UCF Programming Team Practice
Geometry Lecture Notes

Lines

There are several ways to represent a line:

General (implicit) form:	$Ax + By + C = 0$
Slope-intercept:	$y = mx + b$
Parametric:	$x = x_0 + t\,dx$, $y = y_0 + t\,dy$
	where $dx = x_2 - x_1$, and $dy = y_2 - y_1$
	for any two points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ on the line

In general, the parametric forms are easiest to deal with in code, because they let you deal with x and y separately, and they give you a single value to represent any position on the line. This even works for lines in three dimensions; you only need a third parametric equation representing z.

In particular, line segments are especially easy to represent parametrically. Select the two points P_1 and P_2 as the two endpoints of the segment, then your parameter *t* will vary from 0 to 1 along the segment (other values of t are not on the segment).

Line and Line Segment Intersections

Intersections of lines and segments occur in contest problems frequently. The solution is simply to solve the system of equations given by the two lines. For lines (or segments) A and B, solving for the point of intersection (x, y), the equations would look like this:

$$x = Ax_1 + s\,Adx \qquad x = Bx_1 + t\,Bdx$$
$$y = Ay_1 + s\,Ady \qquad y = By_1 + t\,Bdy$$

There are four equations in this system, and there are also four unknowns (x, y, s, and t). However, setting the two x's and the two y's equal to each other:

$$Ax_1 + s\,Adx = Bx_1 + t\,Bdx$$
$$Ay_1 + s\,Ady = Bx_1 + t\,Bdy$$

or,

$$s\,Adx - t\,Bdx = Bx_1 - Ax_1$$
$$s\,Ady - t\,Bdy = By_1 - Ay_1$$

Now, we have a nice, tidy system to solve. There are many techniques to solve systems such as this. A simple one to code is Cramer's Rule (see your Linear Algebra or Calculus book for a thorough discussion). Remember, for segments to intersect, the parametric point of intersection (the s and t values in the equations above) must be between 0 and 1 for *both* segments.

Beware of parallel and collinear lines and segments, these require special handling.

Vectors

Vectors are a very useful geometric tool. The parametric form of a line or line segment can be expressed in a single vector-valued equation, for example:

$$L = \mathbf{P_1}\, t + \mathbf{P_2}\,(1 - t), \text{ for points } \mathbf{P_1} \text{ and } \mathbf{P_2} \text{ on line } L$$

Vectors can also represent the distance and direction between two points, the velocity of a moving object, and many other things. Vectors can be represented in the same way as points, typically a structure of two int's or double's (three in 3D problems).

Vector magnitude

Obtaining the magnitude of a vector is similar to obtaining the distance between two points:

$$\| \mathbf{v} \| = \sqrt{\mathbf{v}_x^2 + \mathbf{v}_y^2}$$

Sometimes, it is necessary to *normalize* a vector (making its magnitude equal to one). This is done by simply dividing each component of the vector by the vector's magnitude.

Dot product

The dot product of two vectors is simply the sum of the products of each vector's respective components. That is,

$$\mathbf{A} \cdot \mathbf{B} = \mathbf{A}_x \mathbf{B}_x + \mathbf{A}_y \mathbf{B}_y$$

also,

$$\mathbf{A} \cdot \mathbf{B} = \| \mathbf{A} \| \| \mathbf{B} \| \cos\theta$$

Note that the dot product is a scalar value (not a vector).

Cross product

The cross product of two vectors is a vector. This vector is always orthogonal (perpendicular) to the other two, unless the original two are parallel or opposite (in which case, it is undefined). For two-dimensional vectors, the cross product is a three-dimensional vector with the x and y components both zero and the z component having a value as shown:

$$\mathbf{A} = < \mathbf{A}_x, \mathbf{A}_y >$$
$$\mathbf{B} = < \mathbf{B}_x, \mathbf{B}_y >$$
$$\mathbf{A} \times \mathbf{B} = < 0, 0, \mathbf{A}_x \mathbf{B}_y - \mathbf{A}_y \mathbf{B}_x >$$

also,

$$\| \mathbf{A} \times \mathbf{B} \| = \| \mathbf{A} \| \| \mathbf{B} \| \sin\theta$$

The scalar cross-product value (the z-component of a two-dimensional cross product) has some useful properties. If the scalar cross product of two vectors is positive, then the rotation from the first vector to the second is counter-clockwise. If negative, it is clockwise.

Angle between two vectors

The angle between two vectors can be easily determined using the dot product, and the scalar value of the cross product. Here's how:

$$\mathbf{A}\cdot\mathbf{B} = \|\mathbf{A}\|\|\mathbf{B}\|\cos\theta$$
$$\|\mathbf{A}\times\mathbf{B}\| = \|\mathbf{A}\|\|\mathbf{B}\|\sin\theta$$

so,

$$\frac{\sin\theta}{\cos\theta} = \frac{\|\mathbf{A}\times\mathbf{B}\|}{\mathbf{A}\cdot\mathbf{B}}$$

and

$$\theta = \tan^{-1}\left(\frac{\|\mathbf{A}\times\mathbf{B}\|}{\mathbf{A}\cdot\mathbf{B}}\right)$$

The atan2() function works well for the inverse tangent in this equation, since it's answer is valid from –pi to pi. If you don't know how atan2() works, educate yourself.

Polygons

Polygons are almost always represented by a set of consecutive points (vertices), and they fall into three categories:

- Convex: Non-intersecting edges, all interior angles < 180 degrees
- Concave: Non-intersecting edges, one or more interior angles > 180 degrees
- Complex: One or more intersecting edges

In general dealing with complex polygons is far more difficult than dealing with simple (non-complex) polygons. Likewise, concave polygons are often trickier to handle than convex polygons.

Area of a Polygon

Finding the area of a polygon is quite simple. For each edge of the polygon (each pair of adjacent vertices, create a trapezoid, using the difference of the two x – coordinates as the "height" and each y coordinate as one of the "bases." Note that the trapezoid's area will be negative if the difference in x coordinates is negative. This is part of the process. Sum the trapezoid areas up and you will have the area of the polygon. If the vertices of the polygon are in counter-clockwise order, the total area will be negative, so you should always take the absolute value of the final area. In case you've forgotten your area formulae:

$$A_{trapezoid} = \frac{1}{2}h(B_1 + B_2)$$

where B_1 and B_2 are the y coordinates of the polygon vertices, and h is the difference in x coordinates.

Point in Polygon

Given a point and a polygon (convex or concave), you are to determine if the point is inside the polygon. This can be done using the angle between two vectors formula. For each pair of adjacent vertices, determine the angle between the vectors from the point to each vertex. Sum each of these angles, and at the end, you'll have 2π if the point is inside the polygon, and 0 if not. This method works for convex and concave polygons.

This method fails if the test point is one of the polygon's vertices. Also, a point on the polygon's edge may not provide a correct answer (the answer will be π, but it may need to be $-\pi$ for your sum to work out particular situation). Finally, make sure you understand what the problem means by "inside." Is a point on the polygon's edge (or a vertex) "inside" or "outside"?

Convex hull

Build the smallest fence you can around a group of trees. Tie a string around a group of nails hammered into a board. Compute the minimum amount of paint a warning boundary around a group of potholes would require. Each of these problems can be solved using a convex hull. A convex hull is the smallest convex polygon that surrounds and encompasses a group of points.

To find a convex hull, start with the bottom-most, left-most point. That is, start with the point with the smallest y coordinate. If multiple points have that y coordinate, pick the one with the smallest x coordinate. This is the first point on the convex hull. At each subsequent point on the hull, form a vector from that point to each remaining point. Select the point with the vector that makes the smallest positive angle with the previous vector (at the first point, your "previous vector" points to the right, in the positive X direction). Stop when you reach the initial point.

Make sure you understand whether the problem wants collinear points included on the convex hull or not. If so, include them. If not, when choosing the next point on the hull from several collinear points, make sure you pick the point farthest from the current point (otherwise you may include collinear points inadvertently).

Distances

It is often required to compute the distance between various geometric features.

Point-Point

This is the simplest distance to compute. Given two points, P and Q, the distance is essentially the magnitude of the vector between them.

$$D = \|\mathbf{Q} - \mathbf{P}\| = \sqrt{(Q_x - P_x)^2 + (Q_y - P_y)^2}$$

Point-Line and Point-Segment

This method requires a little more math. If you're given the line in the general form (Ax + By + C = 0) as well as a point (u, v), the distance from (u, v) to the line is given by the simple formula:

$$D = \frac{|Au + Bv + C|}{\sqrt{A^2 + B^2}}$$

To understand this a little better, realize that the vector $\langle A, B \rangle$ is a normal (perpendicular) vector to the line. Let's denote this as vector **N**. If you assume a point **P** is on the line, then the line equation becomes $\mathbf{N} \cdot \mathbf{P} + C = 0$. Further, C can be computed using another point **Q** on the line as so: $C = -\mathbf{Q} \cdot \mathbf{N}$. Armed with this knowledge, we can create a function (very similar to the *D* equation above),

$$f_s(\mathbf{P}) = \frac{\mathbf{N} \cdot \mathbf{P} + C}{\|\mathbf{N}\|}$$

Now, instead of assuming **P** is on the line, we can test for it. Further, we can determine the distance from the line if it's not. If $f_s(\mathbf{P}) = 0$, then **P** is on the line. If not, then the value of $f_s(\mathbf{P})$ indicates the distance from **P** to the line. Note that $f_s(\mathbf{P})$ actually returns a signed distance, which can be used to decide which half-plane (i.e, which side of the line) that the point **P** is on. If $f_s(\mathbf{P})$ is positive, then it lies on the same side of the line that the normal vector **N** is pointing. If it's negative, then it lies on the opposite side. If you don't care, take the absolute value (as done in the *D* equation above).

More often, you're given two points (call them **P** and **Q**) to describe a line. You can solve for *A* and *B* (vector **N**) and *C* given these two points as follows:

$$\begin{aligned}
&\mathbf{N} \cdot (\mathbf{P} - \mathbf{Q}) = 0 \therefore \\
&\mathbf{N} = (\mathbf{P} - \mathbf{Q})^{\perp} = \langle -(P_y - Q_y), (P_x - Q_x) \rangle = \langle A, B \rangle \\
&C = -\mathbf{Q} \cdot \mathbf{N}
\end{aligned}$$

Once you have **N** and *C*, you can use the formula above. However, a more straightforward way to handle this is to use point rotation (described below). Rotate point **Q** around point **P** so that **P** lies horizontal and to the right of **Q**. Next, rotate the test point (also around **P**) by the same amount. Now, you can simply subtract the y coordinate of the line from the test point's y coordinate.

Finally, if you're finding the distance from a point to a line segment (using either technique), you'll also have to consider the distance to each of the segment's endpoints and return the minimum of the three distances. If you're using the point rotation method, a quick shortcut is to determine if the x coordinate of the rotated test point is between the x coordinates of the rotated endpoints. If so, then the y distance is the correct answer. If not, then you'll need to compute the distance to the closer endpoint and return that.

Point Rotation

To rotate a point *B* around the origin, you can multiply the point as a vector <x, y> by the rotation matrix R_θ as follows:

$$[x' \quad y'] = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = [x\cos\theta - y\sin\theta \qquad x\sin\theta + y\cos\theta]$$

This will result in a new point *B*' that is rotated about the origin by θ radians. To rotate *B* about another point *A* instead of the origin, simply subtract *A* from *B* before rotating, then add *A* back to *B*' when done.

References

Akenine-Möller, Tomas and Eric Haines, *Real-Time Rendering (2nd ed.)*, pp. 732-734, A.K. Peters, Natick, MA, 2002.