CS 543 - Computer Graphics: 2D Viewing, Part 2

by

Robert W. Lindeman

gogo@wpi.edu

(with help from Emmanuel Agu ;-)

Window-to-Viewport Mapping Applications

- Zooming in on a portion of object
- Tiling
  - W-to-V in loop
  - Multiple adjacent viewports
- Flipping drawings
- Stereo viewing
- Mapping different window and viewport aspect ratios (W / H)

Tiling: Example 3.2.4 of Hill

- Problem: want to tile dino.dat in 5x5 across screen

```
// set world window
gluOrtho2D( 0.0, 640.0, 0.0, 440.0 );

for( int i = 0; i < 5; i++ )  {
  for( int j = 0; j < 5; j++ )  {
    // .. now set viewport in a loop
    glViewport( i * 64, j * 44, 64, 44 );
    drawPolylineFile( "dino.dat" );
  }
}
```

Zooming

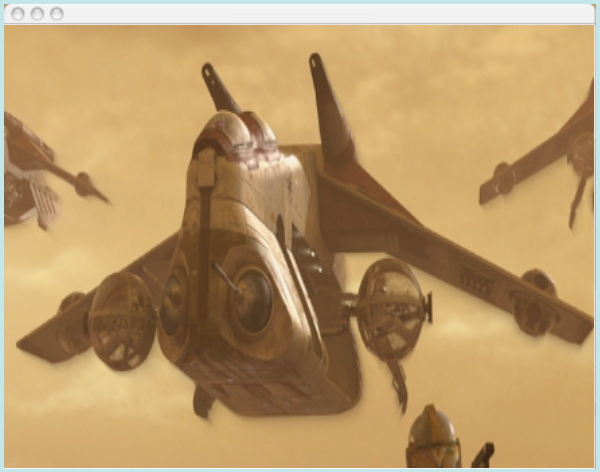
- Problem:
  - dino.dat is currently drawn in entire viewport
  - User wants to zoom into just the head
  - Specifies selection by clicking top-left and bottom-right corners
- Solution (pseudocode):
  1) Program accepts two mouse clicks as rectangle corners
  2) Calculate mapping A of current screen to all of dino.dat
  3) Use mapping A to refer screen rectangle to world
  4) Set window to smaller world rectangle
  5) Remaps small rectangle in world to screen viewport

Unmatched Aspect Ratios

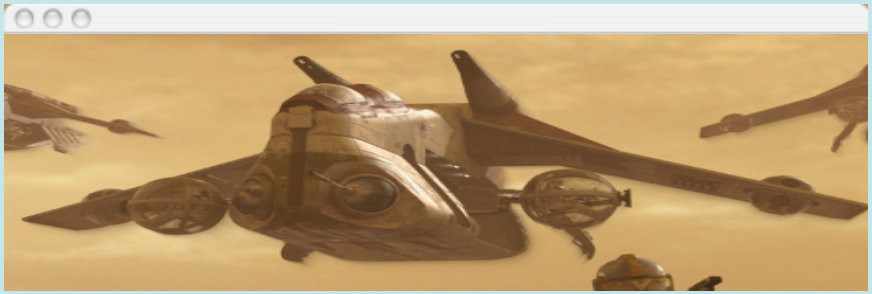
- Aspect ratio
  - **a** = Width/Height
  - W.a = (*W.r* - *W.l*) / (*W.t* - *W.b*)
  - V.a = (*V.r* - *V.l*) / (*V.t* - *V.b*)
- What if Window and Viewport have different *aspect ratios*?
  - $W.a \neq V.a$
  - Have you ever seen this problem before?

Unmatched Aspect Ratios (cont.)

- Two possible cases
  - *W.a* > *V.a*, *W.a* < *V.a*
- With standard mapping



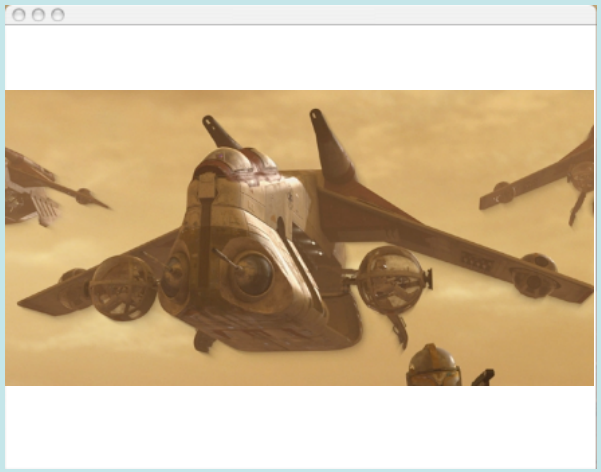
W.a > *V.a*



W.a < *V.a*

Unmatched Aspect Ratios (cont).

- *W.a* > *V.a*
  a) Match width, and leave top/bottom empty
  b) Match height, center, then crop left/right

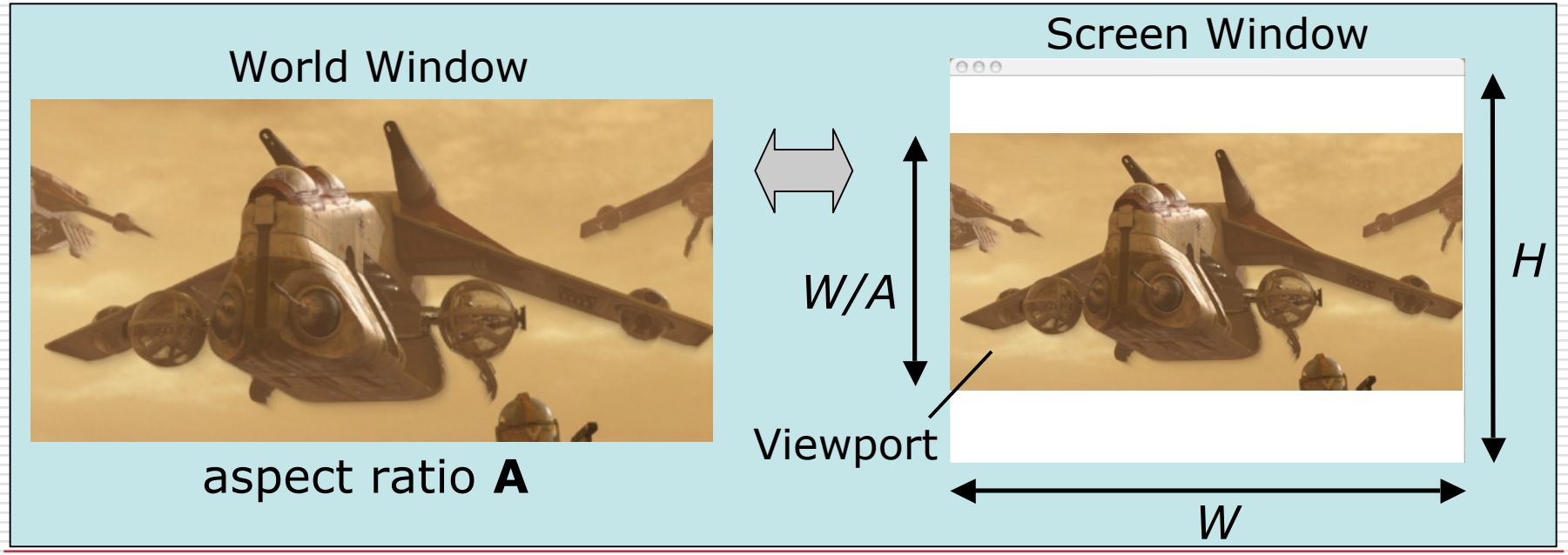


(a)

(b)

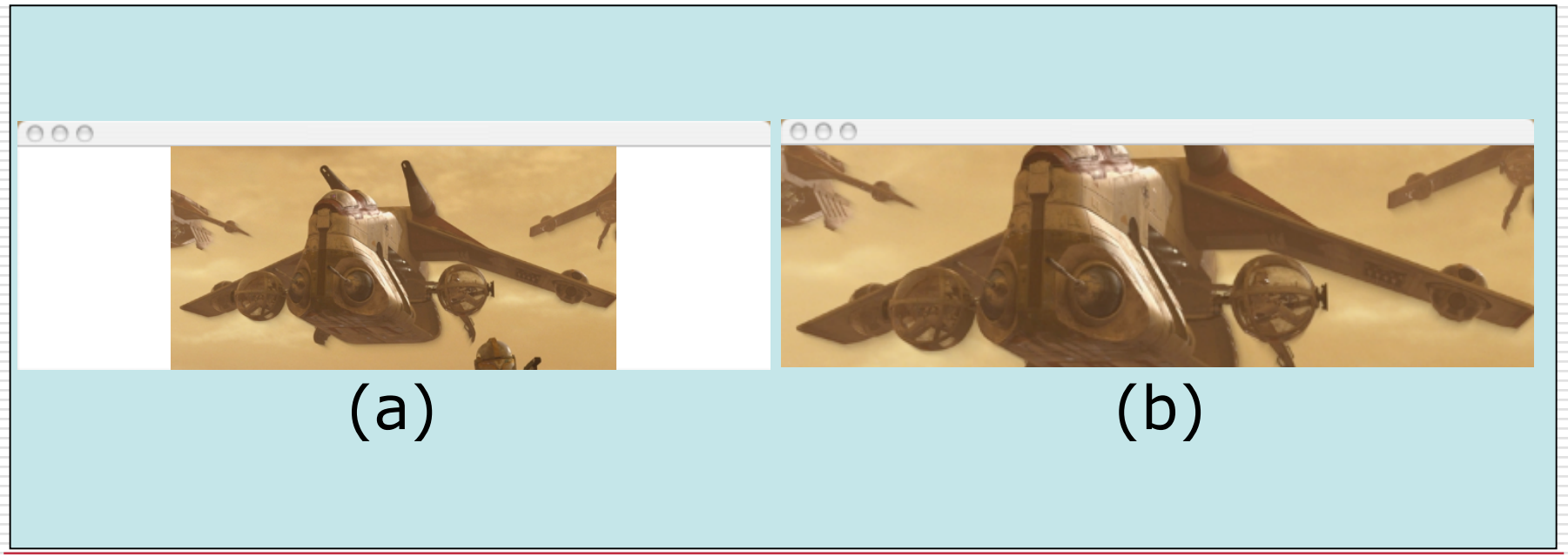
Unmatched Aspect Ratios: *W.a > V.a*

```
glOrtho( W.l, W.r, W.b, W.t );
A = ( W.r - W.l ) / ( W.t - W.b );
if( A > (W/H) )  {
  glViewport( 0, (H * 0.5) - ((W/A) * 0.5), W, W/A );
}
```



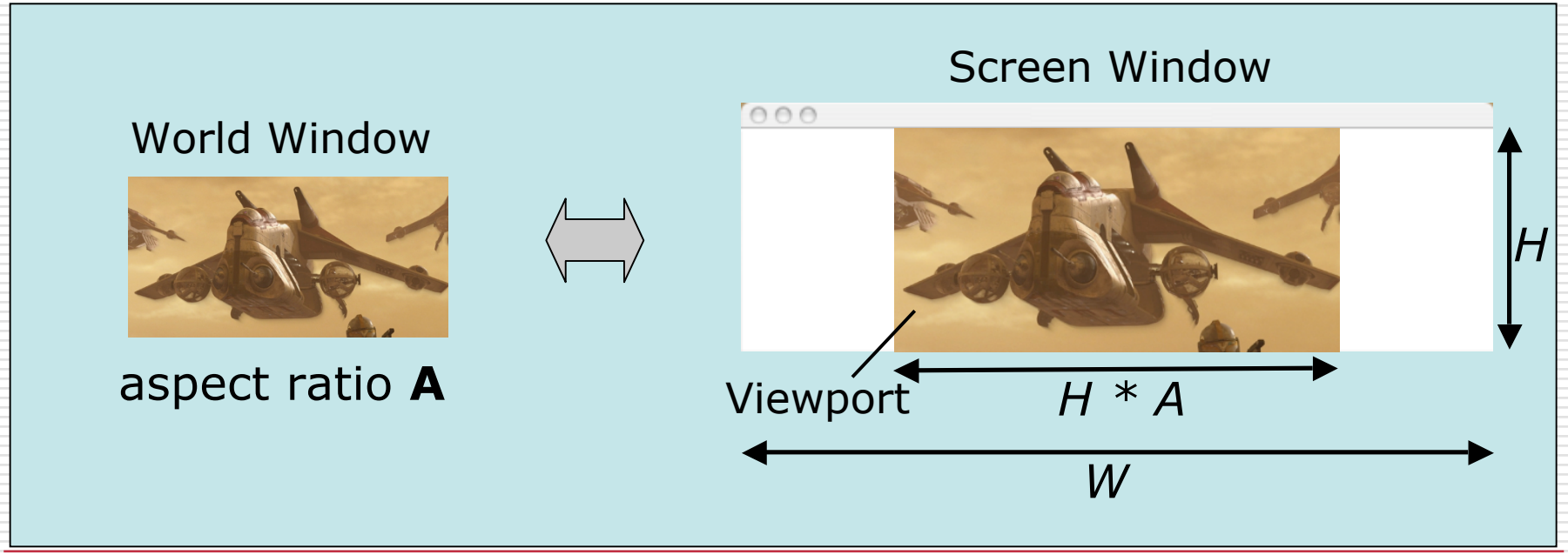
Unmatched Aspect Ratios (cont).

- *W.a* < *V.a*
  a) Match height, and leave sides empty
  b) Match width, center, then crop top/bottom



Unmatched Aspect Ratios: *W.a < V.a*

```
glOrtho( W.l, W.r, W.b, W.t );
A = ( W.r – W.l ) / ( W.t – W.b );
if( A < (W/H) )  {
  glViewport( (W * 0.5) – ((H*A) * 0.5), 0, H*A, H );
}
```



Reshape: Maintain Aspect Ratio

```
// glOrtho( l, r, b, t ); is done previously,
// probably in your draw function

void myReshape( double l, double r, double t, double b,
                double W, double H )  {
  A = (r–l) / (t–b);

  if( A > (W/H) )  {
    glViewport( 0, (H * 0.5) - ((W/A) * 0.5), W, W/A );
  } else  {
    if( A < (W/H) )  {
      glViewport( (W * 0.5) - ((H*A) * 0.5), 0, H*A, H );
    } else  {
      glViewport( 0, 0, W, H );  // equal aspect ratios
    }
  }
}
```

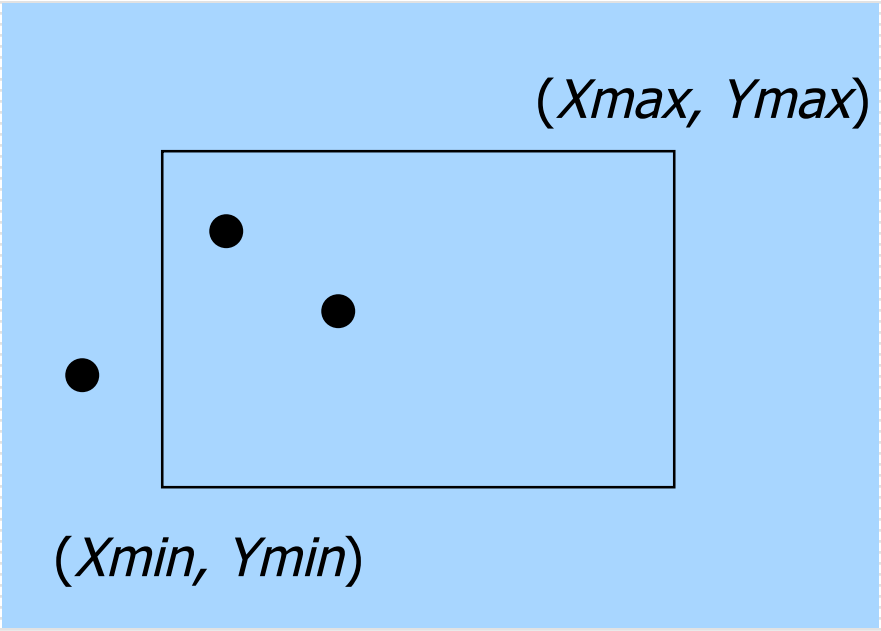
Cohen-Sutherland Clipping

- Frequently want to view only a portion of the picture
- For instance, in dino.dat, you can select to view/zoom in on only the dinosaur's head
- Clipping: eliminate portions not selected
- OpenGL automatically clips for you
- We want algorithm for clipping
- Classical algorithm: Cohen-Sutherland Clipping
- Picture could have 1,000s of segments
  - Efficiency is important

Clipping Points

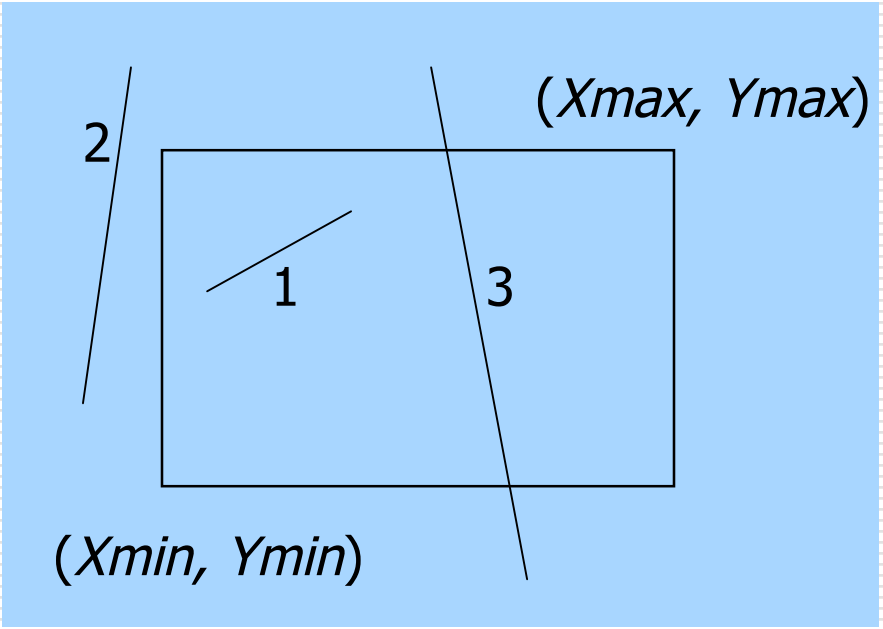
- Determine whether a point (x, y) is inside or outside of the world window

```
if( Xmin <= x <= Xmax ) and
  ( Ymin <= y <= Ymax )
then
  the point (x, y) is inside
else
  the point is outside
```



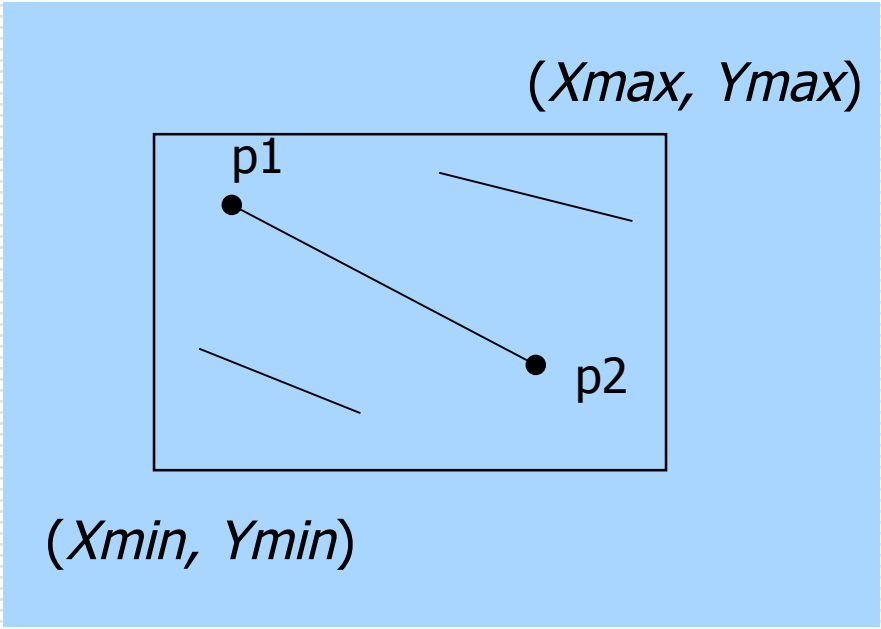
Clipping Lines: Three Cases

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



Clipping Lines: Trivially Accept

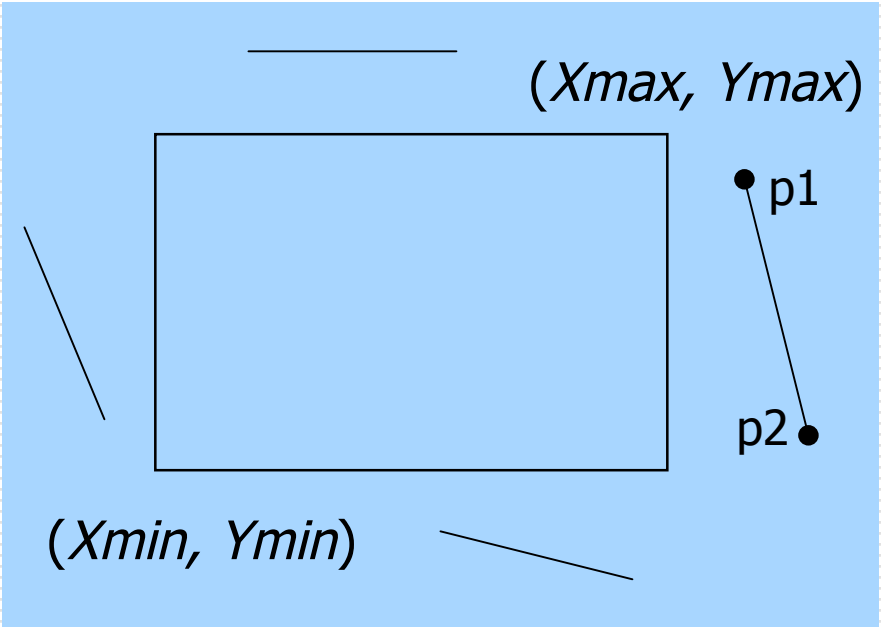
- Case 1
  - All of line inside
  - Test line endpoints
- Method
  - Simply compare x, y values of endpoints to rectangle extents




```
if( Xmin <= p1.x <= Xmax ) and
  ( Xmin <= p2.x <= Xmax ) and
  ( Ymin <= p1.y <= Ymax ) and
  ( Ymin <= p2.y <= Ymax )
then
  draw line completely
```

Clipping Lines: Trivially Reject

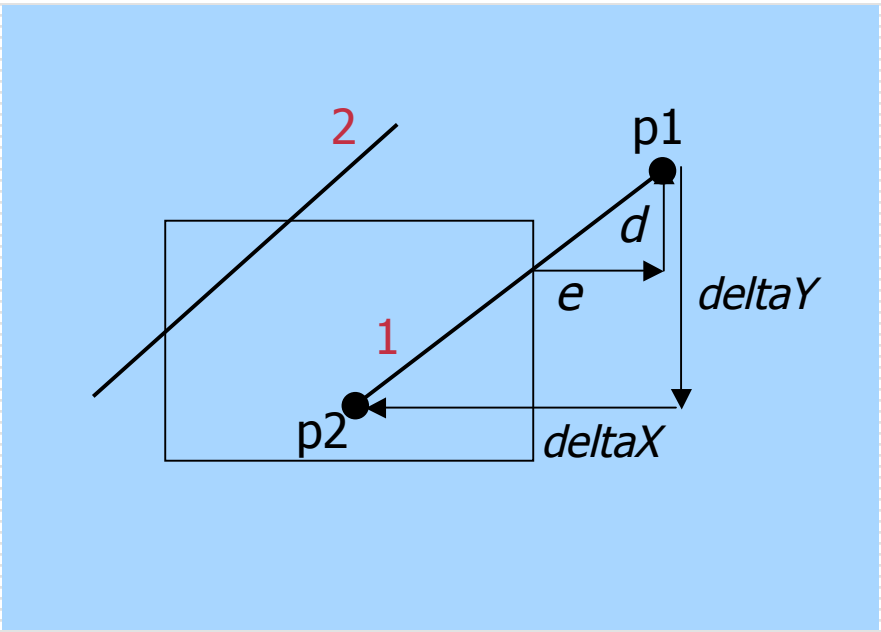
- Case 2
  - All of line outside
  - Test line endpoints
- Method
  - Simply compare x, y values of endpoints to rectangle extents




```
if( ( p1.x < Xmin ) and ( p2.x < Xmin ) ) or
  ( ( p1.x > Xmax ) and ( p2.x > Xmax ) ) or
  ( ( p1.y < Ymin ) and ( p2.y < Ymin ) ) or
  ( ( p1.y > Ymax ) and ( p2.y > Ymax ) ) or
then
  ignore line
```

Clipping Lines: Non-Trivial Case

- Case 3
  - Part in, part out
- Two variations
  1. One point in, other out
  2. Both points out, but line cuts through
- Need to find inside segments
- Use similar triangles to figure out length of inside segments



$$\frac{d}{deltaY} = \frac{e}{deltaX}$$

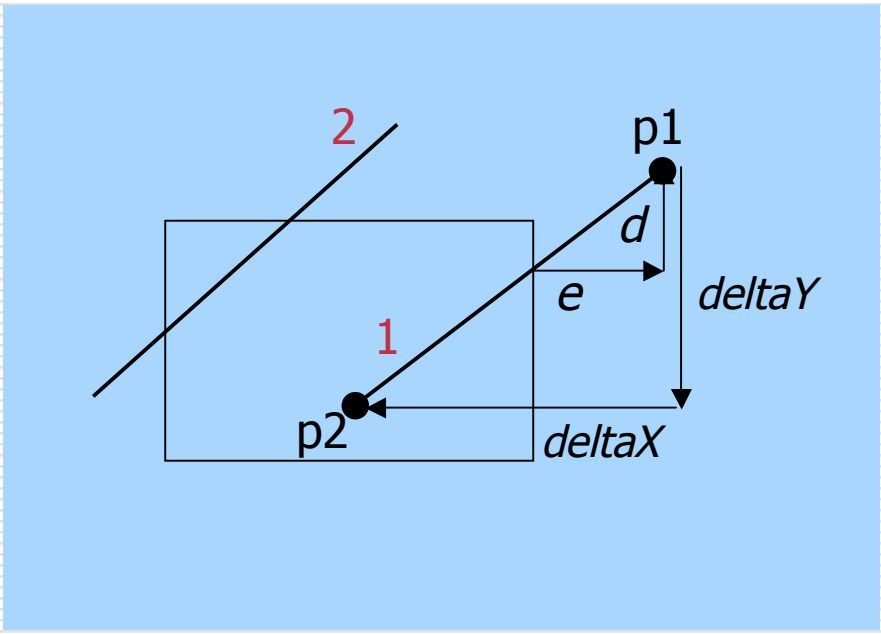
Clipping Lines: Non-Trivial Case

- If clipping window has `(l, r, b, t) =`

  `(30, 220, 50, 240),`

  what happens when the following lines are clipped?

```
(a)   p1 = ( 40, 140)
      p2 = (100, 200)
(b)   p1 = ( 20,  10)
      p2 = ( 20, 200)
(c)   p1 = (100, 180)
      p2 = (200, 250)
```



$$\frac{d}{deltaY} = \frac{e}{deltaX}$$

Cohen-Sutherland Pseudocode

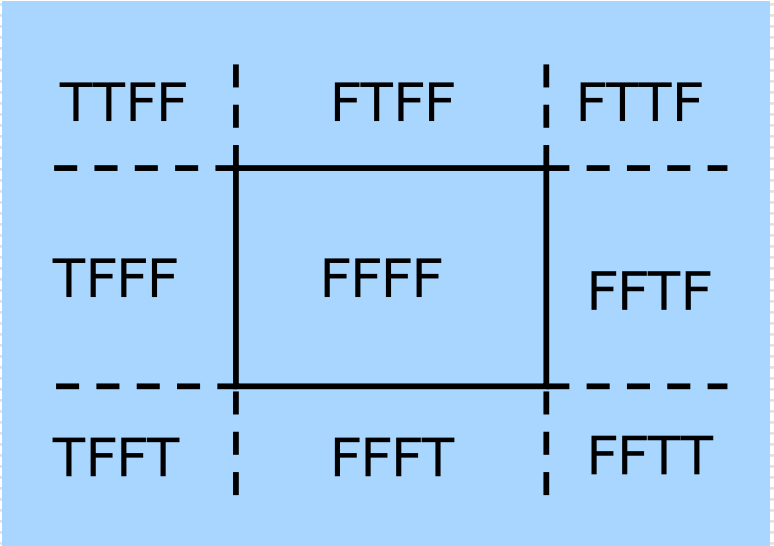
```
int clipSegment( Point2 p1, Point2 p2, RealRect W )  {
  do  {
    // If whole line survives
    if( trivial accept )  { return 1; }
    // If no portion survives
    if( trivial reject )  { return 0; }
    // now clip
    if( p1 is outside )  {
      //  find surviving segment
      if( p1 < W.l )       clip at W.l
      else if( p1 > W.r ) clip to W.r
      else if( p1 < W.b ) clip to W.b
      else if( p1 > W.t ) clip to W.t
    }
(continued on next slide)
```

Cohen-Sutherland Pseudocode (cont.)

```
    else  { // p2 is outside
            // find surviving segment
      if( p2 < W.l )       clip at W.l
      else if( p2 > W.r ) clip to W.r
      else if( p2 < W.b ) clip to W.b
      else if( p2 > W.t ) clip to W.t
    }
  } while( 1 );
}
```

Cohen-Sutherland Implementation

- Need quick, efficient comparisons to get accepts, rejects, clips
- Can use C/C++ bit operations
- Break space into 4-bit words
  - Trivial accept: both points FFFF
  - Trivial reject: T in same position
  - Everything else: clip
- Systematically clips against four edges
- Important: read Hill 3.3



References

- Hill: 3.1 – 3.3, 3.8