



# Vertex Removal in Two Dimensional Delaunay Triangulation: Asymptotic Complexity is Pointless

Olivier Devillers

## ► To cite this version:

Olivier Devillers. Vertex Removal in Two Dimensional Delaunay Triangulation: Asymptotic Complexity is Pointless. [Research Report] RR-7104, INRIA. 2009, pp.15. inria-00433107

**HAL Id: inria-00433107**

**<https://inria.hal.science/inria-00433107>**

Submitted on 18 Nov 2009

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

***Vertex Removal in Two Dimensional Delaunay  
Triangulation:  
Asymptotic Complexity is Pointless***

Olivier Devillers

**N° 7104**

Novembre 2009

Thème SYM

 ***rapport  
de recherche***



# Vertex Removal in Two Dimensional Delaunay Triangulation: Asymptotic Complexity is Pointless

Olivier Devillers

Thème SYM — Systèmes symboliques  
Équipe-Projet Geometrica

Rapport de recherche n° 7104 — Novembre 2009 — 16 pages

**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 a factor of 2.

**Key-words:** Complexity, Convex hull, Quadrics, Cylinders, Projection

This work is partly supported by ANR grant Triangles (ANR-07-BLAN-0319).

# Supprimer dans une triangulation de Delaunay 2D: la complexité asymptotique n'est pas pertinente

**Résumé :** La complexité théorique de la suppression d'un sommet dans une triangulation de Delaunay est en général exprimée en fonction du degré  $d$  du point supprimé, avec des résultats tels que  $O(d)$ ,  $O(d \log d)$  ou  $O(d^2)$ . En fait, la complexité asymptotique présente un intérêt faible car  $d$  est plutôt petit. Dans cet article, nous expliquons la conception de code spécialisé pour la suppression de sommets de petit degré  $3 \leq d \leq 7$ , ce code améliore le comportement global des suppressions d'un facteur 2 dans le cas de points aléatoires.

**Mots-clés :** Complexité, Enveloppe convexe, Quadriques, Cylindres, Projection

## 1 Introduction

The Delaunay triangulation is one of the most famous structure 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 complexity is 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 higher degree than 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].

### Contribution

Deletion in Delaunay triangulation seems to be a solved problem, but the devil is in the details of 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 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 point of view. Experiments are given to support design choices.

The implementation was done using CGAL [2], and is submitted to the CGAL editorial board for integration in the library.

## 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. 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.

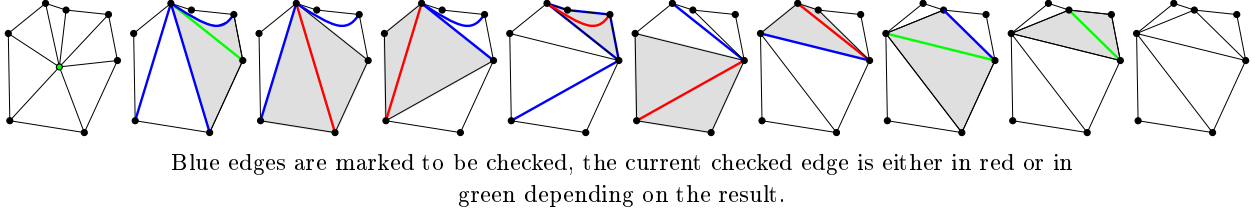


Figure 1: A flipping sequence.

## 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  $v_0, v_1, \dots, v_{d-1}$  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, 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  $O(d^2)$ , 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 points computed by your favorite method. Then the hole boundary is present in both triangulations. 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 is the one of the construction of  $DT_{small}$  plus  $O(d)$  to sew both triangulations, say an overall complexity of  $O(d \log d)$  using an optimal Delaunay construction. This technique is currently used in CGAL 3.5 for the three

dimensional triangulation, but in two dimensions, simply inserting 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 advantage 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

Randomized complexity is  $O(d)$ . 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 asymptotical 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 here pursuing 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

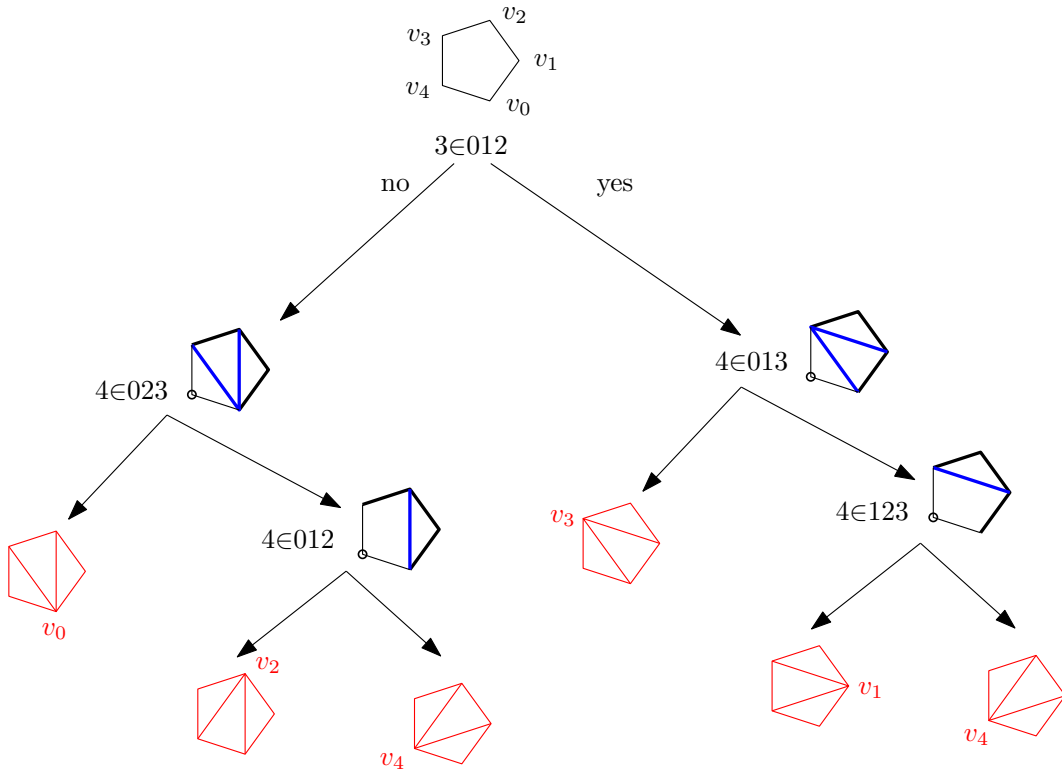


Figure 2: Decision tree for degree 5 deletion.

$i \in jkl$  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 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

## Degree 6

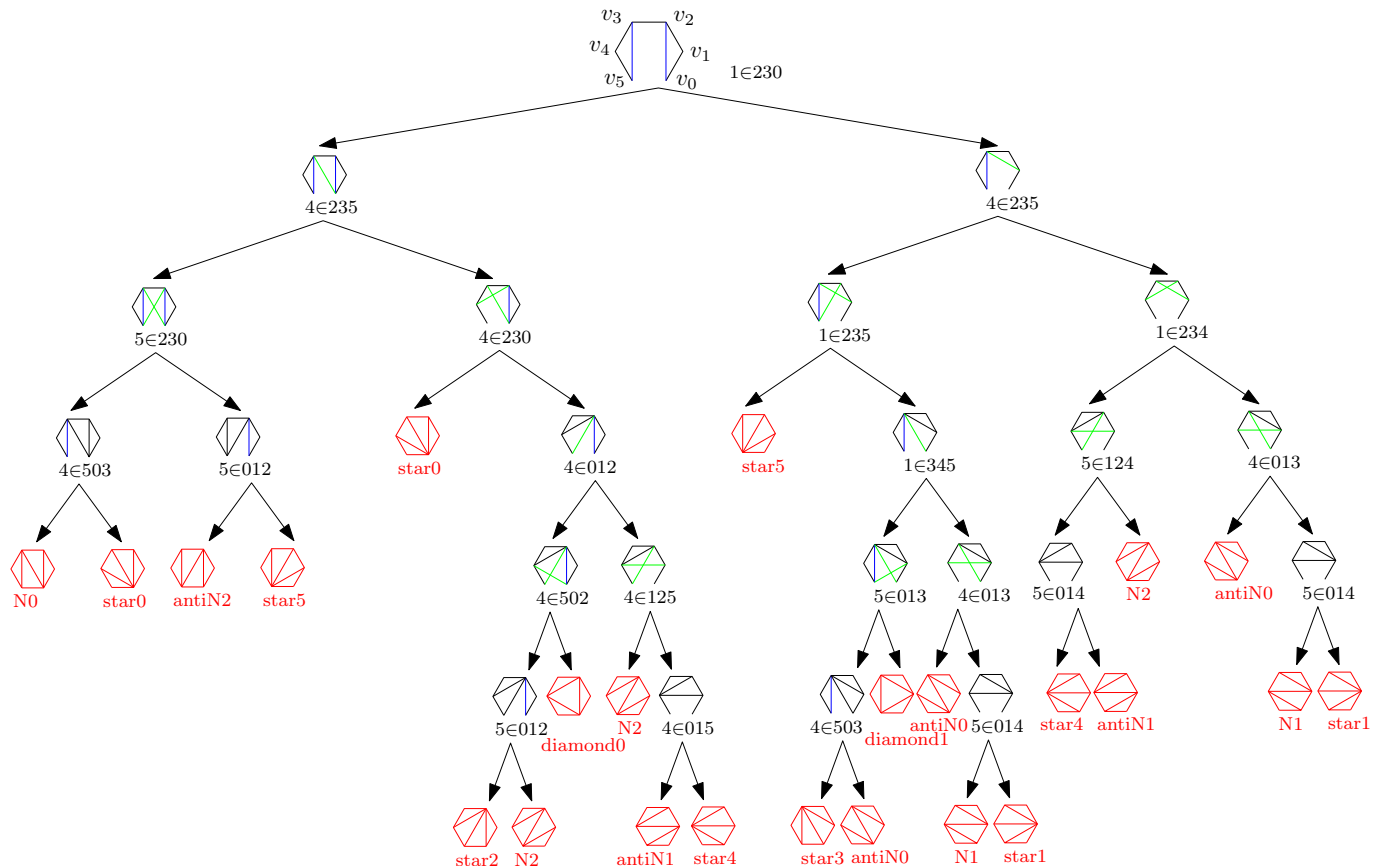


Figure 3: Decision tree for degree 6 deletion.

For degree 6, things start to be a little bit more involved, since the number of possible triangulations of the hexagonal hole<sup>1</sup> is 14. Note that these 14 configurations have in fact four different shapes up to a rotation on the vertex indices, we call these shapes: star, diamond, N, and antiN.

We build a decision tree using the zig-zag merge of the classical divide and conquer Delaunay algorithm [5]. The tree is described in Figure 3, at each node a triangulation is drawn, black edges are certified to be Delaunay, blue edges are Delaunay edges of the right or left part that are not yet certified neither destroyed, green edges are the next candidates to link a right to a left vertex, and red configurations are the results of the decision tree.

<sup>1</sup>Catalan numbers give the number of possible triangulations of a simple polygon

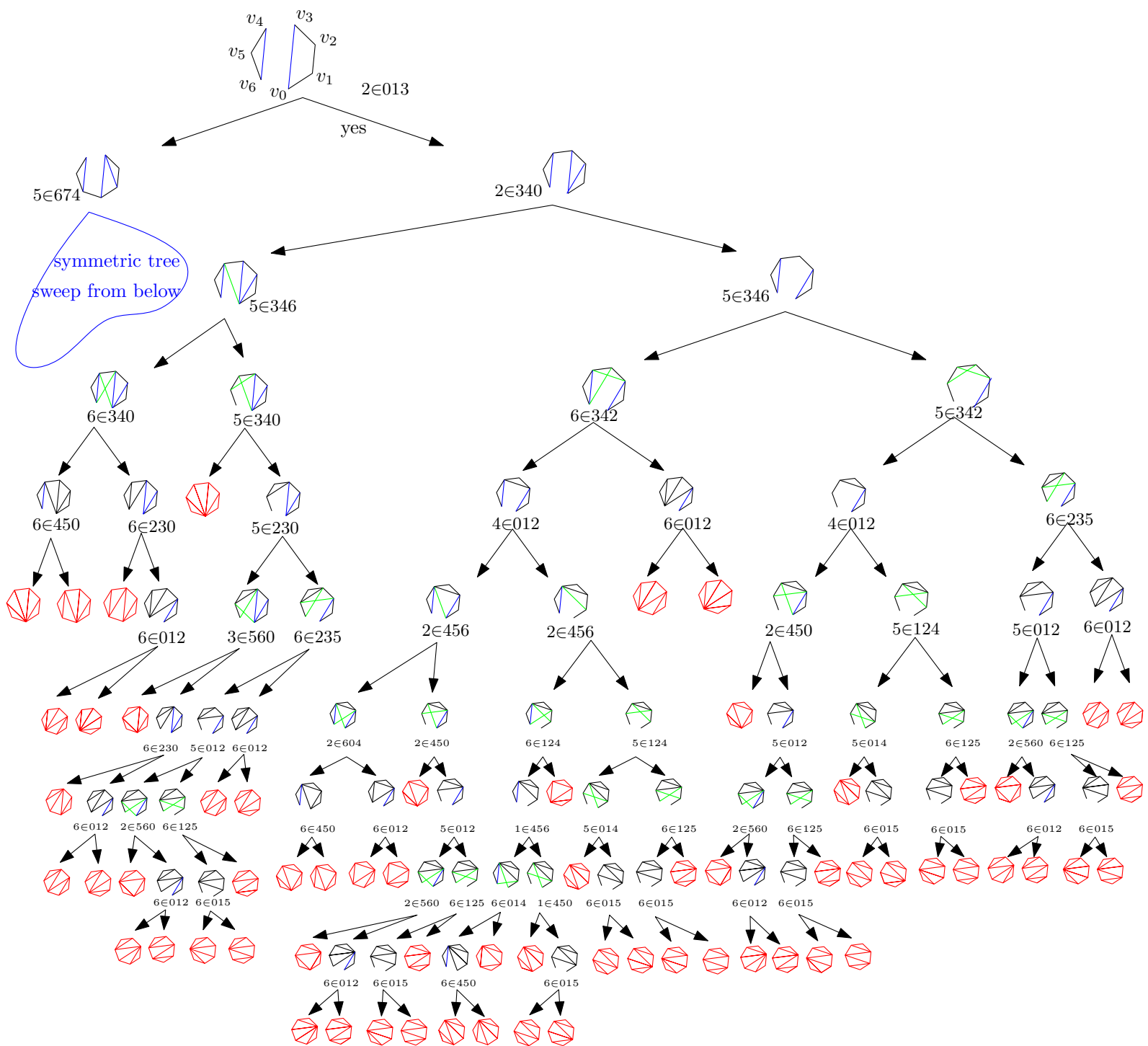


Figure 4: Decision tree for degree 7 deletion.

### Degree 7

As for degree 6, the decision tree (Figure 4) is constructed using the divide and conquer algorithm. While the right part is triangulated (using the first incircle test), a classical zig-zag merge is performed, either top-down or bottom-up depending on the right triangulation (only the top-down merge is described on Figure 4). For degree 7 there are 42 different triangulations of an heptagon that can be organized in 6 different shapes.

### 3.2 Remarks on the 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 (see Appendix A). 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. Figure 5 describes the triangulation of an hexagon: 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.

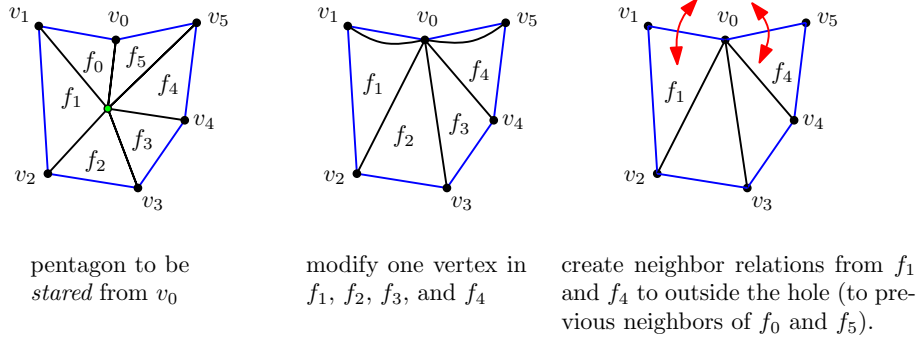


Figure 5: Modifying the triangulation.

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 shapes is the number of triangulations up to rotation of the vertices. Each different shape yields to a different triangulation function that modifies the triangulation.

$\lceil \log_2 \#results \rceil$  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 shapes. 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 <sup>♡</sup>	9	10	11
# shapes <sup>♣</sup>	1	1	1	4	6	19	49	150	442
# results <sup>♣</sup>	1	2	5	14	42	132	429	1430	4862
# leaves	1	2	6	24	130	$\simeq 500$			
$\lceil \log_2 \#results \rceil$	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

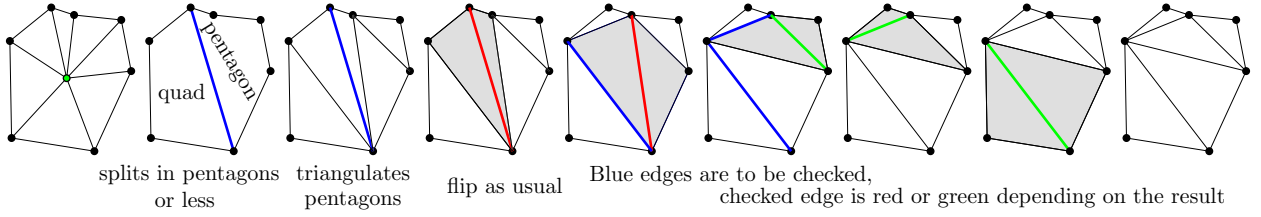


Figure 6: A flipping sequence.

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  $v_0v_{1+3j}$ ,  $1 \leq j \leq \lceil \frac{d}{3} - 1 \rceil$  and triangulate the pentagons, using a small decision tree, with two locally Delaunay edges, then the flipping algorithm starts with only  $\lceil \frac{d}{3} - 1 \rceil$  edges to be checked (see Figure 6).

### 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,
- flipping from pentagons and
- specialized versions for degrees less than 7 and flipping from pentagons for higher degrees

have been tested on random point sets.<sup>2</sup> 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 consists in exploring the incident faces and vertices of the removed vertex to compute its degree, this part is pretty expensive since it loads in cache memory some objects that were not recently accessed; if the initialization is done twice, then the second run is about 6 times faster which confirms that the time is mainly devoted to memory access and not to computations:

— 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).

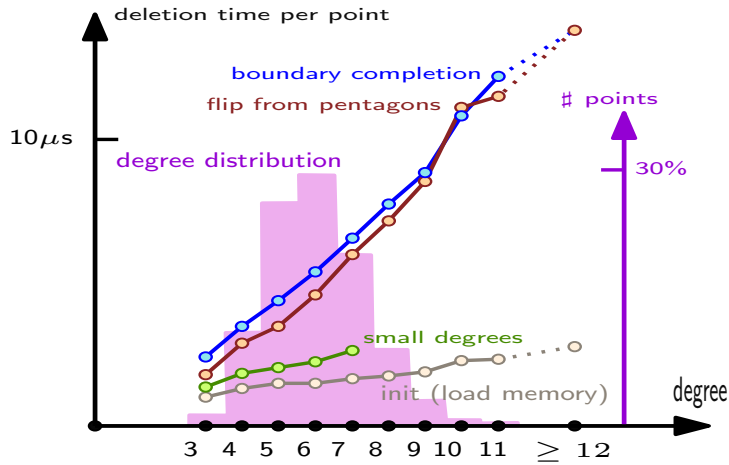


Figure 7: Deletion time per degree and per method

Figure 7 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 a factor of 2. Dealing with the degree 8 case may

<sup>2</sup> Detailed results and precisions on the experimental conditions are given in Appendix B.

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 configurations 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** 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](http://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.

## Appendix A: Example code

As an example, we give below some code. 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 3 and calls a function to triangulate the hole with the right shape, the possible rotations are encoded through the order of the arguments of the shape 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(4,2,3,0,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]);
                    }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]);
                    }
                }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]);
                }else{
                    if(incircle(5,1,2,4,f,w,i)){
                        remove_degree6_N(v,f[2],f[3],f[4],f[5],f[0],f[1],w[2],w[3],w[4],w[5],w[0],w[1],i[2],i[3],i[4],i[5],i[0],i[1]);
                    }else{
                        if(incircle(5,0,1,4,f,w,i)){
                            remove_degree6_antiN(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]);
                        }else{
                            remove_degree6_star(v,f[4],f[5],f[0],f[1],f[2],f[3],w[4],w[5],w[0],w[1],w[2],w[3],i[4],i[5],i[0],i[1],i[2],i[3]);
                        }
                    }
                }
            }
        }
    }
}

```

We give here the function that triangulates an hexagon by linking one vertex to all other vertices (see Figure 5):

```

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, starting 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->tlds().set_adjacency(f1, cw(i1), nn, nn->index(f0));
    nn = f5->neighbor( i5 );
    this->tlds().set_adjacency(f4, ccw(i4), nn, nn->index(f5));
    this->tlds().delete_face(f0);
    this->tlds().delete_face(f5);
    this->tlds().delete_vertex(v);
}

```

## Appendix B: Benchmarks

### Platform

Experiments have been done on a 2.33 GHz processor with 16 GByte RAM Operating system is Linux-FC10 with CGAL3.5. Code was compiled with gcc 4.3.2 in release mode.



Since getting reliable running time, especially for small functions that take few micro-seconds, is a difficult task, we have used three different tools to control the coherence of our results:

- the Linux `time` command for the total time of the code
- the `CGAL::Timer` to get the time of the deletion phase.
- the `CGAL::Profile_timer` and `CGAL::Profile_counter` tools (with a little adaptation) to get detailed numbers per function and per degree.

### Main experiment

The main code inserts 10,000,000 random points in a square, stores the `Vertex_handles` in a vector, shuffles that vector and then deletes all the vertices one by one (except the last three vertices that needs special treatment anyway).

We compare the methods “boundary completion”, “flip from pentagons”, and the optimized versions for “small degree”. Namely we distinguish:

*BC* : test the degree and call Boundary completion,

*BC'*: call directly Boundary completion,

*FP* : test the degree and call Flip from pentagons, and

*SD<sub>d</sub>* : test the degree and call Flip from pentagons for degree strictly higher than *d*, specialized version otherwise.

### Results

First we run *SD<sub>7</sub>* on several trials for the random point set to check the dependence of the results on a given trial. The table below shows that results are reasonably independent from a given trial and thus in the sequel, we will perform a single trial for each version of the code. We also can check on that table, the coherence between the CGAL timer and the Linux time command.

<i>SD<sub>7</sub></i>	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Trial 6	Trial 7	Trial 8
# calls degree 3	112,491	112,232	112,498	112,469	112,672	113,091	112,655	113,196
# calls degree 4	1,068,761	1,070,453	1,070,855	1,071,678	1,068,201	1,069,972	1,069,692	1,070,030
# calls degree 5	2,596,740	2,597,126	2,595,118	2,595,899	2,595,588	2,595,597	2,595,947	2,594,358
# calls degree 6	2,947,407	2,945,804	2,946,768	2,944,563	2,946,049	2,945,969	2,945,232	2,944,019
# calls degree 7	1,983,195	1,984,034	1,985,613	1,985,043	1,986,997	1,984,395	1,986,841	1,986,780
# calls degree 8	900,481	899,876	898,906	900,905	901,027	901,342	899,762	901,444
# calls degree 9	296,856	296,822	296,358	296,247	296,004	296,224	296,636	296,338
# calls degree 10	75,278	74,738	75,176	74,390	74,901	74,621	74,510	75,005
# calls degree 11	15,181	15,267	15,119	15,154	14,952	15,201	15,151	15,214
# calls degree $\geq 12$	3,607	3,645	3,586	3,649	3,606	3,585	3,571	3,613
	Trial 9	Trial 10	Trial 11	Trial 12	Trial 13	Trial 14	Trial 15	Trial 16
CGAL Timer insert <sup>◊</sup>	14.62	14.77	14.49	14.41	14.76	14.55	14.61	14.72
CGAL Timer remove	28.67	28.22	28.04	27.70	27.79	27.91	27.79	28.19
CGAL Timer total*	45.17	44.84	44.38	43.94	44.41	44.31	44.25	44.76
user time <sup>◇</sup> (seconds)	45.20	44.89	44.43	43.98	44.45	44.35	44.28	44.80

<sup>◊</sup> insertion using spatial sort (points are sorted along a space filling curve and inserted in that order in Delaunay triangulation)

\* point generator + insert + collecting and shuffling handles + remove (compiled without profiling)

<sup>◇</sup> from Linux command

Profiling allows to get detailed time per degree, given in the table below.

degree	# calls	<i>BC</i>		<i>FP</i>		<i>SD<sub>7</sub></i>		init	
		time (s)	/ call ( $\mu$ s)	time (s)	/ call ( $\mu$ s)	time (s)	/call ( $\mu$ s)	time (s)	/call ( $\mu$ s)
3	1.1%	0.15	1.4	0.09	0.8	0.04	0.36	0.11	1.0
4	10.7%	2.2	2.1	1.6	1.5	0.47	0.44	1.5	1.4
5	26.0%	7.6	2.9	5.2	2.0	1.4	0.54	3.8	1.5
6	29.4%	11.5	3.9	9.2	3.1	2.2	0.75	4.3	1.5
7	19.8%	9.7	4.9	8.5	4.3	1.9	0.96	3.3	1.7
8	9.0%	5.4	6.0	4.9	5.4			1.6	1.8
9	3.0%	2.1	7.0	2.0	6.7			0.57	1.9
10	0.7%	0.6	8.6	0.62	8.9			0.16	2.3
11	0.15%	0.15	10.0	0.14	9.3			0.034	2.3
$\geq 12$	0.036%	0.04	11.1	0.04	11.1			0.010	2.8
All degrees removal time									
Profiler (without init)		10,000,000	40	4.0	32	14	1.4	15	1.5
Profiler (adding init)		10,000,000	55	5.5	47	29	2.9		
Timer (with init, no profiler)		10,000,000	53	5.3	46	28	2.8		

Notice than the last two lines indicate that the time used to profile is below 2 seconds and cannot invalidate the significance of the results. To confirm that the profiling do not influence the results we run some supplementary experiments without profiling which allows to observe that the time difference between  $SD_d$  and  $SD_{d+1}$  remains the same:






method	<i>BC</i>	<i>BC'</i>	<i>FP</i>	<i>SD<sub>3</sub></i>	<i>SD<sub>4</sub></i>	<i>SD<sub>5</sub></i>	<i>SD<sub>6</sub></i>	<i>SD<sub>7</sub></i>
CGAL timer remove	53	53	46	46	45	41	34	28
				<i>FP - SD<sub>3</sub></i>	<i>SD<sub>3</sub> - SD<sub>4</sub></i>	<i>SD<sub>4</sub> - SD<sub>5</sub></i>	<i>SD<sub>5</sub> - SD<sub>6</sub></i>	<i>SD<sub>6</sub> - SD<sub>7</sub></i>
CGAL timer remove				0	1	4	7	6
CGAL Profiler				0.05	1.1	3.8	7.0	6.6

The initialization time consists essentially in circulating around the vertex to be removed to compute its degree and collect on the fly the incident faces and vertices. The initialization time appears to be relatively high compared to the time to retriangulate the hole, but in fact one of the main effects of this initialization is to load in the cache memory all relevant data which benefit to the deletion afterwards. 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) and that *BC* and *BC'* have close running time (running *BC'* avoids the heavy degree computation, but has to visit the relevant stuff in the memory anyway).

The expensive cost of loading relevant objects in the cache memory is of course linked to the random order used for deleting the vertices. The table below shows results, where the next vertex to remove is a neighbor of a neighbor of the previous one. As expected, the initialization time becomes much smaller since relevant stuff is already in the cache. The non randomness of the choice of the deleted vertex of course modifies the degree distribution (20% of degree  $\geq 8$ ), and the configuration distribution, thus the running times are different. Anyway the overall hierarchy between the 3 methods remains the same.

degree	<i>BC</i>	<i>FP</i>	<i>SD<sub>7</sub></i>	init
	time / call ( $\mu$ s)			
3	0.9	0.6	0.2	0.2
4	1.6	1.1	0.2	0.7
5	2.4	1.7	0.3	0.2
6	3.2	2.7	0.5	0.3
7	4.1	3.8	0.7	0.3
8	5.1	4.8		0.4
9	6.1	6.2		0.4
10	7.2	7.6		0.5
11	8.4	8.9		0.5
$\geq 12$	10.5	11.6		0.6
total time (s)				
Profiler (without init)	38	32	17	3

Another interesting point is to look at the distribution of the different results to triangulate a  $d$ -gon. We measured for  $d = 7$  how many times we get the 6 different shapes and with which rotation. The idea of designing the decision tree to minimize the depth of the most likely configurations should improve the complexity by very little. Anyway, it is surprising to observe significant differences between the different rotations of a given configuration, even if the choice of the vertex 0 around the deleted vertex is not random, but depends of the context of previous insertions and deletions in the neighborhood.

	0	1	2	3	4	5	6	total
	3%	5%	1%	3%	4%	3%	3%	22%
	2%	2%	1%	1%	3%	1%	1%	12%
	1%	4%	1%	1%	3%	2%	1%	14%
	3%	2%	1%	2%	3%	2%	1%	14%
	2%	5%	1%	3%	2%	4%	2%	19%
	2%	4%	1%	3%	3%	2%	4%	19%



---

Centre de recherche INRIA Sophia Antipolis – Méditerranée  
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex  
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier  
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq  
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique  
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex  
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex  
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex  
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex

---

Éditeur  
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)  
<http://www.inria.fr>  
ISSN 0249-6399