Godzilla Onions: A Skit and Applet to Explain Euclidean Half-Plane Fractional Cascading

Richard Berger
Ursinus College Computer Science, Collegeville, PA, USA

Vincent Ha
Ursinus College Computer Science, Collegeville, PA, USA

David Kratz
Ursinus College Computer Science, Collegeville, PA, USA

Michael Lin
Ursinus College Computer Science, Collegeville, PA, USA

Jeremy Moyer
Ursinus College Computer Science, Collegeville, PA, USA

Christopher J. Tralie
Ursinus College Mathematics And Computer Science, Collegeville, PA, USA

Abstract

We provide a skit and an applet to illustrate fractional cascading in the context of half-plane range search for points in the Euclidean plane, which takes $O(\log N + h)$ output-sensitive time. In the video, a group of news anchors struggles to find the correct data structure to efficiently send out an early warning to the residents of Philadelphia who will be overtaken by a marching line of Godzillas. After exploring several options, the group eventually settles on onions and fractional cascading, only to discover that they themselves are in the line of fire! In the applet, we show step by step details of preprocessing to build the onions with fractional cascading and the subsequent query of the "Godzilla line" against the onion layers. Our video skit and applet can be found at `https://ctralie.github.io/GodzillaOnions/`

2012 ACM Subject Classification Human-centered computing → Visualization toolkits; Theory of computation → Randomness, geometry and discrete structures

Keywords and phrases convex hulls, onions, fractional cascading, visualization, d3

Digital Object Identifier 10.4230/LIPIcs.SoCG.2023.62

Category Media Exposition

Supplementary Material *Software (Source Code)*: `https://ctralie.github.io/GodzillaOnions/`

1 Background

Given N points X in the Euclidean plane and a query line ℓ, a naive algorithm to determine the points in X above ℓ would be to check each $x_i \in X$ in turn. However, this approach takes $O(N)$ time.

Alternatively, one can obtain an output-sensitive algorithm if one preprocesses X into an "onions" data structure with a nested sequence of convex hulls from the outside to the inside [2]. For N_L onion layers, refer to the i^{th} layer as L_i, where i indexes the layers in the order in which they are constructed, so L_0 is the outermost layer. If there are h points above the line, each onion layer L_i can be queried with binary search in $O(\log N)$ time to find the line segment with the closest slope to that of ℓ, and this line segment contains the furthest point in that layer from ℓ. If this point is above the line in layer L_i, one can walk to the left and to the right to gather all points above the line in this layer. Overall, this takes

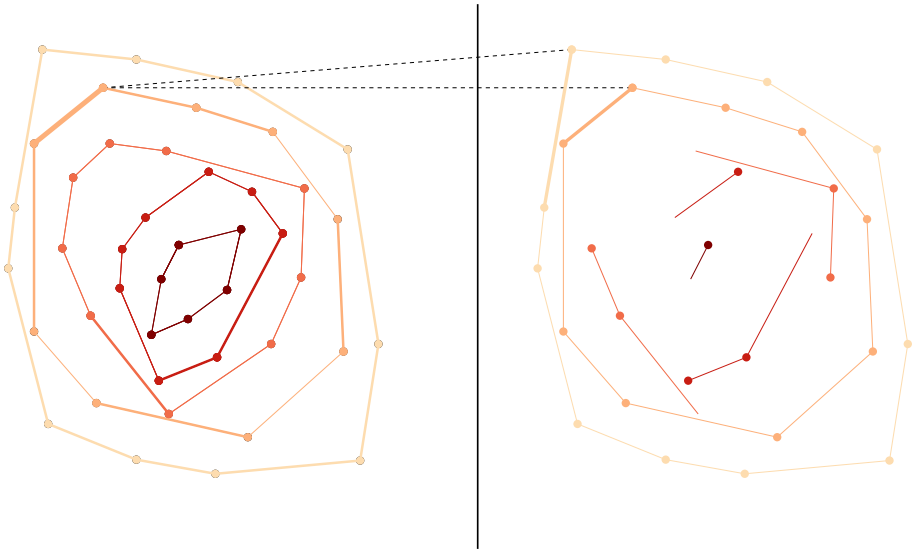

Figure 1 A screenshot from our Javascript d3 interface showing how each point in M_0 has a pointer to a point in L_0 (the outer layer, duplicated on the right) and M_1 (the second from the outer cascaded layer, duplicated on the right) with the closest slope. Preprocessing and storing such pointers allows quick traversal through the structure. Note also: the thickest line segment on the left shows the particular line segment whose pointers the app is highlighting. The middle thick line segments on the left show L_0 and M_1 in the context of the whole onion.

$O(N_l \log N + h)$ time. However, as we note in our video, there may be too many layers; in particular, N_L is $\Theta(N^{2/3})$ for points distributed uniformly and independently at random within any bounded 2D region that contains a disc [3]. The ensuing $O(N^{2/3} \log N + h)$ algorithm is still marginally better than the brute force $O(N)$ approach, but one can do better.

A superior output-sensitive algorithm relies on a more involved onion-based preprocessed data structure that uses fractional cascading [4]. In addition to storing the layers L_i, one constructs parallel layers M_i. The last layer $M_{N_L} = L_{N_L}$. From there, one iteratively constructs M_i as the union of L_i and every other element of M_{i+1}, *sorted by slope*. Each element in M_i also stores a pointer to the points in L_i and M_{i+1} with the nearest slope. After preprocessing, one starts querying the fractionally cascaded onions by first searching for the point in the outer M layer M_0 with the slope closest to ℓ using binary search. Since M_i only takes every other point in M_{i+1}, the number of points in $\cup_i M_i$ is $O(N)$, so the binary search query on M_0 takes $O(\log N)$ time. From there, one follows the pointer to L_0 to extract all points at that layer that are above ℓ. Then, one follows the pointer to M_1 and continues the process until getting to a layer with no points above ℓ. The preprocessed pointers allow one to walk from layer to points above the line in the subsequent layer in constant time, so the overall process takes only $O(\log N + h)$ time.

2 Applet Details

One of our major contributions is an applet to incrementally construct this rather intricate data structure. To that end, we create an interactive applet using d3 in Javascript [1] to construct and query and onions data structure on top of user selected points. We first show the process of constructing the L_i layers, which we color code. Then, we show how each M_i

is created by merging by slope L_i and every other element of M_{i+1}. Once that is finished, we show a few examples of pointers from M_0 to L_0 and M_1 (Figure 1). Finally, the user queries a "Godzilla line" ℓ, and the applet shows how to incrementally walk through the layers and follow the pointers to obtain all of the points above ℓ.

References

1 Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. D^3 data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, 2011.

2 Bernard Chazelle. On the convex layers of a planar set. *IEEE Transactions on Information Theory*, 31(4):509–517, 1985.

3 Ketan Dalal. Counting the onion. *Random Structures & Algorithms*, 24(2):155–165, 2004.

4 Leonidas Guibas and Bernard Chazelle. Fractional cascading: I. a data structuring technique. *Algorithmica*, 1:133–162, 1986.