OpenGL Stages

- After projection, several stages before objects drawn to screen
- These stages are non-programmable
Primitive Assembly

- Transformations and projections applied to vertices individually
- **Primitive assembly**: After transforms, projections, individual vertices grouped back into primitives
- **E.g. v6, v7 and v8** grouped back into triangle
Clipping

- Subsequent operations necessary before display occur per-primitive
- **Clipping**: Remove primitives (lines, polygons, text, curves) outside view frustum

![Clipping lines](image1)

![Clipping polygons](image2)
Rasterization

- Determine which pixels that primitives map to
  - Fragment generation
  - Rasterization or scan conversion
Fragment Processing

- Some tasks deferred until fragment processing
Clipping

- 2D and 3D clipping algorithms
  - 2D against clipping window
  - 3D against clipping volume
- 2D clipping
  - Lines
  - Polygons
  - Curves
  - Text
Clipping 2D Line Segments

- **Brute force approach**: compute intersections with all sides of clipping window
  - Inefficient: one division per intersection
2D Clipping: Cohen-Sutherland Algorithm

- Idea: eliminate as many cases as possible without computing intersections
- Start with four lines that determine the sides of the clipping window

```
<table>
<thead>
<tr>
<th>x = x_{min}</th>
<th>y = y_{max}</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

y = y_{min}  x = x_{max}
```
Determine whether a point \((x,y)\) is inside or outside of the world window?

If \((x_{\text{min}} \leq x \leq x_{\text{max}})\) and \((y_{\text{min}} \leq y \leq y_{\text{max}})\) then the point \((x,y)\) is inside else the point is outside
Clipping Lines

3 cases:
- **Case 1:** All of line in
- **Case 2:** All of line out
- **Case 3:** Part in, part out
Clipping Lines: Trivial Accept

Case 1: All of line in
Test line endpoints:

\[ X_{\text{min}} \leq P1.x, \, P2.x \leq X_{\text{max}} \quad \text{and} \quad Y_{\text{min}} \leq P1.y, \, P2.y \leq Y_{\text{max}} \]

Note: simply comparing x,y values of endpoints to x,y values of rectangle

Result: trivially accept.
Draw line in completely
Clipping Lines: Trivial Reject

Case 2: All of line out
Test line endpoints:

- $p1.x, p2.x \leq X_{min}$ OR
- $p1.x, p2.x \geq X_{max}$ OR
- $p1.y, p2.y \leq y_{min}$ OR
- $p1.y, p2.y \geq y_{max}$

Note: simply comparing x,y values of endpoints to x,y values of rectangle

Result: trivially reject.
Don’t draw line in
Clipping Lines: Non-Trivial Cases

Case 3: Part in, part out

Two variations:
- One point in, other out
- Both points out, but part of line cuts through viewport

Need to find inside segments

Use similar triangles to figure out length of inside segments

\[
\frac{d}{dely} = \frac{e}{delx}
\]
If chopping window has 
(left, right, bottom, top) = (30, 220, 50, 240),
what happens when the following lines are chopped?

(a) p₁ = (40,140), p₂ = (100, 200)

(b) p₁ = (20,10), p₂ = (20, 200)

(c) p₁ = (100,180), p₂ = (200, 250)
int clipSegment(Point2& p1, Point2& p2, RealRect W)
{
    do{
        if(trivial accept) return 1; // whole line survives
        if(trivial reject) return 0; // no portion survives
        // now chop
        if(p1 is outside)
            // find surviving segment
        {
            if(p1 is to the left) chop against left edge
            else if(p1 is to the right) chop against right edge
            else if(p1 is below) chop against the bottom edge
            else if(p1 is above) chop against the top edge
        }
    }
else // p2 is outside
    // find surviving segment
{
    if(p2 is to the left) chop against left edge
    else if(p2 is to right) chop against right edge
    else if(p2 is below) chop against the bottom edge
    else if(p2 is above) chop against the top edge
}
}while(1);
Using Outcodes to Speed Up Comparisons

- For each endpoint, define an outcode

\[ b_0 b_1 b_2 b_3 \]

\[ b_0 = 1 \text{ if } y > y_{\text{max}}, \quad 0 \text{ otherwise} \]
\[ b_1 = 1 \text{ if } y < y_{\text{min}}, \quad 0 \text{ otherwise} \]
\[ b_2 = 1 \text{ if } x > x_{\text{max}}, \quad 0 \text{ otherwise} \]
\[ b_3 = 1 \text{ if } x < x_{\text{min}}, \quad 0 \text{ otherwise} \]

- Outcodes divide space into 9 regions
- Trivial accept/reject becomes bit-wise comparison
References