Graphics Hit-testing

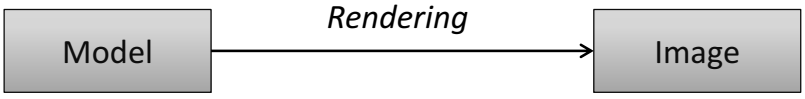
Shape Models

Selecting Lines and Shapes

Graphic Models and Images

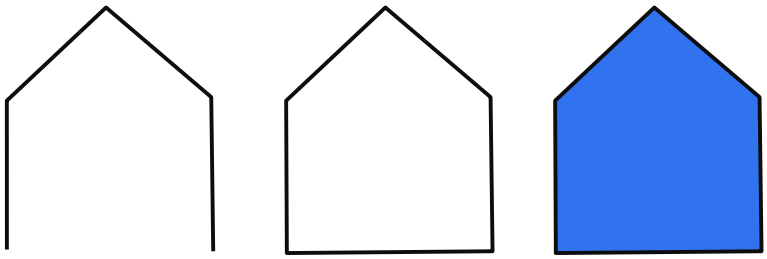
Computer Graphics is the creation, storage, and manipulation of images and their models

- **Model:** a mathematical representation of an image containing the important properties of an object (location, size, orientation, color, texture, etc.) in data structures
- **Rendering:** Using the properties of the model to create an image to display on the screen
- **Image:** the rendered model



Shape Model

- an array of points: $\{P_1, P_2, \ldots, P_n\}$
- isClosed flag (shape is polyline or polygon)
- isFilled flag (polygon is filled or not)
- (and stroke thickness, colours, etc.)

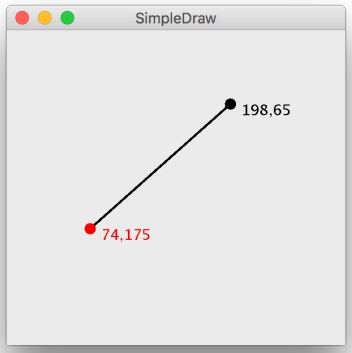


Implementing Direct Manipulation

- Objective: test when a rendered shape is “selected”
  - could be a filled or outlined polygon or a polyline
  - selections that “just miss” the shape should “snap” to shape

- Tasks:
  - create a model of the shape
  - draw it
  - choose a “selection” paradigm
  - implement shape **hit tests**
  - respond to events

SimpleDraw.java




```
// custom graphics drawing
   public void paintComponent(Graphics g) {
        super.paintComponent(g);
        // cast to get 2D drawing methods
        Graphics2D g2 = (Graphics2D) g;

        g2.setColor(Color.BLACK);
        g2.setStroke(new BasicStroke(2));
        g2.drawLine(C.x, C.y, M.x, M.y);
        g2.fillOval(M.x - 5, M.y - 5, 10, 10);

        g2.setColor(Color.RED);
        g2.fillOval(C.x - 5, C.y - 5, 10, 10);

        g2.drawString(String.format("%d,%d", C.x, C.y), ...

        g2.setColor(Color.BLACK);
        g2.drawString(String.format("%d,%d", M.x, M.y), ...
   }
```


Selection Paradigms

- Hit-test selection
  - open shapes like lines and polyline use edge hit-test
  - closed shapes like rectangles, and polygons use inside hit-test
- Alternate approaches we won't cover:
  - Rubberband rectangle
  - Lasso (see Ch 14 of text)

Linear Algebra: Affine Space

s a *scalar*: a single value (real number)

v a *vector*: directed line segment (direction and magnitude)

P a *point*: a fixed location in space (represents a position)

Legal operations:

vector + vector: $\mathbf{v_1} + \mathbf{v_2} = \mathbf{v_3}$

vector multiplied by scalar: $\mathbf{v_1} \times s_1 = \mathbf{v_4}$

point minus point: $P_1 - P_2 = \mathbf{v_5}$

point + vector: $P_2 + \mathbf{v_5} = P_1$

Two ways to multiply vector by vector,

dot (inner) product: $\mathbf{v_1} \bullet \mathbf{v_2} = s_2$

cross (outer) product: $\mathbf{v_1} \times \mathbf{v_2} = \mathbf{v_6}$

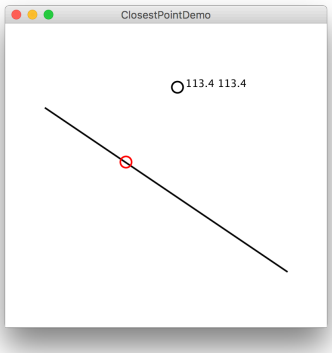
Line Segment Hit-test

- a line model has no “thickness”
- pick a threshold distance from mouse position to line
- point to line distance can be computed using vector projection

(blackboard…)

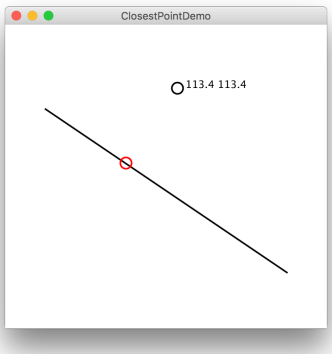
ClosestPointDemo.java

```
// find closest point using projection method
static Point2d closestPoint(Point2d M, Point2d P0, Point2d P1) {

     Vector2d v = new Vector2d();
     v.sub(P1,P0); // v = P2 - P1

     // early out if line is less than 1 pixel long
     if (v.lengthSquared() < 0.5)
        return P0;

     Vector2d u = new Vector2d();
     u.sub(M,P0); // u = M - P1

     // scalar of vector projection ...
     double s = u.dot(v) / v.dot(v);
     // find point for constrained line segment
     if (s < 0)
        return P0;
     else if (s > 1)
        return P1;
     else {
        Point2d I = P0;
        Vector2d w = new Vector2d();
        w.scale(s, v); // w = s * v
        I.add(w); // I = P1 + w
        return I;
     }
  }
```



ClosestPointDemo.java

```
// get distance using Java2D method
double d2 = Line2D.ptSegDist(P0.x, P0.y, P1.x, P1.y, M.x, M.y);
```



vecmath.jar

- you need vecmath.jar to run these demos
- vecmath.jar needs to be included when compiling and running

```
javac -cp vecmath.jar ClosestPointDemo.java
java -cp "vecmath.jar:." ClosestPointDemo
```

- Makefile should have everything you need

Polyline and Multiple Polyline Hit-testing

- a polyline is just many line segments
- check distance from every every polyline to mouse position
  - how can this be optimized?

(blackboard...)

Rectangle Shape Hit-Test

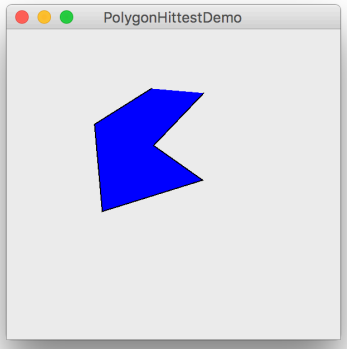
- assume axis-aligned
- rectangle shape useful as a “bounding box”

(blackboard…)

Mouse Inside Polygon Test

- is a point inside or outside a polygon?
- approach: find intersection points of horizontal line with polygon (can be optimized …)
- need special case test when horizontal line passes through end of line segments
- need a line-line intersection routine
- (blackboard…)

PolygonHittestDemo.java




```
public void paintComponent(Graphics g) {
   super.paintComponent(g);
   Graphics2D g2 = (Graphics2D) g;

   if (poly.contains(M.x, M.y))
      g2.setColor(Color.BLUE);
   else
      g2.setColor(Color.RED);

   g2.fillPolygon(poly);

   g2.setColor(Color.BLACK);
   g.drawPolyline(poly.xpoints, poly.ypoints, poly.npoints);
}
```