

14 Clipping

- Fundamentals
- Cohen-Sutherland
- Parametric
- Polygons
- Circles and Curves
- Text

Basic Concepts

The purpose of clipping is to remove objects or parts of objects that lie outside the view volume. Operationally the acceleration in the conversion to pixels (scan conversion) is sufficient to make the process worthwhile.

find parts of primitives within clip region

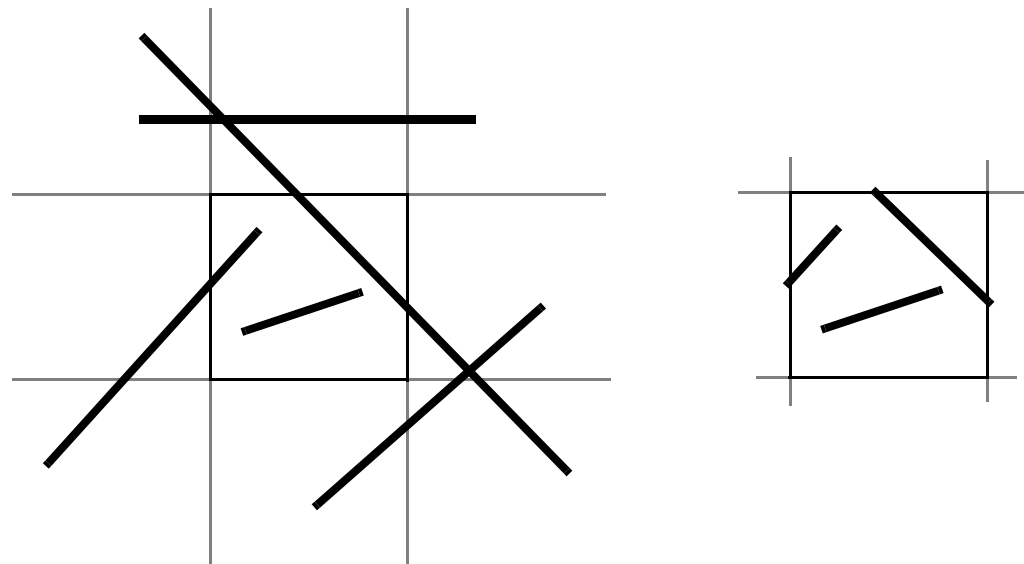
- often rectangular
- basic case

$$x_{min} \leq x \leq x_{max}$$

$$y_{min} \leq y \leq y_{max}$$

clipping usually applied to:

- points
- lines
- polygons
- curves
- text

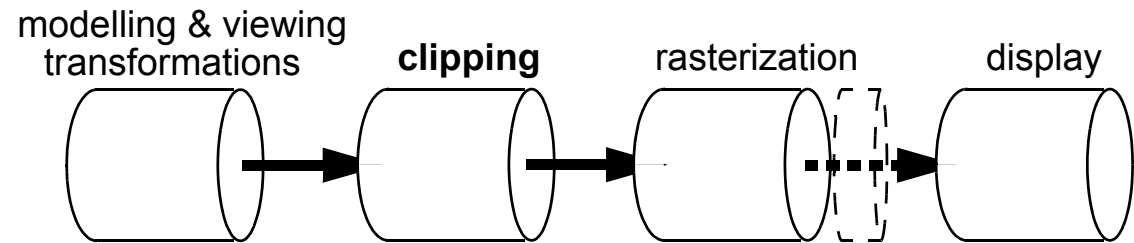


Clipping versus Scissoring

Clipping is carried out on the transformed coordinates in the canonical view volume (CVV). Scissoring is carried out on the pixel in memory, and is usually used to mask of parts of an image.

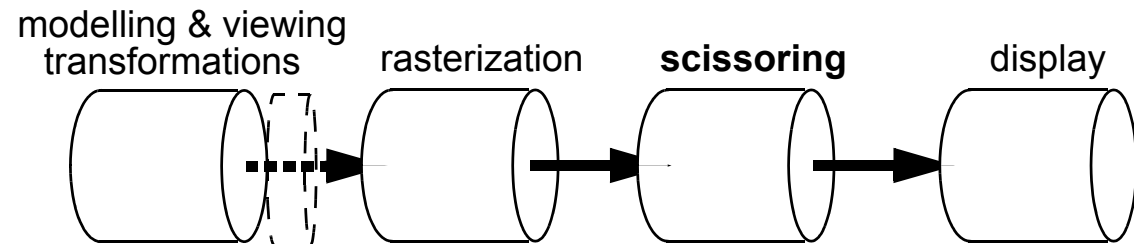
Clipping:

- operates with primitives
- discards, creates or modifies
- computationally expensive
- usually accelerates rasterization



Scissoring:

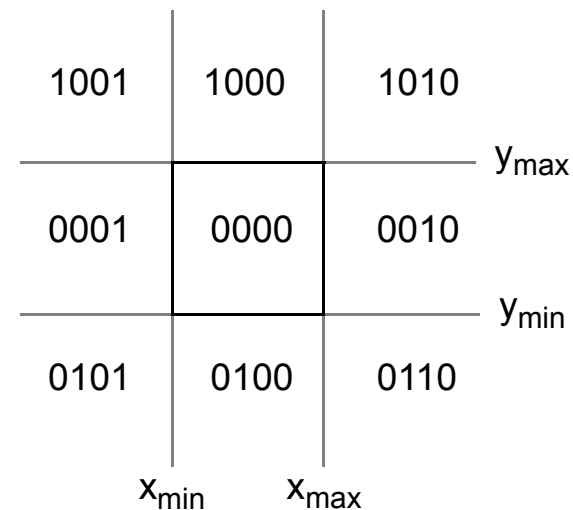
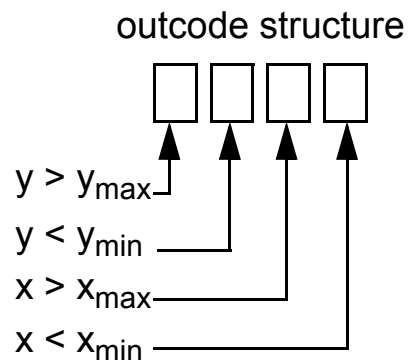
- operates with pixels
- brute force
- is used for arbitrary regions (masks)



Cohen-Sutherland Algorithm for Clipping Line Segments

The Cohen-Sutherland algorithm uses the limits of the view volume to divide the world up into regions and assigning codes called *outcodes* to the regions. Line segments can be classified by the outcodes of their end points.

- assign *outcodes* to line midpoints
- test outcodes => accept, reject, split
- repeat until all line segments accepted or rejected

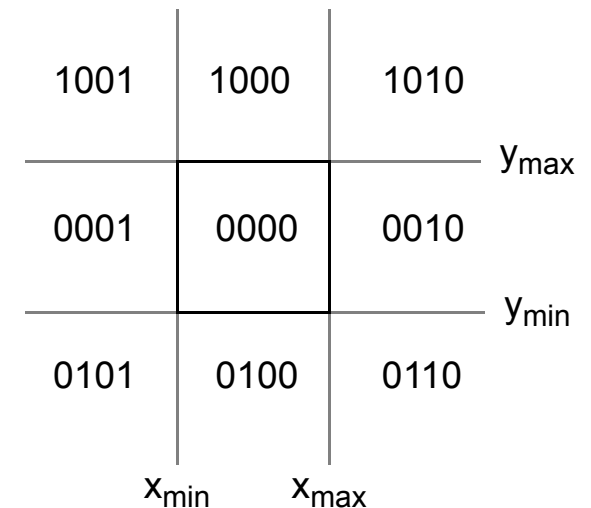


Cohen-Sutherland (cont)

The algorithm works by testing the outcodes at each end of a line segment, together with the logical and of the outcodes. From this 4 condition can be found:

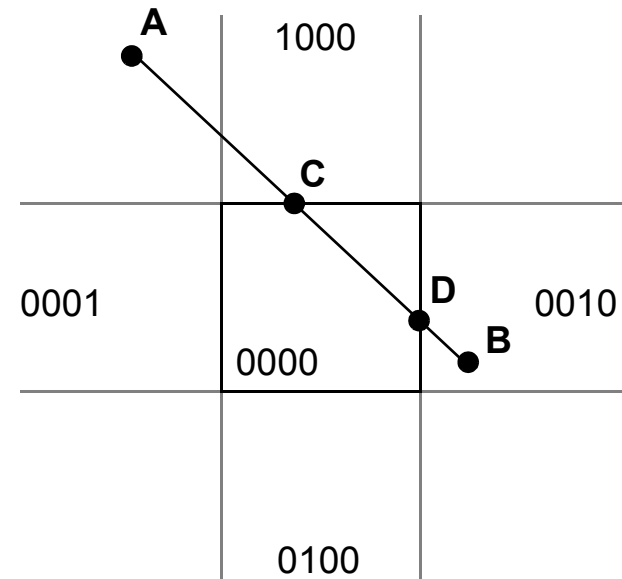
- The line is within the view volume: render the line
- One end of the line is in the view volume: clip the line and render it.
- The line may enter the view volume: clip to the clipping planes and retest
- The line can not be enter the view volume: delete it.

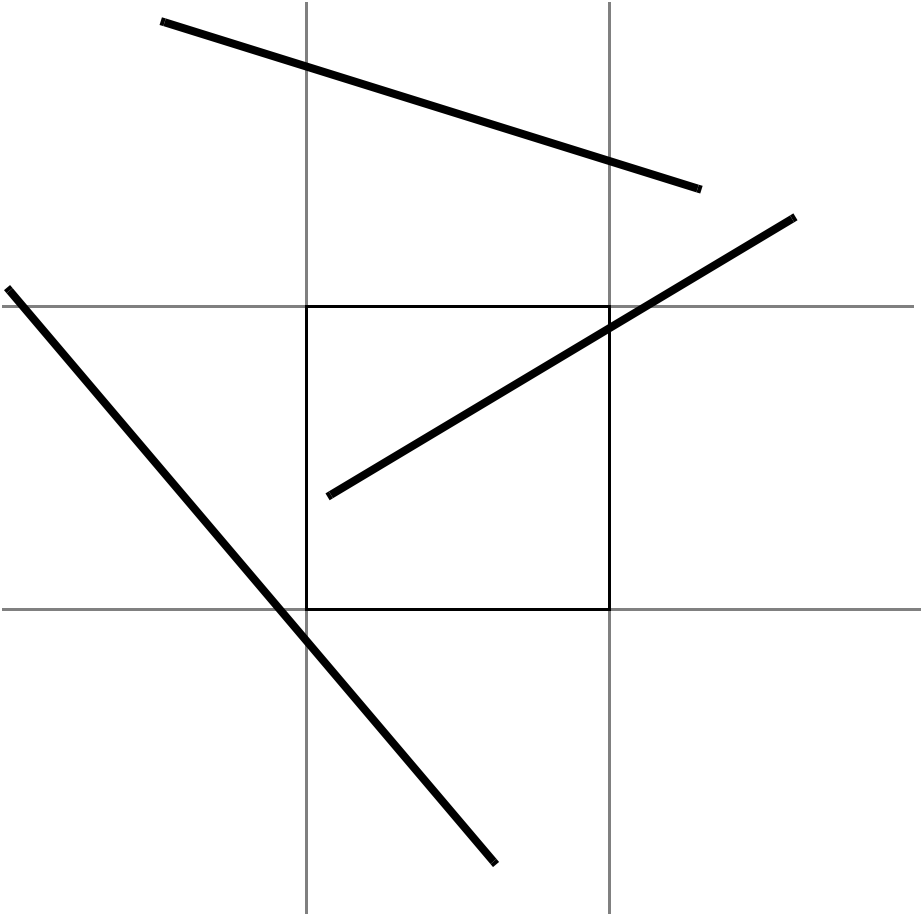
O_1	O_2	$O_1 \& O_2$	Condition
0	0	Don't care	Both ends in view volume: render
0	>0	Don't care	One end in and one out of view volume: clip line to view volume
>0	0	Don't care	
>0	>0	0	Part of line may be in view: more tests
>0	>0	>0	Line can not enter view volume



Cohen-Sutherland Example

1. $o_A = 1001$; $o_B = 0010$
2. $(o_A \text{ AND } o_B) = 0$ so split using first non-zero bit giving AC & CB, throw away “external” segment AC
3. $o_C = 0000$; $o_B = 0010$
4. $o_C = 0$; $o_B \neq 0$ so split using the first non-zero bit giving CD & DB, throw away the “external” segment DB
5. $o_C = 0000$; $o_D = 0000$ so trivially accept





Pros and Cons

- The Cohen-Sutherland algorithm is best when:
 - the clipping region is large compared to the world so there are lots of trivial accepts
 - the clipping region is small compared to the world so there are lots of trivial rejects

- Find the intersection using slope-intercept form:

$$y = y_1 + m(x - x_1)$$

$$x = x_1 + \frac{y - y_1}{m}$$

where x or y is set to edge coordinate

- The calculations involve floating point divisions both to find x and the value of m
- Looking at the example we can see that redundant clipping may occur

Parametric Clipping Algorithms

This algorithm works by deriving the line equation in parametric form. Then the properties of the intersections with the sides of the view volume are qualitatively analysed. Only if absolutely necessary are the intersections carried out.

For a line defined by 2 points (x_1, y_1) and (x_2, y_2) the parametric form is:

$$x = x_1 + u\Delta x$$

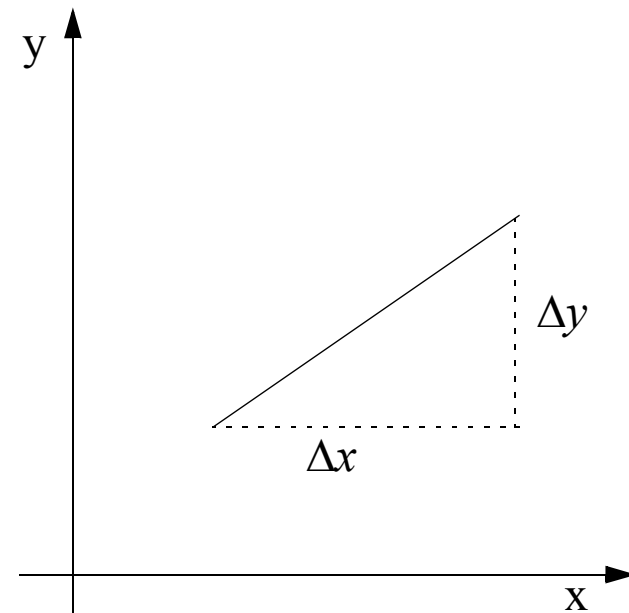
$$y = y_1 + u\Delta y$$

where

$$\Delta x = x_2 - x_1$$

$$\Delta y = y_2 - y_1$$

clearly $u = 0$ at (x_1, y_1) and $u = 1$ at (x_2, y_2) .



Parametric Clipping Algorithms (cont)

Also if $\Delta y = 0$ or $\Delta x = 0$ the line is parallel to an axis and can be handled by comparisons of the x or y values.

If both increments are non zero the line will cut all 4 boundaries of the view volume. Consider the intersection of the line with the lower boundary view volume $x = x_{\min}$. The intersection can be found by solving

$$x_{\min} = x_1 + u_1 \Delta x$$

for u_1 , but this involves a floating point division, which we wish to avoid.

The trick is to find qualitative information on the value of u_1 without having to find the actual value.

Parametric Clipping Algorithms (cont)

To do this write the equation as

$$u_1 \Delta x = x_{\min} - x_1$$

if Δx and $x_{\min} - x_1$ have opposite signs then $u_1 < 0$: so the intersection is not on the segment.

and if $\Delta x > x_{\min} - x_1$ then $u_1 > 1$: so the intersection is not on the segment.

The same process can be applied to the intersections with y_{\min} , x_{\max} and y_{\max} respectively u_2, u_3, u_4 .

We can also rank the u values by without division

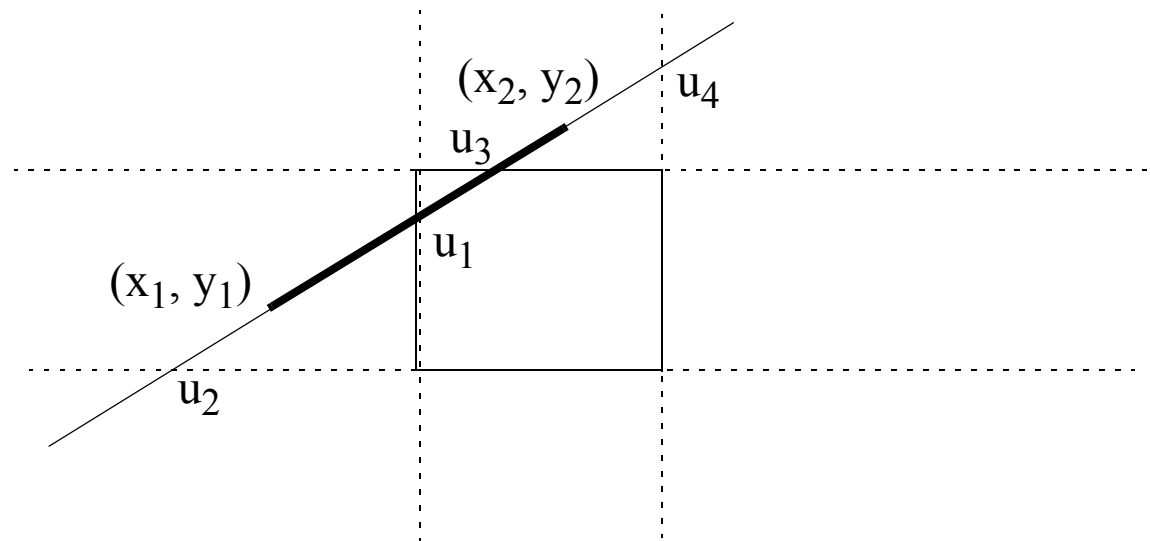
$$(u_1 < u_4) \Rightarrow \left(\frac{x_{\min} - x_1}{\Delta x} < \frac{y_{\max} - y_1}{\Delta y} \right) \Rightarrow ((x_{\min} - x_1)\Delta y < (y_{\max} - y_1)\Delta x)$$

Parametric Clipping Algorithms (cont)

We have the 4 intersections parameter values in order and know if each is less than 0 greater than 1 or in the line segment.

Consider the case where the order is $u_2 < u_1 < u_3 < u_4$ with $u_2 < 0$ and $u_4 > 1$

The only form of line that satisfies these conditions is of the form



Parametric Clipping Algorithms (final)

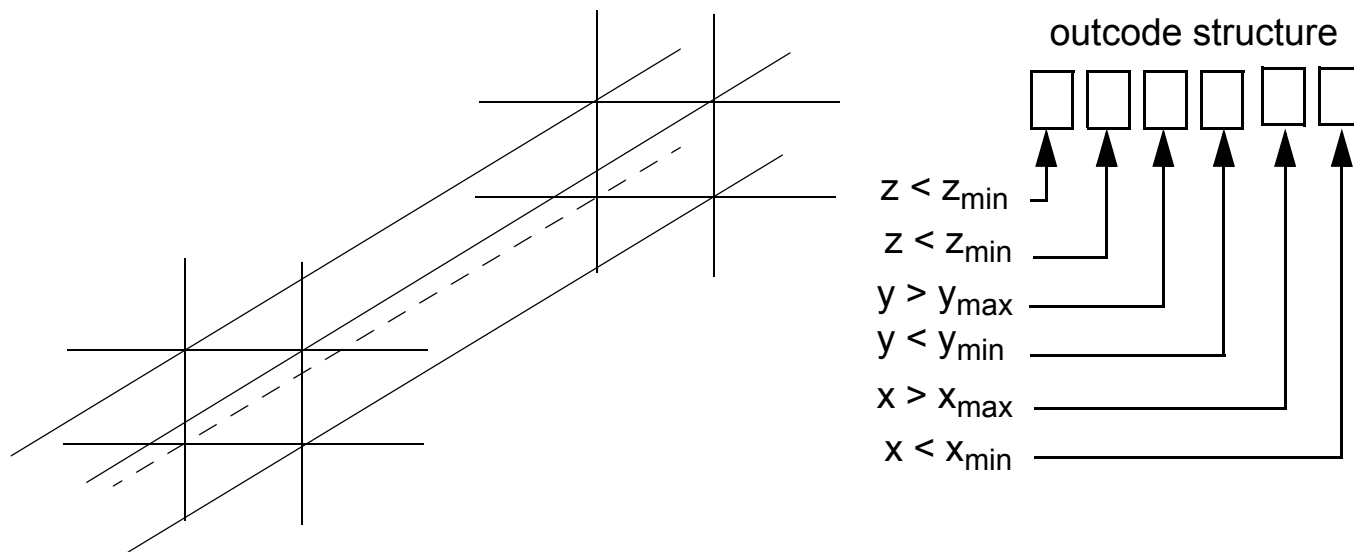
All the possible permutations of u_1 to u_4 can be similarly classified. As a result clipping can be applied only when need.

The main parametric clipping algorithm is the *Ling-Barsky* algorithm that uses a slightly modified version of this approach.

The *Ling-Barsky* algorithm is extendable to 3D.

Clipping in 3D

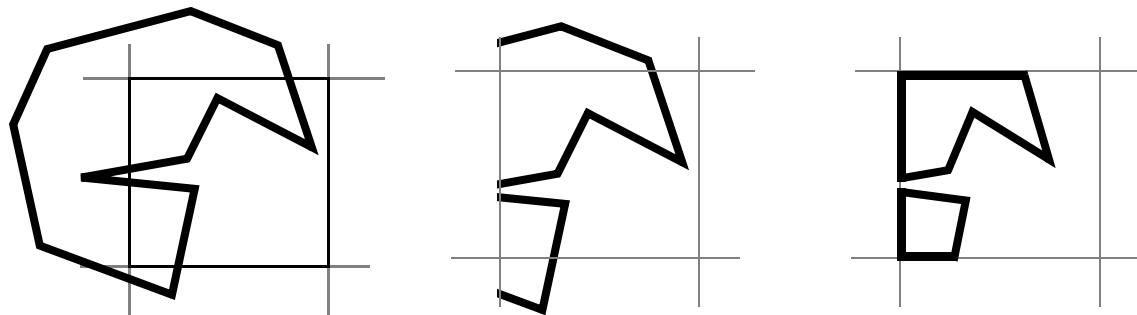
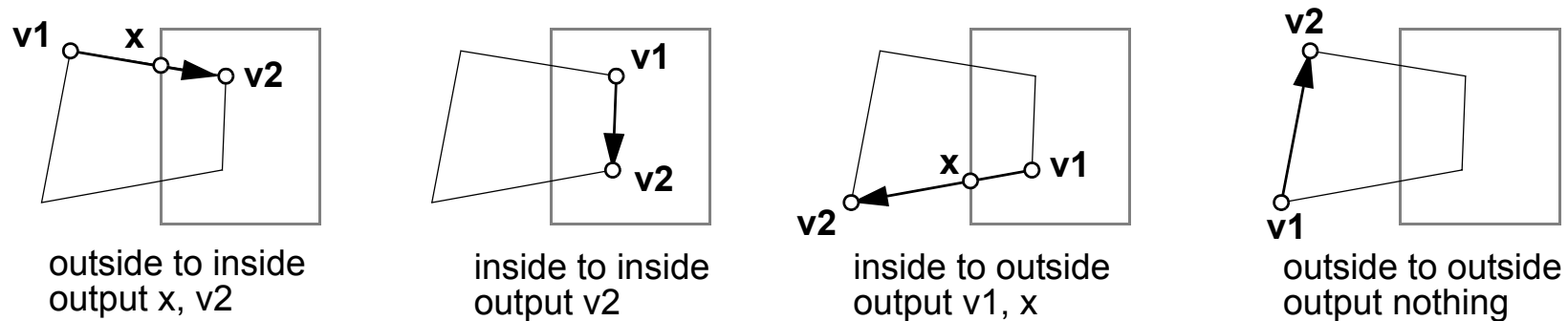
- In 3D we have to clip lines against *planes*.
- The Cohen-Sutherland algorithm can be applied in a modified form.
- The *Ling-Barsky* algorithm can be applied but not other parametric clipping algorithms



Clipping Polygons

clipping polygon \neq line clipping polygon edges (new points must be added to close up polygons)

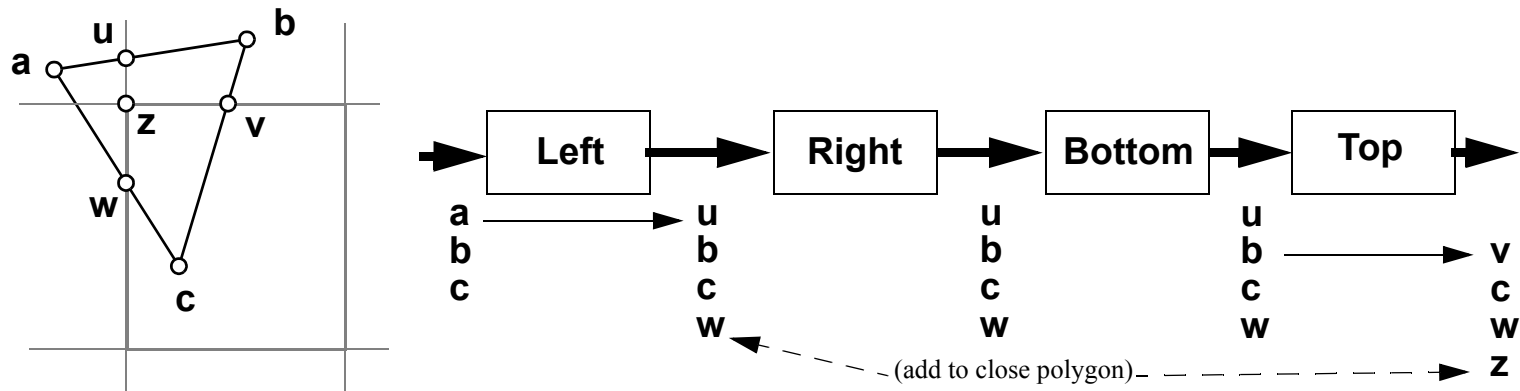
The Sutherland-Hodgman algorithm clips all the lines forming the polygon against each edge of the view volume in turn.



Implementing Sutherland-Hodgman

Conceptually:

- maintain input and out vertex lists
- traverse input list, storing vertices into output list
- clip against each edge in turn, using output from previous clip as input
- pipeline of clippers



- optimise by passing each vertex to *next* clipper immediately (i.e. pipelining)

Circles and Curves

- non-linear equations for intersection tests
- utilise bounding information, for example:

Check bounding box for trivial accept/reject

If necessary, repeat for quadrants.

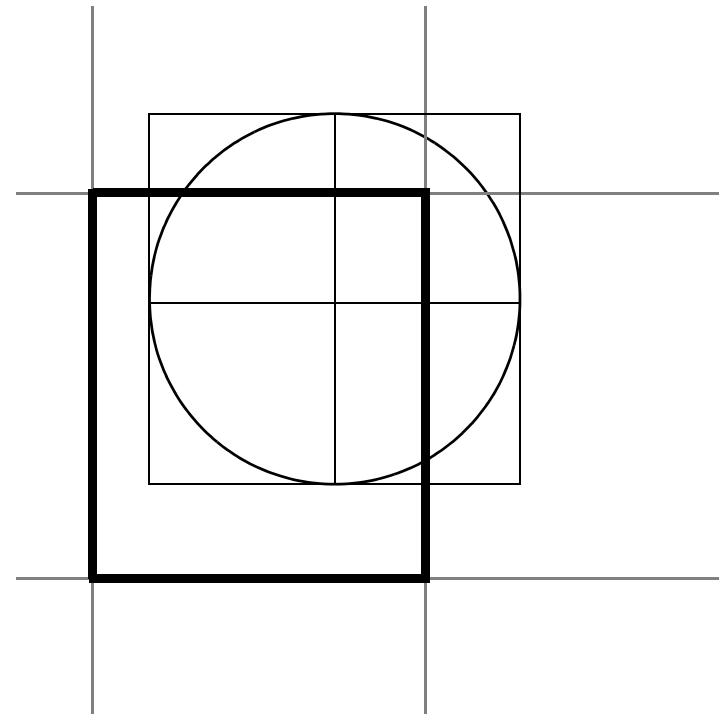
If necessary, repeat for octants

Either:

test intersection analytically

or:

scissor on a pixel-by-pixel basis



Clipping Text

representing text:

- as a collection of lines and curves (outline)
GLUT_STROKE
clipped as a polygon
- as a bitmap
GLUT_BITMAP
scissored in colour buffer

Since characters usually come in a string, we can accelerate clipping by using individual character bounds to identify where to clip the string.

