# CS 543 - Computer Graphics: 2D Viewing, Part 1

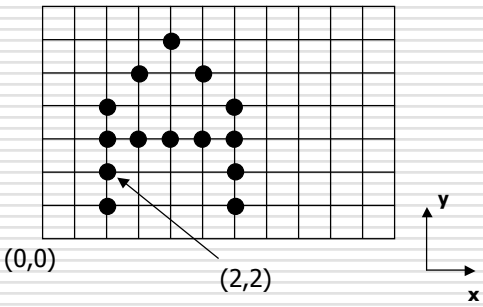
by
Robert W. Lindeman
gogo@wpi.edu
(with help from Emmanuel Agu ;-)

## 2D Graphics: Coordinate Systems

- Screen coordinate system
  - Pixels
- World coordinate system
  - Scene space
- World window
  - View of the scene
- Viewport
  - Drawing of the view of the scene
- Window-to-viewport mapping

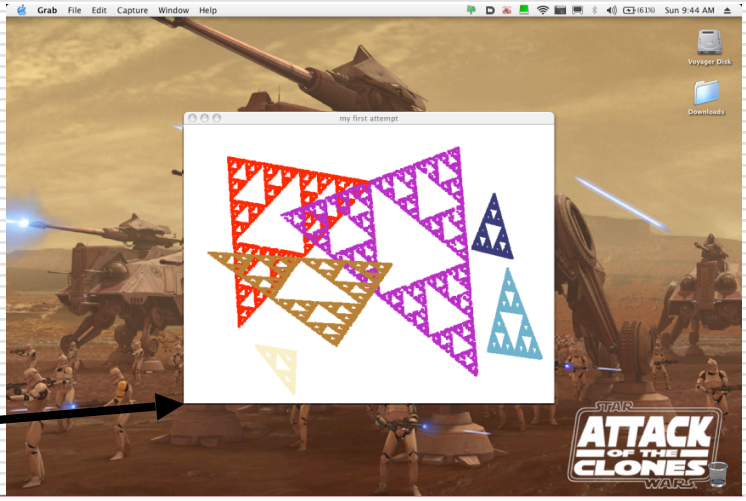
# Screen Coordinate System

- Screen
  - 2D coordinate system (W x H)
- 2D regular Cartesian grid
- Origin (0, 0) at lower left corner
  - OpenGL convention
- X: Horizontal axis
- Y: Vertical axis
- Pixels
  - Grid intersections






# Screen Coordinate System




(200, 100)

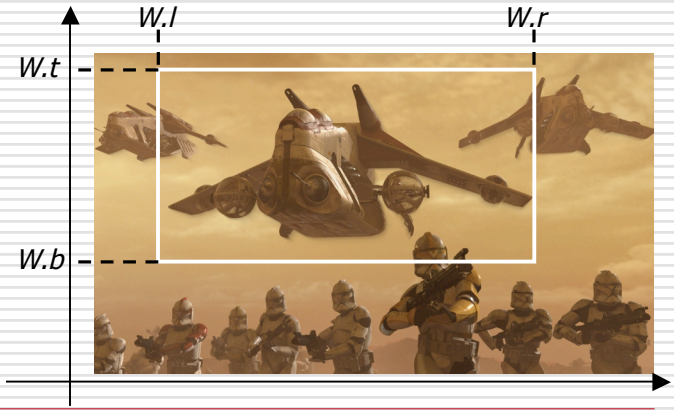
# World Coordinate System

- Problems with drawing in screen coordinates
  - Inflexible
    - How do you zoom?
  - Difficult to use
    - Have to redraw in "new" space
  - A single mapping
    - Not application specific
- World coordinate system
  - Application specific
- Example
  - Drawing dimensions may be in meters, km, feet, *etc.*



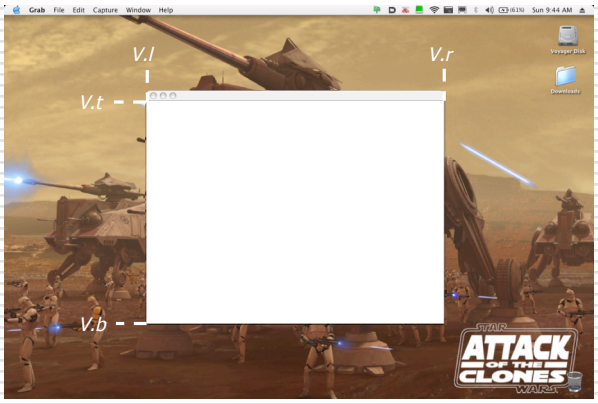
# Definition of a World Window

- World window
  - Rectangular region of drawing (in world coordinates) to be drawn
- Defined by
  - *W.l*
  - *W.r*
  - *W.b*
  - *W.t*



# Definition of a Viewport

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






# Window-to-Viewport Mapping

## 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 in OpenGL

- Define window in *world coordinates*

```
gluOrtho2D( w.l, w.r, w.b, w.t );
```

  - **Side note:** gluOrtho2D is member of glu library
- Define viewport in *screen coordinates*

```
glViewport( v.l, v.b, v.r - v.l, v.t - v.b );
```

  - Do mapping before any drawing (`glBegin( ), glEnd( )`)
  - All subsequent drawings are automatically mapped
- Set up matrices:

```
glMatrixMode( GL_PROJECTION );
glLoadIdentity( );
```

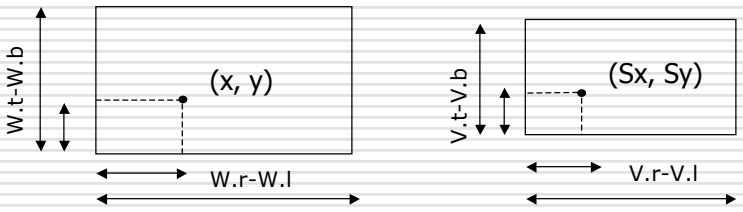
  - We will explain these later (See Hill Fig. 3.4, p. 95, & Practice 3.2.1, p. 98)

# Window-to-Viewport Mapping - Under the Hood

- How is window-to-viewport mapping done?
- Trigonometry
  - Derive Window-to-Viewport mapping
- Basic principles
  - Calculate *ratio*: proportional mapping ratio
  - Account for offsets in window and viewport origins
- You are given
  - World Window: *W.l, W.r, W.b, W.t*
  - Viewport: *V.l, V.r, V.b, V.t*
  - A point (*x, y*) in the world
- Required
  - Calculate corr. point (*Sx*, *Sy*) in screen coordinates



# Window-to-Viewport Mapping Under the Hood (cont.)




$$\frac{(x - W.l)}{W.r - W.l} = \frac{Sx - V.l}{V.r - V.l}$$

$$\frac{(y - W.b)}{W.t - W.b} = \frac{Sy - V.b}{V.t - V.b}$$

## Window-to-Viewport Mapping Under the Hood (cont.)

- Solve for *Sx*, *Sy* in terms of *x*, *y*

$$\frac{(x - W.l)}{W.r - W.l} = \frac{Sx - V.l}{V.r - V.l} \qquad \frac{(y - W.b)}{W.t - W.b} = \frac{Sy - V.b}{V.t - V.b}$$

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

$$= Ax - A(W.l) - V.l$$

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

$$= By - B(W.b) - V.b$$



## Window-to-Viewport Mapping Under the Hood (cont.)

- Solve, given the formulas

$$Sx = Ax - \left(A(W.l) - V.l\right)$$

$$Sy = By - \left(B(W.b) - V.b\right)$$

- 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 Under the Hood (cont.)

$W = (W.l, W.r, W.b, W.t) = (0,4,0,2)$ $V = (V.l, V.r, V.b, V.t) = (60,380,80,240)$

- Solution

$$Sx = Ax - \left(A(W.l) - V.l\right) \qquad Sy = By - \left(B(W.b) - V.b\right)$$

$$A = \frac{V.r - V.l}{W.r - W.l} \qquad B = \frac{V.t - V.b}{W.t - W.b}$$

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

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



# References

- Hill: 3.1 – 3.3, 3.8