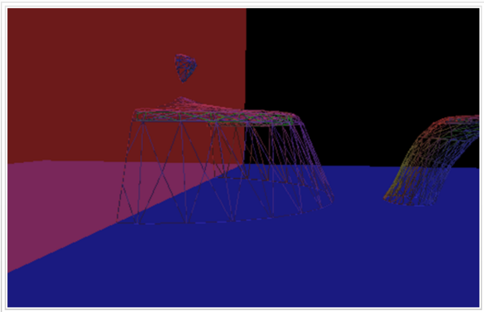
Computer Graphics

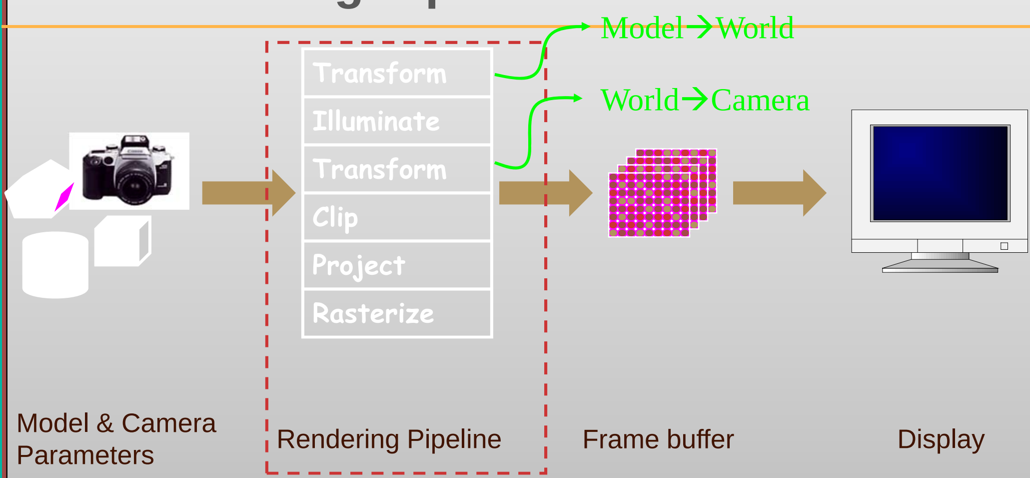


Ch 7
Clipping

Instructor: Dr. MAO Aihua

ahmao@scut.edu.cn

The Rendering Pipeline

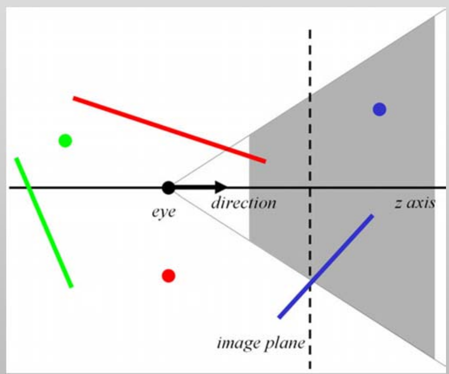


What is clipping?

Analytically calculating the portions of primitives within the view window

Why clip?

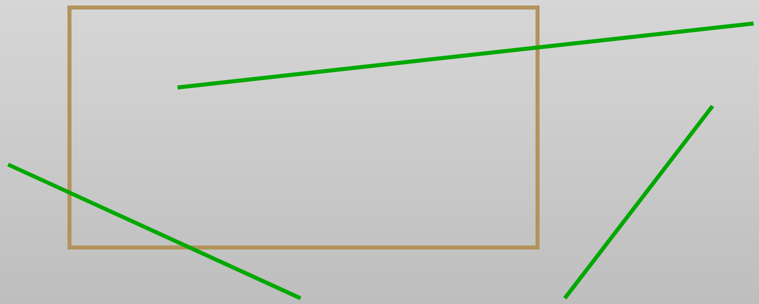
- Avoid degeneracies
  - *Don't draw stuff behind the eye*



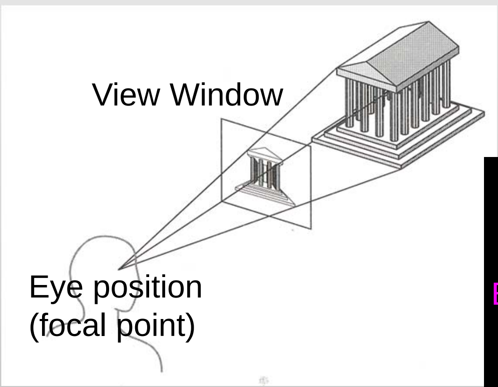
Why clip?

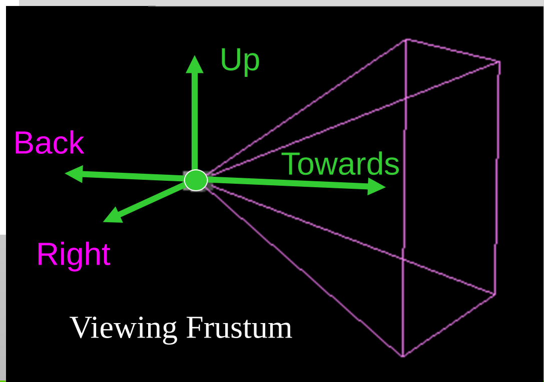
- Efficiency

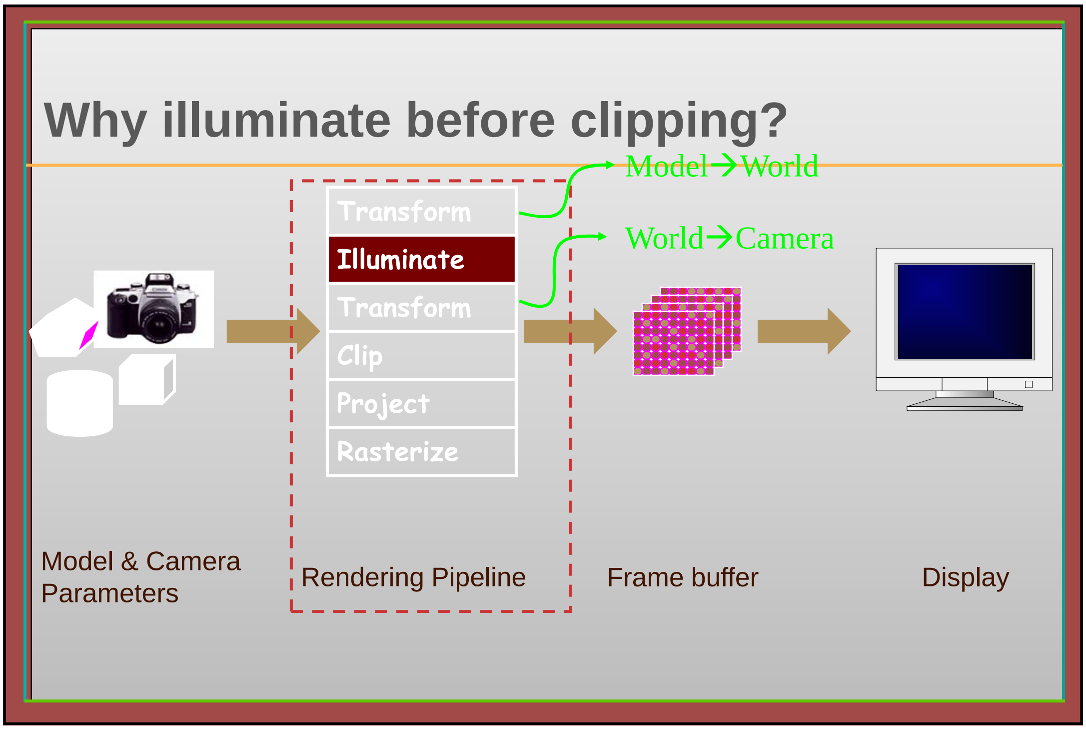
We don't want to waste time rendering objects that are outside the viewing window (or clipping window)

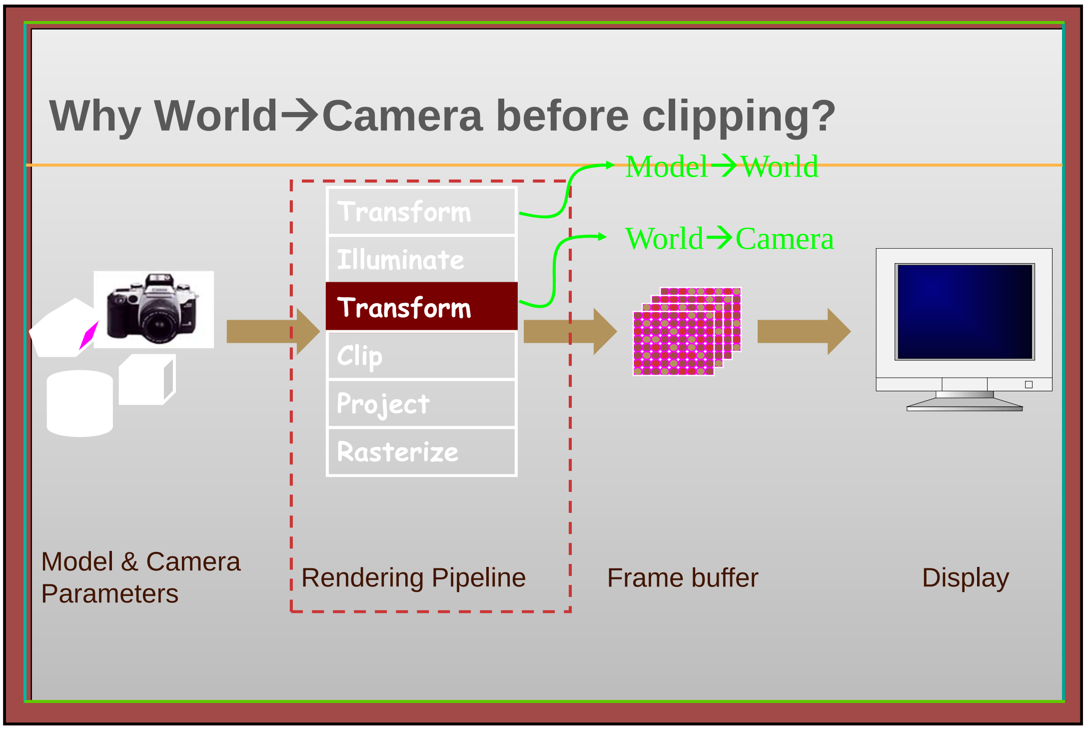


Clip to what?






Why illuminate before clipping?
Model→World
World→Camera
Transform
Illuminate
Transform
Clip
Project
Rasterize
Model & Camera Parameters
Rendering Pipeline
Frame buffer
Display


Why World→Camera before clipping?
Model→World
World→Camera
Transform
Illuminate
Transform
Clip
Project
Rasterize
Model & Camera Parameters
Rendering Pipeline
Frame buffer
Display

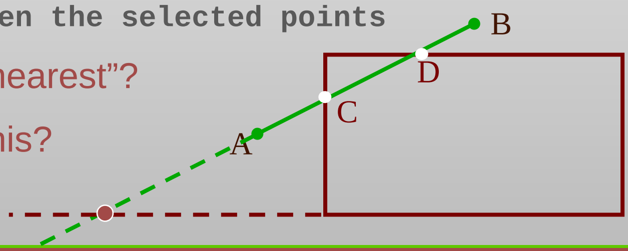
Clipping

The naïve approach to clipping lines:

```
for each line segment
    for each edge of view_window
        find intersection point
        pick “nearest” point
draw the points between the selected points
```

What do we mean by “nearest”?

How can we optimize this?

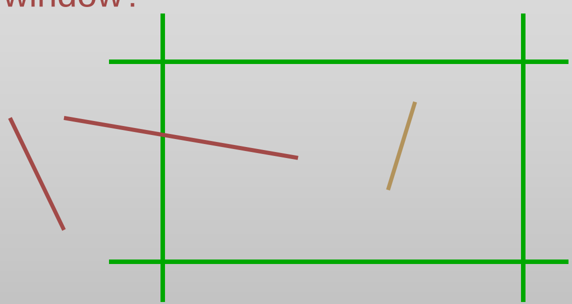


Trivial Accepts

Big optimization: trivial accept/rejects

How can we quickly determine whether a line segment is entirely inside the view window?

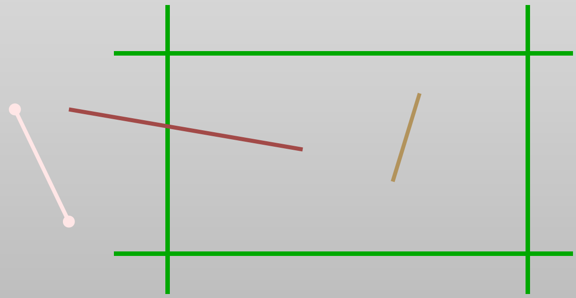
A: test both endpoints.



Trivial Rejects

How can we know a line is outside view window?

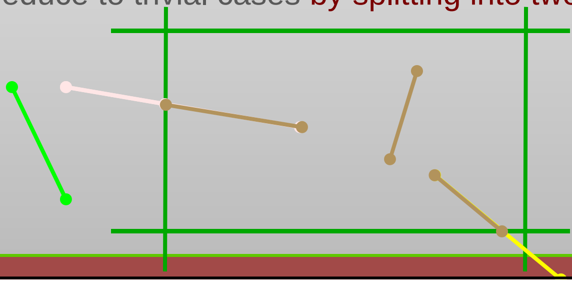
A: if both endpoints on the wrong side of same edge, can trivially reject line



Clipping Lines To Viewport

Combining trivial accepts/rejects

- Trivially accept lines with both endpoints inside all edges of the view window
- Trivially reject lines with both endpoints outside the same edge of the view window
- Otherwise, reduce to trivial cases by splitting into two segments



Cohen-Sutherland Line Clipping

Ivan Edward *Sutherland* (born 1938 in Hastings, Nebraska)

Carnegie-Mellon Univ, Caltech, MIT

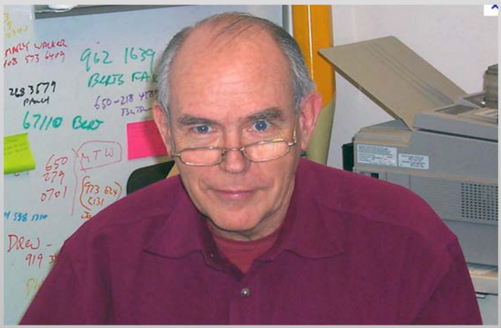
MIT: Sketchpad, 1963, MIT

Asso. Prof., 1966, Harvard

Prof., 1968, Utah

Dean, 1976, Caltech

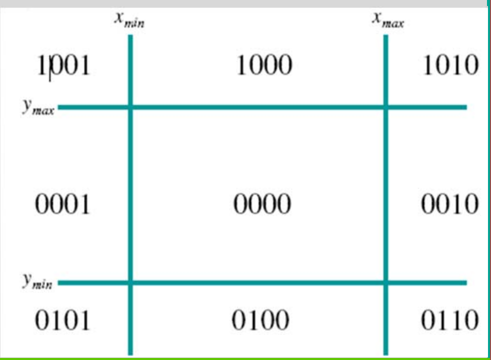
Turing Award, 1988



Cohen-Sutherland Line Clipping

- Divide view window into regions defined by window edges
- Assign each region a 4-bit *outcode*:

```
Ymax; Xmin; Ymin; Xmax
int ComputeOutCode(float x, float y)
{   int code =0;
    if y > Ymax then code = 8
    else if y < Ymin code = 4
    if x > Xmax code = code + 2;
    else if x < Xmin = code + 1;
     return code;
}
```



Cohen-Sutherland Line Clipping

For each line segment

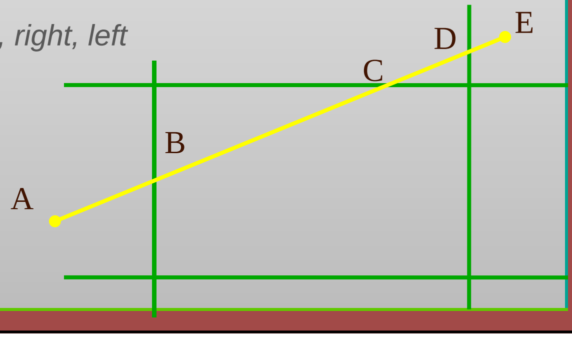
- Assign an outcode to each endpoint according to the area
- If both outcodes = 0 (in the area 0000), trivial accept
  - *Same as performing* `if (bitwise OR = 0)`
- Else
  - *bitwise AND outcodes together*
  - *if result ≠ 0, trivial reject*
  - *else split line segment*

Cohen-Sutherland Line Clipping

If line cannot be trivially accepted, subdivide so that one or both segments can be discarded

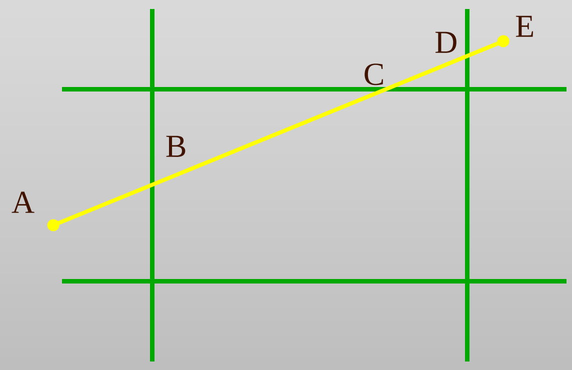
Pick an edge of view window that the line crosses

- Check against edges in same order each time
  - *For example: top, bottom, right, left*



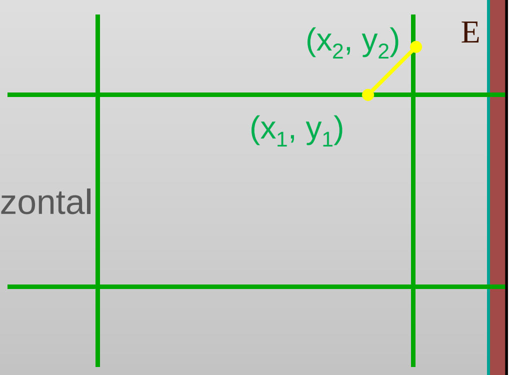
Cohen-Sutherland Line Clipping

Intersect line with edge (how?)



View Window Intersection Code

- (x_1, y_1), (x_2, y_2) intersect with vertical edge at x_{right}
  - $y_{intersect} = y_1 + m(x_{right} - x1)$
    - where $m=(y_2-y_1)/(x_2-x_1)$
- (x_1, y_1), (x_2, y_2) intersect with horizontal edge at y_{top}
  - $x_{intersect} = x_1 + (y_{top} - y1)/m$
    - where $m=(y_2-y_1)/(x_2-x_1)$



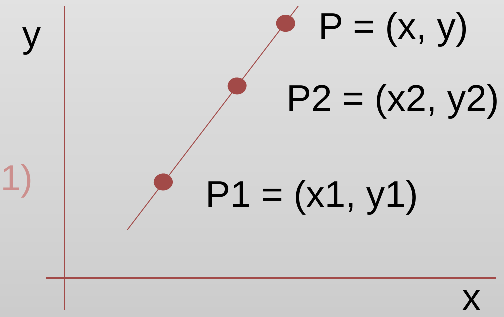
Review: 3D Line – Slope Intercept

Slope =m

= rise / run

Slope = (y - y1) / (x - x1)

= (y2 - y1) / (x2 - x1)



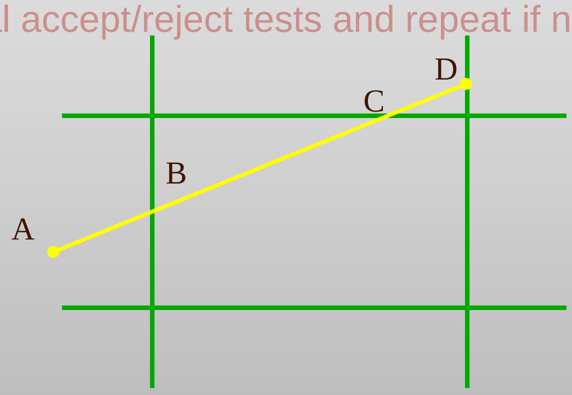
Solve for y:

y = [(y2 - y1)/(x2 - x1)]x + [-(y2-y1)/(x2 - x1)]x1 + y1

or: y = mx + b

Cohen-Sutherland Line Clipping

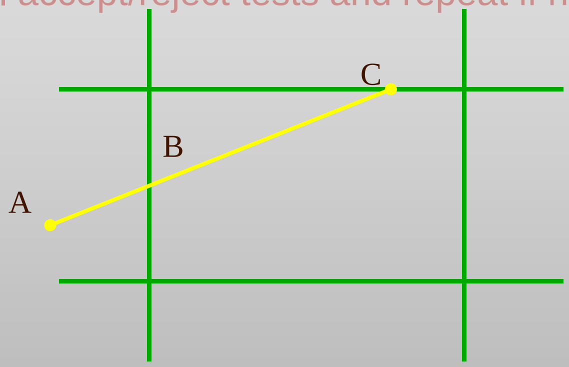
- Discard portion on wrong side of edge and assign outcode to new vertex
- Apply trivial accept/reject tests and repeat if necessary



Cohen-Sutherland Line Clipping

Discard portion on wrong side of edge and assign outcode to new vertex

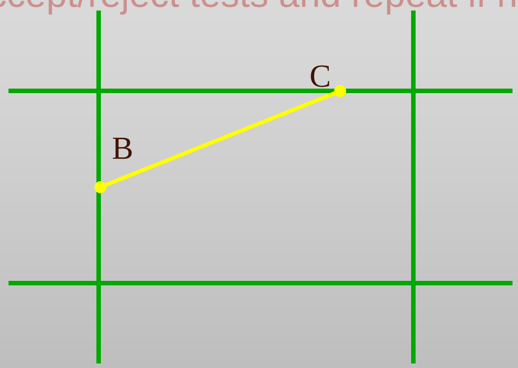
Apply trivial accept/reject tests and repeat if necessary



Cohen-Sutherland Line Clipping

Discard portion on wrong side of edge and assign outcode to new vertex

Apply trivial accept/reject tests and repeat if necessary



Cohen-Sutherland Review

- Use outcodes to quickly eliminate/include lines
  - *Is best algorithm when trivial accepts/rejects are common*
- Must compute viewing window clipping of the remaining lines
  - *Non-trivial clipping cost*
  - *Redundant clipping of some lines*

More efficient algorithms exist

Solving Simultaneous Equations

Equation of a line

- Slope-intercept (explicit equation): y = mx + b
- Implicit Equation: Ax + By + C = 0
- Parametric Equation: Line defined by two points, P_0 and P_1
  - $\mathbf{P}(t) = \mathbf{P}_0 + (\mathbf{P}_1 - \mathbf{P}_0)\, t$, *where* $\mathbf{P}$ *is a vector* $[x, y]^T$
  - $x(t) = x_0 + (x_1 - x_0)\, t$
  - $y(t) = y_0 + (y_1 - y_0)\, t$

Parametric Line Equation

Describes a finite line

Works with vertical lines (like the viewport edge)

- $0 <= t <= 1$
  - *Defines line between P_0 and P_1*
- $t < 0$
  - *Defines line before P_0*
- $t > 1$
  - *Defines line after P_1*

Parametric Lines and Clipping

Define each line in parametric form:

- $P_0(t) \ldots P_{n-1}(t)$

Define each edge of view window in parametric form:

- $P_L(t)$, $P_R(t)$, $P_T(t)$, $P_B(t)$

Perform Cohen-Sutherland intersection tests using appropriate view window edge and line

Line / Edge Clipping Equations

Faster line clippers use parametric equations

Line 0:

- $x^0 = x^0_0 + (x^0_1 - x^0_0)\, t^0$
- $y^0 = y^0_0 + (y^0_1 - y^0_0)\, t^0$

View Window Edge L:

- $x^L = x^L_0 + (x^L_1 - x^L_0)\, t^L$
- $y^L = y^L_0 + (y^L_1 - y^L_0)\, t^L$

$x^0_0 + (x^0_1 - x^0_0)\, t^0 = x^L_0 + (x^L_1 - x^L_0)\, t^L$

$y^0_0 + (y^0_1 - y^0_0)\, t^0 = y^L_0 + (y^L_1 - y^L_0)\, t^L$

- Solve for t^0 and/or t^L

Cyrus-Beck Algorithm

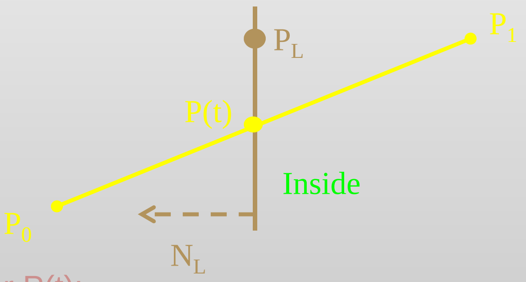
We wish to optimize line/line intersection

- 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

Cyrus-Beck Algorithm

Find t such that

$N_L \bullet [P(t) - P_L] = 0$



Substitute line equation for P(t):

- $N_L \bullet [P_0 + (P_1 - P_0)\, t - P_L] = 0$

Solve for t

- $t = N_L \bullet [P_L - P_0] \;/\; -N_L \bullet [P_1 - P_0]$

Cyrus-Beck Algorithm

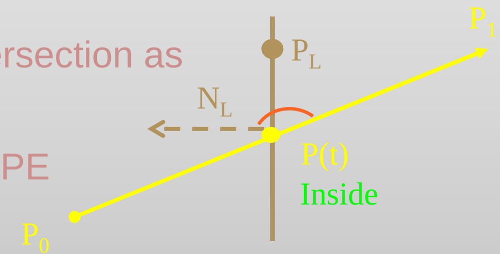
Compute t for line intersection with all four edges

Discard all (t < 0) and (t > 1)

Classify each remaining intersection as

- Potentially Entering (PE)

$N_L \bullet [P_1 - P_0] < 0$ implies PE



- Note that we computed this term when computing t so we can keep it around

Cyrus-Beck Algorithm

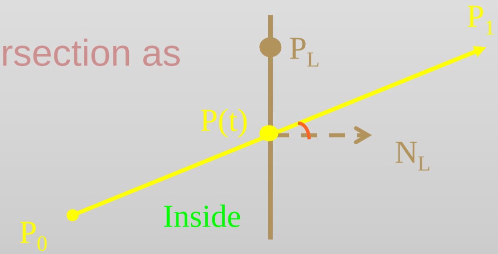
Compute t for line intersection with all four edges

Discard all (t < 0) and (t > 1)

Classify each remaining intersection as

- Potentially Leaving (PL)

$N_L \bullet [P_1 - P_0] > 0$ implies PL



- Note that we computed this term when computing t so we can keep it around

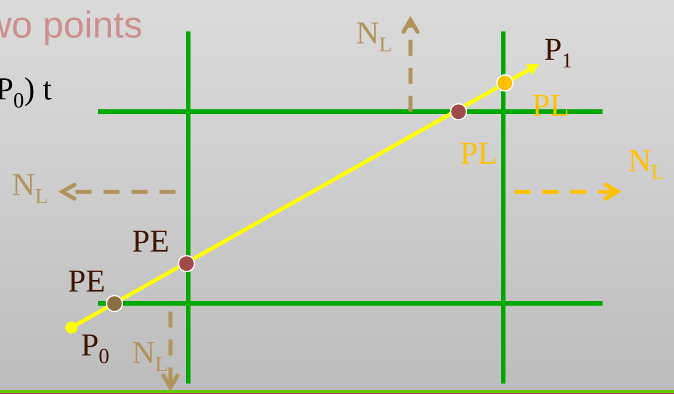
Cyrus-Beck Algorithm

Compute PE with largest t

Compute PL with smallest t

Clip to these two points

$$P(t) = P_0 + (P_1 - P_0)\,t$$



Cyrus-Beck Algorithm

Because of horizontal and vertical edge lines:

- Many computations reduce

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

Pick constant points on edges $(x_{left}, 0)$, $(x_{right}, 0)$, $(0, y_{bottom})$, $(0, y_{top})$

solution for t: $t = N_L\ [P_L - P_0]\ /\ -N_L\ [P_1 - P_0]$

Calculate t for the edges

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

Clipping Polygons

Clipping polygons is more complex than clipping the individual lines

- Input: polygon
- Output: original polygon, new polygon, or nothing

The biggest optimizer we had was trivial accept or reject…

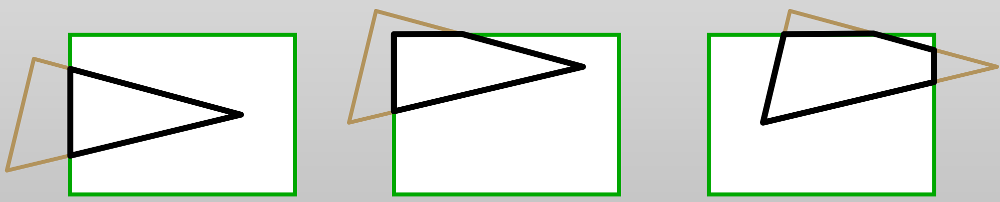
When can we trivially accept/reject a polygon as opposed to the line segments that make up the polygon?

Why Is Clipping Difficult?

What happens to a triangle during clipping?

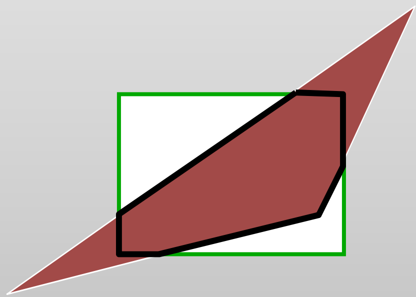
How many sides can a clipped triangle have?

Possible outcomes:



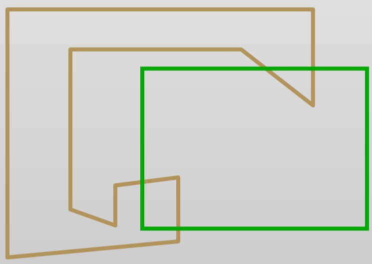
How many sides?

Seven…



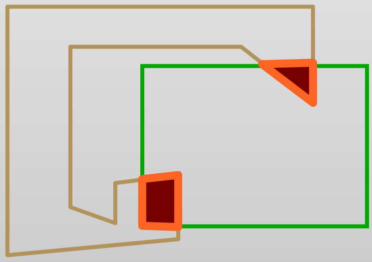
Why Is Clipping Hard?

A really tough case:



Why Is Clipping Hard?

A really tough case:

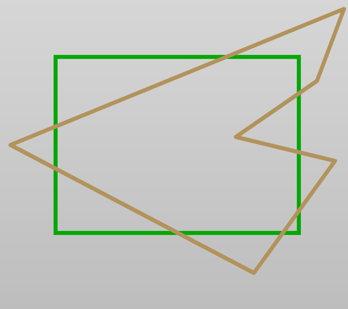


concave polygon multiple polygons

Sutherland-Hodgman Clipping

Basic idea:

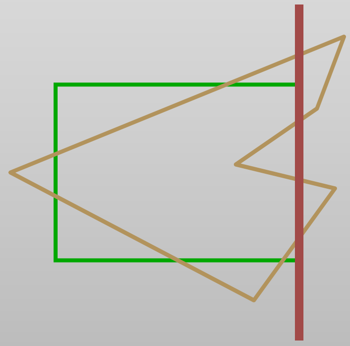
- Consider each edge of the view window individually
- Clip the polygon against the view window edge's equation



Sutherland-Hodgman Clipping

Basic idea:

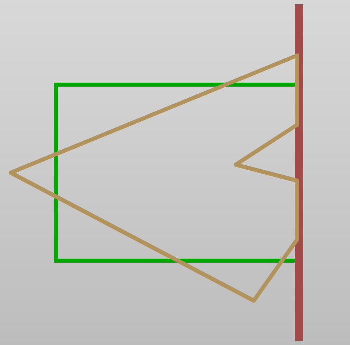
- Consider each edge of the view window individually
- Clip the polygon against the view window edge's equation



Sutherland-Hodgman Clipping

Basic idea:

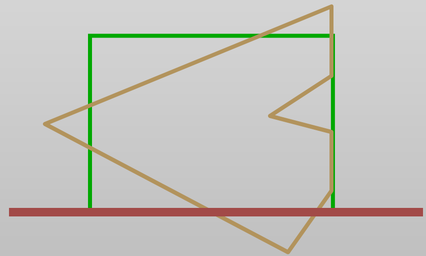
- Consider each edge of the view window individually
- Clip the polygon against the view window edge's equation



Sutherland-Hodgman Clipping

Basic idea:

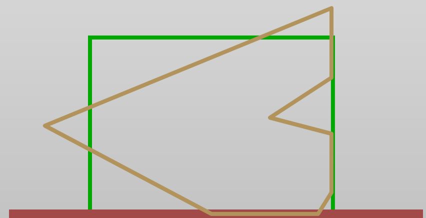
- Consider each edge of the view window individually
- Clip the polygon against the view window edge's equation



Sutherland-Hodgman Clipping

Basic idea:

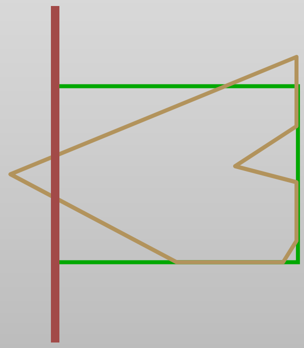
- Consider each edge of the view window individually
- Clip the polygon against the view window edge's equation



Sutherland-Hodgman Clipping

Basic idea:

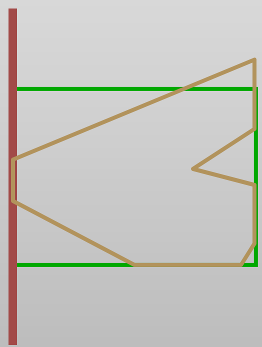
- Consider each edge of the view window individually
- Clip the polygon against the view window edge's equation



Sutherland-Hodgman Clipping

Basic idea:

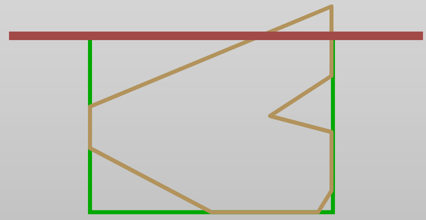
- Consider each edge of the view window individually
- Clip the polygon against the view window edge's equation



Sutherland-Hodgman Clipping

Basic idea:

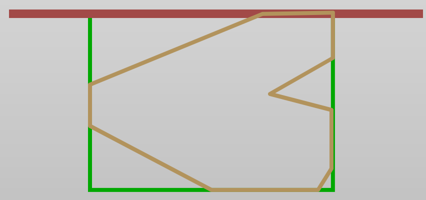
- Consider each edge of the view window individually
- Clip the polygon against the view window edge's equation



Sutherland-Hodgman Clipping

Basic idea:

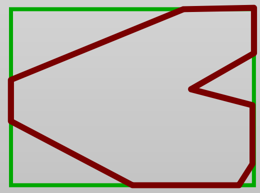
- Consider each edge of the view window individually
- Clip the polygon against the view window edge's equation



Sutherland-Hodgman Clipping

Basic idea:

- Consider each edge of the view window individually
- Clip the polygon against the view window edge's equation
- After doing all edges, the polygon is fully clipped



Sutherland-Hodgman Clipping

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)

Note: this is exactly what we expect from the clipping operation against each edge

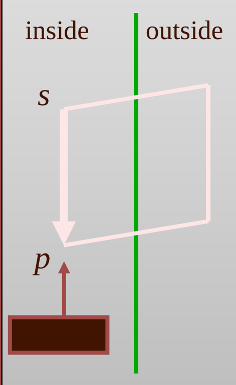
Sutherland-Hodgman Clipping

Sutherland-Hodgman basic routine:

- Go around polygon one vertex at a time
- Current vertex has position *p*
- Previous vertex had position *s*, and it has been added to the output if appropriate

Sutherland-Hodgman Clipping

Edge from s to p takes one of four cases:



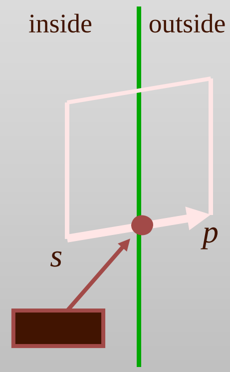
s inside plane and *p* inside plane

Add p to output

Note: s has already been added

Sutherland-Hodgman Clipping

Edge from s to p takes one of four cases:



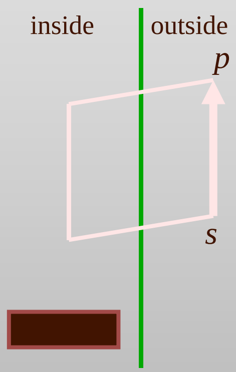
s inside plane and *p* outside plane

- Find intersection point i
- Add i to output

Sutherland-Hodgman Clipping

Edge from s to p takes one of four cases:

(Orange line can be a line or a plane)



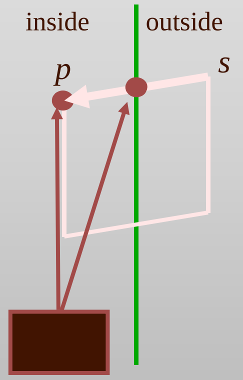
s outside plane and *p* outside plane

- Add nothing

Sutherland-Hodgman Clipping

Edge from s to p takes one of four cases:

(Orange line can be a line or a plane)



s outside plane and *p* inside plane

- Find intersection point i
- Add i to output, followed by p

Point-to-Plane test

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

q is a point on ***P***

n is normal to ***P***

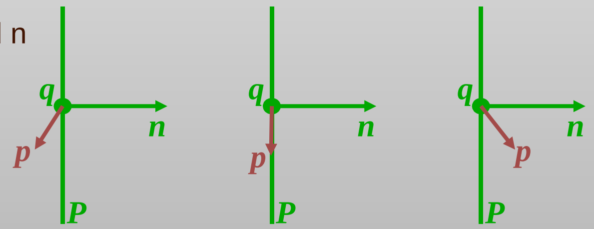
$(p - q) \bullet n < 0$: *p* inside ***P***

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

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

Remember: p • n = |p| |n| cos (θ)

θ = angle between p and n



Finding Line-Plane Intersections

Edge intersects with P where E(t) is on P

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

$$(L_0 + (L_1 - L_0)\, t - \boldsymbol{q}) \bullet \boldsymbol{n} = 0$$

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

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