<th>poly</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>color</th>
<th>flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>quad</td>
<td>3</td>
<td>-2</td>
<td>1</td>
<td>2</td>
<td>r</td>
<td>F</td>
</tr>
<tr>
<td>tri</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>b</td>
<td>F</td>
</tr>
</tbody>
</table>
Visible Surface Determination: Scan Line Algorithm for Planar Surfaces (2)

- Basic strategy:

1. For given scan line at $y = \alpha$,
   Add all edges whose $y_{max} = \alpha$ to active edge table
2. Move left to right along scan line, coloring pixel with current color
3. If $x$ value corresponds to $x$ entry in active edge table,
   (a) Set IN-OUT flag to true
   (b) Set color to polygon that corresponds to edge
4. If encounter a second edge of a polygon whose edge flag is set, reset to false
   - Color will become color of whatever polygon previously obscured (assuming no interpenetration)
5. If encounter leading edge of a different polygon, use plane coefficients to determine which is closer
   - Set edge flags, etc., as above if needed
6. When reach end of scan line,
   Update $x$ values of all entries in active edge table by respective $\Delta x$
   - Any that do not intersect scan line at $y = y + 1$ are removed from active edge table

- If polygons interpenetrate, clip against each other forming non-penetrating sub-polygons
  - Edges of intersection (false edges) used in edge table
- Can use depth coherence to facilitate repeated depth calculations
  - Use $\Delta z$ in same way as $\Delta x$ to calculate $z$ value of edge for each scan line
- Handling background color:
  1. Initialize frame buffer to background color
  2. Create a polygon of background color that fills the view port
  3. Have algorithm use background color when no polygon being drawn
Visible Surface Determination: Scan Line Algorithm for General Surfaces

- More complex than general case, because cannot make valid incremental decisions as in planar case
- Following approaches possible:
  1. Use planar approach
     - Use surface table and active surface table
     - Store info about entire surface, not just edges
  2. Z buffer
     - Use a z buffer and frame buffer that correspond to a single scan line
     - Apply z buffer algorithm to the single scan line
     - Display the frame buffer, clear, and process next scan line
  3. Invisible surface table
     - Based on scan line coherence
       * If surface not visible in a scan line, most likely not visible in next
     - Invisible surface table holds info about invisible surfaces
     - Surface added to IST if \( z_{max} \) < z values in z buffer for previous scan line for \( x_{min}, x_{max} \) values
     - Purpose is to minimize number of surfaces to be processed per scan line
     - IST can contain surfaces that it shouldn’t
       * Compare \( z_{max} \) of each surface with z value of each pixel as traverse scan line
       * If \( z_{max} > z \), add surface to IST
  4. Sechrest and Greenburg algorithm
     - Uses object precision
     - Based on notion that visibility changes only at vertices and at edge intersections
     - Sort these points on \( y \)
     - These \( y \) values segment screen space into horizontal bands
     - Each band represents area in which visibility relationships remain constant
Visible Surface Determination: Area Subdivision - Warnock’s Algorithm

• Object precision (except for scan conversion drawing aspect)
• Divide and conquer strategy
• Based on area coherence
• 2D projection plane divided into grid
• Basic strategy:
  1. Objects projected
  2. For each grid cell, id relation of projection wrt projections of other objects
  3. If relation clearly indicates what should is visible, draw
  4. Otherwise, subdivide the area and recurse on each sub-area
     – Recursion limited by display resolution
• Possible relationships:

 Warnock Polygon Relations

1. Projection is a surrounding polygon (a)
   – It completely contains the area
2. Projection intersects (b)
   – Only part of polygon occupies area
3. Projection contained (c)
   – It lies fully within the area
4. Disjoint (d)
   – Intersection of projection and area is empty
Visible Surface Determination: Area Subdivision - Warnock’s Algorithm (2)

- Can represent area in terms of a tree
  - Leaves of tree are
    1. Empty quadrants, and
    2. Quadrants with a single drawable piece
  - Node represented by
    1. Coordinate of lower left corner
    2. Width

![Warnock Quadtree Diagram](image-url)
Visible Surface Determination: Area Subdivision - Warnock’s Algorithm (3)

- Decision criteria:

  Warnock Decision Criteria

  1. All polygons disjoint (a)
     - Use background color
  2. Exactly one intersecting or 1 contained polygon (b)
     - Fill with background color
     - Scan convert polygon
  3. One surrounding polygon (c)
     - Fill with surrounding polygon color
  4. One surrounding polygon in front of all containing/intersecting polygons (d)
     - Fill with surrounding polygon color
     - Determine $z$ relations using $z$ coordinate of 4 corners of area
     - If cannot resolve, subdivide area

  Alternative is to divide space along polygon edges
Visible Surface Determination: Area Subdivision - Weiler-Atherton Algorithm

- Object precision
- Divides area along polygon edges
- Polygons subdivided at edges using standard clip algorithms
- Basic strategy:
  1. Sort polygons on $z$ coordinate
  2. Polygon closest to viewer becomes clip polygon
  3. Divide polygons into 2 sets:
     (a) Those inside the clip polygon (this includes the clip polygon itself) (a)
     (b) Those outside the clip polygon (b)
  4. Polygons that fall on both sides are clipped into sub-polygons that are inside or outside (c)

5. Inside polygons behind the clip polygon are discarded
6. If any inside polygon is closer than current clip polygon, recurse, using it as clip polygon
7. Display polygons in inside list
- When clip, always clip against *entire, original* polygon; not just current fragment
  - Must maintain link between each fragment and original
Visible Surface Determination: Area Subdivision - Weiler-Atherton Algorithm (2)

- Need to check for cyclical overlap
  - Use stack to hold current set of clip polygons
  - When identify inside polygon in front of current clip polygon
    1. Push onto stack and recurse
    2. If is already on stack, have already recursed on it - its extent overlaps that of current clip polygon

- Draws polygons front to back

- Code:

```c
void WA_visibleSurface (void)
{
    while (polyList != NULL)
        WA_subdivide(first polygon on polyList, &polyList);
}

void WA_subdivide (polygon clipPolygon, polygon **polyList)
{
    polygon *inList, *outList;
    inList = NULL;
    outList = NULL;

    for (each polygon in *polyList)
        clip polygon to ancestor of clipPolygon, placing inside pieces on inList, outside pieces on outList;
    for (each polygon in inList that is not on stack and not a part of clipPolygon) {
        push clipPolygon onto stack;
        WA_subdivide(polygon, &inList);
        pop stack;
    }
    for (each polygon in inList)
        display(polygon);
    *polyList = outList;
}
```
Visible Surface Determination: Area Subdivision - Octrees Intro

- Octrees: 3D (Quadtrees: 2D)
- Uses same approach as Warnock’s algorithm, except in 3D
- Space divided into octants, each a cube
- Octants can be numbered or mnemonically labeled
  - No standard approach
  - Numbering scheme:
    1. Back ”plane”:
       2 3
       0 1
    2. Front ”plane”:
       6 7
       4 5
  - Mnemonic scheme: L/R, U/D, F/B

- Each quadrant can be filled, empty, or partially full
Visible Surface Determination: Area Subdivision - Octrees Intro (2)

• As with Warnock’s algorithm, can represent space as a tree
  – Each node represents a quadrant
  – Full and empty nodes have no children
  – Partially full nodes have 8 children, 1 for each sub-quadrant

• Recurse as necessary
  – Limit is voxel (volume element)
Visible Surface Determination: Area Subdivision - Octree Algs

- Several approaches to using octrees in VSD
- Following assume orthographic projection

1. Basic back-to-front
   - Based on viewpoint
   - Use VPN to determine order of processing octants
   - (Use DOP for non-orthographic projections)
   - Order of display:
     (a) Display farthest
     (b) Display 3 octants that share face with farthest
     (c) Display 3 that share face with nearest
     (d) Display nearest
   - Only first and last octant order is fixed
   - If octant is subdivided, use above recursively
   - Only 3 faces of any octant visible from a viewpoint
     * VPN (or DOP) indicates which
     * x component: R if positive, L if negative
     * y component: U if positive, D if negative
     * z component: F if positive, B if negative
     * 0 means neither face visible
     * e.g., [1 -1 0]: right, bottom faces visible
Visible Surface Determination: Area Subdivision - Octree Algs (2)

2. Process slices
   – Process back-to-front
   – Process slices parallel to an axis
     * Order of selecting axes immaterial
   – Within a slice, process columns and rows (or *vice-versa*)
   – Use signs of VPN components
     * If VPN component of axis positive, process in increasing order
     * If negative, process in decreasing order

\[
\text{VPN} = [-1, 1, 1] \quad \text{VPN} = [1, 0, -1]
\]

3. Meagher’s algorithm
   – Process front-to-back
   – Octants projected into a quadtree
     * Quadtree quadrants initially empty

\[
\begin{array}{cc}
6/2 & 7/3 \\
4/0 & 5/1
\end{array}
\]
Visible Surface Determination: Area Subdivision - Octree Algs (3)

- Decision criteria:
  (a) If octant empty, do not project
  (b) If full or partially full:
    i. If quadrant is full, do not project into
       * Processing complete for this quadrant
    ii. If quadrant empty, project into it
    iii. If partially full
       * Quadrant further subdivided based on intersection of octant and *empty* part of quadrant