A Small Improvement in the Walking Algorithm for Point Location in a Triangulation

Ivana Kolingerová*

Abstract

The paper shows a simple technique which saves some edge tests in the walking algorithm for point location. The walking technique does not achieve the logarithmic per point complexity of the location-data-structure-based methods but does not need any auxiliary data structure and is very simple to implement, therefore, it is very popular in practice. The suggested idea did not bring a substantial improvement in our tests but it is very simple and there is an open door to further more substantial improvement in the future research.

1 Introduction

Point location in a triangulation is a very frequent task. Most effective solutions use hierarchical data structures, such as a DAG [1], [5], a skip list [10], a quadtree, buckets [9], a data structure with a random sampling [7], [2] or a uniform grid [8], [11]. These structures are very effective and bring a complexity $O(\log\ n)$ per point where n is the total number of points in the triangulation. However, their disadvantage is a memory consumption, which, although linear in E^2, still can bring a substantial limitation to the program usefulness as the data sets processed today are very huge. Also, implementation effort for most of these data structures may be nontrivial. Therefore, practical programmers turn very often to a 'pragmatic' solution, represented by the walking type of location algorithm, where the point is located by traversing from one triangle to another according to some kind of test of the point position against the triangle boundaries. Such an approach is less effective, bringing $O(n^{1/3})$ up to $O(n^{1/2})$ per one point location but no extra location data structures are needed and so no extra memory is consumed [4], [3], [6].

There are several walking algorithms, differing in the strategy how to find the next triangle from the current one, the most effective one seems to be the so-called remembering stochastic walk. This paper shows a very simple improvement of this walking algorithm that may save about 8% of edge tests. Theoretical improvement could be up to 50% but we have not been able to achieve this efficiency yet.

Section 2 explains the remembering stochastic walk and the suggested improvement. Section 3 explains experiments and results and section 4 concludes the paper.

2 Remembering stochastic walk and the suggested improvement

Walking strategy generally means that from some starting triangle, we inspect the triangles one after another, traveling over the edge into that triangle neighbour which looks the best choice to approach the triangle containing the query point q. The starting triangle can be chosen in random, or it is the triangle visited most recently, or by brute force choice from a random subset of triangles according to their distances from the located point. The walking can traverse all the triangles intersected by the line segment originating at a vertex of the starting triangle and ending at the query point - this is the so-called *straight walk*. Or the transfer can be divided into 2 axes, approaching first in one and then in the other axis, so-called *orthogonal walk*. The *visibility walk* uses an orientation test of the triangle edge and the query point, a bad sign of the orientation test reveals which edge to cross to continue the search. As the visibility walk can cycle for a non-Delaunay triangulation, it can be more properly implemented in a randomized version as the stochastic walk - randomization means here that the choice of the first triangle edge to be tested is random, which prevents an infinite loop. The last modification is the *remembering stochastic walk* which remembers over which edge we came into the triangle. It does not test this edge because we already know the result of this test, after all. All these walking strategies and their comparison in E^2 and E^3 can be found in [3].

The algorithm of remembering stochastic walk is given in Alg.1 (adapted from [3]). For further acceleration, it is good to combine the walking algorithm with some preprocessing where a randomly selected set of triangle vertices from the triangulation is tested on distance to the query point and the smallest distance from a vertex to the query point chooses the starting triangle [6]. A proper size of this set is $O(n^{1/3})$ triangle vertices where n is the total number

*Department of Computer Science and Engineering, University of West Bohemia, Pilsen, Czech Republic, kolinger@kiv.zcu.cz

of vertices in the triangulation. Although the distance tests are expensive, still this technique brings improvement both in the number of edge tests and in the total runtime.

Remembering stochastic walk (t,q)

```
// traverses the triangulation T
// from the triangle t to the query point q
// using the remembering stochastic walk
begin
  previous := t ; found := false;
  while not found do
  begin
    e := random edge from t;
    p := the vertex of t not contained in e;
    nb := neighbour (t over e);
    if (nb is not equal to previous) and
      (q on the other side of e than p) then
      begin
        previous := t; t := nb
      end
    else
      begin
        e := next edge of t;
        p := the vertex of t not contained in e;
        nb := neighbour (t over e);
        if (nb is not equal to previous) and
          (q on the other side of e than p) then
          begin
            previous := t; t:=nb
          end
        else
          begin
            e := next edge of t;
            p := the vertex of t not contained in e;
            nb := neighbour (t over e);
            if (nb is not equal to previous) and
            (q on the other side of e than p) then
              begin
                previous := t;
                t := nb
              end
            else found := true
          end
      end
  end
end
// now t contains q
```

Remembering stochastic walk

Algorithm 1

An average number of edge tests per triangle is 1.5 because in each triangle (with the exception of the first one), we either find a bad orientation at the first attempt and go to the triangle sharing this edge, or we have to test one more edge and the orientation test either sends us further, or acknowledges that the query point is inside this triangle. The starting triangle may need one more test but this is not important for the average value.

An improvement of this algorithm is straightforward: if we want to be better, we should test only one edge per triangle. Is it possible? With the exception of the first and the last triangle, yes: we know the result of test of the edge which led us to the current triangle. If we test one more edge, we can either get the result 'go to the neighbouring triangle sharing this edge', or the result 'do not go over this edge, you should stay inside or go to the triangle over the third triangle edge'. We cannot decide for sure between the 2 latter possibilities without one more test, but the key point is that the answer 'stay inside' is valid for the last triangle only and is highly improbable in comparison with the answer 'leave the triangle over the non-tested edge', as we usually test many triangles before we come to the triangle containing the query point. In this way, we can save as much as one edge test per triangle.

Black point of this improvement is that with one edge test per triangle, we are not able to recognize that we are already in the triangle containing the query point, and we could continue in the walking for ever without recognizing the end. Therefore, we can use the 'fast walking' strategy only for some number of steps which we expect it is usually necessary to approach the goal triangle, and then continue more slowly by the remembering stochastic walk, until we find the goal triangle.

This combined strategy would work perfectly if we knew exactly how many steps the algorithm will need to find the goal triangle but this is not the case. According to [6] and our experiments, the number of visited triangles is $O(n^{1/3})$ in average, so we can expect about this number of steps to get into the goal triangle. Bad news is that dispersion of the number of visited triangles per query point is very high, therefore, in many cases we stop the faster search too early and in many cases too late, which increases the total number of the tests. The whole algorithm which we called *fast walk* is given as Alg.2 and results can be seen in the next section.

Fast walk (t,q)

```
// traverses the triangulation T
// from the triangle t to the query point q
// using fast walk
// nsteps is the number of steps for the fast walk,
// usually O(n^1/3)
// where n is the number of vertices in the triangulation
begin
  previous:= t;
  for i := 1 to nsteps do
    begin
      e := random edge from t;
      nb := neighbour (t over e);
      while nb is equal to previous do
        begin
          e := next edge of t; nb:= neighbour (t over e);
        end ;
      p := the vertex of t not contained in e;
      if (q on the same side of e as p) then
        begin
          e := next edge of t; nb := neighbour (t over e)
        end;
      previous:= t ; t :=nb
    end
end
// we do not know whether t contains q,
// so we must continue by the remembering
// stochastic walk
Remembering stochastic walk (t, q);
// now t contains q
```

Fast walk

Algorithm 2

3 Experiments and results

We tested the program as a part of an incremental insertion algorithm for the Delaunay triangulation, programmed in Delphi, under MS Windows operating system. In order to have results independent on the platform, we measured the number of edge tests necessary to construct the triangulation for up to 5 million of points. We measured uniformly distributed points, but also points in clusters and other types of data. We show the uniform data as they provided the average values. The suggested improvement is compared to the remembering stochastic walk. Both algorithms were programmed with the use of preprocessing according to [6] as the version with preprocessing is quicker and it tests fewer edges in all cases. Results for the straight walk are not given as they were worse than those of the remembering walk in all cases.

Figure 1 shows how many triangle tests are necessary in the remembering stochastic walk for each query point within the triangulation process. The average number of visited triangles can be approximated by $2n^{1/3}$ to $2.15n^{1/3}$ for $n = 1000$ up to 5 million but the dispersion is high, see an example in Fig.1. Due to this dispersion, we used less fast walk steps than would correspond to this function. (We should stress that in the application for triangulation construction, n is different for each inserted point and is equal to the number of points already inserted into the triangulation, not to the total number of points to be inserted.)

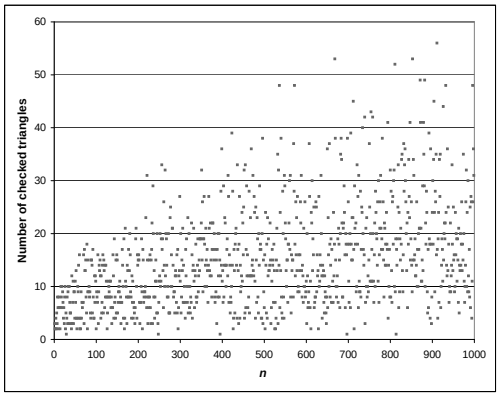

Figure 1: The number of checked triangles in the remembering walk during construction of the Delaunay triangulation on 1000 points.

Table 1 shows comparison between the number of tested edges in the remembering stochastic walk and in the fast walk. We tested various functions derived from $O(n^{1/3})$ for derivation of the number of steps for fast walk (*nsteps* in Alg.2). The best results of the fast walk presented in this table were obtained by the number of fast walk steps equal to $1.15n^{1/3}$. Improvement is much lower than expected, only about 8% of the edge tests. The reason for this smaller success than hoped is the already mentioned dispersion in the number of traversed edges: the expected number of triangles that can be fast walked varies so much that a decrease in the number of tested edges is nearly compensated by inspecting some triangles in vain because the goal triangle was not recognized in time; in some cases the situation is the opposite, fast walk is stopped too early. Unfortunately, there is no way how to find the correct number of tests for the given case in advance. Maybe there exists some simple, elegant and brilliant idea how to get closer to the theoretical bound 50% improvement but we have not come to it yet.

4 Conclusion

The paper presents a fast walk, a simple modification of the remembering stochastic walk algorithm for locating a point in a triangulation. The method is very

simple and does not consume any extra memory for data structure, improvement achieved in the tests is not significant at present but the simplicity of the idea may inspire further research to achieve a theoretically possible 50% improvement against the original remembering walk algorithm.

n	Rem.st.wlak	Fast walk	Savings
10^4	469 462	445 097	5.47
5.10^4	4 028 040	3 759 661	7.14
10^5	10 084 883	9 372 859	7.60
5.10^5	85 496 066	79 068 466	8.13
10^6	214 296 506	197 641 298	8.43
5.10^6	1 816 818 470	1 670 737 631	8.74

Table 1: Number of edge tests for the remembering stochastic walk and for the fast walk

5 Acknowledgement

The author would like to thank to ing. J.Kohout for inspiration to inquire into the walking algorithms.

References

[1] M. de Berg, M. van Kreveld, M. Overmars and O. Schwarzkopf. *Computational geometry, algorithms and applications*, Berlin Heidelberg: Springer, 1997.

[2] O. Devillers. Improved incremental randomized Delaunay triangulation. *Proceedings of the 14th Annual Symposium on Computational Geometry 1998*, 106-115, 2001.

[3] O. Devillers, S. Pion and M. Teillaud. Walking in a triangulation. *Proceedings of the 17th Annual Symposium on Computational Geometry 2001*, 106-114, 2001.

[4] L.J. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Trans Graphics*, 4(2):75-123, 1985.

[5] I. Kolingerová and B. Žalik. Improvements to randomized incremental Delaunay insertion. *Computers & Graphics*, 26, 477-490, 2002.

[6] E.P. Mücke, I. Saias and B. Zhu. Fast randomized point location without preprocessing in two- and three-dimensional Delaunay triangulations. *Proceedings of the 12th Annual Symposium on Computational Geometry 1996*, 274-283, 1996.

[7] K. Mulmuley. Randomized multidimensional search trees: dynamic sampling. *Proceedings of the 7th Annual Symposium on Computational Geometry 1991*, 121-131, 1991.

[8] S.W. Sloan A fast algorithm for constructing Delaunay triangulations in the plane. *Adv Eng Software*, 9(1):34-55, 1987.

[9] P. Su and R.L.S. Drysdale. A comparison of sequential Delaunay triangulation algorithms. *Proceedings of the 11th Annual Symposium on Computational Geometry 1995*, 61-70, 1995.

[10] M. Zadravec and B. Žalik. An almost distribution-independent incremental Delaunay triangulation algorithm. *The Visual Computer,* 21(6):384-396, 2005.

[11] B. Žalik and I. Kolingerová. An incremental construction algorithm for Delaunay triangulation using the nearest-point paradigm. *Int.J. Geographical Information Science*, 17(2):119-138, 2003.