6.851: Advanced Data Structures Spring 2010

Lecture 6 — February 23, 2010

Dr. André Schulz *Scribe: Skye Wanderman-Milne*

1 Overview

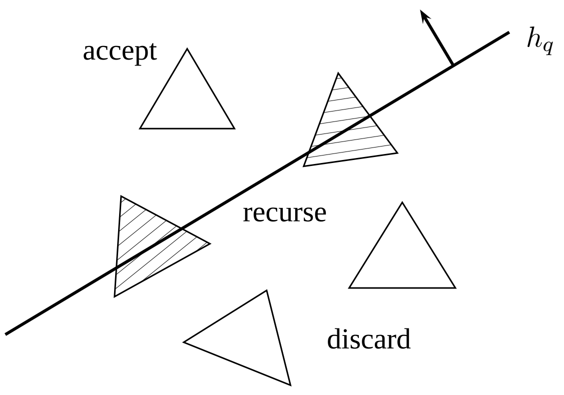
In the last lecture we introduced ray shooting, where we determine which is the first object in a set intersected by a given ray. We overviewed how to solve this problem if our objects are simple polygons. This lecture explores ray shooting more generally, beginning with data structures designed to perform halfspace and simplex range queries such as partition trees, and continuing with an explanation of how to use these data structures to perform ray shooting.

2 Partition Trees

Problem. Given a pointset $S = \{p_1, p_2, \ldots, p_n\}$, we would like to perform two sorts of queries:

1. *Halfspace Range Queries:* find properties relating to the subset of S on one side of a line h_q (e.g., how many points are above h_q?).
2. *Simplex Range Queries:* find properties relating to the subset of S inside a simplex t_q (e.g., how many points lie inside t_q?). In two dimensions, a simplex is a triangle, and we will use two-dimensional examples for the remainder of these notes.

Idea. Partition S into r disjoint subsets $S_1, S_2, \ldots, S_r$. Each subset S_i is associated with a triangle t_i that contains the points in that subset (the triangles need not be disjoint). We call this partition $\Psi = \{(S_1, t_1), (S_2, t_2), \ldots, (S_r, t_r)\}$. When finding the points in the halfplane defined by a query line h_q, we can easily accept or discard as necessary the points lying in a triangle that do not intersect h_q. We then recurse on the remaining subsets in triangles intersecting h_q.



The *crossing number* of Ψ is the maximum number of triangles that can be intersected by some line.

We call Ψ *fine* if $|S_i| \leq \frac{2n}{r}\ \forall\ S_i$, meaning every subset of S has no more than twice the average number of points per subset (i.e., the subsets are fairly equally distributed).

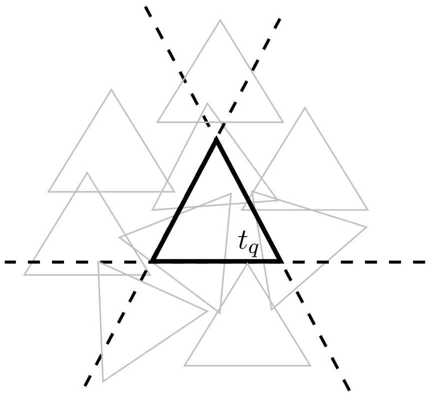
Jiří Matoušek has proved that for every r such that $1 \leq r \leq n$, there exists a fine partition Ψ with size r and crossing number $O(\sqrt{r})$, and that Ψ can be computed in $O(n^{1+\epsilon})$ time for $\epsilon > 0$ [1]. If we partition S in this manner, a query line h_q cannot intersect more than $O(\sqrt{r})$ triangles. Thus we will need to recurse on at most $O(\sqrt{r})$ subsets.

Performance. Parition trees yield the following performance for a halfspace range query:

- Query time: $O(n^{\frac{1}{2}+\epsilon})$
- Storage: $O(n)$
- Preprocessing: $O(n^{1+\epsilon})$

The derivation is left as a homework exercise.

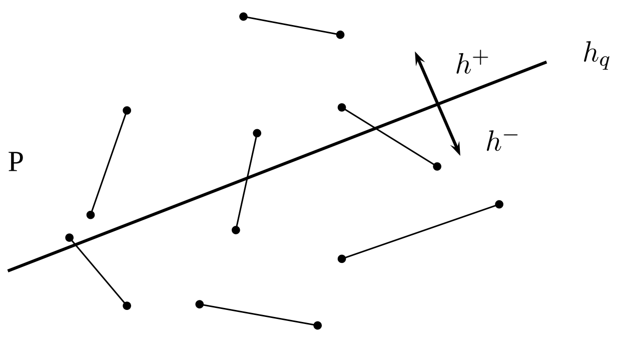
For a simplex range query, we know that the query triangle t_q intersects no more than three times the crossing number of our partition. Thus, the query time depends on t_q, but will never be worse than the bounds given for a halfspace range query.



3 Multi-level Partition Trees

3.1 First attempt

Problem. Given a set of line segments, we would like to find the segments stabbed by a query line h_q; i.e., all the segments with one endpoint in the halfplane above h_q, h^+, and the other in halfplane below h_q, h^-.



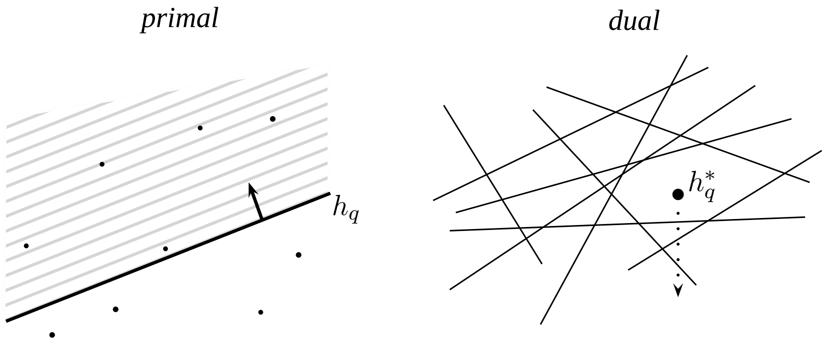
Idea. We split the endpoints of the line segments into two sets, S_l and S_r, such that each line segment has an endpoint in each set. Using a partition tree, we find all segments with an endpoint in h^+. Then using a second-level partition tree, we check if the other endpoint of each segment lies in h^-.

Performance.

- Query time: $O(n^{\frac{1}{2}+\epsilon})$
- Storage: $O(n \log n)$
- Preprocessing: $O(n^{1+\epsilon})$

3.2 Decreasing query time via increased storage

Geometric duality transform. If we have a point $p = (p_x, p_y)$ in the primal, we transform it into a line p^* in the dual with $y = p_x x - p_y$. A line in the primal is likewise transformed into a point in the dual. Thus, if we consider a halfplane range query over pointset S with query line h_q in the dual, we obtain a query point h_q^* and a set $|S^*|$ of lines, which form $O(n^2)$ cells (regions enclosed by a subset of the lines). The points in S above h_q correspond to the lines in S^* below h_q^* (imagine sweeping h_q, which corresponds to moving h_q^* down vertically). These lines will be the same regardless of where h_q^* is in a given cell.



Locus approach. We precompute all cells in the dual, and locate the correct cell given a query.

1. Storage: $O(n^2)$, corresponding to the number of cells.
2. Query time: $O(\log n)$ for finding the correct cell (via trapezoidal maps).

3.3 Other results

- A data structure with $O(\sqrt{n}2^{O(\log^* n)})$ query time [2].
- In dimension d, a data structure with $O(n^{1-\frac{1}{d}+\epsilon})$ query time and $O(n^{1+\epsilon})$ storage [2].
- Query time lower bound: $\Omega(\frac{n}{m^{\frac{1}{d}}\log n})$ (where m is storage) [?].
- Halfspace reporting in two dimensions with $O(\log n)$ query time and $O(n)$ storage [?].
- Simplex reporting in $O(m^{1+\epsilon})$ space and $O(\frac{n}{m^{\frac{1}{d}}}\log^2 n)$ query time [5].

4 Ray Shooting

The ray shooting problem can be thought of in two ways:

1. Decision: Given a point q, direction d, and distance λ, does the line segment of length λ from d along d intersect anything?
2. Optimization: Given a point q and direction d, what is the smallest λ such that the decision problem can be answered positively?

We notice that the decision version of the ray shooting problem is monotone, meaning that after reaching some minimal value λ, the answer to the decision problem will be positive for all larger values of λ.

Given a monotone decision algorithm, we can construct a new algorithm that solves the optimization problem by symbolically evaluating the original decision algorithm. At the start of the execution, we say λ has values ranging from $(-\infty, +\infty)$. Whenever λ is used in a polynomial expression as an input to a branch, we can solve the polynomial for the critical points at which the branch goes one way or the other, and evaluate both under each set.

Obviously, this approach is slow, since the total work increases by a factor of two for every branch executed. We can improve its performance by "batching up" polynomials. If the polynomials have at most p terms, then our optimization algorithm runs in $O(p) + O(\log pT_D)$, where T_D is the time to run the decision algorithm.

To answer a weaker version of the ray shooting decision problem, we consider shooting a ray into a set of hyperplanes. Since λ is given in the decision problem, the problem is to check if there are any hyperplanes between the two ends of the line segment formed by q, the ray starting point, and $q + d\lambda$, the end of the ray. To solve this problem, we can use geometric duality to transform all of the hyperplanes into points and line segment ends into hyperplanes. The two line segment ends will form a wedge out of two hyperplanes. If and only if the line segment intersects any hyperplanes in the primal, there will be points between the two hyperplanes in the dual.

References

[1] J. Matoušeck, *Efficient partition trees*, Discrete & Computational Geometry, 1992.

[2] J. Matoušeck, *Range searching with efficient hierarchical cuttings*, Symposium on Computational Geometry, 1992.

[3] B. Chazelle, *Lower bounds on the complexity of multidimensional searching*, 27th Annual Symposium on Foundations of Computer Science, 27-29 October 1986.

[4] B. Chazelle, H. Edelsbrunner, *Linear space data structures for two types of range search*, Discrete and Computational Geometry, 113-126, 1987.

[5] B. Chazelle, M. Sharir, E. Welzl, *Quasi-optimal upper bounds for simplex range searching and new zone theorems*, Algorithmica, 1992.