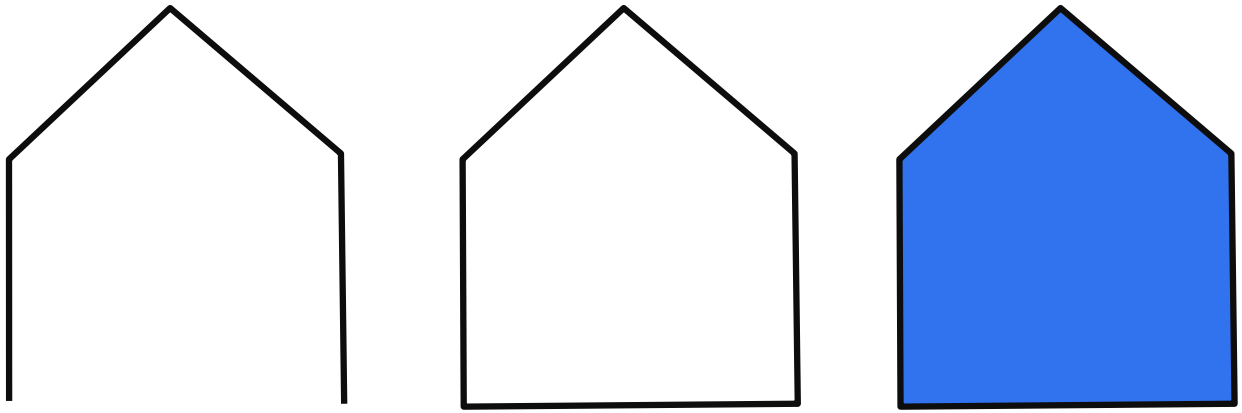
# Graphics Hit-testing

Shape Models

Selecting Lines and Shapes

## Recap - 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

- In a graphical interface, users expect to be able to select content using a mouse, and interact with it directly.
  - Includes graphical content, widgets etc.

- 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

- How do you implement this?
  - Create a model of the shape
  - Draw it
  - Choose a “selection” paradigm
  - Implement shape **hit tests**
  - Respond to events

## 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

## 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: $\mathbf{v_1} \times \mathbf{v_2} = \mathbf{v_6}$

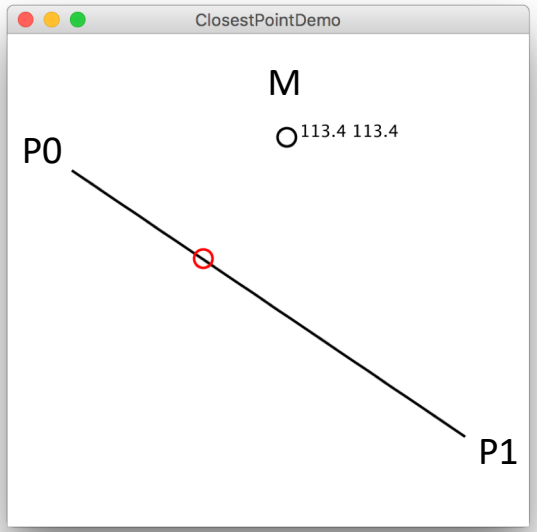
## Line Segment Hit-test

- 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…)

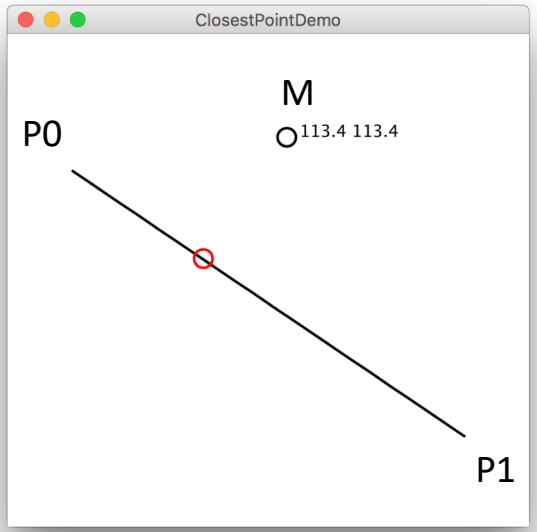
## ClosestPoint.java

```
// 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);
```



## Rectangle Shape Hit-Test

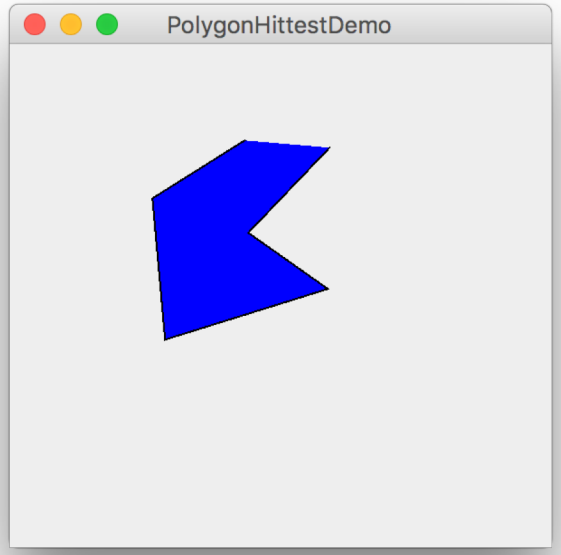
- assume axis-aligned
- rectangle shape useful as a “bounding box”

(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…)

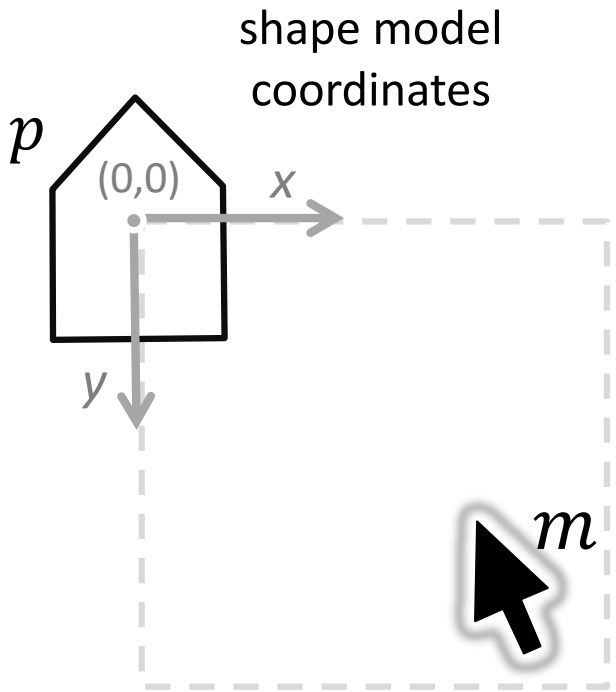
## PolygonHittest.java

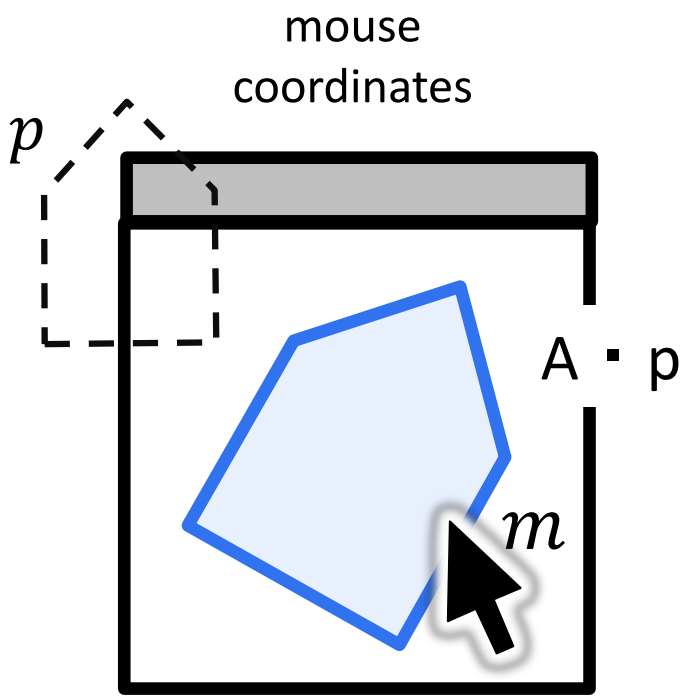



```
public void paintComponent(Graphics g) {
   super.paintComponent(g);
   Graphics2D g2 = (Graphics2D) g;

   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);
}
```

## Hit-testing with Transformed Shapes

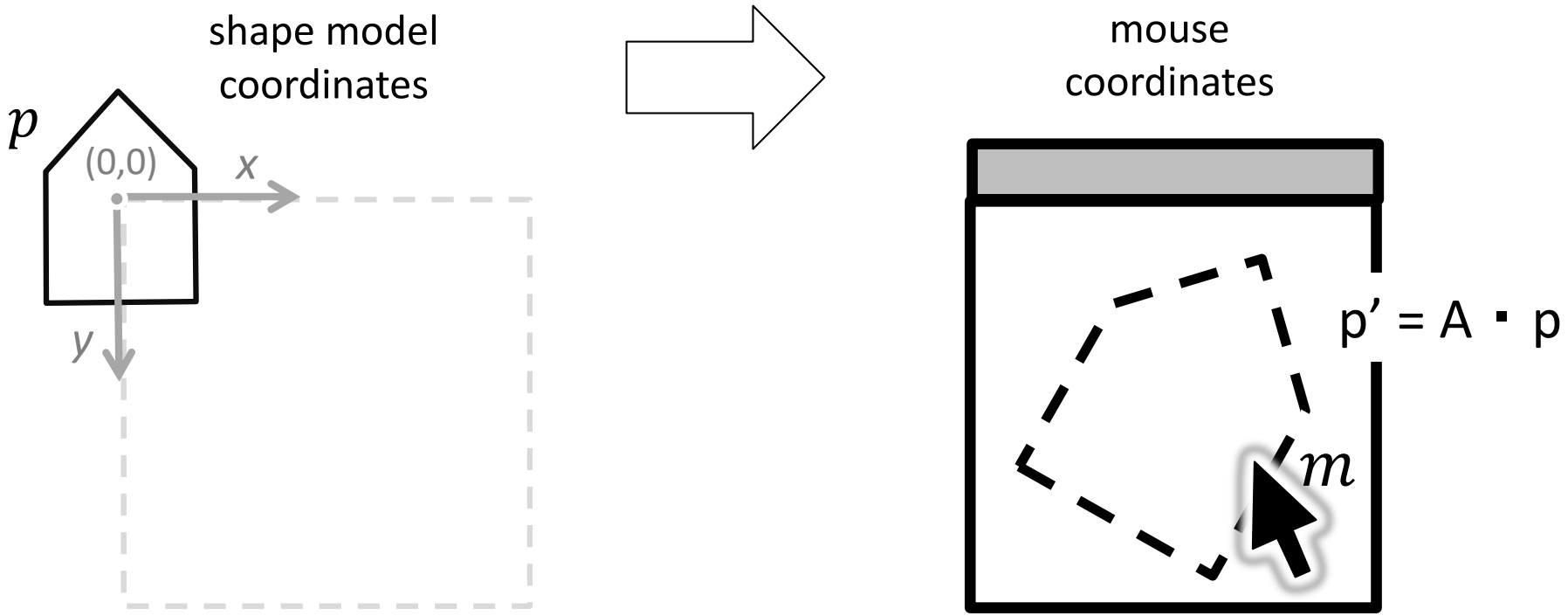
- Mouse and shape model are in different coordinate systems
- Two options for hit testing:
  1. Transform shape model to mouse coordinates
  2. Transform mouse to shape model coordinates



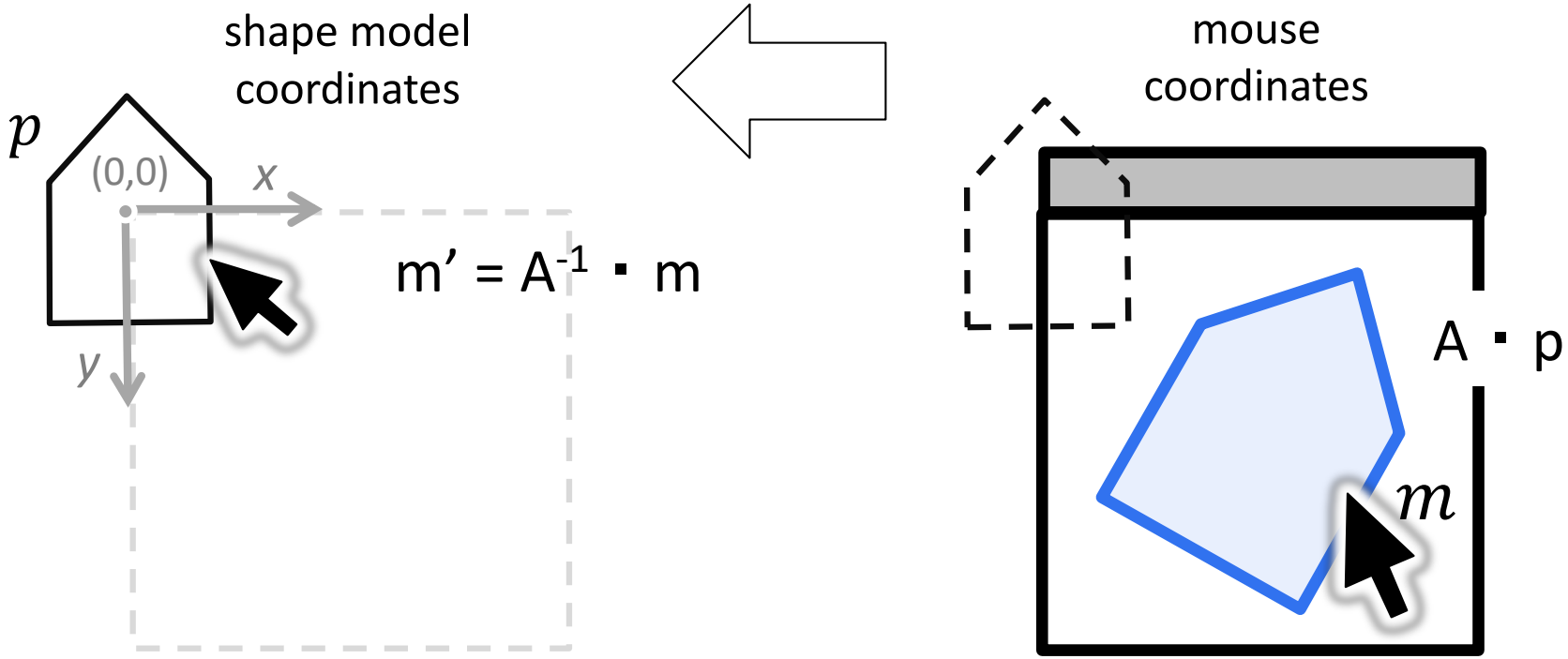
## Transform Shape Model to Mouse Coordinates

- Have to transform every point and/or parameter in shape model before running each hit-test algorithm
  - extra memory, lots of extra calculation



# Transform Mouse to Shape Model Coordinates

- Only one point to transform
- Need to adjust hit-test threshold …
  - e.g. 3 pixels in mouse coords. is how far in model coords?
  - what if non-uniform scale?
- Computing inverse can be costly …



## TransformHittest.java

```
Canvas () {
    // create transformation matrix for shape1
    AT1 = new AffineTransform();
    AT1.translate(350, 100);
    AT1.rotate(Math.toRadians(30));
    // create another transformation matrix for shape2
    AT2 = new AffineTransform();
    AT2.translate(200, 300);
    AT2.rotate(Math.toRadians(30));
    AT2.scale(2, 2);
}

public void paintComponent(Graphics g) {
    …
  // Shape1
    g2.setTransform(AT1);    // Use matrix AT1
    g2.setColor(Color.RED);
    g2.drawPolygon(shape.xpoints, shape.ypoints, shape.npoints);
    …

  // Shape2
    g2.setTransform(AT2);    // Use matrix AT2
    g2.setColor(Color.BLUE);
    g2.drawPolygon(shape.xpoints, shape.ypoints, shape.npoints);
    …
}
```

**TransformHittest.java**

```
public void paintComponent(Graphics g) {
   …
      // hit testing
      Point MT = new Point();
      // create an inverse matrix of AT1
      AffineTransform IAT1 = AT1.createInverse();
      // apply the inverse transformation to the mouse position
      IAT1.transform(M, MT);

      // Hit test with transformed mouse position
      if (shape.contains(MT.x, MT.y))
         g2.setColor(Color.RED);
      else
         g2.setColor(Color.WHITE);
         g2.fillPolygon(shape);
   …
}
```