OpenGL Stages

- After projection, several stages before objects drawn to screen
- These stages are **NOT** programmable

Vertex shader: programmable

In hardware: **NOT** programmable
Hardware Stage: Primitive Assembly

- Up till now: 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
Hardware Stage: Clipping

- After primitive assembly, subsequent operations are per-primitive
- **Clipping**: Remove primitives (lines, polygons, text, curves) outside view frustum (canonical view volume)
Rasterization

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

- Some tasks deferred until fragment processing

Hidden Surface Removal

Antialiasing

Transformation
Projection

Hidden surface Removal Antialiasing

Modeling → Geometric processing → Rasterization → Frame buffer
Clipping

- 2D and 3D clipping algorithms
  - 2D against clipping window
  - 3D against clipping volume

- 2D clipping
  - Lines (e.g. dino.dat)
  - 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

- **Better Idea:** eliminate as many cases as possible without computing intersections
- Cohen-Sutherland Clipping algorithm

```
\begin{align*}
  x &= x_{\text{min}} \\
  y &= y_{\text{max}} \\

  x &= x_{\text{max}} \\
  y &= y_{\text{min}}
\end{align*}
```
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:

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

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) $p_1 = (40,140)$, $p_2 = (100, 200)$

(b) $p_1 = (20,10)$, $p_2 = (20, 200)$

(c) $p_1 = (100,180)$, $p_2 = (200, 250)$

\[
\frac{d}{dely} = \frac{e}{delx}
\]
Cohen-Sutherland pseudocode (Hill)

```cpp
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
        }
    }
```
Cohen-Sutherland pseudocode (Hill)

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);
Liang-Barsky 3D Clipping

**Goal:** Clip object edge-by-edge against Canonical View volume (CVV)

**Problem:**
- 2 end-points of edge: \( A = (Ax, Ay, Az, Aw) \) and \( C = (Cx, Cy, Cz, Cw) \)
- If edge intersects with CVV, compute intersection point \( I = (lx, ly, lz, lw) \)
Determining if point is inside CVV

Problem: Determine if point $(x,y,z)$ is inside or outside CVV?

Point $(x,y,z)$ is inside CVV if

$$(-1 \leq x \leq 1) \quad \text{and} \quad (-1 \leq y \leq 1) \quad \text{and} \quad (-1 \leq z \leq 1)$$

else point is outside CVV

CVV == 6 infinite planes $(x=-1,1; \ y=-1,1; \ z=-1,1)$
Determining if point is inside CVV

- If point specified as (x,y,z,w)
  - Test \((x/w, y/w, z/w)\)!

\[
\begin{align*}
\text{if} & \quad (-1 \leq x/w \leq 1) \\
\text{and} & \quad (-1 \leq y/w \leq 1) \\
\text{and} & \quad (-1 \leq z/w \leq 1)
\end{align*}
\]

else point is outside CVV
Modify Inside/Outside Tests Slightly

Our test: \((-1 < \frac{x}{w} < 1)\)

Point \((x,y,z,w)\) inside plane \(x = 1\) if

\[
\frac{x}{w} < 1 \\
\Rightarrow w - x > 0
\]

Point \((x,y,z,w)\) inside plane \(x = -1\) if

\[
-1 < \frac{x}{w} \\
\Rightarrow w + x > 0
\]
Numerical Example: Inside/Outside CVV Test

- Point \((x,y,z,w)\) is
  - inside plane \(x=-1\) if \(w+x > 0\)
  - inside plane \(x=1\) if \(w-x > 0\)

- Example Point \((0.5, 0.2, 0.7)\) inside planes \((x = -1, 1)\) because \(-1 \leq 0.5 \leq 1\)

- If \(w = 10\), \((0.5, 0.2, 0.7) = (5, 2, 7, 10)\)
- Can either divide by \(w\) then test: \(-1 \leq 5/10 \leq 1\) OR

  To test if inside \(x = -1\), \(w + x = 10 + 5 = 15 > 0\)
  To test if inside \(x = 1\), \(w - x = 10 - 5 = 5 > 0\)
3D Clipping

- Do same for y, z to form boundary coordinates for 6 planes as:

<table>
<thead>
<tr>
<th>Boundary coordinate (BC)</th>
<th>Homogenous coordinate</th>
<th>Clip plane</th>
<th>Example (5,2,7,10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BC0</td>
<td>w+x</td>
<td>x=-1</td>
<td>15</td>
</tr>
<tr>
<td>BC1</td>
<td>w-x</td>
<td>x=1</td>
<td>5</td>
</tr>
<tr>
<td>BC2</td>
<td>w+y</td>
<td>y=-1</td>
<td>12</td>
</tr>
<tr>
<td>BC3</td>
<td>w-y</td>
<td>y=1</td>
<td>8</td>
</tr>
<tr>
<td>BC4</td>
<td>w+z</td>
<td>z=-1</td>
<td>17</td>
</tr>
<tr>
<td>BC5</td>
<td>w-z</td>
<td>z=1</td>
<td>3</td>
</tr>
</tbody>
</table>

- Consider line that goes from point A to C
  - **Trivial accept:** 12 BCs (6 for pt. A, 6 for pt. C) > 0
  - **Trivial reject:** Both endpoints outside (-ve) for same plane
References