


Coordinate Systems

- Modeling/local coordinates
- World coordinates
- Normalized device coordinates
- Device coordinates


1



Modeling/Local Coordinates

- Convenient for object to be drawn
- Typical units: meters, feet, etc.
- Might not be Cartesian
- floats and doubles are common

2



World Coordinates

- Groups of objects are combined
- Form a complete image
- Allows prototype objects
 - Drawn in local coordinates
 - Copied, resized and moved into world coordinates
- Units still feet, meters, etc.

3

Normalized Device Coordinates

- Device-independent
- Horizontal and vertical ranges of 0 to 1
- "Independence" layer between world and various devices
 - Screen (windows of various sizes)
 - Printer

4

Device Coordinates

- Actual pixels to draw
- Allows for movable drawing windows
 - Usually handled by the O/S
- Pixel size (pixels/inch) is relevant
- Typical processing
 - $(x_{mc}, y_{mc}) \rightarrow (x_{wc}, y_{wc}) \rightarrow (x_{ndc}, y_{ndc}) \rightarrow (x_{dc}, y_{dc})$

5

Coordinate System Example

The diagram illustrates the coordinate system transformation process. It starts with a 'World' coordinate system containing three house shapes (blue, green, yellow). A 'Local' coordinate system is defined by a bounding box around the blue house. A red arrow points to a 'Normalized' coordinate system where the blue house is scaled to fit the [0,1]x[0,1] range. A second red arrow points down to the 'Device' coordinate system, where the normalized shapes are scaled and translated to fit the specific device's pixel dimensions.

6

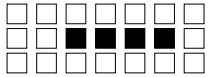
Drawing with Pixels

- Drawing algorithms
 - Point, line, circle, etc.
 - Assume pixels centers as reference
- Real pixels have finite size
 - Affects graphic primitive rendering
- Inter-pixel distances are fixed
 - Limited precision

7

Pixel Addressing

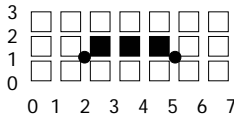
- Addressing a pixel by its center leads to problems
- A pixel occupies a finite space
 - It is not a true "point"
- Consider a line from (2,1) to (5,1)
 - Actual length = 3
 - Drawn length = 4



8


Boundary Addressing (1)

- Address pixels by their "boundaries"



- This "removes" the last pixel
 - Still not ideal


9



Compensating for Pixel Size

- Ignore the problem?
 - May make little difference
 - Lines may be connected anyway
- Shrink object by one pixel?
 - Sometimes done when filling
 - E.g., filling rectangle drops a pixel row, column


10



Boundary Addressing (2)

- We attempt to plot the interior of objects
 - Usually plot point if center is inside boundary
 - Compare with midpoint circle
 - Works better for squares etc.
 - Circles (text, p. 122)
- Still not ideal
 - Point are not infinitesimally small
 - Lines have finite width
 - Inside / outside / somewhere between?

11



Filling

- We can draw outlines
 - Polygons
 - Circles and other conics
- How do we make them solid?
 - Scan lines
 - A carefully drawn outline has pixels on each scan line

12

Basic Approach

- For each scan line
 - Determine the intersection with all boundaries
 - Look for active outline pixels –or–
 - Find them mathematically
 - Sort the intersection points
 - Scan line from left to right
 - Start outside
 - At each intersection, toggle in/outness

13



Example 1

A diagram showing a convex polygon with a horizontal scan line passing through it. The scan line intersects the polygon at two points. The region to the left of the first intersection is labeled 'OUT', the region between the two intersections is labeled 'IN', and the region to the right of the second intersection is labeled 'OUT'.

14



Example 2

A diagram showing a concave polygon with a horizontal scan line passing through it. The scan line intersects the polygon at four points. From left to right, the regions are labeled 'OUT', 'IN', 'OUT', and 'IN', illustrating how the scan line crosses in and out of the polygon's interior.

15

