CS 543: Computer Graphics
Lecture 2 (Part II): Tiling, Zooming and 2D Clipping

Emmanuel Agu

Applications of W-to-V Mapping

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

Tiling: Example 3.2.4 of Hill (pg. 100)

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

```
// set world window
gluOrtho2D(0, 640.0, 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

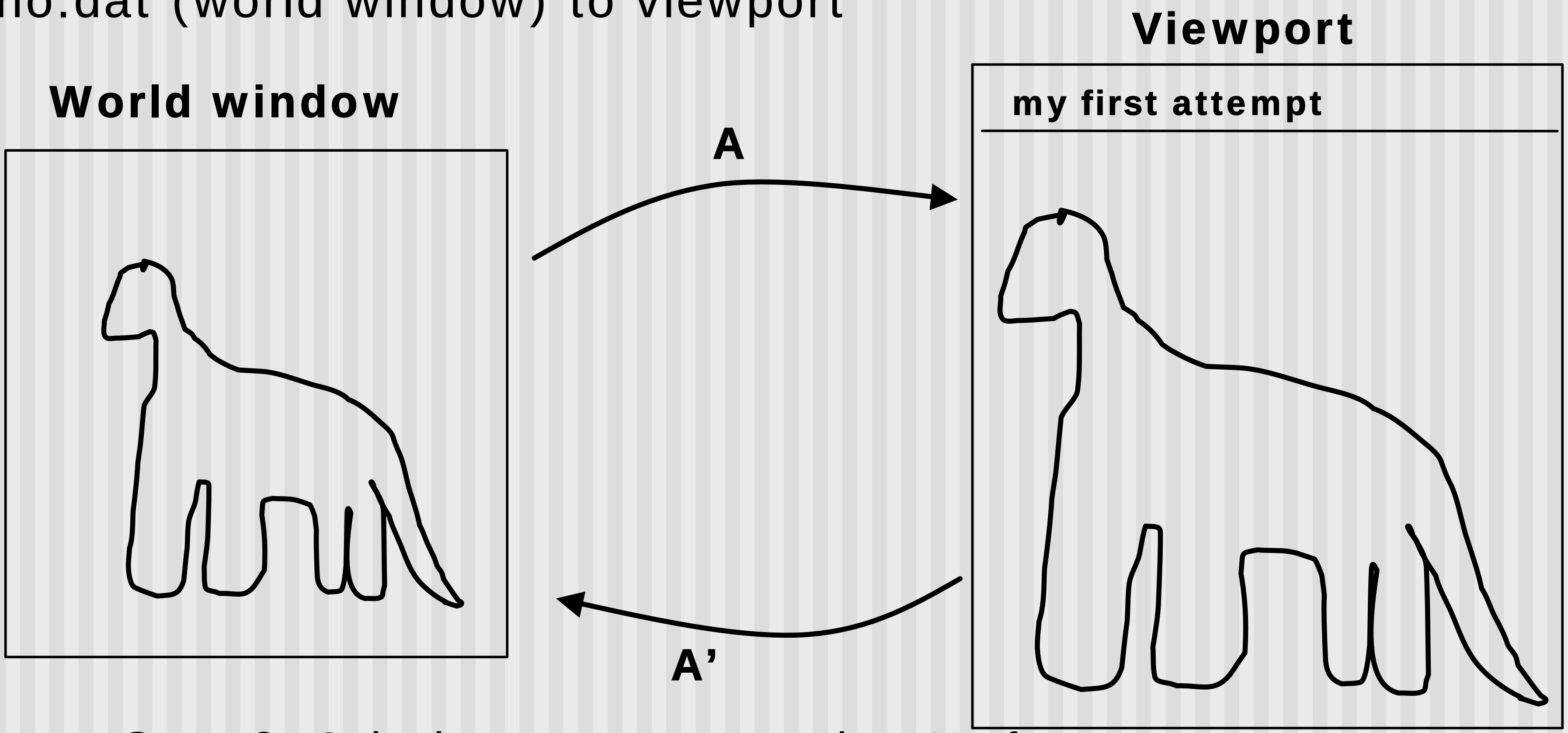
- Problem:
  - dino.dat is currently drawn on entire screen.
  - User wants to zoom into just the head
  - Specifies selection by clicking top-left and bottom-right corners

Example mapping A $\longrightarrow$

$$Sx = Ax - (A(W.L) - V.L)$$
$$Sy = By - (B(W.B) - V.B)$$

Zooming

Step 1 : Calculate mapping A, that maps dino.dat (world window) to viewport



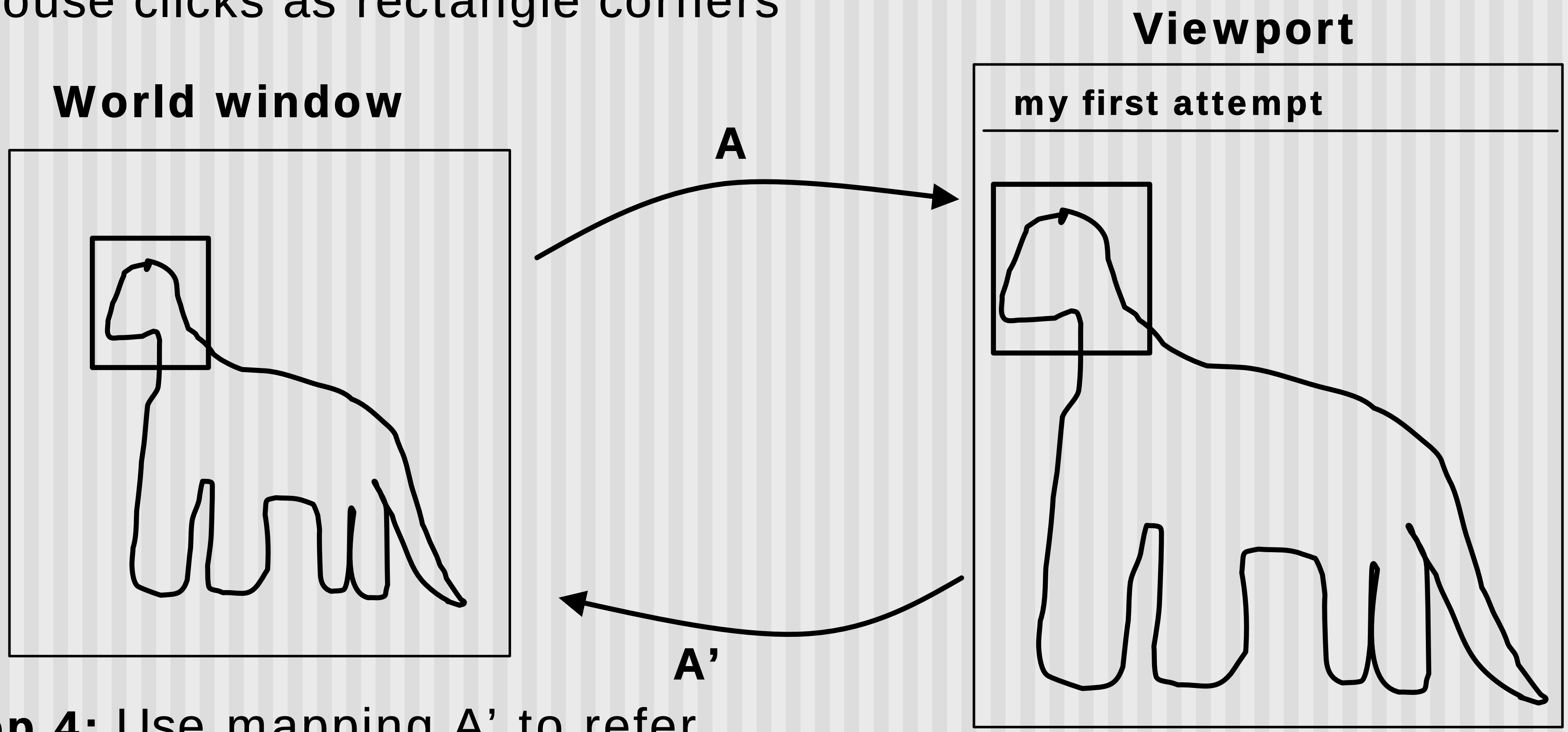
Step 2: Calculate reverse mapping A’ of current viewport back to the entire world window (dino.dat)

Example mapping A $\longrightarrow$

$$Sx = Ax - (A(W.L) - V.L)$$
$$Sy = By - (B(W.B) - V.B)$$

Zooming

Step 3 : Program accepts two mouse clicks as rectangle corners



Step 4: Use mapping A’ to refer selected screen rectangle to world

Step 5: Call gluOrtho2D on smaller rectangle

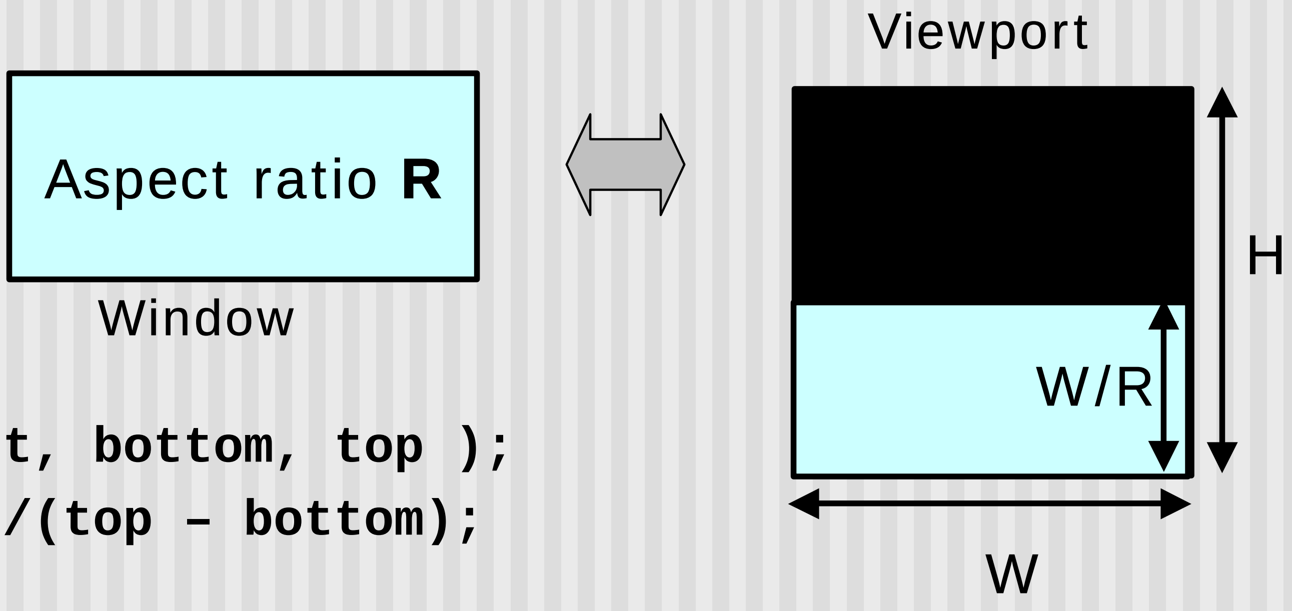
Zooming

- Zooming (pseudocode):

```
1. Calculate mapping A of from world (entire dino.dat)
   to current viewport
2. Derive reverse mapping A’ from viewport to world
3. Program accepts two mouse clicks as rectangle
   corners
4. Use mapping A’ to refer screen rectangle to world
5. Sets world to smaller world rectangle (gluOrtho2D on
   selected rectangle in world coordinates)
6. Remaps small rectangle in world to screen viewport
```

What if Window and Viewport have different Aspect Ratios?

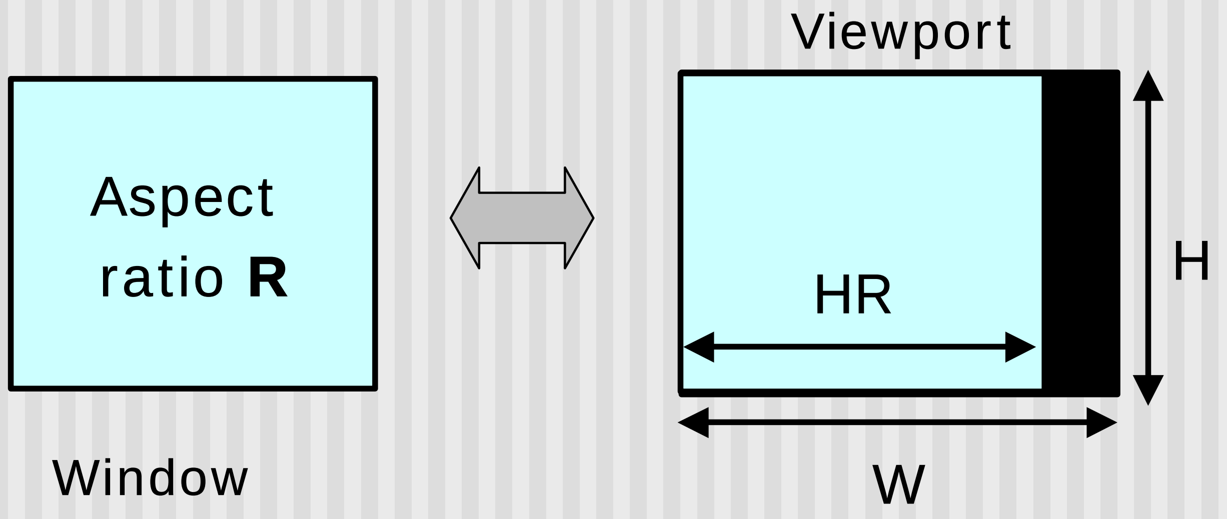
- Aspect ratio: is ratio **R** = Width/Height
- What if window and viewport have different aspect ratios?
- If different, two possible cases:
  - **Case A (R > W / H):** map a wide window to a tall viewport?



```
glOrtho(left, right, bottom, top );
R = (right – left)/(top – bottom);
If(R > W/H)
             glViewport(0, 0, W, W/R);
```

What if Window and Viewport have different Aspect Ratios?

- **Case B (R < W / H):** map a tall window to a wide viewport?



```
glOrtho(left, right, bottom, top );
R = (right – left)/(top – bottom);
If(R < W/H)
              glViewport(0, 0, H*R, H);
```

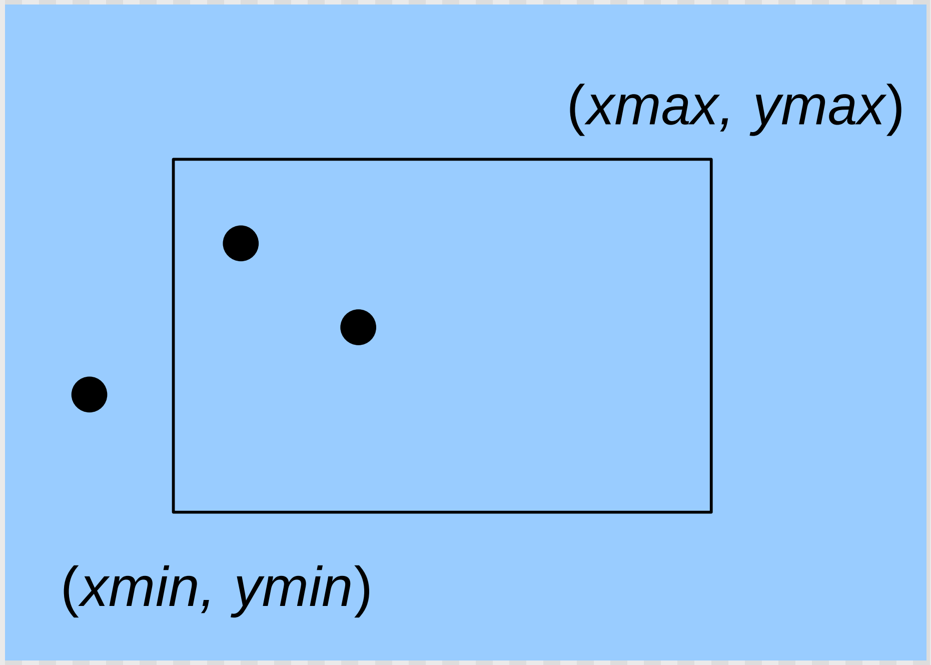
reshape( ) function that maintains aspect ratio

```
// glOrtho(left, right, bottom, top )is done previously,
// probably in your draw function
// function assumes variables left, right, top and bottom
// are declared and updated globally

void myReshape(double W, double H ){
   R = (right – left)/(top – bottom);

   if(R > W/H)
        glViewport(0, 0, W, W/R);
   else if(R < W/H)
        glViewport(0, 0, H*R, 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 has 1000s of segments : efficiency is important

Clipping Points

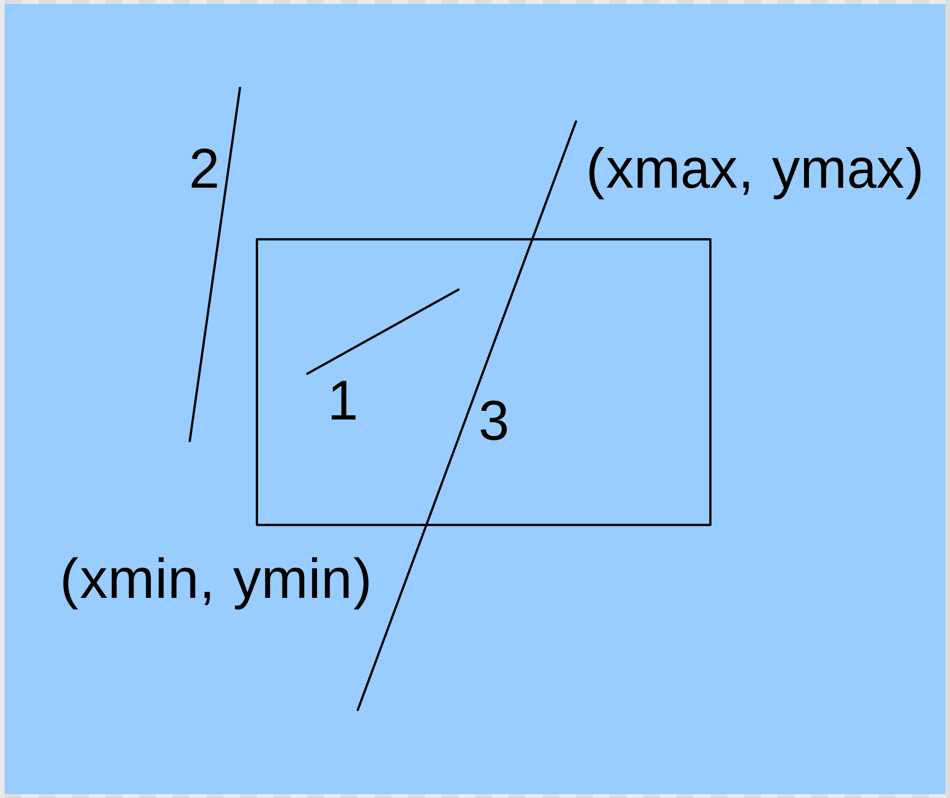


- Determine whether a point (x,y) is inside or outside of the world window?

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

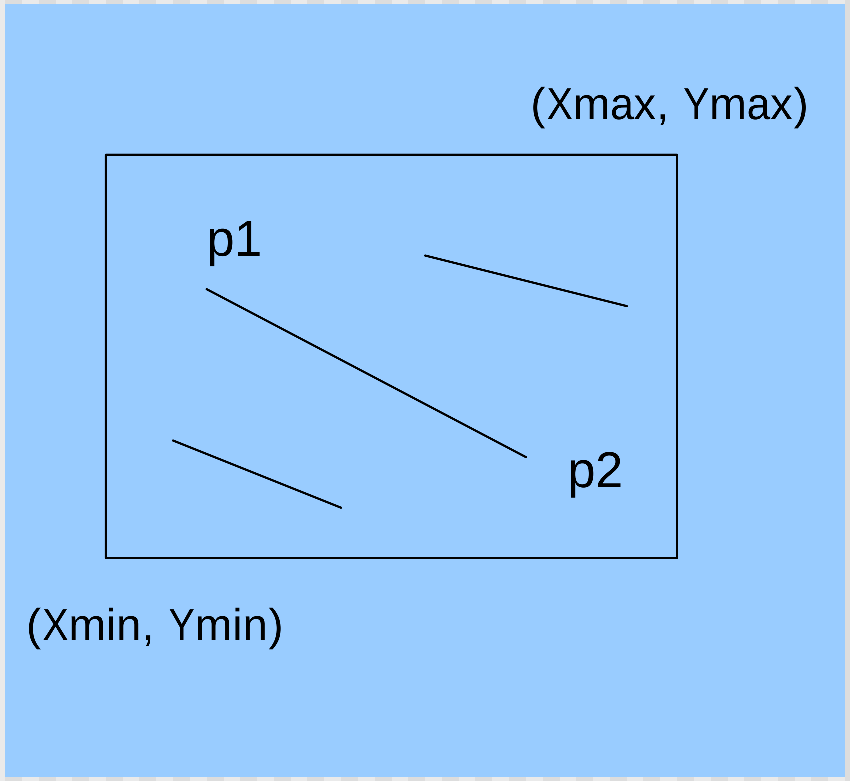
then the point (x,y) is inside
else the point is outside

Clipping Lines



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

Clipping Lines: Trivial Accept

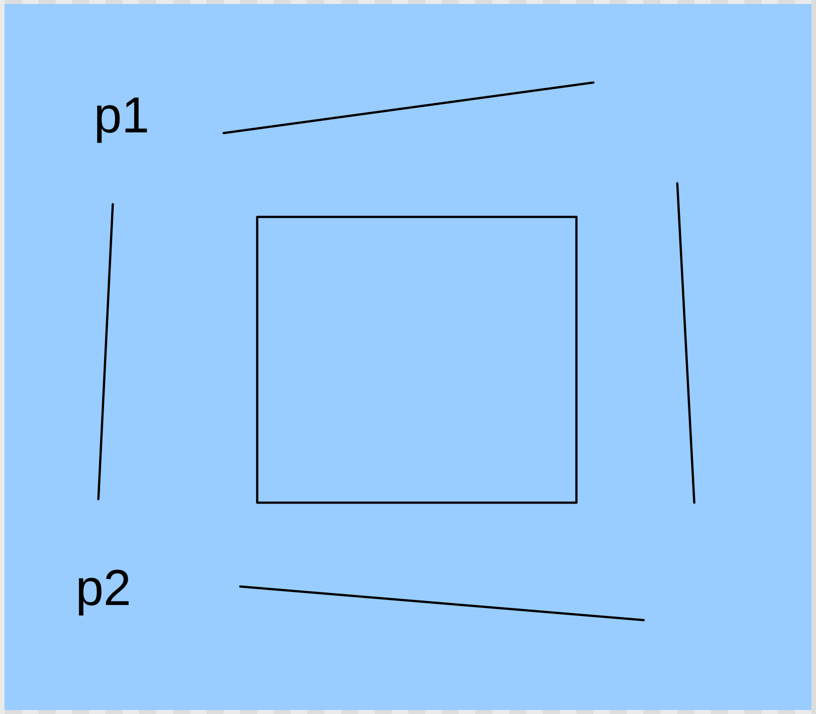


- Case 1: All of line in
- Test line endpoints:

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

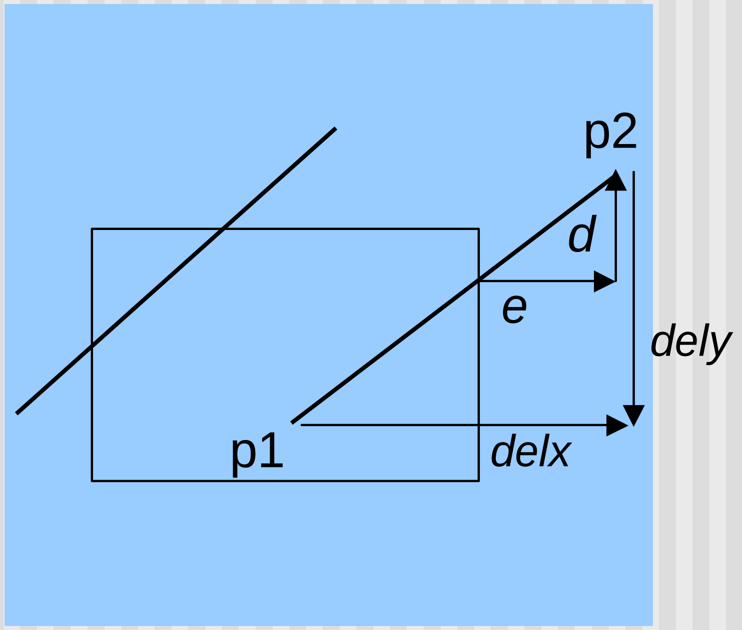


- Case 2: All of line out
- Test line endpoints:

  - *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

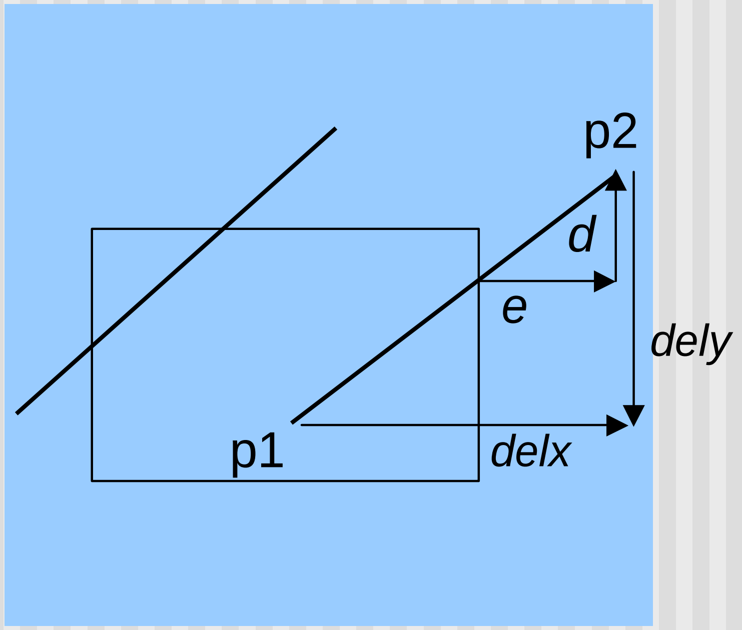
Clipping Lines: Non-Trivial Cases



- 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

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

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)

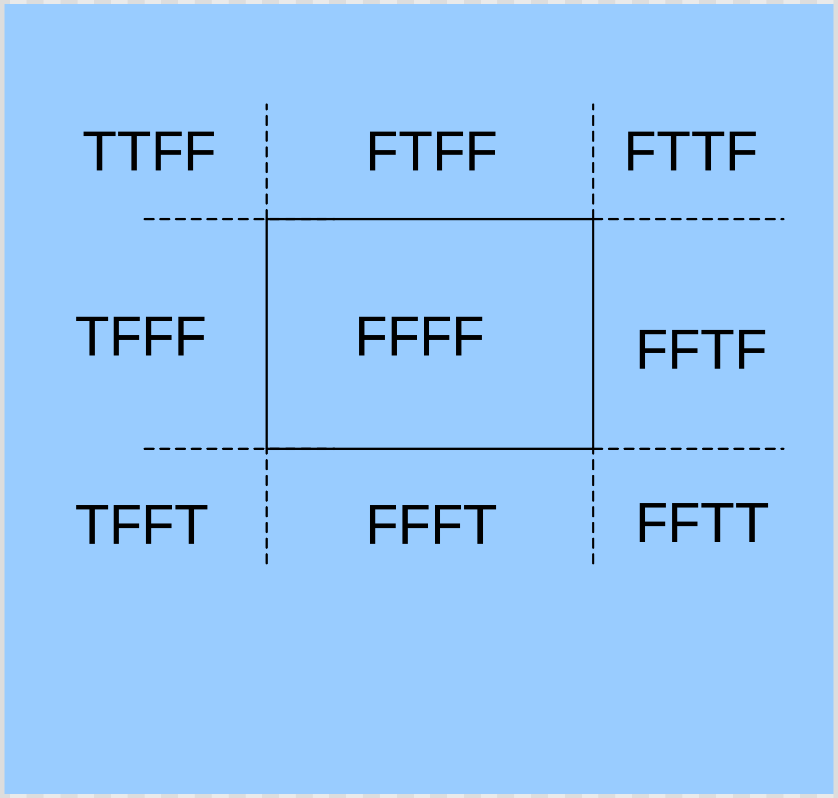
Cohen-Sutherland pseudocode (fig. 3.21)

```
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 (fig. 3.23)

```
        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);
  }
```

Cohen-Sutherland Implementation



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

Remember to read

- Section 3.2.2 on pg. 92 of Hill
- Hill 3.3

References

- Hill, 3.1 – 3.3, 3.8