CS 4731: Computer Graphics
Lecture 4: 2D Graphic Systems

Emmanuel Agu

Announcements

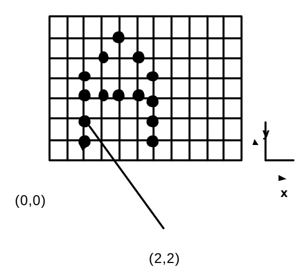
- Room: if few people drop class, I will request room change
- Project 1 should work on any of the CCC unix/Linux machines
- Simply let TA know which machine you worked on
- myWPI:
  - TA's: Jim Nichols and Paolo Piselli
  - SA: Brian Corcoran
  - Treat what they post as "official".

2D Graphics: Coordinate Systems

- Screen coordinate system
- World coordinate system
- World window
- Viewport
- Window to Viewport mapping

Screen Coordinate System

- Screen: 2D coordinate system (WxH)
- 2D Regular Cartesian Grid
- Origin (0,0) at lower left corner (OpenGL convention)
- Horizontal axis – x
- Vertical axis – y
- Pixels: grid intersections



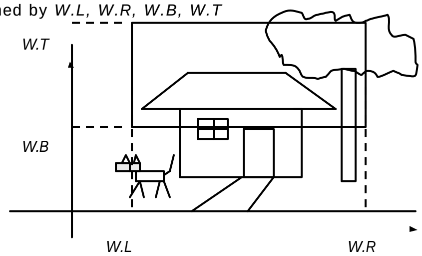
Screen Coordinate System

World Coordinate System

- Problems with drawing in screen coordinates:
  - Inflexible
  - Difficult to use
  - One mapping: not application specific
- World Coordinate system: application-specific
- Example: drawing dimensions may be in meters, km, feet, etc.

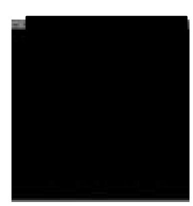
Definition: World Window

- World Window: rectangular region of drawing (in world coordinates) to be drawn
- Defined by *W.L, W.R, W.B, W.T*



Definition: Viewport

- Rectangular region in the screen used to display drawing
- Defined in screen coordinate system



Window to Viewport Mapping

- Would like to:
  - Specify drawing in world coordinates
  - Display in screen coordinates
- Need some sort of mapping
- Called Window-to-viewport mapping
- Basic W-to-V mapping steps:
  - Define a world window
  - Define a viewport
  - Compute a mapping from window to viewport

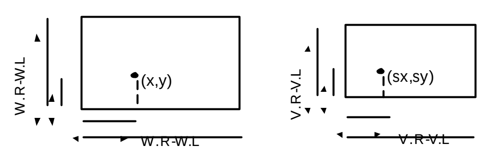
Window to Viewport Mapping (OpenGL Way)

- Define window (world coordinates):
  - **`gluOrtho2D(left, right, bottom, top)`**
  - **Side note:** gluOrtho2D is member of glu library
- Define Viewport (screen coordinates):
  `glViewport(left, bottom, right-left, top-bottom)`
- All subsequent drawings are automatically mapped
- Do mapping before any drawing **`(glBegin( ), glEnd( ))`**
- Two more calls you will encounter to set up matrices:
  - **`glMatrixMode(GL_PROJECTION)`**
  - **`glLoadIdentity( )`**
- Type in as above for now, will explain later
- Ref: Hill Practice exercise 3.2.1, pg 86

Window to Viewport Mapping (Our Way)

- How is window-to-viewport mapping done?
- Trigonometry: derive Window-to-Viewport mapping
- Basic principles:
  - Calculate ratio: proportional mapping ratio (NO distortion)
  - Account for offsets in window and viewport origins
- You are given:
  - World Window: W.R, W.L, W.T, W.B
  - Viewport: V.L, V.R, V.B, V.T
  - A point (x,y) in the world
- Required: Calculate corresponding point (s.x, s.y) in screen coordinates

Window to Viewport Mapping (Our Way)



$$\frac{(x - W.L)}{W.R - W.L} = \frac{Sx - V.L}{V.R - V.L}$$

$$\frac{(y - W.B)}{W.T - W.B} = \frac{Sy - V.B}{V.T - V.B}$$

Window to Viewport Mapping (Our Way)

Solve for Sx, Sy in terms of x, y:

$$\frac{(x - W.L)}{W.R - W.L} = \frac{Sx - V.L}{V.R - V.L}$$

$$\frac{(y - W.B)}{W.T - W.B} = \frac{Sy - V.B}{V.T - V.B}$$

$$Sx = \left(\frac{V.R - V.L}{W.R - W.L}\right)x - \left(\frac{V.R - V.L}{W.R - W.L}W.L - V.L\right)$$

$$Sy = \left(\frac{V.T - V.B}{W.T - W.B}\right)y - \left(\frac{V.T - V.B}{W.T - W.B}W.B - V.B\right)$$

Window to Viewport Mapping (Our Way)

Solve, given the formulas:

$$Sx = \frac{V.R - V.L}{W.R - W.L}x - \left(\frac{V.R - V.L}{W.R - W.L}W.L - V.L\right)$$

$$Sy = \frac{V.T - V.B}{W.T - W.B}y - \left(\frac{V.T - V.B}{W.T - W.B}W.B - V.B\right)$$

What is (Sx,Sy) for point (3.4,1.2) in world coordinates if:

$$W = (W.L, W.R, W.B, W.T) = (0,4,0,2)$$

$$V = (V.L, V.R, V.B, V.T) = (60,380,80,240)$$

Window to Viewport Mapping (Our Way)

Solution:

$$Sx = \frac{V.R - V.L}{W.R - W.L}x - \left(\frac{V.R - V.L}{W.R - W.L}W.L - V.L\right)$$

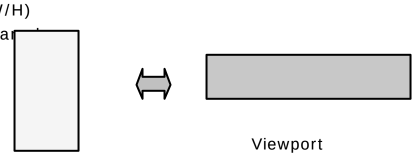
$$Sy = \frac{V.T - V.B}{W.T - W.B}y - \left(\frac{V.T - V.B}{W.T - W.B}W.B - V.B\right)$$

$Sx = 80x + 60 = 332$ $\qquad$ $Sy = 80x + 80 = 176$

Hence, point (3.4,1.2) in world = point (332,176) on screen

More 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)
- Exam



Viewport

Window

Important: Please read on your own, section 3.2.2 on pg. 92 of Hill

Tiling: Example 3.2.4 of Hill (pg. 88)

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

```
gluOrtho2D(0, 640.0, 0, 440.0);
for(int i=0;i < 5;i++)
{
   for(int j = 0;j < 5; j++)
   {
       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
- Solution:
  - 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: Sets world to smaller world rectangle
  - 5: Remaps small rectangle in world to screen viewport

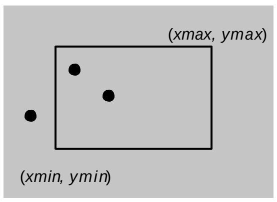
Using mouse to select screen rectangle for zooming (Example 2.4.2, pg 64) for zooming

```
void myMouse(int button, int state, int x, int y)
{
   static GLint corner[2];
   static int numCorners = 0;   // initialize
   if(button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
   {
        corner[numCorners].x = x;
        corner[numCorners].y = y;
        numCorners++;
        if(numCorners == 2)
        {
               glViewport(corner[0], corner[1]);
               numCorners = 0;
        }
   }
}
```

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

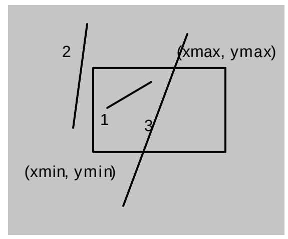


- Determine whether a point (x,y) is inside or outside of the world window?

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

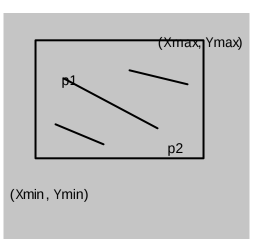
then the point (x,y) is inside
else the point is outside

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

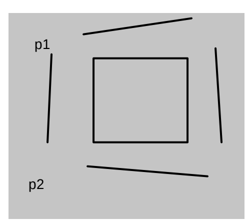


- Case 1: All of line in
- Test line endpoints:

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

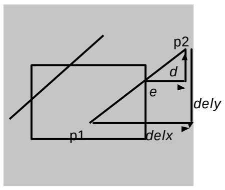


- Case 2: All of line out
- Test line endpoints:

  - *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



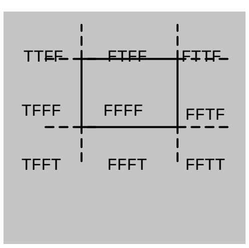
$$\frac{d}{dely} = \frac{e}{delx}$$

- 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

Cohen-Sutherland pseudocode (fig. 3.23)

```
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
      else(p2 is outside)
             // find surviving segment
  }while(1)
}
```

Cohen-Sutherland Implementation



- Breaks space into 4-bit words
  - Trivial accept: both FFFF
  - Trivial reject: T in same position
  - Chop everything else
- Systematically chops against four edges
- Can use C/C++ bit operations
- Important: read Hill 3.3

Parametric Equations

- Implicit form

$$F(x, y) = 0$$

- Parametric forms:
  - points specified based on single parameter value
  - Typical parameter: time t

$$P(t) = P_0 + (P_1 - P_0) * t \qquad 0 \le t \le 1$$

- Some algorithms work in parametric form
  - Clipping: exclude line segment ranges
  - Animation: Interpolate between endpoints by varying t

References

- Hill, 3.1 – 3.3, 3.8