High-Dimensional Computational Geometry

Jingbo Shang
University of Illinois at Urbana-Champaign
Mar 5, 2018
Outline

• 3-D vector geometry
• High-D hyperplane intersections
• Convex hull & its extension to 3 dimensions
• Point inside polyhedron
• Voronoi diagrams
3-D Vector Geometry

• Recall:
  • Vectors have magnitude and direction
  • Can be represented by a coordinate triple \( \mathbf{a} = (a_x, a_y, a_z) \)

• Dot product:
  • In 2-D, \( \mathbf{a} \cdot \mathbf{b} = a_x b_x + a_y b_y = \|\mathbf{a}\| \|\mathbf{b}\| \cos \theta \)
  • In 3-D, \( \mathbf{a} \cdot \mathbf{b} = a_x b_x + a_y b_y + a_z b_z = \|\mathbf{a}\| \|\mathbf{b}\| \cos \phi \), where \( \phi \) is the angle between \( \mathbf{a} \) and \( \mathbf{b} \) in the plane formed by \( \mathbf{a} \) and \( \mathbf{b} \)

• Direction cosines:
  • \( \mathbf{u} = \|\mathbf{u}\| (\cos \alpha \mathbf{i} + \cos \beta \mathbf{j} + \cos \gamma \mathbf{k}) \)
3-D Vector Geometry

• Cross product:
  • In 2-D, $\|\mathbf{a} \times \mathbf{b}\|$ is the area of the parallelogram formed by $\mathbf{a}$ and $\mathbf{b}$
  • In 3-D, $\mathbf{a} \times \mathbf{b}$ describes normal vector for two vectors
    • Right-hand rule

• $\mathbf{a} \times \mathbf{b} = (a_y b_z - a_z b_y)\mathbf{i} + (a_z b_x - a_x b_z)\mathbf{j} + (a_x b_y - a_y b_x)\mathbf{k}$
High-D Hyperplanes

• What is a hyperplane?
  • “Any subspace of one dimension less than its ambient space”
  • In 1-D, a point \( x = a \)
  • In 2-D, a line \( ax + by = c \)
  • In 3-D, a plane \( ax + by + cz = d \)
  • etc.
  • Can also be described by a point and a normal vector (remember the cross-product?)

• How do they intersect?
  • In 2-D, at a point
  • In 3-D, at a line
  • In general, in the form of a hyperplane with one fewer dimension
3-D Lines and Planes

• What about lines in 3-D?
  • Lines have 2 fewer dimensions than their ambient space
  • Usually described using parametric equations:
    • \( x, y, z = (x_0 + at, y_0 + bt, z_0 + ct) \)
  • Can also be described as the intersection of 2 planes
  • It is possible for two lines to not be parallel but not intersect!

• Lines and planes
  • If a line is not parallel to a plane, they will intersect in a point
Convex Hull Problem

• Convex hull: “find the smallest convex polygon completely enclosing a set of points”

• Graham Scan uses polar line sweep
  1. Sort points in angle order
  2. Append points in increasing order to chain
  3. If point causes concave corner, remove previous points until chain becomes convex

Time complexity: $O(n \log n)$
Convex Hull in 3-D

• How do we perform Graham Scan in 3 dimensions?
  • How do we define a 2-dimensional polar angle?

• We need a better way
  • There is a divide-and-conquer algorithm...
Quickhull

1. Find the most extreme points in each dimension (min and max x- and y-values)
2. Draw lines between the points; points inside this quadrilateral cannot be on the convex hull and are removed
3. Splitting the remaining points into 2 sets by a line through the extreme x-value points
4. In each set, find the point that is the farthest from the line; points inside the triangle formed by this point and the line can be removed
5. Recursively repeat this process for the 2 new lines created by the triangle
6. Once there are no more points left to process, the points selected constitute the convex hull

Time complexity: $O(n \log n)$
Quickhull
Convex Hull in 3-D

• Perform Quickhull in 3-D:
  1. Find all extreme points in each dimension (x, y, and z)
  2. Use the extreme points to form a tetrahedron (simplex)
  3. For each face, determine which points are outside the tetrahedron and discard all others
  4. Find point with largest distance from face
  5. Find all faces visible from that point; create new face with each horizon of the face and the point
  6. Remove all points not outside the new faces
  7. Repeat until all points have been removed; chosen set is convex hull

Good explanation: http://thomasdiewald.com/blog/?p=1888
Point Inside Polyhedron

• For 2-D, can use ray-casting (precision problems) or winding number algorithm (see http://geomalgorithms.com/a03-_inclusion.html)

• For 3-D:
  • If polyhedron is planar polygon, can project down into 2-D and solve
  • Otherwise, need to use ray-casting for each face
Voronoi Diagrams

• For a set of points in a plane, Voronoi diagram describes the *nearest neighbor* point for any given point in the plane

• Can be used to solve “largest empty circle” and “nearest neighbor lookup” problems
Fortune’s Algorithm

• Is a sweep line algorithm
  • Keep track of sweep line and beach line
  • Points to the left of the sweep line have been considered
  • For each point to the left of the sweep line, beach line is the boundary of the union of the set of parabolas equidistant from sweep line and the points
  • Intersections between parabolas become edges of the Voronoi diagram
  • Intersections between 3 parabolas become vertices of the Voronoi diagram
  • Keep track of parabolas using BST and priority queue for next points and intersections to consider

Time complexity: $O(n \log n)$
Fortune’s Algorithm

Illustration:

Delaunay Triangulation

- Triangulation of a set of points that maximizes the minimum angle of all triangles
- Delaunay triangulation also ensures that no point \( p \) is in the circumcircle of any triangle in the triangulation
- Centers of circumcircles formed by DT are the vertices in the Voronoi diagram:
Additional topics

• Rotating calipers – can be used to find the diameter and width of a polygon, but has many additional applications (bounding boxes, convex hull of convex polygons, etc.)
  • [http://www.tcs.fudan.edu.cn/rudolf/Courses/Algorithms/Alg_ss_07w/Webprojects/Qinbo_diameter/2d_alg.htm](http://www.tcs.fudan.edu.cn/rudolf/Courses/Algorithms/Alg_ss_07w/Webprojects/Qinbo_diameter/2d_alg.htm)
Example Algorithm Implementations

• O’Rourke’s *Computational Geometry in C* ([link](#))
  • Convex hull (2-D and 3-D)
  • Delaunay triangulation
  • Segment-segment intersection
  • Point in polygon
  • Point in polyhedron

• Rotating calipers ([link](#))

• These implementations are long and include error checking, so they are not suited for contest
  • Practice implementing your own, and use a well-debugged and short version in contests!
Example Problem

- Given two 3D convex hulls, judge whether they have any intersection.
- The number of vertices \( \leq 100 \)
Resources

• Different types of convex hull algorithms:
  http://jeffe.cs.illinois.edu/teaching/compgeom/notes/01-convexhull.pdf

• Fortune’s algorithm for Voronoi diagrams:
  http://blog.ivank.net/fortunes-algorithm-and-implementation.html

• Rotating calipers description:
  • https://en.wikipedia.org/wiki/Rotating_calipers
  • http://www.tcs.fudan.edu.cn/rudolf/Courses/Algorithms/Alg_ss_07w/Webprojects/Qinbo_diameter/2d_alg.htm
Homework

• SPOJ RUNAWAY (2)
• POJ 2820 (2)
• UVa 1111 (2)
• SPOJ TWOCIR (3)
• UVa 10095 (3)
• UVa 1488 (3)