Vertex Removal in Two Dimensional Delaunay Triangulation: Speed-up by Low Degrees Optimization

Olivier Devillers

Vertex Removal in Two-Dimensional Delaunay Triangulation: Speed-up by Low Degrees Optimization

Olivier Devillers *

Computational Geometry: Theory and Applications 44:169–177

Abstract

The theoretical complexity of vertex removal in a Delaunay triangulation is often given in terms of the degree d of the removed point, with usual results $O(d)$, $O(d \log d)$, or $O(d^2)$. In fact, the asymptotic complexity is of poor interest since d is usually quite small. In this paper we carefully design code for small degrees $3 \leq d \leq 7$, it improves the global behavior of the removal for random points by more than 45%.

1 Introduction

The Delaunay triangulation is one of the most famous structures in computational geometry, and its construction has been studied in numerous papers. In this paper, we are interested in the practical efficiency of the removal procedure in a two-dimensional triangulation.

Several algorithms exist for this problem whose complexities are usually given in terms of the degree d of the removed vertex. Few algorithms have linear $O(d)$ complexity, let us mention the algorithm by Aggarwal et al. [1] which provides a quite complicated deterministic solution and the simpler solution by Chew [3] whose complexity is randomized and is still a bit overkill for small degrees. An $O(d \log d)$ complexity is achieved by Devillers [4] using a predicate of degree higher than that of the usual incircle test, another $O(d \log d)$ solution computes the triangulation of the neighbors of the removed points and glues the relevant part of this small triangulation in the hole arising from the removal. In practice, $O(d^2)$ solutions are often preferred for their simplicity, the two most common are the boundary completion and the diagonal flipping [4].

*INRIA Sophia Antipolis — Méditerranée, France. http://www-sop.inria.fr/members/Olivier.Devillers/.

Contribution

Deletion in Delaunay triangulation seems to be a solved problem, but the devil is in the details of the implementation. We show that a careful implementation of low degree cases allows to drastically reduce the deletion time for these degrees, improving the global performance by almost a factor of 2. By the way, a modification of the diagonal flipping algorithm, in the same spirit, improves its efficiency by 20%.

After a brief review of general purpose deletion algorithms, the specialized versions for low degrees are detailed from the algorithmic and implementation points of view. Experiments are given to support design choices.

The implementation was done using CGAL [2], and will be integrated in CGAL 3.7.

2 Removal algorithms for any degree

2.1 Boundary completion

This algorithm first removes the faces incident to the removed vertex, creating a hole in the triangulation, then a queue is initialized with all the edges of the hole boundary. Given an edge in the queue the new face incident to that edge inside the hole is found in linear time and the hole is updated. A simplified treatment is done for the removal of a degree 3 vertex. The hole does not need to remain simple nor connected during the process.

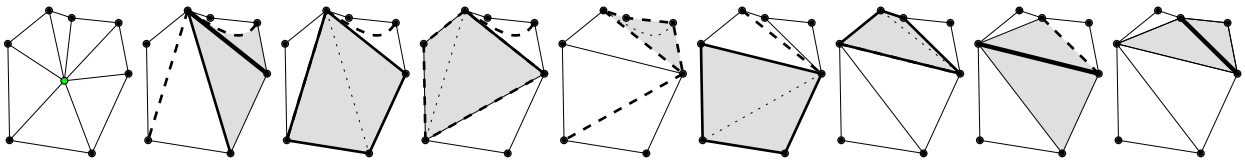
Discussion

The theoretical complexity of such an algorithm is clearly quadratic in the degree d of the removed vertex since each of the $d-4$ new edges is obtained in $O(d)$ time. More precisely, in the worst case, the i^{th} edge popped from the queue is processed needs $d-i-2$ incircle tests, thus an estimation of the complexity is $\sum_{i=1}^{d-3} d-i-2 = \frac{1}{2}(d-3)(d-2)$. The complexity can be better if the hole is split in a balanced manner by the constructed triangles. The current code for vertex deletion in CGAL 3.5 uses this algorithm, the code is about 350 lines and its running time will be given in Section 5.

2.2 Flipping

We implement another algorithm which consists in triangulating the hole in an arbitrary manner and then flipping edges to restore the Delaunay property. The initial triangulation of the hole is combinatorial (that is its embedding may not be planar if the hole is not convex) but the geometric validity comes at the end with the Delaunay property.

Let d be the degree of the removed vertex and $v_0, v_1, \ldots v_{d-1}$ its neighbors. An initial triangulation is obtained by just linking v_0 to all vertices v_j, $2 \leq j \leq d-2$ on the hole boundary. These $d-3$ edges may be non locally Delaunay, and may even be outside the hole to be triangulated, thus they are marked to be checked for local Delaunay validity. Then the usual Delaunay flipping algorithm is used, checking edges and flipping them if necessary,



Dashed edges are marked to be checked, the current checked edge is either dotted or fat depending on the result.

Figure 1: A flipping sequence.

with the slight difference that when a non local Delaunay edge is flipped, the four edges of the quadrilateral are marked to be checked only if they are not edges of the hole ($v_i v_{i+1}$) (see Figure 1).

Discussion

The theoretical worst case complexity is also $\frac{1}{2}(d-3)(d-2) = O(d^2)$, but the behavior may be better if the initial triangulation is not too bad. Our implementation uses about 100 lines. Running times are close to those of the boundary completion technique. An improved variant of the flipping will be presented in Section 4.

2.3 Triangulate and sew

Another possibility is to first compute the Delaunay triangulation of the neighbors of the vertex to be removed. Let DT_{big} be the initial Delaunay triangulation and DT_{small} be the Delaunay triangulation of the neighbors of the removed point computed by your favorite method. Then the hole boundary is present in both triangulations, and the final triangulation uses DT_{big} outside the hole and DT_{small} inside. Indeed, if we considered the *flower* of the removed point, that is the union of the disks circumscribing the triangles of DT_{big} incident it, the circle circumscribing a triangle of DT_{small} inside the hole is inside the flower and thus cannot enclose any remaining point after the deletion (see Figure 2).

Thus for each edge e of the hole we create a new neighborhood relationship between the triangle of DT_{small} incident to e inside the hole and the triangle of DT_{big} incident to e outside the hole. It just remains to throw away all useless triangles to get the new triangulation.

Discussion

The complexity of the construction of DT_{small} depends on the chosen algorithm for Delaunay triangulation, say $O(d \log d)$ using an optimal one. If we use a randomized algorithm and count only the number of incircle tests, we get $10d - O(1)$. The complexity of the sewing part is $O(d)$, thus the overall complexity is $O(d \log d)$. This technique is currently used in CGAL 3.5 for the three-dimensional triangulation, but in two dimensions, simply inserting

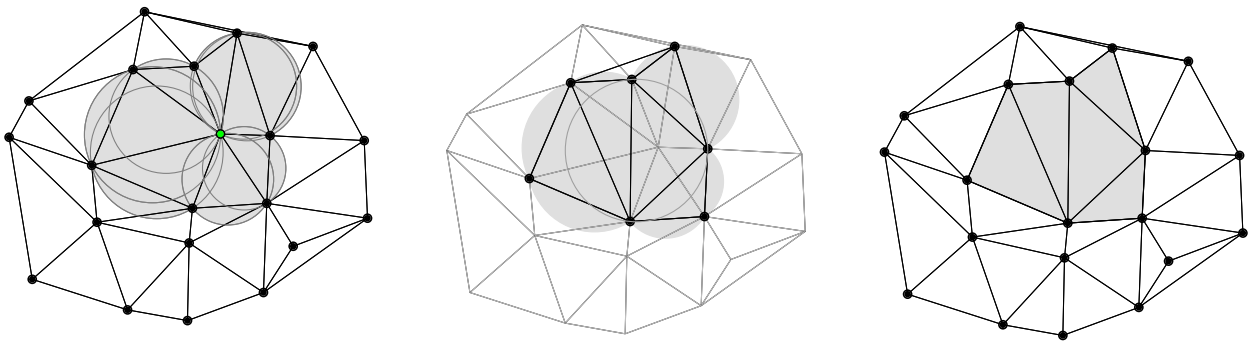


Figure 2: left) DT_{big} and the *flower* of the removed point, middle) DT_{small} and a circumscribing circle, right) final triangulation.

the points in a small triangulation is already much more expensive than other deletion methods.

2.4 Ear queue

An ear is a triangle created inside the hole using two consecutive edges along the hole boundary. To each candidate ear (a candidate ear is defined by a pair of consecutive edges on the hole boundary) is associated a priority which is the power of the removed point with respect to the circle circumscribing the ear. It is proven [4] that the ear with the smallest priority belongs to the Delaunay triangulation. Thus a priority queue of ears can be constructed and the smallest priority ears can be processed in turn.

Discussion

The complexity is $O(d \log d)$. Existing comparisons [4] between this technique and the flipping algorithm did not show a significative improvement in terms of running time. Furthermore, it requires the comparison of the power of the removed point with different ears which involves a new geometric predicate.

2.5 Randomized reinsertion

Chew's randomized algorithm [3] is a variation of the "triangulate and sew" technique. The method used to triangulate the neighbors is modified, basically using the information of the order of the neighbors around the deleted vertex to reduce the location time to be constant.

Discussion

The expected number of incircle tests is $5d - O(1)$. As for "triangulate and sew", the effective cost of constructing a small triangulation is prohibitive, even when not taking into account the sewing part.

3 Optimizing small degrees

The asymptotic complexity of the removal algorithm is not really relevant if the degree is small, which is often the case since the average degree is only 6. Thus specialized, carefully optimized, versions of the deletion algorithm for low degrees can be implemented. This idea was already used for degrees up to 5 [4], we pursue that idea up to degree 7.

3.1 Algorithms for degree 3 to 7

Degree 3

Clearly if the degree is three, the hole is a triangle and the new triangulation is obtained by replacing the three incident triangles by the new one.

Degree 4

If the degree is 4 the hole is a quadrilateral. A single incircle test has to be done to decide which diagonal of the quadrilateral has to be used to triangulate the hole. Notice that if the quadrilateral is not convex, the incircle test can be avoided since a single triangulation is possible; but we prefer to save the convexity test (which is often positive) and directly perform the incircle test which will choose the right triangulation anyway.

Degree 5

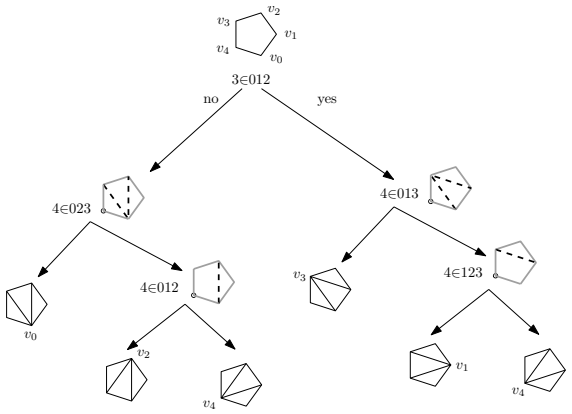


Figure 3: Decision tree for degree 5 deletion.
$i \in jkl$ is a short notation for v_i lying inside the circle passing through $v_j v_k v_l$. Positive answer goes to the right subtree. Dashed edges are edges of the triangulation of v_0, v_1, v_2, v_3 that are not yet certified for the whole triangulation.

For degree 5, the situation remains quite simple. The hole is a pentagon, and as for the quadrilateral case, we do not care about the convexity of the hole, since the incircle tests

on non convex quadrilaterals yield to the right decision anyway. We build a decision tree performing several incircle tests on the neighbors $v_0v_1v_2v_3v_4$ of the deleted vertex to decide what is the right way of triangulating the pentagon. More precisely, we first decide what is the Delaunay triangulation of $v_0v_1v_2v_3$ and then insert v_4 by testing it with respect to the circumcircle of the triangle incident to edge v_3v_0 and to the other triangle if relevant (Figure 3). The result always consists in choosing a vertex v_i and linking it to all other vertices.

Degree 6

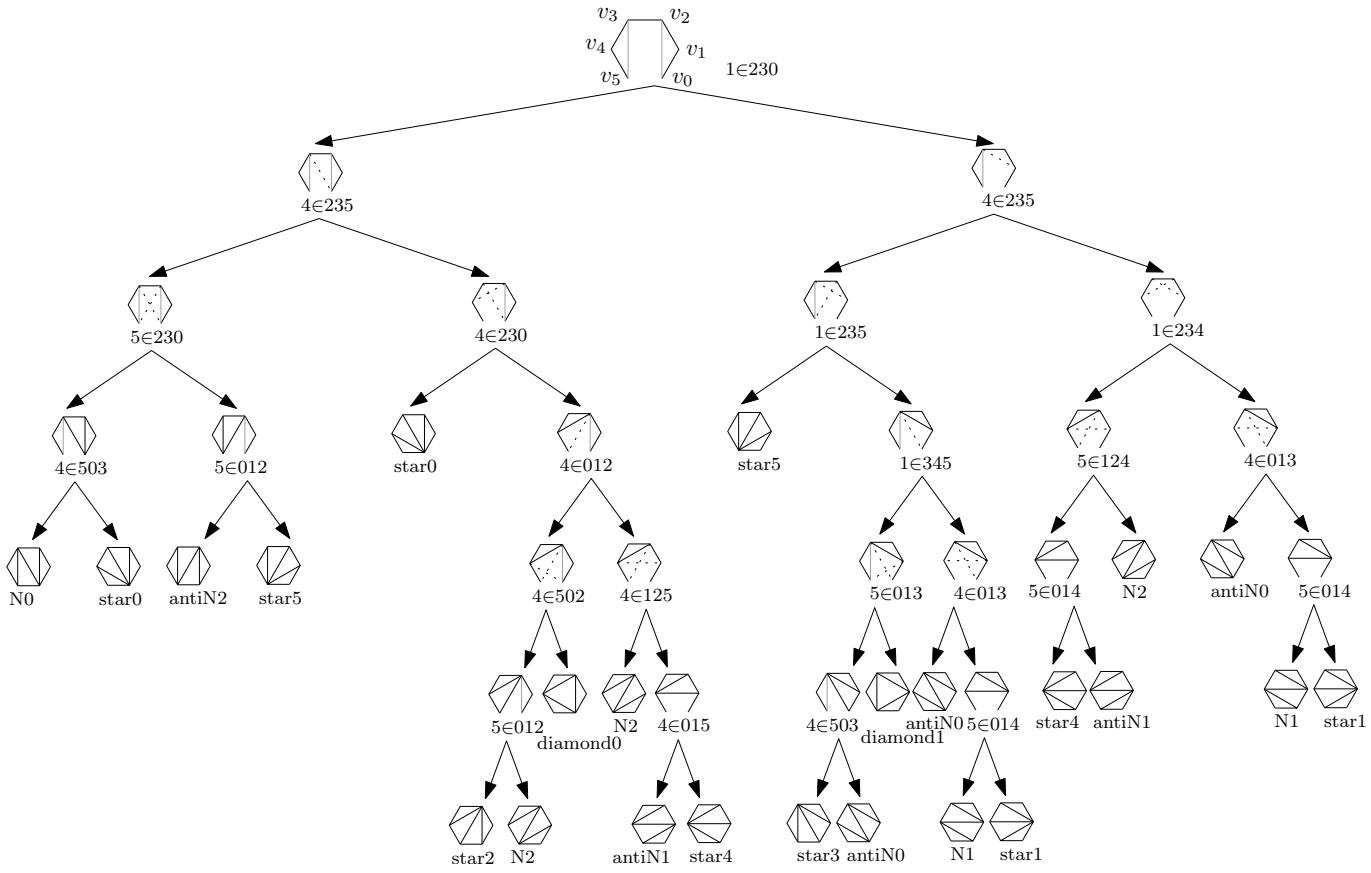


Figure 4: Decision tree for degree 6 deletion.
$i \in jkl$ is a short notation for v_i lying inside the circle passing through $v_jv_kv_l$, positive answer goes to the right subtree.

For degree 6, things start to be a little bit more involved, since the number of possible triangulations of the hexagonal hole[1] is 14. Note that these 14 triangulations have in fact four

[1]Catalan numbers give the number of possible triangulations of a simple polygon

different configurations up to a rotation on the vertex indices, we call these configurations: star, diamond, N, and antiN.

We build a decision tree using the classical divide and conquer Delaunay algorithm [5]. Since we have only six vertices, the triangulations of the two subsets of three points is trivial, and the algorithm reduces to the conquer part. The tree is described in Figure 4, at each node a triangulation is drawn, solid black edges are certified to be Delaunay, gray edges are Delaunay edges of the right or left part that are not yet certified neither destroyed, dotted edges are the next candidates to link a right to a left vertex, the final results are the leaves of the decision tree.

Degree 7

As for degree 6, the decision tree (Figure 5) is constructed using the divide and conquer algorithm. While the right part is triangulated (using the first incircle test), a classical conquer phase is performed, either top-down or bottom-up depending on the right triangulation (only the top-down merge is described on Figure 5). For degree 7 there are 42 different triangulations of a heptagon that can be organized in 6 different configurations.

Figure 5: Decision tree for degree 7 deletion.
$i \in jkl$ is a short notation for v_i lying inside the circle passing through $v_j v_k v_l$,
positive answer goes to the right subtree.

3.2 Implementation

The decision tree is really coded without modifying the triangulation, then, when the entire triangulation of the hole is known, a suitable procedure is called to actually modify the triangulation.

We give below some code of the implementation to be integrated in CGAL. The `remove` function computes the degree, stores the faces and vertices incident to the removed vertex and calls a specialized function for the relevant degree:

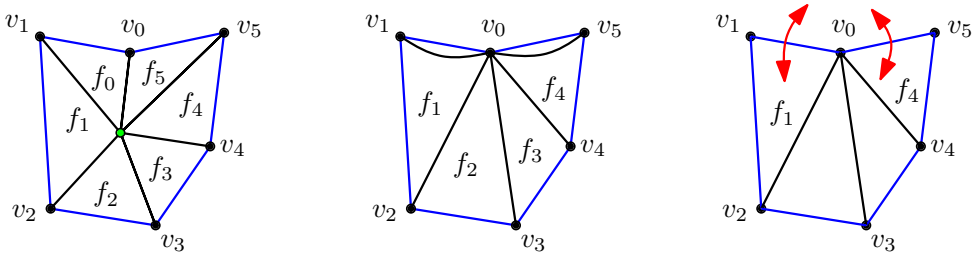
```
template < class Gt, class Tds > void Delaunay_triangulation_2<Gt,Tds>::remove(Vertex_handle v)
{
  if ( this->dimension() <= 1) { Triangulation::remove(v); return; }
  int d=0;
  static int maxd=30;
  static std::vector<Face_handle> f(maxd);
  static std::vector<int> i(maxd);
  static std::vector<Vertex_handle> w(maxd);
  f[0] = v->face();
  do{
    i[d] = f[d]->index(v);
    w[d] = f[d]->vertex( ccw(i[d]) );
    w[d]->set_face( f[d]->neighbor(i[d]));//do no longer bother about set_face
    ++d;    if ( d==maxd) { maxd *=2; f.resize(maxd); w.resize(maxd); i.resize(maxd);}
    f[d] = f[d-1]->neighbor( ccw(i[d-1]) );
  }while(f[d]!=f[0]);

  switch (d) {
  case 3:    remove_degree3(v,f,w,i);    break;
  case 4:    remove_degree4(v,f,w,i);    break;
  case 5:    remove_degree5(v,f,w,i);    break;
  case 6:    remove_degree6(v,f,w,i);    break;
  case 7:    remove_degree7(v,f,w,i);    break;
  default:   remove_degree_d(v,f,w,i,d); break;
  }
}
```

The `remove_degree6` function implements the decision tree of Figure 4 and calls a function to triangulate the hole with the right configuration, the possible rotations are encoded through the order of the arguments of the configuration function:

```
template < class Gt, class Tds > void Delaunay_triangulation_2<Gt,Tds>::remove_degree6
(Vertex_handle v, std::vector<Face_handle> &f, std::vector<Vertex_handle> &w, std::vector<int> &i)
{
  // removing a degree 6 vertex
  if(incircle(1,2,3,0,f,w,i)){
    if(incircle(4,2,3,5,f,w,i)){
      if(incircle(1,2,3,4,f,w,i)){
        if(incircle(4,0,1,3,f,w,i)){
          if(incircle(5,0,1,4,f,w,i)){
            remove_degree6_star(v,f[1],f[2],f[3],f[4],f[5],f[0],w[1],w[2],w[3],w[4],w[5],w[0],i[1],i[2],i[3],i[4],i[5],i[0]);  //star1
          }else{
            remove_degree6_N(v,f[1],f[2],f[3],f[4],f[5],f[0],w[1],w[2],w[3],w[4],w[5],w[0],i[1],i[2],i[3],i[4],i[5],i[0]);     //N1
          }}else{
          remove_degree6_antiN(v,f[0],f[1],f[2],f[3],f[4],f[5],w[0],w[1],w[2],w[3],w[4],w[5],i[0],i[1],i[2],i[3],i[4],i[5]);   //antiN0
        }}else{
...
...
...
}
```

A main difference with a general algorithm is that not even one temporary triangle is created to be destroyed few steps after. Furthermore, the modification of the triangulation is done reusing existing triangles in a way that minimizes the number of modifications. We give here the function `remove_degree6_star` that triangulates a hexagon by linking one vertex to all other vertices (see Figure 6) four triangles are reused, only one vertex has to be modified

in each of them, and only two neighborhood relations need to be changed; two triangles and the removed vertex are deleted.



pentagon to be *stared* from v_0

modify one vertex in f_1, f_2, f_3, and f_4

create neighbor relations from f_1 and f_4 to outside the hole (to previous neighbors of f_0 and f_5).

Figure 6: Modifying the triangulation.

```
template < class Gt, class Tds > inline void Delaunay_triangulation_2<Gt,Tds>::remove_degree6_star
(Vertex_handle &v,
 Face_handle &  f0, Face_handle &  f1, Face_handle &  f2, Face_handle &  f3, Face_handle &  f4, Face_handle &  f5,
 Vertex_handle &v0, Vertex_handle &v1, Vertex_handle &v2, Vertex_handle &v3, Vertex_handle &v4, Vertex_handle &v5,
 int i0, int i1, int i2, int i3, int i4, int i5 )
{ // removing a degree 6 vertex, staring from v0
  Face_handle nn;
  f1->set_vertex( i1, v0) ;  // f1 = v1v2v0
  f2->set_vertex( i2, v0) ;  // f2 = v2v3v0
  f3->set_vertex( i3, v0) ;  // f3 = v3v4v0
  f4->set_vertex( i4, v0) ;  // f4 = v4v5v0
  nn = f0->neighbor( i0 );  this->tds().set_adjacency(f1, cw(i1), nn, nn->index(f0));
  nn = f5->neighbor( i5 );  this->tds().set_adjacency(f4, ccw(i4), nn,  nn->index(f5));
  this->tds().delete_face(f0);  this->tds().delete_face(f5);  this->tds().delete_vertex(v);
}
```

3.3 Remarks on the size of the decision tree

The following table summarizes some characteristics of the decision trees depending on the degree d. The number of results is the number of possible triangulations of a d-gon while the number of configurations is the number of triangulations up to rotation of the vertices. Each different configuration yields to a different triangulation function that modifies the triangulation.

$\lceil \log_2 \sharp results \rceil$ is the height of an optimal decision tree, but it is doubtful that it is possible to actually construct such an optimal tree that relies only on a single incircle test for each decision. Anyway, comparing the number of results to the number of leaves and comparing the height to the optimal height gives an idea of the quality of our tree.

For degree 8, an estimation of the construction of the tree based on the divide and conquer scheme gives an estimation of a tree with about 500 leaves and 500 decision nodes going to 1500 lines of code to implement it; and an estimation of the work needed to triangulate an

octagon gives about 50 lines of code, to be multiplied by the 19 different configurations. Altogether, 2500 lines of code seems a reasonable estimation of an implementation of a similar scheme. For higher degree, it does not seem really tractable to go further.

degree	3	4	5	6	7	$8^{\heartsuit}$	9	10	11
♯ configurations$^{\spadesuit}$	1	1	1	4	6	19	49	150	442
♯ triangulations$^{\clubsuit}$	1	2	5	14	42	132	429	1430	4862
♯ leaves	1	2	6	24	130	$\simeq$*500*			
$\lceil \log_2 \sharp \text{ triangulations} \rceil$	0	1	3	4	6	8	9	11	13
tree height	0	1	3	6	10	$\simeq$*14*			
♯ lines of code	30	40	90	280	700	$\simeq$*2500*			

♡ not implemented. The sizes of the tree and the code are estimated
♠ http://www.research.att.com/∼njas/sequences/ [6]
♣ Catalan number

4 Flip from pentagons

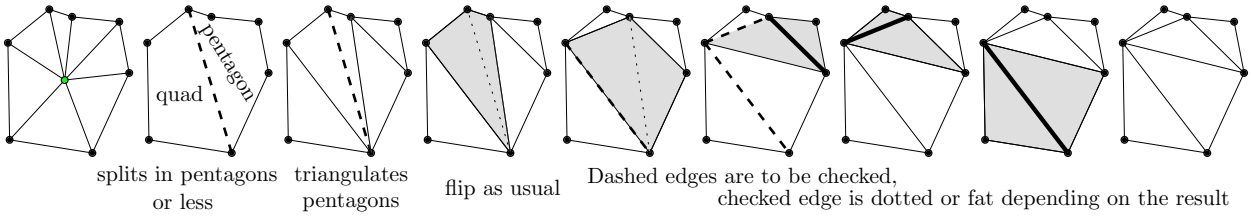


Figure 7: A flipping sequence.

A similar idea can be applied to the flipping algorithm, instead of having all the edges marked to be checked for local Delaunay property after the first triangulation, it is possible to make some local optimization first. Namely, the initial triangulation of the hole is obtained by adding edges v_0v_{1+3j}, $1 \leq j \leq \left\lceil \frac{d}{3} - 1 \right\rceil$ and triangulate the pentagons, using a small decision tree, with two locally Delaunay edges, then the flipping algorithm starts with only $\left\lceil \frac{d}{3} - 1 \right\rceil$ edges to be checked (see Figure 7).

Discussion

The theoretical worst case complexity of this algorithm of course remains $O(d^2)$. Our implementation uses about 400 lines mostly devoted to the pentagons initialization. Performances are about 20% better than the standard flipping algorithm (see Section 5).

5 Benchmarks

Comparisons between
— boundary completion,

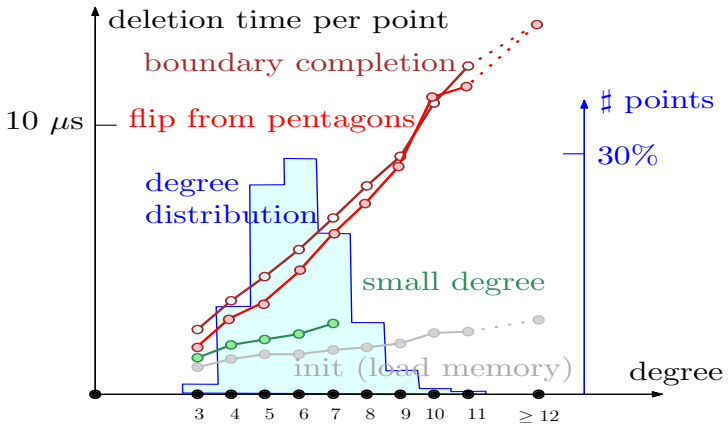


Figure 8: Deletion time per degree and per method

— flipping from pentagons and
— specialized versions for degrees less than 7 and flipping from pentagons for higher degrees have been made on random point sets.[2] All the vertices of a Delaunay triangulation of 10,000,000 points are deleted in a random order.

The total deletion process is split in two parts. The initialization part essentially circulates around the vertex to be removed to compute its degree and collect the incident faces and vertices.
— 15 seconds is the time needed for all initializations.
The second part decides what is the new triangulation and actually modifies the triangulation. Depending on the method we get the following times:
— 40 seconds for boundary completion (thus 40+15= 55 seconds in total)
— 32 seconds for flipping from pentagons (thus 32+15= 47 seconds in total)
— 14 seconds using the specialized versions for small degrees (thus 14+15= 29 seconds in total).

This initialization time appears relatively high when compared to the time to retriangulate a hole, but one of the main effects of initialization is that all relevant data is loaded into the cache, ready for the deletion. The fact that this initialization time is mainly due to memory usage is confirmed by running the initialization twice, the second initialization goes 6 times faster (because all useful data is already in cache memory). Another confirmation is obtained by running the boundary completion *with* and *without* the initialization step (which is not needed by this method) the total running time appears to be the same (running *without init* avoids the heavy degree computation, but has to visit the relevant stuff in the memory at some points anyway). A third confirmation is obtained by removing the vertices in a non random order: if the removed vertex is a neighbor of a neighbor of the

[2]Experiments have been done on a 2.33 GHz processor with 16 GByte RAM Operating system is Linux-FC10 with CGAL 3.5. Code was compiled with gcc 4.3.2 in release mode. Detailed results and precisions on the experimental conditions are given in INRIA Research Report 7104.

previously removed vertex, then some relevant information is already in the cache and the initialization is actually much faster (only 3 μs).

Figure 8 presents the running time per degree, including initialization, for the three methods; it also presents the time for initialization and the distribution of the degree of the removed points. Vertices with "high" degrees are less than 13% of the removed points and most of them have degree 8. Thus the asymptotic complexity in the degree of the removal algorithm seems of poor importance except if some adversary decides to always remove high degree vertices. The running time, if we consider the initialization time as unavoidable, indicates a really big improvement with the small degrees specialized versions. Flipping from pentagons appears to be a bit better than the boundary completion.

6 Conclusion

By a special treatment of the removal of vertices of degree 3 to 7, we improve the average removal time by almost a factor of 2. Dealing with the degree 8 case may continue to decrease the running time by about 10%, but requires to double the code size and it has not been implemented for the moment. The implementation for higher degrees may need to implement a code generator to build the decision tree.

This kind of optimization cannot be applied in three dimensions. In two dimensions we have applied a special treatment to 5 special sizes of the hole (triangle, quadrilateral, pentagon, hexagon, and heptagon). In three dimensions the combinatorics is much more intricate: a configuration of the hole is a polyhedron, but the degree of the removed point is higher than in 2D, and for a given degree there are several possible polyhedra, thus instead of 5 special configurations we should treat thousands of them. More than 6,000 configurations are needed to go up to degree 13, and about 800,000 to go up to degree 17, each one having many possible tetrahedralizations. This seems completely unrealistic to have millions of lines of code, even if they are generated automatically.

Acknowledgments This work is partly supported by ANR grant Triangles (ANR-07-BLAN-0319). Author thanks Sylvain Pion for fruitful discussions and his help in preparing this paper.

References

[1] A. Aggarwal, L. J. Guibas, J. Saxe, and P. W. Shor. A linear-time algorithm for computing the Voronoi diagram of a convex polygon. *Discrete Comput. Geom.*, 4(6):591–604, 1989.

[2] CGAL Editorial Board. *CGAL User and Reference Manual*, 3.5 edition, 2009. www.cgal.org.

[3] L. P. Chew. Building Voronoi diagrams for convex polygons in linear expected time. Technical Report PCS-TR90-147, Dept. Math. Comput. Sci., Dartmouth College, Hanover, NH, 1990.

[4] Olivier Devillers. On deletion in Delaunay triangulation. *Internat. J. Comput. Geom. Appl.*, 12:193–205, 2002.

[5] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 3rd edition, October 1990.

[6] N. J. A. Sloane and Simon Plouffe. *The Encyclopedia of Integer Sequences*. Academic Press, 1995.