A little bit of geometry

Jordi Cortadella
Department of Computer Science

Representing points and lines in $\mathbb{R}^2$

// A point has two coordinates
struct Point {
    double x, y;
};

// A line: $y = mx + b$
struct Line {
    double m; // Slope
    double b; // y-intercept;
};

// A segment represented by two points
struct Segment {
    Point P, Q;
};

Line equation from two points

Finding a line from a segment

// Returns the line defined by the segment.
Line FindLine(const Segment& S) {
    Line L;
    L.m = (S.P.y - S.Q.y)/(S.P.x - S.Q.x);
    L.b = S.P.y - L.m*S.P.x;
    return L;
}

// Be careful with vertical lines!
// A special treatment is required.

$y = mx + b$

$m = \frac{dy}{dx} = \frac{P.y - Q.y}{P.x - Q.x}$

$b = P.y - m \cdot P.x$
Given two segments: do they intersect?

Do the two points fall on the same side of the line?

Relative position of a point

Is A above, below or on the line?
Compute the sign of \( y_3' - y_3 \)

Relative position of two points

\[
y_3' = \frac{dy}{dx} x_3 + \left( y_1 - \frac{dy}{dx} x_1 \right)
\]

\[
y_3' - y_3 = \frac{dy}{dx} x_3 + y_1 - \frac{dy}{dx} x_1 - y_3
\]

Point A: \( dx(y_3' - y_3) = \frac{dy}{dx} (x_3 - x_1) - dx(y_3 - y_1) \)
Point B: \( dx(y_4' - y_4) = \frac{dy}{dx} (x_4 - x_1) - dx(y_4 - y_1) \)

We are only interested on whether \( y_3' - y_3 \) and \( y_4' - y_4 \) have the same sign and not on the actual sign of the expressions.
Points at the same side

// Returns true if A and B at are the same side of the line defined by S, and false otherwise.

bool SameSide(const Segment& S, const Point& A, const Point& B) {
    double dx = S.P.x - S.Q.x;
    double dy = S.P.y - S.Q.y;
    double dxA = A.x - S.P.x;
    double dyA = A.y - S.P.y;
    double dxB = B.x - S.P.x;
    double dyB = B.y - S.P.y;
    return (dy*dxA - dx*dyA > 0) == (dy*dxB - dx*dyB > 0);
    // or also: (dy*dxA - dx*dyA)*(dy*dxB - dx*dyB) >= 0
}

Important: the function works for any line (even vertical lines!). The expressions are perfectly symmetric with regard to the x and y axes.

Note: we work with real numbers.
Some inaccuracies may occur when the points are close to the segment.

Segment intersection

// Returns true if S1 and S2 intersect, and false otherwise.
bool Intersect(const Segment& S1, const Segment& S2) {
    return not (SameSide(S1, S2.P, S2.Q) or
    SameSide(S2, S1.P, S1.Q));
}

The original problem: segment intersection

// Returns true if S1 and S2 intersect, and false otherwise.
bool Intersect(const Segment& S1, const Segment& S2);

Representation of polygons

A polygon can be represented by a sequence of vertices.

Two consecutive vertices represent an edge of the polygon.

The last edge is represented by the first and last vertices of the sequence.

Vertices: (1,3) (4,1) (7,3) (5,4) (6,7) (2,6)

Edges: (1,3)-(4,1)-(7,3)-(5,4)-(6,7)-(2,6)-(1,3)

// A polygon (an ordered set of vertices)
typedef vector<Point> Polygon;
Why artificial vision is so difficult?

Point inside a polygon?

Use the crossing number algorithm:
- Draw a ray (half-line) from the point
- Count the number of crossing edges:
  - even → outside
  - odd → inside

We will use a horizontal ray (x coordinate = ∞)
#include <limits>  // To use the infinity value
// Returns true if the point is inside the polygon, // and false otherwise.
bool InsidePolygon(const Polygon& P, const Point& A) {
    Segment Ray; // Horizontal ray
    Ray.P = A;
    Ray.Q.x = numeric_limits<double>::infinity(); Ray.Q.y = A.y;

    Segment Edge;
    Edge.P = P[P.size() - 1]; // The last point of P
    bool inside = false;
    for (int dst = 0; dst < P.size(); ++dst) {
        Edge.Q = P[dst];
        if (Intersect(Ray, Edge)) inside = not inside;
        Edge.P = Edge.Q;
    }
    return inside;
}

Summary

- Computational geometry has a vast range of applications in different domains: Visualization, Computer Graphics, Virtual Reality, Robotics, etc.

- Challenge: finding fast solutions for complex geometric problems (think of a 3D real-time video game processing millions of pixels per second).

- Dealing with the accuracy of real numbers is always an issue, but no so important if a certain tolerance for errors is allowed.

Again, problems with the accuracy of real numbers. How about the ray intersecting a vertex? (this problem is beyond the scope of this course)