Visible Surface Algorithms

Zbuffer Algorithm

Plane of polygon in screen space  \( Ax + By + Cz + D = 0 \)

\( \text{FB}[x][y].Z \) initialized to BIG
\( \text{FB}[x][y].RGB \) Initialized to Background Color

for each polygon \( P \)
  for each pixel \( (x,y) \)
    if (Inside\( (x,y,P) \)) {
      \( z = (-Ax-By-D)/C; \)
      if (\( z < \text{FB}[x][y].Z \)) {
        \( \text{FB}[x][y].Z = z; \)
        \( \text{FB}[x][y].RGB = P.RGB; \)
      }
    }

Visible Surface Algorithms

PolygonList[] BucketSortYMin(PolygonList Plist)
{
    PolygonList B[Number of scanlines];
    Initialize all B[i] to null;
    for each Polygon P in Plist
    {
        k = min y value over all vertices of P;
        add P to B[k];
    }
    return B;
}

Similiarly for BucketSortYMax
Visible Surface Algorithms

\[x_0 \quad x_1 \quad x_2 \quad x_3\]
Visible Surface Algorithms

\[ z \]

\[ x_0 \quad x_1 \quad x_2 \quad x_3 \]

\[ x \]
Visible Surface Algorithms

Software Scanline Z Buffer Algorithm

ScanZ(polygonList PList)
{
    polygonList Bmin[], Bmax[], ActiveP, ActivePX;
    polygon Q;
    Bmin = BucketSortbyMinY(PList);
    Bmax = BucketSortbyMaxY(PList);
    ActiveP = empty;
    y = 0;
Visible Surface Algorithms

Software Scanline Z Buffer Algorithm

ScanZ(polygonList PList)
{
    polygonList Bmin[], Bmax[], ActiveP, ActivePX;
    polygon Q;
    Bmin = BucketSortbyMinY(PList);
    Bmax = BucketSortbyMaxY(PList);
    ActiveP = empty;
    y = 0;
    repeat
        ActiveP = ActiveP + Bmin[y] - Bmax[y];
        Xint[] = Xintersections(ActiveP,y);
        Sort(X[]);
        for (i=0;i< size of X[]; i++) {
            ActivePX = ActiveOnSpan(ActiveP,X[],i);
            Q = Closest(ActivePX);
            DrawSpan(Q,y,X,i);
        }
        y++;
        until (Bmin[z] == empty for all z>= y)
}
Visible Surface Algorithms

Hardware/Software Scanline Z Buffer Algorithm

Initialize FB[x][y].z = BIG;
Initialize FB[x][y].RGB = background color;

For each polygon P {
    for each scanline y {
        compute span = (x1,x2); //Fill algorithm
        for (x = x1;x<=x2;x++) {
            z = (-Ax-By-D)/C;  // Ax+By+Cz+D=0 is plane of
                              // P in screen space
            if (z< FB[x][y].z) {
                FB[x][y].z = z;
                FB[x][y].RGB = color of P;
            }
        }
    }
}
Visible Surface Algorithms

Warnock's Algorithm

Definition: Let W be a window on the screen and let PList be a polygonList. PList is SOLVABLE in W if any of the following hold:
Visible Surface Algorithms

Warnock's Algorithm

Definition: Let $W$ be a window on the screen and let $PList$ be a polygonList. $Plist$ is SOLVABLE in $W$ if any of the following hold:

1. $Plist$ is empty or contains 1 polygon
Visible Surface Algorithms

Warnock's Algorithm

Definition: Let W be a window on the screen and let PL ist be a polygonList. PL ist is SOLVABLE in W if any of the following hold:

1. PL ist is empty or contains 1 polygon

2. One polygon Q in PL ist completely fills the window and is closer than all other polygons in PL ist (i.e. The maxZ for Q is smaller than the minZ for all other polygons in PL ist).
Visible Surface Algorithms

Warnock's Algorithm

Definition: Let W be a window on the screen and let PList be a polygonList. Plist is SOLVABLE in W if any of the following hold:

1. Plist is empty or contains 1 polygon

2. One polygon Q in Plist completely fills the window and is closer than all other polygons in Plist (i.e. The maxZ for Q is smaller than the minZ for all other polygons in Plist).

3. Other more complicated criteria.
Visible Surface Algorithms
Visible Surface Algorithms
Visible Surface Algorithms
Visible Surface Algorithms
Visible Surface Algorithms

Warnock(Window W, PolygonList PList)
{
    ClipPList = Clip(PList,W);
    if (Solvable(W,ClipPList)) Draw(ClipPList);
    else
        if (size(W) <= pixelSize) Zbuffer(W,ClipPList);
        else {
            (W1,W2,W3,W4) = Quadrisect(W);
            Warnock(W1,ClipPList);
            Warnock(W2,ClipPList);
            Warnock(W3,ClipPList);
            Warnock(W4,ClipPList);
        }
}
Visible Surface Algorithms

\[ \text{xyMiniMaxBox}(\text{Polygon } P) \]
Visible Surface Algorithms

\texttt{zMiniMaxInterval(Polygon \ P)}
Visible Surface Algorithms

Polygon P does NOT obscure Polygon Q if any of the following hold:
Visible Surface Algorithms

Polygon P does NOT obscure Polygon Q if any of the following hold:

1. \( \text{ZminiMaxInterval}(P) \) intersect \( \text{ZminiMaxInterval}(Q) = \text{empty} \) AND \( Z\text{Max}(P) > Z\text{Max}(Q) \)
Visible Surface Algorithms

Polygon P does NOT obscure Polygon Q if any of the following hold:

2. $\text{xyMinMaxBox}(P)$ intersect $\text{xyMinMaxBox}(Q) = \text{empty}$
Visible Surface Algorithms

Polygon P does NOT obscure Polygon Q if any of the following hold:

3. Vertices of P are all Z-farther than plane of Q
Visible Surface Algorithms

Polygon P does NOT obscure Polygon Q if any of the following hold:

4. Vertices of Q are all Z-Closer than plane of P
Visible Surface Algorithms

Polygon P does NOT obscure Polygon Q if any of the following hold:

4. P and Q do not overlap in xy screen space.
Visible Surface Algorithms

Polygon P OBSCURES Polygon Q if all tests 1–5 fail.

Define a relation on polygons in screen space:

P < Q if P Obscures Q

Try to sort the polygons by this relation.
Visible Surface Algorithms

PROBLEM: This relation is not transitive.
Visible Surface Algorithms

PROBLEM: This relation is not transitive.

Q < P,  P < R,  But Q not < R
Visible Surface Algorithms

PROBLEM: This relation is not transitive

SOLUTION: Use the plane of one of the offending polygons to cut the others into smaller polygons.
Visible Surface Algorithms

The Painter's Algorithm

Painter(PolygonList Plist)
{
    Cut polygons by planes of other polygons until polygons can be sorted by the obscures relation.

    Draw polygons from "back-to-front";
}

Visible Surface Algorithms
Visible Surface Algorithms
Visible Surface Algorithms

P < Q < R

Draw R
Draw Q
Draw P
Visible Surface Algorithms

Problem with all these algorithms:
Visible Surface Algorithms

Problem with all these algorithms:

Move your eye to another position and perpectively project polygons again.

Then you have to redo everything from the beginning.
Visible Surface Algorithms

What if you were willing to spend a significant amount of “preprocessing” time so that when you were ready to display the polygons you could solve the visible surface problem from any eye position rapidly, perhaps even in “real-time”.

Visible Surface Algorithms

Let $P$ be a polygon lying in a plane, $\text{Plane}(P)$, with equation

$$Ax + By + Cz + D = 0$$

Polygon $Q$ is on the front side of $\text{Plane}(P)$ if for every vertex $(x,y,z)$ in $Q$,
$$Ax + By + Cz + D \geq 0$$

Polygon $Q$ is on the back side of $\text{Plane}(P)$ if for every vertex $(x,y,z)$ in $Q$,
$$Ax + By + Cz + D < 0$$
Visible Surface Algorithms

If Q is not entirely on the front or back of P, Cut Q into 2 polygons Q1, Q2 one of which is on the front and the other on the back.
Visible Surface Algorithms

This is like the polygon clipping algorithm. Instead of clipping a polygon to an edge in 2-space, we clip it to a plane in 3-space.
The Binary Separating Plane (BSP) Tree
The BSP Tree
The BSP Tree
The BSP Tree

\{ p_1, p_2a \} \quad \{ p_2b, p_3, p_4 \}
The BSP Tree

Diagram showing the BSP Tree with nodes and edges labeled as p1, p2a, p2b, p3, p4, and p5. The tree structure is depicted with p1 as the root, p2a as a child, and p5 as another child with p2b, p3, and p4 as its children.
The BSP Tree
class BSPTree {
    Polygon P;
    double A, B, C, D;
    BSPTree front, back;
}
The BSP Tree

BSPTree* MakeBSPTree(PolygonList Plist)
{
    if (Plist == empty) return null;
    else {
        P = Select(PList);
        BSPTree t = new BSPTree(P);
        PolygonList Back = empty, Front = empty;
        for (Q in Plist, Q != P) {
            if (Q on front side of plane of P)
                add Q to Front;
            else if (Q on back side of plane of P)
                add Q to Back;
            else {
                split Q into QFront and QBack;
                add Qfront to Front and Qback to Back;
            }
        }
        t.front = MakeBSPTree(Front);
        t.back  = MakeBSPTree(Back);
        return t;
    }
}
void TraverseBSP(BSPTree t, Position eye) {
    if (t != null) {
        if (eye on front side of Plane(t.P) { 
            TraverseBSP(t.back, eye); 
            Draw(t.P); 
            TraverseBSP(t.front, eye); 
        }
        else {
            TraverseBSP(t.front, eye); 
            Draw(t.P); 
            TraverseBSP(t.back, eye); 
        }
    }
}
The BSP Tree
The BSP Tree

Draw p2a
The BSP Tree

Draw p2a p1
The BSP Tree

\begin{center}
\begin{tikzpicture}
\node (p5) at (0,0) {p5};
\node (p1) at (-3,-3) {p1};
\node (p2a) at (-1,-6) {p2a};
\node (p4) at (0,-3) {p4};
\node (p3) at (1,-3) {p3};
\node (p2b) at (1,-6) {p2b};
\node (p2) at (-1,0) {p2};
\node (p3) at (0,0) {p3};
\node (p4) at (1,0) {p4};
\node (p5) at (2,0) {p5};
\draw (p1) -- (p2);
\draw (p2) -- (p3);
\draw (p3) -- (p4);
\draw (p3) -- (p5);
\draw (p2) -- (p5);
\draw (p1) -- (p2a);
\draw (p4) -- (p2b);
\node[blue] at (0,-1.5) {eye};
\end{tikzpicture}
\end{center}

Draw p2a, p1, p5
The BSP Tree

```
Draw p2a p1 p5 p2b
```
The BSP Tree

Draw p2a p1 p5 p2b p3
The BSP Tree

```
Draw p2a p1 p5 p2b p3 p4
```
The BSP Tree
The BSP Tree

Draw p4
The BSP Tree

- p1
- p2a
- eye
- p3
- p4
- p2b
- p5

Draw p4 p3
The BSP Tree

Draw p4 p3 p2b
The BSP Tree

Draw p4 p3 p2b p5
The BSP Tree

Draw p4 p3 p2b p5 p1
The BSP Tree

Draw $p_4$ $p_3$ $p_{2b}$ $p_5$ $p_1$ $p_{2a}$