CSC 240 Computer Graphics
Line Clipping

Nick Howe
Smith College
Q. Are all lines straight when doing line clipping (so curves are just broken into many small lines)? Or is it possible to have curved lines that have endpoints outside the viewport and the rest of the line inside the viewport?

A. Clipping is done for straight line segments. Curves would be composed of many short segments, and we would need to check the clipping for each.

Q. How do you know when to use & or the |? Just by looking at the line?

A. Use | to set bits. Use & to compare bits.

Q. How would we write the code to get the 1 and 0 strings?
Endpoint Coding

Let \((x,y) = (200,-120)\), & viewport shown below

\[
\text{getCode}(x,y): \\
\text{code} \leftarrow 0000 \quad \text{if } (x < x_{\min}) \text{ then} \\
\quad \text{code} \leftarrow \text{code}|0001 \quad 200 < -150 : \text{false} \\
\quad \text{code} \leftarrow \text{code}|0010 \quad 200 > 150 : \text{true} \\
\quad \text{if } (x > x_{\max}) \text{ then} \\
\quad \text{code} \leftarrow \text{code}|0010 \\
\quad \text{if } (y < y_{\min}) \text{ then} \\
\quad \text{code} \leftarrow \text{code}|1000 \quad -120 < -100 : \text{true} \\
\quad \text{code} \leftarrow \text{code}|1010 \quad -120 > 100 : \text{false} \\
\quad \text{if } (y > y_{\max}) \text{ then} \\
\quad \text{code} \leftarrow \text{code}|0100 \\
\text{return code}
\]
Q. How to determine whether clipping is required?

A. First check whether it is case 1. Is \texttt{code1 | code2 == 0000}?

Then check whether it is case 2. Is \texttt{code1 & code2 != 0000}?

If neither, then clip the line and check again.

Q. Why is right the x min instead of left?

A. The codes are TBRL (top, bottom, right, left). The check for the left side is against \( x_{\text{min}} \).
Q. How do you evaluate case 3 given 2 endpoints? Is it the same as case 2 where any 0 (in the endpoint code) results in a 0?

A. We know it is not case 1, so \( \text{code1} \mid \text{code2} \neq 0000 \) and there must be at least one point with a non-zero code. Also it is not case 2, so \( \text{code1} \& \text{code2} == 0000 \). So there is a chance of crossing the viewport.

We clip one of the endpoints and check again.
Your Questions

Q. Are we assigning code as number or string "0000"?
A. It is a bit sequence, or base-2 number. Visually we can represent it as a string of 0/1 digits. In memory it is encoded differently. For example, 1101 is encoded as $1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 8 + 4 + 0 + 1 = 13$

Q. What does "&" do? How is it different from "&&"?
A. The & operator compares bit sequences and outputs bit sequence results. The && operator compares boolean values and outputs a boolean result.

Q. Why is the code top, bottom, right, left, instead of top, bottom, left, right?
A. The order is arbitrary, and not universal. I happened to choose TBRL.
Cohen-Sutherland Pseudocode

cohen_sutherland(p1,p2):
    code1 = getCode(p1);
    code2 = getCode(p2);
    if (code1|code2 == 0000) then // case 1
        drawLine(p1,p2) STOP
    else if (code1 & code2 != 0000) then // case 2
        // skip line STOP
    else if (code1 & 1000) then // case 3a
        p1new.x ← (y_{min}-p1.y)/m+p1.x, p1new.y ← y_{min},
        cohen_sutherland(p1new,p2)
    else if (code1 & 0100) then // case 3b
        p1new.x ← (y_{max}-p1.y)/m+p1.x, p1new.y ← y_{max},
        cohen_sutherland(p1new,p2)
else if (code1 & 0010) then // case 3c
    p1new.y ← m(x_{max} - p1.x) + p1.y, p1new.x ← x_{max},
    cohen_sutherland(p1new, p2)
else if (code1 & 0001) then // case 3d
    p1new.y ← m(x_{min} - p1.x) + p1.y, p1new.x ← x_{min},
    cohen_sutherland(p1new, p2)
else if (code2 & 1000) then // case 3e
    p2new.x ← (y_{min} - p2.y)/m + p2.x, p2new.y ← y_{min},
    cohen_sutherland(p1, p2new)
else if (code2 & 0100) then // case 3f
    p2new.x ← (y_{max} - p2.y)/m + p2.x, p2new.y ← y_{max},
    cohen_sutherland(p1, p2new)
else if (code2 & 0010) then // case 3g
    p2new.y ← m(x_{max} - p2.x) + p2.y, p2new.x ← x_{max},
    cohen_sutherland(p1, p2new)
else if (code2 & 0001) then // case 3h
    p2new.y ← m(x_{min} - p2.x) + p2.y, p2new.x ← x_{min},
    cohen_sutherland(p1, p2new)
Other Questions?
Cohen-Sutherland Line Clipping

In this problem, you are given a clipping window (defining the “viewport”) and an example line, and asked to perform the steps of the line clipping algorithm we just learned in class. Assume the origin is at the top left and y is increasing going down (like HTML canvas).

Input: viewport defined by the lines $x_{\text{min}} = 2$, $x_{\text{max}} = 10$, $y_{\text{min}} = 3$, $y_{\text{max}} = 8$.
   line defined by the points $p_1 = (1, 5)$ and $p_2 = (11, 1)$.
Output: $p'_1$ and $p'_2$, the points defining the line that should actually be drawn.

1. Draw out the viewport and the example line, labeling $p_1, p_2, x_{\text{min}}, x_{\text{max}}, y_{\text{min}},$ and $y_{\text{max}}$.
2. Write out the binary 4-digit codes for $p_1$ and $p_2$.
3. Write out what case each point falls under and show how the algorithm would update the points. What are the final $p'_1$ and $p'_2$?
4. Label $p_1'$ and $p_2'$ on your picture and make sure they agree visually with your calculations.

5. How many “rounds” of clipping are required to make this example line within the viewport?
6. Why are there eight subparts to case 3? What does each one represent/do?

7. A four-bit sequence allows for sixteen possible values, yet we only have nine regions. Which bit sequences are not used, and why do they represent nonsensical situations?

8. Which case would be activated for each of the following pairs of codes?
   a. 1010 and 0101
   b. 0000 and 1001
   c. 0110 and 0101
   d. 0001 and 0001