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:
  - 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)

\[
\begin{align*}
\text{\(x\)} & = \frac{W.R - W.L}{W.R - W.L} \\
\text{\(y\)} & = \frac{W.T - W.B}{W.T - W.B}
\end{align*}
\]

\[
\begin{align*}
\text{\(s.x\)} & = \frac{V.R - V.L}{V.R - V.L} \\
\text{\(s.y\)} & = \frac{V.T - V.B}{V.T - V.B}
\end{align*}
\]
Window to Viewport Mapping (Our Way)

Solve for $S_x, S_y$ in terms of $x, y$:

$S_x = \frac{(x - W \cdot L)}{W \cdot R - W \cdot L} \cdot \frac{S_x - V \cdot L}{V \cdot R - V \cdot L}$

$S_y = \frac{(y - W \cdot B)}{W \cdot T - W \cdot B} \cdot \frac{S_y - V \cdot B}{V \cdot T - V \cdot B}$

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

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

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 ($W/H$)

Example:

Window

Viewport

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

Solution:

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

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

$S_x = 80x + 60 = 332$

$S_y = 80x + 80 = 176$

Hence, point (3,4,1.2) in world = point (332,176) on screen
Tiling: Example 3.2.4 of Hill (pg. 88)

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

```c
  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
- 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

```c
  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 \((\text{xmin} \leq x \leq \text{xmax})\) and \((\text{ymin} \leq y \leq \text{ymax})\)
  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:
  \(\text{xmin} \leq P1.x, P2.x \leq \text{xmax} \quad \text{and} \quad \text{ymin} \leq P1.y, P2.y \leq \text{ymax}\)
- 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:
  \(P1.x, P2.x < \text{xmin} \quad \text{OR} \quad P1.x, P2.x > \text{xmax}\)
  \(P1.y, P2.y < \text{ymin} \quad \text{OR} \quad P1.y, P2.y > \text{ymax}\)
- 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

\[
\frac{d}{dely} = \frac{e}{delx}
\]

Cohen-Sutherland pseudocode (fig. 3.23)

```cpp
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 if(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 \( t \)
    \[ P(t) = P_1 + (P_2 - P_1) \cdot t \quad 0 \leq t \leq 1 \]
  - 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