3D Polygon Rendering Pipeline

Clipping
3D Rendering Pipeline (for direct illumination)

3D Primitives

- Modeling Transformation
  - 3D Modeling Coordinates
- Lighting
  - 3D World Coordinates
- Viewing Transformation
  - 3D World Coordinates
- Projection Transformation
  - 3D Camera Coordinates
- Clipping
  - 2D Screen Coordinates
- Viewport Transformation
  - 2D Screen Coordinates
- Scan Conversion
  - 2D Image Coordinates
- Image
  - 2D Image Coordinates
3D Rendering Pipeline (for direct illumination)

3D Primitives
  ↓
Modeling Transformation
  ↓
3D Modeling Coordinates
Lighting
  ↓
3D World Coordinates
Viewing Transformation
  ↓
3D World Coordinates
Projection Transformation
  ↓
3D Camera Coordinates
Clipping
  ↓
2D Screen Coordinates
Viewport Transformation
  ↓
2D Screen Coordinates
Scan Conversion
  ↓
2D Image Coordinates
Image

- 3D Modeling Coordinates
- 3D World Coordinates
- 3D Camera Coordinates
- 2D Screen Coordinates
- 2D Image Coordinates
2D Rendering Pipeline

3D Primitives

2D Primitives

Clipping

Clip portions of geometric primitives residing outside the window

Viewport Transformation

Transform the clipped primitives from screen to image coordinates

Scan Conversion

Fill pixels representing primitives in screen coordinates

Image
2D Rendering Pipeline

3D Primitives → 2D Primitives

- **Clipping**: Clip portions of geometric primitives residing outside the window.
- **Viewport Transformation**: Transform the clipped primitives from screen to image coordinates.
- **Scan Conversion**: Fill pixels representing primitives in screen coordinates.

**Image**
Clipping

- Avoid drawing parts of primitives outside window
  - Window defines part of scene being viewed
  - Must draw geometric primitives only inside window
Clipping

- Avoid drawing parts of primitives outside window
  - Window defines part of scene being viewed
  - Must draw geometric primitives only inside window
• Avoid drawing parts of primitives outside window
  ◦ Points
  ◦ Lines
  ◦ Polygons
  ◦ Circles
  ◦ etc.
Point Clipping

- Is point \((x,y)\) inside the clip window?

\[\text{inside} = (x \geq wx1) \land (x \leq wx2) \land (y \geq wy1) \land (y \leq wy2)\]
Line Clipping

- Find the part of a line inside the clip window
Line Clipping

- Find the part of a line inside the clip window

After Clipping
Cohen-Sutherland Line Clipping

- Use simple tests to classify easy cases first
Cohen-Sutherland Line Clipping

- Use simple tests to classify easy cases first

Clipping is performed by the computation of the intersections with four boundary segments of the window: \( L_i, \ i=1,2,3,4 \)

**Purpose:** Fast treatment of lines that are trivially inside/outside the window.
Let \( P=(x,y) \) be a point to be classified against window \( W \).

**Idea:** Assign \( P \) a binary code consisting of a bit for each edge of \( W \), whose value is determined according to the following table:
<table>
<thead>
<tr>
<th>bit</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$y &lt; y_{\text{min}}$</td>
<td>$y \geq y_{\text{min}}$</td>
</tr>
<tr>
<td>2</td>
<td>$y &gt; y_{\text{max}}$</td>
<td>$y \leq y_{\text{max}}$</td>
</tr>
<tr>
<td>3</td>
<td>$x &gt; x_{\text{max}}$</td>
<td>$x \leq x_{\text{max}}$</td>
</tr>
<tr>
<td>4</td>
<td>$x &lt; x_{\text{min}}$</td>
<td>$x \geq x_{\text{min}}$</td>
</tr>
</tbody>
</table>

![Diagram](image)
Given a line segment $S$ from $p_0=(x_0,y_0)$ to $p_1=(x_1,y_1)$ to be clipped against a window $W$.

- If $\text{code}(p_0) \text{ AND } \text{code}(p_1)$ is not zero - then $S$ is \textit{trivially rejected}.
- If $\text{code}(p_0) \text{ OR } \text{code}(p_1)$ is zero - then $S$ is \textit{trivially accepted}.
Cohen Sutherland Line Clipping

- Classify some lines quickly by AND of bit codes representing regions of two endpoints (must be 0)
Cohen Sutherland Line Clipping

- Classify some lines quickly by AND of bit codes representing regions of two endpoints (must be 0)

<table>
<thead>
<tr>
<th>Bit 1</th>
<th>Bit 2</th>
<th>Bit 3</th>
<th>Bit 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>1001</td>
<td>1000</td>
<td>1001</td>
</tr>
<tr>
<td>P2</td>
<td>P1</td>
<td>P3</td>
<td>P7</td>
</tr>
<tr>
<td>P5</td>
<td>0000</td>
<td>0000</td>
<td>0001</td>
</tr>
<tr>
<td>P6</td>
<td>P4</td>
<td>P8</td>
<td>0100</td>
</tr>
<tr>
<td>P9</td>
<td>0110</td>
<td>0101</td>
<td></td>
</tr>
<tr>
<td>P10</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Cohen Sutherland Line Clipping

- Classify some lines quickly by AND of bit codes representing regions of two endpoints (must be 0)
Cohen Sutherland Line Clipping

- Classify some lines quickly by AND of bit codes representing regions of two endpoints (must be 0)
Cohen Sutherland Line Clipping

- Compute intersections with window boundary for lines that can’t be classified quickly
• Compute intersections with window boundary for lines that can’t be classified quickly.
Cohen Sutherland Line Clipping

- Compute intersections with window boundary for lines that can’t be classified quickly
Cohen Sutherland Line Clipping

- Compute intersections with window boundary for lines that can’t be classified quickly
Cohen Sutherland Line Clipping

- Compute intersections with window boundary for lines that can’t be classified quickly.
Cohen Sutherland Line Clipping

- Compute intersections with window boundary for lines that can’t be classified quickly
Cohen Sutherland Line Clipping

- Compute intersections with window boundary for lines that can’t be classified quickly.
Cohen Sutherland Line Clipping

- Compute intersections with window boundary for lines that can’t be classified quickly
Cohen Sutherland Line Clipping

- Compute intersections with window boundary for lines that can’t be classified quickly
Cohen Sutherland Line Clipping

- Compute intersections with window boundary for lines that can’t be classified quickly

```
Bit 1  Bit 2  Bit 3  Bit 4
1010  0010  0000  0101
1000  0001  0001  0101
1001  0001  0001  0101
P'5   P3    P4    P'7
P'6   P9    P'8   P10
```
Cohen Sutherland Line Clipping

- Compute intersections with window boundary for lines that can’t be classified quickly.
Cohen Sutherland Line Clipping

- Compute intersections with window boundary for lines that can’t be classified quickly
Cohen Sutherland Line Clipping

- Compute intersections with window boundary for lines that can’t be classified quickly
Cohen Sutherland Line Clipping

- Compute intersections with window boundary for lines that can’t be classified quickly
Cohen Sutherland Line Clipping

- Compute intersections with window boundary for lines that can’t be classified quickly
Cohen-Sutherland Algorithm

CompOutCode (x, y : real; var code : outcode);
/* Compute outcode for the point (x,y) */
begin
  code := 0;
  if Y > Ymax then code := code | B1000
  else
    if Y < Ymin then code := code | B0100;
    if X > Xmax then code := code | B0010
    else
      if X < Xmin then code := code | B0001;
  end;
Cohen-Sutherland Algorithm (cont.)

```c
CS (x0,y0,x1,y1,xmin,xmax,ymin,ymax)
{
    boolean accept, done ;
    float outcode0, outcode1, x, y;
    accept := false ;
    done := false ;
    CompOutCod (x0,y0,outcode0);
    CompOutCod (x1,y1,outcode1);
    repeat
        if  ((outcode0 | outcode1) == 0)
        {
            /*  Trivial accept  */
            accept := true ;
            done := true ;
        }
        else
        if ((outcode0 & outcode1) <> 0)
        { /*  Trivial reject  */
            done := true
        }
        else
        /* Failed both tests, so calculate the line segment to clip from an outside point to an intersection with clip edge  */
```
Cohen-Sutherland Algorithm (cont.)

/* At least one endpoint is outside the clip rectangle, pick it*/
if (outcode0 <> 0) {
    outcodeOut := outcode0
} else {
    outcodeOut := outcode1;

/* now find the intersection point by using the formulas:
   y = y0 + slope*(x-x0), and
   x = x0 + (1/slope)*(y-y0) */

if (outcodeOut & 0x1000) then
    divide line at top of clip rectangle;
else if (outcodeOut & 0x0100) then
    divide line at bottom of clip rectangle;
else if (outcodeOut & 0x0010) then
    divide line at right edge of clip rectangle;
else if (outcodeOut & 0x0001) then
    divide line at left edge of clip rectangle;
/* Now we move outside point to intersection point to clip, and get ready for next pass */
if (outcodeOut == outcode0)
{
    x0:=x;y0:=y;CompOutCod (x0,y0,outcode0);
}
else {
    x1:=x;y1:=y;CompOutCod (x1,y1,outcode1);
}
}  /* Subdivide */
    until (done) ;

if (accept) draw_line (x0,y0,y0,y1);
} /* end */
A 2D vector $V$ is defined as: $V=(V_x, V_y)$.

Scalar (dot) product between two vectors $V$ and $U$ is defined:

$$V \cdot U = \begin{bmatrix} V_x \\ V_y \end{bmatrix} \begin{bmatrix} U_x \\ U_y \end{bmatrix} = V_x U_x + V_y U_y = |V||U| \cos \theta$$

If $V \cdot U = 0$ then $V$ and $U$ are perpendicular to each other.
Vector Subtraction

Vector Addition
**Parametric Line**

\[ P_0 = P(t=0) \]
\[ P_1 = P(t=1) \]

\[ P(t) = P_0 + (P_1 - P_0)t \]

**Changing the Origin**

\[ V(t=0) \]
\[ V(t) = P(t) - Q \]
\[ V(t=1) \]
Inside/Outside Test

- Assume WLOG that \( V = (V_1 - V_0) \) is the border vector where "inside" is to its right.
- If \( V = (V_x, V_y) \), \( N \) is a prep' vector pointing outside, where we define:
  \[ N = (-V_y, V_x) \]
- Vector \( U \) points "outside" if \( N \cdot U > 0 \)
- Otherwise \( U \) points "inside".
The parametric line $P(t)=P_0+(P_1-P_0)t$

The parametric line $V(t)=P(t)-Q$

The segment intersects the line $L$ at $t_0$ satisfying $V(t_0)\cdot N=0$.

The intersection point is $P(t_0)$.

The vector $\Delta=P_1-P_0$ points "inside" if $(P_1-P_0)\cdot N<0$.

Otherwise it points "outside".

If $L$ is vertical, intersection can be computed using the explicit equation.
Cyrus-Beck Line Clipping

Denote $p(t) = p_0 + (p_1 - p_0)t$ \( t \in [0..1] \)

Let $Q_i$ be a point on the edge $L_i$ with outside pointing normal $N_i$.

$V(t) = p(t) - Q_i$ is a parameterized vector from $Q_i$ to the segment $P(t)$.

$N_i \cdot V(t) = 0$ iff $V(t) \perp N_i$

We are looking for $t$ satisfying the above equation:
Cyrus-Beck Clipping (cont.)

\[ 0 = \mathbf{N}_i \cdot V(t) = \mathbf{N}_i \cdot (\mathbf{p}(t)-\mathbf{Q}_i) \]

\[ = \mathbf{N}_i \cdot (\mathbf{p}_0+(\mathbf{p}_1-\mathbf{p}_0)t-\mathbf{Q}_i) = \mathbf{N}_i \cdot (\mathbf{p}_0-\mathbf{Q}_i) + \mathbf{N}_i \cdot (\mathbf{p}_1-\mathbf{p}_0)t \]

Solving for \( t \) we get:

\[ t = \frac{\mathbf{N}_i \cdot (\mathbf{p}_0-\mathbf{Q}_i)}{-\mathbf{N}_i \cdot (\mathbf{p}_1-\mathbf{p}_0)} = \frac{\mathbf{N}_i \cdot (\mathbf{p}_0-\mathbf{Q}_i)}{-\mathbf{N}_i \cdot \Delta} \]

where \( \Delta = (\mathbf{p}_1-\mathbf{p}_0) \)

- **Comment**: If \( \mathbf{N}_i \cdot \Delta = 0 \), \( t \) has no solution.
  
  However, in this case \( V(t) \perp \mathbf{N}_i \) and there is no intersection.
potentially Entering and potentially Leaving
Cyrus-Beck Algorithm:

- The intersection of $p(t)$ with all four edges $L_i$ is computed, resulting in up to four $t_i$ values.

- If $t_i < 0$ or $t_i > 1$, $t_i$ can be discarded.

- Based on the sign of $N_i \cdot \Delta$, each intersection point is classified as PE (potentially entering) or PL (potentially leaving).

- PE with the largest $t$ and PL with the smallest $t$ provide the domain of $p(t)$ inside $W$.

- The domain, if inverted, signals that $p(t)$ is totally outside.
Cyrus-Beck Line Clipping

precalculate Ni and select a Pei for each edge;

for each line segment to be clipped
  if ( P1 = P2 ) then
    line is degenerate so clip as a point;
  else {
    tPE = 0;  tPL = 1;
    for each candidate intersection with a clip edge
      if (( <Ni, D> ) <> 0 ) then {
        /* Ignore edges parallel to line for now */
        calculate t;
        sign of <Ni, D> categorizes as PE or PL ;
        if  PE then tPE = max (tPE, t);
        if  PL then tPL = min  (tPL, t);
      }
      if  (tPE > tPL) return null
      else
        return P(tPE) and P(tPL) ;
    /* as true clip intersections */
  };

Clipping

- Avoid drawing parts of primitives outside window
  - Points
  - Lines
  - Polygons
  - Circles
  - etc.

2D Screen Coordinates
Polygon Clipping

- Find the part of a polygon inside the clip window?

Before Clipping
Polygon Clipping

- Find the part of a polygon inside the clip window?

After Clipping
Sutherland Hodgeman Clipping

- Clip to each window boundary one at a time

After each clipping a new set of vertices is produced.
Sutherland Hodgeman Clipping

- Clip to each window boundary one at a time
Sutherland Hodgeman Clipping

- Clip to each window boundary one at a time
Sutherland Hodgeman Clipping

- Clip to each window boundary one at a time
Sutherland Hodgeman Clipping

- Clip to each window boundary one at a time
Clipping to a Boundary

- Do inside test for each point in sequence,
- Insert new points when cross window boundary,
- Remove points outside window boundary
Clipping to a Boundary

- Do inside test for each point in sequence,
- Insert new points when cross window boundary,
- Remove points outside window boundary
Clipping to a Boundary

- Do inside test for each point in sequence,
  Insert new points when cross window boundary,
  Remove points outside window boundary
Clipping to a Boundary

- Do inside test for each point in sequence,
- Insert new points when cross window boundary,
- Remove points outside window boundary
Clipping to a Boundary

- Do inside test for each point in sequence,
  Insert new points when cross window boundary,
  Remove points outside window boundary

![Diagram showing clipping to a boundary with points P1, P2, P3, P4, P5, and P']. The window boundary is indicated with a black line, and inside points are marked with green dots, while outside points are marked with red dots. The diagram illustrates the process of clipping points to the boundary.
Clipping to a Boundary

- Do inside test for each point in sequence,
  Insert new points when cross window boundary,
  Remove points outside window boundary
Clipping to a Boundary

- Do inside test for each point in sequence,
- Insert new points when cross window boundary,
- Remove points outside window boundary
Clipping to a Boundary

- Do inside test for each point in sequence,
- Insert new points when cross window boundary,
- Remove points outside window boundary
Clipping to a Boundary

- Do inside test for each point in sequence,
  - Insert new points when cross window boundary,
  - Remove points outside window boundary
For each clip edge - consider the relation between successive vertices of the polygon; Assume vertex s has been dealt with, vertex p follows:

- p added to output list
- i added to output list
- no output
- i and p added to output list
Sutherland - Hodgman polygon Clipping Algorithm

type
  vertex = point;  /* point holds real x, y */
  edge = array [1..2] of vertex;
/* Max declared as constant */
  vertexArray = array [1..MAX] of vertex;

procedure SutherlandHodgmanPolygonClip (  
inVertexArray: vertexArray;  /* input vertex array */
  var outVertexArray: vertexArray;  /* output vertex array */
  inLength: integer;  /* num of entries in inVertexArray */
  var outLength: integer;  /* num of entries in outVertexArray */
  clipBoundary : edge  /* Edge of clip polygon */
);
Sutherland - Hodgman (cont.)

var
    s, p /* Start, End point of current polygon edge */
    i : vertex; /* Intersection point with clip boundary */
    j : integer; /* vertex loop counter */
begin
    outLength :=0;
    s := inVertexArray [inLength];
    /* Start with the last vertex in inVertexArray*/
    for j:=1 to inLength do
    begin
        p := inVertexArray[j ];
        if Inside (p, clipBoundary ) then
            if Inside (s, clipBoundary ) then
                Output (p,outLength, outVertexArray) /*case #1*/
            else
                begin   /* case # 4 */
                    Intersect (s, p, clipBoundary, i);
                    Output (i, outLength, outVertexArray );
                    Output (p, outLength, outVertexArray );
                end;
        else
            if Inside (s, clipBoundary ) then
                begin   /* case # 2 */
                    Intersect (s, p, clipBoundary, i);
                    Output (i, outLength, outVertexArray );
                end;
        else
            s := p ; /* Advance to next pair of vertices */
    end    /* for */
end;   /* SutherlandHodgmanPolygonClip */
2D Rendering Pipeline

8D Primitives

2D Primitives

Clipping

Clip portions of geometric primitives residing outside the window

Viewport Transformation

Transform the clipped primitives from screen to image coordinates

Scan Conversion

Fill pixels representing primitives in screen coordinates

Image
Viewport Transformation

- Transform 2D geometric primitives from screen coordinate system (normalized device coordinates) to image coordinate system (pixels)
Viewport Transformation

- Window-to-viewport mapping

```
vx = vx1 + (wx - wx1) * (vx2 - vx1) / (wx2 - wx1);
yy = vy1 + (wy - wy1) * (vy2 - vy1) / (wy2 - wy1);
```
Summary of Transformations

\[ \mathbf{p}(x,y,z) \]

- \textbf{Modeling Transformation}:
  - 3D Object Coordinates
- \textbf{Viewing Transformation}:
  - 3D World Coordinates
  - 3D Camera Coordinates
- \textbf{Projection Transformation}:
  - 2D Screen Coordinates
- \textbf{Window-to-Viewport Transformation}:
  - 2D Image Coordinates

\[ \mathbf{p}'(x',y') \]

\{ \text{Modeling transformation} \}
\{ \text{Viewing transformations} \}
\{ \text{Viewport transformation} \}
Summary

3D Primitives

↓

2D Primitives

↓

Clipping

Clip portions of geometric primitives residing outside the window

↓

Viewport Transformation

Transform the clipped primitives from screen to image coordinates

↓

Scan Conversion

Fill pixels representing primitives in screen coordinates

↓

Image