Hit-Testing

- Shape Models
- Inside and Edge Hit-Testing with Various Shapes
- Find Closest Point using Vector Projection

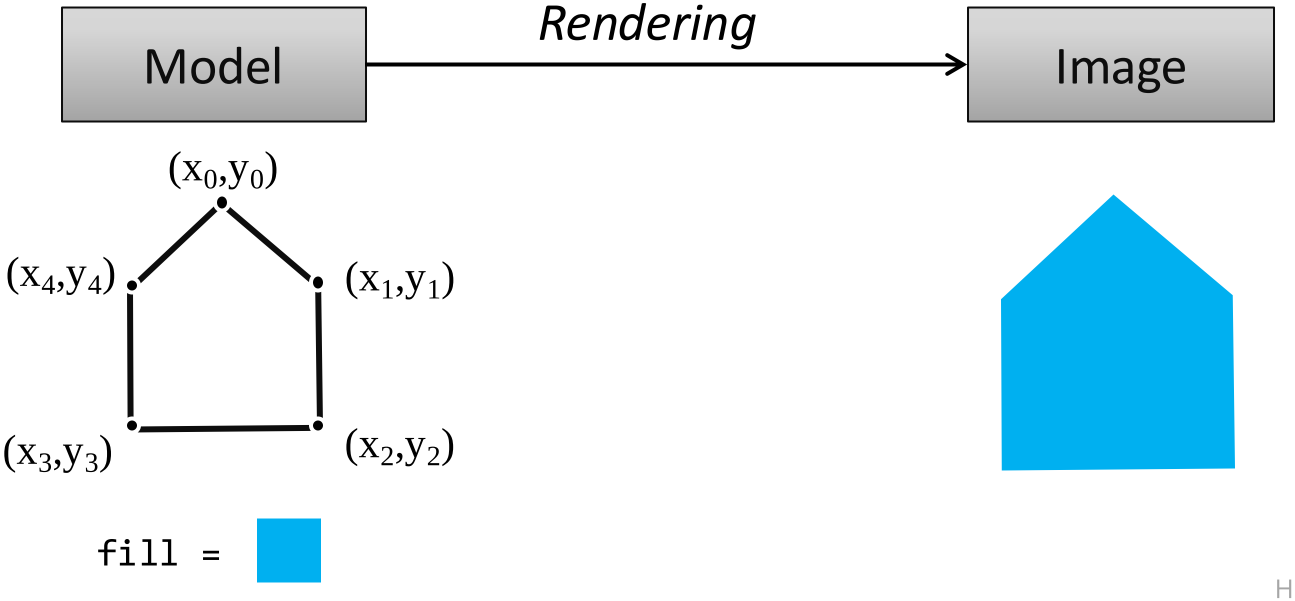
Shape Model vs. Image of Shape

Shape Model: the internal, oftentimes mathematical, representation of a shape

- geometry (points, bounds, key dimensions, …)
- visual style (fill, stroke thickness, …)
- transformations (translations, rotations, …)

Rendering: process to translate model into an image

Shape Image: the rendered “picture” of the shape



Model

Rendering

Image

(x_0,y_0)

(x_1,y_1)

(x_2,y_2)

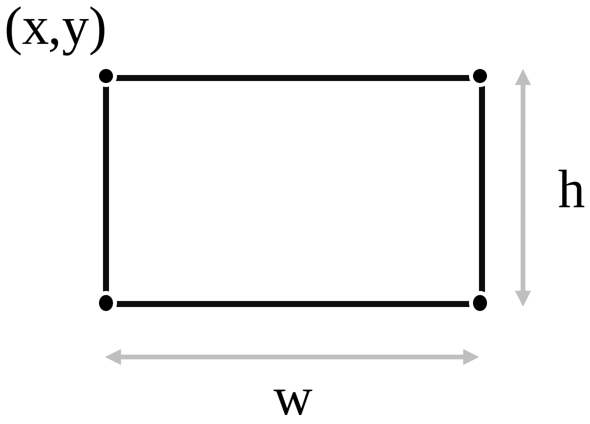
(x_3,y_3)

(x_4,y_4)

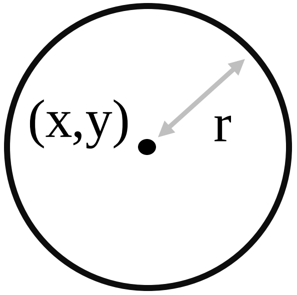
`fill =`

Shape Model Geometry

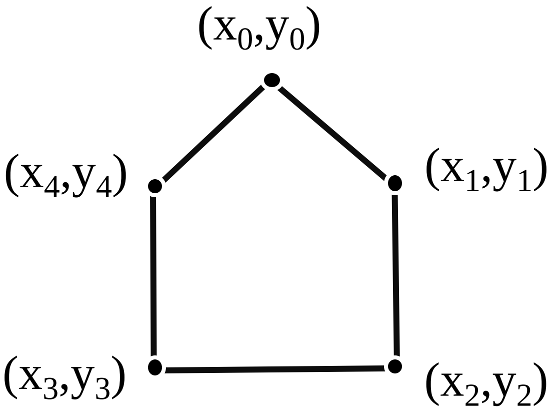
Different shapes have different *geometric representations*



Rectangle
top-left corner point
width and height



Circle
centre point
radius

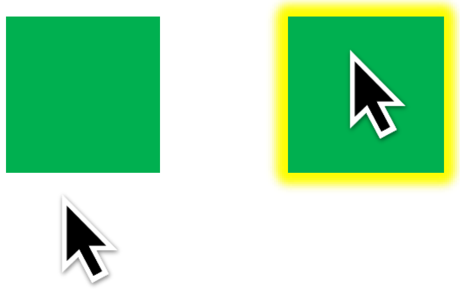


Polygon
list of points

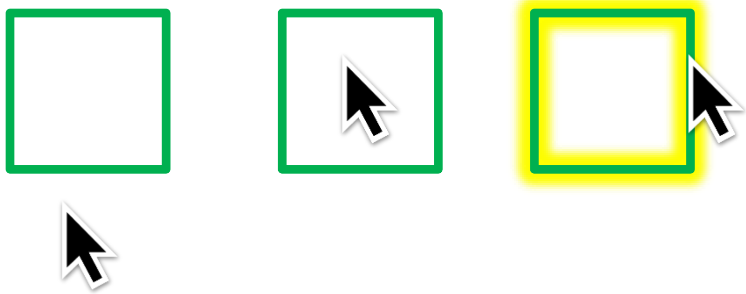
- Many alternate geometric representations possible
- Many other kinds of shapes: Line, Polyline, Ellipse, …
- Shape models can even be combinations of (different) shapes

Hit-Test Paradigms

- Inside Hit-Test
  - is mouse cursor inside shape?
  - closed shapes like Circle, Rectangle, and Polygon
  - usually when rendered with fill
- Edge Hit-Test
  - is mouse cursor on shape stroke?
  - open shapes like Line, Polyline
  - unfilled shapes when rendered with stroke



Inside
Hit-Test



Edge
Hit-Test

Hit-Test Implementation

A hit-test is tailored to the shape type and properties

- if edge hit-test, need to factor in thickness of stroke

```
function hitTest(
  mx: number,
  my: number,

  strokeWidth: number
  ): boolean {
  ...
}
```

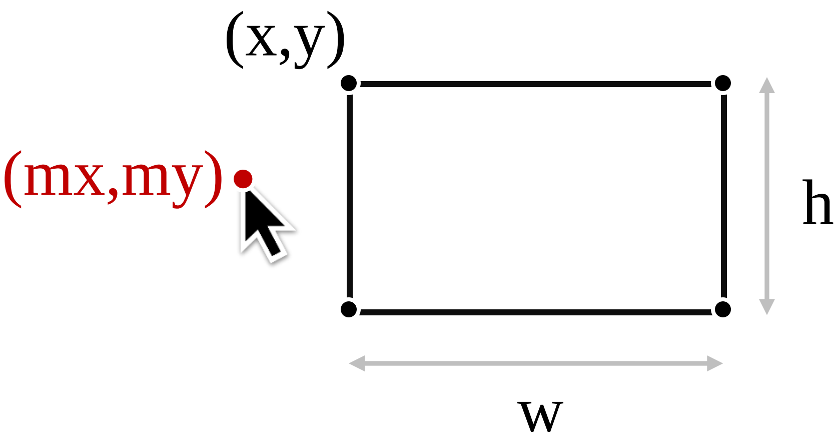
mouse position

shape properties go here

visual style properties
needed for some hit-tests

Rectangle Inside Hit-Test

- Given:
  - mouse position (mx, my)
  - rectangle top-left corner (x, y)
  - rectangle width w and height h
- Inside hit is true when these are true:
  - mx is in range [x, x + w]
  - my is in range [y, y + h]



Rectangle:
top-left corner point, width and height

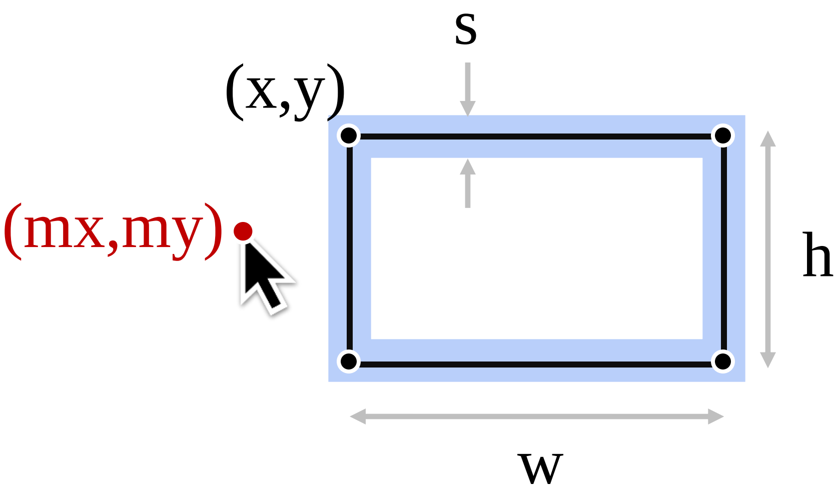
Rectangle Inside Hit-Test

```
function insideHitTestRectangle(
  mx: number,
  my: number,
  x: number, y: number,
  w: number, h: number
) {
  return mx >= x &&
         mx <= x + w &&
         my >= y &&
         my <= y + h
}
```

rectangle shape properties

Rectangle Edge Hit-Test

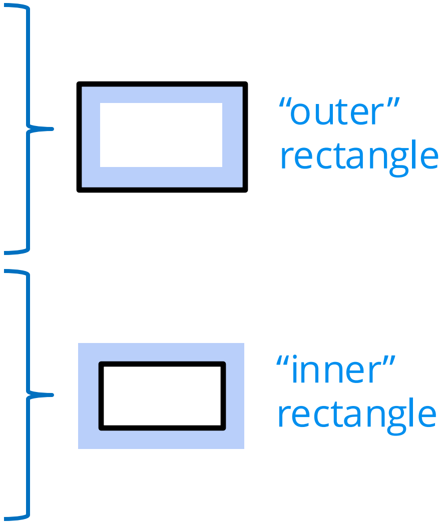
- Given:
  - mouse position (mx, my)
  - rectangle top-left corner (x, y)
  - rectangle width w and height h
  - stroke width s



Rectangle
top-left corner point
width and height

- Edge hit is true when these are true:
  - mx is in range [x – s/2, x + w + s/2]
  - my is in range [y – s/2, y + h + s/2]

  but these are false:
  - mx is in range (x + s/2, x + w - s/2)
  - my is in range (y + s/2, y + h - s/2)

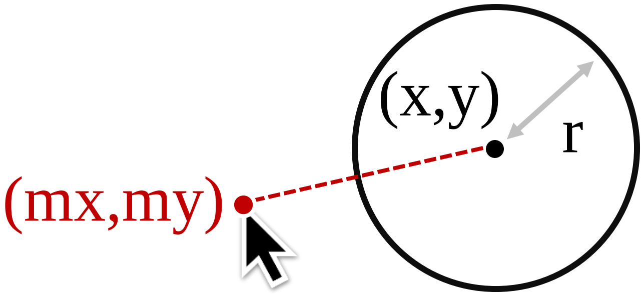


Rectangle Edge Hit-Test

```
function edgeHitTestRectangle(
  mx: number,
  my: number,
  x: number, y: number,
  w: number, h: number,
  strokeWidth: number
) {
  // width of stroke on either side of edges
  const s = strokeWidth / 2;

  // outside rect after adding stroke
  const outer = mx >= x - s && mx <= x + w + s &&
                my >= y - s && my <= y + h + s;

  // but NOT inside rect after subtracting stroke
  const inner = mx > x + s && mx < x + w - s &&
                my > y + s && my < y + h - s;

  return outer && !inner;

}
```

Circle Inside Hit-Test

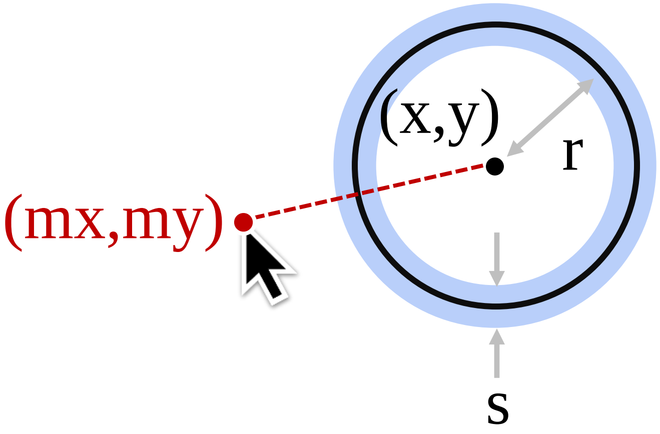
- Given:
  - mouse position (mx, my)
  - circle centre (x, y)
  - circle radius r
- Calculate:
  - distance from (mx, my) to (x, y) (Euclidean distance between the points)
- Inside hit is true when:
  - distance is less than or equal to r



Circle
centre point
radius

Circle Edge Hit-Test

- Given:
  - mouse position (mx, my)
  - circle centre (x, y)
  - circle radius r
  - stroke weight s
- Calculate:
  - distance from (mx, my) to (x, y)
- Edge hit is true when these are true:
  - distance is in range [r – s/2, r + s/2]

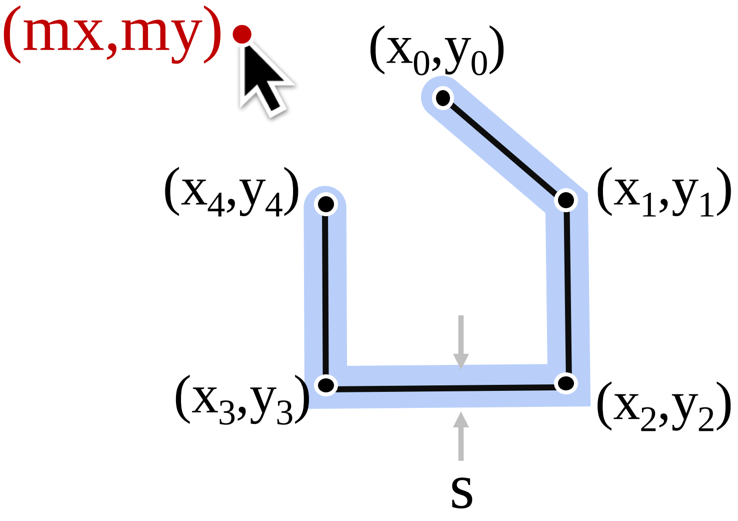


Circle
centre point
radius

Polyline Hit-Test

- Given:
  - mouse position (mx, my)
  - list of points
  - stroke weight s
- Calculate:
- Edge hit is true when:
  - edge hit test true for any **line segment**

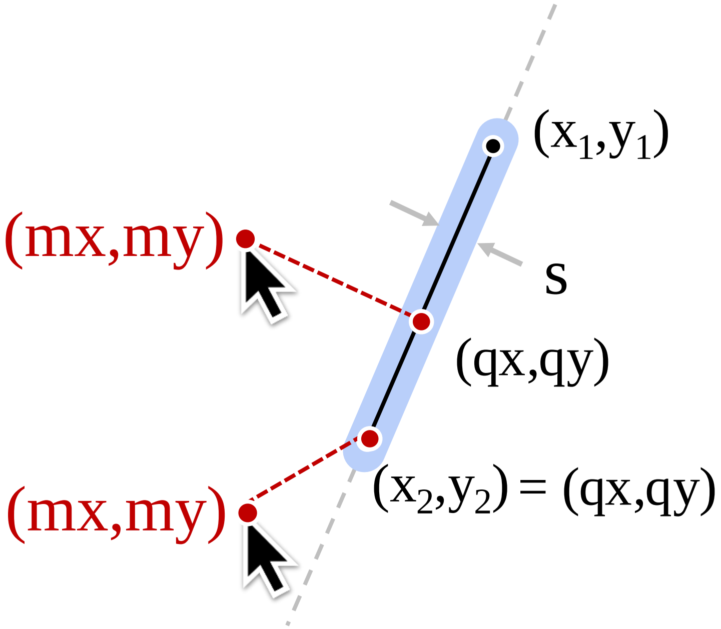
we need this first



Polyline
list of points

Line Edge Hit-Test

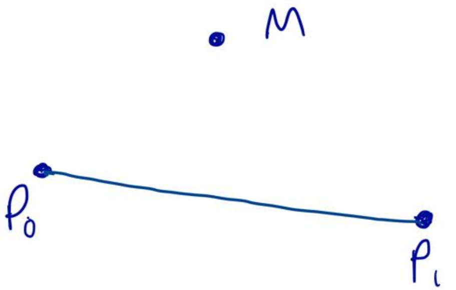
- Given:
  - mouse position (mx, my)
  - line start (x1, y1)
  - line end (x2, y2)
  - stroke weight s



- Calculate:
  - closest point on line segment: (qx, qy)
  - distance from (mx, my) to (qx, qy)
- Edge hit is true when:
  - distance is less than or equal to s/2

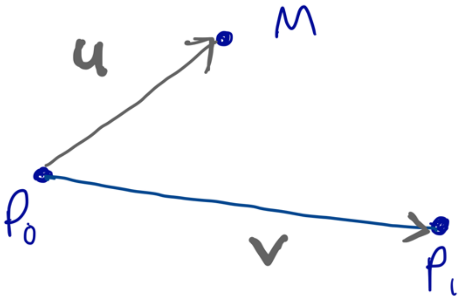
calculated using vector projection

Find Closest Point Q on Line with Vector Projection (1)



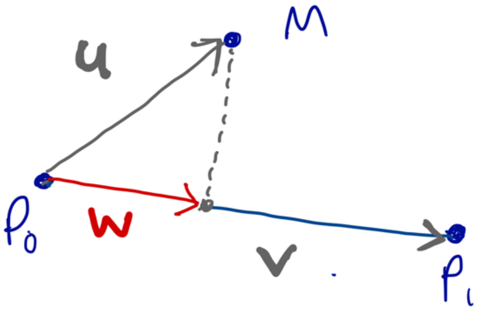
M is mouse position

P_0P_1 is line



$\mathbf{u} = M - P_0$

$\mathbf{v} = P_1 - P_0$



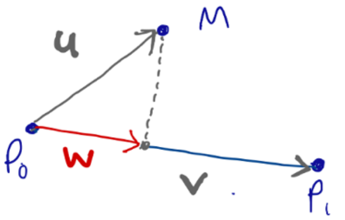
find projection $\mathbf{w}$ of $\mathbf{u}$ onto $\mathbf{v}$

$$\mathbf{w} = \frac{u \cdot v}{v \cdot v}\,\mathbf{v}$$

Find Closest Point Q on Line with Vector Projection (2)

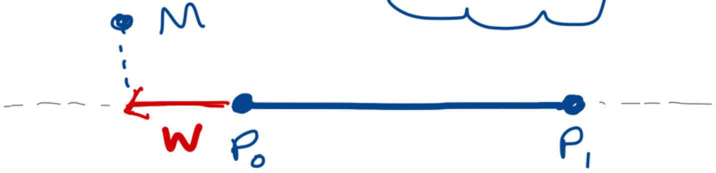
Let s be the scalar of the projection **w**

$$s = \frac{u \cdot v}{v \cdot v}, \quad \mathbf{w} = s\,\mathbf{v}$$

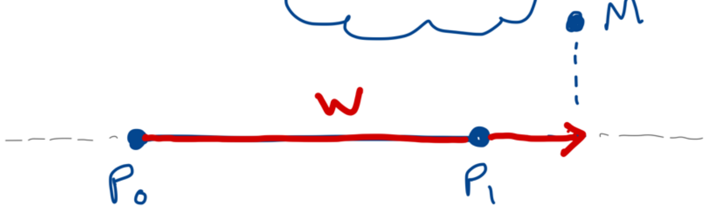


Use s to find closest point on line $P_0 P_1$

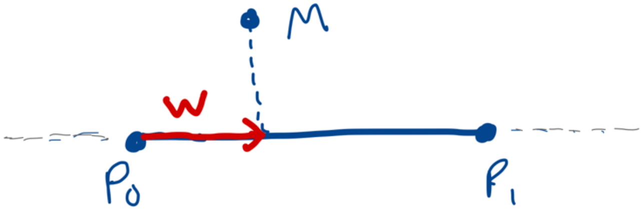
if $s < 0$ $\quad \boxed{Q = P_0}$



if $s > 1$ $\quad \boxed{Q = P_1}$



if $s \geq 0$ and $s \leq 1$ $\quad \boxed{Q = P_0 + \mathbf{w}}$



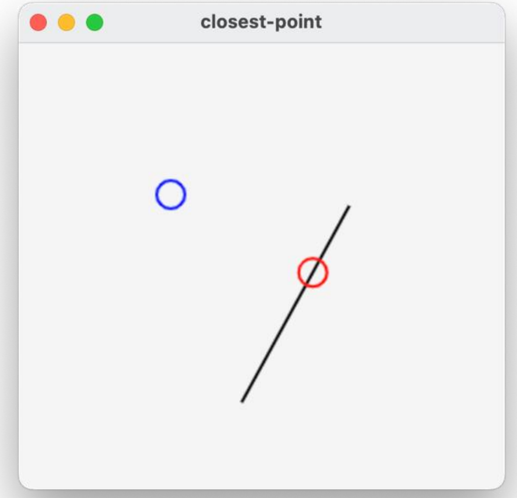
closestpoint

closetPoint.ts

- Direct implementation of math
- Uses Point, Vector, point, vector from SimpleKit utilities
  - Useful classes for applied linear algebra
- Note early return for edge case
- segmentOnly flag for debugging

main.ts

- Another typical SimpleKit canvas-mode app
- width and height variables set in resize event

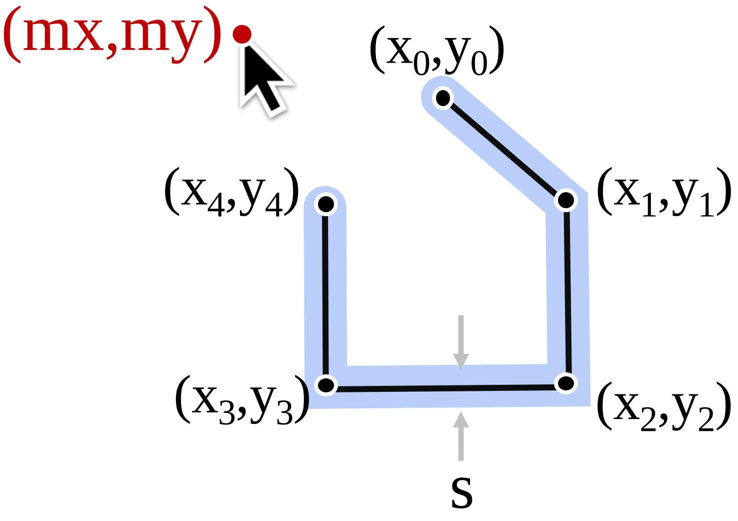


hittest / hittest-line.ts

- Find closest point on the line to the mouse position
- Find distance from mouse to that closest point
- If within half stroke width, it’s a hit

Polyline Hit-Test

- Given:
  - mouse position (mx, my)
  - list of points
  - stroke weight s
- Calculate:
- Inside hit is true when:
  - edge hit test true for any **line segment** ✅
  - note early return if hit



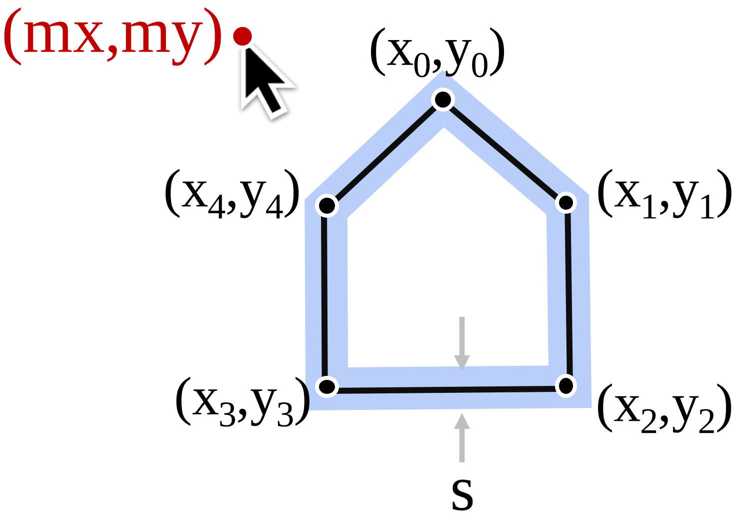
Polyline
list of points

hittest / hittest-polyline.ts

- Iterate through line segments
- If there's a hit, return true immediately
- destructuring and spread to set first point on first segment

Polygon Edge Hit-Test

- Given:
  - mouse position (mx, my)
  - list of points
  - stroke weight s
- Edge hit is true when:
  - edge hit test true for any **line segment**



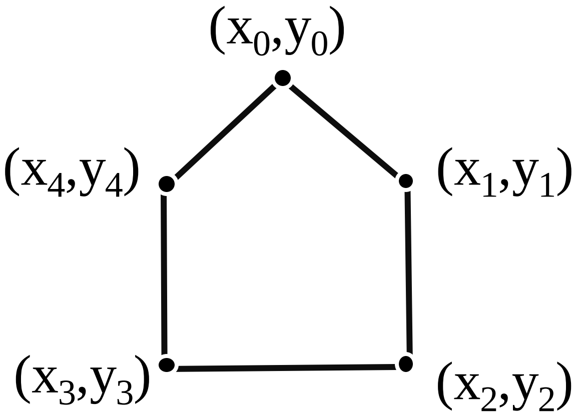
Polygon
list of points

hittest / hittest-polygon.ts

- edgeHitTestPolygon *uses* edgeHitTestPolyline
- Need to repeat the first point to close the polygon

Polygon Inside Hit-Test

- Given:
  - mouse position (mx, my)
  - list of points
- Inside hit is true when:
  (it gets complicated, see next slides …)

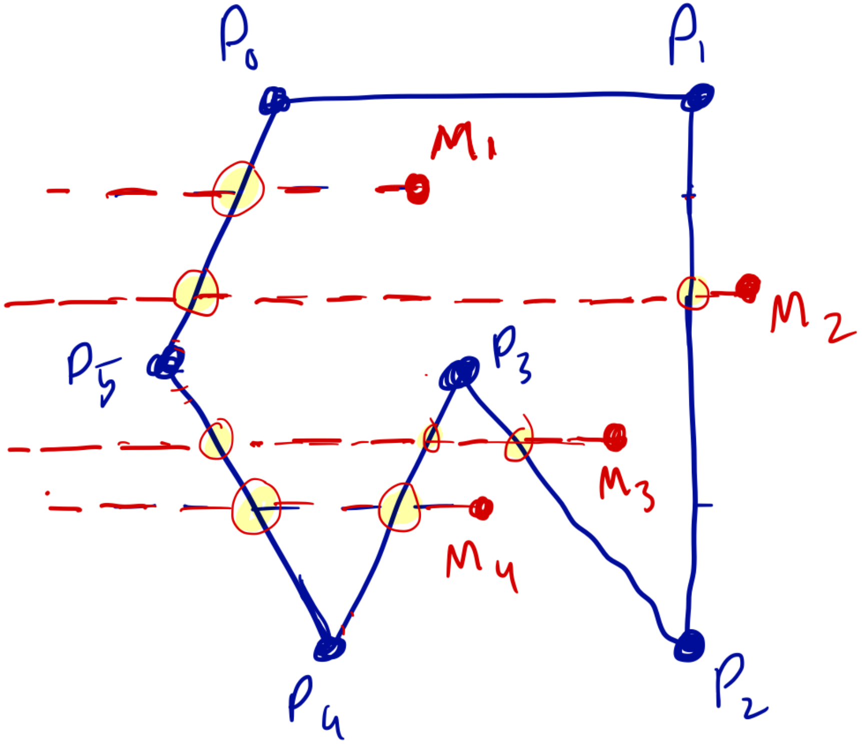


Polygon
list of points

Intuition for Inside Polygon Hit-Test

Cast y=0 ray from mouse position, count how many times it intersects line segments of Polygon

	Num	Result
M_1	1	INSIDE
M_2	2	OUTSIDE
M_3	3	INSIDE
M_4	2	OUTSIDE



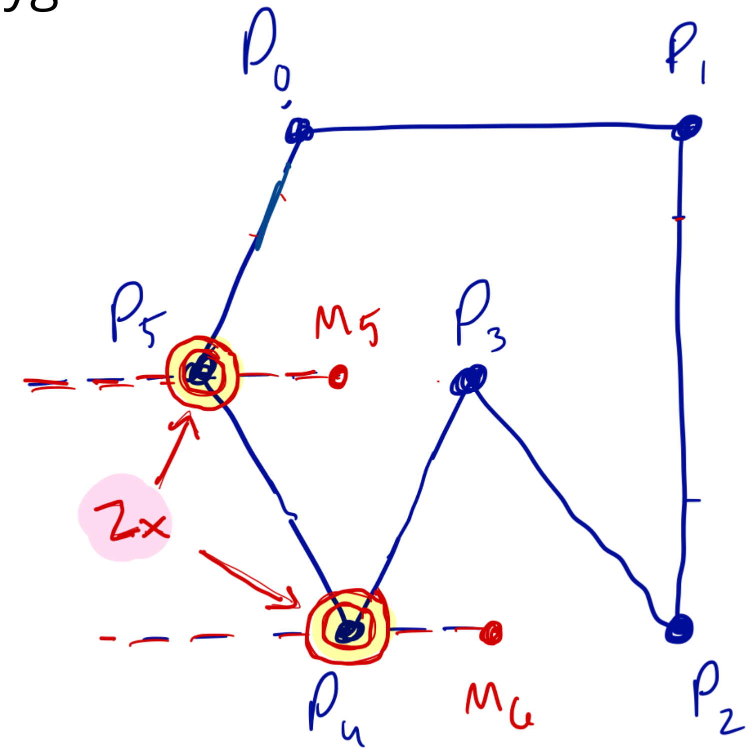
Rule: If odd number of intersections, inside hit-test is TRUE (almost …)

Intuition for Inside Polygon Hit-Test (Problem)

Cast y=0 ray from mouse position, count how many times it intersects line segments of Polygon

	Num	Result
M_5	2	INSIDE
M_6	2	OUTSIDE

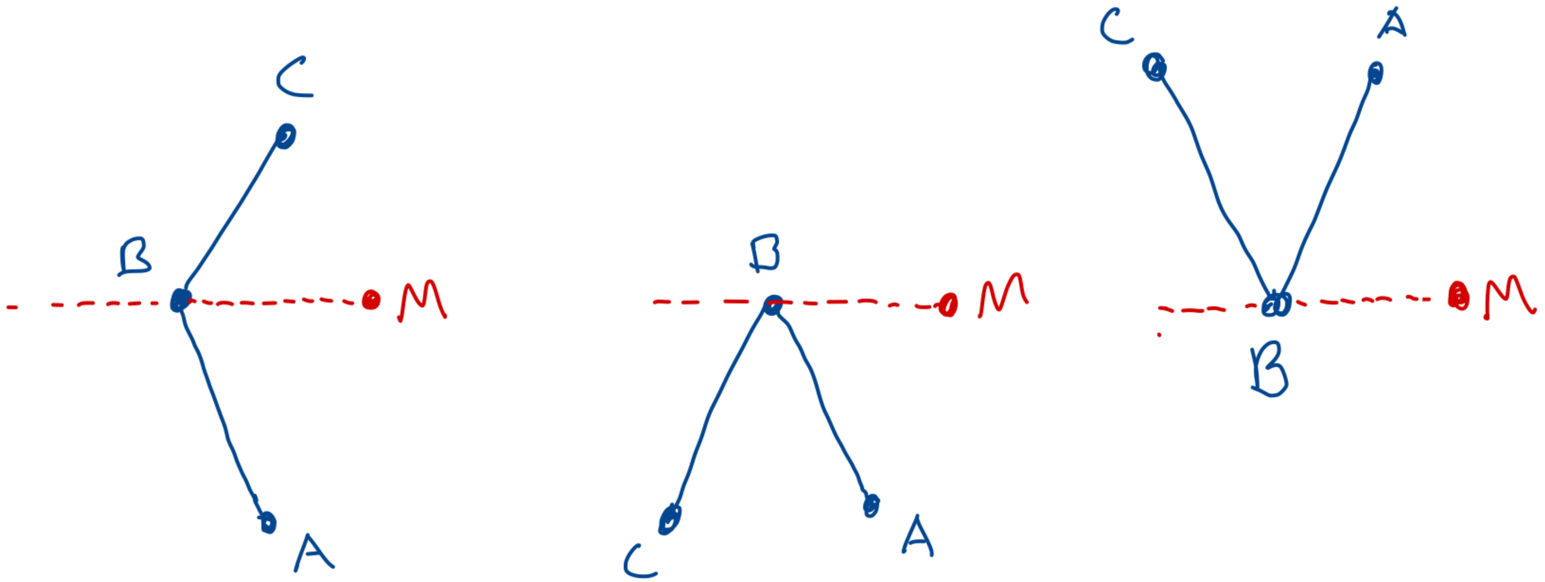
which is it ?



Problem: if ray intersects with a *point*, it intersects *two segments*, and this can happen when the mouse is inside or outside. (treat as special case …)

Intuition for Inside Polygon Hit-Test (Special Case)

If ray intersects with a point defining a line segment, add 1 only if other point on segment is “above” ray



M intersects AB at B

M intersects BC at B

Shape Class

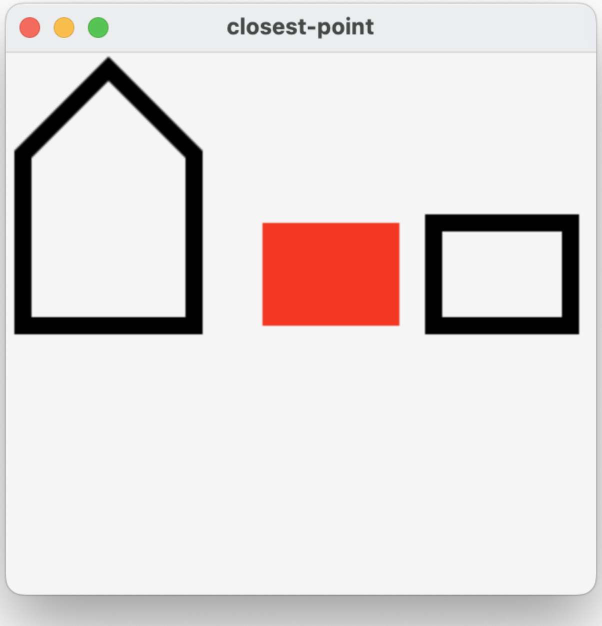
- geometry that defines the shape
- geometry properties (isFilled, isStroked)
- visual style properties (fill, stroke, strokeWeight)
- method to draw into a provided graphics context (like `Drawable`)
- method to do hit-testing with an x-y cursor position

Shape Base Class Implementation

```
abstract class Shape {
  fill: string = "grey";
  stroke: string = "black";
  strokeWidth = 1;

  get isFilled() {
    return this.fill != "";
  }

  get isStroked() {
    return this.stroke != "" && this.strokeWidth > 0;
  }

  abstract draw(gc: CanvasRenderingContext2D): void;

  abstract hitTest(mx: number, my: number): boolean;
}
```

shapes

- Shape abstract base class
- Shape models (esp. hitTest method):
  - Rectangle
  - Polygon
- Uses DisplayList approach for rendering



Hit-test Optimizations

- Hit-testing could become computationally intensive
  - There could be hundreds of shapes in a scene
  - Polygon or Polyline shapes could have hundreds of edges
- Approaches to reduce hit-testing computation:
  - avoid square root in distance calculations
    (for circle, see if squared distance is less than r^2)
  - use simpler less precise hit-test first for an “early” reject
    (e.g. start with a bounding-box, or bounding circle hit-test)
  - split scene into cells, and track which ones each shape is in
    (called octree or binary space partition approaches)

Alternative Methods: Raster Hit Testing in a Buffer

- Use offscreen buffer to draw shape
  - often at lower resolution, using standard transformation
- Transform mouse coordinates to match buffer
- Examine pixel at mouse position in buffer
  - return true if pixel is not #000000
- Can also use pixel alpha (transparency)
- Can also use different colours to hit-test different regions

DOM Canvas API Hit-Testing

Test if point is inside area contained by shape path:

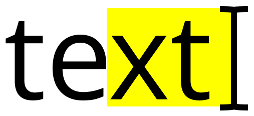
```
// built-in Canvas API hit test
const hitFill = gc.isPointInPath(mx, my);
const hitStroke = gc.isPointInStroke(mx, my);
```

- It handles stroke thickness (lineWeight to graphics context)
  - true if point is anywhere on visible stroke
- It handles unfilled shapes
  - true only if point is on visible stroke area, false if inside

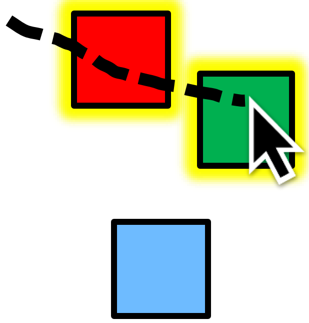
https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/isPointInStroke

Other Hit-Test Selection Paradigms

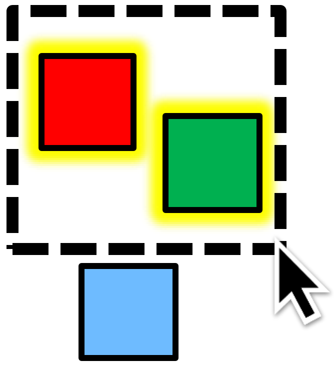
- Text selection
  - insertion point, drag to select
- Crossing intersection
  - select by drawing stroke through shapes
- Shape Intersection
  - Marquee selection (select shapes in oriented bounding box)
  - Lasso selection (select shapes enclosed in freeform path)



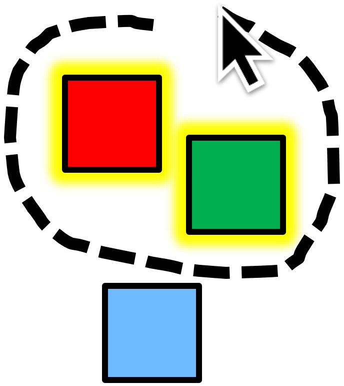
Text Selection



Crossing Selection



Marque Selection

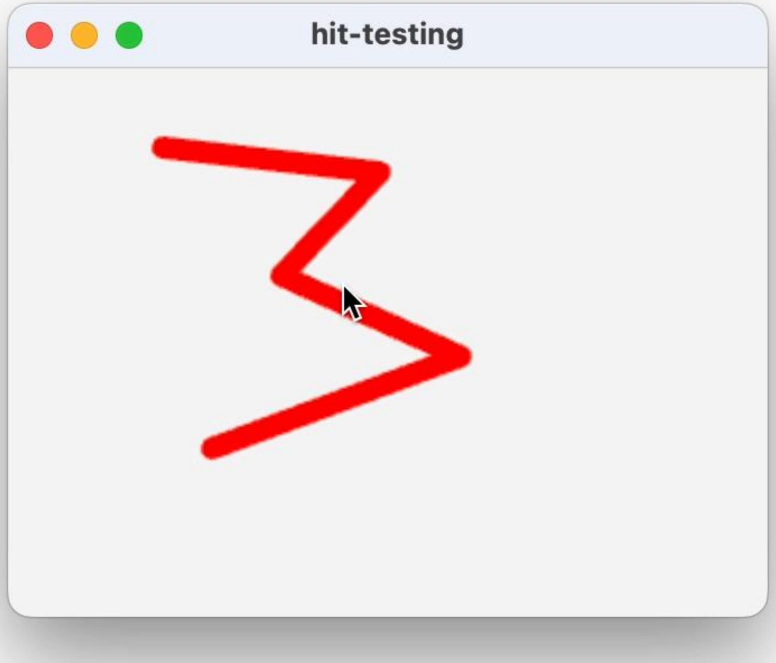


Lasso Selection

Exercise

EXERCISE

- Create a simple line drawing app
- On mousedown, add a point to a poly line:
  - create a PolyLine shape class with the points array as a public property
  - Use thick 10px strokeWidth to draw line
  - Use lineCap and lineJoin canvas drawing methods to make line look nicer
  - draw a dot when only one point
- On mousemove, do hit testing:
  - draw the PolyLine in red if hit
  - Otherwise draw it in black
- Pressing SPACE key clears the line
  - Just set the PolyLine points array to [ ]



Demo of interactions:
https://vault.cs.uwaterloo.ca/s/fKiJJTCW9r6sxqX