Clipping

- Remember, we are in implementation mode now, and so we need to worry about not drawing things out of bounds — a process called *clipping*.
- Clipping isn’t just deleting points that are outside our desired bounds; it also includes calculating where shapes *intersect* the bounds, since that’s what we’ll actually draw.

Clipping Algorithms

- Many clipping algorithms — in pure form, they are independent of number of dimensions, but for simplicity we will look at the 2D versions.
- General criteria for a good clipping algorithm:
  - Quickly identify lines (or polygon edges) that need not be drawn at all: these can be skipped and discarded.
  - Quickly identify lines (or polygon edges) that are completely contained within the drawing bounds: these can be drawn without modification.
  - For the lines that are neither (i.e., they cross display boundaries), calculate their intersections as quickly as possible: the less of these we do, the better.
Cohen-Sutherland Clipping

Translate boundary crossings into bit fields; for example, in 2D this is 4 bits, one each to represent in/out the top, bottom, right, or left boundaries (0 = in bounds, 1 = out of bounds)

<table>
<thead>
<tr>
<th>1001</th>
<th>1000</th>
<th>1010</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>0000</td>
<td>0010</td>
</tr>
<tr>
<td>0101</td>
<td>0100</td>
<td>0110</td>
</tr>
</tbody>
</table>

- For every vertex \((x, y)\), initially assign 0000
- If \(x > R\) then set bit 1 else if \(x < L\) then set bit 0
- If \(y > T\) then set bit 3 else if \(y < B\) then set bit 2
  ◇ Note how when \(x = R\) or \(x = L\), or \(y = T\) or \(y = B\), the corresponding bit is still zero; equality = in bounds
- Now, for every line segment \((x_1, y_1)\) to \((x_2, y_2)\):
  ◇ If both line segments are 0000, then draw line; we’re done (accept)
  ◇ If \(code_1 \& code_2\) (bitwise and), then skip line; we’re done (reject)
  ◇ Otherwise, calculate the intersection with a boundary; re-encode then reconsider this new pair of endpoints
Liang-Barsky Clipping

- Remember linear interpolation? It allows us to write a line in terms of a parameter $u$ from 0.0 to 1.0: $x = x_1 + u(x_2 - x_1), y = y_1 + u(y_2 - y_1)$

- Liang-Barsky asks: for what values of $u$ does a line segment enter or exit the bounds?

- There can be, at most, two of each; we care about the maximum entry value and the minimum exit value

- For each line segment, for each boundary, check the value of $u$ at the intersection of the segment’s line with that boundary
  - If $u < 0$ on entry and $u > 1$ on exit — accept
  - If $u > 1$ on entry or $u < 0$ on exit — reject
  - If $u$ on entry > $u$ on exit — reject
  - Otherwise, clip and try again — note how we don’t need to perform an extra calculation, because the new point can be derived from $u$

Line 1: max entry < 0, min exit > 1 — accept
Line 2: max entry > 1, min exit > 1 — reject
Line 3: max entry < 0, min exit < 0 — reject
Line 4: max entry > min exit — reject
Line 5: max entry > 0, min exit < 1, max entry < min exit — clip
Liang-Barsky Algorithm

• For a given line segment \((x_1, y_1)\) to \((x_2, y_2)\), derive the parametric form of its line: 
  \[ x = x_1 + u(x_2 - x_1), \quad y = y_1 + u(y_2 - y_1) \]

• For each boundary (L, R, T, B), calculate the value of \(u\) for that line at that boundary; note that a point is within the boundary if:
  \[ L \leq x \leq R \text{ and } B \leq y \leq T \]

• Substituting the parametric form, let:
  \[ dx = x_2 - x_1, \quad dy = y_2 - y_1 \]
  \[ L \leq x_1 + u(dx) \leq R \text{ and } B \leq y_1 + u(dy) \leq T \]

• If we break these inequalities up, we get these conditions:
  
  \(-dx \ (u) \leq x_1 - L \rightarrow \text{let } C = -dx, \ q = x_1 - L\)
  \[ dx \ (u) \leq R - x_1 \rightarrow \text{let } C = dx, \ q = R - x_1 \]
  
  \(-dy \ (u) \leq y_1 - B \rightarrow \text{let } C = -dy, \ q = y_1 - B\)
  \[ dy \ (u) \leq T - y_1 \rightarrow \text{let } C = dy, \ q = T - y_1 \]
• Note how, for each $C$ and its corresponding boundary:

  $C < 0 \Rightarrow$ line goes out $\rightarrow$ in: entry

  $C > 0 \Rightarrow$ line goes in $\rightarrow$ out: exit

  $C = 0 \Rightarrow$ line is parallel to the boundary

• So, we can calculate $u$ for each boundary by calculating $q$ and $C$; the value of $C$ tells us if we are looking at an entry or exit point for the boundary. Thus, we can now apply the conditions:

  ◦ If $u < 0$ on entry and $u > 1$ on exit — accept

  ◦ If $u > 1$ on entry or $u < 0$ on exit — reject

```pascal
procedure ClipAndDrawLine(x1, y1, x2, y2: real) is
  u1: real := 0.0;    dx: real := x2 - x1;
  u2: real := 1.0;    dy: real := y2 - y1;

  function Reject(C, q: real) return boolean is
    u: real := q / C;
    begin
      if C < 0 then
        if u > u2 then return true; elsif u > u1 then u1 := u; end if;
        elsif C > 0 then
          if u < u1 then return true; elsif u > u2 then u2 := u; end if;
          else
            if q < 0 then return true;
            end if;
            return false;
      end Reject;
      begin
        if Reject(-dx, x1 - L) then return; end if;
        if Reject(dx, R - x1) then return; end if;
        if Reject(-dy, y1 - B) then return; end if;
        if Reject(dy, T - y1) then return; end if;
        if u2 < 1.0 then (x2, y2) := (x1 + u2 * dx, y1 + u2 * dy); end if;
        if u1 > 0.0 then (x1, y1) := (x1 + u1 * dx, y1 + u1 * dy); end if;
        DrawLine(x1, y1, x2, y2);
      end ClipAndDrawLine;
```
How About Polygons?

- Clipping polygons complicates the problem because we have to take account the enclosed area.

- Also, where line segment clipping always maps two points to two (possibly other) points, polygon clipping may actually change the number of points after clipping.

- In the ultimate extreme case, we may even end up with more than one polygon after clipping!
Sutherland-Hodgman Polygon Clipping

If we consider a clipping algorithm to be a black box that takes a set of vertices then produces a new set of clipped vertices, then without loss of generality we can clip along one border at a time and string them together — like a pipeline:

---

**Diagram:**

- Unclipped vertices
- Top clipper
- Bottom clipper
- Right clipper
- Left clipper
- Completely clipped vertices

---

**Output for this border:** ABFEA
How About 3D?

All algorithms extrapolate to 3D:

- Cohen-Sutherland: Add inside/outside bits for z axis
- Liang-Barsky: Add parametric equation for z coordinate
- Sutherland-Hodgman: Add clipping stages for near and far planes