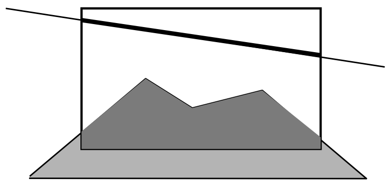
# Line/Polygon Clipping

(FVD 3.12)

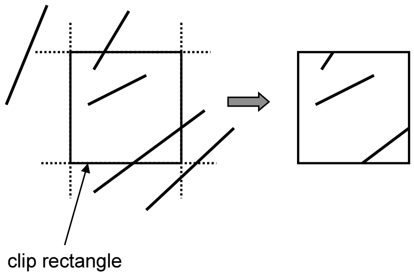


**The problem**:

Given a set of 2D lines or polygons and a window, clip the lines or polygons to their regions that are *inside* the window.

# Motivation

- Efficiency.
- Display in portion of a screen.
- Occlusions.



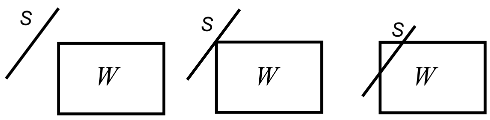
## Clipping (cont.)

- We will deal only with lines (segments).
- Our window can be described by two extreme points:

  $(x_{min},y_{min})$ and $(x_{max},y_{max})$
- A point (x,y) is in the window iff:

  $x_{min} \leq x \leq x_{max}$ and $y_{min} \leq y \leq y_{max}$

## Analytic Solution




0, 1, or 2 intersections between a line and a window

- The intersection of two convex regions is always convex (why?).
- Since both *W* and *S* are convex, their intersection is convex, i.e a single connected segment of *S*.
- **Question**: Can the boundary of two convex shapes intersect more than twice?

## Vector Calculus - Preliminaries

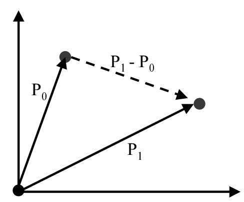
- A 2D vector V is defined as:

  $V=(V_x,V_y)$.
- Scalar (dot) product between two vectors V and U is defined:

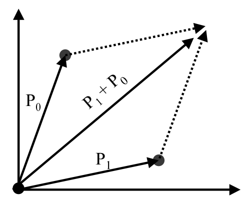
$$V \cdot U = \begin{bmatrix} V_x & V_y \end{bmatrix} \begin{bmatrix} U_x \\ U_y \end{bmatrix} = V_x U_x + V_y U_y = |V||U| \cos\theta$$

- If $V \cdot U = 0$ then V and U are perpendicular to each other.

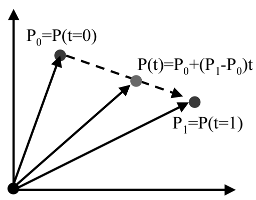
Vector Subtraction



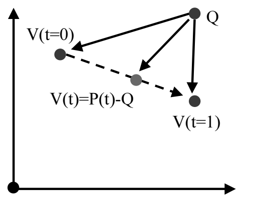

Vector Addition



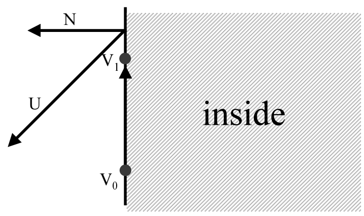
## Parametric Line




## Changing the Origin



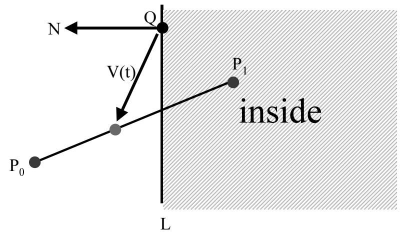

## Inside/Outside Test




- Assume WLOG that V=($V_1$-$V_0$) is the border vector where "inside" is to its right.
- If V=($V_x$,$V_y$), N is a prep' vector pointing outside, where we define:

  N=(-$V_y$,$V_x$)
- Vector U points "outside" if $N \cdot U > 0$
- Otherwise U points "inside".

# Segment-Line Intersection

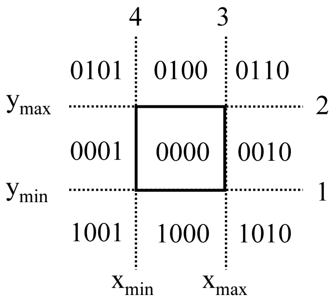



- The parametric line $P(t)=P_0+(P_1-P_0)t$
- The parametric line $V(t)=P(t)-Q$
- The segment intersects the line L at $t_0$ satisfying $V(t_0)\cdot N=0$.
- The intersection point is $P(t_0)$.
- The vector $\Delta=P_1-P_0$ points "inside" if $(P_1-P_0)\cdot N<0$. Otherwise it points "outside".
- If L is vertical, intersection can be computed using the explicit equation.

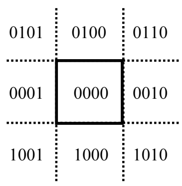
# Cohen-Sutherland for Line Clipping

- Clipping is performed by the computation of the intersections with four boundary segments of the window: $L_i$, i=1,2,3,4
- **Purpose**: Fast treatment of lines that are trivially inside/outside the window.
- Let $P=(x,y)$ be a point to be classified against window $W$.
- **Idea**: Assign $P$ a binary code consisting of a bit for each edge of $W$, whose value is determined according to the following table:

| bit | 1 | 0 |
|---|---|---|
| 1 | $y < y_{min}$ | $y \geq y_{min}$ |
| 2 | $y > y_{max}$ | $y \leq y_{max}$ |
| 3 | $x > x_{max}$ | $x \leq x_{max}$ |
| 4 | $x < x_{min}$ | $x \geq x_{min}$ |

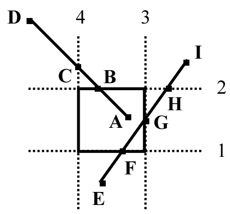



## Cohen-Sutherland (cont.)




- Given a line segment *S* from $p_0=(x_0,y_0)$ to $p_1=(x_1,y_1)$ to be clipped against a window *W*.
- If code($p_0$) **AND** code($p_1$) is not zero - then *S* is *trivially rejected.*
- If code($p_0$) **OR** code($p_1$) is zero - then *S* is *trivially accepted.*

- Otherwise: let assume w.l.o.g. that $p_0$ is outside the window *W*.
  - Find the intersection of *S* with the edge corresponding to the MSB in code($p_0$) that equal to 1. Call the intersection point $p_2$.
  - Rerun the procedure for the new segment ($p_1$ , $p_2$).




# Cohen-Sutherland Algorithm

```
CompOutCode (x, y : real; var code : outcode);
/* Compute outcode for the point (x,y)  */
begin
    code := 0;
    if Y > Ymax then code := code | B1000
       else
          if Y < Ymin then code := code | B0100;
    if X > Xmax then code := code | B0010
       else
          if X < Xmin then code := code | B0001;
end;
```

## Cohen-Sutherland Algorithm

```
CS (x0,y0,x1,y1,xmin,xmax,ymin,ymax)
{
      boolean accept, done :;
      float outcode0, outcode1, x, y;
      accept := false ;
      done    := false ;
      CompOutCod (x0,y0,outcode0);
      CompOutCod (x1,y1,outcode1);
        repeat
           if  ((outcode0 | outcode1 ) == 0)
             {
             /*  Trivial accept  */
                 accept := true ;
                 done    := true ;
             }
           else
       if ((outcode0 & outcode1 ) <> 0){
             /*  Trivial reject  */
                  done := true
            }
           else
/* Failed both tests, so calculate the line
segment to clip from an outside point to an
intersection with clip edge   */
                .
                .
                .
```

## Cohen-Sutherland Algorithm (cont.)

```
/* At least one endpoint is outside the clip
rectangle, pick it*/
if (outcode0 <> 0) {
   outcodeOut := outcode0
else
   outcodeOut := outcode1;

/* now find the intrsection point by using the
formulas:
y= y0 + slope*(x-x0), and
x = x0 + (1/slope)*(y-y0) */

if (outcodeOut & 0x1000) then
    divide line at top of clip rectangle;

else if (outcodeOut & 0x0100) then
    divide line at bottom of clip rectangle;

else if (outcodeOut & 0x0010) then
    divide line at right edge of clip rectangle;

else if (outcodeOut & 0x0001) then
    divide line at left edge of clip rectangle;
```

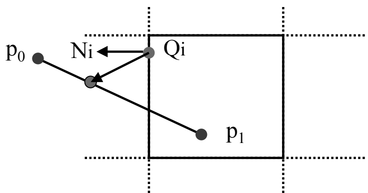
## Cohen-Sutherland Algorithm (cont.)

```
/*  Now we move outside point to intersection point
to clip, and get ready for next pass  */
if (outcodeOut == outcode0)
{
  x0:=x;y0:=y;CompOutCod (x0,y0,outcode0);
}
 else {
 x1:=x;y1:=y;CompOutCod (x1,y1,outcode1);
}

}  /* Subdivide  */
  until (done) ;

 if (accept) draw_line (x0,y0,y0,y1);
} /* end */
```

## Cyrus-Beck Line Clipping




- Denote $p(t)=p_0+(p_1-p_0)t \quad t\in[0..1]$
- Let $Q_i$ be a point on the edge $L_i$ with outside pointing normal $N_i$.
- $V(t) = p(t)-Q_i$ is a parameterized vector from $Q_i$ to the segment P(t).
- $Ni \cdot V(t) = 0$ iff $V(t) \perp N_i$
- We are looking for *t* satisfying the above equation:

## Cyrus-Beck Clipping (cont.)

$0 = Ni \cdot V(t)$

$= Ni \cdot (p(t)-Q_i)$

$= Ni \cdot (p_0+(p_1-p_0)t-Q_i)$

$= Ni \cdot (p_0-Q_i) + Ni \cdot (p_1-p_0)t$

Solving for t we get:

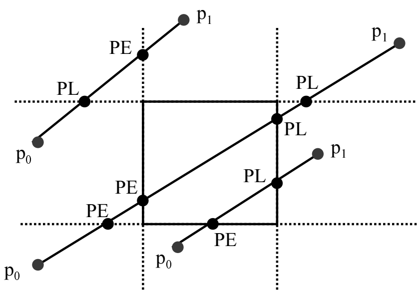
$$t = \frac{Ni \cdot (p_0-Q_i)}{-Ni \cdot (p_1-p_0)} = \boxed{\frac{Ni \cdot (p_0-Q_i)}{-Ni \cdot \Delta}}$$

where $\Delta=(p_1-p_0)$

- **Comment**: If $Ni \cdot \Delta=0$, t has no solution. However, in this case $V(t) \perp N_i$ and there is no intersection.

## Cyrus-Beck Algorithm:

- The intersection of *p(t)* with all four edges $L_i$ is computed, resulting in up to four $t_i$ values.
- If $t_i<0$ *or* $t_i>1$ , $t_i$ can be discarded.
- Based on the sign of $Ni \cdot \Delta$, each intersection point is classified as ***PE*** (potentially entering) or ***PL*** (potentially leaving).
- ***PE*** with the largest *t* and ***PL*** with the smallest *t* provide the domain of p(t) inside *W*.
- The domain, if inverted, signals that p(t) is totally outside.

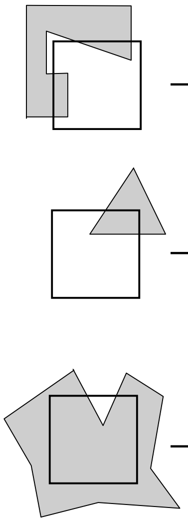



# Cyrus-Beck Line Clipping

```
precalculate Ni and select a Pei for each edge;

for each line segment to be clipped
   if ( P1 = P2 ) then
             line is degenerate so clip as a point;
  else {
         tPE = 0;  tPL = 1;
       for each candidate intersection with a clip edge
          if (( <Ni, D> ) <> 0 ) then {
          /* Ignore edges parallel to line for now */
            calculate t;
           sign of <Ni, D> categorizes as PE or PL ;
                  if  PE then tPE = max (tPE, t);
                  if  PL then tPL = min  (tPL, t);
                  }
              if  (tPE > tPL) return null
              else
                 return P(tPE) and P(tPL) ;
                 /* as true clip intersections */
          };
```

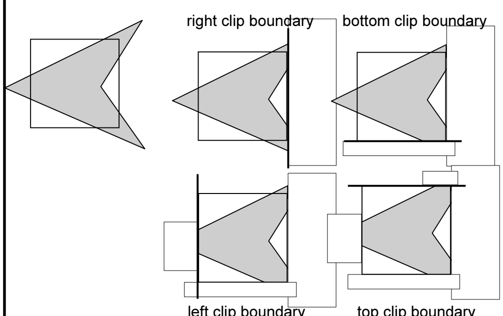
# Polygon Clipping



# Sutherland-Hodgman Polygon-Clipping Algorithm

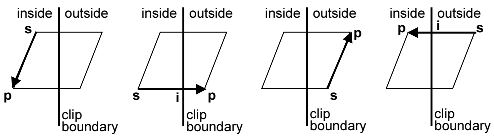
Clip a polygon by successively clipping against each (infinite) clip edge.

After each clipping a new set of vertices is produced.



For each clip edge - consider the relation between successive vertices of the polygon:

Assume vertex **s** has been dealt with, vertex **p** follows:




**p** added to output list | **i** added to output list | no output | **i** and **p** added to output list

# Sutherland - Hodgman polygon Cliping Algoruthm

```
type
    vertex = point;          /* point holds real x, y */
    edge = array [1..2] of vertex;
    /* Max declared as constant */
    vertexArray = array [1..MAX] of vertex;

procedure SutherlandHodgmanPolygonClip (
inVertexArray: vertixArray;              /* input vertex array */
var outVertexArray: vertexArray;      /*output vertex array */
inLength: integer;         /* num of entries in inVertexArray */
var outLength:integer; /*num of entries in outVertexArray */
clipBoundary : edge                    /* Edge of clip polygon */
);
```

# Sutherland - Hodgman (cont.)

```
var
    s, p      /* Start, End point of current polygon edge */
    i : vertix;  /* Intersection point with clip boundary */
    j : integer;                          /* vertex loop counter */
begin
    outLength :=0;
    s := inVertexArray [inLength];
   /* Start with the kast vertex in inVertexArray*/
   for  j:=1 to inLength do
        begin
            p := inVertexArray[j ];
            if  Inside (p, clipBoundary ) then
               if  Inside (s, clipBoundary ) then
                  Output (p,outLength, outVertexArray) /*case #1*/
              else
                 begin   /* case # 4 */
                    Intersect (s, p, clipBoundary, i);
                    Output (i, outLength, outVertexArray );
                    Output (p, outLength, outVertexArray );
                end
            else
              if  Inside (s, clipBoundary ) then
                 begin    /* case # 2 */
                     Intersect (s, p, clipBoundary, i );
                    Output (i, outLength, outVertexArray );
                end;
        s := p ;        /* Advance to next pair of vertices */
          end    /* for */
end;       /* SutherlandHodgmanPolygonClip */
```