OpenGL automatically does clipping to the world window
However, it’s useful to know how to do clipping
- May be using a different package that doesn’t do clipping
Many different clipping algorithms
We’ll study one, the Cohen-Sutherland Clipping Alg.
It determines which part (if any) of a line segment falls within the clipping region
Reports back the endpoints of that portion
These slides cover information available from Ch. 6 in the book. Please see Ch. 6 for other algorithms covered in class.

This diagram illustrates various combinations of endpoint locations:

FIGURE 3.18 Clipping lines at window boundaries.

Trivial Cases
Some combinations of endpoints are easy to handle:

FIGURE 3.19 Trivial acceptance or rejection of a line segment.

Basic Operation
Determining Position
Need a simple way to test for position of line segment relative to the clipping region
- Idea: compute outcodes
  - Called inside-outside code words in text
- Basic concept:
  - Associate one bit with each edge of the clipping region
  - Compute bit values for point – 1 if “outside” relative to that edge, 0 if “inside”

FIGURE 3.20 Examining how point P is disposed with respect to the window.

Outcodes
Code bits follow clipping region in clockwise order
Each point will have one of nine code combinations:

FIGURE 3.21 Inside–outside codes for a point.

- Trivial accept: OR of outcodes = FFFF
- Trivial reject: AND of outcodes ≠ FFFF

Pseudocode

```c
int clipSegment(Point2 p1, Point2 p2, RealRect w) {
    do {
        recompute_outcodes;
        if (trivial_accept) return 1;
        if (trivial_reject) return 0;
        if (p1.is_outside) {
            if (p1.is_left) clip_against_left_edge
            else if (p1.is_right) clip_against_right_edge
            else if (p1.is_below) clip_against_bottom_edge
            else if (p1.is_above) clip_against_top_edge
        } else if (p2.is_outside) {
            if (p2.is_left) clip_against_left_edge
            else if (p2.is_right) clip_against_right_edge
            else if (p2.is_below) clip_against_bottom_edge
            else if (p2.is_above) clip_against_top_edge
        } else {
            // p2 must be outside
            if (p1.is_left) clip_against_left_edge
            else if (p1.is_right) clip_against_right_edge
            else if (p1.is_below) clip_against_bottom_edge
            else if (p1.is_above) clip_against_top_edge
        }
    } while (1);
    return 1;
}
```

- Algorithm guaranteed to terminate
  - Each iteration clips against one boundary
  - After at most four iterations, remaining segment is either entirely inside or entirely outside
How to Clip? Similar Triangles

• We clip by replacing point \( p1 \) with point \( A \).
  - Ratio: \( \Delta y/\Delta x = e/d \)
  - Can calculate \( e = P1.x - W.right \), \( \Delta x = P2.x - P1.x \), \( \Delta y = P2.y - P1.y \)
  - Then, \( \Delta y = e \cdot (P2.y - P1.y) / (P2.x - P1.x) = e \cdot \Delta y / \Delta x \)
  - \( A.x = W.right \)
  - \( A.y = P1.y - \Delta y \)

Example - Similar Triangles

• Clipping region: \( x = 3..9, y = 4..8 \)
• Line: \( (7,5) - (11,8) \)
• \( \Delta x = 11 - 7 = 4 \)
• \( \Delta y = 8 - 5 = 3 \)
• \( e = 11 - 9 = 2 \)
• \( d = \frac{e \cdot \Delta y}{\Delta x} = \frac{2 \cdot 3}{4} = \frac{6}{4} = 1.5 \)
• \( A.x = 9 \)
• \( A.y = P1.y - d = 8 - 1.5 = 6.5 \)
• \( \therefore A = (9, 6.5) \)

How to Clip? Slope/Intercept Form

• Can calculate intercept from \( y = mx + b \) form
  - \( m = (p1.y - p2.y) / (p1.x - p2.x) \)
  - \( b = p1.y - m \cdot p1.x \)
  - \( A.x = W.right \)
  - \( A.y = m \cdot W.right + b \)

Example - Slope/Intercept Form

• Clipping region: \( x = 3..9, y = 4..8 \)
• Line: \( (7,5) - (11,8) \)
• \( m = \frac{8-5}{11-7} = \frac{3}{4} = 0.75 \)
• \( b = y - mx = 5 - 0.75 \cdot 7 = 5 - 5.25 = -0.25 \)
• \( A.x = 9 \)
• \( A.y = 0.75 \cdot 9 - 0.25 = 6.75 - 0.25 = 6.5 \)
• \( \therefore A = (9, 6.5) \)

Notes on C-S Algorithm

• This version of the algorithm clips against left, right, bottom, and top, in that order
• Order doesn’t matter unless we know something about the positions of the vertices
  - Could optimize if, e.g., we knew that both points were above \( W.b \)

Worst case:

Example - Polygon Clipping

• Consider this clipped figure:
  - What is the original?
Polygon Clipping

- If we use a line clipper, we don't know
  - The figure could have come from either source
- Also, it is no longer a polygon
  - It's now simply four line segments
- We want clipped polygons to remain polygons
  - This requires a different approach to clipping

Sutherland-Hodgman Clipping Algorithm

- This is a general-purpose polygon clipper
- Works against convex and concave polygons
- Can be used to clip against any convex clipping polygon
  - We'll look at a rectangle, but it could be a pentagon, etc.
- Input: a collection of vertices \(v_1, v_2, \ldots, v_n\)
- Output: another series of vertices defining the clipped polygon
- Method: apply a general “clip against edge” routine to each side of the clip region in sequence

Cases

- There are four cases which must be dealt with:

Method

- Start with a vertex list:
  \(A, B, C, D, E, F, G, H, I, J\)
- Clip against the rectangle in this order:
  - Right, top, bottom, left
- At each step, replace “outside” vertices with one or more “edge” vertices
- Example: clip against right edge
  - Replace \(G\) with \(G', G''\)
  - Replace \(E\) with \(E', E''\)
  - Output vertex list:
    \(A, B, C, D, E', E'', F, G', G'', H, I, J\)

Next, clip against top
- Replace \(E'\) with \(E'''\)
- Replace \(C\) with \(C', C''\)
- Output list:

Clip against bottom
- Replace \(I\) with \(I', I''\)

Clip against left edge
- No changes!

Final output list:
Implementation

- Makes use of support routines:
  - `inside(point, boundary)` - is point inside boundary?
  - `output(point, length, vector)` - put point into vector, update length
  - `intersect(spoint, epoint, boundary, newpoint)` - put intersection point into newpoint parameter

- Clip in the order of right, top, bottom and left edges.
- We start with the last vertex in the input list as the first predecessor, and apply two rules:
  - Does the line defined by this vertex and its predecessor intersect this edge?
    - Yes → add intersection point to output list
  - Is the vertex itself inside the edge?
    - Yes → add vertex to output list

```c
Implementation

// This algorithm does one side - to clip entire figure:
SHPC( in, out1, inlen, outlen1, edge1 );
SHPC( out1, out2, outlen1, outlen2, edge2 );
SHPC( out2, out3, outlen2, outlen3, edge3 );
SHPC( out3, out4, outlen3, outlen4, edge4 );
```

```c
Implementation

SHPC( inVertices, outVertices, inLength, outLength, clipBoundary ) {
  outLength = 0;
p = inVertices[ inLength - 1 ];
for ( j=0; j<inLength; j++ ) {
  v = inVertices[ j ];
  // Cases 1 & 4
  if ( inside( v, clipBoundary ) ) {
    if ( inside( p, clipBoundary ) ) {
      // Case 1
      output( v );
    } else { // Case 4
      intersect( p, v, clipBoundary, i);
      output(i);
      output(v);
    }
  } else { // Cases 2 & 3
    if ( inside( p, clipBoundary ) ) { // Case 2
      intersect( p, v, clipBoundary, i);
      output(i);
    } else { // Case 3 has no output
    }
    p = v;
  }
}
```

```c
Implementation

Implementation

// This algorithm does one side - to clip entire figure:
SHPC( in, out1, inlen, outlen1, edge1 );
SHPC( out1, out2, outlen1, outlen2, edge2 );
SHPC( out2, out3, outlen2, outlen3, edge3 );
SHPC( out3, out4, outlen3, outlen4, edge4 );
```