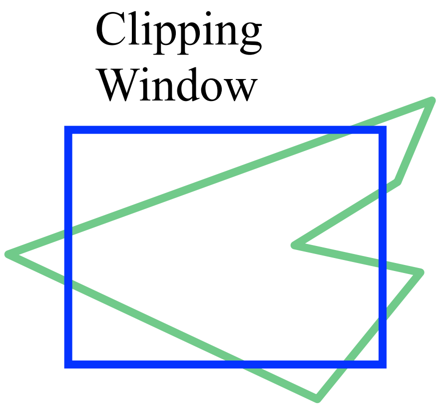
2D Viewing

Tomb of Pashed

2D Viewing pipeline

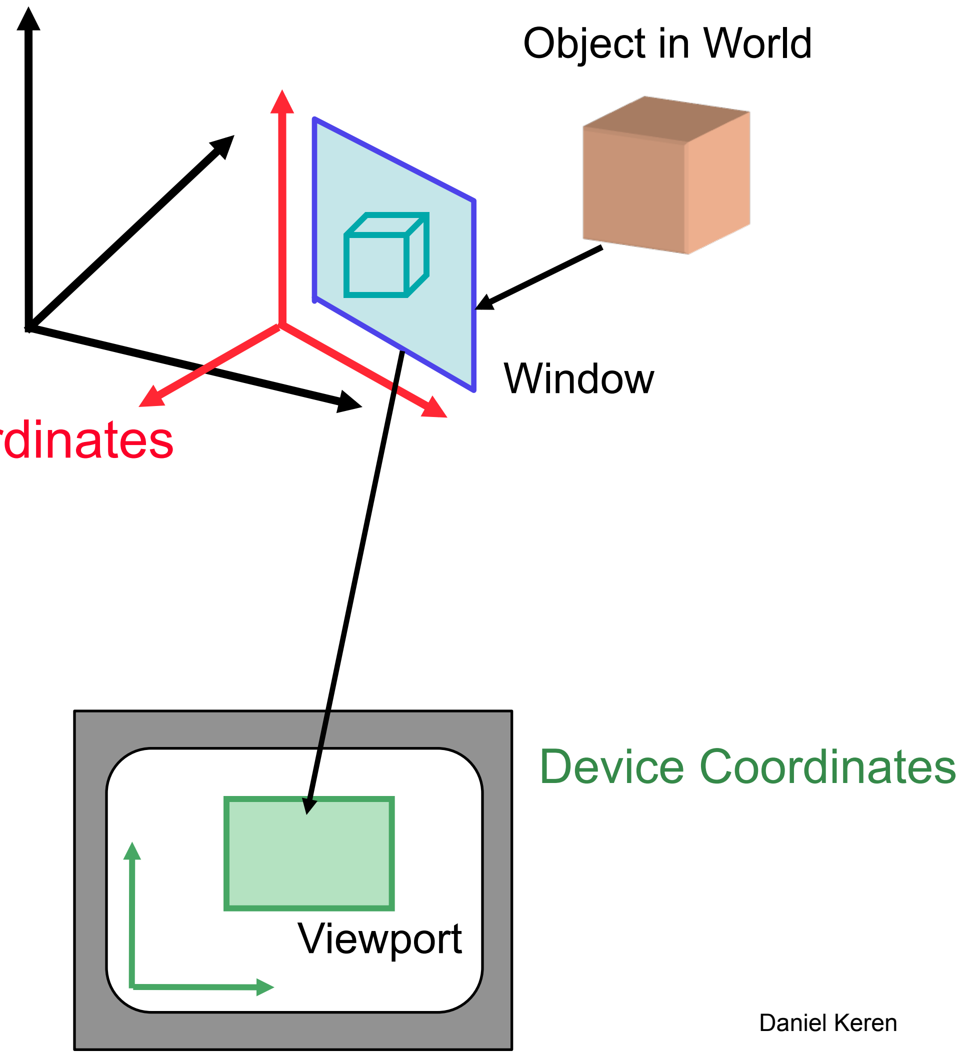
– Clipping window
  - Area of 2D scene that is selected for display
  - "what we see" like in a camera viewfinder

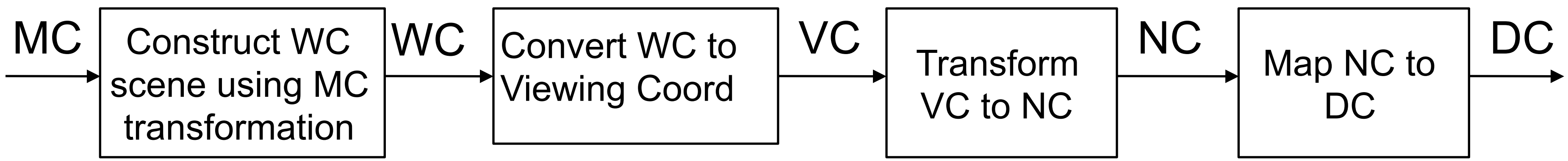


– Viewport
  - Where we see it on the display (screen)

– 2D viewing transformation (window-to-viewport transformation or windowing transformation)
  - Maps scene in 2D world coordinates to device coordinates

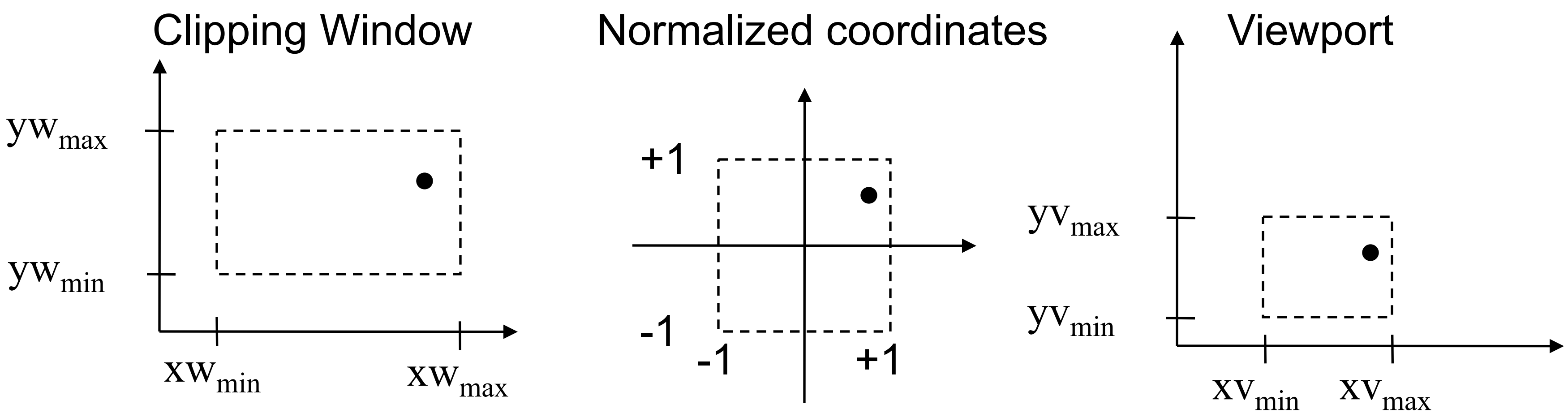
3D Viewing


Object in World
World Coordinates
Window
Viewing coordinates
Device Coordinates
Viewport
Daniel Keren



- MC: Modeling Coordinates
- WC: World Coordinates
- VC: Viewing Coordinates (clipping window)
- NC: Normalized Coordinates
  - Makes viewing device-independent
  - (0..1 or -1..1 better for clipping)
- DC: Device (monitor) Coordinates - the Viewport
- Device-independent transforms concatenated into 1 matrix

Clipping Window to normalized coordinate then to viewport



- Translate and scale to move between coordinates
  - relative position of a point the same on all three
- If aspect ratio of viewport not same as window, may look stretched/squished

Clipping Window to Viewport (matrix form)

Translate to origin and scale to normalized coords (-1..1), (-1..1):

$$\begin{bmatrix} \frac{2}{xw_{\max} - xw_{\min}} & 0 & 0 \\ 0 & \frac{2}{yw_{\max} - yw_{\min}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -\frac{xw_{\max} + xw_{\min}}{2} \\ 0 & 1 & -\frac{yw_{\max} + yw_{\min}}{2} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{2}{xw_{\max} - xw_{\min}} & 0 & -\frac{xw_{\max} + xw_{\min}}{xw_{\max} - xw_{\min}} \\ 0 & \frac{2}{yw_{\max} - yw_{\min}} & -\frac{yw_{\max} + yw_{\min}}{yw_{\max} - yw_{\min}} \\ 0 & 0 & 1 \end{bmatrix}$$

Scale to viewport size (xv_{max}-xv_{min}, yv_{max}-yv_{min}) and translate to (xv_{min}..xv_{max})

$$\begin{bmatrix} 1 & 0 & \frac{xv_{\max} + xv_{\min}}{2} \\ 0 & 1 & \frac{yv_{\max} + yv_{\min}}{2} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{xv_{\max} - xv_{\min}}{2} & 0 & 0 \\ 0 & \frac{yv_{\max} - yv_{\min}}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{xv_{\max} - xv_{\min}}{2} & 0 & \frac{xv_{\max} + xv_{\min}}{2} \\ 0 & \frac{yv_{\max} - yv_{\min}}{2} & \frac{yv_{\max} + yv_{\min}}{2} \\ 0 & 0 & 1 \end{bmatrix}$$

OpenGL Viewing (2D)

- OpenGL has no core 2D viewing functions
  - Use 3D with z as 0

- First set projection mode

```
glMatrixMode( GL_PROJECTION );
```

- Make sure we start with identity matrix

```
glLoadIdentity( );
```

– Define 2D clipping window

```
gluOrtho2D ( xwmin, xwmax, ywmin, ywmax );
```

- Or use OpenGL core-library 3D clipping window

```
glOrtho( xwmin, xwmax, ywmin, ywmax, zmin, zmax );
```

– Define viewport relative to the display window

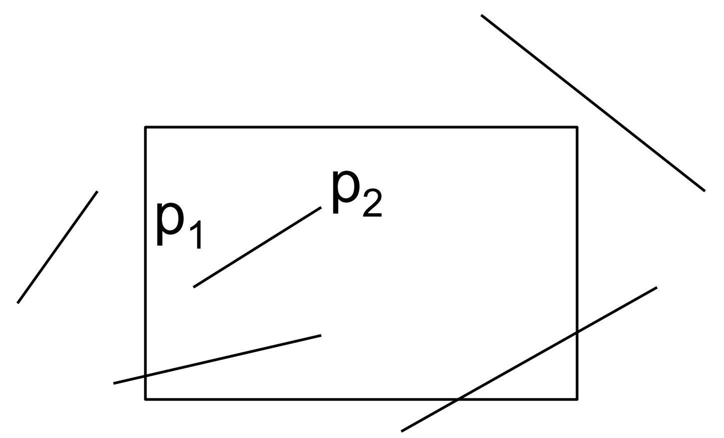
```
glViewport ( xvmin, yvmin, vpWidth, vpHeight );
```

– Define display window using GLUT library calls

2D line clipping

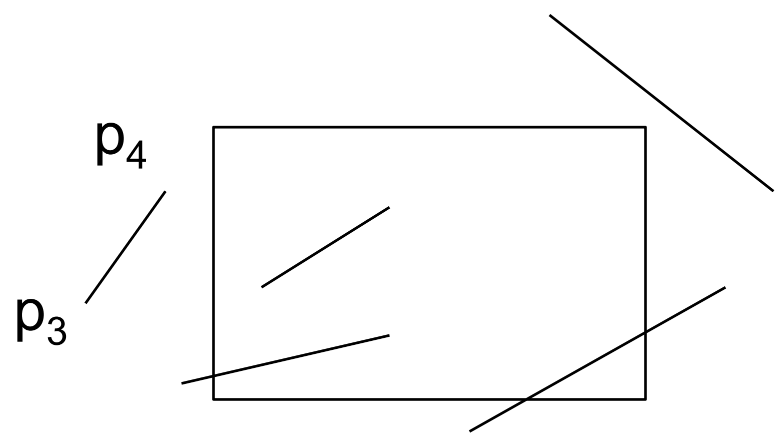
- Expensive to determine intersection of line with clipping area
  - Avoid as much as possible by doing trivial accept or reject first

- Trivial accept
  - If both endpoints within all four clipping boundaries



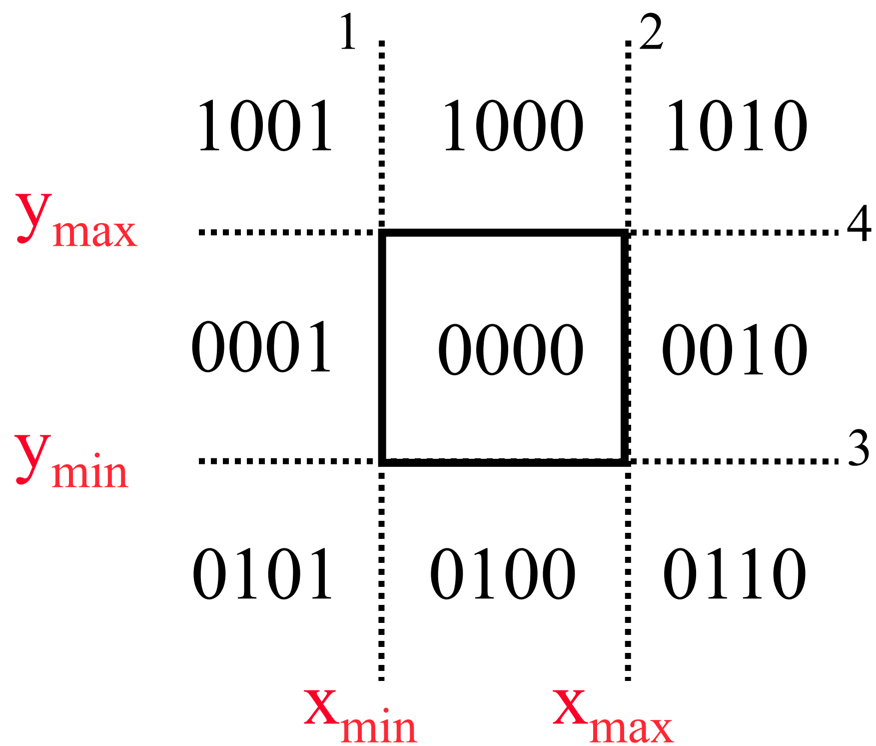
– Trivial reject

- If both endpoints outside one of the boundaries

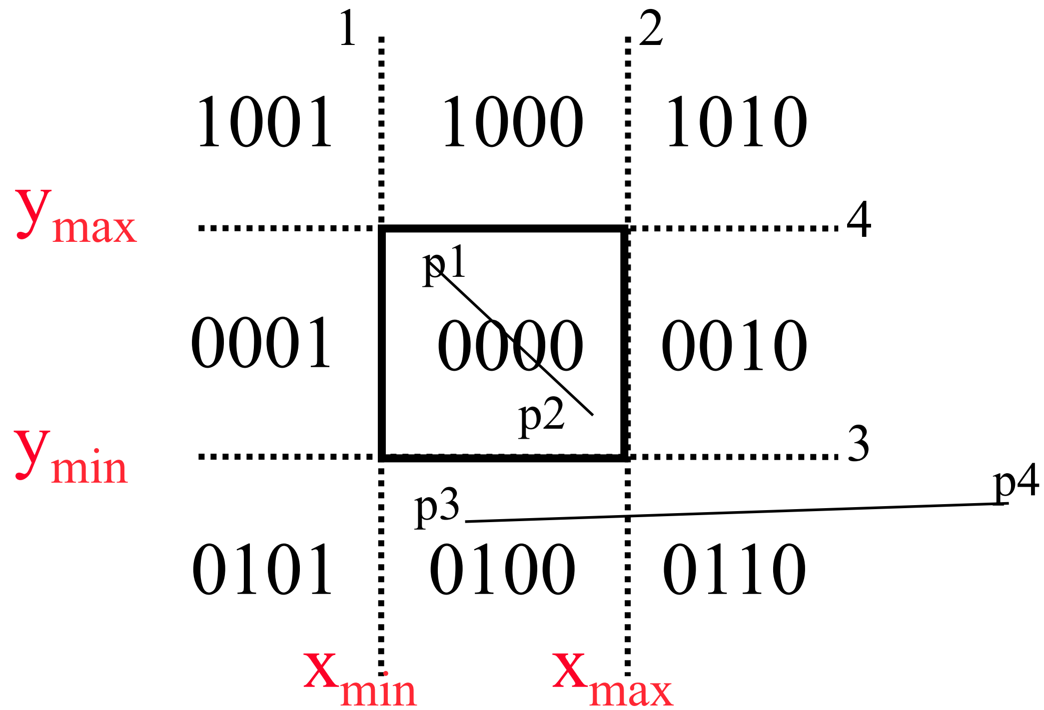


Cohen-Sutherland Line Clipping

- Every line endpoint assigned four-digit binary region code or out code depending on where it is located
  - Bit 1: left
  - Bit 2: right
  - Bit 3: bottom
  - Bit 4: top
  - 0 means in
    1 means out



- Cases that do not require intersection testing



$$p1 = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}$$

$$p2 = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}$$

Trivial Inside Case: All bits 0
(p1|p2)==0

$$p3 = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix}$$

$$p4 = \begin{bmatrix} 0 & 1 & 1 & 0 \end{bmatrix}$$

Trivial Outside Case: Any matching bits both 1:
(p3&p4)!=0

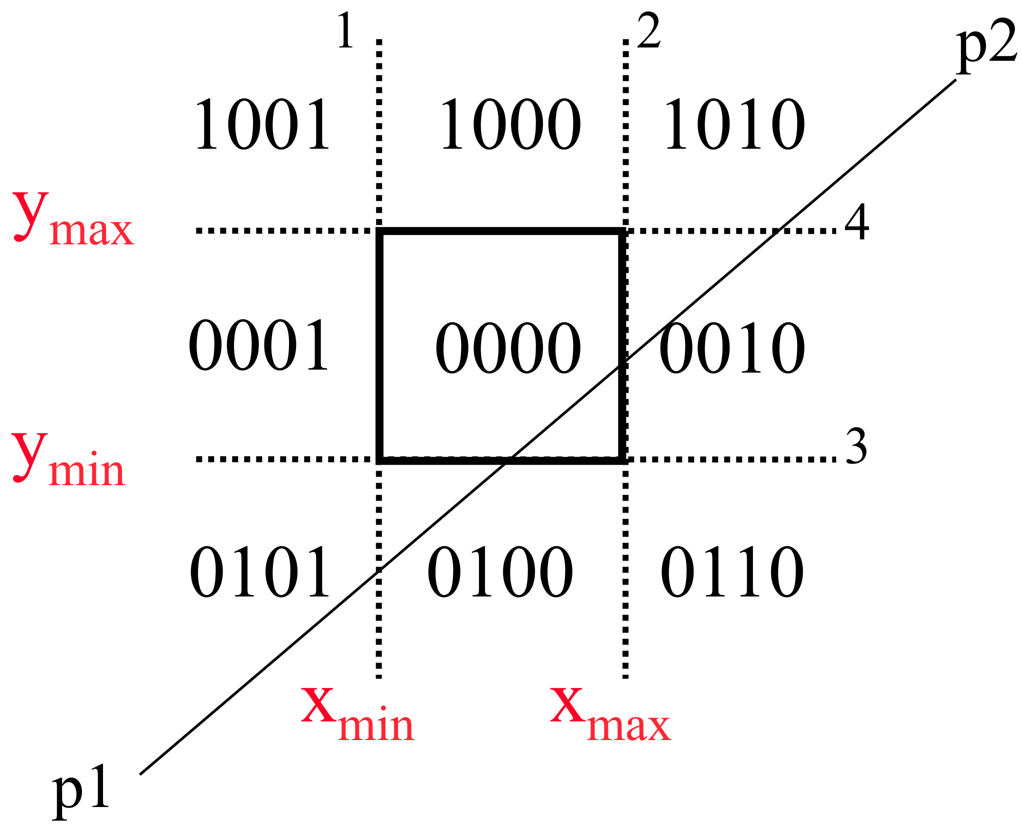
– Intersection points: use point-slope equation for line

- Left/Right: Set x value to xw_{min} or xw_{max} then find y value

$$y = y_0 + m(x - x_0)$$

- Top/Bottom: Set y value to yw_{min} or yw_{max} to find x value

$$x = x_0 + \frac{1}{m}(y - y_0)$$



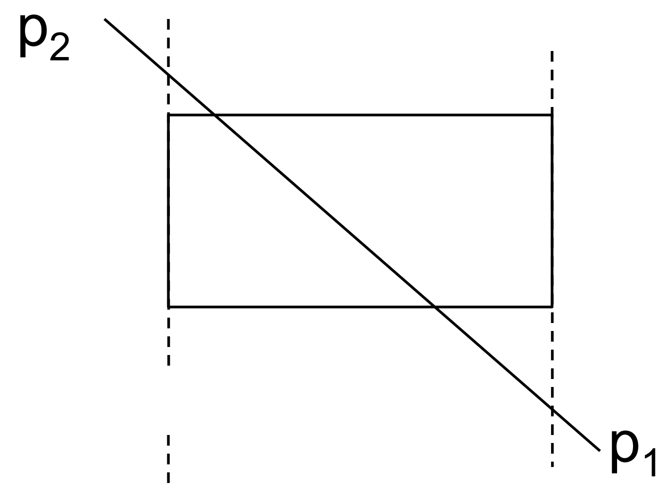
$$p1 = \begin{bmatrix} 0 & 1 & 0 & 1 \end{bmatrix}$$

$$p2 = \begin{bmatrix} 1 & 0 & 1 & 0 \end{bmatrix}$$

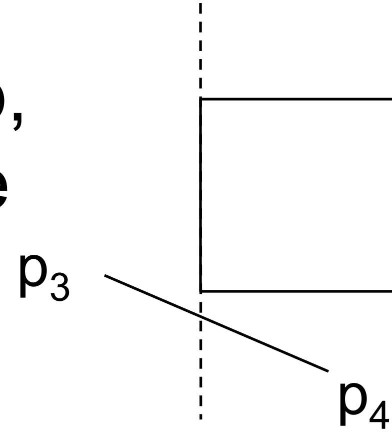
Cross all 4 borders, but didn't actually need to clip x_{min} and y_{max}.

- When bit values are different (one is 0 the other is 1) then the line crosses that boundary
- Clip against each crossed boundary

- If order is left, right, top, bottom, this clips 4 as opposed to the optimal 2.



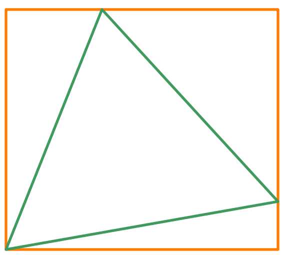
- In this example, after the left clip, determine that the line is outside



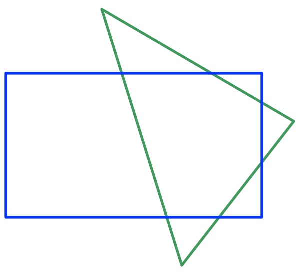
- Re-compute bits after each clip
  - Done when either $p_1|p_2$==0 (inside remain) or p_1&p_2!=0 (outside remain)

Polygon area clipping

- Different than edge clipping since we must maintain a closed polygon (not just a disjointed set of edges)
- Trivial reject done by looking at bounding box
  - Determine the minimum and maximum extents of all the vertices in both x and y direction
  - If the entire bounding box outside the clipping area, discard the polygon

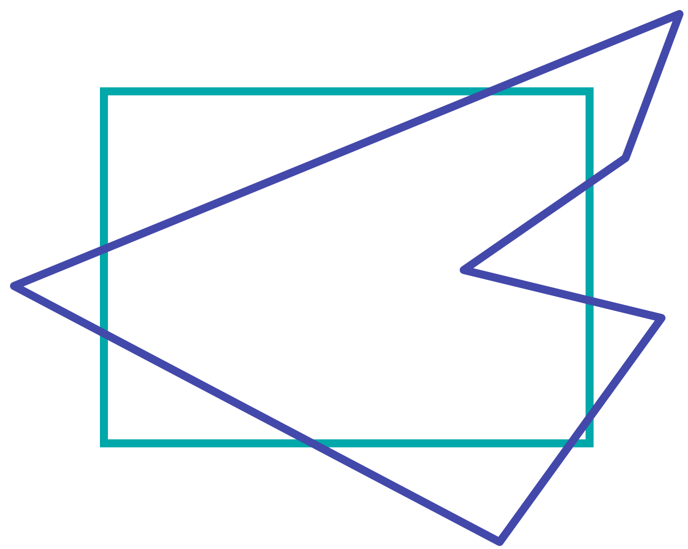


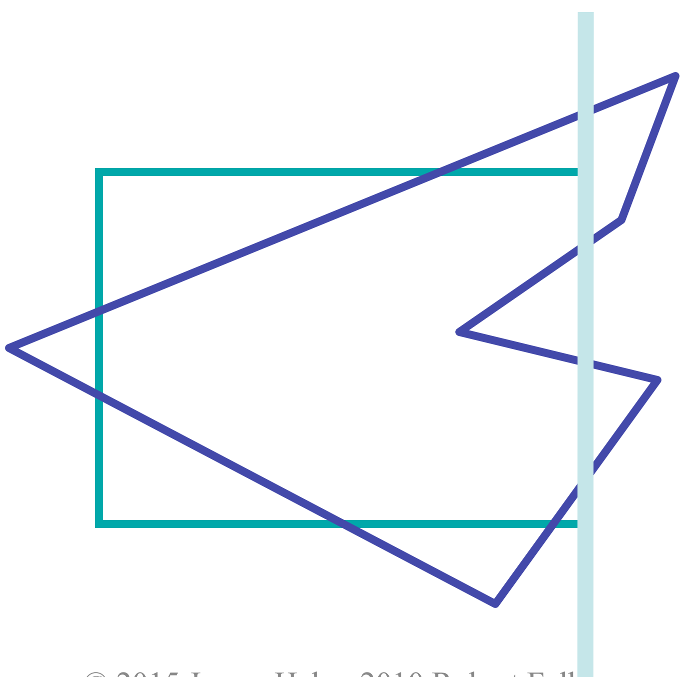
- General strategy: as we clip against each bounding region, re-generate a vertex list for the polygon

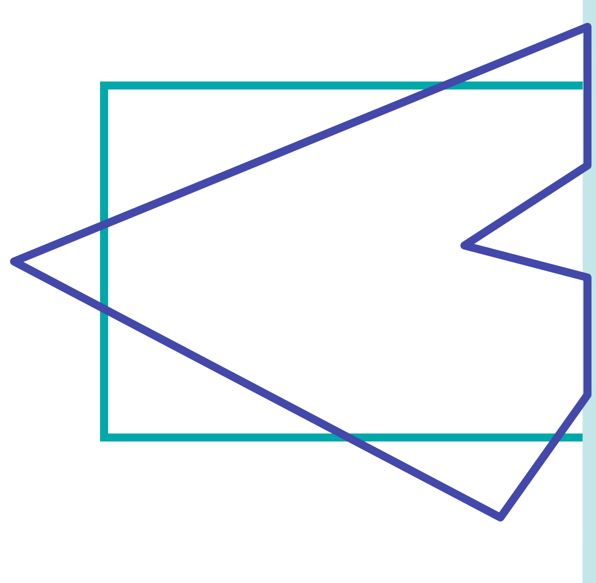


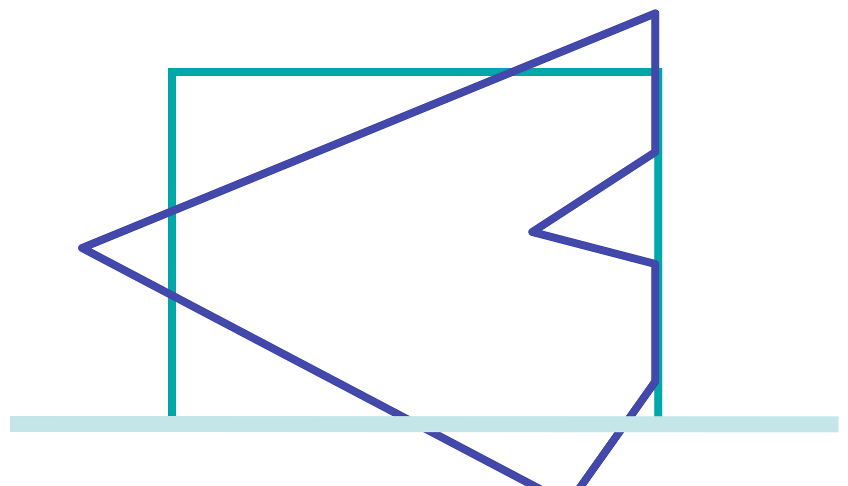
Sutherland-Hodgman Clipping

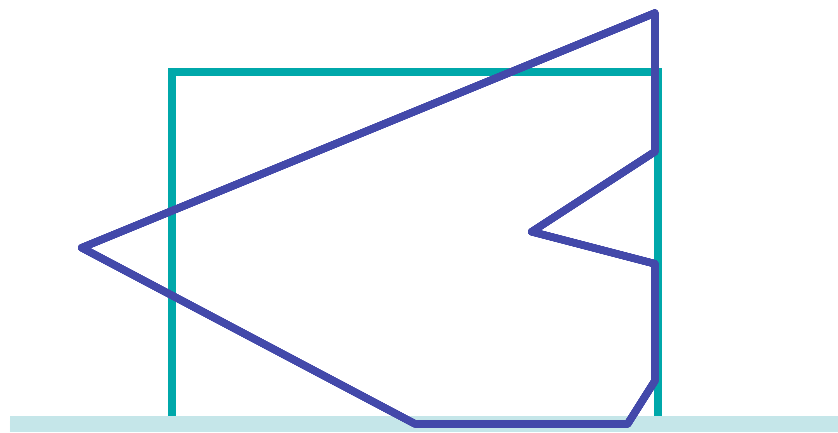
– Consider each edge of the clip region individually
– Clip the polygon against the clip region edge's equation

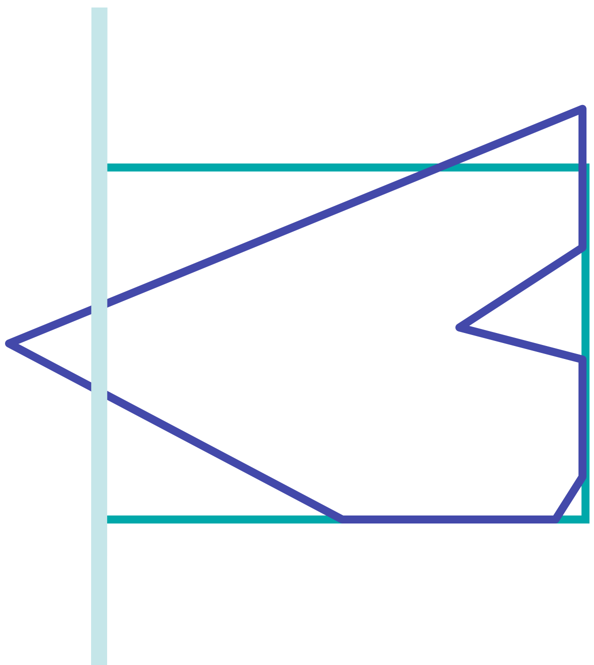


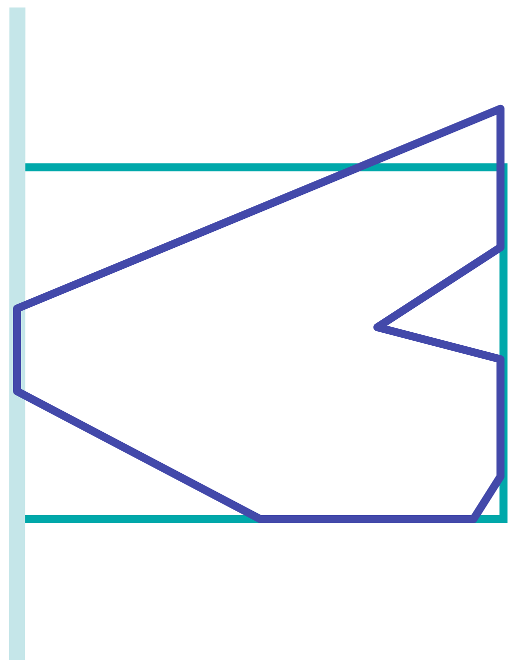


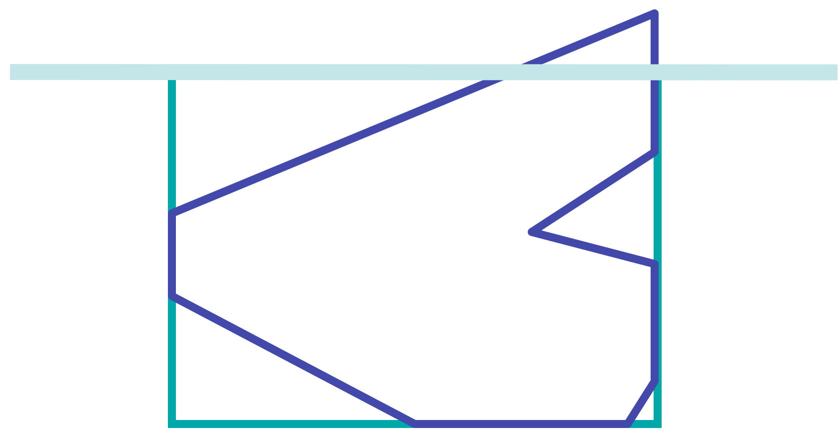


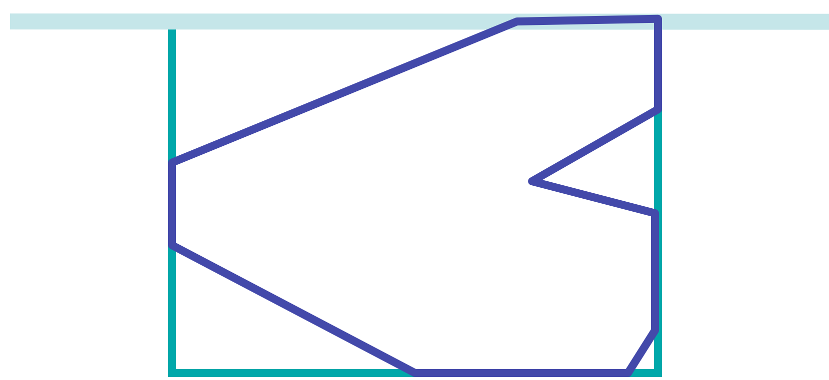


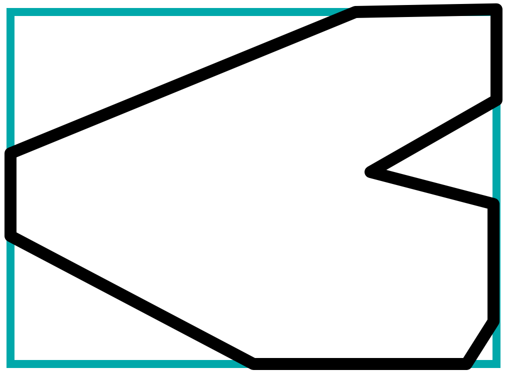






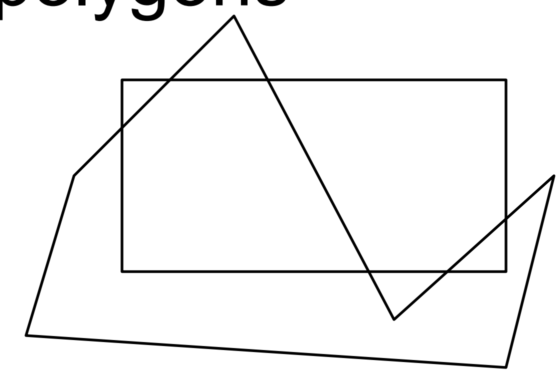




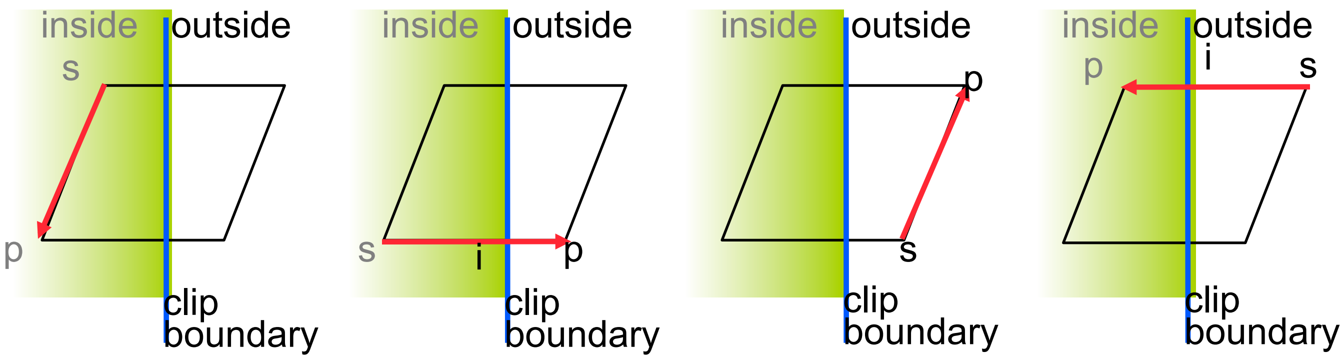


Sutherland-Hodgman Clipping basic routine

- 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
- This will not work for cases in which there are more than one output polygons



- Edge from s to p can be one of four cases (can extend to 3D):



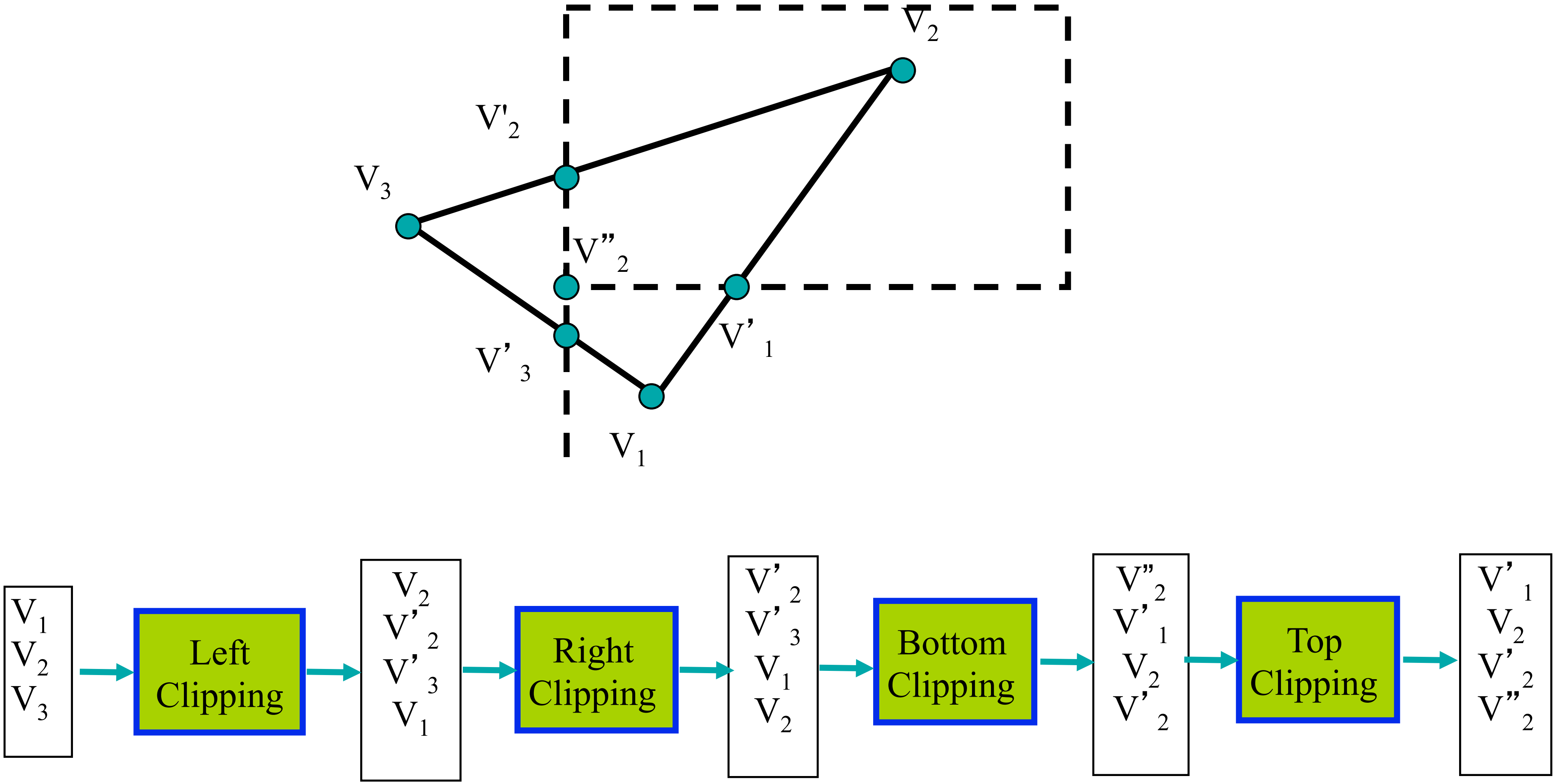
p added to output list

i added to output list

no output

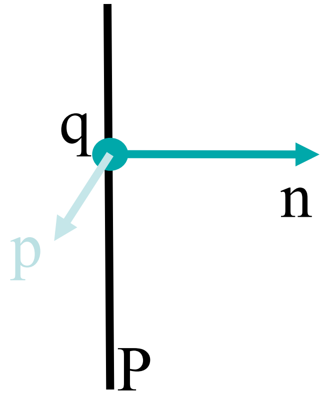
i and p added to output list

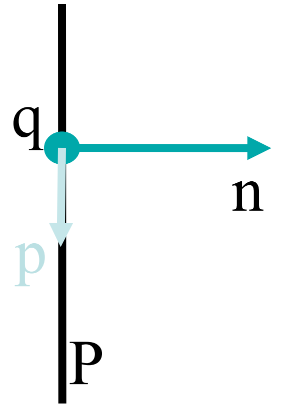
- s inside (clipping) plane and p inside plane
  - Add p to output
- s inside plane and p outside plane
  - Find intersection point i
  - Add i to output
- s outside plane and p outside plane
  - Add nothing
- s outside plane and p inside plane
  - Find intersection point i
  - Add i to output, followed by p

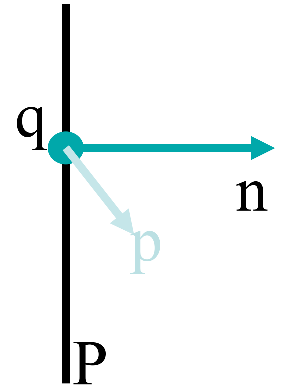

V2
V'2
V3
V"2
V'1
V'3
V1
V1
V2
V3
Left Clipping
V2
V'2
V'3
V1
Right Clipping
V'2
V'3
V1
V2
Bottom Clipping
V"2
V'1
V2
V'2
Top Clipping
V'1
V2
V'2
V"2

– Test to determine if a point p is “inside” a plane P, defined by a 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







Next: 3D Transforms