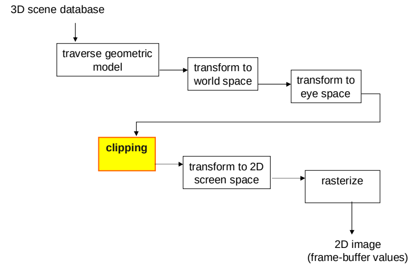
Clipping

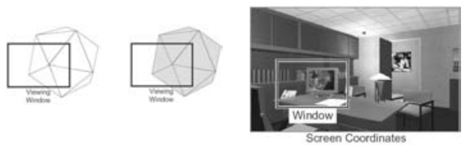
- Cohen-Sutherland line clipping algorithm
- Cyrus-Beck parametric line clipping algorithm
- Sutherland-Hodgman polygon clipping algorithm

Overview of Graphics Pipeline



Clipping

- Avoid drawing parts of primitives outside window
  - window defines parts of the scene to be viewed
  - must draw geometric primitives only inside window (points, lines, polygons, …)

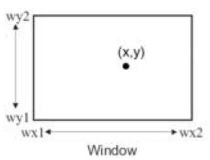


Clipping – How to?

"Oh, lovely – just hundredth time you've managed to cut everyone's head off"

Point Clipping

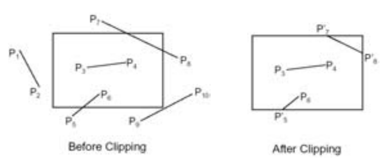
Is point (x,y) inside the clip window?




```
inside =
    (x >= wx1) &&
    (x <= wx2) &&
    (y >= wy1) &&
    (y <= wy2);
```


Line Segment Clipping

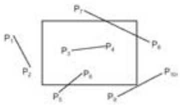
Find the part of a line inside the clip window



Line Segment Clipping (2)

Endpoints:

- if both endpoints are within the clipping rectangle, the line is completely inside (trivially accepted)
- if one end point is inside and the other is outside then we must compute the point of intersection
- if both endpoints are outside, then the line may or may not be inside



Line Segment Clipping (3)

Good approach will find eliminate trivial acceptances or rejections quickly and devote time to those lines which actually intersect the clipping rectangle.

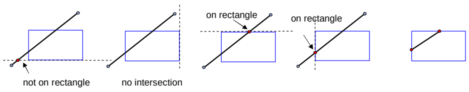
Consider the following methods:

- Analytical: solve simultaneous equations
- Cohen-Sutherland: Region out codes
- Cyrus-beck (Liang-Barsky): parametric line equation

Simultaneous Equations

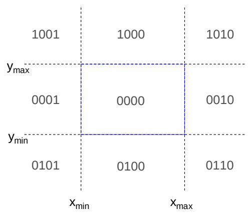
Brute force:

- Intersect the line with each of the 4 clip edges (x_{min}, x_{max}, y_{mix}, y_{max})
- test these intersection points to see if they occur on the edges of the clipping rectangle



Cohen-Sutherland

- we can divide space into 9 regions
- 4-bit outcode determined by comparisons



B_1	B_2	B_3	B_4
top	bottom	right	left

B_1: $y > y_{max}$

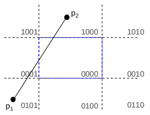
B_2: $y < y_{min}$

B_3: $x > x_{max}$

B_4: $x < x_{min}$

Cohen-Sutherland (2)

algorithm:

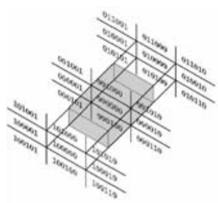


- compute outcode for endpoints
- O1 = O2 = 0000: accept
- O1 & O2 ≠ 0: reject
- pick one of endpoints that is not inside (O ≠ 0000)
- if (O & top) then clip with top edge
- if (O & bottom) then clip with bottom edge
- if (O & right) then clip with right edge
- if (O & left) then clip with left edge
- repeat

top=	1000
bottom=	0100
right =	0010
left=	0001

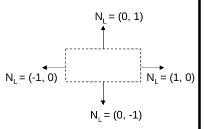
Cohen-Sutherland in 3D

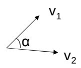
- Use 6 bits for outcodes:
  - B_5: $z > z_{max}$ (front)
  - B_6: $z < z_{min}$ (back)
- Other calculations as before



Cyrus-Beck Algorithm

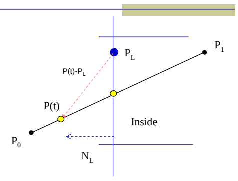
- We wish to optimize line/line intersection
  - Start with parametric equation of line:
    - $P(t) = P_0 + (P_1 - P_0)\, t$
  - And a point and normal for each edge
    - P_L, N_L
  - Dot product: $v_1.v_2 = |v_1|.|v_2|.\cos\alpha$
    - $\alpha < 90$: $v_1.v_2 > 0$
    - $\alpha = 90$: $v_1.v_2 = 0$
    - $\alpha > 90$: $v_1.v_2 < 0$





Cyrus-Beck Algorithm (2)

- Find t such that
  $N_L \,.\, [P(t) - P_L] = 0$

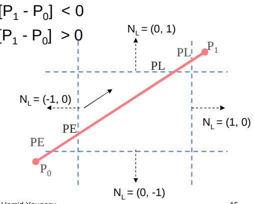


- Substitute line equation for P(t)
- Solve for t

$$t = \frac{N_i.(P_0 - P_L)}{-N_i.D}$$

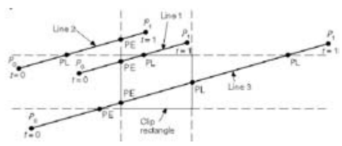
Cyrus-Beck Algorithm (3)

- Compute t for line intersection with all four edges
- Discard all $(t < 0)$ and $(t > 1)$
- Classify remaining intersections as
  - Potentially Entering (PE): $N_L \,.\, [P_1 - P_0] < 0$
  - Potentially Leaving (PL): $N_L \,.\, [P_1 - P_0] > 0$



Cyrus-Beck Algorithm (4)

- Compute PE with largest t (max_{PE})
- Compute PL with smallest t (min_{PL})
  - if ($max_{PE} < min_{PL}$) Clip to these two points
  - else reject the line



Cyrus-Beck Algorithm (5)

Because of horizontal and vertical clip lines:

Many computations reduce

Normals: (-1, 0), (1, 0), (0, -1), (0, 1)

Picking constant points on edges (P_L) solution for t:

$$t = \frac{N_i.(P_0 - P_L)}{-N_i.D}$$

- $t_{left} = -(x_0 - x_{min}) / (x_1 - x_0)$
- $t_{right} = (x_0 - x_{max}) / -(x_1 - x_0)$
- $t_{bottom} = -(y_0 - y_{min}) / (y_1 - y_0)$
- $t_{top} = (y_0 - y_{max}) / -(y_1 - y_0)$

Comparison

Cohen-Sutherland
- Repeated clipping is expensive
- Best used when trivial acceptance and rejection is possible for most lines

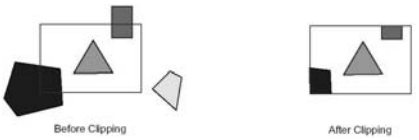
Cyrus-Beck
- Computation of t-intersections is cheap
- Computation of (x,y) clip points is only done once
- Algorithm doesn't consider trivial accepts/rejects
- Best when many lines must be clipped

Liang-Barsky: Optimized Cyrus-Beck

Nicholl et al.: Fastest, but doesn't do 3D

Polygon Clipping

Find the part of a polygon inside the clip window

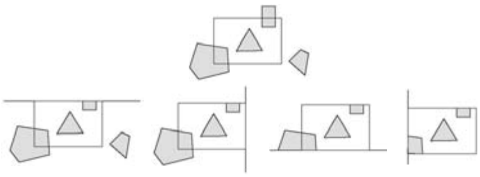


Polygon Clipping (2)

Polygon Clipping | Polygon Clipping is complex even when the polygon is convex | Polygon Clipping is nasty when the polygons are concave

Sutherland-Hodgman

Clip the polygon to each window boundary (edge) one at a time



After doing all edges, the polygon(s) is fully clipped

Sutherland-Hodgman (2)

Input/output for algorithm:

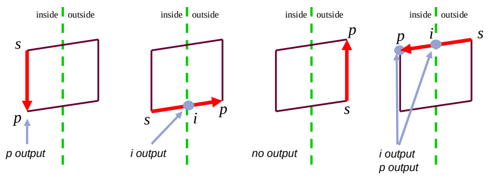
- Input: list of polygon vertices in order
- Output: list of clipped polygon vertices consisting of old vertices (maybe) and new vertices (maybe)

Basic routine:

- Go around polygon one vertex at a time
- Do inside test for each point in sequence,
  - Insert new points when cross window boundary,
  - Remove points outside window boundary

Sutherland-Hodgman (3)

A polygon edge from previous point (s) to current point (p) takes one of the four case:
(boundary can be a line or a plane)



Sutherland-Hodgman (4)

- Four cases:
  - *s* inside plane and *p* inside plane
    - Add *p* to output
    - Note: *s* has already been added
  - *s* inside plane and *p* outside plane
    - Find intersection point *i*
    - Add *i* to output
  - *s* outside plane and *p* outside plane
    - Add nothing
  - *s* outside plane and *p* inside plane
    - Find intersection point *i*
    - Add *i* to output, followed by *p*

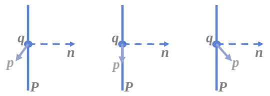
3D Clipping: Point-to-Plane test

- A very general test to determine if a point *p* is "inside" a plane ***P***, defined by point *q* and normal *n*:

  $(p - q) \bullet n < 0$: *p* inside ***P***

  $(p - q) \bullet n = 0$: *p* on ***P***

  $(p - q) \bullet n > 0$: *p* outside ***P***



Remember:

$\mathbf{p} \bullet n = (p_x n_x + p_y n_y + p_z n_z) = |p|\,|n| \cos(\theta)$

θ = angle between p and n

3D Clipping: Line-Plane Intersections

- Edge intersects plane ***P*** where $E(t)$ is on ***P***
  - ***q*** is a point on ***P***
  - ***n*** is normal to ***P***

$$(\mathbf{L}(t) - \mathbf{q}) \bullet \mathbf{n} = 0$$

$$t = [(\mathbf{q} - \mathbf{L_0}) \bullet \mathbf{n}] \,/\, [(\mathbf{L_1} - \mathbf{L_0}) \bullet \mathbf{n}]$$

  - The intersection point $\mathbf{i} = \mathbf{L}(t)$ for this value of t