Computer Graphics (CS 543)
Lecture 9: Clipping

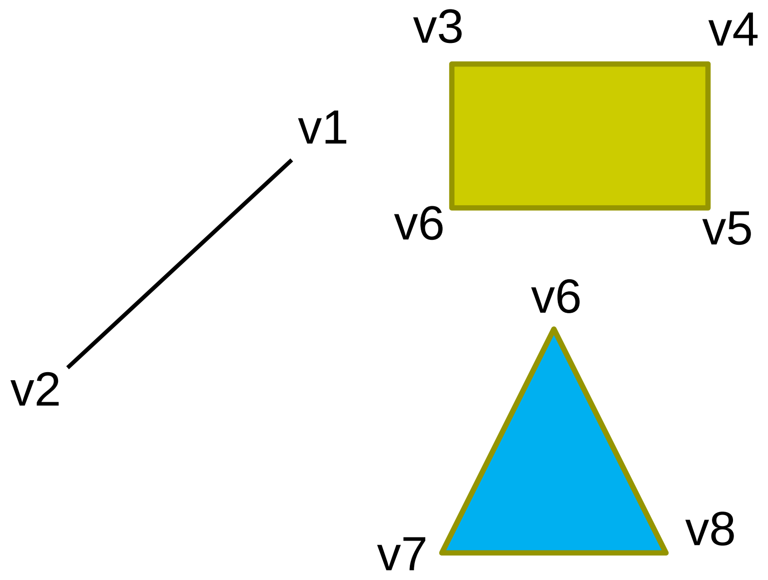
Prof Emmanuel Agu

Computer Science Dept.

Worcester Polytechnic Institute (WPI)

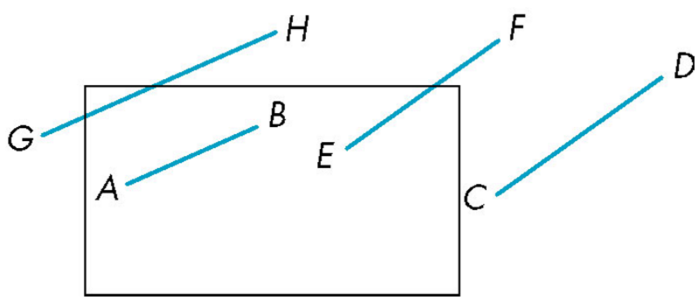
Primitive Assembly

- Transformations and projections are per-vertex operations
- **Primitive assembly:** At end of geometric pipeline, individual vertices are assembled back into primitives
- **E.g. v6, v7 and v8** grouped back into triangle

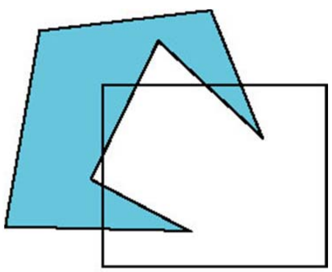


Clipping

- Subsequent operations necessary before display occur per-primitive
- **Clipping:** Remove primitives outside view frustum



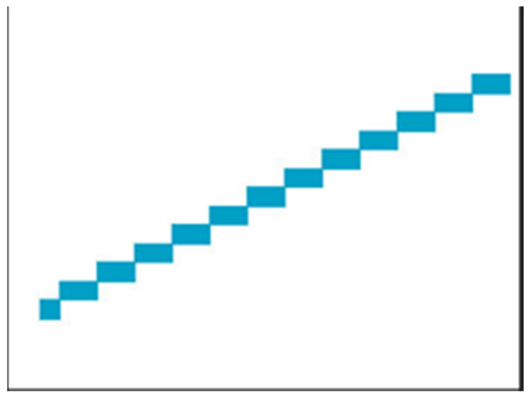
Clipping lines



Clipping polygons

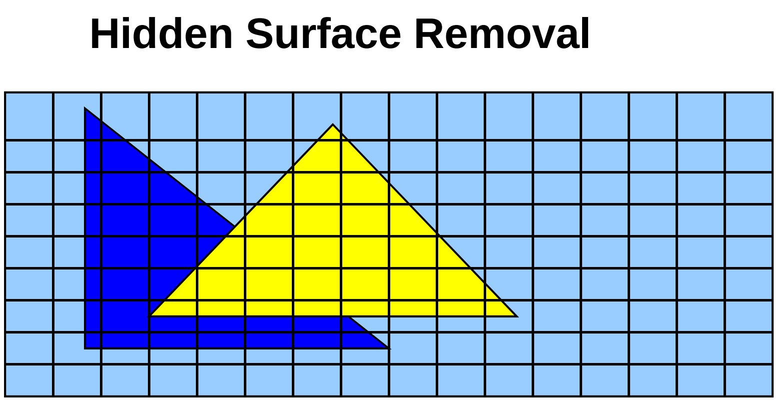
Rasterization

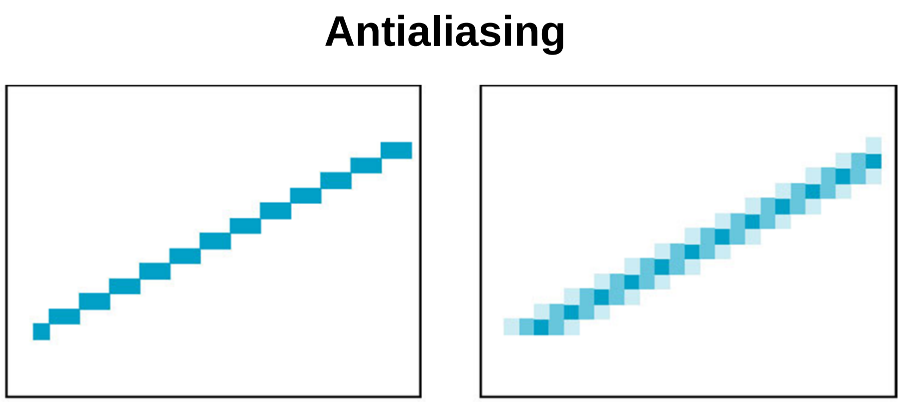
- Determine which pixels that primitives map to
  - Fragment generation
  - Rasterization or scan conversion



Fragment Processing

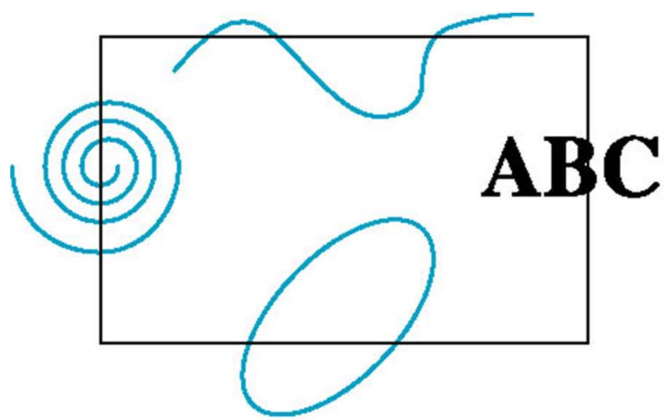
- Some tasks deferred until fragment processing





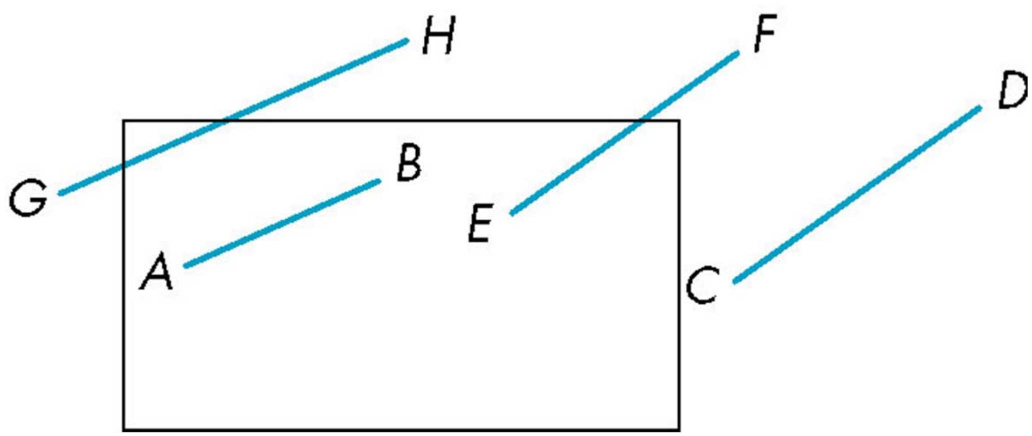
Clipping

- 2D against clipping window
- 3D against clipping volume
- Easy for line segments polygons
- Hard for curves and text
  - Convert to lines and polygons first



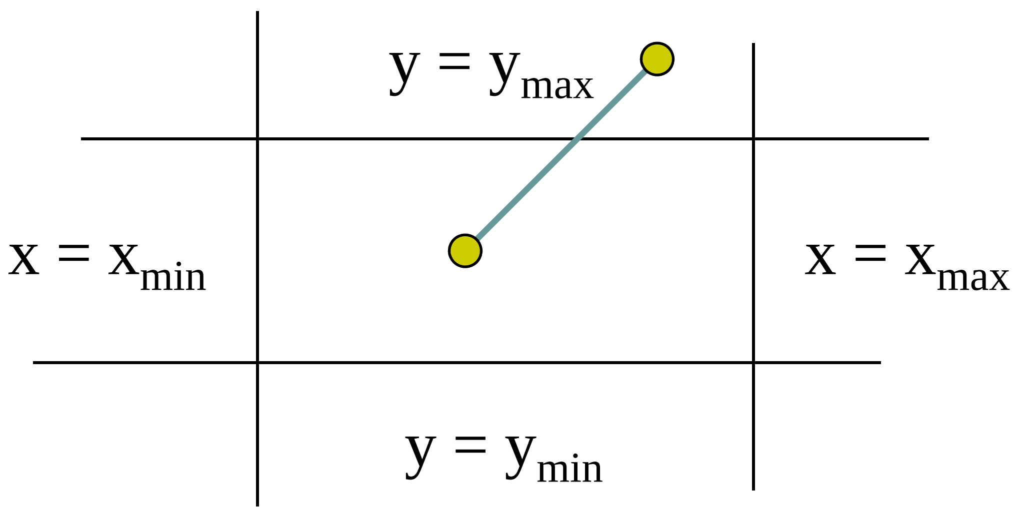
Clipping 2D Line Segments

- Brute force approach: compute intersections with all sides of clipping window
  - Inefficient: one division per intersection

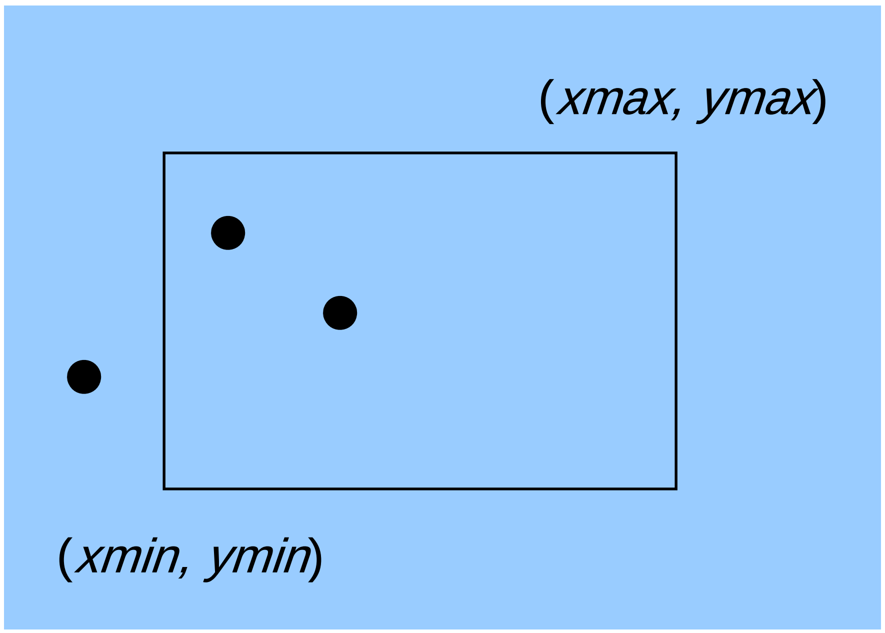


2D Clipping: Cohen-Sutherland Algorithm

- Idea: eliminate as many cases as possible without computing intersections
- Start with four lines that determine the sides of the clipping window



Clipping Points

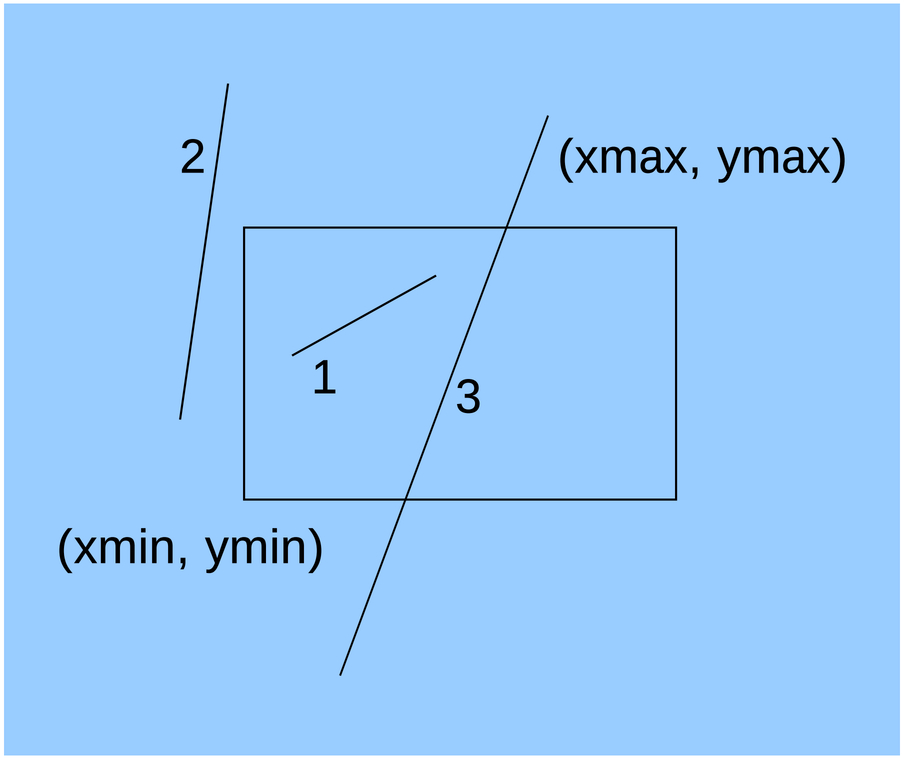


Determine whether a point (x,y) is inside or outside of the world window?

If (xmin <= x <= xmax)
and (ymin <= y <= ymax)

then the point (x,y) is inside
else the point is outside

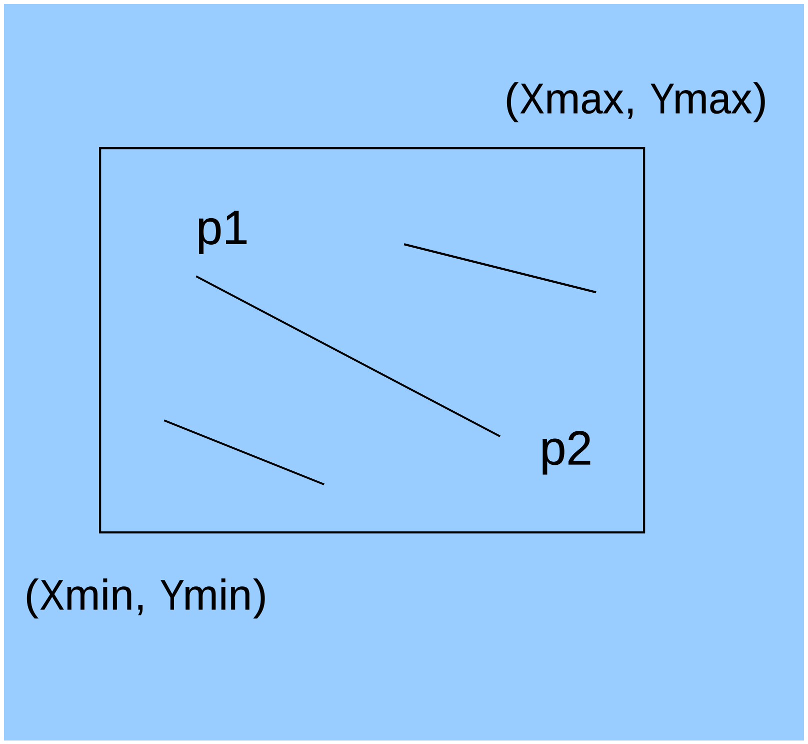
Clipping Lines



3 cases:

- **Case 1:** All of line in
- **Case 2:** All of line out
- **Case 3:** Part in, part out

Clipping Lines: Trivial Accept



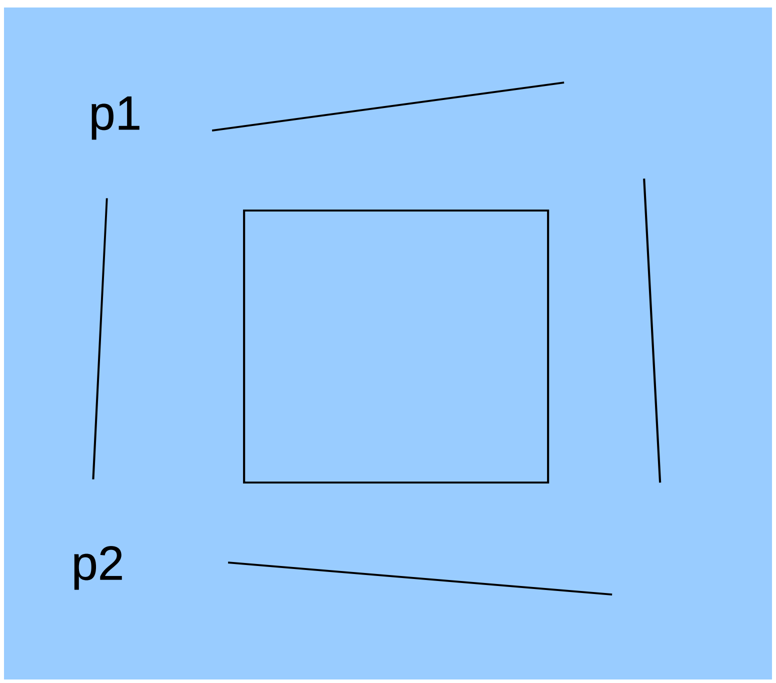
Case 1: All of line in
Test line endpoints:

Xmin <= P1.x, P2.x <= Xmax and
Ymin <= P1.y, P2.y <= Ymax

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



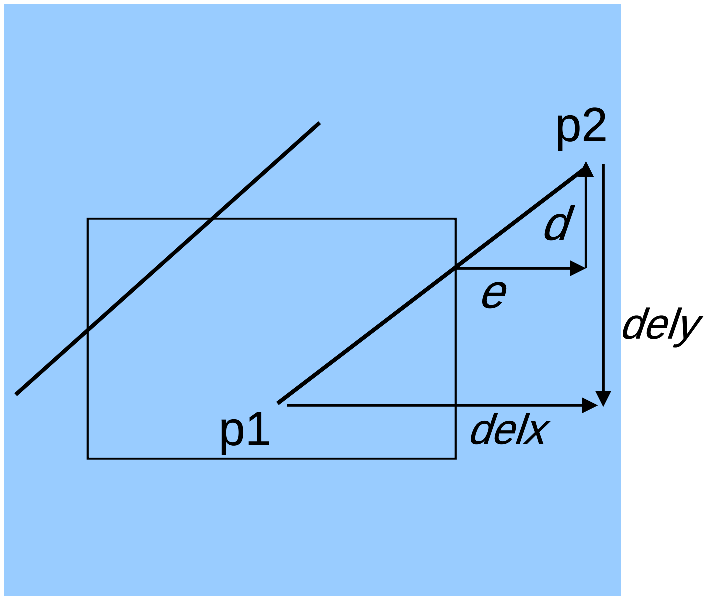
Case 2: All of line out
Test line endpoints:

- ***p1.x, p2.x <= Xmin*** **OR**
- ***p1.x, p2.x >= Xmax*** **OR**
- ***p1.y, p2.y <= ymin*** **OR**
- ***p1.y, p2.y >= ymax***

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



$$\frac{d}{dely} = \frac{e}{delx}$$

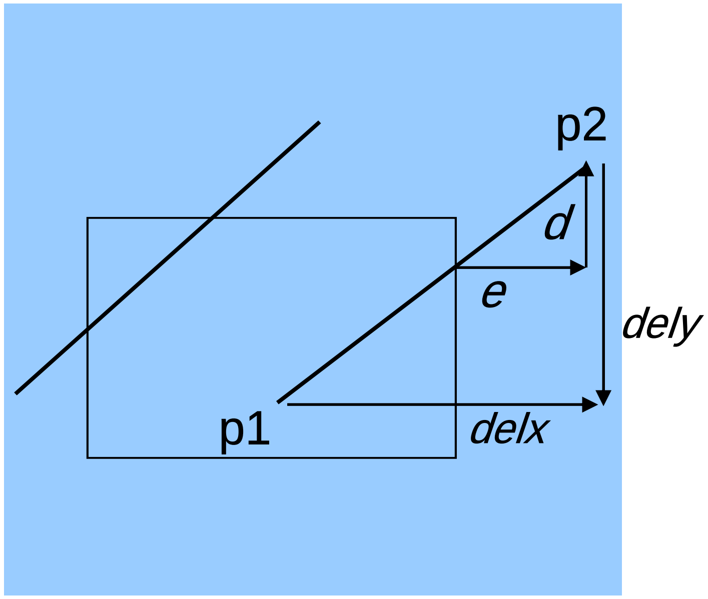
Case 3: Part in, part out

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

Clipping Lines: Calculation example



$$\frac{d}{dely} = \frac{e}{delx}$$

If chopping window has
(left, right, bottom, top) = (30, 220, 50, 240),
what happens when the following lines are chopped?

(a) p1 = (40,140), p2 = (100, 200)

(b) p1 = (20,10), p2 = (20, 200)

(c) p1 = (100,180), p2 = (200, 250)

Cohen-Sutherland pseudocode (Hill)

```
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)

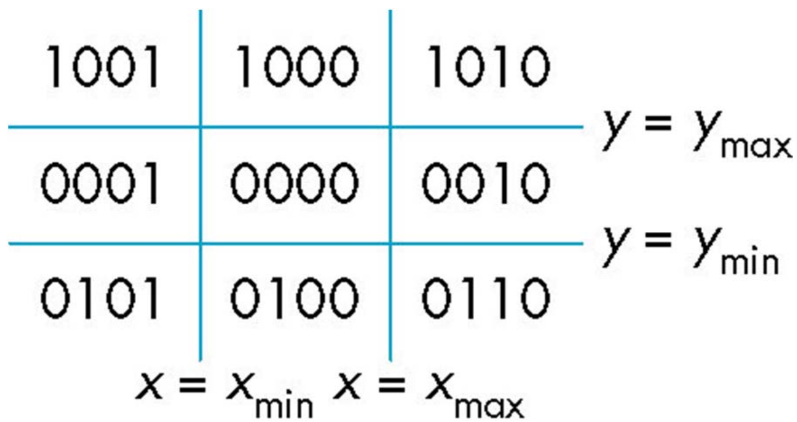
```
        else // p2 is outside
               // find surviving segment
        {
            if(p2 is to the left) chop against left edge
          else if(p2 is to right) chop against right edge
          else if(p2 is below) chop against the bottom edge
          else if(p2 is above) chop against the top edge
        }
    }while(1);
}
```

Using Outcodes to Speed Up Comparisons

- For each endpoint, define an outcode

$$b_0b_1b_2b_3$$

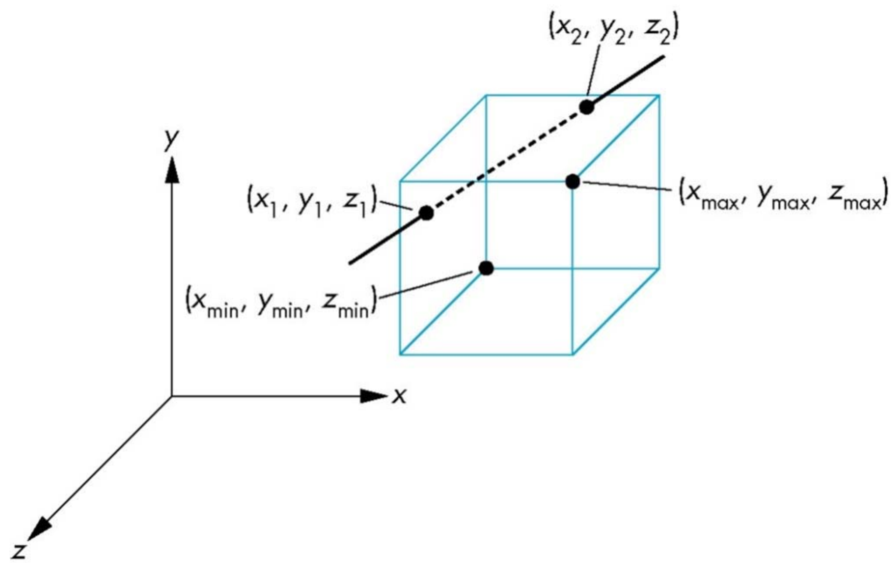
$b_0 = 1$ if $y > y_{max}$, 0 otherwise
$b_1 = 1$ if $y < y_{min}$, 0 otherwise
$b_2 = 1$ if $x > x_{max}$, 0 otherwise
$b_3 = 1$ if $x < x_{min}$, 0 otherwise

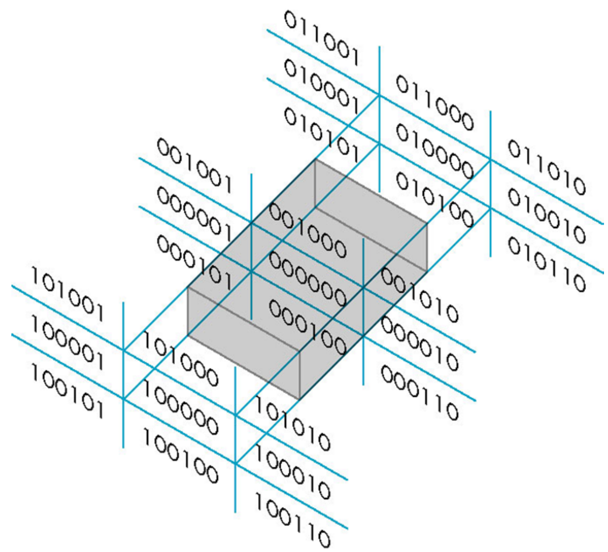


- Outcodes divide space into 9 regions

Cohen Sutherland in 3D

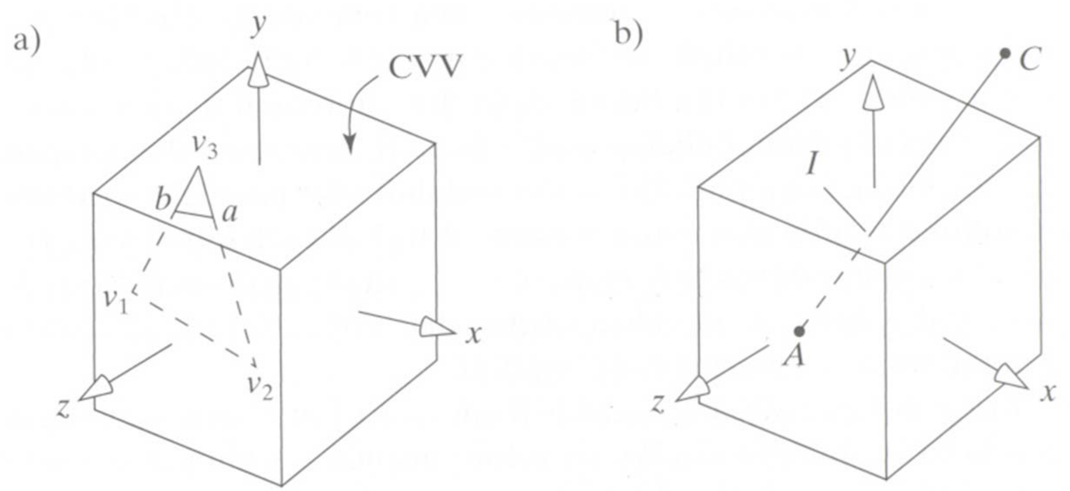
- Use 6-bit outcodes
- When needed, clip line segment against planes



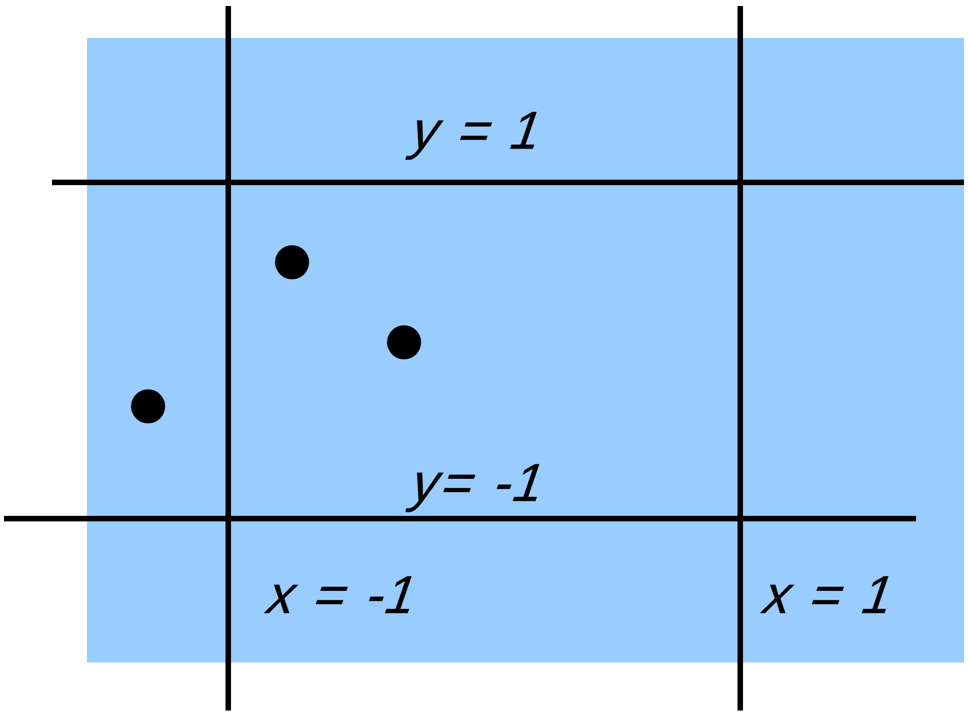


Liang-Barsky 3D Clipping

- Want to clip edge-by-edge of an object against CVV
- Now describe a version embellished by Jim Blinn
- Problem:
  - Two points, A = (Ax, Ay, Az, Aw) and C = (Cx, Cy, Cz, Cw), in homogeneous coordinates
  - If segment intersects with CVV, need to compute intersection point I-=(Ix,Iy,Iz,Iw)



Determining if point is inside CVV



- Determine whether a point (x,y,z) is inside or outside CVV?

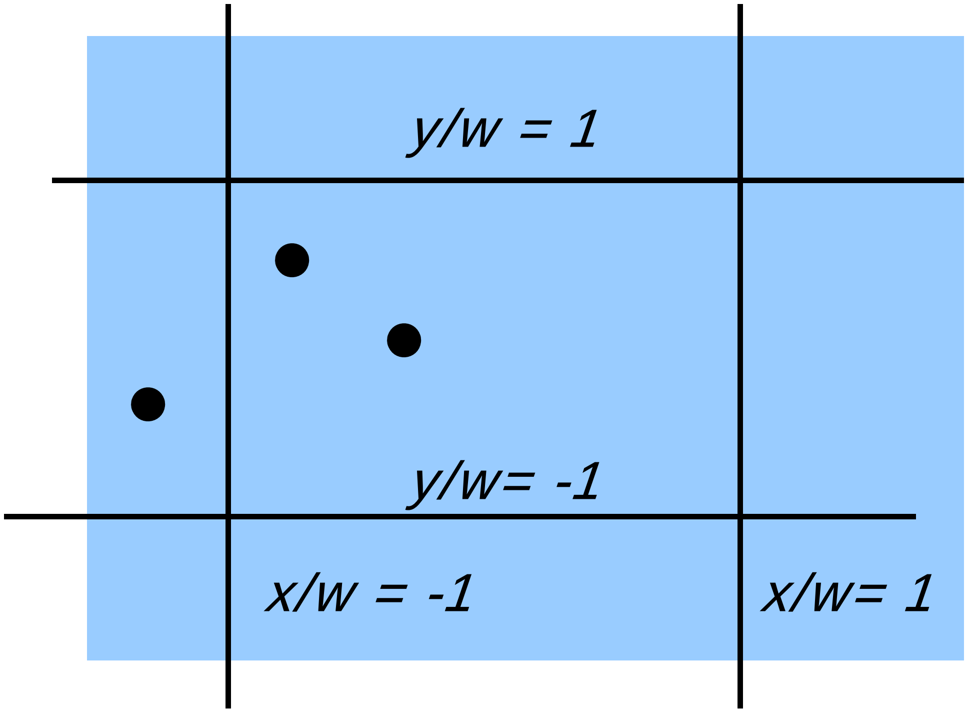
Point (x,y,z) is inside CVV

if $(-1 <= x <= 1)$
and $(-1 <= y <= 1)$
and $(-1 <= z <= 1)$

else the point is outside CVV

- CVV == 6 infinite planes (x=-1,1; y=-1,1; z=-1,1)

Determining if point is inside CVV



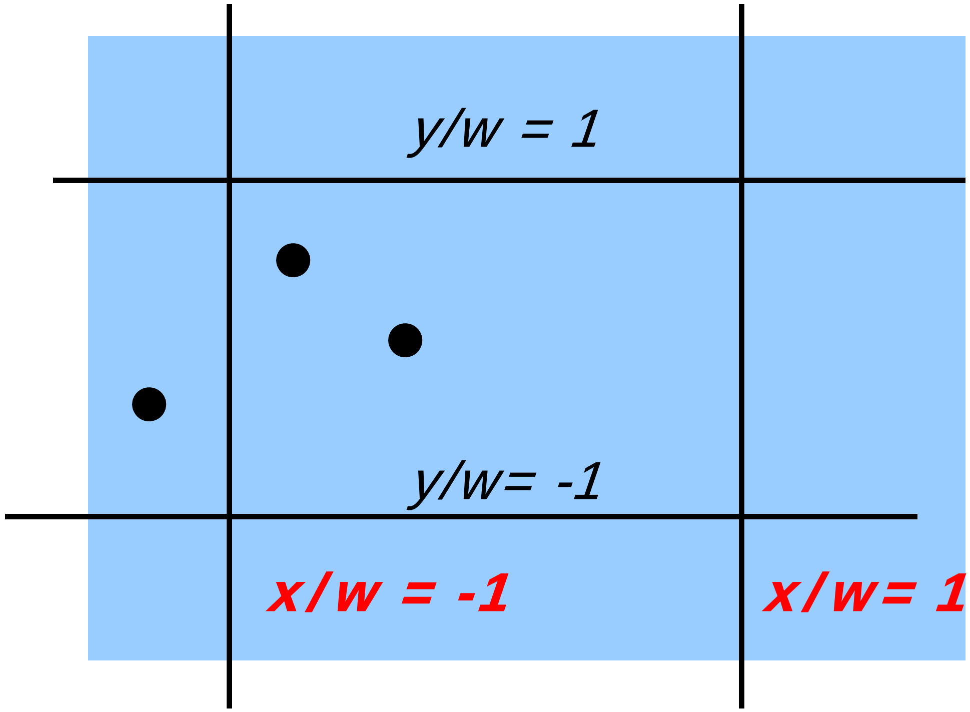
- What if point is in homogeneous coordinates?
- Point specified as (x,y,z,w)
- **Use scaled version of x,y,z!**

Point (x,y,z) is inside CVV

if (-1 <= x/w <= 1)
and (-1 <= y/w <= 1)
and (-1 <= z/w <= 1)

else the point is outside CVV

Determining if point is inside CVV



- Consider plane x = 1, point A = (Ax,Ay,Az,Aw) is inside if

$$Ax/Aw < 1$$
$$\Rightarrow Aw - Ax > 0$$
$$\text{or} \quad w - x > 0$$

- Point A = (Ax,Ay,Az,Aw) plane x = -1 if

$$Ax/Aw > -1$$
$$\Rightarrow Aw + Ax > 0$$
$$\text{or} \quad w + x > 0$$

Determining if point is inside CVV

- So, point is
  - inside (right of) plane x=-1 if w+x >0
  - inside (left of)   plane x=1 if w - x>0

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

  - 1 <= 0.5 <= 1

- If w = 10, (0.5, 0.2, 0.7) = (5, 2, 7, 10)
- Use scaled version, point is inside because – 1 <= 5/10 <= 1

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

- Notation (Aw +Ax) = w + x, boundary coordinates for 6 planes as:

Boundary coordinate (BC)	**Homogenous coordinate**	**Clip plane**	**Example (5,2,7,10)**
BC0	w+x	x=-1	15
BC1	w-x	x=1	5
BC2	w+y	y=-1	12
BC3	w-y	y=1	8
BC4	w+z	z=-1	17
BC5	w-z	z=1	3

- **Trivial accept:** 12 BCs (6 for pt. A, 6 for pt. C) are positive
- **Trivial reject:** Both endpoints outside of same plane

Edges as Parametric Equations

- Implicit form $F(x, y) = 0$
- Parametric forms:
  - points specified based on single parameter value
  - Typical parameter: time *t*

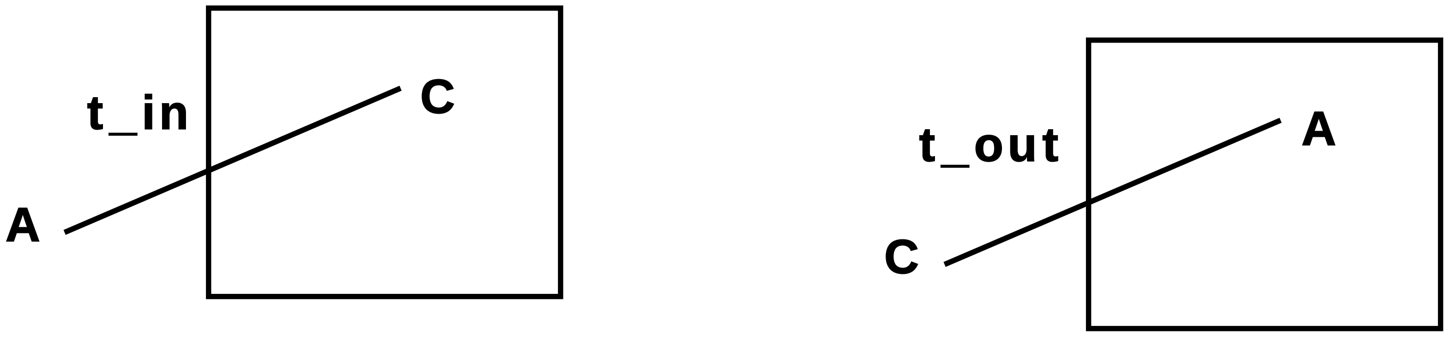
$$P(t) = P_0 + (P_1 - P_0) * t \qquad 0 \le t \le 1$$

- Some algorithms work in parametric form
  - Clipping: exclude line segment ranges
  - Animation: Interpolate between endpoints by varying t
- Represent each edge parametrically as A + (C – A)t
- Intepretation: a point is traveling such that:
  - at time t=0, point at A
  - at time t=1, point at C

Inside/outside?

- Test against 6 walls
- If BCs have opposite signs = edge hits plane at time t_hit
- Define: “entering” = as t increases, outside to inside

  i.e. if pt. A is outside, C is inside
- Define “leaving”: as t increases, inside to outside (A inside, C outside)



Calculating hit time (t_hit)

- How to calculate t_hit?
- Represent an edge t as:

$$Edge(t) = ((Ax + (Cx - Ax)t, (Ay + (Cy - Ay)t, (Az + (Cz - Az)t, (Aw + (Cw - Aw)t)$$

- E.g. If x = 1,

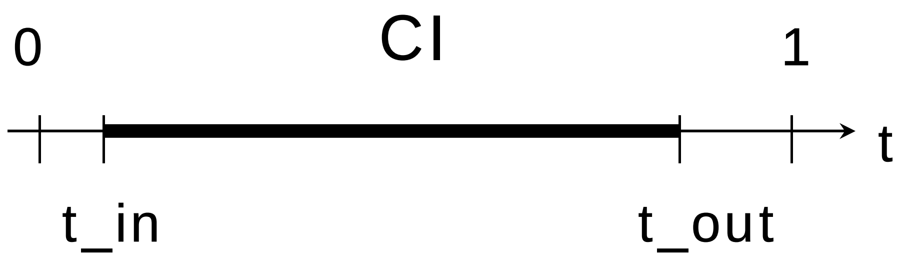
$$\frac{Ax + (Cx - Ax)t}{Aw + (Cw - Aw)t} = 1$$

- Solving for t above,

$$t = \frac{Aw - Ax}{(Aw - Ax) - (Cw - Cx)}$$

Candidate Interval

- If not trivial accept/reject, then clip
- Define Candidate Interval (CI) as time interval during which edge might still be inside CVV. i.e. CI = t_in to t_out
- Initialize CI to [0,1]



- Conversely: values of t outside CI = edge is outside CVV

Shortening Candidate Interval

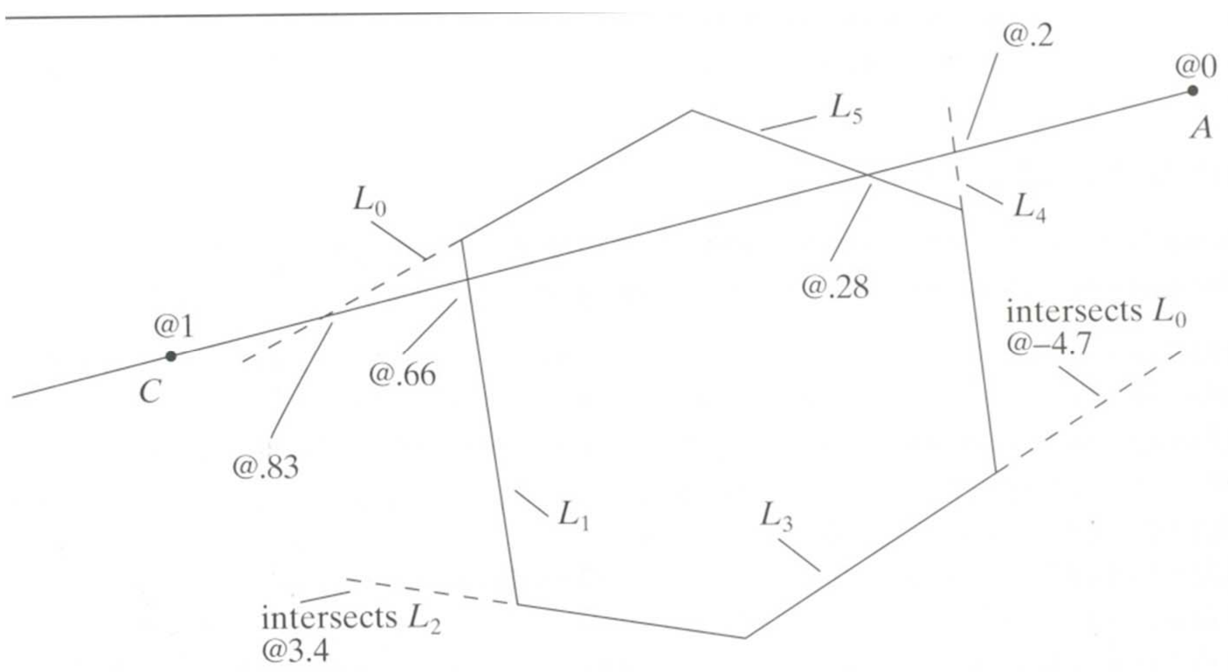
- **Algorithm:**
  - Test for trivial accept/reject (stop if either occurs)
  - Set CI to [0,1]
  - For each of 6 planes:
    - Find hit time t_hit
    - If t_in,   new t_in = max(t_in,t_hit)
    - If t_out, new t_out = min(t_out, t_hit)
    - If t_in > t_out => exit (no valid intersections)

Note: seeking smallest valid CI without t_in crossing t_out

Shortening Candidate Interval

Example to illustrate search for t_in, t_out

Note: CVV is different shape. This is just example



Line test	t_{in}	t_{out}
0	0	0.83
1	0	0.66
2	0	0.66
3	0	0.66
4	0.2	0.66
5	0.28	0.66

Calculate choppped A and C

- If valid t_in, t_out, calculate adjusted edge endpoints A, C as

- A_chop = A + t_in (C – A) (calculate for Ax,Ay, Az)
- C_chop = A + t_out (C – A) (calculate for Cx,Cy,Cz)

3D Clipping Implementation

- Function clipEdge( )
- Input: two points A and C (in homogenous coordinates)
- Output:
  - 0, if no part of line AC lies in CVV
  - 1, otherwise
  - Also returns clipped A and C
- Store 6 BCs for A, 6 for C

Store BCs as Outcodes

- Use outcodes to track in/out
  - Number walls 1… 6 (or 0.. 5)
  - Bit *i* of A’s outcode = 0 if A is inside ith wall
  - 1 otherwise
- Example: outcode for point outside walls 1, 2, 5

Wall no.	0	1	2	3	4	5
OutCode	0	1	1	0	0	1

Trivial Accept/Reject using Outcodes

- Trivial accept: inside (not outside) all walls

Wall no.	0	1	2	3	4	5
A Outcode	0	0	0	0	0	0
C OutCode	0	0	0	0	0	0

Logical bitwise test: A | C == 0

- Trivial reject: point outside **same** wall. Example Both A and C outside wall 1

Wall no.	0	1	2	3	4	5
A Outcode	0	1	0	0	1	0
C OutCode	0	1	1	0	0	0

Logical bitwise test: A & C != 0

3D Clipping Implementation

- Compute BCs for A,C store as outcodes
- Test A, C outcodes for trivial accept
- Test A,C outcodes for trivial reject
- If not trivial accept/reject:
  - Compute tHit
  - Update t_in, t_out
  - If t_in > t_out, early exit

3D Clipping Pseudocode

```
int clipEdge(Point4& A, Point4& C)
{
      double tIn = 0.0, tOut = 1.0, tHit;
      double aBC[6], cBC[6];
      int aOutcode = 0, cOutcode = 0;

      …..find BCs for A and C
      …..form outcodes for A and C

      if((aOutCode & cOutcode) != 0) // trivial reject
         return 0;
      if((aOutCode | cOutcode) == 0) // trivial accept
         return 1;
```

3D Clipping Pseudocode

```
for(i=0;i<6;i++)  // clip against each plane
{
      if(cBC[i] < 0)  // exits: C is outside
      {
                tHit = aBC[i]/(aBC[i] – cBC[I]);
                tOut = MIN(tOut, tHit);
      }
      else if(aBC[i] < 0)  // enters: A is outside
      {
                tHit = aBC[i]/(aBC[i] – cBC[i]);
                tIn = MAX(tIn, tHit);
      }
      if(tIn > tOut) return 0; // CI is empty: early out
}
```

3D Clipping Pseudocode

```
Point4 tmp;  // stores homogeneous coordinates
If(aOutcode != 0) // A is out: tIn has changed
{
    tmp.x = A.x + tIn * (C.x – A.x);
    // do same for y, z, and w components
}
If(cOutcode != 0) // C is out: tOut has changed
{
    C.x = A.x + tOut * (C.x – A.x);
    // do same for y, z and w components
}
A = tmp;
Return 1; // some of the edges lie inside CVV
}
```

References

- Angel and Shreiner, Interactive Computer Graphics, 6th edition
- Hill and Kelley, Computer Graphics using OpenGL, 3rd edition