2D Geometrical Transformation

Translation

$P(x, y) \rightarrow P'(x', y')$

Move from original position to a new position by adding a translation distance (or translation vector): $P' = P + T$

swift

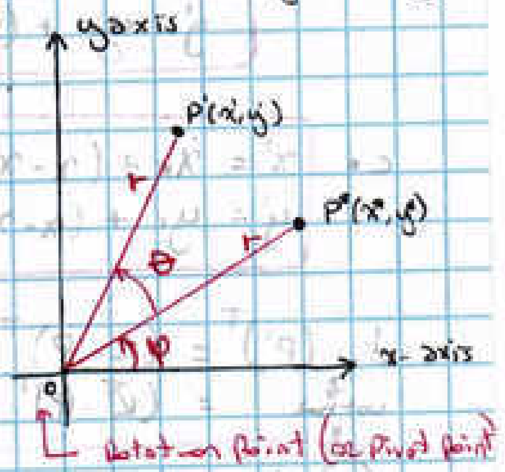
$$x' = x + T_x \qquad y' = y + T_y \qquad \left\{ P = \begin{bmatrix} x \\ y \end{bmatrix} \quad P' = \begin{bmatrix} x' \\ y' \end{bmatrix} \quad T = \begin{bmatrix} T_x \\ T_y \end{bmatrix} \right.$$

y-axis
T
P'
P
x axis

$$P' = P + T = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

Rotation

Repositioning an object $P(x, y)$ to a new position $P'(x', y')$ along a circular path in the xy plane



1. Rotation at the origin:

a. Positive values of rotation are defined as a rotation counter clock wise

Rotation angle θ

b. $x' = r\cos(\varphi + \theta)$
$= r\cos\varphi\cos\theta - r\sin\varphi\sin\theta$
$y' = r\sin(\varphi + \theta)$
$= r\cos\varphi\sin\theta + r\sin\varphi\cos\theta$

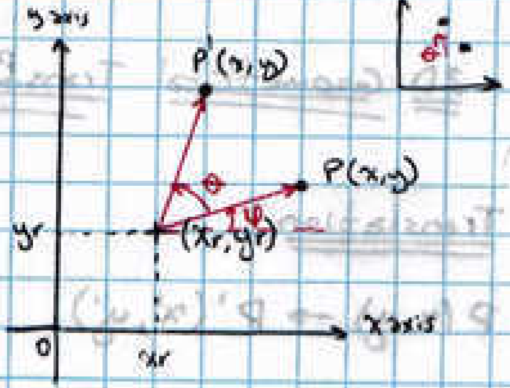
original point / Polar coordinates

c. $x = r\cos\varphi$
$y = r\sin\varphi$

d. $\boxed{x'} = \underbrace{r\cos\varphi}\cos\theta - \underbrace{r\sin\varphi}\sin\theta$
$= \boxed{x\cos\theta - y\sin\theta}$
$\boxed{y'} = \underbrace{r\cos\varphi}\sin\theta + \underbrace{r\sin\varphi}\cos\theta$
$= \boxed{x\sin\theta + y\cos\theta}$

$$P' = R \cdot P = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

2. Rotation on an arbitrary pivot position:



a. Two way to solve it:

i) rotate ~~at~~ pivot point $(P')^T = P^T \cdot R^T$

ii) Translate to origin point

Rotate

Translate back to pivot point

b. $x' = x\cos\theta - y\sin\theta$

$$\Downarrow$$

$$x' - xr = (x - xr)\cos\theta - (y - yr)\sin\theta$$

$$\Downarrow$$

$$\boxed{x' = xr + (x - xr)\cos\theta - (y - yr)\sin\theta}$$

~~$y' = x\sin\theta + r\sin\theta$~~

$y' = x\sin\theta + y\cos\theta$

$$\Downarrow$$

$$y - yr = (x - xr)\sin\theta + (y - yr)\cos\theta$$

$$\Downarrow$$

$$\boxed{y' = yr + (x - xr)\sin\theta + (y - yr)\cos\theta}$$

c. $$\boxed{\begin{aligned} x' &= xr + (x - xr)\cos\theta - (y - yr)\sin\theta \\ y' &= yr + (x - xr)\sin\theta + (y - yr)\cos\theta \end{aligned}}$$

d. $$\left.\begin{aligned}(P')^T &= (R \cdot P)^T \\ &= (R^T \cdot (P)^T\end{aligned}\right\}$$ Representation as row vector instead of column vector

not true

$\Downarrow$

e. ⊗ $P' + T = R + T \cdot P + T$ ⊗ cant be done

Wrong

$$\begin{bmatrix} x' \\ y' \end{bmatrix} + \begin{bmatrix} tx \\ ty \end{bmatrix} = \left[\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} + \begin{bmatrix} tx \\ ty \end{bmatrix}\right] \cdot \left[\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} tx \\ ty \end{bmatrix}\right]$$

we can't add matrices that are not of the same size

~~$= tx\cos\theta - ty\sin\theta$~~

$=$

$$\Longrightarrow$$

~~Rotations (continued)~~

Scaling

1. Alter the size of an object by multiplyig the object's coordinate values (x,y) by the scaling values (Sx,Sy)

2. P→P'

$$\left.\boxed{\begin{aligned} x' &= x \cdot Sx \\ y' &= y \cdot Sy \end{aligned}}\right\} \quad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} \cdot \begin{bmatrix} Sx & 0 \\ 0 & Sy \end{bmatrix} \qquad P' = S \cdot P$$

3. Any positive valuo can be assigned to Sg and Sy

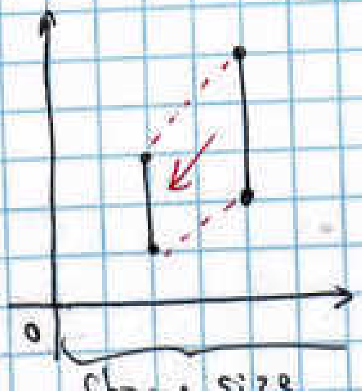
4. ~~$0 < Sx < 1$~~

$0 < S < 1$: Reduce size of object
$S = 0$: Do not change object
$1 < S < \infty$: Enlarge size of object

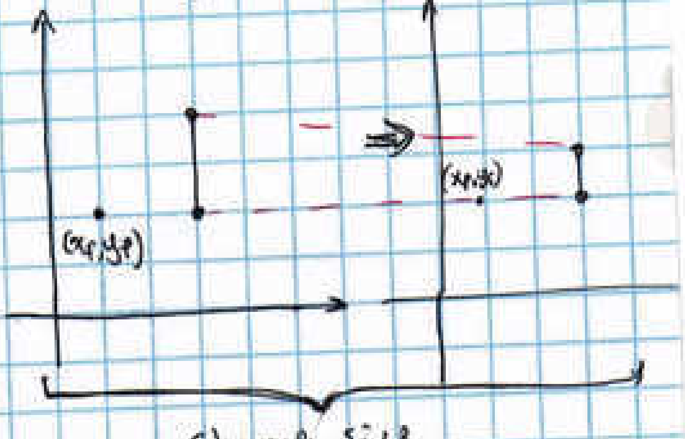
5. Uniform scaling: Maintaining relative object proportion
$\boxed{Sx = Sy}$

6. Differencial scaling: changing shape of object
$\boxed{Sx \neq Sy}$

7. ~~The~~ scale anobject relative to a fixed point.
A fixed point is the location in which we can control the scaled object.
Why we care? without a fixed point When we ~~change~~ scale the object it will change places. ej



Change size
&
Change Position



Change size
do &
Do Not change position

a) $x' = x \cdot S_x$

$y' = y \cdot S_y$

~~b) $x_f + x$~~

b) $$\left.\begin{aligned} x' - x_f &= (x - x_f) \cdot S_x \\ y' - y_f &= (y - y_f) \cdot S_y \end{aligned}\right\} \quad \boxed{\begin{aligned} x' &= x_f + (x - x_f)\, S_x \\ y' &= y_f + (y - y_f)\, S_y \end{aligned}}$$

$\Downarrow$

~~$x' = x\, S_x - x_f$~~

c) $$\begin{aligned} \boxed{x'} &= x_f + (x - x_f)\, S_x \\ &= x_f + x \cdot S_x - x_f \cdot S_x \\ &= x \cdot S_x + x_f - x_f \cdot S_x \\ &= \boxed{x \cdot S_x + x_f (1 - S_x)} \end{aligned}$$

$$\begin{aligned} \boxed{y'} &= y_f + (y - y_f)\, S_y \\ &= y_f + y \cdot S_y - y_f \cdot S_y \\ &= y\, S_y + y_f - y_f \cdot S_y \\ &= \boxed{y \cdot S_y + y_f (1 - S_y)} \end{aligned}$$

$$\boxed{\begin{aligned} x' &= x \cdot S_x + x_f (1 - S_x) \\ y' &= y \cdot S_y + y_f (1 - S_y) \end{aligned}}$$

$x_f(1 - S_x)$ and $y_f(1 - S_y)$ are constant for all points in the object

Matrix Representation And Homogeneous Coordinate (MRAHC)

- Many Graphics involve sequence of geometric transformation such as translation, rotation, scaling.etc

- Each basic transformation can be expressed in the general form:

$$P' = M_1 \cdot P + M_2$$

P' & P: represented as column vectors
M1: Matrix array 2x2 (containing multiplicative factors)
M2: . 2element column matrix (containing translational terms)
- Terms are associated w/ the pivot point or scaling fixed point

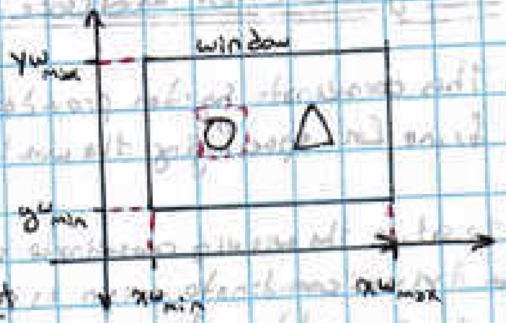
A different approach: combine transformations so final coordinate positions are obtained directly from initial coordinates.

By expanding the 2by2 matrices to 3 by3, we can combine the multiplicative and translational terms for two dimensional geometric transformation into a single matrix

To express any 2D transformation as a matrix multiplication, Represent each Cartesian coordinate

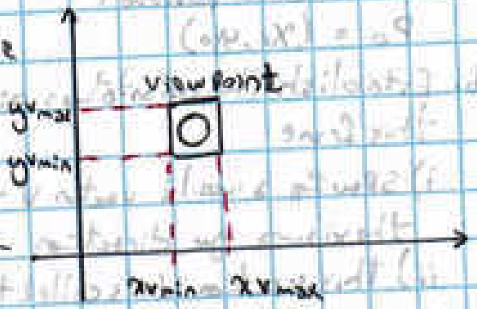
2D Viewing

1. A world-coordinate area selected for display is called a window

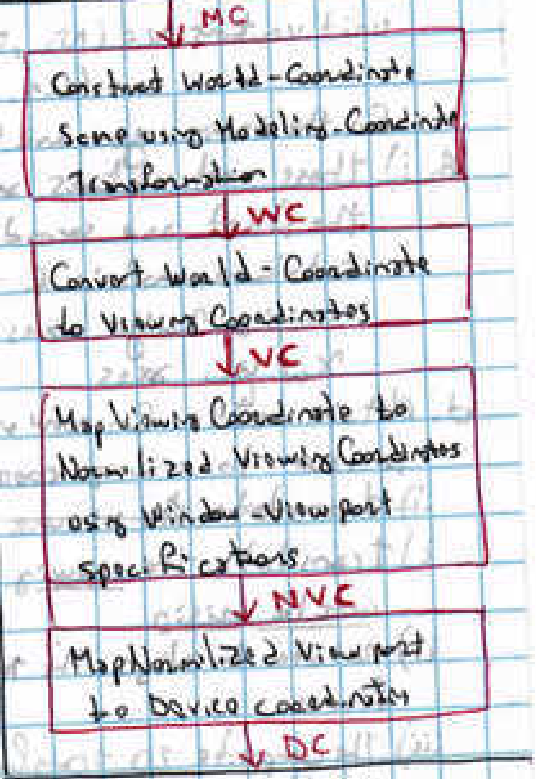


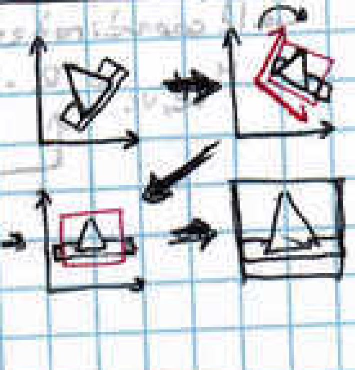
2. An area on de device to which a window is mapped is called a viewport

3. the mapping of a part of a world-coordinate scene to device coordinate is referred to as a viewing transformation
   a. A 2D viewing transformation is also refereed as window-to-viewport transformation or the windowing transformation



4. we carry out the viewing transformation in several steps:
   i) Construct the scene world coordinate using the output primitives and attributes.
   ii) set up a 2D view-coordinate system in the world-coordinate plane, to obtain a particular orientation for the window.
   iii) Define a window in the viewing-coordinate system
   note) The viewing coordinate reference frame is used to provide a method for setting up arbitrary orientations for rectangular windows.
   iv) Once the viewing reference frame is setup, transform discriptions in the world coordinate to view coordinates (in range from 0 to 1)
   v) map the viewing-coordinate description of the scene to normalized coordinates
   vi) All parts of the picture that lie outside the viewport are clipped
   vii) the content of the viewport are transfered to the device coordinate.
   note) By changing the position of the viewport, we can view objects at different positions on the display area of an output display


MC
Construct World-Coordinate Scene using Modeling-Coordinate Transformation
WC
Convert World-Coordinate to Viewing Coordinates
VC
Map Viewing Coordinate to Normalized Viewing Coordinates using Window-Viewport specifications
NVC
Map Normalized Viewport to Device coordinates
DC



8 Viewing Coordinate Reference Frame

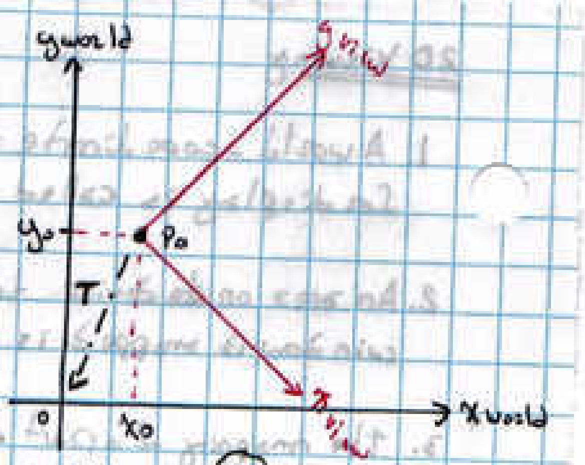
1. This coordinate system provides the reference frame for specifying the world Coordinate window.

2. to setup the viewing coordinate system: using
   a. A view-coordinate origin is selected at some world position
      $P_0 = (x_0, y_0)$
   b. Establish the orientation or rotation of this frame
      i) Specify a world vector V that defines the viewing y_v direction
      ii) this vector V is called the view up vector.
   c. Given V, calculate the components of unit vectors $v = (v_x, v_y) \leftrightarrow (y_v)$
      $u = (u_x, u_y) \leftrightarrow (x_v)$
      for the viewing y_v and x_v axes
3. i) these unit vectors are used to form the first and second rows of the rotation matrix R that aligns the viewing $x_v y_v$ axes with the world $x_w y_w$ axes
   d. obtain matrix for converting world coordinate position to viewing coordinates:
      ~~i) translate the viewing coordinate~~
      i) translate the viewing origin to the world origin
      ii) Rotate to align the two coordinate reference frames
      iii) The composite 2D transformation to ~~convert~~ convert world coordinates to viewing coordinates is

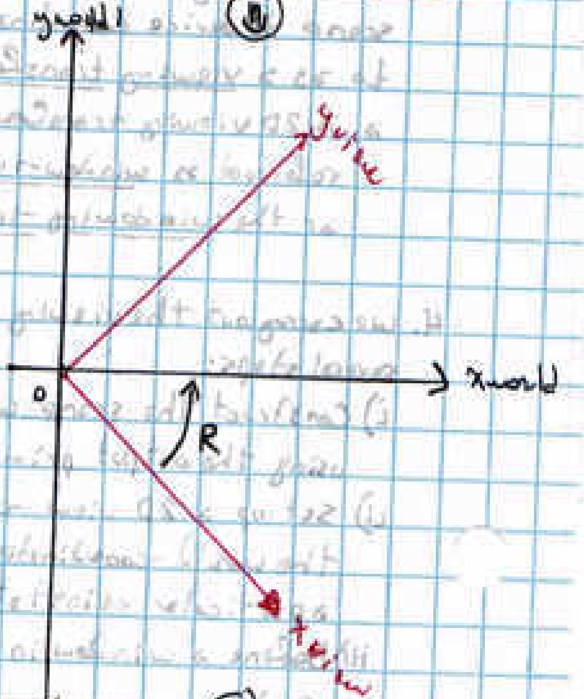
$$M_{WC,VC} = R \cdot T$$

← T: Translation matrix takes viewing world origin P_0 to world origin

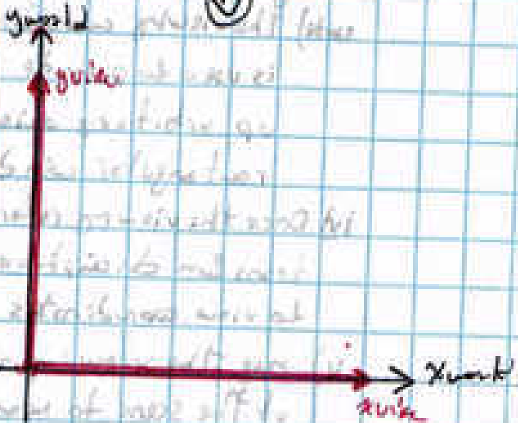
R: rotation matrix that aligns the axes of the two reference frames



(i)

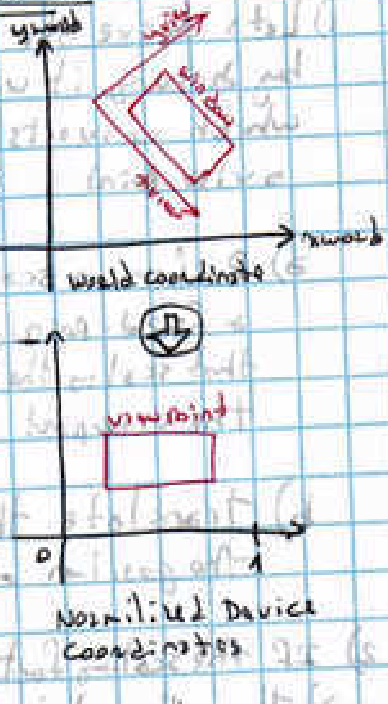


(ii)



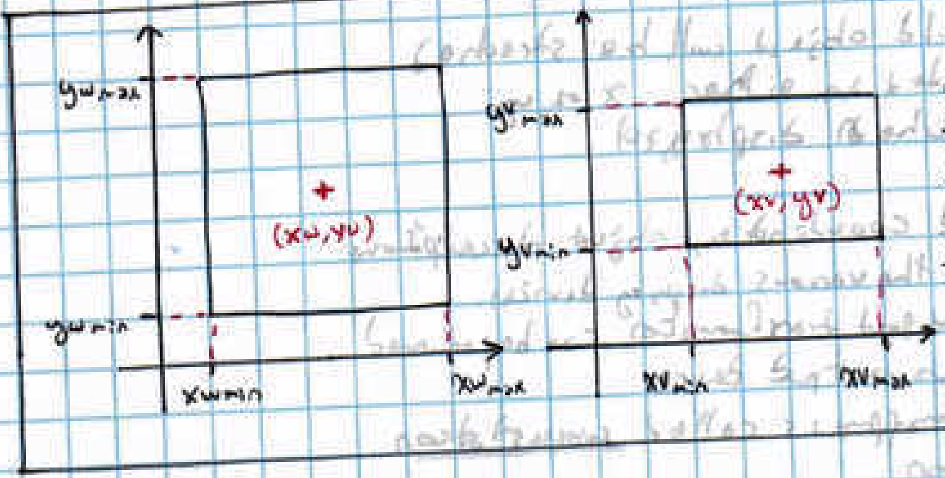
~~View~~

Window-To-Viewport Coordinate transformation



1. Once object description have been transfered to to viewing reference frame
   a) Choose the window extents in viewing coordinates
   b) select the viewport limits in normilized coordinates

2) object description are then transfered to normilized coordinates
   a) Do this using a transformation that maintains the same relative placement of objects in normalized space as they had in viewing coordinates.
      i) If a coordinate position is at the center of the viewing window, it will be displayed at the center of the viewport



A point at position (xw, yw) in a designated window is mapped to viewport coordinates (xv, yv) so that relative positions in the two areas are the same

3) To maintain the same relative placement in the viewport:

a) $$\frac{xv - xv_{min}}{xv_{max} - xv_{min}} = \frac{xw - xw_{min}}{xw_{max} - xw_{min}} \quad \text{and} \quad \frac{yv - yv_{min}}{yv_{max} - yv_{min}} = \frac{yw - yw_{min}}{yw_{max} - yw_{min}}$$

b) The viewport position (xv, yv): $xv = xv_{min} + (xw - xw_{min})\,Sx$

$xy = yv_{min} + (yw - yw_{min})\,Sy$

where: $Sx = \dfrac{xv_{max} - xv_{min}}{xw_{max} - xw_{min}}$

$Sy = \dfrac{yv_{max} - yv_{min}}{yw_{max} - yw_{min}}$

P221

4) Another way to perform the conversion:

1) Let's derive the view port position (xv, yv) by deriving it with a set of translation which converts the window area into a view point

a) Perform a scaling transformation using a fixed-point position of (xwmin, ywmin) that scales the window area to the size of the viewport

b) translate the scaled window area to the position of the viewport

2) If the scaling factors are the same (sx = sy)

a) then the relative proportions of objects are maintained

b) else world object will be streched or contracted in either x or y direction when displayed

3) From normalized coordinates, object descriptions are mapped to the various display devices.

a) window-to-view port transformation can be performed for each open output device

b) this way of mapping is called <u>workstation transformation</u>

c) this is accomplished by selecting a window area in normalized space and a viewport area in the coordinates of the display device.

4) With the workstation transformation, we can have more control over the positioning of parts of a scene for different output devices

Clipping Operations

1) the recognition of portions of a picture which are either inside or outside of a specific Region of space is called a clipping Algorithm (or clipping)

2) the region against which an object is to be clipped is called clip window

3) For the viewing transformation, we wish to display only the parts of the picture which are inside the window area.
Everything outside the window area is discarded

4) On raster systems, clipping algorithms are combined w/scan conversion

5) Different types of clipping Algorithms
   a) Point Clipping
   b) Line clipping (straight-line segments) } standard of graphic packages
   c) Area Clipping (polygons)
   d) Curve clipping
   e) Text Clipping

Point Clipping

1) Let assume that the clip window is a ~~rectangular~~ rectangle in standard position
2) Save point $P = \{x, y\}$ for display if these inequalities are satisfied
   a) $xw_{min} \leq x \leq xw_{max}$
   -and-
   b) $yw_{min} \leq y \leq yw_{max}$

3) The edges of the clip window are:
   xw_{min}, xw_{max}, yw_{min}, yw_{max}
   ~~a) these can be window world or [illegible]~~
   ~~or~~ a) these can be world or viewport
   ~~b)~~
4) if not inside inequalities then drop point

Line Clipping

1) Test a given line segment to determine whether it lies complitly inside the clipping window

2) If a given line segment does not lies complettly inside the clipping window, we must perform intersection calculation w/ one or more clipping boundaries

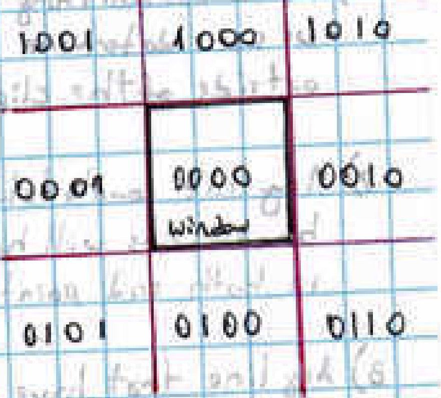
3) Process line through the "inside-outside" test by checking the end points
   a) A line w/ both end points such as P1 to P2 is saved
   b) else any one of the endpoints is outside the edge reduce the line to fit inside the clipping window

4) Let assume we have a line segment w/ end points (x1,y1) & (x2,y2) ~~2) the~~ and one or both end points are outside the clipping rectangle.
   a) the parametric representation.

$$\left.\begin{array}{l} x = x_1 + u(x_2 - x_1) \\ y = x_1 + u(y_2 - y_1) \end{array}\right\} 0 \leq u \leq 1$$

   can be used to determine values of parameter u for intersection w/ a rectangle boundary ~~edge~~ coordinate
   i) If the value u for an intersection is outside the range 0 to 1 the line does not enter the interior
   ii) If the value of u is inside the range 0 to 1 the segment does need a cross into clipping area.
   iii) line segments that are parallel to window edges can be handled as special cases.

Cohen - Sutherland Line Clipping



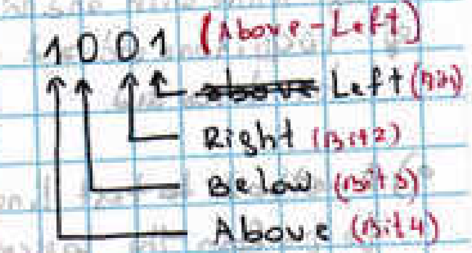
1001	1000	1010
0001	0000 window	0010
0101	0100	0110

1) This algorithm perfoam an initial test in order to reduce the number of intersections that must be calculated. Therefore, the processing of line segments is speeded up.

2) Every line end-point is assigned a four-digit binary code called a region code which identifies the location of the point relative to the boundaries of the clipping rectangle.

3) Each bit position in the region code is used to indicate one of the four relative coordinate positions of the point with respect to the clip window ⇨

a) A value of 1 in any bit position indicates that the point is in that relative position else the bit is set to 0



4) By comparing end point coordinates values (x,y) to the clip boundaries, we can determine the bit values of the region code. eg:

a) Bit 1 (Left): set to 1 if $x < xw_{min}$

b) Bit 2 (Right): set to 1 if $x > xw_{max}$

c) Bit 3 (Below): set to 1 if $y < yw_{min}$

d) Bit 4 (Above): set to 1 if $y > yw_{max}$

5) If bit manipulation is possible, the region code can be determine by:

a) Calculate differences between endpoint coordinates and clipping boundaries.

b) Use the resultant sign bit of each different calculation to set the corresponding value in the region code.

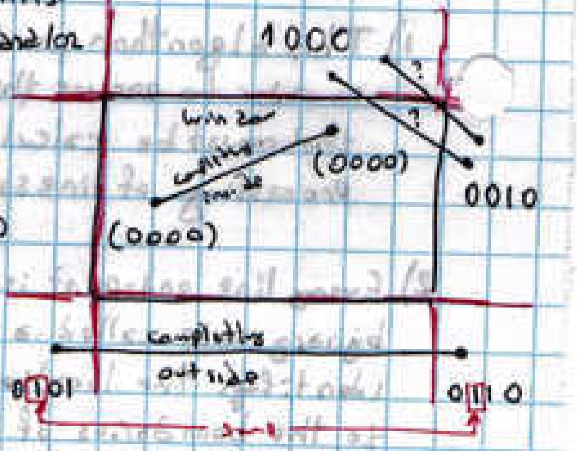
i) Bit 1 (Left): sign bit of $x - xw_{min}$

ii) Bit 2 (Right): sign bit of $xw_{max} - x$

iii) Bit 3 (Below): sign bit of $y - yw_{min}$

iv) Bit 4 (Above): sign bit of $yw_{max} - y$

6) After establishing the region codes for all endpoints: we can determine which lines are inside and/or outside of the clip window completely



7) Any line completely contained within the boundaries will have the region code 0000 in both end points

8) Any line that have a 1 in the same bit position in the region point is completely outside
eg: A line with end point with the region code
1001 in one end and 0101 in another end would be discarded

9) A method to test line for total clipping is to perform the logical 'and' operation with both region codes

$$\begin{array}{r} 1010 \\ 0101 \\ \hline \underbrace{0000}_{\text{inside}} \end{array} \qquad \begin{array}{r} 1001 \\ 1000 \\ \hline \underbrace{1000}_{\text{outside}} \end{array}$$

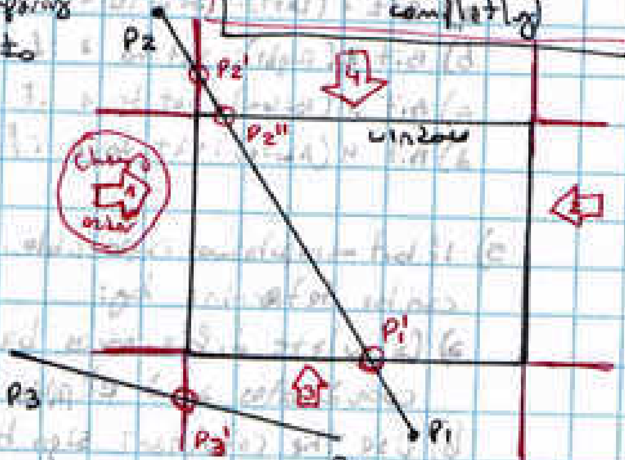
$$\begin{array}{r} 1000 \\ 0010 \\ \hline 0000 \end{array}$$

inside (but not completely)

special case

10) If a line cannot be checked as completely inside or outside the clip window must be checked for intersection with the window boundaress

11) Begin the clipping process for a line by comparing an outside end point to a clipping boundary to determine how much of the line can be discarded

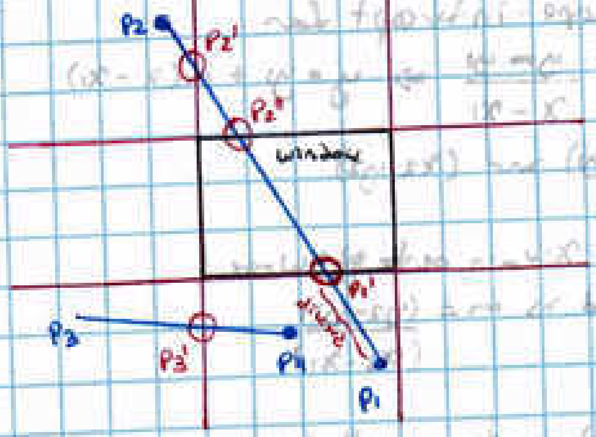


12) then the remaining part of the line is checked against the other boundaries.
Continue until either the line is totally discarded or a section is found inside the window.

a) the order of checking line end points against the clipping boundaries is in this order:
1- Left
2- Right
3- bottom
4- top

Cohen-Sutherland Clipping steps:

1) Lets assume we got the following lines:



a) Start with the line $P_1 P_2$ (fr P_1 to P_2)

b) Begin with the endpoint P_1

i) Check against left, right, and bottom boundaries in turn

ii) find that point is below the clipping rectangle.

c) find the intersection point P_1' with the bottom boundarie and discard the line section from P_1 to P_1'

d) the line now is reduced from P_1' to P_2

2) P_2 is outside the clip window

a) Check this end point against boundaries and find that it is above the left of the clip window.

b) Intersection P_2' is calculated but this point is above the window

c) So, final intersection calculation yields P_2''.

3) the line from P_1' to P_2'' is saved!!

4) Next line $P_3 P_4$

a) ~~Pass~~ Point P_3 is to the left of the clipping rectangle

~~b) Check region codes~~

b) we determine the intersection P_3'

c) we eliminate the line section for P_3 to P_3'

5) By checking the region codes for line P_3' to P_4, we find the remainder line is below the clip window and it is discarded

Cohen-Sutherland Algorithm Calculations

1) We are calculating the intersection point with a clipping boundary using the slope-intercept form of line equation slope$(m) = \frac{y - y_1}{x - x_1} \Rightarrow y = y_1 + m(x - x_1)$

for a line with end points (x_1, y_1) and (x_2, y_2)

a) the x value is set either xw_{min} or to ~~y~~ xw_{max}.

b) the slope of line is calculated as $m = \frac{(y_2 - y_1)}{(x_2 - x_1)}$

2) If we are looking for the intersection with the a horizontal boundary, the x coordinate can be calculated as:

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

with y set either to yw_{min} or to yw_{max}

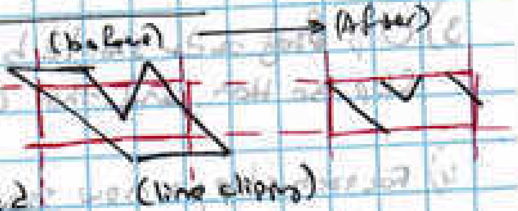
Cohen- Sutherland Line-Clipping Algorithm code example

```
#define Round (a) ((int) (a + 0.5))
// Bit masks encode a point's position relative to the clip edges.
// A point's status is encoded by OR'ing together appropriate bit masks.
#define LEFT_EDGE 0x1
#define RIGHT_EDGE 0x2
#define BottoM_EDGE 0x4
#define TOP_EDGE 0x8
/* Points encoded as 0000 are completly inside the clip window
    all others are outside at least one edge.
   If OR'ing two codes is FALSE (no bits are set in either code),
   the line can be accepted
   If the AND operation between two codes is TRUE, the line
    defined by those end points is completly outside the
    clip window and can be rejected*/
#define INSIDE (a)  (!a)
#define REJECT (a, b)  (a&b)
#define ACCEPT (a, b) (!(a|b))

unsigned char. encode (wcPt2 pt, dcPt winMin, dcPt winMax){
   unsigned char code = 0x00
   if (pt.x < winMin.x)
      code = code | LEFT_EDGE;
   if(pt.x > winMax.x)
      code = code | RIGHT EDGE;
   if(pt.y < winMin.y)
      code = code | Botton_EDGE;
   if (pt.y> winMax.y)
      code = code | ~~Botton~~top_EDGE;
   return(code)
}

void swapCodes (unsigned char *c1, unsigned char *c2){
  unsigned char tmp;
  tmp = *c1; *c1 = *c2; c2 = tmp;
}
```

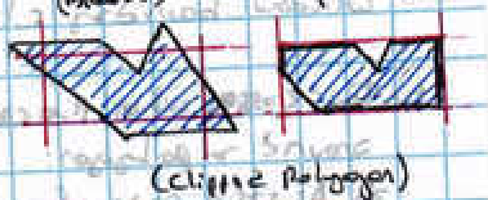
```
void clipline (dcPt winMin, dcPt winMax, wcPt2 p1, wcPt2 p2){
  unsigned char code1, code2;
  int done = FALSE, draw = FALSE;
 float m;
 while (! done){
  code1 = encode (p1, winMin, winMax);
  code2 = encode (p2, winMin, winMax);
  if (ACCEPT (code1, code2)){
     done = true;
     draw = true
  } else
    if (Reject (code1, code2)){
    done = true
  }else{
    //Ensure that p1 is outside window
    if (INside (code1)) {
      swap Ptr (&p1, &p2);
      swap Codes (&code1, &code2)
   }
  //use slope (m) to find line-clip edge intersections
  if (p2.x != p1.x)
    m = (p2.y - p1.y) / (p2.x - p1.x)
  if (code1 & LEFT_EDGE){
    p1.y += (winMin.x - p1.x) * m;
    p1.x = winMin.x;
  }else
  if (code & Right_EDGE){
    p1.y + = (winMax.x - p1.x) * m;
    p1.x = winMax.x;
 }else
  if (code1 & Bottom_EDGE){
    //Need to update p1.x for non-vertical lines only
    if (p2.x != p1.x)
       p1.x += (winMin.y - p1.y) /m;
    p1.y = winMin.y
  } else
  if (code1 & top EDGE)){
    if(p2.x != p1.x)
       p1.x+= (wMax.y - p1.y) /m;
    p1.y = winMax.y
 }
```

```
      }
    }
  }
  if (draw){
    line(Round(p1.x), Round(p1.y), Round(p2.x), Round(p2.y));
  }
}
```

Polygon Clipping



1) A polygon boundary processed w/ a line clipped may be displayed as a series of unconnected line segment.



2) What we really want to display is a bounded area after the clipping.

3) We required an algorithm that will generate one or more closed areas that are then scan converted for the appropriate area fill.

4) the output should be a sequence of vertices that defines the clipped polygon boundaries