Vers des algorithmes dynamiques randomisés en géométrie algorithmique

Monique Teillaud

ORSAY
numéro d'ordre : 1866

UNIVERSITE DE PARIS-SUD
CENTRE D'ORSAY

THESE

présentée
pour obtenir

Le GRADE de DOCTEUR EN SCIENCES
DE L'UNIVERSITE PARIS XI ORSAY
par

Monique Teillaud

SUJET :

Vers des algorithmes dynamiques randomisés en Géométrie algorithmique

soutenue le 10 décembre 1991 devant la Commission d'examen

MM.	Dominique Gouyou-Beauchamps	Président
	Jean-Marc Steyaert	Rapporteur
	Jean Berstel	
	Jean-Daniel Boissonnat	
	Claude Puech	
MM.	Bernard Chazelle	Rapporteur
	Kurt Mehlhorn	Rapporteur

Note - Août 1997

Une version révisée de cette thèse a été publiée en 1993 par *Springer Verlag*, dans la collection *Lecture Notes in Computer Science*, numéro 758, sous le titre *Towards Dynamic Randomized Algori thms in Computational Geometry*.

Voir aussi le rapport interne INRIA numéro 1727 (juillet 1992)
`ftp://ftp.inria.fr/INRIA/publication/publi-ps-gz/RR/RR-1727.ps.gz`

Remerciements*

Je remercie Jean-Daniel Boissonnat, qui a initié cette thèse, il y a bien longtemps, de l'intérêt évident qu'il a témoigné pour ces travaux, et de sa participation enthousiaste, depuis mon arrivée à l'INRIA Sophia-Antipolis. J'y profite avec plaisir de l'atmosphère stimulante qu'il sait entretenir dans le projet PRISME ; j'y bénéficie aussi du confort assez exceptionnel qu'offre l'INRIA à ses chercheurs.

Bernard Chazelle et Kurt Mehlhorn, deux noms célèbrissimes de la Géométrie algorithmique, m'ont fait l'honneur d'être rapporteurs, et je leur en adresse tous mes remerciements.

Je suis très honorée que Dominique Gouyou-Beauchamps préside ce jury ; merci également de m'avoir accompagnée dans des difficultés administratives en cascade...

Je suis reconnaissante à Jean-Marc Steyaert, d'avoir accepté sans hésitation la charge d'être rapporteur de ce travail, d'avoir lu le manuscrit dans le détail, et de faire partie du jury.

J'apprécie la présence de Jean Berstel dans ce jury, en souvenir de nos discussions dans ce bureau partagé pendant quelques mois.

Je remercie Claude Puech, pour sa gentillesse discrète, et pour m'avoir accueillie en transit au Laboratoire d'Informatique de l'Ecole Normale Supérieure, lorsque j'en ai eu besoin. Je lui exprime aussi ici ma reconnaissance, pour m'avoir conseillée et soutenue jadis, à la sortie de l'ENSJF ; pour la confiance qu'il m'a montrée alors.

Je tiens à exprimer ici ma profonde gratitude à Alain Cabanes, pour son soutien, désintéressé et chaleureux, au cours de mes années d'enseignement à l'Institut d'Informatique d'Entreprise. Je garde un souvenir très gai de mon passage dans cette Grande Ecole.

Ce travail a été partiellement financé par le projet ESPRIT Basic Research Action Nr. 3075 (ALCOM).

Je remercie évidemment Olivier (Devillers pour les non-initiés) qui a accepté une reconversion (conversion ? !) aussi radicale qu'inattendue, pour me donner en déménageant une chance de mener à bien ces travaux ; pour notre collaboration depuis et pour avoir programmé vite et bien beaucoup de nos algorithmes.

Claire Kenyon m'a permis de trouver l'envie de reprendre l'étude de l'Arbre de Delaunay, en me signalant l'existence de travaux comparables, et par son intérêt pour mes recherches. Sans elle, il est clair que le contenu de cette thèse n'aurait rien eu de commun avec ce qu'il est.

Ces travaux ont donné lieu à plusieurs articles, et je remercie les co-auteurs non encore cités des discussions qui ont conduit à leur élaboration : Mariette Yvinec, longtemps compagne de bureau, qui a par ailleurs relu très soigneusement ce manuscrit et a contribué à en améliorer la clarté ; Stefan Meiser, grâce à qui nos quelques échanges INRIA - Max Planck Institut für Informatik ont été fructueux et agréables ; et enfin René Schott.

Merci à ceux qui ont programmé des applications du Graphe d'Influence : Serge Vaudenay pour le diagramme de Voronoï de segments et Denis Barthou pour l'enveloppe convexe.

Bien que notre collaboration porte sur d'autres sujets que ceux relatés dans cette thèse, je remercie ici André Cérézo pour les discussions fructueuses que nous avons régulièrement, et sa façon modeste de nous éclairer de ses connaissances indispensables.

Non, je ne remercierai pas Jean-Pierre Merlet for supplying me with his interactive drawing preparation system JPdraw —*enfin, si, quand même, pour avoir toujours supporté avec patience mes questions et mes multiples demandes de perfectionnement de ce* marvellous *logiciel !— mais simplement pour sa présence riche, rassurante dans la technopole aseptisée et superficielle.*

Grâce aux talents de dessinateur de Bernhard Geiger, les personnes feuilletant cette thèse seront sans doute nombreuses. La triangulation de Delaunay n'est pas la seule corde de sa contrebasse !

J'ai partagé des bureaux divers avec de nombreuses personnes, dont j'ai apprécié la compagnie. Je pense en particulier à Michel Pocchiola qui m'a apporté une bouffée d'humour sympathique, à son ami Francis Avnaim qui m'a offert un accueil enthousiaste, et puis à Katrin Dobrindt qui fait, curieusement, d'énormes progrès en conversation française à mon contact !

Merci à tous les membres du projet PRISME, pour la bonne humeur qui règne dans cette équipe ; au personnel de l'INRIA en général, en particulier aux non-chercheurs qui font souvent tout ce qui est en leur pouvoir pour nous donner de bonnes conditions de travail.

Last but not least, *je termine par une pensée pour mes amis lointains, heureusement néanmoins fidèles — les Anciennes de chez Jules, le Troisième, et tous les autres...*

Avant-propos

La plupart des chapitres de ce document sont rédigés en Anglais, afin d'en faciliter la diffusion à l'étranger.

Ces chapitres en Anglais correspondent au travail original effectué au cours de cette thèse.

Leur contenu a donné lieu par ailleurs à des présentations dans des conférences et des publications dans des revues, avec différents co-auteurs, selon les articles : Jean-Daniel Boissonnat, Olivier Devillers, Mariette Yvinec, Stefan Meiser et René Schott.

Notations et conventions

Expressions mathématiques usuelles

$\mathbb{N}$ ..... ensemble des entiers naturels
$\mathbb{E}^d$ ..... espace euclidien de dimension d
$\langle \cdot, \cdot \rangle$ ..... produit scalaire
$\delta(\cdot, \cdot)$ ..... distance euclidienne
$|\cdot|$ ..... cardinal d'un ensemble
$\mathrm{Prob}(\cdot)$ ..... probabilité d'un évènement
$E[\cdot]$ ..... espérance mathématique d'une variable aléatoire

Conventions d'écriture

$n, r, i, j, k, \ldots$, en italiques ..... éléments de $\mathbb{N}$
caractères $\mathtt{p}, \mathtt{q}, \mathtt{r}, \mathtt{1}, \mathtt{2}, \ldots$ ..... points de $\mathbb{E}^d$ ou objets
$T, F, \ldots$, en majuscules italiques ..... simplexes ou régions, bicycles ...
$\mathcal{S}$, $\mathcal{R}$, $\mathcal{F}$, $\mathcal{G}$, ... ensembles d'objets, de régions, de bicycles ...

Les notations suivantes seront utilisées tout au long de cette thèse. Elles sont définies dans la section 2.1 et le chapitre 4.

$\mathcal{O}$ ..... univers des objets
$\mathcal{F}$ ..... univers des régions
$\mathcal{S}$ ..... ensemble d'objets $\mathcal{S} \subset \mathcal{O}$
$\mathcal{S}^{(b)}$ ..... ensemble des parties de $\mathcal{S}$ d'au plus b éléments
$\mathcal{F}(\mathcal{S})$ ..... ensemble des régions définies par $\mathcal{S}$
$\mathcal{S}(F)$ ..... ensemble des objets de $\mathcal{S}$ en conflit avec la région F
$\mathcal{F}_j(\mathcal{S})$ ..... ensemble des régions de $\mathcal{F}(\mathcal{S})$ de largeur j
$\mathcal{F}_{\leq j}(\mathcal{S})$ ..... ensemble des régions de $\mathcal{F}(\mathcal{S})$ de largeur $\leq j$
$\mathcal{F}_j^{(i)}(\mathcal{S})$ ..... ensemble des régions de $\mathcal{F}_j(\mathcal{S})$ définies par i objets
$\mathcal{F}_{\leq j}^{(i)}(\mathcal{S})$ ..... ensemble des régions de $\mathcal{F}_{\leq j}(\mathcal{S})$ définies par i objets
$f_j(r, \mathcal{S})$ ..... espérance de $|\mathcal{F}_j(\mathcal{R})|$ pour $\mathcal{R} \subset \mathcal{S}$ de taille r
$f_j^{(i)}(r, \mathcal{S})$ ..... espérance de $|\mathcal{F}_j^{(i)}(\mathcal{R})|$ pour $\mathcal{R} \subset \mathcal{S}$ de taille r
$\phi_j(r, \mathcal{S})$ ..... maximum de $f_j(r', \mathcal{S})$ pour $r' \leq r$
$\mathcal{G}(\mathcal{S})$ ..... ensemble des bicycles définis par $\mathcal{S}$
$\mathcal{G}_j(\mathcal{S})$ ..... ensemble des bicycles de $\mathcal{G}(\mathcal{S})$ de largeur j
$\vdots$ ..... et autres notations dérivées de la même façon

Introduction

La Géométrie algorithmique a pour but de concevoir et d'analyser des algorithmes pour résoudre des problèmes géométriques. C'est un domaine récent de l'Informatique théorique, qui s'est très rapidement développé depuis son apparition dans la thèse de M.I. Shamos [Sha78] en 1978.

Ce développement a déjà atteint un haut degré de sophistication, qui affirme parfois encore plus l'aspect théorique de cette matière : dans le but d'obtenir une complexité optimale, il est parfois nécessaire de construire des algorithmes très compliqués, faisant appel à des structures de données difficiles à mettre en œuvre dans un programme.

Malgré l'apport incontestable de telles recherches, il peut être utile, de temps à autres, de s'intéresser à des aspects plus réalistes. C'est d'autant plus vrai, lorsque ceci présente également des côtés théoriques séduisants, et lorsque les méthodes restent rigoureuses. C'est le cas des algorithmes randomisés, introduits dans le domaine par K.L. Clarkson dès 1985 [Cla85]. L'étude de ce type d'algorithmes a progressivement intéressé les chercheurs, pour ensuite connaître un engouement impressionnant parmi la communauté, et devenir même un des sujets «chauds» (un peu trop parfois à mon goût, bien que la course soit stimulante !) de ces deux dernières années.

La randomisation permet d'éviter le recours à des structures compliquées, et s'avère très efficace, tant du point de vue de la complexité théorique, que des résultats pratiques. On peut cependant souligner le fait que, dans la littérature, bien que l'accent soit souvent mis sur cette efficacité pratique, celle-ci est seulement évoquée et rarement mise en application.

Cette efficacité n'est obtenue, en contrepartie, qu'en moyenne : alors que dans son sens plus classique en algorithmique, l'analyse en moyenne prend pour hypothèse une certaine distribution probabiliste des données, l'analyse randomisée fait la moyenne sur tous les déroulements possibles de l'algorithme. C'est effectivement le comportement de l'algorithme qui possède un caractère aléatoire, et pas la configuration des données, ni le résultat de l'algorithme qui, lui, reste totalement déterministe et fournit une solution unique et exacte.

L'analyse randomisée s'effectue en moyenne sur tous les déroulements possibles de l'algorithme, mais dans le pire des cas sur toutes les configurations possibles des données. C'est donc un modèle d'analyse réaliste : dans la pratique, les données auxquelles on s'intéresse sont souvent des points mesurés sur des objets, et leur distribution est donc loin d'être homogène, ou de vérifier quelque loi de probabilité que ce soit dans l'espace. On supposera simplement que les points sont indépendants les uns des autres. La randomisation est intéressante de ce point de vue lorsque cette complexité moyenne est meilleure que la complexité dans le pire des cas des algorithmes déterministes connus ou, à complexité égale,

lorsque les algorithmes sont plus simples (ce qui est très souvent vrai).

Il est aussi possible d'introduire dans l'analyse des quantités représentant la complexité du résultat, ce qui permet d'obtenir des algorithmes dont la complexité soit sensible, dans une certaine mesure, à la taille de la sortie.

Nous nous sommes intéressés plus particulièrement à la conception d'algorithmes dynamiques : en pratique, il est fréquent que l'acquisition des données d'un problème soit progressive. Il n'est évidemment pas question de recalculer le résultat à chaque nouvelle donnée, doù la nécessité d'utiliser des schémas (semi)-dynamiques. Nous avons étudié les algorithmes, à la fois du point de vue de la complexité théorique, de leur mise en œuvre pratique et de l'efficacité des programmes. Cette thèse fait la synthèse de tous nos travaux dans ce domaine.

Dans un premier chapitre, les définitions et les principales propriétés des structures classiques en Géométrie algorithmique sont rappelées, ainsi qu'un outil fondamental —la dualité— dont la présentation est nouvelle, et permet de démontrer des résultats bien connus d'une manière plus intuitive que la présentation habituelle. On pourra se rapporter à ce chapitre pour trouver toutes les bases nécessaires à la compréhention de nos algorithmes.

Les travaux antérieurs concernant les algorithmes incrémentaux randomisés sont présentés dans le chapitre 2. Ces algorithmes sont ceux qui ont marqué le début des recherches dans le domaine. Ils sont statiques, bien qu'incrémentaux, car ils utilisent tous le Graphe de Conflits, structure qui nécessite la connaissance de la totalité des données, dès l'initialisation. On peut signaler que K.L. Clarkson s'est également beaucoup intéressé aux algorithmes par «division-fusion», qui ne permettent pas d'extension dynamique, et qui ne sont donc pas développés. On présente différents schémas d'analyse, dûs à K.L. Clarkson, K. Mulmuley et R. Seidel.

L'Arbre de Delaunay est introduit dans le chapitre 3. C'est, (pré ! -)historiquement, notre première structure semi-dynamique [BT]. Cette structure permet de construire la triangulation de Delaunay d'un ensemble de points, en dimension quelconque, sans connaissance préliminaire de cet ensemble. L'arbre de Delaunay est comparé à une structure analogue, proposée en 1990 par L.J. Guibas, D.E. Knuth, et M. Sharir.

Le chapitre 4 montre comment l'idée qui est à la base de l'Arbre de Delaunay mène à l'élaboration d'une structure de données très générale, le Graphe d'Influence, qui permet de construire de façon semi-dynamique de nombreuses structures géométriques. Schématiquement, on peut dire que cette structure s'applique aux mêmes problèmes que ceux pour lesquels le Graphe de Conflits était précédemment utilisé, mais en autorisant des insertions.
Deux types d'analyse de complexité sont présentés, valables sous des hypothèses

comparables à celles imposées pour l'analyse de algorithmes basés sur le Graphe de Conflits. On étudie ensuite également la possibilité d'éviter ces restrictions.
Plusieurs applications sont développées. On discute également dans ce chapitre l'utilisation du Graphe d'Influence pour effectuer des requêtes, avec une complexité intéressante.
Des résultats expérimentaux pour le calcul de l'enveloppe convexe en dimensions 3 et 4, et de la triangulation de Delaunay en dimensions 2 et 3, démontrent l'efficacité pratique des algorithmes proposés.

Dans le chapitre 5, l'idée de l'Arbre de Delaunay est reprise, et enrichie de façon à pouvoir s'appliquer au calcul des diagrammes de Voronoï d'ordre supérieur, en dimension quelconque, qui ne rentre pas dans le cadre du chapitre 4. Des résultats expérimentaux sont proposés.

Le chapitre 6 pose enfin les premiers pas vers la dynamisation du Graphe d'Influence. Le schéma général de la suppression d'une donnée est clair. On verra, sur deux exemples —triangulation de Delaunay de points et arrangement de segments du plan— que cependant les détails sont assez techniques. Ceci est inévitable, sauf au prix d'omissions évasives. La complexité théorique est néanmoins excellente, et l'algorithme est très efficace en pratique.

Parallèlement à nos recherches sur le Graphe d'Influence, plusieurs autres structures ont été conçues, par de nombreux auteurs. On trouvera dans le chapitre 7 une présentation succinte de ces travaux, pour la plupart très récents.

Chapitre Premier

Quelques structures fondamentales

Dans ce chapitre sont présentées les définitions et les principales propriétés des structures fondamentales en géométrie algorithmique, pour lesquelles nous proposerons dans les chapitres suivants des algorithmes de construction. La complexité des meilleurs algorithmes déterministes connus est aussi indiquée. Ces résultats sont extraits pour la plupart de [Ede87, Meh84, PS85]. Pour plus de détails, en particulier en ce qui concerne les algorithmes, qui ne sont pas étudiés ici, ces ouvrages pourront être consultés.

$\mathbb{E}^d$ désigne l'espace euclidien de dimension d, muni du produit scalaire noté $\langle \cdot, \cdot \rangle$, et de la distance euclidienne δ. On emploiera souvent les termes usuels tels que « verticale », « dessous », « dessus » : la verticale est la direction du dernier axe de $\mathbb{E}^d$, l'hyperplan qui lui est orthogonal étant donc horizontal.

1.1 Enveloppe convexe

L'*enveloppe convexe* d'un ensemble fini $\mathcal{S}$ de n points de $\mathbb{E}^d$, notée $E(\mathcal{S})$, peut être définie de façon équivalente comme :

- le plus petit ensemble convexe contenant $\mathcal{S}$,
- l'intersection de tous les convexes contenant $\mathcal{S}$,
- l'intersection de tous les demi-espaces contenant $\mathcal{S}$.

Le calcul de l'enveloppe convexe est un problème central en géométrie algorithmique, et il a été énormément étudié, non seulement parce qu'il a des applications pratiques, mais aussi car c'est un préliminaire à la résolution d'un grand nombre d'autres questions.

Les résultats mathématiques classiques sur les polyèdres convexes (rappelons ici qu'une k-face est une face de dimension k) permettent d'énoncer la propriété suivante :

> *Propriété 1.1* Le nombre de k-faces et le nombre d'incidences entre k-faces et $k+1$-faces de $E(\mathcal{S})$ sont en $O\left(n^{\min(\lfloor \frac{d}{2} \rfloor, k+1)}\right)$, pour $0 \leq k \leq d-1$. Ces bornes sont asymptotiquement serrées.

En particulier, en dimension 2 et 3, ce nombre est linéaire.

Le comportement de la complexité de $E(\mathcal{S})$ a également été étudié sous des hypothèses sur la distribution des points de $\mathcal{S}$. Par exemple, si les points sont

choisis indépendamment selon une distribution normale, alors la taille moyenne de $E(\mathcal{S})$ est $O\left((\log n)^{\frac{d-1}{2}}\right)$.

Le problème du calcul de l'enveloppe convexe consiste à calculer le polyèdre qu'elle forme, c'est à dire la description complète de sa frontière.

Commençons par une première observation : dans le plan, il est facile de voir que ce problème est au moins aussi compliqué que le *tri* de n nombres : il suffit de prendre des points situés sur une parabole d'axe vertical, et il est clair que dans ce cas la solution est le polygone formé par ces n points, *dans l'ordre de leurs abscisses*. Tout algorithme calculant l'enveloppe convexe de n points du plan a donc une complexité temporelle $\Omega(n \log n)$.

Des algorithmes optimaux statiques existent en dimension 2 et 3.

En dimension supérieure, R. Seidel [Sei81] a donné un algorithme de complexité $O\left(n \log n + n^{\lfloor \frac{d+1}{2} \rfloor}\right)$ donc optimal en dimension paire ≥ 2 seulement. La place mémoire nécessaire est en $O\left(n^{\lfloor \frac{d}{2} \rfloor}\right)$. L'algorithme est incrémental, mais son analyse est amortie sur les n insertions.

La recherche d'un algorithme optimal en dimension quelconque est restée infructueuse pendant de nombreuses années, mais B. Chazelle a très récemment mis au point un tel algorithme [Cha91].

Un algorithme dont la complexité $O(n^2 + f \log n)$, où f est la taille du résultat (le nombre de faces de l'enveloppe convexe) est décrit dans [Sei86].

Dans le plan, des algorithmes déterministes en ligne, de complexité optimale $O(\log n)$ par insertion, sont connus (voir par exemple [AES85]).

1.2 Diagramme de Voronoï

1.2.1 Diagramme de Voronoï de points dans $\mathbb{E}^d$

Le *diagramme de Voronoï* d'un ensemble $\mathcal{S}$ de n points, appelés *sites*, de $\mathbb{E}^d$ est une structure géométrique permettant de résoudre des problèmes de proximité, tels que la recherche des plus proches voisins d'un site donné, ou de la paire formée des deux sites les plus proches.

Cette structure a été abondamment étudiée dans la littérature, et a été le point de départ de nos travaux. F. Aurenhammer lui a récemment consacré un article très complet [Aur91], dans lequel il en expose les applications, les propriétés et de nombreux algorithmes de calcul.

Pour chaque site $\mathtt{p}$ la *cellule* de Voronoï $V(\mathtt{p})$ de $\mathtt{p}$ est l'ensemble des points de $I\!E^d$ qui sont plus proches de $\mathtt{p}$ que de tous les autres sites de $\mathcal{S}$.

$$V(\mathtt{p}) = \{\mathtt{x} \in I\!E^d, \forall \mathtt{q} \in \mathcal{S} \setminus \{\mathtt{p}\}, \delta(\mathtt{x},\mathtt{p}) < \delta(\mathtt{x},\mathtt{q})\}$$

$V(\mathtt{p})$ s'écrit aussi

$$V(\mathtt{p}) = \bigcap_{\mathtt{q} \in \mathcal{S}\setminus\{\mathtt{p}\}} H(\mathtt{p},\mathtt{q})$$

où $H(\mathtt{p},\mathtt{q})$ est le demi-espace limité par l'hyperplan bissecteur de $\mathtt{p}$ et $\mathtt{q}$, contenant $\mathtt{p}$.

Le diagramme de Voronoï de $\mathcal{S}$ est la décomposition de $I\!E^d$ formée par les cellules de Voronoï des sites.

On supposera toujours que les points de $\mathcal{S}$ vérifient l'hypothèse de position générale suivante : $d+2$ points quelconques de $\mathcal{S}$ ne sont jamais cosphériques. Dans ce cas, les sommets du diagramme sont les centres de sphères passant par $d+1$ sites exactement.

> *Propriété 1.2* La complexité du diagramme de Voronoï de n sites dans $I\!E^d$ est $O\left(n^{\lceil \frac{d}{2} \rceil}\right)$.

En particulier, cette complexité est quadratique en dimension 3.

La *triangulation de Delaunay* est le *graphe dual* du diagramme de Voronoï : deux sites de $\mathcal{S}$ sont reliés dans la triangulation si et seulement si leurs cellules sont adjacentes. La triangulation de Delaunay est une partition de $I\!E^d$ en simplexes dont les sommets sont les sites. Elle vérifie la propriété suivante, qui la définit de façon unique :

> Les sphères circonscrites aux simplexes de la triangulation de Delaunay ne contiennent aucun site dans leur intérieur.

Cette propriété permet de concevoir un algorithme incrémental très simple. Cet algorithme est dû à Green et Sibson [GS78] pour la dimension 2, et a été généralisé ensuite à des dimensions quelconques [Bow81]. Comme beaucoup d'algorithmes incrémentaux, celui-ci est très peu performant dans le pire des cas : $O(n^2)$ dans le plan et $O\left(n^{\lceil \frac{d}{2} \rceil + 1}\right)$ en dimension d. Les auteurs montrent que la complexité peut être améliorée pour atteindre $O\left(n^{1+\frac{1}{d}}\right)$ en moyenne dans le cas de distributions homogènes. Cependant, la démonstration n'est pas très formelle, les hypothèses ne sont pas clairement précisées, et le cas de distributions dégénérées, par exemple des points répartis sur des surfaces, conduit à de mauvais résultats.

L'algorithme consiste à introduire les points un par un et à mettre à jour la structure après chaque insertion. Lorsqu'un nouveau point m est inséré, les simplexes dont la sphère circonscrite contient m doivent être détruits, ils ne font plus partie de la triangulation. L'union de ces simplexes est une région $R(\mathtt{m})$ simplement connexe. Si $F(\mathtt{m})$ désigne l'ensemble des facettes de la frontière de $R(\mathtt{m})$, les nouveaux simplexes sont simplement obtenus en reliant m aux facettes de $F(\mathtt{m})$ (voir figure 1.1). Cet algorithme est très simple, et permet l'acquisition

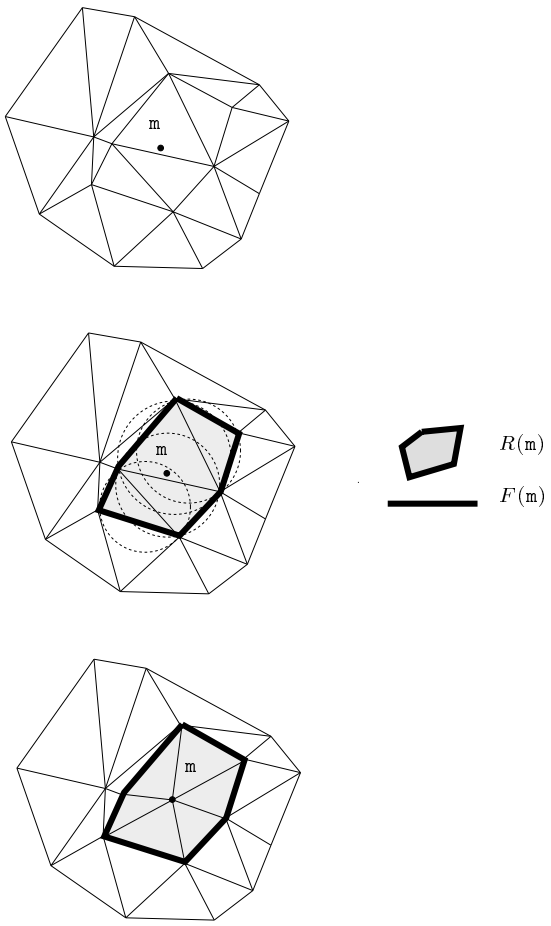


Figure 1.1 : Insertion d'un site dans la triangulation de Delaunay

progressives de données, ce qui est très intéressant en pratique. Le problème à résoudre pour en augmenter l'efficacité est de trouver une structure qui permette

de trouver l'ensemble $R(\mathtt{m})$ rapidement. C'est ce problème qui a été le point de départ de nos travaux (chapitre 3).

De nombreux algorithmes optimaux existent dans le cas du plan. L'algorithme de B. Chazelle pour les enveloppes convexes donne à présent par dualité (section 1.4) un algorithme optimal dans le pire des cas.

Dans le cas d'une distribution uniforme, la complexité du diagramme de Voronoï est $O(n)$ en moyenne [Dwy91]. En pratique (en dimension 3, pour des points mesurés sur la surface d'objets divers), le diagramme de Voronoï est souvent également de taille linéaire. Une des questions ouvertes fondamentales est de trouver des algorithmes dont la complexité dépende de la taille t du résultat. De tels algorithmes auraient une complexité au moins égale à $O(n \log n + t)$. Ceci a été résolu dans le cas très particulier de points situés dans deux plans dans [Boi88, BCDT91].

1.2.2 Diagramme de Voronoï d'ordre supérieur

Cas du plan

$\mathcal{S}$ désigne un ensemble de n sites du plan, en position générale. Si $\mathcal{T}$ est un sous-ensemble de points de $\mathcal{S}$, le polygone de Voronoï généralisé de $\mathcal{T}$ est défini par :

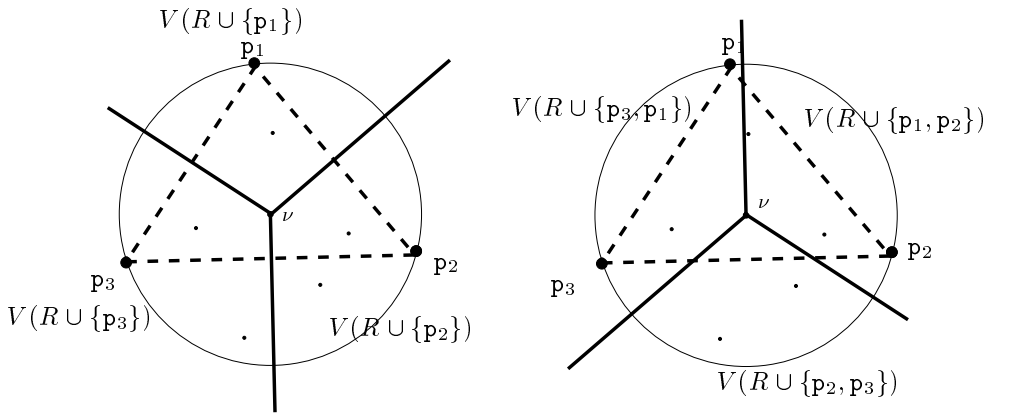
$$V(\mathcal{T}) = \{\mathtt{p}, \forall \mathtt{v} \in \mathcal{T}, \forall \mathtt{w} \in \mathcal{S} \setminus \mathcal{T}, \delta(\mathtt{p}, \mathtt{v}) < \delta(\mathtt{p}, \mathtt{w})\}$$

$V(\mathcal{T})$ est le lieu des points $\mathtt{p}$ plus proches de chaque site de $\mathcal{T}$ que de tous les autres points de $\mathcal{S} \setminus \mathcal{T}$. Le diagramme de Voronoï d'ordre k de $\mathcal{S}$ est ($|\cdot|$ désigne le cardinal d'un ensemble)

$$Vor_k(\mathcal{S}) = \{V(\mathcal{T}),\ \mathcal{T} \subseteq \mathcal{S},\ |\mathcal{T}| = k\}$$

Soient $\mathtt{p}_1, \mathtt{p}_2, \mathtt{p}_3$ trois sites de $\mathcal{S}$, ν le centre de leur cercle circonscrit, frontière du disque ouvert $B(\mathtt{p}_1, \mathtt{p}_2, \mathtt{p}_3)$. Comme les sites sont en position générale, ce cercle ne passe par aucun autre site. Soit R l'ensemble des sites contenus dans le disque $B(\mathtt{p}_1, \mathtt{p}_2, \mathtt{p}_3)$. Si $|R| = k$, alors ν est un sommet de $Vor_{k+1}(\mathcal{S})$ et $Vor_{k+2}(\mathcal{S})$:

- ν est le point commun aux régions $V(R \cup \{\mathtt{p}_1\})$, $V(R \cup \{\mathtt{p}_2\})$ et $V(R \cup \{\mathtt{p}_3\})$ dans $Vor_{k+1}(\mathcal{S})$. ν est appelé *sommet de type proche* dans ce diagramme (voir figure 1.2).
- ν est le point commun aux régions $V(R \cup \{\mathtt{p}_1, \mathtt{p}_2\})$, $V(R \cup \{\mathtt{p}_2, \mathtt{p}_3\})$ et $V(R \cup \{\mathtt{p}_3, \mathtt{p}_1\})$ dans $Vor_{k+2}(\mathcal{S})$. ν est un *sommet de type lointain* dans ce diagramme (voir figure 1.2).



Sommet de type proche dans $Vor_{k+1}(\mathcal{S})$ Sommet de type lointain dans $Vor_{k+2}(\mathcal{S})$

Figure 1.2 : Deux types de sommets

En résumé, tout triangle dont les sommets sont des sites de $\mathcal{S}$ et dont le disque circonscrit contient k sites en son intérieur correspond à (*est dual* d') un sommet de $Vor_{k+1}(\mathcal{S})$ et $Vor_{k+2}(\mathcal{S})$.

Réciproquement, un sommet de $Vor_k(\mathcal{S})$ est dual d'un triangle dont le disque circonscrit contient $k-1$ ou $k-2$ sites en son intérieur.

Dimension d

Les résultats précédents peuvent être généralisés aux dimensions supérieures.

Le diagramme de Voronoï d'ordre k est fait de cellules de dimensions $0, \ldots, d$. Les sommets du diagramme sont duaux de simplexes dont les sommets sont des sites de $\mathcal{S}$. Si $p_1, p_2, \ldots, p_{d+1}$ sont $d+1$ sites de $\mathcal{S}$, $B(\{p_1, \ldots, p_{d+1}\})$ est leur sphère circonscrite, ν son centre et $\mathcal{R}$ le sous-ensemble des points de $\mathcal{S}$ contenus dans $B(\{p_1, \ldots, p_{d+1}\})$. Si $|\mathcal{R}| = l$, ν est un sommet de tous les diagrammes de Voronoï $Vor_k(\mathcal{S})$ pour $l+1 \leq k \leq l+d$.

- ν est le point commun aux régions $V(\mathcal{R} \cup \{p_i\})$ pour $i \in \{1, \ldots, d+1\}$ dans $Vor_{l+1}(\mathcal{S})$: comme en dimension 2, on appelle ν un *sommet de type proche*.
- Dans $Vor_{l+h}(\mathcal{S})$, ν est le point commun aux régions $V(\mathcal{R} \cup \mathcal{P})$ où $\mathcal{P}$ peut être un sous-ensemble quelconque de taille h de $\{p_1, \ldots, p_{d+1}\}$; ν est incident à

$\binom{d+1}{h}$ régions. Si $1 < h < d$ on appelle ν un *sommet de type moyen.*

- ν est le point commun aux régions $V(\mathcal{R} \cup \{\mathtt{p}_1, \ldots, \mathtt{p}_{d+1}\} \setminus \{\mathtt{p}_i\})$ pour $i \in \{1, \ldots, d+1\}$ dans $Vor_{l+d}(\mathcal{S})$: comme en dimension 2, on appelle ν un *sommet de type lointain.*

En résumé, un sommet de type proche d'un diagramme de Voronoï d'ordre supérieur reste dans d diagrammes d'ordres successifs. Plus généralement, une h-face d'un diagramme reste dans $d-h$ diagrammes d'ordres successifs.

Résultats de complexité

Le diagramme de Voronoï d'ordre k a été introduit dans [SH75] pour traiter les problèmes de k plus proches voisins. Lee [Lee82] donne un résultat de complexité, dans le plan :

> *Propriété 1.3* Dans le plan, la taille du diagramme de Voronoï d'ordre k est $O(k(n-k))$. La taille des diagrammes de Voronoï d'ordres $\leq k$ est donc $O(nk^2)$.

La technique de l'échantillonnage aléatoire (section 2.3.1) a permis d'obtenir le résultat suivant, dans le pire des cas, en dimension d [CS89].

> *Propriété 1.4* En dimension $d > 2$, la taille des diagrammes de Voronoï d'ordres 1 à k est $O\left(k^{\lceil \frac{d+1}{2} \rceil} n^{\lfloor \frac{d+1}{2} \rfloor}\right)$.

Lee a aussi donné le premier algorithme de calcul, pour un ensemble de n points dans le plan. L'algorithme construit le diagramme d'ordre k à partir du diagramme d'ordre $k-1$ en temps $O(k(n-k)\log n)$. Les diagrammes d'ordre $\leq k$ peuvent donc être calculés en temps $O(k^2 n \log n)$. [AGSS89] permet de resserrer cette borne en $O(n \log n + k^2 n)$.

B. Chazelle et H. Edelsbrunner [CE85] ont développé deux versions d'un algorithme qui est meilleur pour les grandes valeurs de k. Le premier a une complexité temporelle $O(n^2 \log n + k(n-k)\log^2 n)$, avec une mémoire en $O(k(n-k))$ alors que le second prend un temps $O(n^2 + k(n-k)\log^2 n)$ et une place mémoire en $O(n^2)$. H. Edelsbrunner, J. O'Rourke et R. Seidel [EOS86] utilisent la dualité (section 1.4) pour construire tous les diagrammes d'ordres $\leq n-1$ en temps et mémoire $O(n^{d+1})$.

1.2.3 Diagramme de Voronoï de segments

$\mathcal{S}$ désigne à présent un ensemble d'*objets* : points ou segments de droite, dans le plan euclidien $I\!E^2$. Le diagramme de Voronoï de $\mathcal{S}$ est défini exactement de la même façon que dans le cas précédent, c'est la décomposition du plan en cellules $V(\mathtt{s})$ associées aux segments $\mathtt{s} \in \mathcal{S}$:

$$V(\mathtt{s}) = \{\mathtt{x} \in I\!E^2, \forall \mathtt{s}' \in \mathcal{S} \setminus \{\mathtt{s}\}, \delta(\mathtt{x}, \mathtt{s}) < \delta(\mathtt{x}, \mathtt{s}')\}$$

Un exemple en est donné dans la figure 1.3.

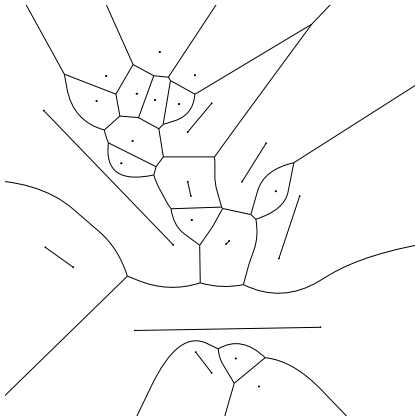


Figure 1.3 : Diagramme de Voronoï de segments

Précisons dès maintenant la structure du diagramme de Voronoï. Les détails seront utiles pour la section 4.3.3.1.

Le cercle vide maximal centré en un sommet de ce diagramme touche trois objets et le cercle vide maximal centré sur une arête touche deux objets.

Une arête Γ de ce diagramme est une partie de la médiatrice de deux objets $\mathtt{p}$ et $\mathtt{q}$, et ses deux extrémités sont équidistantes respectivement de $\mathtt{p}$, $\mathtt{q}$ et $\mathtt{r}$ et de $\mathtt{p}$, $\mathtt{q}$ et $\mathtt{s}$. Γ est définie par quatre segments et est notée $(\mathtt{pq}, \mathtt{r}, \mathtt{s})$. Elle est constituée de portions de droites et de paraboles. Pour décrire le cas des arêtes infinies du diagramme de Voronoï, on ajoute le symbole ∞. La région $(\mathtt{pq}, \mathtt{r}, \infty)$ correspond à une partie non bornée de la médiatrice de $\mathtt{p}$ et $\mathtt{q}$ (figure 1.5). De plus, pour assurer

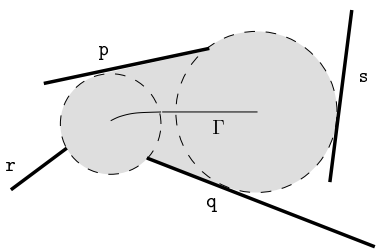


Figure 1.4 : Arête $(\mathtt{pq}, \mathtt{r}, \mathtt{s})$

la connectivité du diagramme, on ajoute des arêtes à l'infini, qui limiteront les cellules non bornées : l'arête $(\mathtt{p}\infty, \mathtt{r}, \mathtt{s})$ de la figure 1.5 est l'ensemble des centres (à l'infini) des cercles (de rayon infini, donc ce sont des droites) touchant $\mathtt{p}$, et elle est limitée par les cercles touchant respectivement $\mathtt{r}$ et $\mathtt{s}$. Dans les figures 1.4 et 1.5, les parties grisées sont les intérieurs des cercles vides centrés sur l'arête Γ.

Dans certains cas ambigus, l'étiquette $(\mathtt{pq}, \mathtt{r}, \mathtt{s})$ peut désigner deux arêtes différentes, qui seront distinguées de la façon suivante : $(\mathtt{pq}, \mathtt{r}, \mathtt{s})^+$ et $(\mathtt{pq}, \mathtt{r}, \mathtt{s})^-$ (figure 1.6).

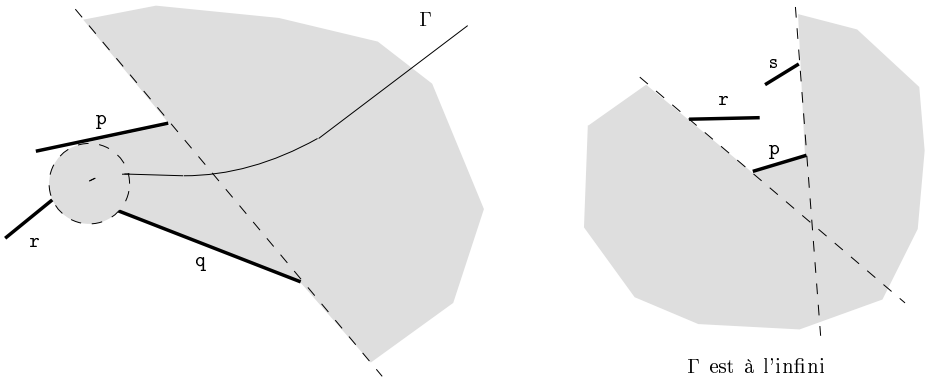
Le dual géométrique du diagramme de Voronoï de segments est la triangulation de Delaunay d'arêtes. C.K. Yap [Yap87] donne un algorithme optimal en $O(n \log n)$ pour calculer le diagramme de Voronoï de segments de droites et de portions de cercles.

1.3 Arrangements

Un arrangement est la partition de $\mathbb{E}^d$ induite par un ensemble fini d'hypersurfaces ou de portions d'hypersurfaces.

1.3.1 Arrangements d'hyperplans

$\mathcal{S}$ est ici un ensemble d'hyperplans de $\mathbb{E}^d$. La condition de position générale impose que $d+1$ hyperplans ont une intersection vide, ainsi un sommet d'un arrangement est commun à exactement d hyperplans. L'arrangement est constitué d'un ensemble de cellules, qui sont des polyèdres convexes.



arête définie par $(\mathtt{pq}, \mathtt{r}, \infty)$ arête définie par $(\mathtt{p}\infty, \mathtt{r}, \mathtt{s})$

Figure 1.5 : Arêtes infinies

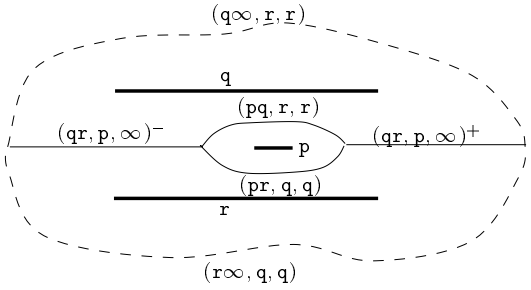


Figure 1.6 : Ambiguïté de l'étiquette $(\mathtt{pq}, \mathtt{r}, \mathtt{s})$

Un arrangement de n hyperplans de $\mathbb{E}^d$ peut être calculé, ainsi que son graphe d'incidence, en temps optimal $O(n^d)$. L'algorithme est incrémental, et s'appuie sur le Théorème de la Zone [ESS], qui permet d'affirmer que l'insertion d'un hyperplan s'effectue en $O\left(n^{d-1}\right)$.

Le *niveau* k dans un arrangement est défini de la façon suivante : c'est l'ensemble des points $\mathtt{p}$ de $\mathbb{E}^d$ tels que le nombre d'hyperplans de $\mathcal{S}$ strictement au dessus de $\mathtt{p}$ soit $\leq k-1$ et le nombre d'hyperplans strictement au dessous de $\mathtt{p}$ soit $\leq n-k$ (les hyperplans sont supposés non verticaux). La figure 1.7 montre un niveau 2 dans un arrangement de droites du plan.

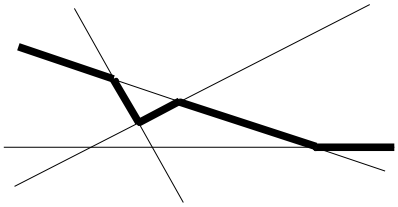


Figure 1.7 : 2-niveau dans un arrangement d'hyperplans

K.L. Clarkson (voir section 2.3.1) a montré que :

Propriété 1.5 La taille des niveaux d'ordres 1 à k dans un arrangement de n hyperplans de $\mathbb{E}^d$ est $O\left(n^{\lfloor \frac{d}{2} \rfloor} k^{\lceil \frac{d}{2} \rceil}\right)$.

1.3.2 Carte des trapèzes

Dans le cas où $\mathcal{S}$ est constitué de n segments de droite dans le plan $\mathbb{E}^2$, on peut définir la *carte des trapèzes* de cet arrangement. Cette carte est obtenue en traçant des verticales issues des extrémités des segments et des points d'intersection entre les segments, vers le haut et vers le bas, et limitées par les premiers segments rencontrés dans chacune des deux directions (figure 1.8).

Il est facile de voir que, si a désigne le nombre de points d'intersection entre les n segments, cette carte a $O(n+a)$ trapèzes, et que son calcul permet de décrire l'arrangement.

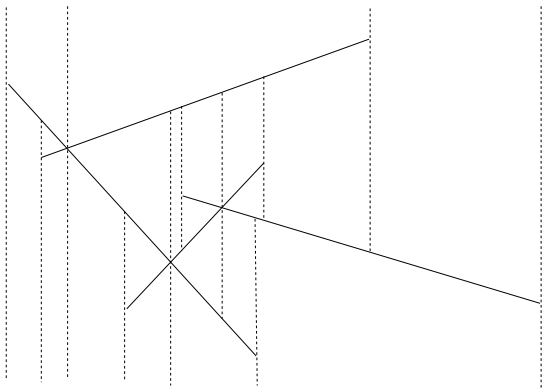


Figure 1.8 : Carte des trapèzes

Le meilleur algorithme déterministe est dû à B. Chazelle et H. Edelsbrunner [CE88], qui calculent l'arrangement de n segments de droite en temps $O(n \log n + a)$ avec un espace $O(n + a)$.

1.4 Transformations géométriques

On utilise souvent des transformations géométriques pour passer d'une structure à une autre, et déduire des résultats en utilisant des problèmes équivalents. Ces transformations sont souvent désignés par le nom global de *dualité*. C'est un outil fondamental, dont nous allons préciser différents aspects.

A l'origine, la dualité est la *polarité* mathématique, qui associe à un point de $I\!E^d$ un hyperplan de $I\!E^d$, et réciproquement : un point $\mathtt{p}$ de coordonnées

$$(p_1, p_2, \ldots, p_d)$$

est dual de l'hyperplan d'équation

$$(\pi_{\mathtt{p}}) \quad x_d = 2p_1x_1 + 2p_2x_2 + \ldots + 2p_{d-1}x_{d-1} - p_d,$$

et vice versa. Cette transformation préserve l'incidence et inverse la relation «dessus-dessous». Elle permet de voir simplement que

> l'enveloppe convexe d'un ensemble de points est duale de l'intersection d'un ensemble de demi-espaces

Les transformations géométriques peuvent faire intervenir des espaces de dimension différente : H. Edelsbrunner et F. Aurenhammer (et d'autres auteurs) ont beaucoup travaillé sur les résultats que des transformations géométriques permettent d'obtenir en Géométrie algorithmique (voir [Bro79] qui applique pour la première fois cet outil au diagramme de Voronoï, [ES86], et [Aur91] pour une vue d'ensemble). Leur approche est la suivante : si on prend le paraboloïde unité de $\mathbb{E}^{d+1}$, d'équation

$$(\Pi) \quad x_{d+1} = x_1^2 + x_2^2 + \ldots + x_d^2$$

alors l'intersection avec Π de l'hyperplan dual $\pi_{\mathtt{p}}$ d'un point $\mathtt{p}$ à l'extérieur de Π est constituée des points de Π en lesquels l'hyperplan tangent à Π passe par $\mathtt{p}$. En particulier, si $\mathtt{p}$ est sur Π, $\pi_{\mathtt{p}}$ est tangent à Π.

Cette remarque permet de voir que la projection de $\pi_{\mathtt{p}} \cap \Pi$ sur le plan horizontal d'équation $x_{d+1} = 0$ est une sphère dont le centre est la projection de $\mathtt{p}$.

Une conséquence importante de ceci est la suivante : soit $\mathcal{S}$ un ensemble de n points de $\mathbb{E}^d$, si on plonge $\mathbb{E}^d$ dans $\mathbb{E}^{d+1}$ en l'identifiant à l'hyperplan horizontal, alors

> le diagramme de Voronoï de $\mathcal{S}$ est la projection sur $\mathbb{E}^d$ du niveau 1 dans l'arrangement (enveloppe supérieure) dans $\mathbb{E}^{d+1}$ des hyperplans duaux des points obtenus en projetant sur Π les sites de $\mathcal{S}$.

En effet, l'intersection de deux hyperplans, tangents à Π en deux points respectifs projetés de $\mathtt{p}$ et $\mathtt{q}$ sur Π, se projette sur l'hyperplan horizontal $\mathbb{E}^d$ selon l'hyperplan médiateur de $\mathtt{p}$ et $\mathtt{q}$ (figure 1.9). De la même façon, le point d'intersection de $d+1$ hyperplans de cet arrangement, associés à $d+1$ sites de $\mathcal{S}$, se projette sur le centre ν de la sphère passant par ces $d+1$ sites. Le nombre de sites à l'intérieur de la sphère est le nombre d'hyperplans associés dans l'arrangement, qui passent au dessus de ce point d'intersection. Plus généralement, on obtient :

> le diagramme de Voronoï d'ordre k de $\mathcal{S}$ est dual du niveau k dans l'arrangement dans $\mathbb{E}^{d+1}$ des hyperplans duaux des points de $\mathcal{S}$.

K.Q. Brown démontrait dans [Bro79] la dualité entre diagramme de Voronoï et enveloppe convexe, en utilisant des inversions. Dans [ES86, Aur91], les démonstrations de ce genre de propriété sont purement analytiques. Nous proposons dans [DMT92] une interprétation beaucoup plus géométrique, qui permet d'éviter tous

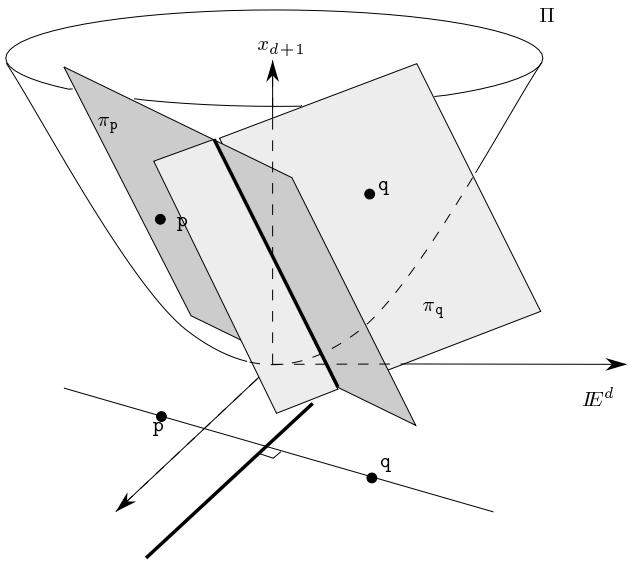


Figure 1.9 : Dualité entre diagramme de Voronoï et arrangement

les calculs (voir aussi [BCDT91]). Nous donnons ici une idée des raisonnements obtenus grâce à ce nouveau point de vue.

L'idée centrale consiste à définir l'*espace des sphères* σ de $I\!E^d$. Une sphère de $I\!E^d$ d'équation

$$(\mathtt{p})\ \ \langle \mathtt{x}, \mathtt{x} \rangle - 2\langle \mathtt{x}, \Phi \rangle + \chi = 0$$

est représentée par le point de coordonnées

$$\mathtt{p} = (\Phi, \chi)$$

dans σ. L'espace des sphères est isomorphe à $I\!E^{d+1}$.

On remarque immédiatement que l'ensemble des sphères réduites à leur centre (sphères de rayon nul) est le paraboloïde Π, qui apparaît donc de façon naturelle comme un ensemble privilégié de $I\!E^{d+1}$.

Tout se ramène ensuite simplement à revenir à la polarité mathématique. L'hyperplan dual $\pi_{\mathtt{p}}$ n'est autre que l'hyperplan polaire de $\mathtt{p}$ par rapport au paraboloïde Π, ou encore l'ensemble des conjugués de $\mathtt{p}$ par rapport à Π, c'est à dire l'ensemble des sphères orthogonales à $\mathtt{p}$ dans $I\!E^d$.

Ces observations permettent d'utiliser des résultats mathématiques très classiques et d'en déduire de façon très simple tous les résultats de dualité concernant des diagrammes de Voronoï généralisés. Les détails sont présentés dans [DMT92].

Chapitre II

Algorithmes incrémentaux randomisés statiques

Différentes techniques d'analyse d'algorithmes incrémentaux randomisés sont présentées dans ce chapitre, ainsi qu'une structure de donneés, le graphe de conflits dû à K.L. Clarkson. Cette structure impose le caractère *statique* des algorithmes incrémentaux l'utilisant : si une nouvelle donnée doit être prise en compte, il n'y a aucune possibilité de mise à jour du résultat, on doit calculer le nouveau résultat en reprenant l'exécution de l'algorithme au début.

Une formalisation du problème, empruntée à quelques détails près à [CS89], est d'abord décrite. Ce formalisme général pourra s'appliquer à de nombreux problèmes géométriques. Les notations introduites seront utilisées dans tous les chapitres suivants.

Ce chapitre ne tente pas de faire un inventaire exhaustif de tous les résultats que l'on peut obtenir grâce à des algorithmes randomisés statiques, mais de présenter l'essentiel des idées les plus importantes, idées (particulièrement celles de K.L. Clarkson, pionnier de la randomisation en géométrie algorithmique) qui ont suscité une quantité impressionnante d'articles.

2.1 Formalisation du problème

Tout problème géométrique est formulé en termes très généraux.

Les données du problème sont des *objets*, éléments d'un univers $\mathcal{O}$. Les objets considérés sont des sous-ensembles de l'espace de travail, par exemple les points, les segments, les droites... de l'espace euclidien $\mathbb{E}^d$ de dimension d.

Les *régions* appartiennent à un univers $\mathcal{F}$, formé également de sous-ensembles de l'espace de travail.

On s'intéresse aux régions définies par les objets. Si $\mathcal{S}$ est un ensemble d'objets, on note $\mathcal{S}^{(b)}$ l'ensemble des parties de $\mathcal{S}$ ayant au plus b éléments, pour un entier $b \in \mathbb{N}^*$; soit ∇ une relation entre $\mathcal{F}$ et $\mathcal{S}^{(b)}$. On dira que $F \in \mathcal{F}$ est *définie* par l'ensemble $\mathcal{S}$ s'il existe un élément X de $\mathcal{S}^{(b)}$ avec $F \nabla X$. On dira que X *détermine* F. $\mathcal{F}(\mathcal{S})$ désignera l'ensemble des régions définies par $\mathcal{S}$.

$$\mathcal{F}(\mathcal{S}) = \{F \in \mathcal{F} \ / \ (\exists X \in \mathcal{S}^{(b)})\ F \nabla X\}$$

La relation ∇ est supposée *fonctionnelle*, c'est-à-dire que pour toute région $F \in \mathcal{F}(\mathcal{S})$, le sous-ensemble X de $\mathcal{S}$ déterminant F est unique. Ceci formalise les hypothèses habituelles de *position générale* sur l'ensemble des objets de $\mathcal{S}$.

A chaque région est associée sa *zone d'influence* qui est un sous-ensemble de l'univers des objets $\mathcal{O}$. Les objets appartenant à la zone d'influence d'une région F seront dits *en conflit* avec F. La zone d'influence d'une région F possède par

définition la propriété de ne contenir aucun des éléments de X déterminant F. La définition d'une zone d'influence, et donc d'un conflit, devra évidemment être donnée de façon précise pour chaque application.

Pour $F \in \mathcal{F}$ et $\mathcal{S} \subset \mathcal{O}$ fini, on note $\mathcal{S}(F)$ l'ensemble des objets de $\mathcal{S}$ en conflit avec F, et on appelle *largeur* de F par rapport à $\mathcal{S}$ le nombre $|\mathcal{S}(F)|$.

On supposera dans un premier temps (on étudiera un problème plus général dans le chapitre 5) que le problème peut se mettre sous la forme suivante :

> *Calculer les régions définies par $\mathcal{S}$ et sans conflit avec les objets de $\mathcal{S}$ (régions de largeur nulle).*

Illustrons ces définitions sur l'**exemple de la triangulation de Delaunay**. Si on se place dans $I\!E^d$, l'univers $\mathcal{O}$ est l'ensemble des points de $I\!E^d$. L'univers $\mathcal{F}$ des régions est l'ensemble des simplexes et des demi-espaces de $I\!E^d$. Si $\mathcal{S}$ est un ensemble de points, une région peut être déterminée par d ou $d+1$ points. Posons $b = d+1$, $\mathcal{F}(\mathcal{S})$ est alors l'ensemble des simplexes ayant pour sommets $d+1$ points de $\mathcal{S}$ et des demi-espaces dont la frontière est un hyperplan passant par d points de $\mathcal{S}$.
La zone d'influence d'un simplexe est l'intérieur de la boule circonscrite à ce simplexe, et la zone d'influence d'un demi-espace est son intérieur. Dire que la relation ∇ est fonctionnelle revient à supposer que $d+2$ points de $\mathcal{S}$ seront toujours non cospheriques, et que $d+1$ points de $\mathcal{S}$ ne seront jamais cohyperplanaires. Calculer la triangulation de Delaunay de l'ensemble $\mathcal{S}$, c'est exactement calculer l'ensemble des régions sans conflit définies par $\mathcal{S}$.

Dans l'**exemple de l'intersection de demi-espaces**, les objets sont des demi-espaces de $I\!E^d$, et les régions sont des arêtes de l'intersection, déterminées par $d+1$ objets. Une arête et un demi-espace sont en conflit si l'arête n'est pas contenue dans le demi-espace fermé. L'intersection des demi-espaces est donnée par l'ensemble des arêtes sans conflit.

L'**exemple du calcul des intersections d'un ensemble de segments** sera lui aussi étudié dans ce qui suit. L'algorithme construit une carte de trapèzes (section 1.3.2). Un objet (un segment) est en conflit avec une région (un trapèze) si le segment rencontre le trapèze. Un trapèze est déterminé par au plus 4 segments. L'ensemble des trapèzes vides détermine l'arrangement de l'ensemble de segments.

Quelques autres notations seront nécessaires.
Si j est un entier, on note $\mathcal{F}_j(\mathcal{S})$ l'ensemble des régions de $\mathcal{F}(\mathcal{S})$ de largeur j par rapport à $\mathcal{S}$, et $\mathcal{F}_{\leq j}(\mathcal{S})$ l'ensemble des régions de largeur inférieure ou égale à j.

$\mathcal{F}_j^{(i)}(\mathcal{S})$ sera le sous-ensemble de $\mathcal{F}_j(\mathcal{S})$ des régions déterminées par des éléments X de $\mathcal{S}^{(b)}$ de cardinal $|X| = i \leq b$. $\mathcal{F}_{\leq j}^{(i)}(\mathcal{S})$ est défini de façon analogue.

Toutes les notions qui précèdent peuvent être définies de la même manière pour des sous-ensembles quelconques $\mathcal{R}$ de $\mathcal{S}$. La largeur d'une région définie par $\mathcal{R}$ pourrait être calculée par rapport à n'importe quel ensemble d'objets. Pour la concision du langage, on appellera «région de largeur j (sans plus de précision) définie par $\mathcal{R}$» une région définie par $\mathcal{R}$ et de largeur j par rapport à $\mathcal{R}$. Si l'algorithme est incrémental, et si $\mathcal{R}$ est l'ensemble des objets présents à un instant donné ($\mathcal{R}$ est l'ensemble courant d'objets), on appellera *largeur courante* d'une région définie par $\mathcal{S}$ sa largeur par rapport à $\mathcal{R}$.

Nous définissons dès maintenant les notations relatives à l'échantillonnage aléatoire. Un sous-ensemble $\mathcal{R}$ de $\mathcal{S}$ est un *échantillon aléatoire* de $\mathcal{S}$ si les éléments de $\mathcal{R}$ sont choisis au hasard parmi ceux de $\mathcal{S}$. Tous les échantillons aléatoires de taille r de $\mathcal{S}$ sont équiprobables, de probabilité $\frac{1}{\binom{n}{r}}$ où n est le cardinal de $\mathcal{S}$.
On notera $f_j(r, \mathcal{S})$ l'espérance mathématique $E_{(r,\mathcal{S})}[|\mathcal{F}_j(\mathcal{R})|]$ du nombre $|\mathcal{F}_j(\mathcal{R})|$ de régions de largeur j définies par un échantillon aléatoire $\mathcal{R}$ de taille r de $\mathcal{S}$. De même $f_j^{(i)}(r, \mathcal{S}) = E_{(r,\mathcal{S})}[|\mathcal{F}_j^{(i)}(\mathcal{R})|]$. Enfin, $\phi_j(r, \mathcal{S})$ est le maximum des espérances $f_j(r', \mathcal{S})$ pour $1 \leq r' \leq r$.

2.2 Une structure de données : le graphe de conflits

L'idée générale conduisant au graphe de conflits de K.L. Clarkson n'est pas spécifique aux problèmes géométriques.

Le principe du graphe de conflits est illustré dans [CS89] par l'exemple du tri par insertion. Le tri de n éléments par insertion peut être vu comme une alternative au schéma «division-fusion» du tri rapide. A chaque étape, un nouvel élément est mis à sa place. Si on maintient une liste triée des éléments déjà insérés, ce tri peut être coûteux, car de nombreux éléments de la liste seront examinés à chaque nouvelle insertion.

Deux idées sont utilisées pour améliorer cet algorithme :

- Supposons que l'on connaisse, pour chaque valeur n'ayant pas encore été insérée, sa place dans la liste courante, ainsi que, pour chaque place, la liste des valeurs non insérées correspondantes. Ces informations constituent le graphe de conflits. Lors de l'insertion de l'élément c, si sa place est entre a et b, on peut l'insérer directement. Il reste ensuite à mettre à jour dans le

graphe de conflits l'ensemble des valeurs non insérées comprises entre a et b, en les comparant à c. Le temps d'insertion de c est donc proportionnel au nombre d'éléments entre a et b.

- Si de plus les nombres sont insérés de façon randomisée, c'est à dire dans un ordre aléatoire, à l'étape r, les valeurs déjà insérées sont bien réparties dans l'ensemble de nombres à trier. Le nombre de valeurs comprises entre a et b est proche de $\frac{n}{r}$.

Ces deux hypothèses permettent de conclure à une complexité randomisée de

$$O\left(\sum_{r=1}^{n} \frac{n}{r}\right) = O(n \log n)$$

Ceci résume le schéma des algorithmes incrémentaux randomisés utilisant le graphe de conflits. On peut remarquer que, dans ces algorithmes, le coût d'insertion est dominé par le coût de mise à jour de l'information concernant les conflits.

Formalisons ce schéma dans le cadre défini dans la section 2.1. Le problème géométrique se ramène à construire l'ensemble $\mathcal{F}_0(\mathcal{S})$ des régions de largeur nulle définies par un ensemble $\mathcal{S}$ de n objets.

La démarche incrémentale consiste à ajouter un à un les objets de $\mathcal{S}$ au sous-ensemble courant $\mathcal{R}$ de $\mathcal{S}$ tout en maintenant l'ensemble $\mathcal{F}_0(\mathcal{R})$ des régions de largeur nulle définies par $\mathcal{R}$.

A l'étape r, lorsque l'objet o est ajouté au sous-ensemble courant $\mathcal{R}$, l'ensemble des régions de largeur courante nulle est mis à jour en supprimant les régions de $\mathcal{F}_0(\mathcal{R})$ en conflit avec o et en y ajoutant de nouvelles régions de largeur courante nulle : les régions définies par les sous-ensembles de $\mathcal{R} \cup \{o\}$ contenant o, et sans conflit avec les objets de $\mathcal{R} \cup \{o\}$.

Pour rendre cette mise à jour plus rapide, l'algorithme maintient, en plus de l'ensemble de régions $\mathcal{F}_0(\mathcal{R})$, le *graphe de conflits*. Le graphe de conflits est un graphe bipartite, défini sur le produit cartésien $\mathcal{F}_0(\mathcal{R}) \times \mathcal{S} \setminus \mathcal{R}$, avec une arête pour chaque couple (F, o) formé d'une région F de $\mathcal{F}_0(\mathcal{R})$ et d'un objet o de $\mathcal{S} \setminus \mathcal{R}$ en conflit avec F.

Le graphe de conflits permet donc de trouver rapidement (en temps linéaire par rapport au nombre de régions trouvées) les régions de $\mathcal{F}_0(\mathcal{R})$ en conflit avec le nouvel objet o traité à l'étape courante. En contrepartie, chaque étape incrémentale doit maintenant inclure une phase de mise à jour du graphe de conflits. Les arêtes de conflit incidentes aux régions de $\mathcal{F}_0(\mathcal{R})$ en conflit avec o sont supprimées, et de nouvelles arêtes sont créées pour représenter les conflits impliquant

les nouvelles régions de largeur courante nulle (régions de $\mathcal{F}_0(\mathcal{R} \cup \{\circ\}) \setminus \mathcal{F}_0(\mathcal{R})$) et les objets restant à insérer (objets de $\mathcal{S} \setminus (\mathcal{R} \cup \{\circ\})$). La complexité de chaque étape incrémentale est donc au minimum proportionnelle au nombre d'arêtes du graphe de conflits mises à jour (c'est-à-dire créées ou détruites) à cette étape.

La définition du graphe de conflits impose la connaissance de l'ensemble $\mathcal{S}$ dès l'étape d'initialisation. C'est pourquoi les algorithmes l'utilisant sont intrinsèquement statiques, bien qu'incrémentaux. Le but des chapitres suivants est d'introduire une structure tout aussi générale, mais permettant de s'affranchir de cette contrainte.

2.3 Techniques d'analyse

2.3.1 Echantillonnage aléatoire

Une première idée

P. Erdös et J. Spencer [ES74] ont, les premiers, consacré un ouvrage entier aux techniques probabilistes. Leur but était de faire connaître à la communauté scientifique ces méthodes puissantes, dont les applications à divers problèmes combinatoires étaient déjà nombreuses.

Ce livre contient de nouvelles démonstrations de théorèmes déjà connus à l'époque, dans différents domaines de la combinatoire, principalement en ce qui concerne la théorie des graphes. La technique consiste à remplacer des arguments de «comptage» par des arguments utilisant des variables aléatoires, dans des démonstrations d'existence non constructives.

Considérons un des premiers exemples de l'ouvrage.
Un *tournoi* sur un ensemble V de sommets est un graphe dirigé T (ensemble de couples de V^2) dans lequel, pour toute paire $\{x, y\}$ d'éléments de V, avec $x \neq y$, on ait : soit $(x, y) \in T$, soit $(y, x) \in T$, mais pas les deux. Soit $\mathcal{T}_n$ l'ensemble des tournois sur $V_n = \{1, \ldots, n\}$ pour un entier n donné. Un *chemin hamiltonien* P dans un tournoi $T \in \mathcal{T}_n$ est un élément de l'ensemble Σ_n des permutations de V_n, tel que, pour tout $i \in \{1, \ldots, n-1\}$, $(P(i), P(i+1)) \in T$. Le résultat suivant est dû à T. Szele[1].

Théorème [Szele] *Il existe $T \in \mathcal{T}_n$ contenant au moins $n!\, 2^{-n+1}$ chemins hamiltoniens.*

[1] Kombinatorikai vizsgálatok az irányított teljes graffál kapcsolatban, *Mat. Fiz. Lapok* 50, 1943, 223-256

Démonstration. de T. Szele
Soit $U = \{(T, P)/T \in \mathcal{T}_n, P \in \Sigma_n \text{ chemin hamiltonien dans } T\}$

Le cardinal de $\mathcal{T}_n$ est $2^{\binom{n}{2}}$.
Si $P \in \Sigma_n$ est fixé, le nombre de tournois T tels que $(T, P) \in U$ est $2^{\binom{n}{2}-(n-1)}$ puisque P détermine $n-1$ arêtes de T. Comme il y a $n!$ permutations de V_n, U a $|U| = n!\, 2^{\binom{n}{2}-(n-1)}$ éléments.
D'autre part, $|U| = \sum_{T \in \mathcal{T}_n} |\{P \in \Sigma_n/(T, P) \in U\}|$.

Cette somme a $2^{\binom{n}{2}}$ termes, donc il existe un $T \in \mathcal{T}_n$ pour lequel on ait

$$|\{P \in \Sigma_n/(T, P) \in U\}| \geq n!\, 2^{-(n-1)}$$

□

Démonstration. de P. Erdös et J. Spencer
Soit $\mathbf{T}$ une variable aléatoire à valeurs dans $\mathcal{T}_n$, telle que

$$\forall T \in \mathcal{T}_n, \text{Prob}(\mathbf{T} = T) = 2^{-\binom{n}{2}}$$

ou encore telle que

$$\forall i, j \in V_n, \text{Prob}((i, j) \in \mathbf{T}) = \frac{1}{2}$$

avec des probabilités indépendantes pour chaque paire $\{i, j\}$.
Pour $P \in \Sigma_n$ fixé, définissons alors une fonction f_P sur $\mathcal{T}_n$ par

$$f_P(T) = \begin{cases} 1 \text{ si } P \text{ est un chemin hamiltonien dans } T \\ 0 \text{ sinon} \end{cases}$$

$h(T) = \sum_{P \in \Sigma_n} f_P(T)$ est le nombre de chemins hamiltoniens dans T.
L'additivité de l'espérance mathématique permet d'écrire :

$$\begin{aligned} E[h(T)] &= E\left[\sum_{P \in \Sigma_n} f_P(T)\right] \\ &= \sum_{P \in \Sigma_n} E\left[f_P(T)\right] \\ &= \sum_{P \in \Sigma_n} 2^{-n+1} \\ &= n!\, 2^{-n+1} \end{aligned}$$

Nécessairement, il existe donc un $T \in \mathcal{T}_n$ pour lequel on ait

$$h(T) \geq n!\, 2^{-n+1}$$

□

Application à la géométrie algorithmique

K.L. Clarkson a utilisé cette idée pour construire une technique de démonstration destinée à prouver des résultats combinatoires, en géométrie algorithmique. Dans ses nombreuses publications, il l'applique au début pour des algorithmes spécifiques —problème du bureau de poste, enveloppes convexes, intersections de segments— [Cla85, Cla87, CS88], puis il définit, avec P.W. Shor, un formalisme très général [CS89] repris pour l'essentiel dans la section 2.1.

Les notations utilisées sont celles de la section 2.1, on pourra se reporter à la page viii qui en fournit un résumé.

K.L. Clarkson et P.W. Shor bornent le nombre $|\mathcal{F}^{(i)}_{\leq k}(\mathcal{S})|$ de régions de largeur au plus k définies par un ensemble $\mathcal{S}$ d'objets de taille n. Nous allons donner la preuve du théorème car elle est représentative de toutes les démonstrations utilisant l'échantillonnage aléatoire. La recherche de cette borne utilise deux lemmes fondamentaux.

Lemme 2.1 *Soit $\mathcal{S}$ un ensemble de n objets et F une région déterminée par i objets de $\mathcal{S}$ et de largeur j par rapport à $\mathcal{S}$ ($F \in \mathcal{F}^{(i)}_j(\mathcal{S})$). La probabilité $p^{(i)}_{j,k}(r)$ pour que F soit une région de largeur k définie par un échantillon aléatoire de taille r de $\mathcal{S}$ est :*

$$p^{(i)}_{j,k}(r) = \frac{\binom{j}{k}\binom{n-i-j}{r-i-k}}{\binom{n}{r}}$$

Démonstration. Soit $\mathcal{R}$ un échantillon aléatoire de taille r de $\mathcal{S}$. Il y a $\binom{n}{r}$ possibilités pour choisir $\mathcal{R}$. La région F de $\mathcal{F}^{(i)}_j(\mathcal{S})$ appartient à $\mathcal{F}^{(i)}_k(\mathcal{R})$ si les i objets qui la déterminent appartiennent à $\mathcal{R}$, et si de plus elle est en conflit avec k objets de $\mathcal{R}$, ce qui est réalisé lorsque $\mathcal{R}$ contient également k objets parmi les j objets de $\mathcal{S}$ en conflit avec F. $\mathcal{R}$ doit ensuite contenir $r-i-k$ autres objets choisis parmi les $n-i-j$ objets de $\mathcal{S}$ restants. □

Lemme 2.2 *Soit $\mathcal{S}$ un ensemble de n objets et $\mathcal{R}$ un échantillon aléatoire de taille r de $\mathcal{S}$. L'espérance mathématique $f^{(i)}_k(r,\mathcal{S})$ du nombre de régions déterminées par i objets de $\mathcal{R}$ et de largeur k par rapport à $\mathcal{R}$ est donnée par :*

$$f^{(i)}_k(r,\mathcal{S}) = \sum_{j=0}^{n-i} |\mathcal{F}^{(i)}_j(\mathcal{S})| \, \frac{\binom{j}{k}\binom{n-i-j}{r-i-k}}{\binom{n}{r}}$$

Démonstration. $f_k^{(i)}(r,\mathcal{S})$ est donnée en fonction de la probabilité pour qu'une région $F \in \mathcal{F}(\mathcal{S})$, déterminée par i objets, appartienne à $\mathcal{F}_k^{(i)}(\mathcal{R})$:

$$\begin{aligned} f_k^{(i)}(r,\mathcal{S}) &= \sum_{j=0}^{n-i} \sum_{F \in \mathcal{F}_j^{(i)}(\mathcal{S})} \text{Prob}(F \in \mathcal{F}_k^{(i)}(\mathcal{R})) \\ &= \sum_{j=0}^{n-i} |\mathcal{F}_j^{(i)}(\mathcal{S})|\, p_{j,k}^{(i)}(r) \end{aligned}$$

Le lemme 2.1 permet de conclure. □

Ces deux lemmes permettent de démontrer le théorème :

Théorème 2.3 [Clarkson-Shor] *Soit $\mathcal{S}$ un ensemble de n objets. Le nombre $|\mathcal{F}_{\leq k}(\mathcal{S})|$ (pour $k \geq 2$) de régions de largeur au plus k définies par $\mathcal{S}$ est majoré, en fonction de l'espérance mathématique $f_0\left(\lfloor \frac{n}{k} \rfloor, \mathcal{S}\right)$ du nombre de régions de largeur nulle définies par un échantillon aléatoire de taille $\frac{n}{k}$ de $\mathcal{S}$, de la façon suivante :*

$$|\mathcal{F}_{\leq k}(\mathcal{S})| \leq 4(b+1)^b k^b f_0\left(\lfloor \tfrac{n}{k} \rfloor, \mathcal{S}\right)$$

(b désigne toujours, comme dans la définition de $\mathcal{F}(\mathcal{S})$, le nombre maximal d'objets déterminant une région)

Démonstration. L'inégalité $|\mathcal{F}_{\leq k}^{(i)}(\mathcal{S})| \leq 4(b+1)^i k^i f_0^{(i)}\left(\lfloor \frac{n}{k} \rfloor, \mathcal{S}\right)$ est démontrée pour chaque valeur de i comprise entre 1 et b.

D'après le lemme 2.2, l'espérance mathématique du nombre $f_0^{(i)}(r,\mathcal{S})$ du nombre de régions de largeur nulle définies par un échantillon aléatoire de taille r de $\mathcal{S}$ vaut :

$$\begin{aligned} f_0^{(i)}(r,\mathcal{S}) &= \sum_{j=0}^{n-i} |\mathcal{F}_j^{(i)}(\mathcal{S})| \frac{\binom{n-i-j}{r-i}}{\binom{n}{r}} \\ &\geq \sum_{j=0}^{k} |\mathcal{F}_j^{(i)}(\mathcal{S})| \frac{\binom{n-i-j}{r-i}}{\binom{n}{r}} \end{aligned}$$

Des calculs sur les factorielles montrent que, pour $j \leq k$,

$$\frac{\left(\begin{array}{c} n-i-j \\ r-i \end{array}\right)}{\left(\begin{array}{c} n \\ r \end{array}\right)} \geq \frac{r(r-1)\ldots(r-i-1)}{n(n-1)\ldots(n-i-1)} \left(\frac{n-r-k+1}{n-i-k+1}\right)^k .$$

Pour $r = \lfloor \frac{n}{k} \rfloor$ et $k \geq 1$, on a de plus

$$\frac{n-r-k+1}{n-i-k+1} \geq 1 - \frac{1}{k},$$

ce qui permet de minorer :

$$\left(\frac{n-\lfloor \frac{n}{k} \rfloor - k+1}{n-i-k+1}\right)^k \geq \frac{1}{4} \text{ pour } k \geq 2.$$

En reportant ces minorations dans l'inégalité obtenue pour $f_0^{(i)}(r, \mathcal{S})$, on trouve :

$$|\mathcal{F}_{\leq k}^{(i)}(\mathcal{S})| \leq 4 \frac{n(n-1)\ldots(n-i-1)}{\lfloor \frac{n}{k} \rfloor \left(\lfloor \frac{n}{k} \rfloor - 1\right) \ldots \left(\lfloor \frac{n}{k} \rfloor - i - 1\right)} f_0^{(i)}\left(\lfloor \tfrac{n}{k} \rfloor, \mathcal{S}\right)$$

et en utilisant l'inégalité $\frac{n}{k} - 1 < \lfloor \frac{n}{k} \rfloor$, on obtient enfin

$$|\mathcal{F}_{\leq k}^{(i)}(\mathcal{S})| \leq 4k^i \frac{1}{\left(1 - \frac{bk}{n}\right)^i} f_0^{(i)}\left(\lfloor \tfrac{n}{k} \rfloor, \mathcal{S}\right)$$

d'où l'on déduit le résultat en majorant k par $\frac{n}{b+1}$. □

Remarque 2.4 Ce théorème ne s'applique que pour les valeurs de k comprises entre 2 et $\frac{n}{b+1}$. Il permet néanmoins de déduire les majorations suivantes :

$$|\mathcal{F}_0(\mathcal{S})| \leq |\mathcal{F}_{\leq 1}(\mathcal{S})| \leq |\mathcal{F}_{\leq 2}(\mathcal{S})| \leq 4(b+1)^b 2^b f_0\left(\lfloor \tfrac{n}{2} \rfloor, \mathcal{S}\right)\left(1 + O\left(\frac{1}{n}\right)\right)$$

D'autre part, pour les valeurs de k proches de n, on a la borne triviale :

$$|\mathcal{F}_{\leq k}(\mathcal{S})| \leq |\mathcal{F}(\mathcal{S})| = O(n^b)$$

Le théorème fournit des majorations intéressantes lorsqu'une borne supérieure pour $f_0\left(\lfloor \frac{n}{k} \rfloor, \mathcal{S}\right)$ est connue. Prenons l'exemple d'un ensemble $\mathcal{S}$ d'hyperplans de $\mathbb{E}^d$. Le théorème permet de montrer, comme annoncé dans la section 1.3.1 :

Corollaire 2.5 [Clarkson-Shor] *Pour tout ensemble d'hyperplans en position générale dans $\mathbb{E}^d$, et pour tout entier $k \geq 2$, la taille des niveaux d'ordre $\leq k$ dans l'arrangement est bornée par* $O\left(n^{\lfloor \frac{d}{2} \rfloor} k^{\lceil \frac{d}{2} \rceil}\right)$.

et un autre résultat déjà cité dans la section 1.2.2 :

Corollaire 2.6 [Clarkson-Shor] *Pour tout ensemble de points en position générale dans $\mathbb{E}^d$, et pour tout entier $k \geq 2$, la taille des diagrammes de Voronoï d'ordres $\leq k$ est bornée par* $O\left(n^{\lfloor \frac{d+1}{2} \rfloor} k^{\lceil \frac{d+1}{2} \rceil}\right)$.

Considérons à présent un algorithme incrémental. Cet algorithme est randomisé si les objets sont introduits de façon aléatoire, c'est-à-dire que, à chaque instant, chacun des objets a la même probabilité d'être tiré. Dans ce cas, l'ensemble $\mathcal{R}$ des objets présents à une étape donnée est un échantillon aléatoire de $\mathcal{S}$.

Comme on l'a vu dans la section 2.2, si on utilise un graphe de conflits, la complexité de chaque étape incrémentale est au minimum proportionnelle au nombre d'arêtes du graphe de conflits créées ou détruites à cette étape. On impose la condition suivante :

Condition d'actualisation : à chaque étape incrémentale, la mise à jour de l'ensemble des régions de largeur courante nulle et du graphe de conflits sont effectuées en un temps de calcul proportionnel au nombre d'arêtes du graphe de conflits créées ou détruites à cette étape.

Le théorème fondamental sur la complexité des algorithmes randomisés est présenté ci-dessous. Sa démonstration est effectuée dans [CS89], elle utilise la technique de l'échantillonnage aléatoire, de façon similaire à la démonstration du théorème 2.3. On montre précisément que le temps moyen d'insertion d'un objet à l'étape $r+1$ est

$$O(\phi_0(r, \mathcal{S}))\frac{n-r}{r^2}.$$

($\phi_0(r, \mathcal{S})$est le maximum de $f_0(r', \mathcal{S})$ pour $r' \leq r$)

Théorème 2.7 [Clarkson-Shor] *Un algorithme incrémental randomisé qui utilise un graphe de conflits et satisfait la condition d'actualisation, traite un ensemble d'objet $\mathcal{S}$ de taille n en temps moyen*

$$O\left(n \sum_{r=1}^{n} \frac{\phi_0(r, \mathcal{S})}{r^2}\right).$$

Ainsi, par exemple,

- si $\phi_0(r, \mathcal{S}) = O(r)$, l'algorithme a une complexité moyenne $O(n \log n)$
- si $\phi_0(r, \mathcal{S}) = O(r^\alpha)$ avec $\alpha > 1$, la complexité moyenne est $O(n^\alpha)$.

Citons deux applications de ce théorème :

Théorème 2.8 [Clarkson-Shor] *Soit $\mathcal{S}$ un ensemble non dégénéré de n segments du plan, dont a paires se coupent. Les a intersections peuvent être calculées par un algorithme incrémental randomisé en un temps moyen $O(a+n \log n)$, avec un espace mémoire moyen $O(n + a)$.*

Théorème 2.9 [Clarkson-Shor] *Un algorithme incrémental randomisé calcule l'intersection de n demi-espaces de $\mathbb{E}^d$ en temps moyen $O(n \log n)$ pour $d \leq 3$ et $O\left(n^{\lfloor \frac{d}{2} \rfloor}\right)$ pour $d \geq 4$.*

2.3.2 Jeux probabilistes et séries Θ

K. Mulmuley s'est intéressé, parallèlement à K.L. Clarkson, aux algorithmes incrémentaux randomisés. Il utilise une technique d'analyse totalement différente, ne reposant pas sur l'échantillonnage aléatoire. Les résultats obtenus sont malgré cela similaires, et la structure de données est également un graphe de conflits. Il a étudié tout d'abord les arrangements de courbes dans le plan [Mul88, Mul89a] puis les arrangements d'hyperplans dans l'espace et les diagrammes de Voronoï d'ordres supérieurs [Mul89b, Mul91a].

Dans [Mul91a], on construit les k premiers niveaux de l'arrangement d'un ensemble $\mathcal{S}$ de n hyperplans de $\mathbb{E}^d$. $v(l)$ désigne la taille du niveau l. On définit

$$\Theta_{l_0}(s, j, \mathcal{S}) = \sum_{l=1}^{l_0} v(l) + \sum_{l=l_0+1}^{j} \left(\frac{l_0}{l}\right)^s v(l)$$

La complexité de l'algorithme est exprimée en fonction de $\Theta_k(d, n, \mathcal{S})$.

Le modèle probabiliste utilisé pour le tirage aléatoire des hyperplans est une suite d'épreuves de Bernoulli indépendantes : pour chaque hyperplan de l'ensemble $\mathcal{S}$, il lance indépendamment une pièce ayant une probabilité $\frac{1}{r}$ de succès. Les hyperplans sélectionnés sont ceux pour lesquels le tirage a été gagnant. $\Phi_r^{l_0}$ désigne le nombre de sommets des niveaux inférieurs ou égaux à l_0 dans l'arrangement de l'échantillon ainsi obtenu. On calcule alors l'espérance $E[\Phi_r^{l_0}]$ de deux manières différentes.

D'une part, on effectue un raisonnement simple (similaire à celui de la démonstration du lemme 2.1), qui lui permet de dire qu'un sommet v de niveau $l-1$ apparaît, dans le cas où $l \leq l_0$, si les d hyperplans le déterminant sont tirés, et si, de plus, dans le cas où $l > l_0$, au plus $l_0 - 1$ hyperplans parmi les $l-1$ en dessous de v sont tirés. Ceci conduit à une expression de $E[\Phi_r^{l_0}]$. Le modèle probabiliste choisi permet à K. Mulmuley d'utiliser, d'autre part, des outils mathématiques classiques tels que la fonction Γ ou la borne de Chernoff pour majorer $E[\Phi_r^{l_0}]$ (il utilise aussi un théorème de la zone généralisé qu'il démontre dans le même article).

La comparaison des deux résultats permet de donner des bornes sur les séries Θ, d'où découle le résultat :

Théorème 2.10 [Mulmuley] *Les k premiers niveaux d'un arrangement d'hyperplans de $\mathbb{E}^d$ peuvent être calculés en un temps moyen $O\left(k^{\lceil \frac{d}{2} \rceil} n^{\lfloor \frac{d}{2} \rfloor}\right)$ pour $d \geq 4$ ($O\left(kn \log \frac{n}{k}\right)$ si $d = 2$ et $O\left(k^2 n \log \frac{n}{k}\right)$ si $d = 3$) en utilisant un algorithme incrémental randomisé.*

Les diagrammes de Voronoï d'ordre 1 à k sont obtenus par la même méthode en un temps moyen $O\left(k^{\lceil \frac{d+1}{2} \rceil} n^{\lfloor \frac{d+1}{2} \rfloor}\right)$ pour $d \geq 3$ ($O(nk^2 + n \log n)$ si $d = 2$).

Les travaux récents de K. Mulmuley sur les algorithmes dynamiques randomisés sont présentés dans la section 7.3.2.

2.3.3 Analyse "en arrière"

Cette méthode permet des démonstrations très belles et élégantes, car étonnamment simples. [Sei90] est un des premiers articles dans lesquels cette idée a été publiée. R. Seidel attribue à P. Chew la paternité de cette technique restée longtemps dans l'ombre. P. Chew l'aurait le premier utilisée pour analyser un algorithme de calcul de la triangulation de Delaunay d'un polygone convexe, algorithme jamais publié avant la rédaction très récente de R. Seidel [Sei91a] ou O. Devillers [Dev] (qui utilise d'ailleurs aussi une analyse du même type, qui sera présentée dans le chapitre 4).

La méthode consiste à analyser un algorithme comme s'il se déroulait en remontant le temps, du résultat vers les données. Elle ne nécessite aucun formalisme ni technique de calcul spécifiques.

Etudions l'exemple classique du calcul de l'enveloppe convexe d'un ensemble $\mathcal{S}$ de points de taille n. On maintient un graphe de conflits entre les arêtes de l'enveloppe convexe courante et les points non encore insérés.

Décrivons l'algorithme de calcul de l'enveloppe convexe de $\mathcal{R} \subset \mathcal{S}$ de manière récursive :
Si $|\mathcal{R}| = 3$, on obtient le triangle formé par les trois points de $\mathcal{R}$.
Sinon, choisir un point q au hasard dans $\mathcal{R}$. Soit $\mathcal{R}' = \mathcal{R} \setminus \{\mathtt{q}\}$. Calculer récursivement l'enveloppe convexe de $\mathcal{R}'$, puis insérer q de la façon suivante :
Si q est contenu dans l'enveloppe convexe des points de $\mathcal{R}'$, il n'y a rien à faire. Dans le cas contraire, le graphe de conflits donne une arête visible depuis q, et un parcours de l'enveloppe convexe permet de trouver toutes les arêtes visibles depuis q, puisqu'elles forment une chaîne. Il suffit ensuite de remplacer cette chaîne par les deux arêtes issues de q.

Le coût de la mise à jour de l'enveloppe convexe est proportionnel au nombre d'arêtes détruites lors de l'insertion de q, qui peut être grand. Cependant, comme une arête détruite ne réapparaît jamais, ce coût peut être chargé à la création de ces arêtes. Or, à chaque insertion, deux arêtes au plus sont créées, et donc le coût total de mise à jour de l'enveloppe convexe est linéaire.

Reste à analyser le coût des mises à jour du graphe de conflits. Il suffit de maintenir, pour chaque point p non encore inséré, l'arête particulière $E_{\mathtt{p}}$ de l'enveloppe convexe traversée par le segment [cp], où c est un point fixé intérieur à l'enveloppe convexe (par exemple le barycentre des trois premiers points). Réciproquement, pour chaque arête E, on maintient l'ensemble des points p pour lesquels $E = E_{\mathtt{p}}$.

Le coût de la mise à jour du graphe de conflits, lors de l'insertion de q, est proportionnel au nombre de points p pour lesquels l'arête $E_{\mathtt{p}}$ change. Pour un point p donné, $E_{\mathtt{p}}$ change si, après l'insertion de q (l'analyse "en arrière" intervient ici), q est l'une des extrémités de $E_{\mathtt{p}}$. Comme q est choisi au hasard dans $\mathcal{R}$, la probabilité que l'on choisisse pour q l'une des deux extrémités de $E_{\mathtt{p}}$ est $\frac{2}{r}$, où $r = |\mathcal{R}|$. Le nombre moyen de changements pour un point p est donc au plus $\frac{2}{r}$ (si p est déjà dans $\mathcal{R}$, il n'y a aucun changement). Si on somme pour toutes les valeurs de $r \leq n$, on obtient un nombre total de changements pour un point p valant au plus $O(\log n)$.

Comme l'initialisation du graphe de conflits est effectuée en temps linéaire, le coût total de l'algorithme est $O(n \log n)$.

Cet exemple illustre la simplicité de la méthode : aucun calcul ni raisonnement compliqué n'a été nécessaire.

Conclusion

Nous avons présenté les travaux antérieurs à ceux décrits dans les chapitres suivants, concernant les algorithmes incrémentaux randomisés. Un point commun à tous ces algorithmes est que les résultats sont valides quelle que soit la distribution des points, contrairement aux analyses plus classiques, comme celle de [Dwy91] par exemple. La seule condition imposée est que la séquence d'insertions soit randomisée.

Ces algorithmes sont tous statiques car ils utilisent le graphe de conflits. Cette structure a été utilisée avec succès par de nombreux auteurs, pour calculer toutes sortes de structures géométriques. Citons par exemple [MMO91] qui calcule des diagrammes de Voronoï "abstraits", [Mul88, Mul89a] qui construit des arrangements dans le plan, [Mul89b, Mul91a] qui étudie les niveaux dans un arrangement d'hyperplans, et bien sûr [CS89] qui développe de nombreuses applications. L'un des buts des chapitres suivants sera l'élaboration d'une structure dynamique.

Remarquons de plus que, dans les théorèmes démontrés jusqu'ici, le coût d'insertion d'un objet est dominé par le coût de mise à jour du graphe de conflits, qui n'est pas directement relié à la taille du changement effectif dans la structure que l'on cherche à calculer. Schématiquement, le coût d'insertion des premiers objets et très élevé, alors que le changement du résultat est de taille constante.

De très nombreux autres résultats ont été obtenus parallèlement, ou postérieurement aux nôtres. Ils sont résumés pour certains au cours des chapitres consacrés à nos travaux, et les autres sont regroupés dans le chapitre 7.

Chapitre III

L'arbre de Delaunay

The Delaunay Tree is a hierarchical data structure which is defined from the Delaunay triangulation and, roughly speaking, represents a triangulation as a hierarchy of balls. It allows an "on-line" construction of the Delaunay triangulation of a finite set of n points in any dimensions : the points are not supposed to be known in advance, as in Chapter 2. On the contrary, the Delaunay Tree allows the data to be given while the incremental algorithm is running, because it makes possible to look for the conflicts of the point to be inserted, without storing them explicitly.

To this aim, the idea of maintaining the *history* of the construction —the successive versions of the triangulation, linked together in the tree— is introduced for the first time in [BT86]. The history allows a hierarchical location in the current triangulation, which is very efficient. The update of the structure is easy. This idea has been then often used again in randomized dynamic algorithms (see Sections 3.3 and Chapter 7). However, K. Mulmuley created a totally different dynamic data structure (see Section 7.3.2).

We only present the algorithm and the data structure in this chapter. The analysis will be given in Section 4.3.3.1, as a consequence of the general analysis of the Influence graph (see also [BT]).

A similar structure, introduced afterwards by L.J. Guibas, D.E. Knuth and M. Sharir [GKS90], is presented at the end of this chapter.

3.1 Structure

The nodes of the tree are associated with the successive simplices of the triangulation. The word "simplex" will denote the geometric object as well as the corresponding node.

For the initialization step we choose $d+1$ sites of $\mathcal{S}$. They generate one finite simplex and $d+1$ *infinite* ones (see Figure 3.1) : a halfspace will also be called a simplex, and will be considered as having d finite vertices and one vertex at infinity. The ball circumscribing an infinite simplex is exactly the halfspace itself, that can be considered as a ball with infinite radius. These $d+2$ simplices will be the sons of the root of the tree.

Then the other points are inserted one after another. The triangulation is updated as indicated in Section 1.2.1. As already defined in Section 2.1, we say that a site `p` is in conflict with a simplex if `p` lies in the interior of it circumscribing ball.

After the insertion of site `p`, the simplices in conflict with `p` disappear from the triangulation, but remain in the Delaunay Tree. They are called *dead* and

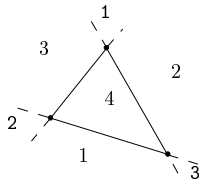


Figure 3.1 : Initialization step

p is their *killer*. Some of those simplices have a facet on the boundary $F(\mathtt{p})$ of the region $R(\mathtt{p})$ formed by the simplices in conflict with p. Let T be one of the simplices in conflict with p that has a facet F belonging to $F(\mathtt{p})$. We construct a new simplex S, *created* by p, having vertex p and facet F. Let N be the simplex sharing facet F with T before the insertion of p. Because the triangulation is a Delaunay one, we have the following property (see Figure 3.2) :

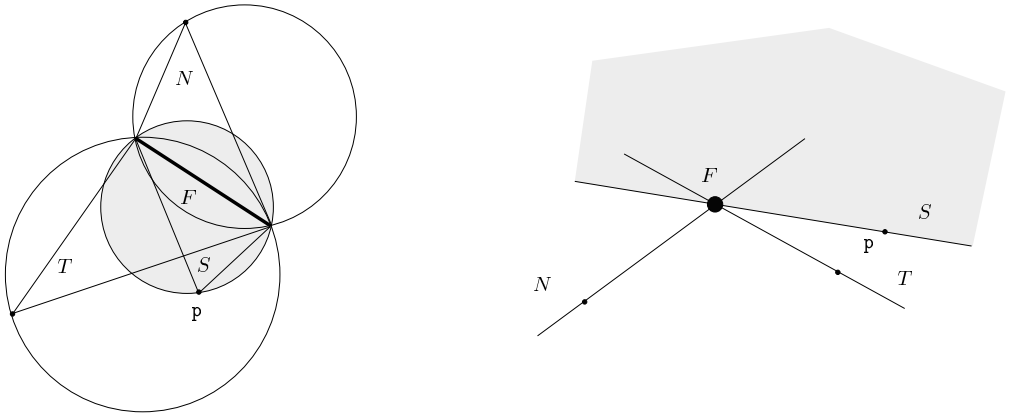
> *Property 3.1* The circumscribing ball of S is included in the union of the two balls circumscribing T and N

This property is fundamental for the correctness of the algorithm, as will be seen in the sequel.

The newly created simplex S will be called : *son of* T and *stepson of* N through facet F. These edges will never been modified during the construction.

When it is killed, a simplex receives from 0 (if it has no facet on $F(\mathtt{p})$) to $d+1$ sons. As long as it is not dead, it can on the other hand receive an arbitrary number of stepsons : when a new point $\mathtt{p}'$ will be inserted, in conflict with S but not with N, S will be killed in turn, and its son S' having vertex $\mathtt{p}'$ and facet F will be another stepson of N. Thus a node has at most one son and one list of stepsons through each facet, that is : 0 to $d+1$ sons and 0 to $d+1$ *lists* of stepsons. The following property can nevertheless be used to bound the size of the structure :

> *Property 3.2* The total size of the stepson lists in the Delaunay Tree is less than the number of nodes, since each newly created node (the $d+2$ sons of the root excepted) has exactly one stepfather. This is true in any dimension.



Case of a finite facet F Case of an infinite facet F

Figure 3.2 : Insertion of p

This hierarchical structure is called a *Delaunay Tree* for short, but it is more exactly a rooted direct acyclic graph. This graph contains a tree : the tree whose links are the links between fathers and sons.

A simplex of the current triangulation is not dead, and so corresponds to a node having no son, but possibly stepsons.

Note that we also maintain adjacency relationships between the simplices of the current triangulation. This will be developed in Section 5.1.3.5.

Let us summarize the structure of a node of the Delaunay Tree :
• the triangle : creator vertex, two other vertices, circumscribed circle
• a mark **dead**
• pointers to the at most three sons and the list of stepsons
• the three current neighbors if the triangle is not dead, the three neighbors at the death otherwise
• a pointer **killer** to the site that killed the triangle

3.2 Constructing the Delaunay triangulation

Let p be a site to be introduced in the triangulation. Two steps are performed: first, we locate p in order to find the set $R(\mathtt{p})$ of all the simplices in conflict with p and then we create the new simplices.

3.2.1 Location

If p is in conflict with a simplex T, Property 3.1 implies that p is in conflict with the father or the stepfather of T (or both of them). So we will be able to find all the simplices which are killed by p by recursively exploring the Delaunay Tree. Procedure location given in Figure 3.3 describes this traversal.

Procedure location(p,T) :

if T has not been visited yet
and p is in conflict with T **then**
  for each stepson S of T location(p,S) ;
  for each son S of T location(p,S) ;
  if T is not dead **then**
    mark T killed by p ;
    add T to the list $R(\mathtt{p})$ of the killed simplices.

Figure 3.3 : Location of a site in the Delaunay Tree

Remark 3.3 It would also be possible to stop the recursive traversal as soon as a first simplex of the current triangulation in conflict with p is found. The other simplices in conflict with p would then be obtained by following neighborhood relations (see the end of Section 3.1).

Remark 3.4 All edges between a simplex and its successive stepsons may be traversed by Procedure location. Figure 3.4 shows an example. Points 1 to 7 are numbered in insertion order. The subgraph of the

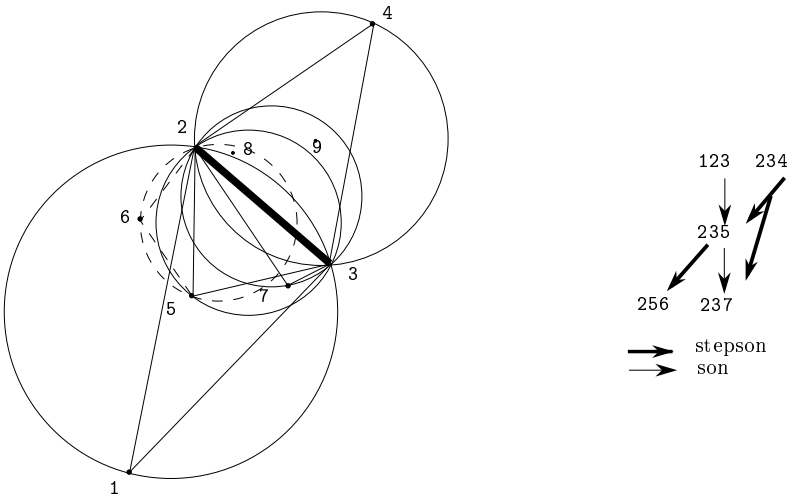


Figure 3.4 : All stepsons may be useful

Delaunay Tree corresponding to triangles 123, 234, 235, 256 et 237 is shown on the right side of the figure.

If site 8 is now introduced, it is in conflict with 235 and 256. but not with 123. The edges traversed to locate 8 are those from 234 to 235 and from 235 to 256. No path passing through 237 leads from 234 to 235.

If we want to locate 9, which is in conflict with 237, but not with 235, we must use the link from 234 to 237.

Thus, it is necessary to store both stepsons of 234.

3.2.2 Creating the new simplices

We go through the list $R(\mathtt{p})$ of the killed simplices. A facet F of a simplex T is on $F(\mathtt{p})$ if the simplex N neighbor of T through F is not killed. In this case, we create the simplex S with vertex p and facet F, and the two edges between S, its father T and its stepfather N. Moreover N and S are neighbors through F. Figure 3.5 describes Procedure creation realizing these updatings.

The adjacency relations between the new simplices are obtained by exploiting

---

Procedure creation(p):

for each simplex T killed by p
  for each neighbor N of T through a facet F
    if p is not in conflict with N **then**
      create the simplex S having vertex p and facet F ;
      replace the adjacency relation between N and T
        by the adjacency relation between N and S ;
      create the relations: S son of T and S stepson of N through facet F ;
create the adjacency relationships between the new simplices.

Figure 3.5 : Creating the new simplices

---

the relations between old ones. For the sake of clarity, the dimensions of the faces are precised. If f is a $(d-2)$-face of $F(\mathtt{p})$, common to two $(d-1)$-faces F and F' of $F(\mathtt{p})$, there exists a sequence of simplices killed by p and all sharing the same $(d-2)$-face f. The first simplex of this sequence has facet F, and the last one has facet F'. Theses simplices are neighbors, taken two by two. The traversal of this sequence of simplices, using their adjacency relations, allows to link the two new simplices S and S' having vertex p and respective facets F and F', that are neighbors through the $(d-1)$-face formed by p and f. Repeating this process around each $(d-2)$-face of $F(\mathtt{p})$, we obtain all adjacency relations. Figure 3.6 illustrates this.

Each killed simplex has at most $d+1$ $(d-1)$-faces on $F(\mathtt{p})$, each of them has d $(d-2)$-faces, and each $(d-2)$-face is shared by two $(d-1)$-faces. A killed simplex is thus traversed at most $\frac{d(d+1)}{2}$ times during this search for neighbors among the new simplices.

The analysis of the algorithm is presented in section 4.3.3.1. Let us only give the result stated in Proposition 4.21 :

The Delaunay triangulation of n points in d-space can be computed on-line with $O\left(n^{\lfloor\frac{d+1}{2}\rfloor}\right)$ expected space in dimension $d \geq 2$. The expected update time for an insertion is $O(\log n)$ if $d = 2$ and $O\left(n^{\lfloor\frac{d+1}{2}\rfloor-1}\right)$ if

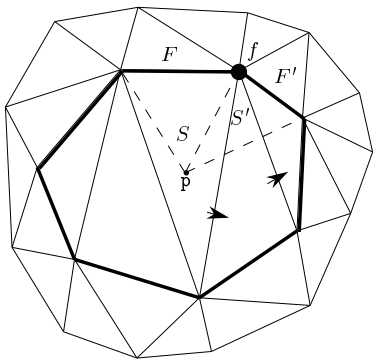


Figure 3.6 : Adjacency relations between new simplices

$d \geq 3$. *These results are optimal.*

3.3 Another structure

L.J. Guibas, D.E. Knuth and M. Sharir proposed another structure [GKS90], also based on the history of the construction.

The construction is incremental, but the updating of the triangulation is different. It relies on the technique of switching the diagonal inside the quadrilateral composed by two adjacent triangles : if two adjacent triangles `pqr` and `rsp` are given, if `s` is in conflict with `pqr`, then `pqr` and `rsp` are replaced by `pqs` and `qrs`.

When a new site `p` is inserted, it must be located, which means here that we look for the triangle `abc` of the Delaunay triangulation containing `p`, and not those whose circumscribing circle contains `p`. Then the triangulation is updated as follows : first, `abc` is replaced by `pab`, `pbc`, `pca` ; then there is a propagation in the triangulation by examining the triangles by pairs of adjacent triangles, and possibly exchanging the diagonals.

The structure contains all triangles, Delaunay triangles and non Delaunay triangles, created at any time of the construction. Edges are created between a killed triangle and the new triangles partially covering it. In this way, the lists of stepsons are avoided, and the number of edges outcoming from a triangle is constant. but some triangles that have never been Delaunay triangles remain

in the structure. The randomized analysis of this algorithm, based on random sampling, yields, despite these differences, to the same complexity as the Delaunay Tree (however, the analysis that is presented in [GKS90] is amortized).

A major drawback of this technique was, until a very recent time, that it was based on an updating process (the exchange of diagonals), that had never been generalized to dimensions greater than 2. But V.T. Rajan [Raj91] has just filled this lack, which might allow the extension of this structure to higher dimensions.

Conclusion

We have defined a semi-dynamic data structure for constructing the Delaunay triangulation, based on the knowledge of the history of the incremental construction.

It is possible to generalize this structure, keeping the same idea of remembering the previous steps of the construction. This general structure, the Influence graph, is presented in Chapter 4.

Chapitre IV

Une structure générale : le graphe d'influence

The Influence Graph [BDS*] generalizes the Delaunay tree. It is presented here under the formalism and notations introduced in Section 2.1 (see also Page viii). A randomized analysis of the algorithms using this structure is proposed in Section 4.1.1. We also describe a different analysis, due to O. Devillers. Then, in Section 4.2, we study the efficiency of the Influence Graph for locating a site.

Section 4.3 develops several applications to semi-dynamic constructions (allowing insertions) : arrangements, Delaunay triangulations of point sites in $\mathbb{E}^d$, convex hull in $\mathbb{E}^d$, Voronoi diagrams of segments in the plane. In the case of convex hulls and Delaunay triangulations, some experimental results are given.

The last section of the chapter is devoted to some remarks about our results.

4.1 The general framework

The different algorithms presented here are incremental and introduce objects one by one. Let $\mathcal{S}$ be the set of objects which have already been introduced. At a given stage, the incremental algorithm inserts a new object in $\mathcal{S}$ and updates the set $\mathcal{F}_0(\mathcal{S})$ of regions of current zero width defined by $\mathcal{S}$ (recall that a region is determined by at most b objects). This is performed through the maintenance of a dynamic structure called the influence graph (I-DAG for short) described below.

The I-DAG is a rooted directed acyclic graph whose nodes are associated with regions that, at some stage of the algorithm, have appeared as regions of zero width defined by the set of objects that have been introduced at that stage. Although the I-DAG is not strictly speaking a tree, we speak of leaves, fathers, sons etc. in the obvious way. The nodes of the I-DAG associated with regions of $\mathcal{F}_0(\mathcal{S})$ are marked. When a new object o is added to $\mathcal{S}$, one new node is created for each region in $\mathcal{F}_0(\mathcal{S} \cup \{o\})$ that is not a region of $\mathcal{F}_0(\mathcal{S})$.

As in the Delaunay Tree, the already existing nodes are never deleted from the I-DAG but possibly unmarked (they are *dead*). The new nodes are linked by edges to nodes already present in the I-DAG. These edges are constructed in such a way that a new object o can be efficiently located in the structure ; locating o means here to traverse all the nodes whose associated regions are in conflict with o. When a new node corresponding to a region F of $\mathcal{F}_0(\mathcal{S} \cup \{o\})$ is added to the I-DAG, we put edges between already existing nodes and the new nodes so that the influence range of F is included in the union of the ranges of its parents.

The I-DAG structure is characterized by the two following fundamental properties :

Property 1 At each stage of the incremental process, the regions of current zero width ($\mathcal{F}_0(\mathcal{S})$) are leaves of the I-DAG.

Property 2 The influence range of the region associated with a node is included in the union of the ranges of its parents.

The construction of the I-DAG can be sketched as follows:

- We initialize $\mathcal{S}$ with the b first objects. A node of the I-DAG is created for each region of $\mathcal{F}_0(\mathcal{S})$ and made son of the root of the I-DAG. The influence range associated with the root is the whole objects universe $\mathcal{O}$.
- At each subsequent step, a new object $\mathtt{o}$ is added to $\mathcal{S}$ and the I-DAG is updated. The two following substeps are performed:

  location substep. This substep finds all the nodes of the I-DAG whose regions have zero width and are in conflict with $\mathtt{o}$. This is done by traversing every path from the root of the I-DAG down to the first node which is not in conflict with object $\mathtt{o}$.

  creation substep. From the information collected during the first substep, the creation substep creates a new node for each region in $\mathcal{F}_0(\mathcal{S} \cup \{\mathtt{o}\}) \setminus \mathcal{F}_0(\mathcal{S})$ and links the new nodes to already existing nodes in the structure so that Properties 1 and 2 still hold. The details of this substep depend on each particular application.

4.1.1 Randomized analysis of the I-DAG

This subsection aims at providing a randomized analysis of the space and time required to build the I-DAG structure. Randomization concerns here only the order in which the inserted objects are introduced in the structure. Thus, if the current set of objects is a set $\mathcal{S}$ of cardinality n, our results are expected values that correspond to averaging over the $n!$ possible permutations of the inserted objects, each one being equally likely to occur.

The main results of this chapter are stated in Theorems 4.3 and 4.13 below, they give quite general upper bounds on the size and the update time of the I-DAG as functions of the expected size of the output for a sample of the input.

For the sake of clarity, we will make some hypotheses that simplify somewhat the analysis. These conditions are fulfilled by a large class of geometric problems and allow one to express the results in a simple way. However these hypotheses are not really necessary and will be removed in Section 4.1.4.

The update conditions

We first assume that three update conditions are satisfied. These conditions are almost equivalent to Clarkson's. As already mentioned these hypotheses are mainly for clarity and will be removed later.

(1) The number of sons of a node of the I-DAG is bounded.

(2) Given a region F and an object o, the test to decide whether or not o is in conflict with F can be performed in constant time.

(3) If the new object o added to the current set $\mathcal{S}$ is found to be in conflict with k regions of $\mathcal{F}_0(\mathcal{S})$ then the creation substep requires $O(k)$ time.

Expected storage

Lemma 4.1 *If $\mathcal{S}$ has cardinality n, the expected size of the I-DAG of $\mathcal{S}$ is :*

$$O\left(\sum_{j=1}^{n} \frac{f_0(\lfloor \frac{n}{j} \rfloor, \mathcal{S})}{j}\right)$$

Proof. The expected number of nodes $\eta(\mathcal{S})$, in the I-DAG of $\mathcal{S}$ can be obtained by summing, for all the regions F of $\mathcal{F}(\mathcal{S})$, the probability that F occurs as a node in the I-DAG, which is $\frac{i!j!}{(i+j)!}$ (the i objects determining F must be inserted before the j objects in conflict with F).

So,

$$\begin{aligned}
\eta(\mathcal{S}) &= \sum_{i=1}^{b}\sum_{j=0}^{n-i} |\mathcal{F}_j^{(i)}(\mathcal{S})| \frac{i!j!}{(i+j)!} \\
&= \sum_{i=1}^{b}\left(|\mathcal{F}_0^{(i)}(\mathcal{S})| + \sum_{j=1}^{n-i}\left(|\mathcal{F}_{\leq j}^{(i)}(\mathcal{S})| - |\mathcal{F}_{\leq (j-1)}^{(i)}(\mathcal{S})|\right)\frac{i!j!}{(i+j)!}\right) \\
&= \sum_{i=1}^{b}\left(\sum_{j=1}^{n-i-1} |\mathcal{F}_{\leq j}^{(i)}(\mathcal{S})| i\frac{i!j!}{(i+j+1)!} + |\mathcal{F}_{\leq (n-i)}^{(i)}(\mathcal{S})|\frac{i!(n-i)!}{n!}\right) \\
&= O\left(\sum_{j=1}^{n} \frac{f_0(\lfloor \frac{n}{j} \rfloor, \mathcal{S})}{j}\right) \text{ by Theorem 2.3.}
\end{aligned}$$

According to Update condition 1, this bound applies also to the size of the I-DAG. □

Expected time

Lemma 4.2 *Under the update conditions, if $\mathcal{S}$ has cardinality n, the expected time for inserting the last object in the I-DAG is*

$$O\left(\frac{1}{n}\sum_{j=1}^{n} f_0(\lfloor \tfrac{n}{j} \rfloor, \mathcal{S})\right)$$

Proof. Under Update condition 2, the computing time spent to locate the last inserted object o is proportional to the total number of nodes of the I-DAG visited when locating o. Due to Update condition 1, the number of nodes visited when locating o is at most proportional to the number of nodes of the I-DAG associated with regions in conflict with o. Thus the expected time for locating the last inserted object o is proportional to the expected number, $\theta(\mathcal{S})$, of nodes of the I-DAG associated with regions in conflict with o.

Let F be a region of $\mathcal{F}_j^{(i)}(\mathcal{S})$. F is a region in conflict with o associated with a node of the I-DAG, if o is one of the j objects in conflict with F (thus the choice, for o, of any of the j objects among the n is possible) and if the i objects defining F have been inserted before the j objects in conflict with F. This occurs with probability $\frac{j}{n}\frac{i!(j-1)!}{(i+j-1)!}$. The expected number $\theta(\mathcal{S})$ of nodes visited during the last insertion is then obtained by summing, for all the regions F of $\mathcal{F}(\mathcal{S})$, the above probability. Using Theorem 2.3, this yields :

$$\theta(\mathcal{S}) = \sum_{i=1}^{b}\sum_{j=1}^{n-i} |\mathcal{F}_j^{(i)}(\mathcal{S})| \frac{i!j!}{n(i+j-1)!} = O\left(\frac{1}{n}\sum_{j=1}^{n} f_0(\lfloor \tfrac{n}{j} \rfloor, \mathcal{S})\right),$$

from a calculation similar to the proof of Lemma 4.1

Due to Update condition 3, the computing time of the last creation substep is also dominated by a term proportional to the number of nodes of the I-DAG associated with a region in conflict with o and admits the same expected upper bound as $\theta(\mathcal{S})$. □

Main theorem

Lemmas 4.1 and 4.2 prove the main result of this section :

Theorem 4.3 *If the set of already inserted objects $\mathcal{S}$ has cardinality n and if the update conditions are fulfilled, the I-DAG of $\mathcal{S}$ requires*

$$O\left(\sum_{j=1}^{n} \frac{f_0(\lfloor \frac{n}{j} \rfloor, \mathcal{S})}{j}\right)$$

expected memory space. The insertion of the n^{th} object can be done in

$$O\left(\frac{1}{n}\sum_{j=1}^{n} f_0(\lfloor \tfrac{n}{j} \rfloor, \mathcal{S})\right)$$

expected update time.

Corollary 4.4 *Under the update conditions, the total expected time to build a I-DAG for a set $\mathcal{S}$ of n objects is :*

$$O\left(\sum_{j=1}^{n} \phi_0(\lfloor \tfrac{n}{j} \rfloor, \mathcal{S})\right)$$

Proof. Notice that in that corollary, $\mathcal{S}$ is no longer the current subset of inserted objects, but the final set of objects. From Theorem 4.3, we know that, if a subset $\mathcal{R}$ of $\mathcal{S}$ with cardinality r has been inserted at a given time, the expected time to insert the last object of $\mathcal{R}$ is : $O\left(\sum_{j=1}^{r} \frac{1}{r} f_0(\lfloor \frac{r}{j} \rfloor, \mathcal{R})\right)$. This expected time accounts for averaging over the $r!$ permutations of the objects of $\mathcal{R}$. Now, the expected time to insert the r^{th} object of $\mathcal{S}$ results from further averaging over the r-random samples of $\mathcal{S}$ which yields : $O\left(\sum_{j=1}^{r} \frac{1}{r} f_0(\lfloor \frac{r}{j} \rfloor, \mathcal{S})\right)$. The proof of Corollary 4.4 results from summing over r from 1 to n, knowing that $\phi_0(r, \mathcal{S})$ is a non decreasing function (recall that $\phi_0(r, \mathcal{S})$ is the maximum of the values $f_0(r', \mathcal{S})$, for $r' \leq r$). □

Two immediate consequences of Theorem 4.3 are the following :

- If $f_0(x, \mathcal{S}) = \Theta(x)$, the space complexity is $O(n)$ and the time to insert a new object is $O(\log n)$.
- If $f_0(x, \mathcal{S}) = \Theta(x^\alpha)$ (with $\alpha \geq 1$), the space complexity is $O\left(n^\alpha\right)$ and the time to insert a new object is $O\left(n^{\alpha-1}\right)$.

4.1.2 Comparison with the complexity of the Conflict graph

This result is the same as the complexity obtained by using the conflict graph, which is

$$O\left(n\sum_{r=1}^{n}\frac{\phi_0(r,\mathcal{S})}{r^2}\right)$$

(see Theorem 2.7).

In fact, $\lfloor\frac{n}{j}\rfloor = r$ for j such that $\frac{n}{r+1} < j \leq \frac{n}{r}$. By changing $\frac{n}{j}$ to r in our result, and grouping partial sums (we sum together all terms corresponding to values of j yielding the same $\lfloor\frac{n}{j}\rfloor$), yields the expression of K.L. Clarkson and P.W. Shor.

Another, maybe more pleasant, way to see the similarity is to look precisely at the total number of nodes traversed in the I-DAG in order to locate an object, during the whole construction : it is the sum, over all regions ever created, of the number of objects in conflict with this region. In other words, it is exactly the number of Conflict graph edges ever created.

The only difference between the two results (independently from the on-line possibility in the I-DAG) lies in the fact that the non amortized complexity of an insertion is totally different : the work done in the Conflict graph is more expensive for the first objects than for the last ones, while it is the contrary in the I-DAG.

4.1.3 Another analysis

The above analysis of the algorithms constructing the I-DAG uses random sampling to bound the number of regions in conflict with at most k objects, and deduces time and space bounds for the algorithms. O. Devillers proposes in [Dev] a simple analysis, using the principle of backwards analysis presented in Section 2.3.3. He avoids the use of random sampling techniques. He also analyzes the Conflict Graph in a similar manner, and applies his results to the design of efficient algorithms combining Conflict and Influence graphs (see Section 7.1).

Lemma 4.5 *The expected value of the number of conflicts of the l^{th} object with regions that had no conflict at stage k ($k < l$) is $\frac{f_1(k+1,\mathcal{S})}{k+1}$.*

> *Proof.* Let ω be one of the $n!$ possible orderings on $\mathcal{S}$. By averaging over ω, the first k objects plus the l^{th} object o may be any sample of size $k+1$ with the same probability. Suppose thus that o is introduced immediately after the first k elements. Then the regions that were defined by those k elements and that are in conflict with o, are

determined by a subset of those $k+1$ elements and their width is 1 after the insertion of $\circ$. Finally, $\circ$ may be any element of the sample of size $k+1$, with probability $\frac{1}{k+1}$, which yields the result. □

Lemma 4.6 *The expected number of regions created by the insertion of the k^{th} object is less than $\frac{bf_0(k,\mathcal{S})}{k}$.*

Proof. A region created by the k^{th} object $\circ$ is a region defined by these k objects and without any conflict with them. In the same way as done in the preceding lemma, we can say that the first k objects may be any sample of $\mathcal{S}$ of size k with the same probability. Then $\frac{b}{k}$ is an upper bound for the probability that $\circ$ be one of the at most b objects determining a region. □

Lemma 4.7 *The expected value of the number of conflicts of the l^{th} object with the regions created by the insertion of the k^{th} object ($k < l$) is less than $\frac{b}{k}\frac{f_1(k+1,\mathcal{S})}{k+1}$.*

Proof. Let us denote $E_{k,l}$ this expected value. $E_{k,l}$ is the sum, over all regions F defined by $\mathcal{S}$, of the probability $p_{F,k,l}$ that F be created by the insertion of the k^{th} object and in conflict with the l^{th} object $\circ$.

Let us decompose this probability $p_{F,k,l}$, conditionally with the event that F is in conflict with $\circ$ and had no conflict at stage k. In the case where F had no conflict at stage k, the probability that the object that created F be the last among the k objects is less than $\frac{b}{k}$, as in the preceding lemma. In the case where F had a conflict at stage k, F could not have been created by the k^{th} object, so this case does not contribute to the value of $p_{F,k,l}$. So we conclude by using Lemma 4.5, which gives the expectation of the conditional event. □

Theorem 4.8 *The complexity of the operations on the Influence graph are the following :*

(1) The expected size of the Influence graph at stage k is less than $\sum_{j=0}^{k} \frac{bf_0(j,\mathcal{S})}{j}$.

(2) The expected cost of inserting the l^{th} object in the Influence graph is less than $\sum_{j=0}^{l-1} \frac{b}{j}\frac{f_1(j+1,\mathcal{S})}{j+1}$.

(3) The expected cost of inserting the l^{th} object in the Influence graph, knowing the conflicts of this object with the regions that had no conflict at stage k, is less than $\sum_{j=k}^{l-1} \frac{b}{j}\frac{f_1(j+1,\mathcal{S})}{j+1}$.

The result *(3)* is the central point that allows to build accelerated algorithms (Section 7.1).

Proof.

(1) Since the number of sons is bounded, the size of the influence graph is proportional to its number of nodes. This number is simply the sum over all the regions of the probability for a region to be a node of the graph. By Lemma 4.6 the expected number of nodes created at stage j is less than $\frac{bf_0(j,\mathcal{S})}{j}$.

(2) During the insertion of the l^{th} object, the conflicts are located by a traversal of the influence graph. A node F is visited, and yields a positive answer, if it is in conflict with the l^{th} object. By summing over the stage of creation j of F we get $\left(\sum_{j=1}^{l-1} E_{j,l}\right)$. According to update conditions, the number of visited nodes, with positive answer, in the influence graph is linearly related to the cost of the insertion.

(3) Same result starting the summation at $j = k$.

□

In the applications $f_0(r, \mathcal{S})$ and $f_1(r, \mathcal{S})$ are often both linear. In such a case, the complexities get a more explicit expression stated in the following theorem. Furthermore, if a direct expression of $f_1(r, \mathcal{S})$ is not available, it is possible to show (see [CS89], Theorem 2.3 and Remark 2.4) that $f_1(r, \mathcal{S}) = O\left(f_0\left(\lfloor \frac{r}{2} \rfloor, \mathcal{S}\right)\right)$, so it is sufficient to suppose that $f_0(r, \mathcal{S})$ is linear.

Theorem 4.9 *If $f_0(r, \mathcal{S}) = O(r)$,*

(1) The expected size of the influence graph at stage k is $O(k)$.

(2) The expected cost of inserting the l^{th} object in the influence graph is $O(\log l)$. The whole cost of the algorithm is $O\left(\sum_{l=1}^{n} \log l\right) = O(n \log n)$.

(3) The expected cost of inserting the l^{th} object in the influence graph, knowing the conflicts of this object with the regions that had no conflict at stage k, is $O\left(\log \frac{l}{k}\right)$.

Proof. This theorem is simply a corollary of Theorem 4.8. □

4.1.4 Removing the update conditions

In some cases the three update conditions can be removed. We will show in Section 4.3 that removing the update conditions (especially Condition 1, that is not verified in the case of the Delaunay Tree) will lead, in some cases, to simpler algorithms.

Constant test time and linear update time

Update conditions 2 and 3 can be relaxed. If the time required to check conflict between a region and an object exceeds $O(1)$, the overcost will simply appear as a multiplicative factor in the overall complexity. If the time required by the creation substep surpasses a linear function of the number of conflicts, it is in general not difficult to charge the overcost to the overall complexity. This kind of analysis will be done for example for the incremental construction of higher order Voronoi diagrams (see Section 5.1.3.5).

Bounded number of sons

It is more interesting to attempt to remove Update condition 1. The preceding analysis works because all the relevant quantities can be expressed as functions of the number of nodes. If the condition is not fulfilled, we must count the number of edges of the I-DAG and to that purpose, we introduce the notion of bicycles. A *bicycle* is a pair of regions of $\mathcal{F}(\mathcal{S})$ occurring as a father and one of its sons in the I-DAG associated with at least one permutation of the object set $\mathcal{S}$.

Notice that the maximum number of objects defining a bicycle is at most $2b$ and thus is still bounded. An object is in conflict with a bicycle if it is not any of the objects that determine the bicycle and if it is in conflict with at least one of the two regions forming the bicycle. Thus the influence range of a bicycle is the union of the influence ranges of the two regions forming the bicycle, excepting the objects defining these regions.

In analogy with the notation used for regions, the additional notations $\mathcal{G}_j(\mathcal{S})$ (for the set of bicycles defined by $\mathcal{S}$ of width j), $\mathcal{G}_{\leq j}(\mathcal{S})$ (for bicycles of width at most j)... are naturally derived. We define $g_0(r, \mathcal{S})$ to be the expected size of $\mathcal{G}_0(\mathcal{R})$ for r-random samples $\mathcal{R}$ of $\mathcal{S}$.

With these definitions, the following lemma can be proved, using the random sampling technique, in a way similar to Theorem 2.3.

Lemma 4.10

$$|\mathcal{G}_{\leq j}(\mathcal{S})| \quad = \quad O\left(j^{2b} g_0(\lfloor \tfrac{n}{j} \rfloor, \mathcal{S})\right)$$

We can now compute the expected storage required by the I-DAG :

Lemma 4.11 *If $\mathcal{S}$ has cardinality n, the expected size of the I-DAG of $\mathcal{S}$ is :*

$$O\left(\sum_{j=1}^{n} \frac{g_0(\lfloor \frac{n}{j} \rfloor, \mathcal{S})}{j}\right)$$

Proof. The size of the I-DAG is linearly related to the number of its edges. A necessary condition for a given bicycle G to occur as an edge in the I-DAG is that the i objects defining G are inserted before the j objects in conflict with G (i.e. in conflict with one of the two regions associated with G). So the probability that $G \in \mathcal{G}(\mathcal{S})$ arises in the I-DAG is less than $\frac{i!j!}{(i+j)!}$. Calculations analogous to those appearing in the proof of Lemma 4.1 yield the result. □

Lemma 4.12 *The expected time for inserting the n^{th} object in the I-DAG is*

$$O\left(\frac{1}{n}\sum_{j=1}^{n} g_0(\lfloor \tfrac{n}{j} \rfloor, \mathcal{S})\right)$$

Proof. Let o be the n^{th} object to be inserted. The number of times a node is visited is equal to the number of its parents which are in conflict with o. Thus the number of performed tests is no more than the number of bicycles in conflict with o occurring in the I-DAG.

Let G be a bicycle of $\mathcal{G}_j^{(i)}(\mathcal{S})$. G is in conflict with o and occurs as an edge in the I-DAG if the following two necessary conditions are satisfied : the i objects defining G are inserted before the j objects in conflict with G, and one of the j objects is o. The probability that G is in conflict with o is thus no more than $\frac{j}{n}\frac{i!(j-1)!}{(i+j-1)!}$. The result is then achieved as in Lemma 4.2. □

Lemmas 4.11 and 4.12 prove the main results of this section, generalizing Theorem 4.3 and Corollary 4.4 :

Theorem 4.13 *If the set of already inserted objects $\mathcal{S}$ has cardinality n and if Update conditions 2 and 3 are fulfilled (but not Update condition 1), the I-DAG of $\mathcal{S}$ requires*

$$O\left(\sum_{j=1}^{n} \frac{g_0(\lfloor \frac{n}{j} \rfloor, \mathcal{S})}{j}\right)$$

expected memory space. The insertion of the n^{th} object can be done in

$$O\left(\frac{1}{n}\sum_{j=1}^{n} g_0(\lfloor \tfrac{n}{j} \rfloor, \mathcal{S})\right)$$

expected update time.

Corollary 4.14 *Under Update conditions 2 and 3, the total expected time to build a I-DAG for a set $\mathcal{S}$ of n object is :*

$$O\left(\sum_{j=1}^{n} \gamma_0(\lfloor \tfrac{n}{j} \rfloor, \mathcal{S})\right)$$

where $\gamma_0(r, \mathcal{S})$ denotes the maximum value of $g_0(r', \mathcal{S})$ for $r' \leq r$.

4.2 Locating with the influence graph

4.2.1 Faster object location

If the following additional property is verified, it is possible to get a better complexity result for the search of a single region in conflict with a new object.

Property 3 The influence range of the region associated to a node is included in the union of the ranges of its children.

If Property 3 holds, then a conflict with a given new object can be found by following a simple path from the root of the I-DAG to a leaf.

Theorem 4.15 *If Property 3 holds then a conflict with* any *new object can be found in $O(\log n)$ time provided that the n objects that were inserted in the I-DAG have been inserted in random order.*

Proof. Let o be the new object and F be a region on the path from the root to a leaf of the I-DAG in conflict with o. Suppose that F has zero width after the insertion of the k^{th} object. If F is determined by i objects, the conditional probability that F has been created during the insertion of the k^{th} object is $\frac{i}{k} \leq \frac{b}{k}$. Indeed, for F to be created at step k, the k^{th} inserted object must be one of the i objects defining F. (It is important to notice that the above probability is conditioned by the fact that F has width zero after the insertion of the first k objects.)

Averaging this probability over the $\binom{n}{k}$ possible subsets of k objects introduced first in the I-DAG yields a probability less than $\frac{b}{k}$ that the node on the path would change after the insertion of the k^{th} object. Thus the number of visited nodes is less than $\sum \frac{b}{k} = O(\log n)$. □

Let us make some remarks about Theorem 4.15. First it is important to notice that the cost studied here is expected over all possible orders to insert the objects in the I-DAG, but there is no hypothesis on the new object, by opposition to Lemma 4.2 where all objects, including the last one, are supposed to satisfy the randomization hypothesis. Secondly, in many applications, it is possible to find all conflicts with a new object from a single one in time proportional to the actual number of conflicts (for example by the use of some neigborhood notions). The faster location may be used as a first step of the insertion of a new object in the I-DAG. A last remark concerns the worst case ; though we are interested in randomized complexities, the faster location has a worst case running time $O(n)$ which is better than the general location step.

4.2.2 Queries

In some applications, queries consist in finding the regions having zero width which are in conflict with a given element of the object universe : this is just a special instance of a location substep. In such cases, the I-DAG can be used and the randomized analysis of Theorem 4.3 holds, provided that the query object q together with the set of objects $\mathcal{S}$ introduced in the I-DAG satisfy the randomization hypothesis, i.e. the $(n+1)!$ permutations of $\{\mathtt{q}\} \cup \mathcal{S}$ are likely to occur.

In other applications, regions and queries are such that the answer to a query consists of exactly one region of $\mathcal{F}_0(\mathcal{S})$ for any set $\mathcal{S}$. Such a query will be answered by a location substep that will traverse only one path from the root to a leaf. This yields the following strong variant of Lemma 4.2.

Theorem 4.16 *Assume that regions and queries are such that the answer to a query consists of exactly one region of $\mathcal{F}_0(\mathcal{S})$ for any set $\mathcal{S}$. Then* any *query can be answered in $O(\log n)$ time provided that the n objects that were inserted in the I-DAG have been inserted in random order.*

Proof. The proof is similar to the proof of Theorem 4.15. □

4.3 Applications

We study several cases in which the update conditions are fulfilled : one algorithm for convex hulls (Section 4.3.1), arrangements (Section 4.3.2) and Voronoi diagrams of line segments (Section 4.3.3.2). We will also see cases where they are not fulfilled : another algorithm for convex hulls, and the case Voronoi diagrams (Section 4.3.3.1).

4.3.1 Convex hulls

We consider the geometric problem of computing the convex hull of a set of points. We present in this section two algorithms that are both on-line and whose expected performances are optimal in any dimension. For simplicity, we expose here firstly the two dimensional case. The extension to higher dimensional spaces is quite straightforward and will be shortly described next.

First algorithm

Objects are points of the plane. Regions are determined by three points. The region (p,q,r) associated with p, q and r consists of the union of two half-planes : one bounded by line (pq) and not containing r and the other bounded by line (qr) and not containing p (see Figure 4.1). An object is in conflict with a region if and only if it lies in the region.

Now let p, q and r be three points in $\mathcal{S}$, the set of points already inserted. The region associated with these points has zero width if and only if p, q and r are three consecutive vertices of the convex hull. So computing the convex hull of $\mathcal{S}$ is equivalent to computing the zero width regions.

Let us now describe the algorithm. Suppose that the I-DAG has been constructed for the points in $\mathcal{S}$ and that we want to insert a new point m. The location substep gives the regions of $\mathcal{F}_0(\mathcal{S})$ containing m. If m belongs to the interior of the convex hull, there is no such region, $\mathcal{F}_0(\mathcal{S} \cup \{\mathtt{m}\}) = \mathcal{F}_0(\mathcal{S})$ and the I-DAG is not

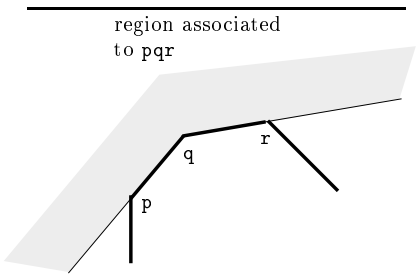


Figure 4.1 : Definitions of regions for the convex hull problem

modified. Otherwise, let $\mathtt{p}_1$ and $\mathtt{p}_k$ be the two vertices adjacent to $\mathtt{m}$ in the new convex hull and $\mathtt{p}_2, \ldots, \mathtt{p}_{k-1}$ the chain of vertices which are no longer vertices of the convex hull after the insertion of $\mathtt{m}$ (see Figure 4.2). The regions of $\mathcal{F}_0(\mathcal{S})$

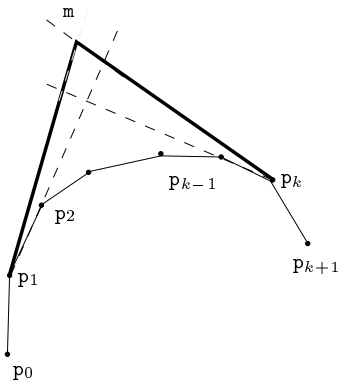


Figure 4.2 : Inserting a new point in the convex hull

containing $\mathtt{m}$ are $(\mathtt{p}_{l-1}, \mathtt{p}_l, \mathtt{p}_{l+1})$, for $1 \leq l \leq k$. By a simple test on these regions, we can determine $(\mathtt{p}_0, \mathtt{p}_1, \mathtt{p}_2)$ and $(\mathtt{p}_{k-1}, \mathtt{p}_k, \mathtt{p}_{k+1})$ ($\mathtt{m}$ belongs to only one of the two half-planes defining the region). The I-DAG is then modified in the following manner : all the $(\mathtt{p}_{l-1}, \mathtt{p}_l, \mathtt{p}_{l+1})$, for $1 \leq l \leq k$, are killed by $\mathtt{m}$, their width are incremented and three new regions are created, namely

- $(\mathtt{p}_0, \mathtt{p}_1, \mathtt{m})$ as a son of $(\mathtt{p}_0, \mathtt{p}_1, \mathtt{p}_2)$
- $(\mathtt{m}, \mathtt{p}_k, \mathtt{p}_{k+1})$ as a son of $(\mathtt{p}_{k-1}, \mathtt{p}_k, \mathtt{p}_{k+1})$
- $(\mathtt{p}_1, \mathtt{m}, \mathtt{p}_k)$ as a son of both $(\mathtt{p}_0, \mathtt{p}_1, \mathtt{p}_2)$ and $(\mathtt{p}_{k-1}, \mathtt{p}_k, \mathtt{p}_{k+1})$.

It is clear that the properties of the I-DAG are preserved and that the update conditions are satisfied. Here $f_0(r, \mathcal{S})$ is the expected size of the convex hull of r points of $\mathcal{S}$ which is clearly $O(r)$, so applying Theorem 4.3 we deduce :

Proposition 4.17 *The convex hull of n points in the plane can be computed on-line with $O(n)$ expected space and $O(\log n)$ expected update time.*

These results can be generalized to any dimension. The regions are determined by $d+1$ points and are unions of two half-spaces. The zero width regions correspond to $(d-2)$-faces of the convex hull and their two adjacent $(d-1)$-faces. The expected size $f_0(r, \mathcal{S})$ of the convex hull of r points is $O\left(r^{\lfloor \frac{d}{2} \rfloor}\right)$.

Proposition 4.18 *The convex hull of n points in d-space can be computed on-line with $O\left(n^{\lfloor \frac{d}{2} \rfloor}\right)$ expected space and $O(\log n)$ expected update time if $d \leq 3$ and $O\left(n^{\lfloor \frac{d}{2} \rfloor - 1}\right)$ expected update time if $d > 3$.*

These results are optimal.

As far as queries are concerned, the above results show that one can decide if a point lies inside or outside the convex hull of n points in $O(\log n)$ expected time in the plane and $O\left(n^{\lfloor \frac{d}{2} \rfloor - 1}\right)$ expected time in d-space.

Second algorithm

It might look more natural to take half-spaces as regions. In that case, as will become clear below, the number of sons is not bounded and Update condition 1 is not satisfied. However the result of Section 4.1.4 proves that the resulting algorithm has the same complexity as the one above.

We describe the algorithm for the two dimensional case. It can be generalized to any dimension with no difficulty. $\mathcal{O}$ still denotes the set of points of the plane. Regions are now determined by only two points. The regions $(\mathtt{p}, \mathtt{q})$ and $(\mathtt{q}, \mathtt{p})$ are the two half planes limited by the line $(\mathtt{pq})$. A point is in conflict with the

region $(\mathtt{p},\mathtt{q})$ if it lies inside the corresponding half-plane. If a region $(\mathtt{p},\mathtt{q})$ has zero width, then $[\mathtt{pq}]$ is an edge of the convex hull.

In addition to the standard information stored in each node of the I-DAG, we also maintain at each leaf which is associated with an edge E of the convex hull two pointers towards the two leaves associated with the two edges of the convex hull adjacent to E. When a new point $\mathtt{m}$ is inserted, the location substep provides all the half-planes with current width 0 in conflict with $\mathtt{m}$. These half-planes correspond to a chain of edges of the convex hull $[\mathtt{p}_1\mathtt{p}_2],\ldots,[\mathtt{p}_{k-1}\mathtt{p}_k]$ (see Figure 4.2). The two extremal edges $[\mathtt{p}_1\mathtt{p}_2]$ and $[\mathtt{p}_{k-1}\mathtt{p}_k]$ are identified, by testing if their two neighbors are not both in conflict with $\mathtt{m}$. Two new regions $(\mathtt{p}_1,\mathtt{m})$ and $(\mathtt{m},\mathtt{p}_k)$ are created. In order to satisfy Property 2, $(\mathtt{p}_0,\mathtt{p}_1)$ and $(\mathtt{p}_1,\mathtt{p}_2)$ are made parents of $(\mathtt{p}_1,\mathtt{m})$; similarly $(\mathtt{p}_{k-1},\mathtt{p}_k)$ and $(\mathtt{p}_k,\mathtt{p}_{k+1})$ are made parents of $(\mathtt{m},\mathtt{p}_k)$. The neighborhood relationships are updated : $(\mathtt{p}_0,\mathtt{p}_1)$ and $(\mathtt{p}_1,\mathtt{m})$ become adjacent and similarly $(\mathtt{p}_k,\mathtt{p}_{k+1})$ and $(\mathtt{m},\mathtt{p}_k)$, and $(\mathtt{p}_1,\mathtt{m})$ and $(\mathtt{m},\mathtt{p}_k)$.

Notice than the width of $(\mathtt{p}_0,\mathtt{p}_1)$ is still zero and that this region may have other sons in the future : Update condition 1 is not satisfied.

As described in Section 4.1.4, we introduce the notion of a bicycle. Here a bicycle is defined by two regions $(\mathtt{p},\mathtt{q})$ and $(\mathtt{q},\mathtt{r})$ sharing a point of definition (a bicycle here is a region of the previous section ; see Figure 4.1). The zero width bicycles are of two kinds. The first ones are associated with two regions with zero width : $(\mathtt{p}_0,\mathtt{p}_1)$ and $(\mathtt{p}_1,\mathtt{m})$ in Figure 4.2. They correspond to two consecutive edges of the convex hull. The second ones are associated with a region of zero width and a region in conflict with the additional point of definition of the other : $(\mathtt{p}_1,\mathtt{p}_2)$ and $(\mathtt{p}_1,\mathtt{m})$ in Figure 4.2. They correspond to an edge E of the convex hull and to an edge E', incident to one of the endpoint of E and whose supporting line separates a 1-set of $\mathcal{S}$. Using the results on the number of k-sets (see, for instance, [Ede87]) we conclude that $g_0(r)$ is $O\left(r^{\lfloor\frac{d}{2}\rfloor}\right)$. Thus Theorem 4.13 implies that this simpler algorithm has the same complexity as the algorithm of the previous section.

Experimental results

The second algorithm has been coded in both dimensions 3 and 4. The results appear to be really good in the case where there are a lot of sites on the convex hull. In the case where the number of faces of the convex hull is small, compared to the number of sites, a lot of points must be located in the I-DAG, without creating any new result (this problem will not occur in the case of Voronoi diagrams, as will be seen next). Observe however that, in most cases, the number of nodes visited to locate a site is much smaller than the size of the current convex hull. Compare for example the results of Figures 4.3 and 4.4 in 3D.

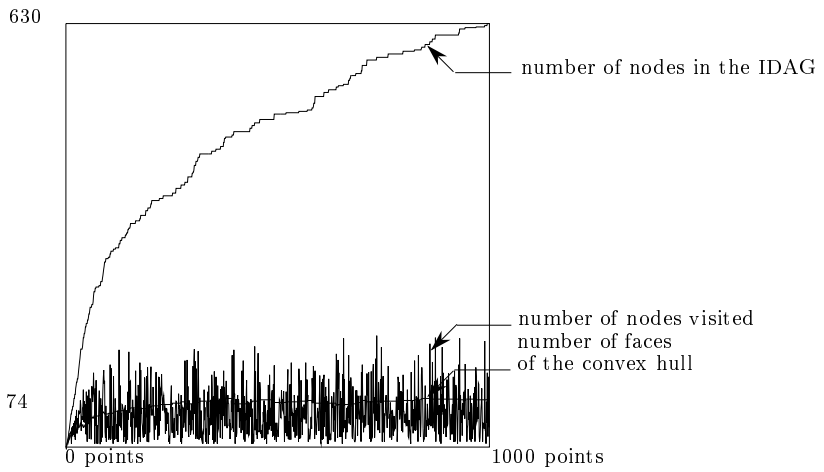


Figure 4.3 : Convex hull : 1000 sites in the interior of a 3D-cube

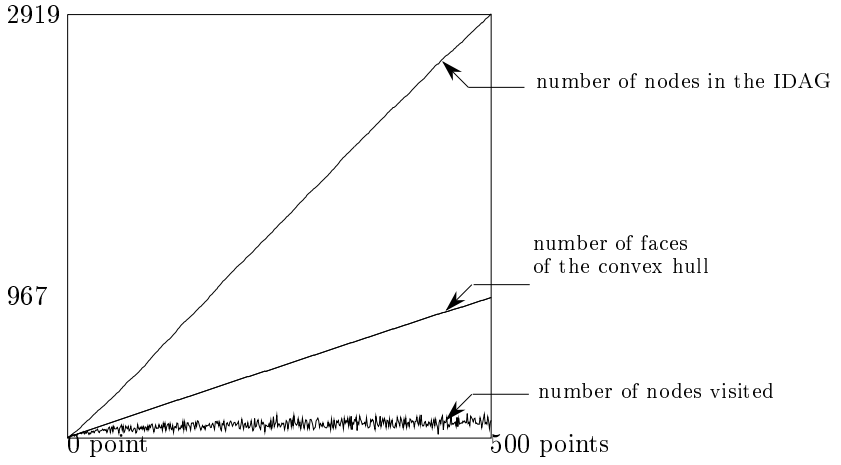


Figure 4.4 : Convex hull : 500 sites on the surface of a 3D-sphere

Figure 4.5 shows the results obtained for a set of points lying on a closed surface in 3D. Finally, Figures 4.6 and 4.7 show results in dimension 4.

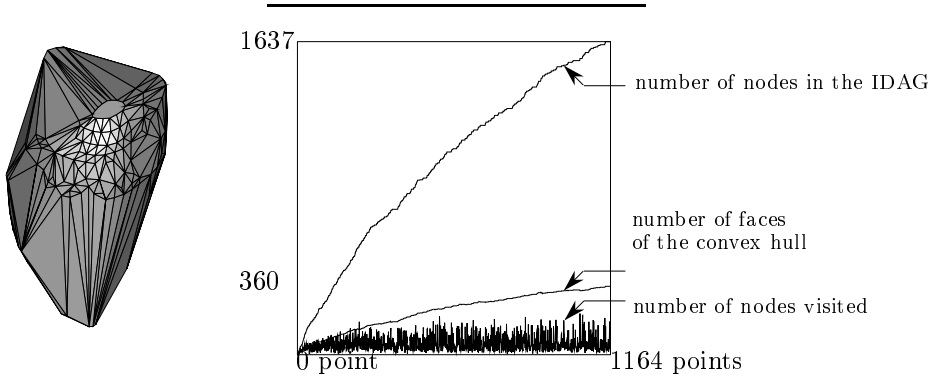


Figure 4.5 : Convex hull : Points lying on the surface of a heart

4.3.2 Arrangements

Let us consider first the case of line segments. The general framework of Section 4.1 can be applied to solve this problem. The algorithm builds the trapezoidal map of $\mathcal{S}$ (see Section 1.3.2). Objects are here line segments and regions are trapezoids (i.e. cells of the trapezoidal diagram). A trapezoid is determined by at most four segments. A line segment and a trapezoid are in conflict if and only if the segment intersects the interior of the trapezoid. For each leaf of the I-DAG, we store also some neighbors of the corresponding zero width trapezoid. More precisely, we store two kinds of adjacency relationships : adjacent trapezoids through vertical edges of the map are called *horizontal neighbors* and adjacent trapezoid through line segments are called *up* or *down neighbors*. In general position a leaf of the I-DAG has at most four horizontal neighbors and an arbitrary number of up and down neighbors.

When a new segment `s` is inserted, we traverse the I-DAG to collect the set $\mathcal{L}(\mathtt{s})$ of all the zero width trapezoids in conflict with `s`. Each such region is subdivided into at most four subregions and these subregions are possibly merged to form new trapezoids (vertical segments intersecting `s` have to be shortened) which is easily done using the adjacency relationships stored in the nodes of the I-DAG. The resulting trapezoids are the new nodes we attach to the I-DAG. The

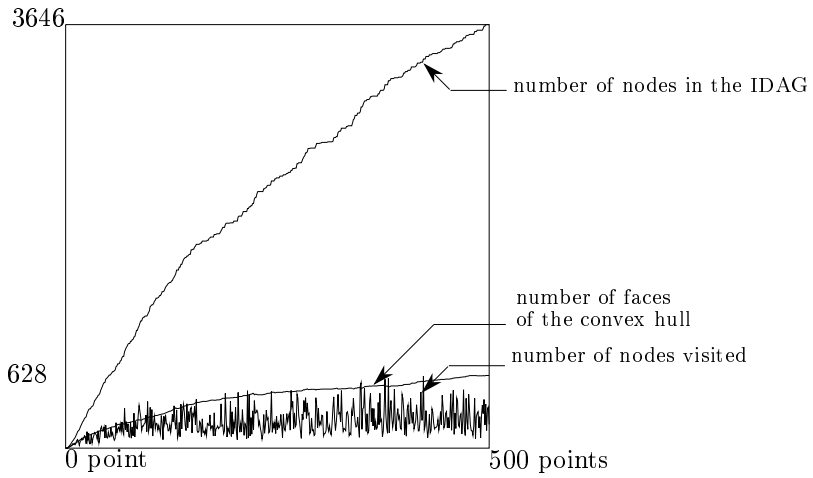


Figure 4.6 : Convex hull : Points lying in the interior of a 4 dimensional cube

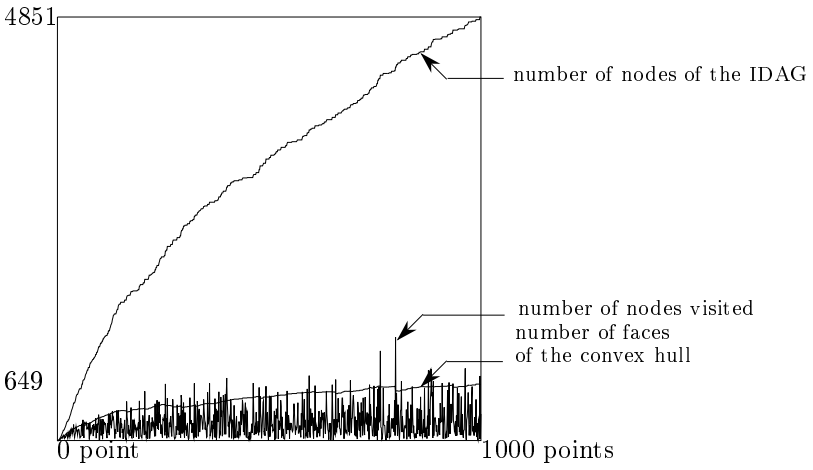


Figure 4.7 : Convex hull : Points lying on the surface of a 4 dimensional cube

parents of a new node are the trapezoids of $\mathcal{L}(S)$ which intersect that node (see Figure 4.8). Notice that a trapezoid may have several parents, for example in this figure, a portion of 4 is merged with a portion of 6 to form c ; thus 4 and 6 are together parents of c.

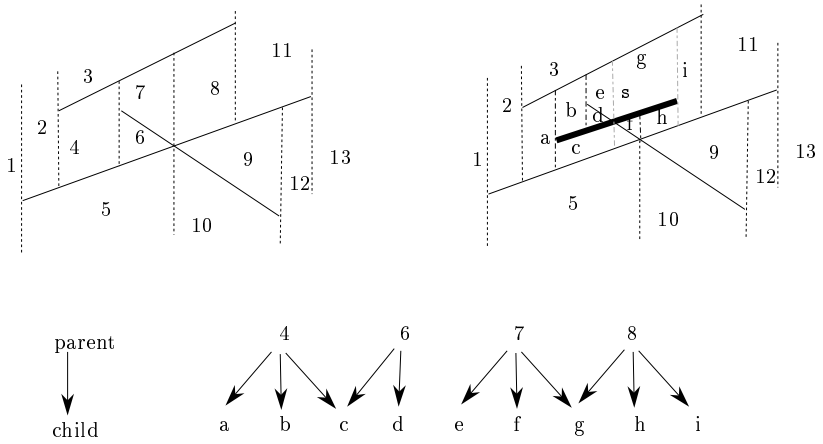


Figure 4.8 : Insertion of s in the Influence Graph

Let us detail the whole information stored in a node of the graph.
First a node corresponds to a trapezoid T, so we store the four segments determining T ; one of them is its creator.
In order to merge trapezoids during the insertion of a segment, we need to store horizontal neighbors, that is at most 2 left and 2 right neighbors, and links to some parents.
For the removal of a segment (see Section 6.2), we will need to store for each trapezoid some vertical neighbors : the up-right neighbor, namely its up neighbor adjacent to its right side neighbor, and the up-left, down-right and down-left neighbors, at the time when the trapezoid was created. To initialize these neighbors, for a new trapezoid, during an insertion, we need to know all current up and down neighbors of its parents. We have two ways to perform this : the first one is to maintain all those vertical neighbors for each trapezoid in the map ; the second one is to look for them each time when we need them ; to do so, we only need one of them, and find the other ones using horizontal neighbors. The first solution makes the structure of a node rather heavy, and we can easily see that the second solution does not increase the time complexity and gives all necessary information

[CS89]. So we will consider that we know every up and down neighbor.

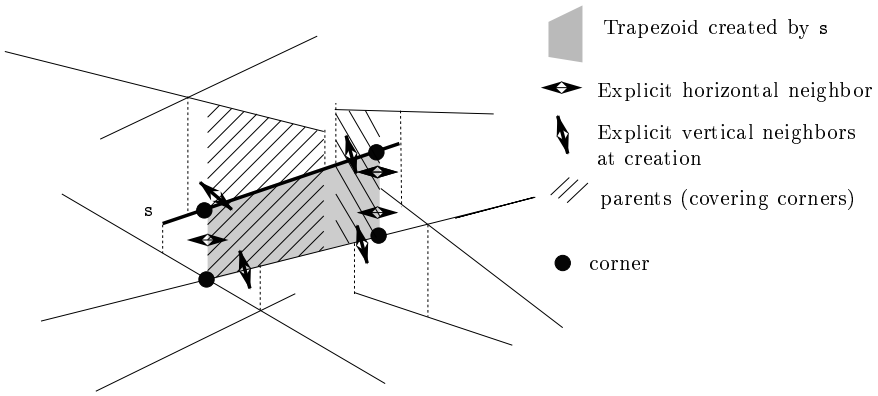


Figure 4.9 : A trapezoid, its neighbors and parents

Let us summarize the structure of a node (see Figure 4.9) :
• at most 4 segments defining it (one of them is the creator)
• the at most 4 horizontal current neighbors
• its at most 4 children
• its killer
• its at most 4 corners (i.e. its vertices)
• all current vertical neighbors (they are not explicitly stored)
• one vertical neighbor at creation per corner
• at most 4 parents : for each corner, the parent covering it
If the node is dead, all current neighbors become neighbors at the death.

Figure 4.10 shows the insertion of a new segment `s`. `s` is the new segment and T is a node of the Influence Graph. For the first call to `Insert`, T is the root of the Influence Graph.

A node has at most four sons so that the update conditions are fulfilled (but the number of parents for a trapezoid is unbounded). An easy lemma, proved in [CS89], shows that $f_0(r, \mathcal{S}) = O\left(r + a\frac{r^2}{n^2}\right)$. In fact, if $\mathcal{R}$ is a random subset of $\mathcal{S}$ of size r, since the trapezoidal map of $\mathcal{R}$ is a planar map, the number of its trapezoids is $O(n + a(\mathcal{R}))$, where $a(\mathcal{R})$ denotes the number of intersecting

```
Insert(s, T)
    if s is not in conflict with T
        return ;
    if T is dead then
        for each child S of T Insert(s, S) ;
    else
        s is the killer of T ;
        split T into pieces, that become children of T ;
        deduce the neighbors of the children from the neighbors of T ;
        for each child of T
            if it has an intersection as a corner
                put a parent link from it to T ;
            look for a merge with horizontal neighbors ;
            if it has no parent pointer then link it to T ;
        update all neighbor relations of the neighbors of T ;
```

Figure 4.10 : Insertion of a new segment

pairs of segments of $\mathcal{R}$. Let $\mathtt{z}$ be an intersection between two segments of $\mathcal{S}$, the probability that $\mathtt{z}$ be an intersection between segments of $\mathcal{R}$ is $\frac{\binom{r}{2}}{\binom{n}{2}}$, since the two segments meeting at $\mathtt{z}$ must both be in $\mathcal{R}$. The result follows from summing this probability for all intersections between segments of $\mathcal{S}$.

This result can be readily extended to planar arrangements of curves of bounded degree : the expression for $f_0(r, \mathcal{S})$ is the same, and the number of sons of a node is bounded by a constant depending on the kind of curves considered (it is 11 for circles, for example).

Proposition 4.19 *An arrangement of n planar curves (of bounded degree) can be computed on-line with $O(n+a)$ expected space and $O\left(\log n + \frac{a}{n}\right)$ expected update time, where a is the complexity of the arrangement.*

We achieve the same complexity as Clarkson and Shor [CS89] and Mulmuley [Mul89b], whose algorithms use the conflict graph, and thus are static. The trapezoids with zero width partition the plane, so that Theorem 4.16 applies :

Proposition 4.20 *A point can be located in an arrangement of n planar curves (of bounded degree) in $O(\log n)$ expected time using $O(n+a)$ expected space and $O(n \log n + a)$ expected preprocessing time, where a is the complexity of the arrangement.*

4.3.3 Voronoi diagrams

4.3.3.1 Voronoi diagrams of point sites in $\mathbb{E}^d$

Using the well known correspondence between Voronoi diagrams in d dimensions and convex hulls in $d+1$ dimensions (see Section 1.4), we immediately deduce from the previous section two optimal on-line algorithms to construct the Voronoi diagrams of points in any dimension.

A direct presentation that does not use this correspondence has already been described in detail in Chapter 3 : the Delaunay Tree is in fact nothing else but an Influence Graph. Here, Update condition 1 is not fulfilled, so, for the analysis, we need to define a bicycle as a pair of simplices sharing a facet. We use the fact that the number of simplices arising in a Delaunay triangulation of n sites is $O\left(n^{\lfloor \frac{d+1}{2} \rfloor}\right)$ (see for example [Kle80]) and thus, that the number of bicycles of width zero has the same complexity (d is fixed). The analysis given in Theorem 4.13 allows to state :

Proposition 4.21 *The Delaunay triangulation of n points in d-space can be computed on-line with $O\left(n^{\lfloor \frac{d+1}{2} \rfloor}\right)$ expected space in dimension $d \geq 2$. The expected update time is $O(\log n)$ if $d = 2$ and $O\left(n^{\lfloor \frac{d+1}{2} \rfloor - 1}\right)$ if $d \geq 3$* [1].

These results are optimal.

Remarque 4.22 The results stated in Proposition 4.21 are obtained by using worst-case bounds for the size of a Delaunay triangulation ($O\left(n^{\lfloor \frac{d+1}{2} \rfloor}\right)$ is used to bound $|\mathcal{F}_0(\mathcal{S})|$).

If we assume that the set of sites to be triangulated is uniformly distributed in the space, we know from the results of [Dwy91] that $|\mathcal{F}_0(\mathcal{S})|$ (and thus $|\mathcal{G}_0(\mathcal{S})|$) is $O(n)$. Consequently, the random sampling technique used in Theorem 2.3 provides a linear complexity for the size of $\mathcal{F}_{\leq j}(\mathcal{S})$ and $\mathcal{G}_{\leq j}(\mathcal{S})$ and our algorithm runs in $O(n \log n)$ expected time in any dimension.

Experimental results

The algorithm has been implemented in both dimensions 2 and 3. It is to be noted that the algorithm is extremely simple.

Moreover, the numerical computations involved are also very simple : when a simplex is created, we can compute the coordinates of the center of its circumscribing sphere, and its squared radius. This is achieved by first writing the equation of a sphere in d dimensions, then writing that the $d+1$ sites defining the simplex belong to the sphere, and solving the linear system which results from that, with $d+1$ equations and $d+1$ unknowns, in $O(d^3)$ time.
The center and the squared radius of a simplex is computed once for all and stored in the corresponding node. For testing if a site is in conflict with a simplex, we only have to compute its squared distance to the center of the simplex, which costs $O(d)$, and compare it with the squared radius. The constant in Footnote 1 has been computed according to this method.

The algorithm has run on many examples with different kinds of point distributions, in two and three dimensions.

[1] The constants depend on the dimension as a $\frac{(d+5)!}{\lceil \frac{d}{2} \rceil !}$ factor for the time complexity and a $\frac{1}{(\lceil \frac{d}{2} \rceil - 1)!}$ factor for the space complexity. We will omit the computations of these constants here.

In each case, several orders of insertion of the sites have been tried to analyze the running time down to the constants. All randomized orders that we experimented yielded roughly the same results.

Some results are presented on Figures 4.11, 4.12 and 4.13 for the planar case. See also some related figures in Chapter 5 (5.8 to 5.16) and in Chapter 6 (6.9 to 6.11).

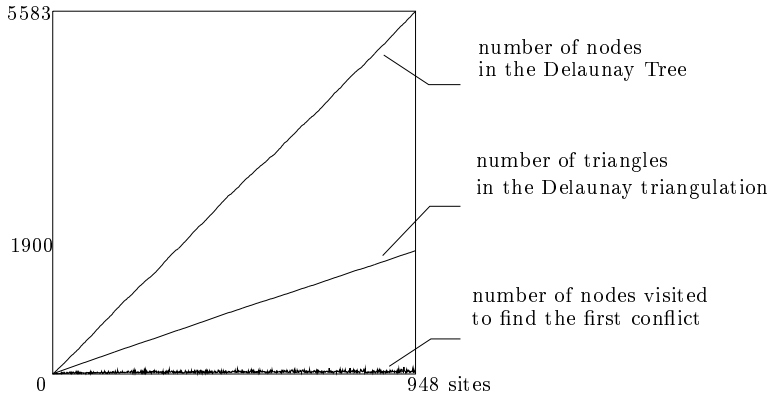


Figure 4.11 : Voronoi diagram : Random sites in the plane

Figures 4.14, 4.15 and 4.16 present results in 3-space. Figures 4.14 and 4.15 show results in the case where the size of the final Delaunay triangulation is linear in the number of sites, whereas Figure 4.16 was obtained for a distribution where the number of tetrahedra was quadratic (the sites are lying on two non-coplanar line segments).

Storage of the Delaunay Tree

The number of nodes in the Delaunay Tree has been studied with respect to the number of inserted sites. This must be compared with the number of simplices in the final triangulation, which is the size of the output.

In the two dimensional case, the ratio between the number of nodes in the Delaunay Tree to the size of the output is less than 3, in any example. In the space, this ratio becomes 4 in the case where the size of the triangulation is linear, and only 2 in the quadratic case.

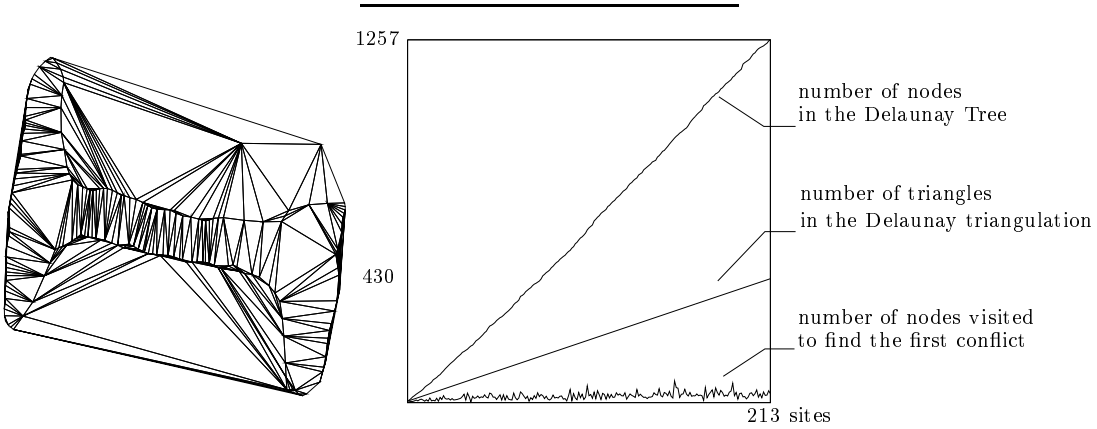


Figure 4.12 : Voronoi diagram : A non-convex curve

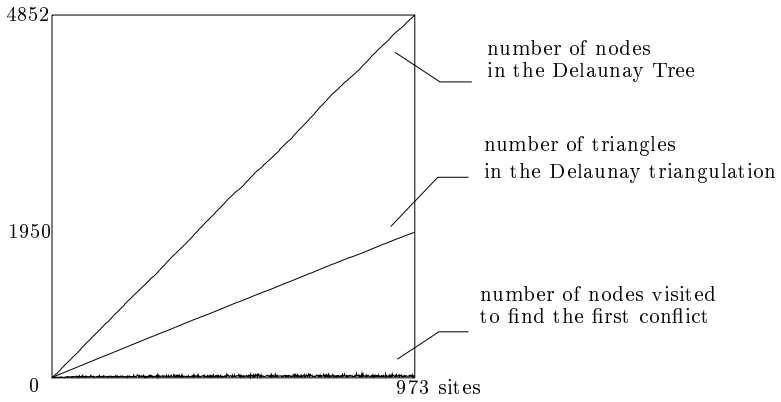


Figure 4.13 : Voronoi diagram : An ellipsis

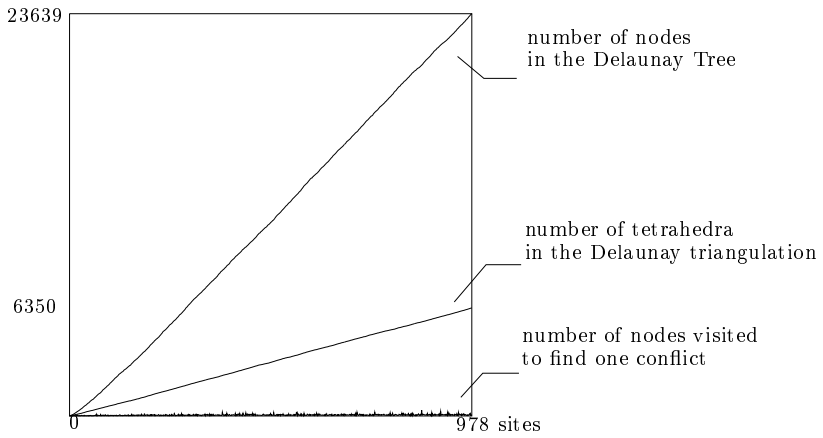


Figure 4.14 : Voronoi diagram : Random sites in 3-space

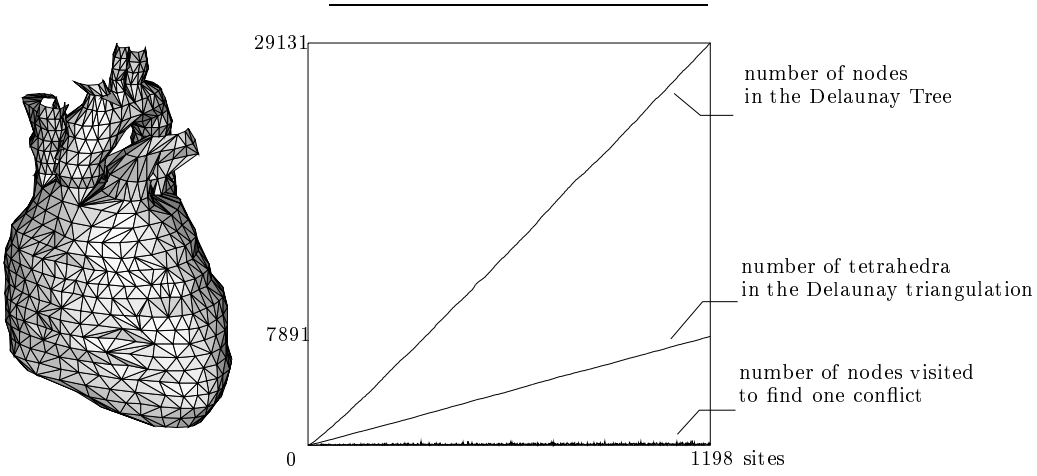


Figure 4.15 : Voronoi diagram : A closed surface (a heart)

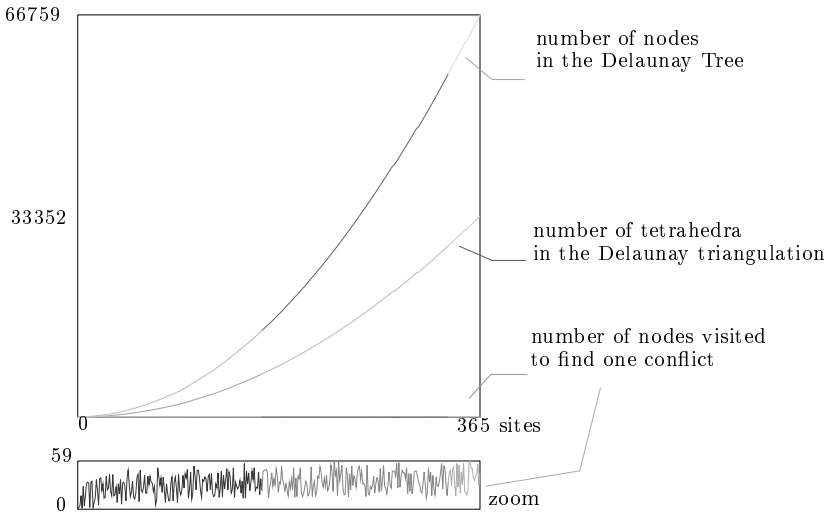


Figure 4.16 : Voronoi diagram : A quadratic example

Inserting a site in the Delaunay Tree

The chosen parameter to evaluate the cost of inserting a site is the number of nodes visited by Procedure location to find the first conflict. According to Remark 3.3, the remainder cost of this procedure consists of an output sensitive search in the Delaunay triangulation, and the cost of Procedure creation also depends on the number of modifications in the Delaunay triangulation.

The cost of locating a site in the Delaunay Tree appears to be very small compared with the size of the Delaunay triangulation.

Some empirical investigations have been done to evaluate the constants. In the plane, the variations of the number of nodes visited to find the first conflict can be roughly (after smoothing) assimilated to the variations of the function $x \mapsto 3 \log_2 x$, in all examples. In 3-space, in the case where the size of the triangulation is linear, the function is $x \mapsto 7 \log_2 x$, which has been explained in Remark 4.22.

We have given a theoretical analysis for the expected number of nodes visited by Procedure location to find all the simplices in conflict with a new site m, but no such analysis has been done for the number of nodes visited to find the first conflict. We can empirically see that, even in the quadratic case that we have tested, this number remains logarithmic. Moreover, the constant appears to be

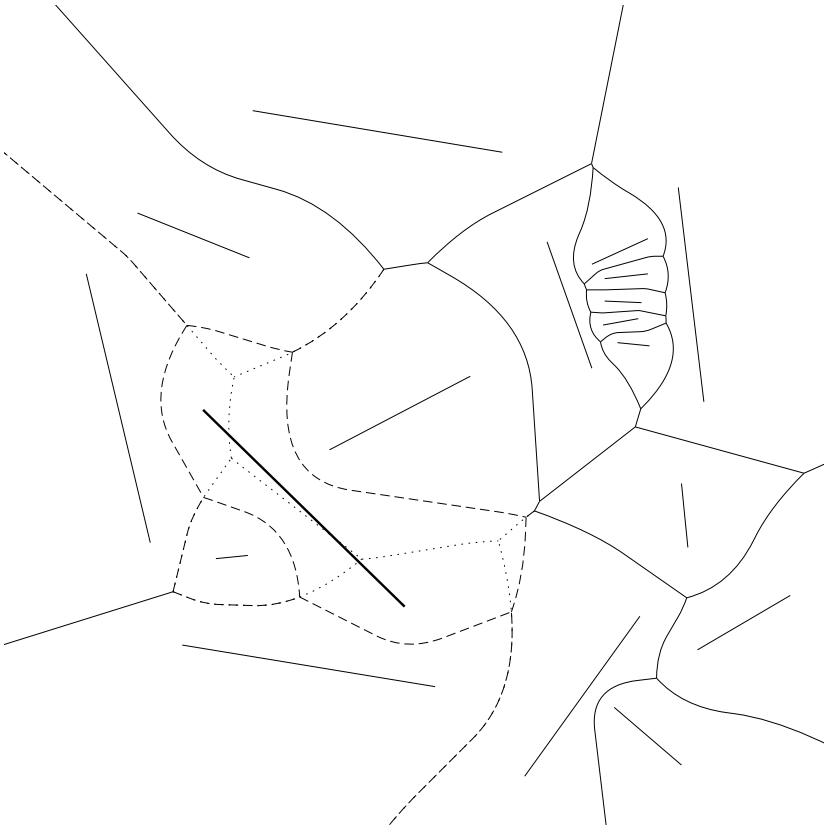
better than in the linear case, since it is about 4.

4.3.3.2 Voronoi diagrams of line segments in the plane

Let us consider now the case of Voronoi diagrams of line segments in two dimensions (see Section 1.2.3). Here $\mathcal{O}$ is the set of all line segments of the euclidean plane. Let p, q, r and s be four segments and let Γ be the portion of the bisector of p and q extending between the two points equidistant from p, q, r and p, q, s respectively (see Section 1.2.3). The region, noted $(\mathtt{pq}, \mathtt{r}, \mathtt{s})$, associated with p, q, r and s is the union of the interiors of the disks tangent to p and q whose centers lie on Γ. Notice that, with this definition, a region is what was previously called a bicycle for the analysis of the Delaunay Tree. A line segment and a region are in conflict if and only if they intersect. A region has zero width if and only if Γ is an edge of the Voronoi diagram.

The update algorithm is as follows. We find in the I-DAG all regions in conflict with the new segment m : these regions correspond to the edges which disappear in the new diagram. The set of disappearing edges form a tree. In fact, an edge (or a part of an edge) E disappears if the points lying on it are closer to m than to the segments determining E. Then, after the insertion of m, these points will belong to the Voronoi cell $V(\mathtt{m})$ of m. Assume now that the set of disappearing edges contains a cycle. Necessarily, this cycle must surround at least one segment s. In the new diagram, the cell $V(\mathtt{s})$ will form a hole in $V(\mathtt{m})$, which is impossible because all cells of the diagram are simply connected.

Let E be a disappearing edge. If one of the two end-points of E is still a valid vertex of the Voronoi diagram (that is if the segment m does not intersect the corresponding disk) we compute in constant time the portion of E that remains in the new diagram. The new region associated with that new edge becomes a son of the region associated with E. We then connect the new vertices of the Voronoi diagram (which are new end-points lying on old edges) by edges supported by new bisectors (see Figure 4.17). The region corresponding to a new edge E' is made son of the regions associated with the unique path of disappearing edges that joins the two end points of E'. This ensures that this region is contained in its parents : an edge E' is the bisector of m and some $\mathtt{s} \in \mathcal{S}$. E' lies entirely in the Voronoi cell $V(\mathtt{s})$ of the diagram before the insertion of m. Let p be a point belonging to the region associated to E'. p lies in the interior of a disk tangent to both m and s, which has no conflict. In the previous diagram, before the insertion of m, this disk can grow larger, without having any conflict, until it touches one segment of $\mathcal{S}$. Then its center necessarily lies on some edge on the boundary of the previous cell $V(\mathtt{s})$. So p lies in the region associated to one father of the region associated to E'.



The new segment `m` is in bold line.
The dotted edges correspond to regions in conflict with `m`.
The dashed edges correspond to new regions created by `m`.

Figure 4.17 : Insertion of `m` in the Voronoi diagram

The update conditions are satisfied and, by the Euler relation, f_0 is linearly related to the number of segments.

Remark 4.23 We can remark that, if the analogous definition had been given for regions in the case of point sites only, we would have obtained another possible definition for the Delaunay Tree, that would have satisfied Update condition 1.

Proposition 4.24 *The Voronoi diagram of n line segments in the plane can be computed with $O(n)$ expected space and $O(\log n)$ expected update time.*

Notice that the algorithm has been coded.

4.4 About complexity results

4.4.1 Randomization

Our analysis of the space and time required to build the I-DAG structure is randomized. As previously noted, randomization concerns here only the order in which the inserted objects are introduced in the structure. No assumption is made as to the distribution of the input. As already said, our results are expected values that correspond to averaging over the $n!$ possible permutations of the n inserted objects, each supposed to be equally likely to occur.

4.4.2 Amortization

We have been able to bound the cost of inserting the k^{th} object in the I-DAG. This cost is not amortized as opposed to the results in [BDT, GKS90] but the k^{th} object may be any one of the inserted objects with the same probability.

It must be noted however that the bound given in Theorem 4.3 cannot be a bound for the cost of inserting a given object. Indeed let us consider the construction of the Delaunay triangulation (the dual of the Voronoi diagram) of a set of n points in the plane. We take $\frac{n-1}{2}$ points close to one line segment, $\frac{n-1}{2}$ points close to another line segment, and one point, say $\circ$, between the two segments (see Figure 4.18). For appropriate positions of the points, the insertion of $\circ$ will modify most of the triangles; thus the expected cost of inserting $\circ$ in the I-DAG at step k is $\Omega(k)$ whatever k is.

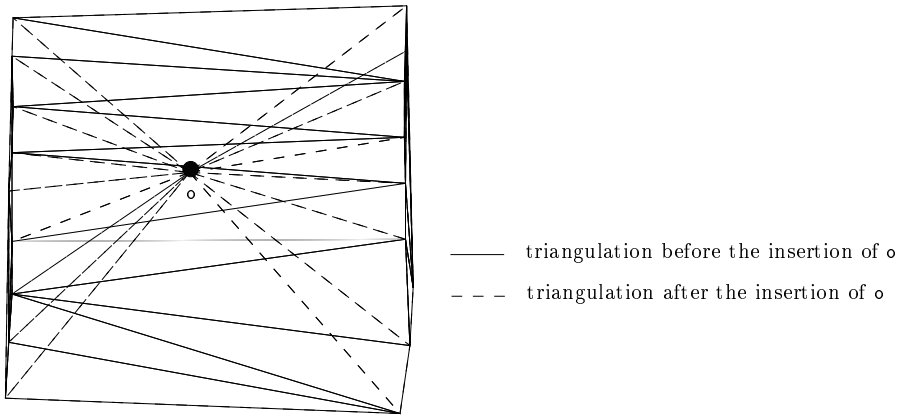


Figure 4.18 : Cost of inserting point o

This is not in contradiction with our result. Indeed, the cost of inserting a given object appears weighted by the probability factor $1/n$ in the expected cost of step k. Our bound on the cost of the k^{th} insertion proves that objects requiring expensive updates are rare whatever the set $\mathcal{S}$ of input data may be.

4.4.3 Output sensitivity

An algorithm is said to be output sensitive if, for a given set of input data, its complexity depends on the actual size of the output. It is clearly impossible, in general, to have incremental algorithms that are sensitive to the final output because at some stage of the incremental construction the intermediate results may be greater than the final one. We may illustrate this with the example of the Voronoi diagram in three dimensions. Let $\mathcal{S}$ be a set of n points lying on two non coplanar line segments. The Voronoi diagram of $\mathcal{S}$ is quadratic but, if a point o between the two segments is added to $\mathcal{S}$, the diagram of $\mathcal{S} \cup \{o\}$ becomes linear.

In view of this fact, it is interesting to define *on-line output sensitive* algorithms as algorithms whose update complexities depend on the actual size of the current output.

Our algorithms are not on-line output sensitive because the expected complexity of each step depends on $f_0(r, \mathcal{S})$ (or $g_0(r, \mathcal{S})$) for some $r \leq n$. Let us

consider again the case of the Voronoi diagram of the set of points above. Inserting a $n + 2^{\text{nd}}$ point to $\mathcal{S} \cup \{\mathtt{o}\}$ will take $O(n^2)$ expected time using the I-DAG although the current output is $O(n)$.

However, in many situations, the expected value $f_0(r, \mathcal{S})$ is a well behaved function of the size r of the random sample which is sensitive to the actual size of the output for $\mathcal{S}$. In such a case, the expected complexity of the I-DAG is on-line output sensitive.

A first illustration is the case of an arrangement of planar curves which has been described in Section 4.3.2. As a second illustration, let us consider the case of the Voronoi diagram in higher dimensions. For some distributions of the input data, the diagrams built on the entire set of points as well as on most of the samples have a linear size. For example, the expected size of the Voronoi diagram of a set $\mathcal{S}$ of n points evenly distributed in the unit d-ball is $O(n)$ and the expected size of the Voronoi diagram for a r-random sample $f_0(r, \mathcal{S})$ is $O(r)$ [Dwy91]. This result readily implies that the Voronoi diagram of n points evenly distributed in the unit d-ball can be computed on-line with $O(n)$ space and $O(\log n)$ update time in any dimension.

Conclusion

We have presented in this chapter a general framework for the design and analysis of on-line algorithms. This framework has been applied successfully to various problems : convex hulls and Voronoi diagrams in any dimension, Voronoi diagrams of segments in the plane, arrangements of curves in the plane. The algorithms are randomized, simple and in some cases output sensitive. They have been coded easily and preliminary experiments have provided strong evidence that they are very efficient in practice. The I-DAG can be used to solve several other problems and provides simple on-line algorithms with the same worst-case complexities as the best (in general static) deterministic algorithms. We simply mention some of them :

- Computing the upper envelope of triangular surface patches in the three dimensional space, with a complexity of $O\left(n^2\alpha(n)\log n\right)$ in the worst case where the size of the output is $O\left(n^2\alpha(n)\right)$ [BD91].
- Computing abstract Voronoi diagrams ([MMO91] propose a static randomized algorithm).
- Computing arrangements of surface patches in space (see [CEG*90] for combinatorial bounds).
- Computing the intersection of n half spaces : this problem is dual to constructing convex hulls.
- Computing the union of n balls in d space : consider the d-dimensional space as an hyperplane of a $d+1$-space and use an inversion with a point outside the hyperplane as its pole : the problem reduces to that of computing the intersection of n half spaces.
- Computing the visibility graph of a set of line segments in the plane (see [Wel85]) : take as regions the triangles containing two edges of the visibility graph incident to a common vertex and consecutive when sorted by polar angle around this vertex. A line segment is in conflict with a region if it intersects the region.

Our technique assumes that the geometric structure to be computed is closely related to the regions of zero width. One may be interested in computing instead regions of width $\leq k$ to construct, for example, k-sets or Voronoi diagrams of order k. It is possible to generalize the I-DAG and to obtain results similar to the ones described here. The complexity results will depend on the expected number of regions of width $\leq k$ defined by random samples, as will be seen in Chapter 5.

We will study the possibility of allowing deletions as well as insertions, thus making the structure fully dynamic, in Chapter 6.

Chapitre V

Le k-arbre de Delaunay

The k-Delaunay Tree is an extension of the Delaunay Tree, and is used for the construction of higher order Voronoi diagrams defined in Section 1.2.2.

K.L. Clarkson's randomized algorithm [Cla87] determines the order k Voronoi diagram of n sites in the plane in expected time $O(kn^{1+\varepsilon})$ with a constant factor that depends on ε.
K. Mulmuley [Mul91a] achieves a complexity of $O\left(k^{\lceil \frac{d+1}{2} \rceil} n^{\lfloor \frac{d+1}{2} \rfloor}\right)$ for $d \geq 3$ ($O(nk^2 + n \log n)$ if $d = 2$), for the randomized construction of the Voronoi diagram of order 1 to k (see Section 2.3.2). These algorithms are static.

We present here an algorithm that is semi-dynamic. After each insertion of a new site, the algorithm updates a data structure, called the k-Delaunay Tree [BDT]. This structure generalizes the Delaunay Tree to compute the Delaunay triangulation (and, by duality, the Voronoi diagram) of a set of points. The k-Delaunay Tree contains all the successive versions of the order $\leq k$ Voronoi diagrams and allows fast point location.

As in the preceding chapter, we show that randomization allows to obtain an efficient complexity on the average, which is, as usual, impossible in the worst case for a semi-dynamic algorithm. Our main result states that if we randomize the insertion sequence of the n sites, the k-Delaunay Tree (and thus the order $\leq k$ Voronoi diagrams) can be constructed in expected time $O(n \log n + k^3 n)$ in two dimensions and expected storage $O(k^2 n)$.

Our algorithm extends to higher dimensions. For a given value d of the dimension, its expected time complexity is $O\left(k^{\lceil \frac{d+1}{2} \rceil + 1} n^{\lfloor \frac{d+1}{2} \rfloor}\right)$ and its expected space complexity is $O\left(k^{\lceil \frac{d+1}{2} \rceil} n^{\lfloor \frac{d+1}{2} \rfloor}\right)$.

Very recently, F. Aurenhammer and O. Schwarzkopf [AS91] proposed a dynamic algorithm for the construction of the order k diagram in the plane whose randomized complexity is $O\left(k^2 n + kn \log^2 n\right)$ (see Chapter 7).

The overall organization of the chapter is the following. In Section 5.1, we define the k-Delaunay Tree in two dimensions and present an algorithm for its construction. In Section 5.2, we analyse the complexity of the randomized construction of the k-Delaunay Tree and thus, of all order $\leq k$ Voronoi diagrams. Section 5.3 shows how to use the k-Delaunay Tree for searching the l nearest neighbors of a given point. In Section 5.4, we extend our results to d dimensions. Last but not least, Section 5.5 presents experimental results which provide evidence that the algorithm is very effective in practice for small values of k.

5.1 The k-Delaunay Tree in two dimensions

$\mathcal{S}$ is a non degenerated set of n sites in the euclidean plane. We use the terms defined in Section 1.2.2.

We first show a lemma which will be useful in the sequel.

Lemma 5.1 *Let $\mathcal{T} \subset \mathcal{S}$. The Voronoi polygon $V(\mathcal{T})$ does not change if we add to $\mathcal{T}$ and $\mathcal{S}$ some new points lying in the convex hull of $\mathcal{T}$. More precisely $V(\mathcal{T})$ in $Vor_{|\mathcal{T}|}(\mathcal{S})$ is equal to $V(\mathcal{T} \cup \mathcal{R})$ in $Vor_{|\mathcal{T}\cup\mathcal{R}|}(\mathcal{S} \cup \mathcal{R})$ if the points of $\mathcal{R}$ lie in the interior of the convex hull of $\mathcal{T}$.*

Proof. Let $\mathcal{R}$ be a set of points not in $\mathcal{S}$, and lying in the interior of the convex hull of $\mathcal{T} \subset \mathcal{S}$. We denote $V_{\mathcal{S}}(\mathcal{T})$ the Voronoi polygon of $\mathcal{T}$, where $\mathcal{T}$ is considered as a subset of $\mathcal{S}$ (i.e. in $Vor_{|\mathcal{T}|}(\mathcal{S})$). We first prove that $V_{\mathcal{S}}(\mathcal{T}) \subset V_{\mathcal{S}\cup\mathcal{R}}(\mathcal{T} \cup \mathcal{R})$.

Let $\mathtt{m} \in V_{\mathcal{S}}(\mathcal{T})$, $\mathtt{p} \in \mathcal{T} \cup \mathcal{R}$, and $\mathtt{q} \in (\mathcal{S} \cup \mathcal{R}) \setminus (\mathcal{T} \cup \mathcal{R}) = \mathcal{S} \setminus \mathcal{T}$.

- if $\mathtt{p} \in \mathcal{T}$, then $\delta(\mathtt{m}, \mathtt{p}) < \delta(\mathtt{m}, \mathtt{q})$, since $\mathtt{m} \in V_{\mathcal{S}}(\mathcal{T})$.
- if $\mathtt{p} \in \mathcal{R}$, then $\mathtt{p} = \sum_i \alpha_i \mathtt{t}_i$, where $\mathtt{t}_i \in \mathcal{T}$ and $\alpha_i \in I\!R^+, \sum_i \alpha_i = 1$.

$$\begin{aligned} \delta(\mathtt{m}, \mathtt{p}) &\leq \sum_i \alpha_i \delta(\mathtt{m}, \mathtt{t}_i) \\ &\quad \text{since } \delta \text{ is associated to the euclidean norm,} \\ &\quad \text{and thus } \mathtt{x} \mapsto \delta(\mathtt{m}, \mathtt{x}) \text{ is a convex function} \\ &< \sum_i \alpha_i \delta(\mathtt{m}, \mathtt{q}) \text{ since } \mathtt{t}_i \in \mathcal{T} \\ &= \delta(\mathtt{m}, \mathtt{q}) \end{aligned}$$

In both cases, $\delta(\mathtt{m}, \mathtt{p}) < \delta(\mathtt{m}, \mathtt{q})$, from which we deduce that : $\mathtt{m} \in V_{\mathcal{S}\cup\mathcal{R}}(\mathcal{T} \cup \mathcal{R})$.

The reciprocal inclusion is straightforward, which achieves the proof. □

5.1.1 Including and excluding neighbors

In the sequel, we will denote $B(T)$ the interior of the disk circumscribing triangle T.

For a triangle T, we will define 2 neighbors through each of its 3 edges : one will be called the *including* neighbor and the other one the *excluding* neighbor.

This notion of neighborhood corresponds to the actual notion of adjacency in the higher order Voronoi diagrams.

Let E be an edge of T and $\mathtt{p}$ be the opposite vertex of T. Let us consider a moving disk B whose boundary passes through the end points of E, and whose center moves along the bisecting line of E. Starting from $B = B(T)$, we can move B in two opposite directions : the one such that $\mathtt{p} \in B$ is called the including direction and the other such that $\mathtt{p} \notin B$ is called the excluding direction. We stop moving B as soon as its boundary encounters a site different from the end points of E. Let $\mathtt{q}_i$ (resp. $\mathtt{q}_e$) be the first site encountered in the including (resp. excluding) direction. The triangle T_i (resp. T_e) having E as an edge and $\mathtt{q}_i$ (resp. $\mathtt{q}_e$) as a vertex will be called the *including neighbor* (resp. *excluding neighbor*) of T through edge E (see Figure 5.1). Notice that $\mathtt{q}_i$ and $\mathtt{q}_e$ may be on either side of the line supporting E.

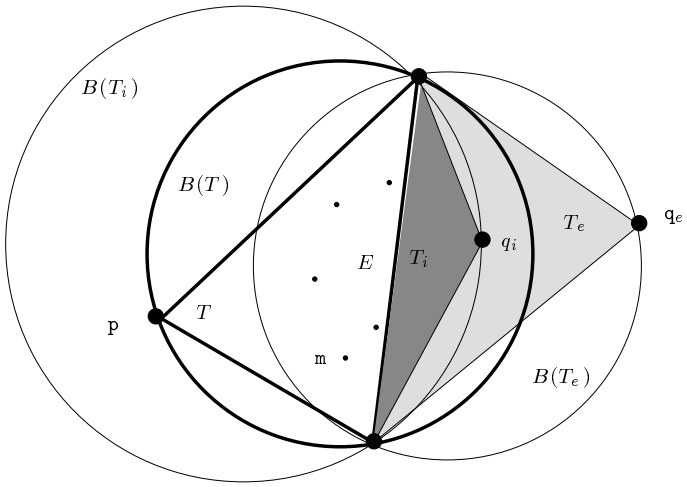


Figure 5.1 : Including and excluding neighbors

Remark 5.2 In the sequel, we will also speak of the neighbors of a triangle T in the direction including a site $\mathtt{m}$ in $B(T)$ and in the direction excluding $\mathtt{m}$. The neighbor of T through E in the direction excluding $\mathtt{m}$ (resp. including $\mathtt{m}$) is the excluding (resp. including) neighbor of T if $\mathtt{m}$ and $\mathtt{p}$ are on the same side of E and the including (resp. excluding) neighbor of T otherwise.

Remark 5.3 If S is the excluding neighbor through E for T, then T is a neighbor of S, but T may be either the including neighbor through E of S, if $\mathtt{q}_e$ lies on the same side as T with respect to the supporting line of E, or the excluding one on the other case.

In the same way, if S is the including neighbor of T, T is the including neighbor of S if $\mathtt{q}_i$ does not lie on the same side as T with respect to E, and the excluding neighbor otherwise.

The neighborhood relationships are reciprocal if and only if S and T do not lie on the same side with respect to the supporting line of E. They are inversed otherwise

Remark 5.4 The following property will be useful in the sequel :

$$B(T) \subset B(T_i) \cup B(T_e)$$

Hence, if a site $\mathtt{m}$ lies inside $B(T)$, we can deduce that $\mathtt{m}$ lies into either $B(T_i)$ or $B(T_e)$.

If $B(T)$ contains k sites, $B(T_e)$ contains k sites if $\mathtt{q}_e$ lies outside $B(T)$ (the k sites in $B(T)$), or $k-1$ sites if $\mathtt{q}_e$ lies inside $B(T)$ (the k sites in $B(T)$ minus $\mathtt{q}_e$). Speaking in terms of Voronoi vertices, if $B(T)$ contains k sites, T is dual to a close-type vertex τ of Vor_{k+1} and its excluding neighbors are dual to the adjacent vertices of τ in Vor_{k+1}.

Similarly, if $B(T)$ contains k sites, then $B(T_i)$ contains $k+1$ sites if $\mathtt{q}_i$ lies outside $B(T)$ (the k sites in $B(T)$ plus $\mathtt{p}$), or k sites if $\mathtt{q}_i$ lies inside $B(T)$ (the k sites in $B(T)$ plus $\mathtt{p}$ minus $\mathtt{q}_i$). Then T is dual to a far-type vertex τ' of Vor_{k+2}, and its including neighbors are dual to the adjacent vertices of τ' in Vor_{k+2}.

5.1.2 A semi-dynamic algorithm for constructing the order $\leq k$ Voronoi diagrams

Our algorithm is a generalization of the semi-dynamic algorithm presented in Chapter 3 for constructing the Delaunay triangulation. Each site is introduced, one after another, in each of the order $\leq k$ Voronoi diagrams and each diagram is subsequently updated. We will describe this algorithm at the same time as the k-Delaunay Tree, in the following sections, but it is actually independent of that data structure.

5.1.3 Construction of the k-Delaunay Tree

The k-Delaunay Tree is a hierarchical structure that we use to construct the order $\leq k$ Voronoi diagrams. As in the Delaunay Tree, the nodes are associated with triangles. We will often use the same word *triangle* for both a triangle and the node associated with it. The k-Delaunay Tree is not really a tree but a rooted direct acyclic graph.

Recall that the width of a triangle is the number of sites in conflict with it or, in other words, the number of sites lying in the interior of its circumscribing disk.

Following our algorithm, each site is introduced in turn and we keep all triangles in a hierarchical manner in the k-Delaunay Tree, creating appropriate links between "old" (i.e. created before the introduction of the site) and "new" triangles (i.e. created after the introduction of the site). This will allow to efficiently locate a new site in the current structure.

5.1.3.1 Initialization

For the initialization step we choose 3 sites. They define one finite triangle and six half planes (considered as infinite triangles) limited by the supporting lines of the finite triangle. Three of the half planes contain one site, the other ones are empty. The 4 triangles of current width zero will be the sons of the root of the tree. Their neighborhood relationships in the Delaunay triangulation are created. The 3 triangles of current width 1 are linked to the preceding ones by their neighborhood relationships in the order 2 Voronoi diagram.

5.1.3.2 Inserting a new site

The k-Delaunay Tree will be constructed so that it satisfies the following property :

> ($\mathcal{P}$) *all the triangles of current width strictly less than k are present in the k-Delaunay Tree.*

Hence, the triangles dual to the vertices of the order $\leq k$ Voronoi diagrams are all present in the structure. Moreover, we keep their adjacency relationships in the corresponding Voronoi diagrams. The k-Delaunay Tree thus contains the whole information necessary to construct all the order $\leq k$ Voronoi diagrams.

Let us suppose that the k-Delaunay Tree has been constructed for the already inserted sites and satisfies the above property $(\mathcal{P})$. Let $\mathtt{m}$ be the next site to be inserted. We will see how to update the k-Delaunay Tree so that it still satisfies $(\mathcal{P})$ after the insertion of $\mathtt{m}$.

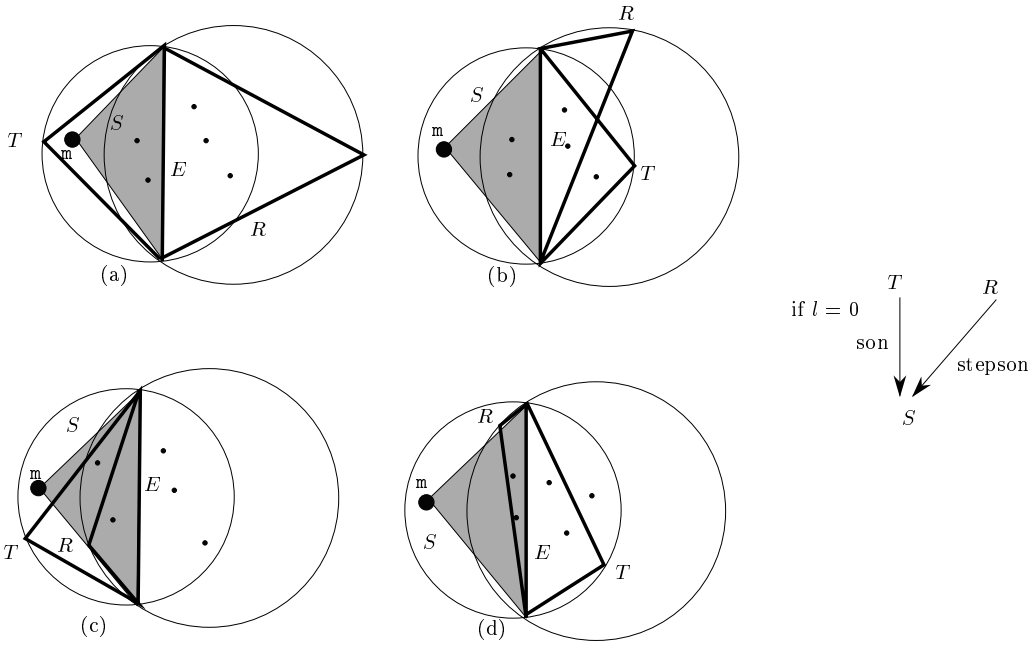
Let S be a triangle dual to a vertex of some of the order $\leq k$ Voronoi diagrams, to be created after the insertion of $\mathtt{m}$ in order to satisfy $(\mathcal{P})$. S is called a *new triangle*. S has $\mathtt{m}$ as a vertex ; let E be the edge of S not containing $\mathtt{m}$. S has an including neighbor T through E, and an excluding neighbor R through E. It is plain to observe that R and T were necessarily neighbors before the insertion of $\mathtt{m}$ and that $B(T)$ contains $\mathtt{m}$ while $B(R)$ does not. We have obtained a necessary condition for the creation of a triangle. Figure 5.2 shows the four typical situations.

Conversely, we show that this condition is sufficient. Let T be a triangle dual to a vertex of some of the order $\leq k$ Voronoi diagrams, whose disk $B(T)$ contains $\mathtt{m}$. Let E be an edge of T, and $\mathtt{p}$ be the third vertex of T. If the neighbor R of T through E in the direction excluding $\mathtt{m}$ does not contain $\mathtt{m}$ in its disk, then T and R cannot stay neighbors any longer. We must create a new triangle S by linking $\mathtt{m}$ to E. T becomes the including neighbor of S through E and R its excluding one. (Notice that, in Case (d) of Figure 5.2, if $l = k$, R does not need any including neighbor, which would belong to the Voronoi diagram of order $k+1$, so S is not created in this case.)

If now the current width of S is l , the current width of T, before the insertion of $\mathtt{m}$, was l or $l-1$ and the current width of R is still l or $l-1$, as before the insertion of $\mathtt{m}$ (see Figure 5.2). Thus property $(\mathcal{P})$ implies that T and R were associated to nodes in the k-Delaunay Tree before the insertion of $\mathtt{m}$, the current width of T is increased by one and is now $l+1$ or l. If this number is larger or equal to k, then T is marked as *dead* (T is no longer dual to a vertex of some order $\leq k$ Voronoi diagram). (Remember that, if $l = k$, which can happen only in Case (d), S is not created.)

Furthermore, if the current width of S is zero when it is created (which can only occur in Case (a) of Figure 5.2), we then update the tree by making S a *son* of T and a *stepson* of R.

> *Remark 5.5* We can thus notice that the subgraph of the k-Delaunay Tree obtained by recursively traversing all links to sons and stepsons, starting from the root, is the Delaunay Tree presented in Chapter 3 (which is also the 1-Delaunay Tree) : a triangle can be created in the (1-)Delaunay Tree only if its current width is zero ; at this moment, it is the neighbor of its stepfather, and the current width of its father is incremented, so it becomes at once dead. There are no



If l is the current width of the new triangle S, the current width of T before the insertion of $\mathtt{m}$, the current width of T after the insertion and the current width of R are respectively :

- l, $l+1$ and l in the upper-left case (a)
- $l-1$, l and l in the upper-right case (b)
- l, $l+1$ and $l-1$ in the bottom-left case (c)
- $l-1$, l and $l-1$ in the bottom-right case (d).

Figure 5.2 : Inserting a new site

additional neighborhood relationships involving triangles of current width larger than zero, because the (1-)Delaunay Tree only deals with neighborhood relationships between Delaunay triangles.

We also update the neighborhood relationships between the triangles which are not marked as dead.

In order to ensure that $(\mathcal{P})$ still holds, it is sufficient to apply the above procedure to *all* the triangles T whose circumscribing disks contain `m`. Thus we need to find all those triangles. This will be performed by Procedure location while the creation of the new triangles and the maintenance of the neighborhood relationships will be performed by Procedure creation. These two procedures are detailed below.

When a node receives sons, its width increases, and it can only get new sons when its width is zero, so the total number of sons of a node is at most 3. Notice however that the number of stepsons is unbounded ; unfortunately, all stepsons are useful as shown in Chapter 3.

Nevertheless, we have the following property :

Lemma 5.6 *The total number of links to sons and stepsons in the k-Delaunay Tree is less than twice the number of nodes.*

Proof. Each newly created node (the 7 first ones excepted) has exactly one father and one stepfather. □

5.1.3.3 Structure of the k-Delaunay Tree

As previously noted, the k-Delaunay Tree is a directed acyclic graph. Each node of the k-Delaunay Tree is associated to a triangle. The node associated to triangle T contains the following informations :

- three links to the three sites which are vertices of T
- the center and the radius (more exactly the squared radius) of $B(T)$, which can also be used to mark T as finite or infinite
- its at most 3 sons
- the list of its stepsons
- the last site located in T

- the last site propagated in T (these last two fields will be used by Procedure location)
- the current width of T, which can also be used to mark T as dead
- the three excluding neighbors of T, and its three including neighbors if its width is at most $k-2$

The first six fields are the same as in the (1-)Delaunay Tree (see Remark 5.5). In the (1-)Delaunay Tree, as soon as the current width of a triangle T is equal to 1, it becomes dead, so there is only a mark remembering whether T is dead or not ; another difference is that there are only three neighbors (they are excluding neighbors), that are the neighbors of T in the Delaunay triangulation, if T is not dead.

> *Remark 5.7* As already noticed, the two types of neighbors correspond to neighbors in different Voronoi diagrams, and allow, starting from one vertex of the order 1 Voronoi diagram, to reach all adjacent vertices in each of the higher order Voronoi diagrams. This point is crucial for Procedure propagate, called by procedure location, as will be seen in the next section.

5.1.3.4 Procedure location

If `m` belongs to the circumscribing disk of a triangle, we know (see Remark 5.4) that it belongs to the union of the circumscribing disks of its father and of its stepfather. So the traversal of the Delaunay Tree (Remark 5.5) gives all triangles whose disks contain `m`.

In fact, we traverse it until we find only one Delaunay triangle in conflict with `m`. Then we follow the neighborhood relationships to find all triangles, of current width strictly less than k, in conflict with `m` (see Remark 5.7). We store them in a list $T(\mathtt{m})$, the edges of these triangles are the possible candidates to be used for the creation of new triangles.

Procedure location and Procedure propagate are described in Figure 5.3. Observe that Procedure location is the same as in Chapter 3, because it consists of a traversal of the Delaunay Tree.

5.1.3.5 Procedure creation

We go through the list $T(\mathtt{m})$. Let T be a triangle of this list and E one of its edges. If the neighbor R of T through E in the direction excluding `m` does not contain `m`,

---

Initialize the list $T(\mathtt{m})$ as the empty list
location(m,root of the k-Delaunay Tree)

Procedure location(m,node)
    ($\star$ The node is associated to triangle T $\star$)
        if the last site located in T is not m and
        if m lies into $B(T)$, **then**
            m becomes the last site located in T;
            for each son, location(m,son);
            for each stepson, location(m,stepson);
            if the current width of T is 0, **then**
                propagate(m,T);
                stop Procedure location.

Procedure propagate(m,T)
    m becomes the last site propagated in T;
    add T to the list $T(\mathtt{m})$;
    increment the current width of T;
    if the current width of T is now k, **then** mark T as dead;
    for each neighbor N of T
        if the last site propagated in N is not m and
        if N is not dead, **then**
            if m lies into $B(N)$, **then** propagate(m,N).

Figure 5.3 : Locating a site in the k-Delaunay Tree

---

we then create S by linking $\mathtt{m}$ to E, and the son and stepson relations involving S, if the current width of S is 0. Procedure creation is described in Figure 5.4.

Procedure creation$(T(\mathtt{m}))$

for each triangle T of $T(\mathtt{m})$
  for each edge E of T
    if the neighbor R of T through E in the direction
        excluding $\mathtt{m}$ does not contain $\mathtt{m}$ in its disk, **then**
      - create the triangle S having vertex $\mathtt{m}$ and edge E,
        and its associated node
        (except in the case where the width of S is k);
      - declare R as the excluding neighbor of S through E
        and update the reciprocal relation;
      - declare T as the including neighbor of S through E
        and update the reciprocal relation;
      - **if** the current width of S is 0, **then**
        create the relations : S son of T and S stepson of R;
create the neighborhood relationships between the new triangles.

Figure 5.4 : Creating the new triangles

Let us now see how we can maintain the neighborhood relationships between triangles.

As already mentioned, when S is created, R is the excluding neighbor of S through edge E common to T and R, and T is the including one. We can easily compute the reciprocal relations : R and T were neighbors before the insertion of $\mathtt{m}$ (notice that R may be either an excluding or an including neighbor of T ; remember also that the relations between neighbors are not symmetric, see Remark 5.3). If T was the excluding (resp. including) neighbor of R through E, then now S is the excluding (resp. including) neighbor of R. Similarly, if R was the excluding (resp. including) neighbor of T through E, then now S is the excluding (resp. including) neighbor of T.

In order to create the neighborhood relationships between the new triangles, we proceed as follows. Let us suppose that the insertion of $\mathtt{m}$ creates $x(\mathtt{m})$ new triangles $T_1, T_2, \ldots, T_{x(\mathtt{m})}$, where $T_i = (\mathtt{m}, \mathtt{s}_i^0, \mathtt{s}_i^1), i = 1, 2, \ldots, x(\mathtt{m})$. The T_i's are dual to vertices of various $Vor_l(\mathcal{S}), l \leq k$. The adjacency relationships must be

created in each $Vor_l(\mathcal{S}), l \leq k$. Our goal is to find, for each T_i $(i = 1, \ldots, x(\mathtt{m}))$, its excluding and its including neighbors through both edges $\mathtt{ms}_i^0$ and $\mathtt{ms}_i^1$. To this aim, for each $\mathtt{s}_i^j, i = 1, 2, \ldots, x(\mathtt{m}), j = 0, 1$, we sort the list of new triangles having $\mathtt{s}_i^j$ as a vertex, according to the abscissa of their center on the bisecting line of $[\mathtt{m}, \mathtt{s}_i^j]$. By definition, $(\mathtt{m}, \mathtt{s}_i^j, \mathtt{v})$ and $(\mathtt{m}, \mathtt{s}_i^j, \mathtt{v}')$ are neighbors through $[\mathtt{m}, \mathtt{s}_i^j]$ *iff* they are successive in this order. We thus only have to go through the list of sorted triangles to obtain the neighborhood relationships through the corresponding edge.

For each edge $(\mathtt{m}, \mathtt{s}_i^j)$, the complexity of this sorting is $O(n_i^j \log n_i^j)$, where n_i^j denotes the number of new triangles having $(\mathtt{m}, \mathtt{s}_i^j)$ as an edge, which is bounded by $2k$. The whole complexity of creating the neighborhood relationships is thus $O(x(\mathtt{m}) \log k)$ for each new site $\mathtt{m}$.

Remark 5.8 The $\log k$ appearing in this complexity could be dropped, but it would complicate our algorithm. As we shall see in the sequel, the cost of Procedure creation is dominated by the cost of Procedure location, therefore we do not elaborate on improving its complexity.

5.2 Analysis of the randomized construction

The above algorithm allows to insert new sites in a dynamic way. As in Chapter 4, for the analysis, we assume that all sequences of insertion have the same probability.

This section proves the main result of this paper, stated in the following theorem :

Theorem 5.9 *For any set of n sites, if we randomize the sequence of their insertion, the k-Delaunay Tree (and thus the order $\leq k$ Voronoi diagrams) of the n sites can be constructed in expected time $O(n \log n + k^3 n)$ in two dimensions, using expected storage $O(k^2 n)$.*

The remaining of Section 5.2 is devoted to the proof of Theorem 5.9. Section 5.2.2 analyzes the expected space used to store the k-Delaunay Tree, Section 5.2.3 the expected cost of locating the n sites and Section 5.2.4 the cost of constructing the successive triangles and their neighborhood relationships.

5.2.1 Results on triangles and bicycles

Let $\mathtt{x}, \mathtt{y}, \mathtt{z}, \mathtt{t}$ be four sites. As in Chapter 4, the *bicycle* $\mathtt{x(yz)t}$ is the figure drawn by $B(\mathtt{xyz})$ and $B(\mathtt{yzt})$ (see Figure 5.5 which represents one of the possible configurations of a bicycle).

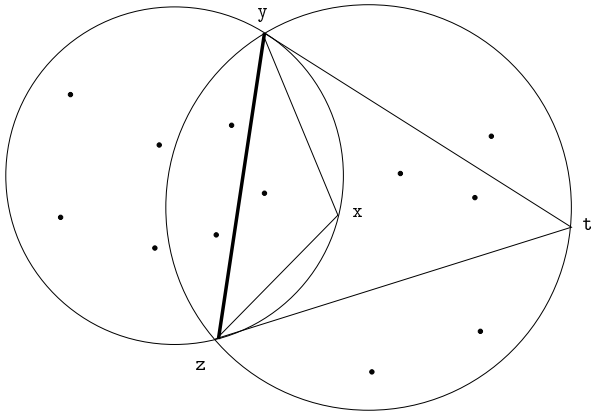


Figure 5.5 : The bicycle $\mathtt{x(yz)t}$

The width of a bicycle $\mathtt{x(yz)t}$ is the number of sites belonging to $B(\mathtt{xyz})$ and $B(\mathtt{yzt})$, where we do not take $\mathtt{x}, \mathtt{y}, \mathtt{z}$ and $\mathtt{t}$ into account. $\mathcal{F}_j(\mathcal{S})$ (resp. $\mathcal{G}_j(\mathcal{S})$) is the set of triangles (resp. bicycles) having width j and $\mathcal{F}_{\leq j}(\mathcal{S})$ (resp. $\mathcal{G}_{\leq j}(\mathcal{S})$) the set of triangles (resp. bicycles) of width at most j.

Lemma 5.10 *Let* $\mathtt{xyz}$ *be a triangle having width* j. $\mathtt{xyz}$ *will arise as a vertex of some order* $\leq k$ *Voronoi diagram during the construction with probability*

$$\begin{cases} \dfrac{k(k+1)(k+2)}{(j+1)(j+2)(j+3)} & \text{if} \quad j \geq k \\ 1 & \text{if} \quad j < k \end{cases}$$

Proof. Let l be the current width of $\mathtt{xyz}$ when $\mathtt{xyz}$ is created. The property holds true if and only if one of the three sites $\mathtt{x}, \mathtt{y}$ and $\mathtt{z}$ is introduced after the l sites inside $B(\mathtt{xyz})$ (in any order, hence $3(l+2)!$

possible orders for the $l+3$ first sites, and $\left(\begin{array}{c} j \\ l \end{array} \right)$ possible sets of l sites among the j ones), and before the $j-l$ remaining sites (which may also be introduced in any order, that is $(j-l)!$ possibilities). There are $(j+3)!$ permutations on the $j+3$ sites. Thus `xyz` of width j appears with current width l with probability :

$$\frac{3\left(\begin{array}{c} j \\ l \end{array} \right)(l+2)!\,(j-l)!}{(j+3)!} = \frac{3(l+1)(l+2)}{(j+1)(j+2)(j+3)}$$

The required probability is the sum of the last ones, for all l :

- if $j \geq k$, l can only be $< k$, so we get

$$\sum_{l=0}^{k-1} \frac{3(l+1)(l+2)}{(j+1)(j+2)(j+3)} = \frac{k(k+1)(k+2)}{(j+1)(j+2)(j+3)}$$

- if $j < k$

$$\begin{aligned} \sum_{l \leq j} \frac{3(l+1)(l+2)}{(j+1)(j+2)(j+3)} &= \frac{(j+1)(j+2)(j+3)}{(j+1)(j+2)(j+3)} \\ &= 1 \end{aligned}$$

which agrees with Property $(\mathcal{P})$: a triangle with width $< k$ will necessarily appear at some stage of the construction. □

Lemma 5.11 *Let* `x(yz)t` *be a bicycle of width* j *such that* `yzt` *is a son or a stepson of* `xyz`. *The probability that such a bicycle appears during the construction is*

$$\frac{3!}{(j+1)(j+2)(j+3)(j+4)}$$

Proof. A bicycle `x(yz)t` is always created with current width 0. Now, to get `yzt` as a son or a stepson of `xyz`, we need the following condition : `t` must be inserted after `x`, `y` and `z` ; thus there are 3! possibilities to insert `x`, `y`, `z` and `t`. Then the j sites inside the bicycle can be inserted in any of the $j!$ possible orders. So the probability is :

$$\frac{3!j!}{(j+4)!} = \frac{3!}{(j+1)(j+2)(j+3)(j+4)}$$

This result also agrees with property $(\mathcal{P})$: a bicycle with width 0 will always appear, and the last site inserted among `x`, `y`, `z` and `t` is `t` with probability $\frac{1}{4}$. □

The following lemma is a direct consequence of the bound on the size of higher order Voronoi diagrams. It gives in this case a bound for $|\mathcal{F}_j(\mathcal{S})|$, more precise that the general bound given by Theorem 2.3 for $|\mathcal{F}_{\leq j}(\mathcal{S})|$.

Lemma 5.12 *The number of triangles having width j is*

$$|\mathcal{F}_j(\mathcal{S})| \leq (2j+1)n$$

Proof. $|\mathcal{F}_j(\mathcal{S})|$ is exactly the number of close-type vertices of the order $j+1$ Voronoi diagram. This number is computed in [Lee82], which gives the result. □

We propose here an alternative proof for Lemma 4.10, owing to the preceding lemma.

Lemma 5.13 *The number of bicycles having width at most j is*

$$|\mathcal{G}_{\leq j}(\mathcal{S})| = O(n(j+1)^3)$$

Proof. We first notice that a given segment joining two points of $\mathcal{S}$ is an edge of at most $2j+2$ triangles of width less than j : in Figure 5.6, every triangle defined by E as an edge, and a site lying outside the two drawn circles will be of width superior to j, and the sites lying on or inside these circles allow us to form at most $2j+2$ triangles of width at most j.

We then bound the number of bicycles of width $\leq j$ by subdividing a bicycle into two adjacent triangles, one of width l and the other of width less than j.

$$\begin{aligned}
|\mathcal{G}_{\leq j}(\mathcal{S})| &\leq \sum_{l=0}^{j} |\mathcal{F}_l(\mathcal{S})| 3(2j+2) \\
&\leq 6(j+1) \sum_{l=0}^{j} O(n(l+1)) \\
&\leq 6(j+1) O(n(j+1)^2) \\
&= O(n(j+1)^3)
\end{aligned}$$

□

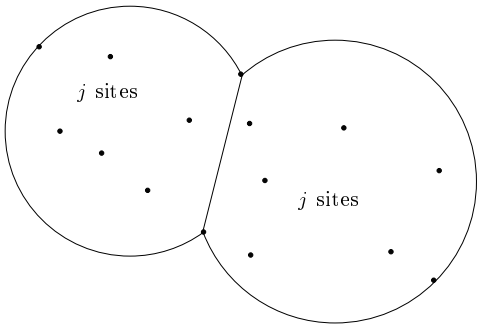


Figure 5.6 : For the proof of Lemma 5.13

5.2.2 Analysis of the expected space used by the k-Delaunay Tree

Lemma 5.14 *The expected number of nodes in the k-Delaunay Tree is $O(k^2 n)$.*

Proof. This number is the number of all successive vertices arising in all order $\leq k$ Voronoi diagrams during the construction. It is less than

$$\sum_{j=0}^{n-3} \sum_{S \in \mathcal{F}_j(\mathcal{S})} \text{Prob}(S \text{ arises during the construction})$$

$$= \sum_{j=0}^{k-1} \sum_{S \in \mathcal{F}_j(\mathcal{S})} 1 + \sum_{j=k}^{n-3} \sum_{S \in \mathcal{F}_j(\mathcal{S})} \frac{k(k+1)(k+2)}{(j+1)(j+2)(j+3)}$$

(using Lemma 5.10)

$$= \sum_{j=0}^{k-1} |\mathcal{F}_j(\mathcal{S})| + k(k+1)(k+2) \sum_{j=k}^{n-3} \frac{|\mathcal{F}_j(\mathcal{S})|}{(j+1)(j+2)(j+3)}$$

$$= O(k^2 n) \text{ , using Lemma 5.12 and } \sum_{j=k}^{n-3} \frac{1}{(j+2)(j+3)} = O(\frac{1}{k})$$

□

We already know that the total number of links to sons and stepsons is less than the number of nodes (Lemma 5.6), and that each node has at most 6 neighbors. We conclude :

Proposition 5.15 *The expected space complexity of the k-Delaunay Tree is* $O(k^2 n)$.

5.2.3 Analysis of the expected cost of Procedure location

We want here to count the number of nodes visited during the construction. During the insertion of `m`, Procedure location looks at first for a Delaunay triangle in conflict with `m`. To this end, a triangle `yzt` may be visited if `yzt` is the son or the stepson of a triangle `xyz` such that $\mathtt{m} \in B(\mathtt{xyz})$. So a bicycle `x(yz)t` of width j obliges to visit up to j nodes. Secondly, Procedure propagate is called, and a triangle is visited if it is the neighbor of a triangle `xyz` such that $\mathtt{m} \in B(\mathtt{xyz})$, and `xyz` is not dead. So a triangle of width j obliges to visit up to $6\inf(j, k-1)$ nodes.

Lemma 5.16 *The expected total number of visited nodes during the construction is* $O(n \log n + k^3 n)$.

Proof. The total number of visited nodes during the search for a triangle of the Delaunay triangulation in conflict with a new site is inferior to :

$$\sum_{j=0}^{n-4} \sum_{\mathtt{x(yz)t} \in \mathcal{G}_j(\mathcal{S})} j \operatorname{Prob} \left(\begin{array}{c} \mathtt{yzt} \text{ arises as a son} \\ \text{or a stepson of } \mathtt{xyz} \\ \text{during the construction} \end{array} \right)$$

We then bound this expression by $O(n \log n)$, using Lemmas 5.11 and 5.13, in a way that is similar to the proof of Lemma 4.1. When such a triangle has been found, Procedure propagate then explores all triangles in conflict with the site, using the neighborhood relationships. The total number of triangles visited during the propagation is less than :

$$\sum_{j=0}^{n-3} \sum_{S \in \mathcal{F}_j(\mathcal{S})} 6\inf(j, k-1) \operatorname{Prob}(S \text{ arises during the construction})$$

Similar calculations prove that this is $O(k^3 n)$, using Lemma 5.10. □

Since it takes constant time to process a node, Lemma 5.16 yields the following proposition :

Proposition 5.17 *The expected cost of Procedure* location *is* $O(n \log n + k^3 n)$.

5.2.4 Analysis of the expected cost of Procedure creation

This cost is the cost of creating all the vertices of the order $\leq k$ Voronoi diagrams, plus the cost of maintaining the adjacency relationships, after each insertion of a new site $\mathtt{m}$. If $x(\mathtt{m})$ is the number of new triangles created after the insertion of $\mathtt{m}$, the first cost is $O(x(\mathtt{m}))$, since creating a triangle costs a constant time. As we have seen in Section 5.1.3.5, the second cost is $O(x(\mathtt{m}) \log k)$. The overall cost is thus, using Lemma 5.14 :

$$O\left(\sum_{\mathtt{m}} x(\mathtt{m})\right) + O\left(\sum_{\mathtt{m}} x(\mathtt{m}) \log k\right) \leq O(k^2 n \log k)$$

As mentioned in Remark 5.8, this could be improved to $O(k^2 n)$ complexity.

Proposition 5.18 *The expected cost of Procedure* creation *is* $O(k^2 n \log k)$.

Altogether, Propositions 5.15, 5.17 and 5.18 prove Theorem 5.9.

5.3 l-nearest neighbors

The k-Delaunay Tree contains the combinatorial structure of any order l Voronoi diagram ($l \leq k$). We show in the following section how such a diagram can be extracted from the k-Delaunay Tree.

As shown by the analysis of Procedure location, the k-Delaunay Tree is an efficient data structure to perform point location. Section 5.3.2 shows that it can be readily used to search the l nearest neighbors of a given point.

5.3.1 Deducing the order l Voronoi diagram from the k-Delaunay Tree ($l \leq k$)

Theorem 5.19 *$Vor_l(\mathcal{S})$ ($l \leq k$) can be deduced from the k-Delaunay Tree in time proportional to the size of $Vor_l(\mathcal{S})$, which is $O(ln)$.*

> *Proof.* Remember that the k Delaunay Tree maintains all the adjacency relations in $Vor_l(\mathcal{S})$. So we can find $Vor_l(\mathcal{S})$ in time proportional to its size, as soon as we know one of its vertices. We thus only have to find one vertex of each Voronoi diagram.
>
> Assume we know an infinite triangle $\mathtt{pq}\infty$ of current width 0. Its finite edge $[\mathtt{pq}]$ is necessarily an edge of the current convex hull. Let $\mathtt{s}_0$ be the

site such that $\mathtt{pqs}_0$ is a neighbor of $\mathtt{pq}\infty$ and that $B(\mathtt{pqs}_0)$ is empty. Let $\mathtt{s}_1, \ldots, \mathtt{s}_{k-1}$ be the sites such that triangle $\mathtt{pqs}_i$ is the including neighbor of triangle $\mathtt{pqs}_{i-1}$, $i = 1, \ldots, k-1$. Such sites always exist (provided that $k < n-1$) since $[\mathtt{pq}]$ is an edge of the convex hull of the already inserted sites. Each triangle $\mathtt{pqs}_i$ is associated to a node in the k-Delaunay Tree and its current width is i. Triangle $\mathtt{pqs}_i$ is dual to a vertex ν_i of $Vor_{i+1}(\mathcal{S})$.

If we maintain a pointer to an infinite triangle of current width 0 (which can be done in constant time at each stage of the construction), we can find the $\mathtt{s}_i$, $i \leq l$, and thus a vertex of each $\leq l$ Voronoi diagrams, in time $O(l)$. □

Remark 5.20 Now, suppose we want to label the regions of $Vor_i(\mathcal{S})$, $i \leq k$, by the i nearest sites of an (arbitrary) point of the interior of that region. We remark that the label of one region of $Vor_{i+1}(\mathcal{S})$ incident to ν_i is $\{\mathtt{s}_0, \mathtt{s}_1, \ldots, \mathtt{s}_i\}$. As soon as we know the label of one particular region, we can deduce the labels of all the other regions by traversing the diagram. Each time we cross an edge E, we come out of one region, say C, and enter another region, say C'. The labels of C and C' differ by only one site. The site of the label of C which is not in the label of C', and the site of the label of C' which is not in the label of C, are precisely the two sites whose bisector supports E. We have to substitute this first site by the second one in the label of C to get the label of C'.

5.3.2 Finding the l nearest neighbors

Let us consider now the problem of finding the l nearest neighbors ($l \leq k$) of a given point. This problem is equivalent to finding the label of the region $V(\{\mathtt{p}_1, \mathtt{p}_2, \ldots, \mathtt{p}_l\})$ of $Vor_l(\mathcal{S})$ which contains point $\mathtt{m}$. Procedure location gives the vertices of the Voronoi diagrams whose associated disk contains $\mathtt{m}$. It remains to show how to find the label of a region. This point must be clarified since our structure represents vertices of the Voronoi regions and we know, from Lemma 5.1, that the label of a Voronoi polygon is not included in the union of the labels of its vertices and thus, cannot be deduced from them (the label of a vertex consists of the three sites which are the vertices of its dual triangle ; it is also the symmetric difference of the labels of its incident regions).

However, we can take advantage of the fact that we know all the order $\leq l$ Voronoi diagrams to compute the label of a region. The following lemma will help in that task.

Lemma 5.21 *Let* $\mathtt{m} \in V(\{\mathtt{p}_1, \mathtt{p}_2, \ldots, \mathtt{p}_l\})$ *in* $Vor_l(\mathcal{S})$*, where* $\mathtt{p}_1, \mathtt{p}_2, \ldots, \mathtt{p}_l$ *are some sites of* $\mathcal{S}$ *(with* $\delta(\mathtt{p}_i, \mathtt{m}) \leq \delta(\mathtt{p}_j, \mathtt{m})$ *if and only if* $i \leq j$*). Then, for each* $i \in \{1, \ldots, l\}$*, there exists a vertex* ν_i *of* $Vor_i(\mathcal{S})$*, such that* ν_i *is dual to a triangle* S_i *having* $\mathtt{p}_i$ *as a vertex, and such that* $\mathtt{m} \in B(S_i)$*.*

Proof. Since $\mathtt{p}_l$ is a l^{th} nearest site from $\mathtt{m}$, we have $\delta(\mathtt{p}_i, \mathtt{m}) \leq \delta(\mathtt{p}_l, \mathtt{m})$, $\forall i = 1, \ldots, l$.

Let $\mathtt{q}$ be a point on the line $(\mathtt{m}\mathtt{p}_l) : \mathtt{q} = t\mathtt{m} + (1-t)\mathtt{p}_l, t \in I\!R$.

$$\begin{aligned} \delta(\mathtt{p}_l, \mathtt{q}) &= |t|\delta(\mathtt{p}_l, \mathtt{m}) \\ \delta(\mathtt{m}, \mathtt{q}) &= |1-t|\delta(\mathtt{p}_l, \mathtt{m}) \end{aligned}$$

Let us suppose that $t \geq 1$, i.e. $\mathtt{m}$ lies between $\mathtt{q}$ and $\mathtt{p}_l$.

The following inequalities hold, for $i = 1, \ldots, l$:

$$\begin{aligned} \delta(\mathtt{p}_i, \mathtt{q}) &\leq \delta(\mathtt{p}_i, \mathtt{m}) + \delta(\mathtt{m}, \mathtt{q}) \\ &\leq \delta(\mathtt{p}_l, \mathtt{m}) + (t-1)\delta(\mathtt{p}_l, \mathtt{m}) \\ &= t\delta(\mathtt{p}_l, \mathtt{m}) \\ &= \delta(\mathtt{p}_l, \mathtt{q}) \end{aligned}$$

So, if we move $\mathtt{q}$ on the half line $D^+_{\mathtt{m}\mathtt{p}_l} = \{t\mathtt{m} + (1-t)\mathtt{p}_l, t \geq 1\}$ supported by $(\mathtt{m}\mathtt{p}_l)$, a furthest site from $\mathtt{q}$ among $\{\mathtt{p}_1, \mathtt{p}_2, \ldots, \mathtt{p}_l\}$ remains $\mathtt{p}_l$, the same as from $\mathtt{m}$. We can deduce that the intersection of $D^+_{\mathtt{m}\mathtt{p}_l}$ with the boundary of $V(\{\mathtt{p}_1, \mathtt{p}_2, \ldots, \mathtt{p}_l\})$ is a point $\mathtt{q}$ belonging to the bisecting line $Bis(\mathtt{p}_l, \mathtt{p}')$ of $[\mathtt{p}_l, \mathtt{p}']$, where $\mathtt{p}'$ is a site of $\mathcal{S}$ not in $\{\mathtt{p}_1, \mathtt{p}_2, \ldots, \mathtt{p}_l\}$. Let us denote by F the edge of $V(\{\mathtt{p}_1, \mathtt{p}_2, \ldots, \mathtt{p}_l\})$ supported by $Bis(\mathtt{p}_l, \mathtt{p}')$ (see Figure 5.7).

Let us define $R = \{r \in Bis(\mathtt{p}_l, \mathtt{p}') / \mathtt{m} \in C(r, \mathtt{p}_l)\}$ where $C(r, \mathtt{p}_l)$ denotes the disk of center r and radius $\delta(r, \mathtt{p}_l)$. R is a half line of $Bis(\mathtt{p}_l, \mathtt{p}')$ containing $\mathtt{q}$. So $F \cap R \neq \emptyset$, which implies that one of the two end points of F, say ν_l, belongs to R. ν_l is a vertex of $Vor_l(\mathcal{S})$ dual to a triangle S_l having $\mathtt{p}_l$ as one of its vertices, and whose disk contains $\mathtt{m}$.

The lemma is thus proved for $\mathtt{p}_l$. For the other $\mathtt{p}_i, i = 1, 2, \ldots, l-1$, the same proof still holds, considering $Vor_i(\mathcal{S})$ instead of $Vor_l\mathcal{S}$). □

If we want to find the label of the region of $Vor_l(\mathcal{S}), l \leq k$, which contains $\mathtt{m}$, we can locate it, by applying Procedure location to the l-Delaunay Tree. We first find all the triangles whose balls contain $\mathtt{m}$. We then have to find, among the vertices of those triangles, the l nearest sites from $\mathtt{m}$, which can be done in a straightforward way.

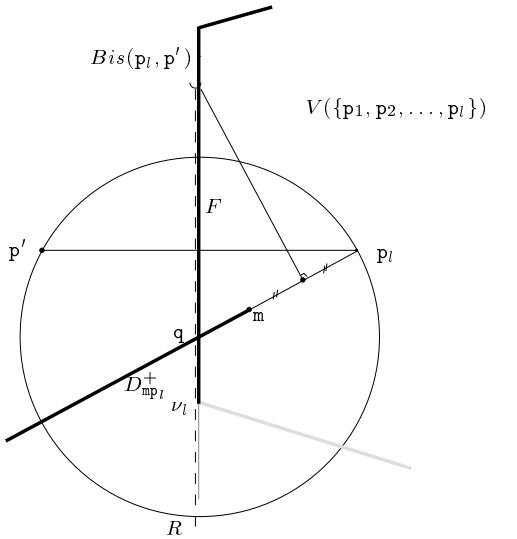


Figure 5.7 : For the proof of Lemma 5.21

5.4 The k-Delaunay Tree in higher dimensions

In this section, we generalize the previous results to higher dimensions. Let $\mathcal{S}$ be a set of n sites in d-space such that no subset of $d+2$ sites lie on a same sphere and no subset of $d+1$ sites are coplanar.

The notion of including and excluding neighbors can be generalized. A simplex has $d+1$ excluding neighbors and $d+1$ including ones, one through each of its facets. To any pair of adjacent simplices correspond an edge in some higher order Voronoi diagram. If ν is a close-type vertex (resp. far-type vertex) of $Vor_k(\mathcal{S})$, the edges of $Vor_k(\mathcal{S})$ issued from ν correspond to the neighborhood relationships between the simplex dual to ν and its excluding (resp. including) neighbors. If ν is a medium-type vertex (see section 1.2.2), the edges of $Vor_k(\mathcal{S})$ issued from ν correspond to both types of neighborhood relationships.

5.4.1 The d dimensional k-Delaunay Tree

We can generalize the structure developed in Section 5.1.3. The d-dimensional Delaunay Tree is a direct acyclic graph satisfying the following property :

> ($\mathcal{P}$) *all the simplices of current width strictly less than k are present in the d-dimensional k-Delaunay Tree.*

The extension of the construction of the k-Delaunay Tree to higher dimensions is straightforward.

The technique of Section 5.3.1 that deduces the order l Voronoi diagram from the k-Delaunay Tree can be extended to higher dimensions in a straightforward manner. As in Section 5.3.1, we can traverse the graph consisting of the vertices and the edges of the order l Voronoi diagram and compute the labels of each region. From this graph and the labels, it is plain to compute the faces of all dimensions of the diagram.

The algorithm presented in Section 5.3.2 can also be extended without difficulty, to compute the l nearest neighbors of a given site ($l \leq k$).

5.4.2 Analysis of the randomized construction

This section presents the following theorem, which generalizes Theorem 5.9 to d dimensions :

Theorem 5.22 *For any set of n sites, if we randomize the sequence of their insertions, the k-Delaunay Tree (and thus the order $\leq k$ Voronoi diagrams) of the n sites can be constructed in expected time $O\left(k^{\lceil \frac{d+1}{2} \rceil + 1} n^{\lfloor \frac{d+1}{2} \rfloor}\right)$ using expected storage $O\left(k^{\lceil \frac{d+1}{2} \rceil} n^{\lfloor \frac{d+1}{2} \rfloor}\right)$.*

Our bounds are not as good as those of K. Mulmuley [Mul91a] : the increase by one in the exponent of k is a consequence of the fact that we maintain, at each stage of the incremental insertion, the complete informations relative to all order $\leq k$ Voronoi diagrams, whereas Mulmuley only maintains the vertices of the diagrams, without maintaining their order (which can be deduced at the end of the construction).

We will see in Chapter 7 the works of K. Mulmuley concerning dynamic algorithms.

We only give the main steps achieving the proof of Theorem 5.22.

As in Section 5.2.3, we can define, for $d+2$ sites $\mathtt{x}_1, \mathtt{x}_2, \ldots, \mathtt{x}_{d+2}$, the bicycle $\mathtt{x}_1(\mathtt{x}_2 \ldots \mathtt{x}_{d+1})\mathtt{x}_{d+2}$. We also define the width of a simplex and the width of a bicycle.

The following lemmas generalize Lemmas 5.10 and 5.11.

Lemma 5.23 *Let $\mathtt{x}_1\mathtt{x}_2 \ldots \mathtt{x}_{d+1}$ be a simplex having width j. $\mathtt{x}_1\mathtt{x}_2 \ldots \mathtt{x}_{d+1}$ will arise as a vertex of some order $\leq k$ Voronoi diagram during the construction with probability :*

$$\begin{cases} \dfrac{k(k+1)\ldots(k+d)}{(j+1)(j+2)\ldots(j+d+1)} & \text{if} \quad j \geq k \\ 1 & \text{if} \quad j < k \end{cases}$$

Lemma 5.24 *Let $\mathtt{x}_1(\mathtt{x}_2 \ldots \mathtt{x}_{d+1})\mathtt{x}_{d+2}$ be a bicycle of width j such that $\mathtt{x}_2\mathtt{x}_3 \ldots \mathtt{x}_{d+2}$ is a son or a stepson of $\mathtt{x}_1\mathtt{x}_2 \ldots \mathtt{x}_{d+1}$. The probability that such a bicycle appears during the construction is*

$$\frac{(d+1)!}{(j+1)\ldots(j+d+2)}$$

Applying Theorem 2.3, we have :

Lemma 5.25 *The number of simplices having width at most j is*

$$|\mathcal{F}_{\leq j}(\mathcal{S})| = O\left(n^{\lfloor \frac{d+1}{2} \rfloor}(j+1)^{\lceil \frac{d+1}{2} \rceil}\right)$$

Lemma 5.26 *The number of bicycles having width at most j is*

$$|\mathcal{G}_{\leq j}(\mathcal{S})| = O\left(n^{\lfloor \frac{d+1}{2} \rfloor}(j+1)^{\lceil \frac{d+1}{2} \rceil+1}\right)$$

Let us now compute the complexity of the k-Delaunay Tree.

Proposition 5.27 *The expected space complexity of the k-Delaunay Tree is*

$$O\left(k^{\lceil \frac{d+1}{2} \rceil} n^{\lfloor \frac{d+1}{2} \rfloor}\right)$$

Proof. The proof is similar to the one of Proposition 5.15, using Lemmas 5.23 and 5.25, and the fact that, as in 2 dimensions, the total number of links to sons and stepsons is less than the number of nodes (Lemma 5.6), and each node has at most $2(d+1)$ neighbors. □

Proposition 5.28 *The expected cost of Procedure* location *is*

$$O\left(k^{\lceil \frac{d+1}{2} \rceil+1} n^{\lfloor \frac{d+1}{2} \rfloor}\right)$$

Proof. Lemmas 5.24 and 5.26 allow to prove the result, using arguments similar to those used in the proof of Lemma 5.16. □

The cost of Procedure creation is the cost of creating all the vertices of the order $\leq k$ Voronoi diagrams plus the cost of maintaining the neighborhood relationships, after each insertion of a new site `m`. The computation of the neighborhood relationships between the new simplices created by the insertion of a new point `m` is the same as in the two dimensional case, an edge is simply replaced by a $(d-1)$-face.

So we obtain :

Proposition 5.29 *The expected cost of Procedure* creation *is*

$$O\left(k^{\lceil \frac{d+1}{2} \rceil} n^{\lfloor \frac{d+1}{2} \rfloor} \log k\right)$$

Altogether, Propositions 5.27, 5.28 and 5.29 prove Theorem 5.22.

5.5 Experimental results

It is to be noted that the algorithm is simple, even if its description and its analysis may look rather intricate ! The core of the algorithm is given in Figures 5.3 and 5.4. Moreover, the numerical computations involved are also quite simple : they consist mostly of comparisons of (squared) distances in order to check if a point lies inside or outside a ball. The algorithm has been implemented in the two dimensional case. The program consists of less than 1000 lines of C. It has run on many examples with different kinds of point distributions. Some results and statistics are presented in Figures 5.8 to 5.16.

5.5.1 Influence of randomization

In Figure 5.8, the points lie on three ellipses, two for the *eyes* and one for the *head*. We tried several permutations of the points for the computation of the 3-Delaunay Tree :

(1) Points of the head in the order along the ellipse, and then points in the order along each eye.

(2) The same as the preceding, except that one point of an eye is inserted first.

(3) Points on the eyes in order, and then points on the head in order.

(4) A random permutation.

(5) Another random permutation.

In Figure 5.9, the function drawn in dotted line shows the total number of vertices of the order ≤ 3 Voronoi diagrams, versus the number of inserted sites. The functions drawn in bold line show the numbers of nodes in the 3-Delaunay Tree, for the different permutations. The functions drawn in thin line show the numbers of nodes visited by the first part of Procedure location (we will call those nodes *1-visited nodes* for short, since they correspond to a traversal of the *(1-)Delaunay Tree*), to find a Delaunay triangle in conflict with the new point.

Figure 5.10 gives some statistics about the computation using the different permutations : the size of the 3-Delaunay Tree and the sum of the sizes of the order ≤ 3 Voronoi diagrams at the end of the execution, and for the insertion of one point, the maximal and average numbers of 1-visited nodes and created nodes.

This example shows that, if degenerate orders may be really unefficient, the behaviour for random order, or even for the third permutation is satisfying.

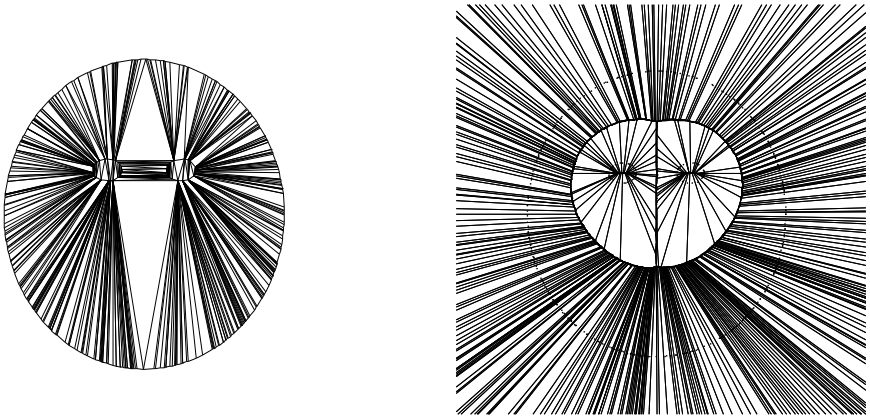


Figure 5.8 : Delaunay triangulation and order 3 Voronoi diagram of a set of 415 points

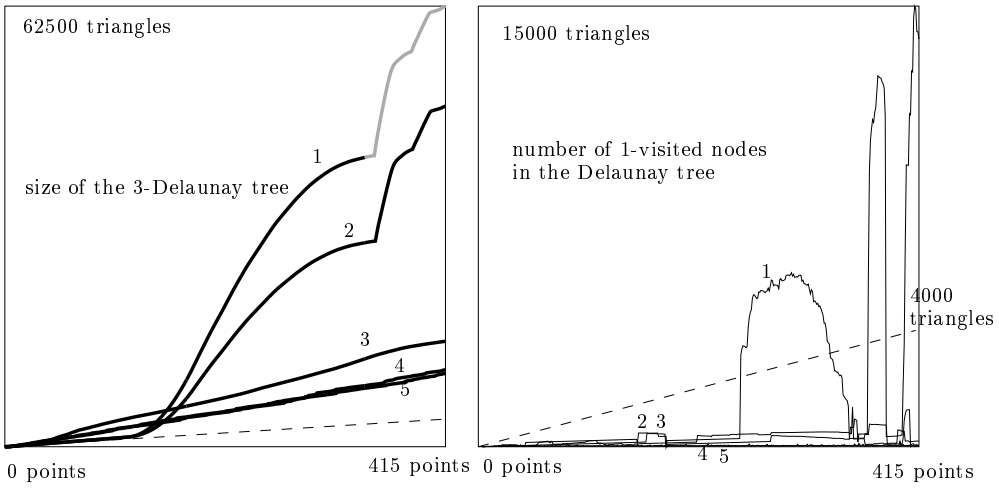


Figure 5.9 : Results for the set of points of Figure 5.8, $k = 3$, different permutations for the insertion

Permutation	1	2	3	4	5
Size of the 3-Delaunay Tree	62522	48334	14958	10903	10395
Size of the ≤ 3 Voronoi	3917	3917	3917	3917	3917
Max nb of 1-visited nodes	14835	1245	1329	181	73
Average nb of 1-visited nodes	2123	245	239	33.5	27
Max nb of created triangles	1044	853	71	210	179
Average nb of created triangles	151	117	36	26.4	25.1

Figure 5.10 : Statistics for the set of points of Figure 5.8

5.5.2 Influence of k

To study the effect of increasing k on the algorithm, we use the set of points depicted in Figure 5.11.

Figure 5.12 presents, on the left side, the size of the (1-)Delaunay Tree in bold line, the size of the (order 1) Voronoi diagram in dotted line and in thin line the number of nodes that are 1-visited by Procedure location, which does not depend on k. The right side of Figure 5.12 shows the size of the k-Delaunay Tree and the sum of the sizes of the order $\leq k$ Voronoi diagrams for k = 1,2,3,4,6 and 10.

Figure 5.13 presents statistics similar to those in Figure 5.10.

5.5.3 Influence of the point distribution

These last statistics concern the influence of the point distribution. To this aim, we compute the 3-Delaunay Tree for the four sets of 400 points described in Figures 5.8, 5.11 and 5.14. Figure 5.15 shows in this left part the size of the 3-Delaunay Tree, and the sum of the sizes of the order ≤ 3 Voronoi diagrams and the number of 1-visited nodes for the four sets. This figure demonstrates that, with a randomization of the input data, the average behaviour of the algorithm does not depend on the distribution of the points. The better result for points of Figure 5.8 is only due to the smallest size of the output (which is related to the points on the k-hulls). One can see also that the location time is very small in comparison with the sum of the sizes of the ≤ 3 Voronoi diagrams. The right part of Figure 5.15 shows only the location time for the forty last points. Figure 5.16 presents statistics as in Figures 5.10 and 5.13.

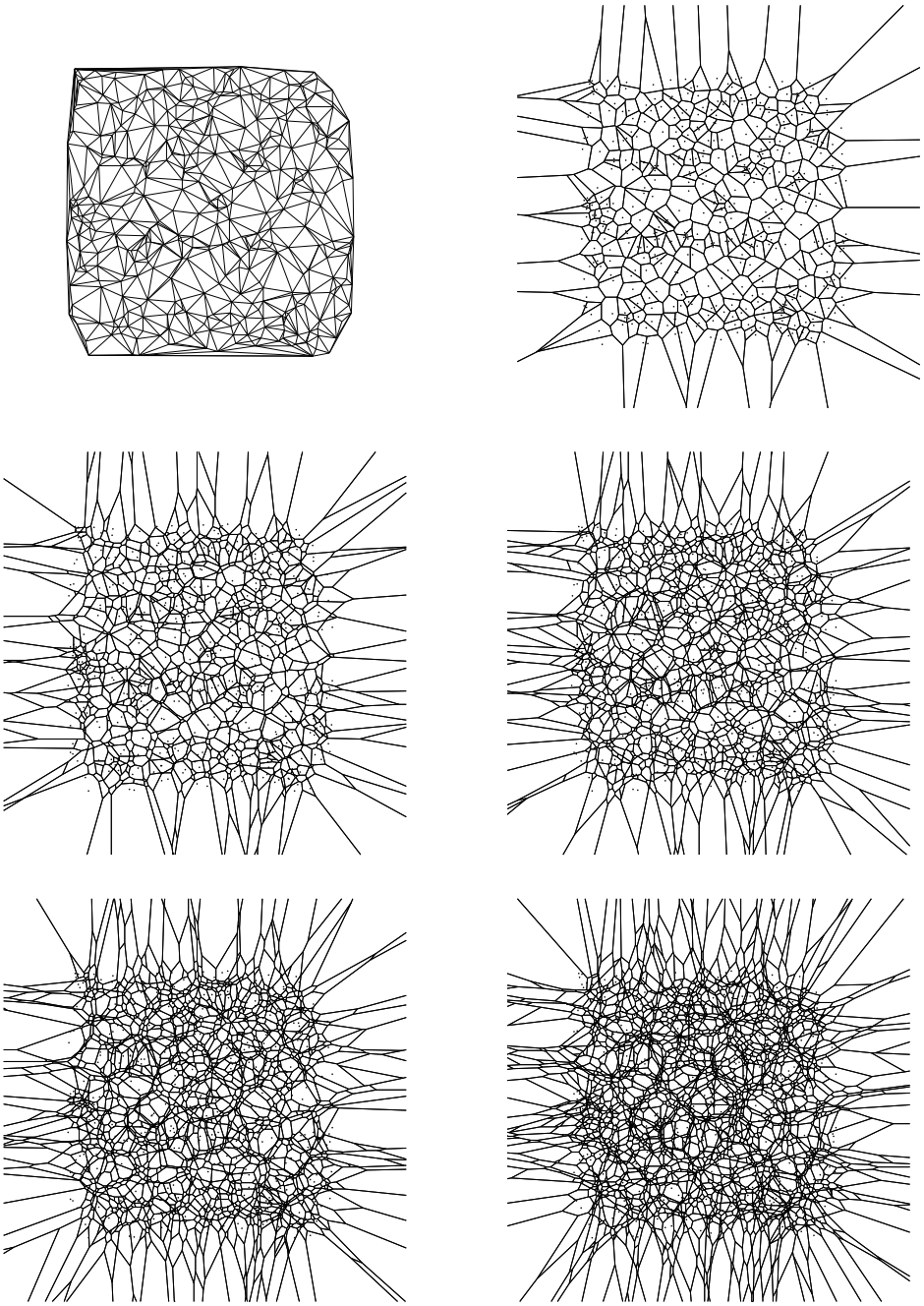


Figure 5.11 : Delaunay triangulation and order 1,2,3,4 and 6 Voronoi diagrams of a set of 400 random points in a square

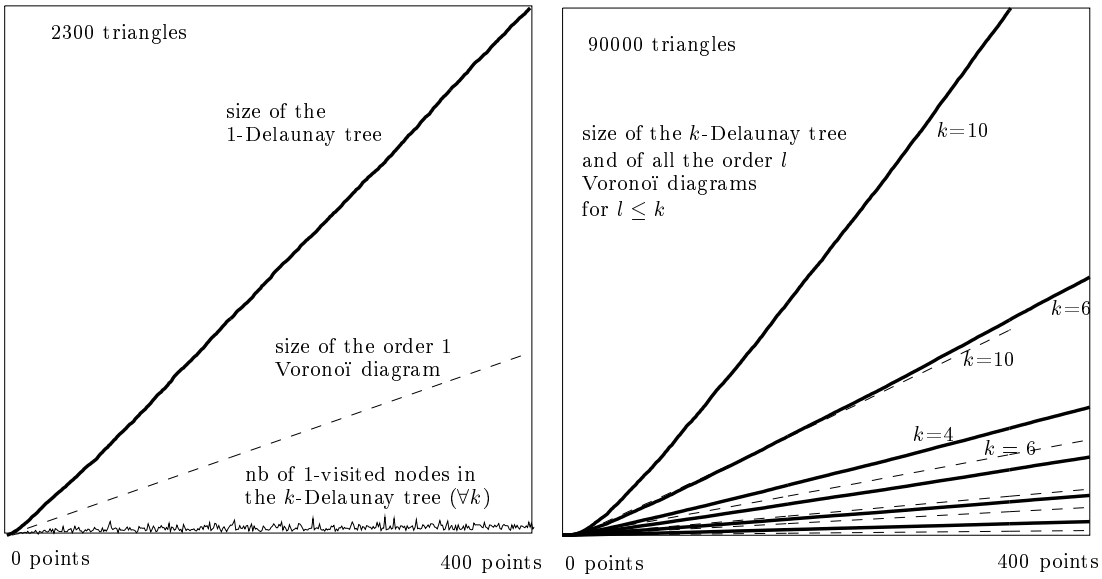


Figure 5.12 : Results for the set of points of Figure 5.11, for different values of k

k	1	2	3	4	6
Size of the k-Delaunay Tree	2307	6748	13246	21694	43740
Size of the $\leq k$ Voronoi	799	2378	4720	7815	16198
Max nb of 1-visited nodes	79	79	79	79	79
Average nb of 1-visited nodes	31	31	31	31	31
Max nb of created triangles	10	29	50	80	150
Average nb of created triangles	5.8	16.9	33.2	54.5	110

Figure 5.13 : Statistics for the set of points of Figure 5.11

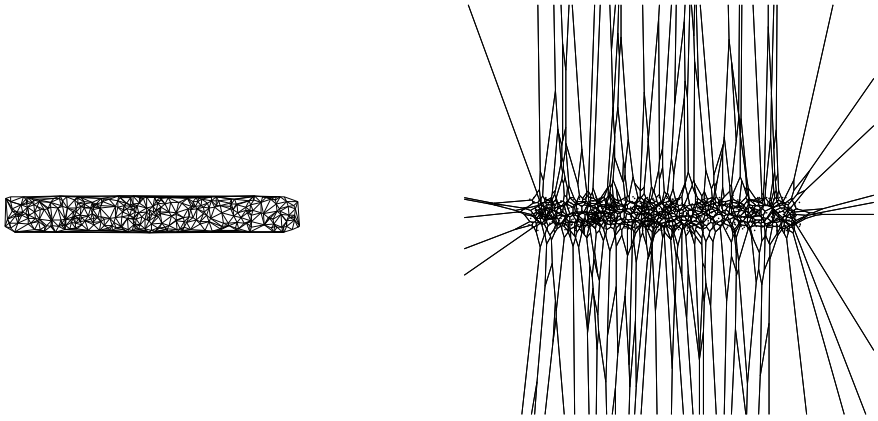


Figure 5.14 : Delaunay triangulation and order 3 Voronoi diagram of a set of 400 random points in a thin rectangle

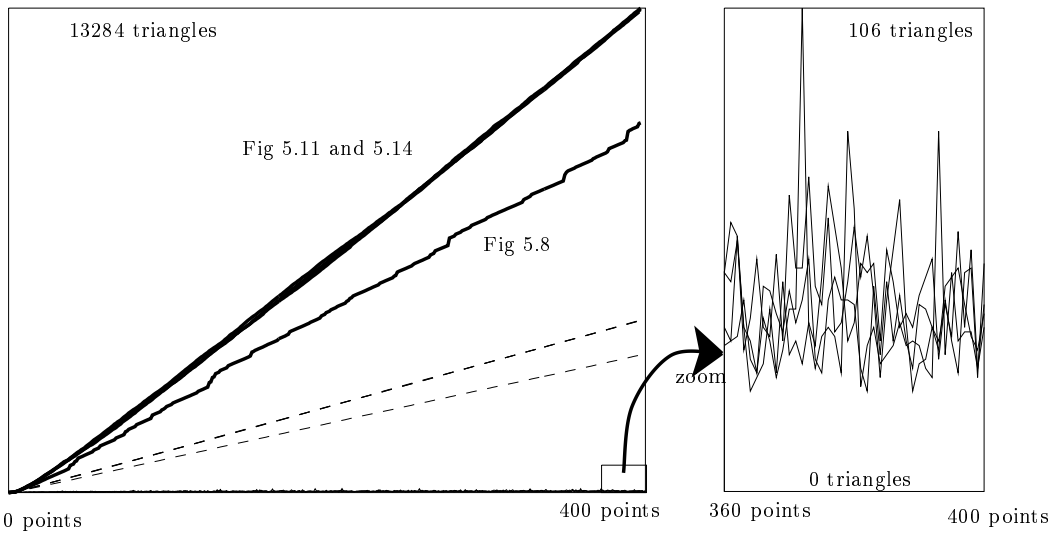


Figure 5.15 : Results for the set of points of Figures 5.8, 5.11 and 5.14 for $k = 3$

Set of points	Fig. 7	Fig. 10	Fig. 13	Fig. 14
Size of the 3-Delaunay Tree	10141	13246	13284	13184
Size of the ≤ 3 Voronoi	3776	4720	4718	4718
Max nb of 1-visited nodes	109	79	90	78
Average nb of 1-visited nodes	30	31	33	34
Max nb of created triangles	236	50	67	57
Average nb of created triangles	25.6	33.2	33.1	33.1

Figure 5.16 : Statistics for the sets of points of Figures 5.8, 5.11 and 5.14

The experimental results show that the algorithm performs well even on degenerate distributions of points.

Conclusion

We have shown that the k-Delaunay Tree of n points can be constructed in $O(n \log n + k^3 n)$ (resp. $O\left(k^{\lceil \frac{d+1}{2} \rceil + 1} n^{\lfloor \frac{d+1}{2} \rfloor}\right)$) randomized expected time in the plane (resp. in d space). Its randomized expected size is $O(k^2 n)$ (resp. $O\left(k^{\lceil \frac{d+1}{2} \rceil} n^{\lfloor \frac{d+1}{2} \rfloor}\right)$). The k-Delaunay Tree allows to compute the order $\leq k$ Voronoi diagrams of n points within the same bounds. Any order $l \leq k$ Voronoi diagram can be deduced from the k-Delaunay Tree in time proportional to its size, which is $O(ln)$ in two dimensions. Moreover, the k-Delaunay Tree can be used to find the l nearest neighbors of a given point.

The structure could be extended to deal with k-levels of general arrangements. But in this case, it is not sufficient to store the Delaunay Tree augmented with additional neighborhood relations, since the hyperplanes are not supposed to have one face on the lowest level. We must store parent pointers between old and new triangles of all widths.

An important point is that these results hold whatever the point distribution may be.

The algorithm is simple and, moreover, the numerical computations involved are also quite simple : they consist mostly of comparisons of (squared) distances in order to check if a point lies inside or outside a ball. Experimental results, for uniform as well as degenerate distributions of points, have provided strong evidence that this algorithm is very effective in practice, for small values of k.

For large values of k, a similar structure, based on the order k furthest neighbors Voronoi diagrams, could be derived. It would provide results similar to the ones above to construct all order $\geq n - k$ Voronoi diagrams and to find l furthest neighbors for $l \leq k$.

Chapitre VI

Vers une structure complètement dynamique

We describe here the dynamization of the Influence Graph, on two examples : the Delaunay triangulation of point sites in any dimension, and the arrangement of line segments in the plane. In both cases, the main idea is the same : when an object is removed from the structure, we decide to "reconstruct the past", as if this object had never existed.

However, in spite of this similarity, the details of the algorithms differ a lot, as will be seen, and it seems to be difficult to design a general —and precise, too— framework for dealing with deletion of an object.

[CMS92] uses the same idea, of restoring the history as if the object had never been inserted, for convex hulls in any dimension. [Sch91] computes the whole history of insertions, but also deletions, and in this way gets a bigger history that he has to clean up sometimes. [Mul91b] uses a different technique which does not involve the history of the construction. All these approaches will be rapidly presented in Chapter 7.

6.1 Removing a site from the Delaunay triangulation

In this section, we describe an algorithm maintaining the Delaunay Tree under insertions and deletions of sites. This can be done in $O(\log n)$ expected time for an insertion and $O(\log \log n)$ expected time for a deletion in the plane [DMT], where n is the number of sites currently present in the structure. For deletions, by expected time, we mean averaging over all already inserted sites for the choice of the deleted sites. The algorithm has been effectively coded and experimental results are given. The method extends to higher dimensions.

Let $\mathcal{S}$ be a set of n sites in the euclidean plane, such that no four sites are cocircular.

We assume that the Delaunay Tree has been constructed for the set $\mathcal{S}$, by using the incremental randomized algorithm (see Chapter 3). We now want to remove a site `p` of $\mathcal{S}$. All the triangles incident to `p` must be removed from the Delaunay Tree : some of them are triangles of the Delaunay triangulation of $\mathcal{S}$ (so they are leaves of the Delaunay Tree), but other ones already died ; they correspond to internal nodes of the Delaunay Tree, and must be removed, too. Moreover, we must restore the Delaunay Tree in the same state it would be in if `p` had never been inserted, and if the other sites had been inserted in the same order. That way, we preserve the randomized hypothesis on the sequence of sites, and the conditions for further insertions or deletions are fulfilled.

We must thus reconstruct a past for the final triangulation in which `p` takes

no part. The deletion of p creates a "hole" in each successive triangulation after the insertion of p, which the tree keeps a trace of. The idea of our algorithm is to fill each hole with the right Delaunay triangulation.

Let us describe the structure of a node of the Delaunay Tree, obtained from the structure given in Chapter 3 by adding a few new fields (some of them have not been used yet and will be defined in the following) :
• the triangle : creator vertex, two other vertices, circumscribed circle
• a mark **dead**
• pointers to the at most three sons and the list of stepsons
• pointers to the father and the stepfather
• the three current neighbors if the triangle is not dead, the three neighbors at the death otherwise
• the three neighbors at the time of the creation
• two special neighbors
• a pointer **killer** to the site that killed the triangle
• a mark **to be removed**
• three pointers **star** to elements of structure **Star**

The insertion phase described in Chapter 3 can obviously maintain those additional informations.

6.1.1 Different kinds of modified nodes

Let us describe how the deletion of p affects the nodes in the Delaunay Tree.

Some nodes must be removed : they correspond to triangles having p as a vertex. Depending on its time of creation there are two cases for such a node : either it has been created by the insertion of p, or it has been created by the insertion of some site afterwards ; the latter occurs *iff* its father and stepfather both have p as a vertex and thus both the parents must be removed, too. During the construction of the Delaunay Tree, some sites did not create any triangle to be removed now, but if a site x created such a triangle, it created in fact two triangles to be removed : the two triangles created by x sharing edge px, see Figure 6.1. One of them, say $\mathtt{xx}_1\mathtt{p}$, is oriented clockwise, and the second one, $\mathtt{xx}_2\mathtt{p}$, is oriented counterclockwise.

A node to be deleted $\mathtt{xx}_2\mathtt{p}$ may have a son (or a stepson) $\mathtt{xx}_2\mathtt{q}$ that does not have p as a vertex and thus must remain in the Delaunay tree, see Figure 6.1. Such a node loses just one of its two parents and is therefore called *unhooked*. We must find a new parent in replacement of the lacking one.

The sketch of the method is the following :

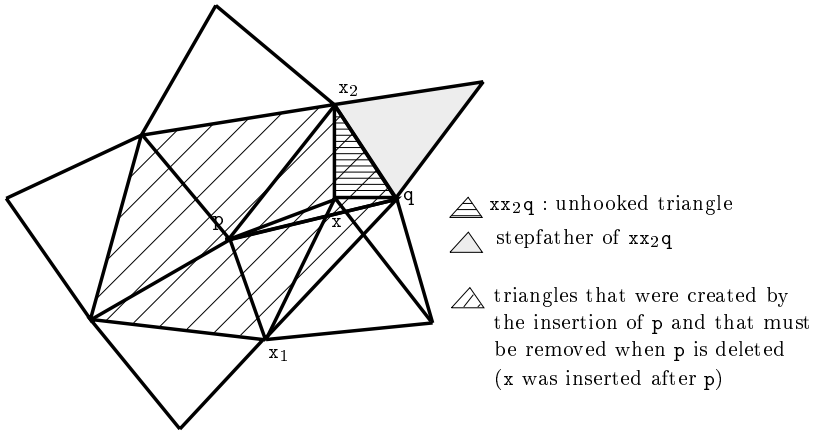


Figure 6.1 : The two kinds of modified nodes

Search step : Find all nodes of the Delaunay Tree that have to be removed, and all unhooked nodes
Reinsertion step : Locally reinsert the sites that are creators of the triangles found during the Search step, and update the triangulation

6.1.2 The Search step

By the discussion above, the set of nodes to be removed can be found by searching the Delaunay Tree starting from the nodes that were created by `p`. At each node marked to be removed we visit all its sons and stepsons recursively. If one of them has `p` as a vertex, it will be marked to be removed as well. Otherwise it is an unhooked node. The creator of both these types of triangles must be reinserted, in order to replace the removed triangles by other triangles, and to hang up unhooked triangles again.

In order to be able to perform the Reinsertion step, we must store the list of sites to be reinserted :

We need an auxiliary structure, **Reinsert**, which is a balanced binary tree consisting of the set of sites which created the nodes to be removed and the unhooked nodes ; the sites are sorted by order of insertion. This will allow us to reconstruct the triangles which will fill the holes

in the successive triangulations, and to hang up again the unhooked nodes.

An element of **Reinsert** contains :
• the site `x` to be reinserted
• pointers to the two triangles $\texttt{xx}_1\texttt{p}$ and $\texttt{xx}_2\texttt{p}$ that were created by the insertion of `x`, if they exist (see Section 6.1.1). $\texttt{xx}_1\texttt{p}$ is oriented clockwise
• the list of unhooked triangles that were created by the insertion of `x`

The search is initialized by the set C of nodes created by `p`.

To this aim, we must maintain an auxiliary array, **Created**, containing, for each site `s` of $\mathcal{S}$, a pointer to one of the nodes created by `s`.

From this node, we can then compute the set C using the neighborhood relations at the time of creation and examining the creator of the triangles (Figure 6.2).

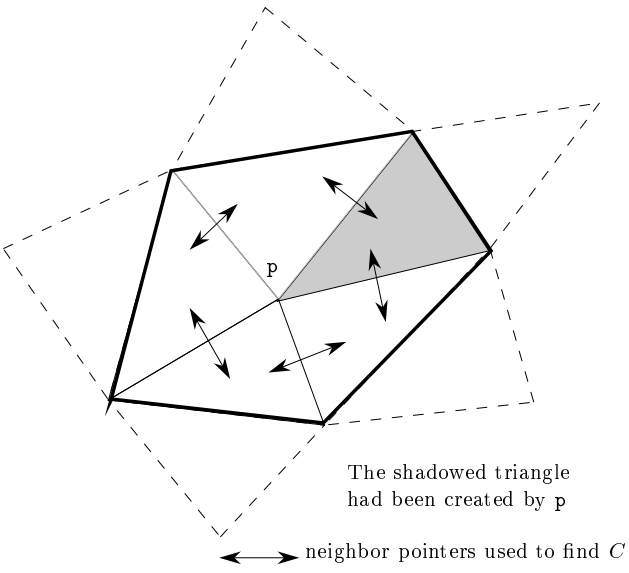


Figure 6.2 : The search step : Initialization

```
for each element S of C
    for each son or stepson U of S
        examine(U)
    endfor ;
    remove links between S and its father and stepfather ;
    put S in "garbage collector"
endfor
```

We then simply recursively traverse the subgraph consisting of removed and unhooked nodes. Each son or stepson of a removed node is removed if it has p as a vertex and unhooked otherwise. All these nodes are added in the element of **Reinsert** associated to their creator. This process is detailed above.

```
examine(T)
if T is marked to be removed
    { T has already been visited }
    nothing
else
if p is a vertex of T
{ T = xsp}
    mark T to be removed ;
    for each son or stepson S of T
        examine(S)
    endfor ;
    locate the creator x of T in Reinsert ;
        { if the location fails, a new element is created }
    if xsp turns clockwise
        store T as the field xx1p of x in Reinsert
    else
        store T as the field xx2p of x in Reinsert
    endif
else
    { T is either son or stepson of a removed node }
    locate the creator x of T in Reinsert ;
    add T to the list of unhooked nodes associated to x ;
    set the pointer to the removed parent of T as null
endif
```

Observe that in the search step the triangles are only visited. They are removed from the Delaunay Tree later during the reinsertion step.

6.1.3 The Reinsertion step

The sites contained in **Reinsert** must be reinserted in the Delaunay Tree in order to construct the successive triangulations without site `p`. The scheme of the reinsertion of a site `x` is the same as the usual scheme of insertion, except that everything happens locally : the location of a site `x` to be reinserted in the whole Delaunay Tree is unnecessary and would be too expensive.

The location in a generally small set A (for active) of triangles is sufficient. At the beginning of the reinsertion set A is initialized with all triangles killed by the insertion of `p`. They can be found by looking at the fathers of the triangles in C and following their neighbor pointers at their death.

Then, during the Reinsertion step, A is maintained so that it is the set of triangles in conflict with `p` in the Delaunay triangulation at the time just preceding the reinsertion of `x`. In each step, A is modified as follows : all the triangles of A in conflict with `x` are killed by `x` and thus disappear from the Delaunay triangulation and from A. The triangles created by the reinsertion of `x` appear in A (Figure 6.3), because they are in conflict with `p` (otherwise they would have existed in the triangulation containing `p`). The triangles of A not in conflict with `x` still remain in A. The triangles outside A are not modified by a reinsertion since they are not in conflict with `p` ; only their neighborhood or stepson relations involving removed nodes must be updated.

More precisely, the set A of triangles must be organized so that the location of conflicts is efficient. We can notice that the triangles in A form a star-shaped polygon with respect to `p`, since they are in conflict with `p`.

The edges and vertices (sites) of this polygon are stored in counterclockwise order in a circular list called **Star**. Note that the vertices of **Star** are the sites that are adjacent to `p`. Furthermore we maintain some pointers :

- Each edge of **Star** points to the adjacent triangle in A.
- Each triangle in A points to the adjacent edges of **Star** (at most three, pointers **star** in the description of a node).
- Each site which is a vertex of **Star** points to the edge of **Star** following the site.

Some elements of A are not represented in **Star**, but the whole set A can nevertheless be traversed using **Star** and pointers to neighbors.

Star can be initialized by first choosing a vertex of a triangle in A. We then follow the boundary of the star-shaped polygon using the neighborhood relations, and the pointers **star**.

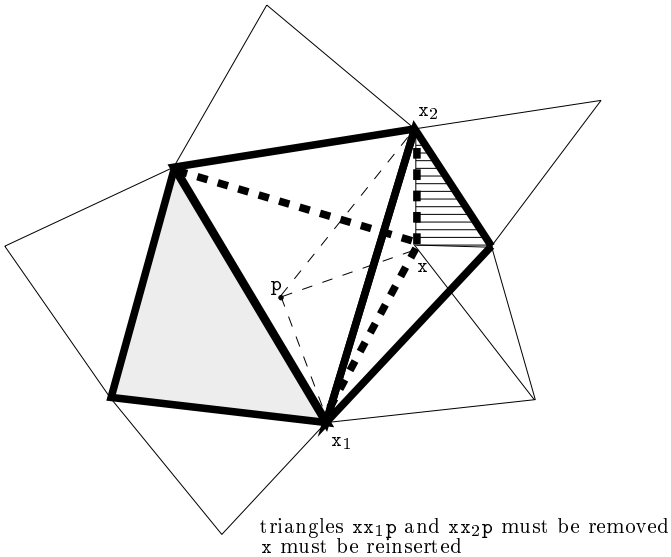


Figure 6.3 : A reinsertion

We know the current neighbors of each triangle of A. Each edge e of **Star** is an edge of a triangle U of A and of a triangle V that does not belong to A.

Special neighbors :
The current neighbor of U through e is V, but the reciprocal relation does not always exist ; the neighbor pointer of V through e may reach another triangle W created a long time later.

If W must not be removed, this pointer must remain after the deletion of `p`. So we do not want to systematically modify the current neighbors of triangles not belonging to A. We need to put a special neighbor pointer from V to U. Thus each triangle outside A, having an edge on the boundary of A, has a special neighbor pointer to the adjacent triangle in A. It is easy to see that two special neighbor pointers are enough : at most two edges of a given triangle lie on the boundary of a star-shaped polygon, if it is exterior to it. The special neighbor pointers store intermediate relations between triangles.

Nevertheless, if W must be removed, then a new neighbor must be found for V, and the current neighbor must be maintained identical to the special one. When we update a special neighbor U of V, if the neighbor at creation of V is removed, then U is also the neighbor at creation of V. Similarly, if the current neighbor of V is removed, or if it belongs to A, then it must be updated to be U.

Everything is now set up to start the reinsertion. Each site in Structure **Reinsert** is reinserted in the right order (the order used for first insertion).

Processing the unhooked triangles
Each element of **Reinsert** contains a site `x` to be reinserted, and the list of corresponding unhooked triangles. To hang up such a triangle T again, we only have to go to the remaining parent of it, which must have an edge in **Star**, and then hang T up to the appropriate special neighbor of this parent. There may also exist some removed triangles created by `x` (Figure 6.4). Notice that this is not always true (Figure 6.5). If there is no removed triangle, the unhooked triangle necessarily needs a stepfather, which is also the neighbor at creation and the special neighbor, all these three triangles are set to the special neighbor of the father.

Replacing the triangles to be removed by new ones
For each element `x` of **Reinsert**, we check if triangle $\mathtt{xx}_1\mathtt{p}$ (and $\mathtt{xx}_2\mathtt{p}$) exists. If $\mathtt{xx}_1\mathtt{p}$ and $\mathtt{xx}_2\mathtt{p}$ do not exist in the triangulation, then nothing has to be done.

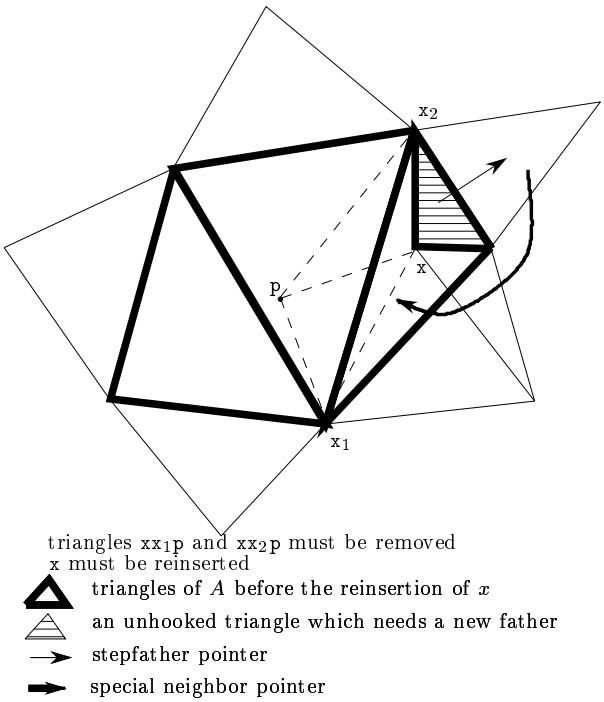


Figure 6.4 : An unhooked triangle with some removed triangles

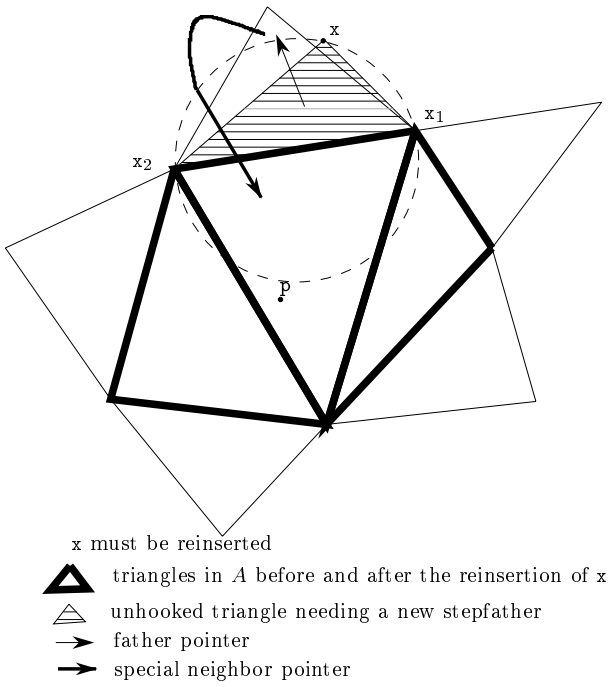


Figure 6.5 : An unhooked triangle in the case that there is no removed triangle

Otherwise we have to fill the gap of triangles incident at $\mathtt{x}$ between edges $\mathtt{x}\mathtt{x}_1$ and $\mathtt{x}\mathtt{x}_2$. We must look at **Star** in order to find the triangles that have to be created by the reinsertion of $\mathtt{x}$. There are two cases : there may exist no triangle of A in conflict with $\mathtt{x}$, or several such triangles.

First note that $\mathtt{x}_1$ and $\mathtt{x}_2$ both belong to **Star** : in fact, before the insertion of $\mathtt{x}$, in the triangulation containing $\mathtt{p}$, the edges $\mathtt{p}\mathtt{x}_1$ and $\mathtt{p}\mathtt{x}_2$ already existed (by definition, the insertion of $\mathtt{x}$ created $\mathtt{x}\mathtt{x}_1\mathtt{p}$ and $\mathtt{x}\mathtt{x}_2\mathtt{p}$) ; we know that the vertices adjacent to $\mathtt{p}$ lie on **Star**. Let U be the triangle of A adjacent to the edge following $\mathtt{x}_1$ on **Star** (remember that **Star** is oriented counterclockwise and $\mathtt{x}\mathtt{x}_1\mathtt{p}$ clockwise). U serves to distinguish between the two cases above. We have a direct access from $\mathtt{x}_1$ to U, via edge $\mathtt{x}\mathtt{x}_2$.

After the reinsertion of $\mathtt{x}$, the edges $\mathtt{x}\mathtt{x}_1$ and $\mathtt{x}\mathtt{x}_2$ will be on the boundary of the new set A of triangles in conflict with $\mathtt{p}$. So, if there are some vertices on the current boundary of A, between $\mathtt{x}_1$ and $\mathtt{x}_2$, the triangles adjacent to the edges of this chain of vertices must be in conflict with $\mathtt{x}$. U is such a particular triangle. Thus, if U is not in conflict with $\mathtt{x}$, $\mathtt{x}_1\mathtt{x}_2$ is an edge on the boundary of A, and consequently of U, and the first case occurs, otherwise the second case occurs.

First case, see Figure 6.6 :
In this case, the only way to fill the gap is to replace the removed triangles $\mathtt{x}\mathtt{x}_1\mathtt{p}$ and $\mathtt{x}\mathtt{x}_2\mathtt{p}$ around $\mathtt{x}$ by only one new triangle $\mathtt{x}\mathtt{x}_1\mathtt{x}_2$. The new triangle $\mathtt{x}\mathtt{x}_1\mathtt{x}_2$ has U as stepfather and the neighbor of U, which does not belong to A, as father. $\mathtt{x}$ points to edge $\mathtt{x}\mathtt{x}_2$ of **Star**, $\mathtt{x}_1$ points to edge $\mathtt{x}_1\mathtt{x}$. Both these edges point to triangle $\mathtt{x}\mathtt{x}_1\mathtt{x}_2$ of A. Details concerning other neighbor pointers can be found in the following pseudo code procedure :

Reinsertion - removed triangles - First case
create triangle $\mathtt{x}\mathtt{x}_1\mathtt{x}_2$,
with U as stepfather and the neighbor of U through $\mathtt{x}_1\mathtt{x}_2$ as father ;
update the neighbor through $\mathtt{x}_1\mathtt{x}_2$ of U to be $\mathtt{x}\mathtt{x}_1\mathtt{x}_2$;
find the neighbors at the creation of $\mathtt{x}\mathtt{x}_1\mathtt{x}_2$, by looking at those of $\mathtt{x}\mathtt{x}_1\mathtt{p}$ and $\mathtt{x}\mathtt{x}_2\mathtt{p}$;
update the special neighbor pointers of the neighbors of $\mathtt{x}\mathtt{x}_1\mathtt{x}_2$
and their neighbor at creation, and current neighbor, if necessary ;
throw $\mathtt{x}\mathtt{x}_1\mathtt{p}$ and $\mathtt{x}\mathtt{x}_2\mathtt{p}$ away in "garbage collector" ;
add edges $\mathtt{x}\mathtt{x}_1$ and $\mathtt{x}\mathtt{x}_2$ in **Star** ;
add pointers from $\mathtt{x}$ to edge $\mathtt{x}\mathtt{x}_2$, and from $\mathtt{x}_1$ to edge $\mathtt{x}_1\mathtt{x}$;
update **Star** by letting $\mathtt{x}\mathtt{x}_1$ and $\mathtt{x}\mathtt{x}_2$ be incident to $\mathtt{x}\mathtt{x}_1\mathtt{x}_2$;
put two **star** pointers from $\mathtt{x}\mathtt{x}_1\mathtt{x}_2$ to the elements $\mathtt{x}\mathtt{x}_1$ and $\mathtt{x}\mathtt{x}_2$ of **Star** ;
in **Created**, $\mathtt{x}\mathtt{x}_1\mathtt{x}_2$ is a triangle created by $\mathtt{x}$

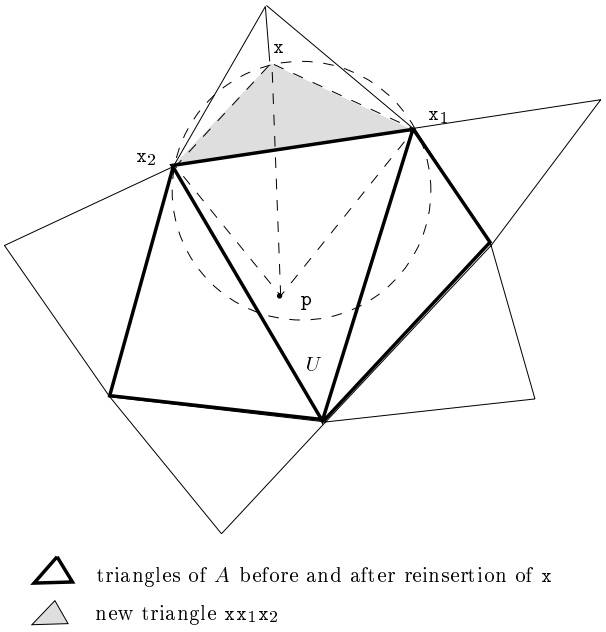


Figure 6.6 : Reinsertion - removed triangles - First case

Second case, see Figures 6.7 and 6.8 :
We know that U is in conflict with $\mathtt{x}$. We must find all the triangles in A in conflict with $\mathtt{x}$. Those triangles may be fathers for the nodes that will be created by $\mathtt{x}$. They form a connected subset of A, so they will be found owing to neighbor pointers in the following way :

Starting with U we visit the triangles in A incident at $\mathtt{x}_1$ in counterclockwise order until we reach a triangle not in conflict with $\mathtt{x}$. Let V denote the last such triangle in conflict with $\mathtt{x}$ and let e denote the edge of V at which the visit stops. Let $e = \mathtt{x}_1\mathtt{x}'$. We create triangle $W' = \mathtt{x}\mathtt{x}_1\mathtt{x}'$ and start this process again at vertex $\mathtt{x}'$ with V as starting triangle. When vertex $\mathtt{x}_2$ is reached, all the new triangles have been created. The iteration continues until vertex $\mathtt{x}_1$ is reached in order to mark the triangles of A killed by $\mathtt{x}$.

During the traversal, when a new triangle is created, we update the neighbor and **star** pointers of its neighbors. We also update its relevant vertex to point on the corresponding edge of **Star**. Once this is achieved, it remains to compute all kinds of neighborhood relations involving edges $\mathtt{x}\mathtt{x}_1$ and $\mathtt{x}\mathtt{x}_2$. Particularly, as the current neighbor of the just created triangle having edge $\mathtt{x}\mathtt{x}_1$, we take the neighbor of the now removed triangle $\mathtt{x}\mathtt{x}_1\mathtt{p}$ at its creation. The same holds for edge $\mathtt{x}\mathtt{x}_2$. The pseudo code procedure below formalizes these operations.

Reinsertion - removed triangles - Second case
    $V \leftarrow U$
    $\mathtt{s} \leftarrow \mathtt{x}_1$
    repeat
        while the neighbor of V sharing vertex s is in conflict with x
            { we turn around s counterclockwise }
            $V \leftarrow$ this neighbor ;
            the **killer** of V is x
        endwhile ;
        $e \leftarrow$ edge of V through which we stopped finding conflict ;
        create W' with edge e and vertex x ;
        W' is the son of V, and the stepson of the neighbor of V through e ;
        the neighbor at creation of W' through e is its stepfather ;
        if e is an edge of **Star**
            update the element of e in **Star** by replacing V by W' ;
            let the **star** pointer of W' reach e ;
            put a pointer from s to e ;
            update the special neighbor of the neighbor of V through e to be W'
                and the ordinary neighbor if necessary
        else

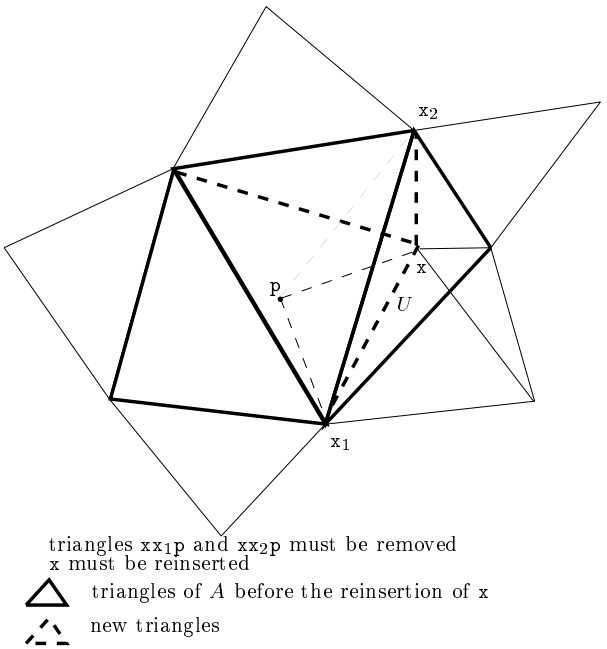


Figure 6.7 : Reinsertion - removed triangles - Second case

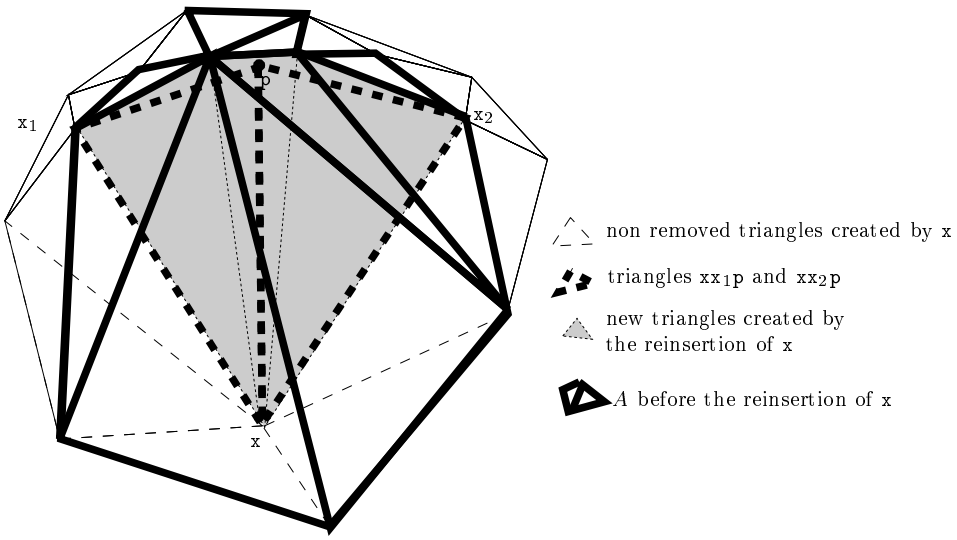


Figure 6.8 : A non trivial example for reinsertion

update the neighbor of the neighbor of V through e to be W'
endif ;
if $\mathtt{s} \neq \mathtt{x}_1$
W' and W are neighbors and neighbors at creation through edge $\mathtt{xs}$
endif ;
if $\mathtt{s} = \mathtt{x}_1$
$W_1 \leftarrow W'$;
$W \leftarrow$ neighbor at creation of $\mathtt{xx}_1\mathtt{p}$ through $\mathtt{xx}_1$;
W' and W are neighbors at creation through edge $\mathtt{xx}_1$;
the ordinary neighbor through $\mathtt{xx}_1$ of W_1 is W ;
the special neighbor through $\mathtt{xx}_1$ of W is W_1 ;
the ordinary neighbor through $\mathtt{xx}_1$ of W is W_1 if necessary
endif ;
$\mathtt{s} \leftarrow$ the other vertex of e ;
if $\mathtt{s} = \mathtt{x}_2$
$W_2 \leftarrow W'$;
$W \leftarrow$ neighbor at creation of $\mathtt{xx}_2\mathtt{p}$ through $\mathtt{xx}_2$;
W' and W are neighbors at creation through edge $\mathtt{xx}_2$;
the ordinary neighbor through $\mathtt{xx}_2$ of W_2 is W ;
the special neighbor through $\mathtt{xx}_2$ of W is W_2 ;
the ordinary neighbor through $\mathtt{xx}_2$ of W is W_2 if necessary
endif ;
$W \leftarrow W'$
{ W must be stored for future neighborhood relations }
until $\mathtt{s} = \mathtt{x}_2$;
store for example W_1 as created by x in **Created** ;
throw $\mathtt{xx}_1\mathtt{p}$ and $\mathtt{xx}_2\mathtt{p}$ away in "garbage collector" ;
repeat
while the neighbor of V sharing vertex s is in conflict with p
{ we turn around s counterclockwise }
$V \leftarrow$ this neighbor ;
the **killer** of V is x
endwhile ;
$\mathtt{s} \leftarrow$ the third vertex of V
until $\mathtt{s} = \mathtt{x}_1$;
replace the polygonal chain of **Star** between $\mathtt{x}_1$ and $\mathtt{x}_2$
by the edges $\mathtt{x}_1\mathtt{x}$ and $\mathtt{xx}_2$, associated with W_1 and W_2 ;
the **star** pointers of W_1 and W_2 reach respectively $\mathtt{xx}_1$ and $\mathtt{xx}_2$;

6.1.4 Analysis

We assume that p is a random site in $\mathcal{S}$, i.e. p is any of the precedingly inserted sites, with the same probability and independently from the insertion order. More precisely, an event is now one of the $n!$ permutations and one of the n sites. Each event occurs with the same probability $\frac{1}{n.n!}$.

Lemma 6.1 *The expected number of removed nodes is constant.*

Proof. Since p is chosen independently from the insertion order, the expected number of removed nodes is

$$\begin{aligned} & \sum_{T \text{ triangle}} Prob(p \text{ vertex of } T) Prob(T \text{ exists in the Delaunay Tree}) \\ = \ & \frac{3}{n} \times \text{ expected number of vertices of the Delaunay Tree} \\ = \ & O(1) \end{aligned}$$

□

Lemma 6.2 *The expected number of unhooked nodes is constant.*

Proof. In fact the number of unhooked nodes is bounded by the number of edges disappearing in the Delaunay tree, which is :

$$\begin{aligned} & \sum_{T,S \text{ adjacent triangles}} Prob(p \text{ vertex of } T \text{ or } S) \\ & \qquad \times Prob(TS \text{ is an edge of the Delaunay Tree}) \\ = \ & \frac{4}{n} \times \text{ expected number of edges of the Delaunay Tree} \\ = \ & O(1) \end{aligned}$$

□

Lemma 6.3 *The expected number of nodes created by the deletion of* p *is constant.*

Proof. The number of created triangles during the deletion of p is

$$\sum_{T \text{ triangle}} Prob(T \text{ appears during the deletion of p})$$

A triangle T of width j will appear *iff* p is one of the j sites in conflict with T, and p and the 3 vertices of T are introduced before the $j-1$ other sites in conflict with T, and p is not inserted after the 3 vertices of T. So the probability that T appears is :

$$\frac{j}{n} 3 \frac{3!(j-1)!}{(j+3)!} = \frac{3}{n} \frac{3!j!}{(j+3)!}$$

Hence by virtue of the proof of Lemma 4.1, the expected number of triangles created by the deletion of p is :

$$\frac{3}{n} \sum_{T} Prob(T \text{ appears during the insertion phase}) = O(1)$$

□

Lemma 6.4 *The expected cost of a deletion is* $O(\log\log n)$.

Proof. The expected number of triangles killed by p is constant using Lemma 6.1 (which also implies that the initialization of **Star** is achieved in constant time), and the traversal that is done during the Search step visits a constant number of nodes by Lemma 6.2. For each node, we must locate the creator of the node in **Reinsert**, which can be done in $O(\log\log n)$ worst case deterministic time, by using a bounded ordered dictionary [vEBKZ77]. The universe for this dictionary is the insertion age of the points, or in other words the number of the sites. The required finiteness of the universe can be circumvented using standard dynamization techniques, see for example [Ove83, section5.2].

To preserve the simplicity of the auxiliary data structures we can use a simple balanced binary search tree [AHU83]. In this way we achieve a complexity of $O(\log n)$ time. During the reinsertion phase **Star** can be updated in time proportional to the number of removed nodes.

The total cost of the work on unhooked triangles is constant, since we only have to reach the neighbor of the parent of each of them, and by Lemma 6.2.

For the triangles deleted by the reinsertion of x, the cost is linear in the number of triangles in conflict with both p and x, which is linear in the number of triangles created by the reinsertion of x. By Lemma 6.3, this expected cost is thus constant.

The expected whole cost is then less than $O(\log\log n)$. □

It is important to notice that the randomized hypothesis is preserved by a deletion. Namely, consider now the permutation σ of $\mathcal{S}\setminus\{\mathtt{p}\}$ obtained by removing $\mathtt{p}$ from the insertion order ; there are n permutations of $\mathcal{S}$ which give the same σ, so the probability that σ occurs is $n \times \frac{1}{n!}$ and σ is really a random permutation of $\mathcal{S} \setminus \{\mathtt{p}\}$. The randomization of the $n-1$ currently present sites is actual and the deletion of $\mathtt{p}$ does not affect the analysis of further insertions or deletions. Thus Lemmas 4.1, 4.2 and 6.4 yield the following theorem :

Theorem 6.5 *The Delaunay triangulation (or the Voronoi diagram) of a set $\mathcal{S}$ of n sites in the plane can be dynamically maintained in $O(\log n)$ expected time to insert or locate a point and $O(\log\log n)$ expected time to delete a point. This result holds provided that, at any time, the order of insertion on the sites remaining in $\mathcal{S}$ may be each order with the same probability, and when a site is deleted, it may be any site with the same probability.*

It is possible to avoid the hypothesis that the random deleted site and the random insertion permutation are independent. It is clear that the deletion of the first inserted site is more expensive that the deletion of the last one, but we show in the sequel that even the deletion of the first inserted site can be done with a good complexity. For this other kind of analysis of deletions, the probability space is only the set of permutations for the insertion, each being equally likely to occur.

Lemma 6.6 *The expected number of removed, unhooked and created nodes during the deletion of the first inserted site is $O(\log n)$.*

Proof. We here only give the proof for the removed nodes, the other quantities can be obtained in the same way. A triangle T of width j exists in the Delaunay tree and is removed during the deletion of the first inserted site if the first site is a vertex of T and if the two other vertices of T are inserted before the j sites in conflict with T. This happens with probability $\frac{3}{n}\frac{2!j!}{(j+2)!}$. By summing for all triangles T, the number of nodes removed in the Delaunay Tree is :

$$\sum_{j=0}^{n-3} |\mathcal{F}_j(\mathcal{S})| \frac{3}{n}\frac{2!j!}{(2+j)!} = O(\log n)$$

□

The same result holds for the k^{th} site : if we do not consider the first site but the k^{th} site, the probability that a triangle is removed during the deletion of the k^{th} site is clearly less than $\frac{3}{n}\frac{2!j!}{(j+2)!}$.

Therefore, the proof of Lemma 6.4 can be modified in a straightforward manner to obtain the following result :

Theorem 6.7 *The expected cost of deleting the k^{th} site is $O(\log n \log\log n)$.*

Thus, for any deletion sequence, the whole set of sites can be deleted in expected time $O(n \log n \log\log n)$, where the expectation is only on the insertion sequence.

6.1.5 d-dimensional case

When the dimension is greater than 2, the general scheme of the algorithm is the same. The difference lies in the fact that the simplices of A form now a polytope of dimension d, that is star-shaped from `p`. A can still be represented by its boundary. In order to be able to build all the necessary information concerning adjacency on the boundary of A, we must maintain the whole incidence graph of the Delaunay triangulation. The complexity does not increase, though the constants are of course higher.

Though the details are more involved than in the planar case, the analysis extends without any problem, providing an expected cost of $O\left(n^{\lfloor \frac{d+1}{2} \rfloor - 1} \log\log n\right)$ for the deletion of a given site.

6.1.6 Practical results in the planar case

The algorithm described here has been effectively coded. This section presents practical results. The sites are first inserted in a random order, and afterwards they are all deleted in another random order. Figures 6.9, 6.10 and 6.11 show the size of the Delaunay Tree in bold line, the size of the Delaunay triangulation in dashed line, and in thin line, a measure of the complexity of the operation. For insertions, it is the number of visited nodes, for deletions it is the number of unhooked triangles plus the numbers of triangles created during this deletion plus the cost of each location in structure **Reinsert**. The cost of deleting a site has a higher variance than the cost of inserting a site ; it may be important if the site had been inserted at the beginning of the construction, but this happens with a low probability.

Even in the case of sites lying on a parabola, which gives a bad behaviour for most of the existing algorithms, our algorithm has a good behaviour in practice.

The Delaunay triangulation of 15000 random sites in a square has been computed in 35 seconds on a Sun 4/75 and the deletion phase has been computed in 50 seconds.

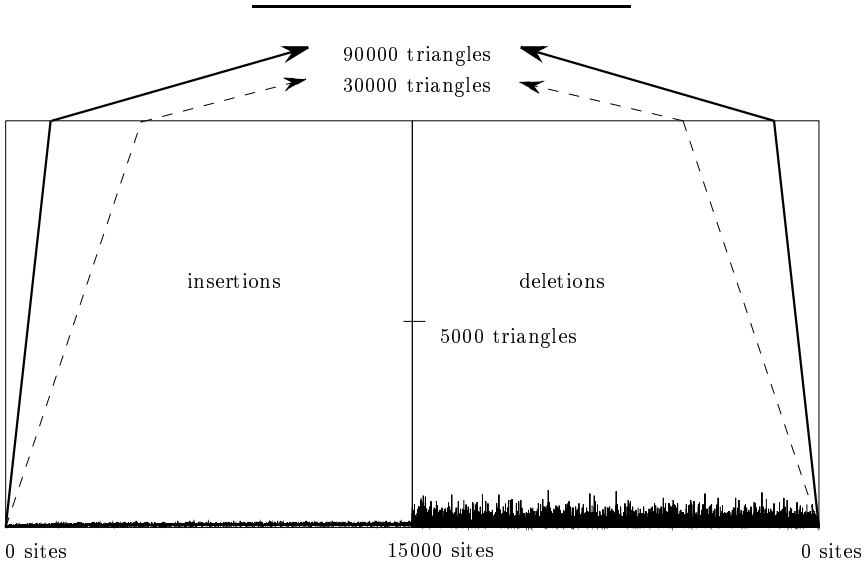


Figure 6.9 : Statistics on 15000 random sites in a square

6.2 Removing a segment from an arrangement

This section is roughly similar to the preceding one. We need however to develop some different details (see also [DTY91]).

We assume that the I-DAG has already been constructed as in Section 4.3.2. We need to introduce an auxiliary kind of entity, consisting of the *corners* of trapezoids. When we write that a trapezoid T has at most 4 corners, we now mean that the node of the I-DAG corresponding to T has pointers to those corners. This corners are crucial in the removing process, as will be seen in the sequel. It is not difficult to maintain them, during the insertion step : in Procedure Insert (Figure 4.10) we only have to deduce the corners of the children of T from the corners of T, and to create new corners.

Let us now detail the removing of a segment `s`. As for Delaunay triangulations,

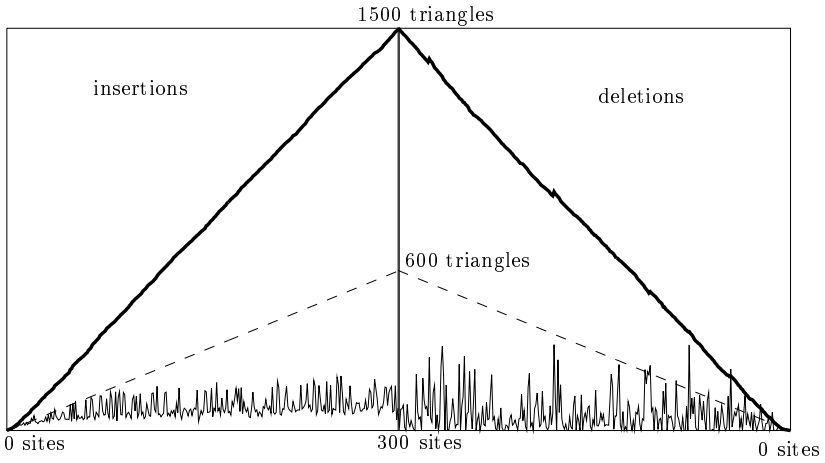


Figure 6.10 : Statistics on 300 random sites on an ellipse

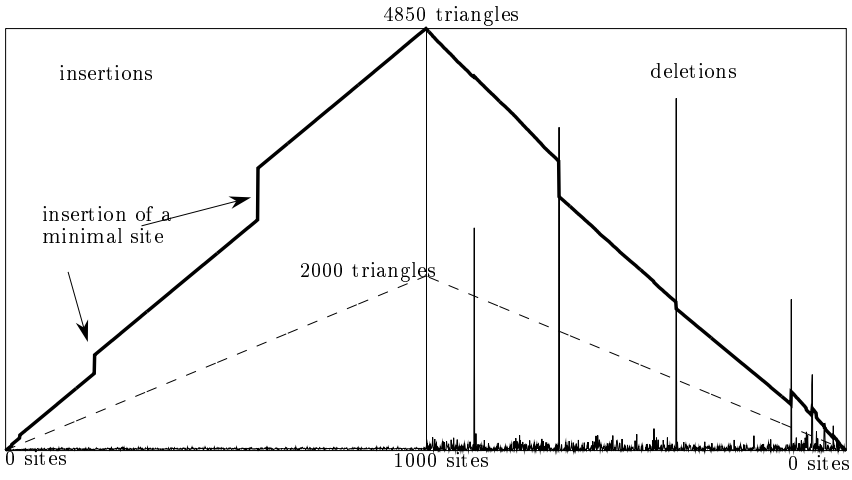


Figure 6.11 : Statistics on 1000 random sites on a parabola

we distinguish the unhooked and the removed nodes.

The first part of the removing of `s` will consist in finding all removed nodes. Then we must reinsert the creators of these nodes, while maintaining at each step the set A of trapezoids in conflict with `s`. The unhooked nodes will be processed while processing the removed nodes : the influence range of a trapezoid is included in the influence ranges of its children. So, a removed node cannot have only unhooked nodes. Therefore, an unhooked node must have a removed sibling, and it will be processed in the same time.

The initialization of A is the same as in the preceding section.

6.2.1 The Search step

We do not really look for whole segments to be reinserted. We only look for all parts of them to be reinserted, without trying to connect them. More precisely, what we need to reinsert segments is a list of triplets $(\mathtt{x}, T, F)$ consisting of a segment to be reinserted, a node T to be removed, and one parent F of T also to be removed. The reinsertion of such a triplet takes in account only the part of `x` on the boundary of $T \cap F$ (which can be reduced to an extremity of `x`).

We obtain this list by traversing the Influence Graph, starting from the trapezoids in A. Each time we find a node F to be removed, we look at its children, and for each child T defined by `s` and created by the insertion of a segment `x`, we add $(\mathtt{x}, T, F)$ to the list.

Furthermore, the segments need to be reinserted in the same order as they had been inserted first, so we need to sort these triplets (the ordering is along `x` only). This can be done in $O(\log\log n)$ time worst case deterministic time as in the preceding section.

6.2.2 Corners and bridges

When we reinsert a segment, we must first of all find the trapezoids in the current set A that are in conflict with it. So we need a location structure for A.

By definition, A is the set of trapezoids in conflict with the segment removed `s` at some stage of the history of the construction. So, the segment `s` crosses all trapezoids in A.

Notice that a removed node is defined by `s`, so it has at least one corner on `s` (except if it is defined by a vertex of `s`, this case is not difficult and can be solved easily).

The general idea is, when we process a triplet $(\mathtt{x}, T, F)$, to use the corners of T or F to locate in A the part of $\mathtt{x}$ defining T (that is an approximate idea, the details are given below).

So, rather than maintaining the set A, we maintain a doubly linked list of *bridges* allowing to deduce the trapezoids of A in conflict with $\mathtt{x}$ the corners of T, F or theirs suitable neighbors. The bridges are some corners that appear on $\mathtt{s}$. These bridges are ordered by the order in which they appear on $\mathtt{s}$. More precisely, a corner of a trapezoid is a bridge, if and only if it is on the common boundary of two trapezoids of A, and we store both of them in it (see Figure 6.12). This list of bridges is the analogous of the structure **Star** that we used for the Delaunay triangulation.

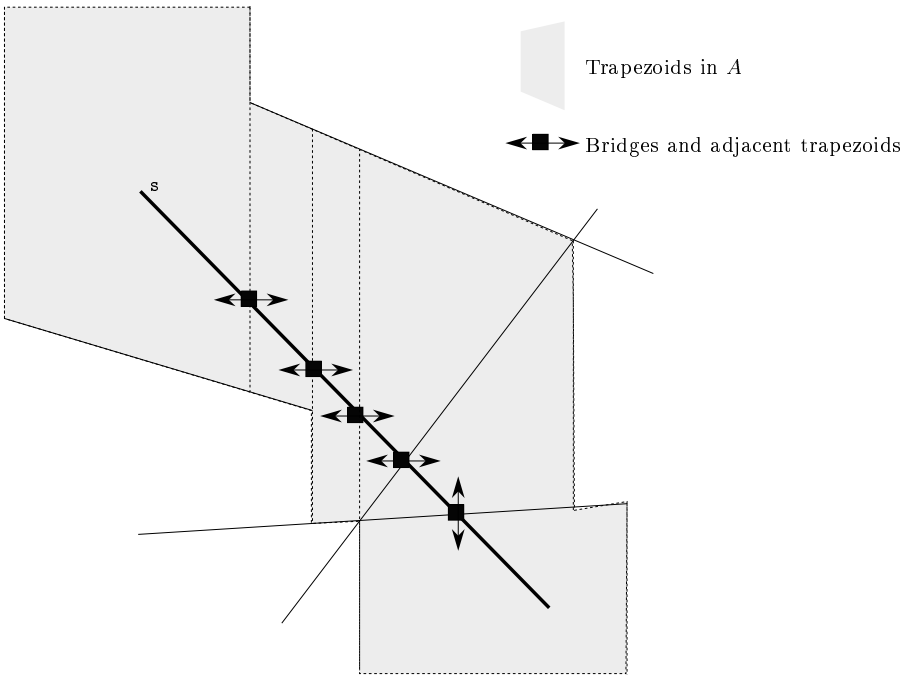


Figure 6.12 : The bridges along $\mathtt{s}$

In order to initialize the list of bridges, we traverse the set A consisting of all nodes killed by $\mathtt{s}$, by using neighborhood relations. For each node T in A, and each of its children, the corners c of this child that lie on $\mathtt{s}$ now appear as bridges

on s. T is one of the two trapezoids in A to be stored in c.

6.2.3 Reinsertion of a triplet $(\mathtt{x}, T, F)$

We now describe how a triplet $(\mathtt{x}, T, F)$ is reinserted. These such triplets are processed in the order used for the first insertion of segments, the order for triplets having the same x is indifferent.

Let us recall that in the triplet $(\mathtt{x}, T, F)$, x is a segment, T is a removed node created by x and F is a removed parent of T. At the current stage of the reinsertion, the creator of F is either s or has already been reinserted (because this creator is before x in the insertion order).

The removed node T has necessarily at least one corner c on the segment s that we delete, we first look for a bridge b *close* to this corner, which means that there is no other bridge on s between b and c. Such a bridge b points to a trapezoid in A in conflict with x. The main problem is to find b using suitable parent and neighbor pointers. To find b we have to look for trapezoid adjacent to s in the old trapezoidal map, trapezoids above and below give two candidate bridges (b_1 and b_2) and b is the closest bridge to c among b_1 and b_2.

Let us assume without loss of generality that c is the down left corner of T. There are several cases to examine.

(1) If the down left corner of F is c then c is already a bridge, so $c = b$ (see Figure 6.13).

(2) If c is not the intersection $\mathtt{s} \cap x$, then s is necessarily the down side of T (otherwise, we are in case 1). b_1 is the down left corner of F. b_2 is the left up corner of the down neighbor at creation N of T (see Figure 6.14). b is the closest bridge to c among b_1 and b_2.

(3) If $c = p \cap x$, as in the preceding case b_1 is the down left corner of F. Then, we determine the trapezoid N adjacent to c below s. The down side of T can be s (trapezoid T' in Figure 6.15) or can be x (trapezoid T in Figure 6.15). In the first case N is the down neighbor at creation of T and in the second case N is the down neighbor of the down neighbor at creation of T. N cannot be used directly to determine b_2 since the corners the creator of N is also x and it is not possible to ensure that the corner of N is already a bridge. So we use for b_2 the left up corner of N', parent of N covering the corner $c = x \cap p$ of N. b is the closest bridge to c among b_1 and b_2.

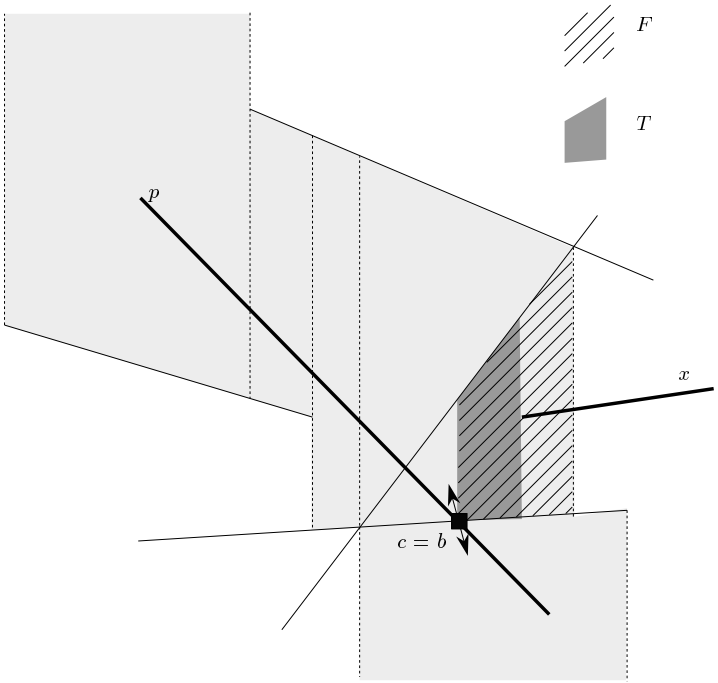


Figure 6.13 : Bridge b is c

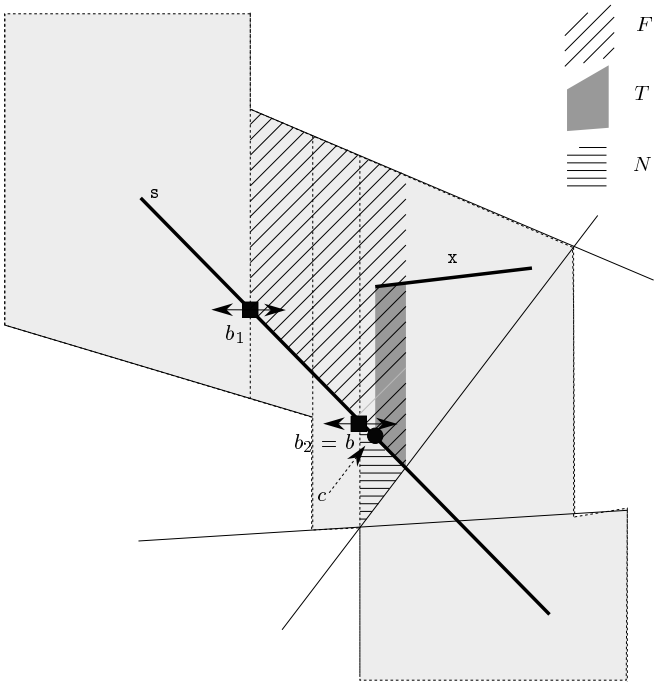


Figure 6.14 : Bridge b is the closest to c among b_1 and b_2

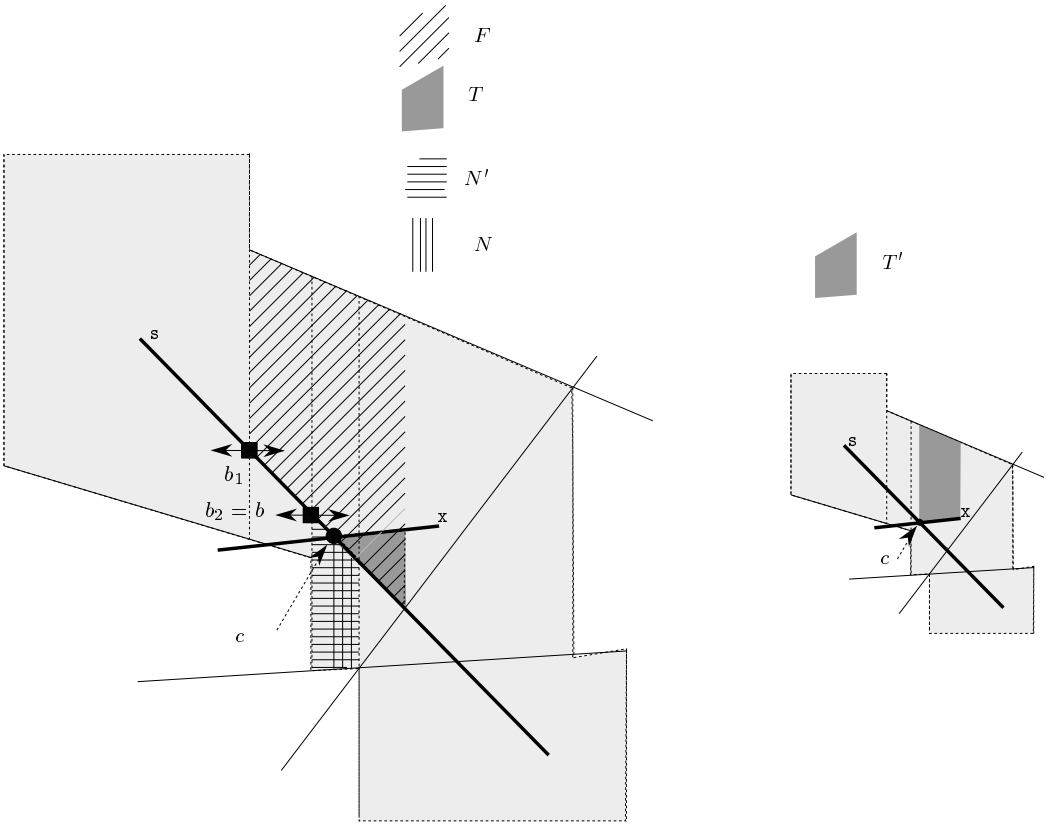


Figure 6.15 : Bridge b is the closest to c among b_1 and b_2

From b we have a direct access to the two trapezoids of A covering b, let S be the one intersecting T. This trapezoid is in conflict with `x`, so we update the graph in the following way : first `x` is the (new) killer of S, then `x` splits S in pieces, these pieces are children of S and some of them that are in conflict with `s` must be created now and added to A. The children of S which are not in conflict with `s` already exist in the graph and have not to be created, but they are unhooked and must be hung up to S ; these nodes can be found among the children of F. Figure 6.16 describes these operations in the case of Figure 6.13. We must also update the list of bridges, the neighborhood relations and possibly merge some newly created nodes.

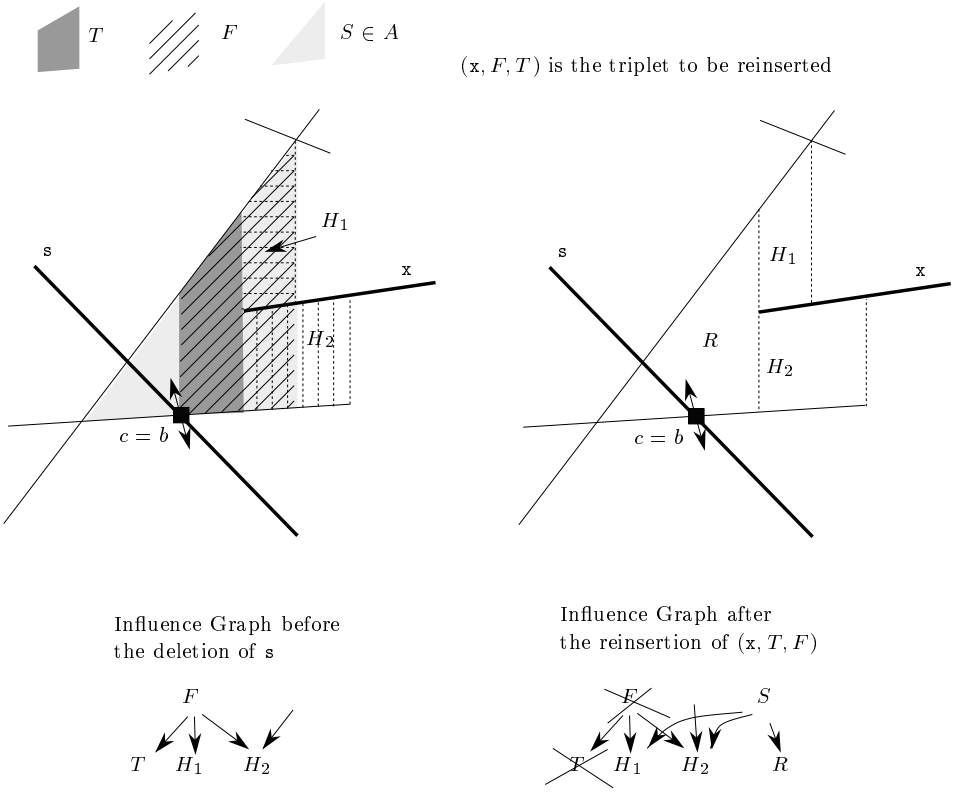


Figure 6.16 : Computing the new trapezoid R and hanging up the unhooked nodes H_1 and H_2

The following pseudo-code procedure summarizes the reinsertion of a triplet.

Reinsert(x, T, F)

{wlog, the corner c of T lying on s is its down left corner}
{Looking for the closest bridge b to c :}
    if c is already a bridge then $b = c$
    else b_1 = left down corner of F ;
        if $c \neq x \cap p$ then
            b_2 = left up corner of the down neighbor at creation of T
        else {the down neighbor of T is a sibling of T, so its corners are not bridges yet}
            {in this case we have intersecting points,
            so the stored parent is not any one of the parents (see Section 4.3.2)}
            b_2 = left up corner of the parent of the down neighbor of T
        endif ;
        b = closest bridge to c between b_1 and b_2
    endif ;

    from b, determine the trapezoid S in A which intersects T,
        it is in conflict with x ;
    x is the killer of S ;
    split S with x ;
    for each child R of S
        if R is in conflict with s {R must be added to A}
            if the corner c of T becomes a new bridge (i.e. $b \neq c$)
                create this bridge with R as one of its two trapezoids
                and find its other trapezoid, which is another child of S
                and find its other trapezoid, which is another child of S
                insert c in the doubly linked list of bridges as a successor of b
            else {$b = c$}
                update b : R is one of its two trapezoids
            endif ;
        else {R must be already existing as an unhooked node}
            find R among the children of F and hang it up to S
        endif ;
        update all other fields in R, and neighbor relations, as for a usual insertion ;
        look for a possible merge with neighbors as for an insertion
    endfor.

6.3 Analysis

The extra work to analyze the deletion of a segment is extremely simple. n is the number of segments currently present in the structure, a is the number of intersection points between these n line segments. Let us first recall the result for insertion proved in Section 4.3.2 : The expected size of the Influence Graph is $O(n+a)$, the expected number of visited nodes during the insertion of the last segment in the Influence Graph is $O(\log n + \frac{a}{n})$, the cost of locating a query point in the trapezoidal map is $O(\log n)$. This cost is expected over the randomization of the segments (not on the query point).

We now deal with the analysis of the deletion phase.

Lemma 6.8 *The expected number of removed nodes is* $O(1+\frac{a}{n})$.

Proof. If the deleted segment `s` is randomly chosen among the n segments present in the arrangement, the expected number of removed nodes is

$$\begin{aligned} & \sum_{T \text{ trapezoid}} Prob(p \text{ defines } T)\, Prob(T \text{ exists in the Influence Graph}) \\ \leq\ & \frac{4}{n} \times \text{ expected number of nodes of the Influence Graph} \\ =\ & O(1+\frac{a}{n}) \end{aligned}$$

□

Since a node has at most four sons, there are at most four triplets $(\mathtt{x}, T, F)$ with the same removed node F, thus the above bound apply also to the number of triplets processed by the algorithm. These triplets must be sorted according to the age of the insertion of the creator segment, this is done in $O((1+\frac{a}{n})\log\log n)$ expected time using a bounded ordered dictionary (or in $O((1+\frac{a}{n})\log n)$ expected time using a simpler priority queue).The reinsertion of a triplet involves a constant number of operations, so the expected time of deleting a random segment is $O((1+\frac{a}{n})\log\log n)$.

As the Influence Graph is restored as if `s` was never inserted, the results stated for the insertion step are still valid after a segment has been removed, and we obtain the main result of this section :

Theorem 6.9 *An arrangement of line segments in the plane can be maintained dynamically in* $O(\log n + \frac{a}{n})$ *time for an insertion and* $O((1+\frac{a}{n})\log\log n)$ *time*

for a deletion. The space complexity is $O(n + a)$. All complexities are expected with respect to the randomized insertion of the line segments. Here n denotes the current number of segments present in the arrangement, and a denotes the current complexity of the arrangement.
The expected cost of the location of any point in the arrangement is $O(\log n)$, where the expectation is over the randomization of the order of insertion of the line segments present in the arrangement, these queries can be done persistently with respect to the insertions.

Conclusion

The common points between these two examples seem to reside only in the general idea of reconstructing the past. However, we can also see similarities in the analysis.

Other approaches appeared recently in the literature, they will be rapidly presented in Chapter 7.

Chapitre VII

Travaux parallèles

Dans ce chapitre, sont présentées rapidement des articles dont la publication est pour la plupart très récente ou même à venir, de façon à replacer nos travaux au milieu de ceux de la communauté.

Seules les idées générales des algorithmes sont données, sans aucun détail.

7.1 Algorithmes statiques accélérés

Le mariage entre Graphe de Conflits et Graphe d'Influence permet d'accélérer les algorithmes randomisés, sous certaines hypothèses.

R. Seidel a utilisé le premier une telle combinaison, pour le calcul de la carte des trapèzes induite par une chaîne polygonale simple de taille n [Sei91b]. Il en déduit un algorithme de complexité moyenne $O(n \log^\star n)$ (plus généralement, $O(n \log^\star n + k \log n)$ lorsque les segments forment un graphe planaire ayant k composantes connexes).

O. Devillers [Dev] a généralisé ce résultat à d'autres cas, où l'on possède aussi une information de même type (un certain ordre de parcours des segments) sur les données : par exemple le cas du calcul du squelette d'un polygone simple, ou de la triangulation de Delaunay d'un ensemble de points dont on connaît l'arbre couvrant de longueur minimale.

Rappelons que, pour $n > 0$, $\log^\star n$ est le plus grand entier l tel que l'on ait $\log^{(l)} n \leq 1$, $x \mapsto \log^{(l)} x$ étant l'application obtenue par l itérations de la fonction $x \mapsto \log x$. C'est une fonction qui croît extrêmement lentement ($\log^\star n < 5$ pour $n < 2^{65536}$).

L'idée de l'algorithme est la suivante : Soit $\mathtt{s}_1, \mathtt{s}_2, \ldots, \mathtt{s}_n$ une permutation aléatoire parmi les $n!$ permutations de l'ensemble $\mathcal{S}$ de segments. On initialise le processus en calculant la carte des trapèzes $\mathcal{T}_1$ induite par la premier segment $\mathtt{s}_1$. On réalise ensuite h étapes qui consistent, à chaque étape $i \geq 2$, à :

- Insérer $N_i - N_{i-1}$ segments dans la carte des trapèzes $\mathcal{T}_{i-1}$, en construisant un graphe d'influence. On obtient une carte des trapèzes $\mathcal{T}_i$.
- Utiliser l'ordre de parcours donné par les informations connues sur $\mathcal{S}$ (par exemple, les segtments forment un polygone simple), pour construire le graphe de conflits entre les $n - N_i$ segments non encore insérés et les trapèzes de $\mathcal{T}_i$. Ceci est effectué en parcourant $\mathcal{T}_i$ grâce aux relations de voisinage.

Lorsque l'on insère un nouveau segment, au cours de l'étape suivante $i + 1$, on utilise le graphe de conflits des trapèzes de $\mathcal{T}_i$ pour trouver les conflits dans cette

carte de trapèzes, puis on en déduit les conflits avec les trapèzes plus récents en traversant uniquement la portion du graphe d'influence que l'on est en train de construire.

Ceci permet d'utiliser le point *(3)* du théorème 4.9. Un choix judicieux des paramètres ($h = \log^\star n$ et $N_i = \left\lfloor \frac{n}{\log^{(i)} n} \right\rfloor$) permet d'obtenir la complexité annoncée.

7.2 Algorithmes semi-dynamiques

Calcul d'une cellule dans un arrangement de segments

Dans [CEG*91], B. Chazelle, H. Edelsbrunner, L. Guibas, M. Sharir et J. Snoeyink calculent une cellule précise dans un arrangement de n segments. L'algorithme a une complexité randomisée de $O(n\alpha(n)\log n)$, alors que le meilleur algorithme déterministe connu prenait un temps $O(n\alpha(n)\log^2 n)$ dans le pire des cas. $\alpha(n)$ désigne la pseudo-inverse de la fonction d'Ackermann ; α croît vers l'infini, mais excessivement lentement, et reste inférieure à 3 pour toute valeur «raisonnable» (plus petite que $2^{2^{2^{\cdot^{\cdot^{\cdot^{2^{65536}}}}}}}$) de n. Rappelons que la taille d'une cellule est $\Omega(n\alpha(n))$ dans le pire des cas.

L'algorithme est semi-dynamique, et fonctionne grâce au maintien de l'histoire de la construction de la carte des trapèzes induite par les segments, dans un graphe analogue au I-DAG. Afin de déterminer si un nouveau segment coupe le bord de la cellule cherchée, les composantes connexes de son bord sont stockées dans une structure classique autorisant les opérations *union* et *find*, qui sont nécessaires pour la mise à jour de la carte des trapèzes, lors de l'insertion d'un nouveau segment.

7.3 Algorithmes dynamiques

Plusieurs auteurs se sont récemment intéressés à la possibilité de suppressions dans les structures randomisées.

7.3.1 Avec stockage de l'histoire

Des approches générales

O. Schwarzkopf [Sch91] construit une histoire plus large de la structure, en stockant non seulement les insertions, comme nous le faisons, mais aussi les suppressions. De cette façon, il obtient une structure qui prend une place plus importante en mémoire. Il doit donc la reconstruire, lorsque cette place est trop grande, en ne gardant que les objets qui ont été insérés mais pas encore détruits, à l'étape courante.

Dans le cas particulier du diagramme de Voronoï dans le plan, chaque opération d'insertion a un coût moyen de $O(\log n)$, et on peut effectuer des requêtes de localisation avec une complexité de $O(\log^2 n)$, avec grande probabilité.

Il est aussi possible de construire la carte des trapèzes d'un ensemble de n segments, ne se coupant pas, en temps moyen $O(\log^2 n)$ par mise à jour.

F. Aurenhammer et O. Schwarzkopf utilisent une approche similaire pour les diagrammes de Voronoï d'ordres supérieurs [AS91].

Dans un nouvel article, K. Mulmuley reprend le schéma général de O. Schwarzkopf [Mul91c]. La nuance est que sa structure n'est pas basée sur l'histoire réelle de la construction, mais sur une autre séquence, imaginaire, d'insertions et de suppressions. Lorsque cette histoire est trop importante, il en reconstruit une autre. Une autre alternative, plus puissante, est de rééquilibrer la structure, en effectuant des opérations similaires aux rotations dans les arbres binaires de recherche, toujours en manipulant ce passé imaginaire.

Enveloppes convexes

K.L. Clarkson, K. Mehlhorn et R. Seidel [CMS92] résolvent le problème pour l'enveloppe convexe, en dimension quelconque, en utilisant le même principe général que le nôtre : reconstruction d'un nouveau passé. Mais l'histoire de la construction est gardée dans l'enveloppe convexe elle-même : plus précisément, les auteurs maintiennent une triangulation de l'enveloppe convexe, qui leur permet de localiser le nouveau point à insérer.

Si le point est à l'intérieur de l'enveloppe convexe courante, c'est-à-dire qu'il appartient à l'un des simplexes de sa triangulation, on conserve l'information d'appartenance de ce point au simplexe. Si le point est à l'extérieur, on construit les simplexes formés par ce point et les faces visibles de l'enveloppe convexe, et on les y rajoute.

Cette triangulation permet également, lors de la suppression d'un sommet de l'enveloppe convexe, de trouver les points qui sont à l'intérieur, et qui sont susceptibles d'apparaître comme nouveaux sommets : ce sont des points intérieurs aux simplexes incidents à ce sommet. Ces points seront réinsérés. Lors de la suppression d'un sommet de la triangulation, intérieur à l'enveloppe convexe, il faut également réinsérer certains des points intérieurs aux simplexes incidents à ce sommet, afin de reconstruire l'histoire.

Une analyse randomisée fournit une complexité moyenne de $O(\log n)$ en dimension ≤ 3, et $O\left(n^{\lfloor \frac{d}{2} \rfloor - 1}\right)$ en dimension supérieure, pour chaque opération.

7.3.2 Sans stockage de l'histoire

K. Mulmuley [Mul91b, MS91] utilise une approche radicalement différente, et évite avec succès le stockage de l'histoire de la construction, en reprenant l'idée de [AS89].

Il construit une structure hiérarchique : à l'étage 1, tous les objets sont présents. Ensuite, pour tout $i \leq 2$, on détermine les objets présents à l'étage i en tirant à pile ou face pour chacun des objets présents à l'étage $i-1$. A chaque étage, la structure géométrique cherchée est calculée.

Entre deux étages, les conflits, entre une région déterminée par des objets de l'étage i, et les objets de l'étage $i-1$ qui ne sont pas à l'étage i, sont stockés dans un graphe de conflits. De plus, une structure *Descent* permet de déduire la localisation d'un objet à l'étage i, connaissant sa localisation à l'étage $i-1$.

Lorsqu'un nouvel objet est inséré, on lance la pièce et l'objet est inséré à chaque étage, jusqu'à ce que la pièce donne une réponse négative. Les structures entre étages doivent être mises à jour. La suppression d'un objet n'est plus ici que l'opération exactement inverse d'une insertion. La seule différence est qu'une insertion nécessite une étape de recherche de conflits, ce qui augmente la complexité.

Cette idée est appliquée à la construction dynamique de la triangulation de Delaunay dans le plan. Les complexités moyennes sont $O(\log n)$ pour une insertion, $O(1)$ pour une suppression, et $O(\log^2 n)$ pour une localisation. Dans le cas des arrangements de segments dans le plan, une insertion coûte $O(\log a)$ en moyenne, une suppression $O(1)$ en moyenne et une localisation $O(\log a)$ avec grande probabilité, où a est la taille de la carte des trapèzes. Les analyses utilisent les résultats de l'échantillonnage aléatoire.

7.4 Stratégies d'exécution

Des auteurs se sont dernièrement intéressés à des problèmes de nature différente. Si le temps de calcul t_A d'un algorithme A est donné par sa valeur moyenne, c'est à dire son espérance, l'inégalité de Markov donne la probabilité pour que le temps de calcul soit supérieur à une valeur donnée. C'est la meilleure borne connue dans le cas où l'on n'a pas d'information supplémentaire sur la distribution probabiliste de t_A.

Supposons que l'on réalise maintenant l'expérience suivante : on fait tourner A pendant un temps t_1, puis, si A ne s'est pas arrêté avant, on recommence pendant un temps t_2, et ainsi de suite. On obtient ainsi un algorithme modifié, et on s'intéresse à la recherche de la séquence optimale $t_1, t_2, \ldots$. H. Alt, L. Guibas, K. Mehlhorn, R. Karp et A. Widgerson démontrent, parmi d'autres résultats, qu'il existe toujours une stratégie optimale, si la valeur de l'espérance de t_A est connue [AGM*91].

K. Mehlhorn, M. Sharir et E. Welzl étudient un problème identique, mais s'intéressent à l'espace mémoire utilisé, et obtiennent des bornes très fines pour la probabilité que la place mémoire s'écarte de sa moyenne d'un facteur donné [MSW92].

Conclusion

Nous avons présenté ici les différentes tendances, concernant les structures de données et les algorithmes, dans le domaine des algorithmes dynamiques randomisés. Vu que les publications dans le domaine se bousculent, cette présentation s'arrête délibérement à tout ce dont j'ai eu connaissance ce jour fatidique du 14 octobre 1991 ! Nul doute que la matière inspirera encore les chercheurs dans l'avenir.

Conclusion

La boucle est-elle bouclée ? On peut le croire. Notre publication présentant l'Arbre de Delaunay [BT86] annonçait un temps moyen d'insertion en $O(\log n)$ par point, en dimension 2, ainsi qu'en dimension 3 dans le cas où la triangulation de Delaunay était de taille linéaire. Les démonstrations avaient le défaut majeur de ne pas poser clairement l'hypothèse de randomisation des points, hypothèse qui n'était pas courante à cette époque. Elle n'étaient évoquée que dans les constatations expérimentales : on remarquait que, pourvu que les points soient insérés dans un ordre aléatoire, des distributions fortement dégénérées étaient traitées avec la même efficacité que des distributions homogènes. Avec les connaissances actuelles, nous pouvons poser un regard plus clair sur ces démonstrations. On s'aperçoit alors que les arguments centraux n'étaient autres que ceux qui sont devenus habituels.

Précisons ceci, en s'arrêtant sur ces arguments. On suppose que l'on introduit n points selon une permutation ω quelconque parmi les $n!$ possibles. Limitons nous au cas du plan, puisque dans le cas de la dimension 3, lorsque la taille de la triangulation était linéaire, les mêmes démonstrations restaient valables. La formule d'Euler nous permettait de majorer par 6 le degré moyen d'un point dans la triangulation de Delaunay, en divisant le nombre d'arêtes par le nombre de points. On en déduisait que le nombre moyen de triangles crées par l'insertion du $k^{ième}$ point $\circ$ était au plus 6. C'est exactement le même argument, qui est un argument d'analyse «en arrière», que l'on retrouve dans le lemme 4.6 : $\circ$ est l'un quelconque des k points présents après son insertion, puisque ω est une permutation quelconque. Il crée donc l'un des triangles présents à l'étape k avec probabilité $\frac{3}{k}$.

Un autre point crucial consistait à dire que, lorsque le $l^{ième}$ point $\circ$ était inséré, le nombre de triangles présents à l'étape k et qui sont en conflit avec $\circ$ était le le même que si $\circ$ était inséré comme $(k+1)^{ième}$ point. On reconnait là tout à fait l'idée du lemme 4.5, qui n'est qu'une moyenne sur toutes les permutations possibles.

On peut donc remarquer qu'il a fallu attendre [Dev] et l'analyse «en arrière» introduite par R. Seidel, pour établir nos démonstrations de 1986 dans toute leur rigueur. Entre temps, K.L. Clarkson a présenté la technique nouvelle de l'échantillonnage aléatoire, et nous l'avons suivi, comme la quasi-totalité de la communauté internationale. Il aurait peut-être été possible d'arriver directement aux analyses «en arrière».

Nos recherches ont démontré que l'approche suivie pour la conception de l'Arbre de Delaunay —approche consistant à mémoriser l'histoire de la construction

effectuée par l'algorithme— était bonne, puisqu'on a pu généraliser cette structure de plusieurs façons, en suivant le formalisme de K.L. Clarkson : le Graphe d'Influence permet de calculer des structures géométriques diverses, à condition qu'elles puissent être définies comme ensemble de régions sans conflits ; Le k-Arbre de Delaunay étend les possibilités au cas de régions ayant moins de k conflits. On a également prouvé que le Graphe d'Influence pouvait être rendu pleinement dynamique.

Une des qualités des algorithmes incrémentaux randomisés est leur simplicité, la littérature récente en fournit des preuves abondantes, les nôtres n'en sont qu'une illustration supplémentaire.

La comparaison des complexités des algorithmes utilisant le Graphe d'Influence, avec celles présentées dans les articles publiés parallèlement —de nombreux parmi eux ont repris la même approche, basée sur l'histoire— met en évidence l'efficacité théorique de cette structure. L'Arbre de Delaunay a été la première structure de données permettant la conception d'algorithmes géométriques semi-dynamiques efficaces, c'est également une des toutes premières structures dynamiques. La programmation de ces algorithmes a démontré également leur efficacité pratique.

Malgré l'efficacité des algorithmes randomisés, les recherches sur les algorithmes déterministes n'en restent pas moins ouvertes. B. Chazelle et J. Friedman, par exemple, se sont intéressés dès 1988 à la possibilité de «dérandomiser» des algorithmes basés sur l'échantillonnage aléatoire et le schéma de division-fusion [CF90]. Beaucoup de ces recherches n'ont qu'un intérêt théorique, car les algorithmes déterministes optimaux sont souvent très compliqués et difficiles à mettre en œuvre. Elles sont cependant utiles, car la découverte d'un tel algorithme, même inutilisable en pratique, montre au moins l'existence d'une solution, et ouvre la voie à la recherche d'autres solutions plus simples.

Dans [BT86], on posait la question de la recherche d'un ordre d'insertion des points dans l'Arbre de Delaunay, qui permettrait de garantir un coût maximum pour une localisation quelconque, alors que l'analyse randomisée ne garantit qu'un coût moyen. L'algorithme perdrait alors son caractère «en ligne», car la connaissance de l'ensemble des données serait nécessaire pour trouver cet ordre. C'est en se posant la même question que B. Chazelle vient de trouver un algorithme optimal pour le calcul de l'enveloppe convexe, en toutes dimensions [Cha91], problème qui était non résolu précédemment, bien que l'enveloppe convexe soit une

des structures les plus étudiées.

Les algorithmes les plus intéressants en pratique sont sans doute les algorithmes dont la complexité dépend de la taille du résultat, plus encore que les algorithmes optimaux, qui ne sont optimaux que dans le pire des cas.

On a en effet vu que les algorithmes incrémentaux randomisés sont sensibles, non pas à la taille du résultat final, mais à la taille de tous les résultats intermédiaires. Lorsque l'on étudie des structures dont la complexité croît par rapport au nombre des données, ceci est suffisant. C'est le cas par exemple pour la construction de la carte des trapèzes pour le calcul d'un arrangement de segments du plan.

Cependant, pour le problème de parties cachées, par exemple, cette approche ne peut pas donner de bons résultats : supposons qu'un gros objet cache une scène qui, elle, a une complexité quadratique. Cet objet sera inséré, avec probabilité $\frac{1}{2}$, parmi les $\frac{n}{2}$ derniers objets. De nombreux résultats intermédiaires sont donc de taille quadratique, alors que le résultat final est de taille constante. La conception d'algorithmes sensibles à la taille de la sortie est donc cruciale dans ce domaine. Citons par exemple le travail de M. de Berg et M. Overmars [dBO91], qui ont fourni une des premières solutions simples au problème.

Quelques algorithmes déterministes sensibles à la taille du résultat, concernant les structures géométriques étudiées dans cette thèse, existent. La triangulation de Delaunay, dès la dimension 3, est un autre exemple qui pose le même problème : même si la triangulation finale est de taille linéaire, il peut y avoir eu des triangulations partielles de taille quadratique. Dans [Boi88, BCDT91], on obtient un algorithme de complexité optimale $O(n \log n + t)$ (n est le nombre de points, et t est le nombre de tétraèdres de la triangulation), mais dans le cas très restrictif où l'on suppose que les points appartiennent à deux plans. L'algorithme de R. Seidel [Sei86] pour le calcul de l'enveloppe convexe est en $O(n^2 + f \log n)$, en dimension quelconque, où f est le nombre de faces de l'enveloppe.

Trop peu de résultats existent dans ce domaine, et l'on peut supposer que ce sujet sera très étudié dans l'avenir.

Références bibliographiques

[AES85] D. Avis, H. ElGindy, and R. Seidel. Simple on-line algorithms for convex polygons. In G.T. Toussaint, editor, *Computational Geometry*, pages 23–42, North Holland, 1985.

[AGM*91] H. Alt, L. Guibas, K. Mehlhorn, R. Karp, and A. Widgerson. A method for obtaining randomized algorithms with small tail probabilities. 1991. In preparation.

[AGSS89] A. Aggarwal, L.J. Guibas, J. Saxe, and P.W. Shor. A linear time algorithm for computing the Voronoï diagram of a convex polygon. *Discrete and Computational Geometry*, 4:591–604, 1989.

[AHU83] A. Aho, J. Hopcroft, and J. Ullman. *The design and ananlysis of computer algorithms. Computer science and information processing*, Addison Wesley, 1983.

[AS89] C. Aragon and R. Seidel. Randomized search trees. In *IEEE Symposium on Foundations of Computer Science*, pages 540–545, 1989.

[AS91] F. Aurenhammer and O. Schwarzkopf. A simple on-line randomized incremental algorithm for computing higher order Voronoï diagrams. In *7th ACM Symposium on Computational Geometry in North Conway*, pages 142–151, June 1991. Full paper available as Technical Report B 91-02 Universität Berlin,to appear in IJCGA.

[Aur91] F. Aurenhammer. Voronoï diagrams — a survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23(3), September 1991.

[BCDT91] J.D. Boissonnat, A. Cérézo, O. Devillers, and M. Teillaud. Output sensitive construction of 3D Delaunay triangulation of constrained sets of points. In *Third Canadian Conference on Computational Geometry in Vancouver*, August 1991. Full paper available as Technical Report INRIA 1415.

[BD91] J-D. Boissonnat and K. Dobrindt. *Randomized construction of the upper envelope of surface patches in three dimensions.* Technical Report, Institut National de Recherche en Informatique et Automatique, (France), 1991. In preparation.

[BDS*] J.D. Boissonnat, O. Devillers, R. Schott, M. Teillaud, and M. Yvinec. Applications of random sampling to on-line algorithms in computational geometry. *Discrete and Computational Geometry.* To be published. Available as Technical Report INRIA 1285. Abstract published in IMACS 91 in Dublin.

[BDT] J.D. Boissonnat, O. Devillers, and M. Teillaud. A semi-dynamic construction of higher order Voronoï diagrams and its randomized analysis. *Algorithmica.* To be published. Available as Technical Report INRIA 1207. Abstract published in Second Canadian Conference on Computational Geometry 1990 in Ottawa.

[Boi88] J.D. Boissonnat. Shape reconstruction from planar cross-sections. *Computer Vision, Graphics, and Image Processing*, 44:1–29, 1988.

[Bow81] A. Bowyer. Computing Dirichlet tesselations. *The Computer Journal*, 24(2), 1981.

[Bro79] K.Q. Brown. Voronoï diagrams from convex hulls. *Information Processing Letters*, 9:223–228, 1979.

[BT] J.D. Boissonnat and M. Teillaud. On the randomized construction of the Delaunay tree. *Theoretical Computer Science.* To be published. Available as Technical Report INRIA 1140.

[BT86] J.D. Boissonnat and M. Teillaud. A hierarchical representation of objects: the Delaunay Tree. In *Second ACM Symposium on Computational Geometry in Yorktown Heights*, June 1986.

[CE85] B. Chazelle and H. Edelsbrunner. An improved algorithm for constructing k^{th}-order Voronoï diagrams. In *First ACM Symposium on Computational Geometry in Baltimore*, pages 228–234, June 1985.

[CE88] B. Chazelle and H. Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. In *IEEE Symposium on Foundations of Computer Science*, pages 590–600, 1988.

[CEG*90] K.L. Clarkson, H. Edelsbrunner, L.J. Guibas, M. Sharir, and E. Welzl. Combinatorial complexity bounds for arrangements of curves and surfaces. *Discrete and Computational Geometry*, 5:99–160, 1990.

[CEG*91] B. Chazelle, H. Edelsbrunner, L.J. Guibas, M. Sharir, and J. Snoeyink. Computing a face in an arrangement of line segments. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 441–448, 1991.

[CF90] B. Chazelle and J. Friedman. A deterministic view of random sampling and its use in geometry. *Combinatorica*, 10:229–249, 1990.

[Cha91] B. Chazelle. *An optimal convex hull algorithm for point sets in any fixed dimension.* Technical Report CS-TR-336-91, Computer Science Department, Princeton University, (USA), June 1991.

[Cla85] K.L. Clarkson. A probabilistic algorithm for the post office problem. In *17th Annual SIGACT Symposium*, pages 75–184, 1985.

[Cla87] K.L. Clarkson. New applications of random sampling in computational geometry. *Discrete and Computational Geometry*, 2:195–222, 1987.

[CMS92] K.L. Clarkson, K. Mehlhorn, and R. Seidel. Four results on randomized incremental constructions. In *STACS 92*, Springer-Verlag, 1992.

[CS88] K.L. Clarkson and P.W. Shor. Algorithms for diametral pairs and convex hulls that are optimal, randomized, and incremental. In *4th ACM Symposium on Computational Geometry in Urbana*, pages 12–19, 1988.

[CS89] K.L. Clarkson and P.W. Shor. Applications of random sampling in computational geometry, II. *Discrete and Computational Geometry*, 4(5), 1989.

[dBO91] M. de Berg and M.H. Overmars. Hidden surface removal for axis-parallel polyhedra. In *IEEE Symposium on Foundations of Computer Science*, pages 252–261, 1991.

[Dev] O. Devillers. Randomization yields simple $o(n \log^\star n)$ algorithms for difficult $\omega(n)$ problems. *International Journal of Computational Geometry and Applications.* To be published. Full paper available as Technical Report INRIA 1412. Abstract published in the Third Canadian Conference on Computational Geometry 1991 in Vancouver.

[DMT] O. Devillers, S. Meiser, and M. Teillaud. Fully dynamic Delaunay triangulation in logarithmic expected time per operation. *Computational Geometry Theory and Applications.* To be published. Available

as Technical Report INRIA 1349. Abstract published in LNCS 519 (WADS'91, august 1991).

[DMT92] O. Devillers, S. Meiser, and M. Teillaud. *An universal framework for the geometric transforms involving generalized Voronoï diagrams.* Technical Report, Institut National de Recherche en Informatique et Automatique, (France), 1992. In preparation.

[DTY91] O. Devillers, M. Teillaud, and M. Yvinec. *Dynamic location in an arrangement of line segments in the plane.* Technical Report 1558, Institut National de Recherche en Informatique et Automatique, (France), November 1991.

[Dwy91] R.A. Dwyer. Higher-dimensional Voronoï diagrams in linear expected time. *Discrete and Computational Geometry*, 6:343–367, 1991.

[Ede87] H. Edelsbrunner. *Algorithms on Combinatorial Geometry.* Springer-Verlag, 1987.

[EOS86] H. Edelsbrunner, J. O'Rourke, and R. Seidel. Constructing arrangements of lines and hyperplanes with applications. *SIAM Journal on Computing*, 15:341–363, 1986.

[ES74] P. Erdös and J. Spencer. *Probabilistic Methods in Combinatorics.* Academic Press, 1974.

[ES86] H. Edelsbrunner and R. Seidel. Voronoï diagrams and arrangements. *Discrete and Computational Geometry*, 1:25–44, 1986.

[ESS] H. Edelsbrunner, R. Seidel, and M. Sharir. On the Zone Theorem for hyperplane arrangements. In preparation.

[GKS90] L.J. Guibas, D.E. Knuth, and M. Sharir. Randomized incremental construction of Delaunay and Voronoï diagrams. In *ICALP 90*, pages 414–431, Springer-Verlag, July 1990. To be published in Algorithmica.

[GS78] P.J. Green and R. Sibson. Computing Dirichlet tesselations in the plane. *The Computer Journal*, 21, 1978.

[Kle80] V. Klee. On the complexity of d-dimensional Voronoï diagrams. *Archiv der Mathematik*, 34:75–80, 1980.

[Lee82] D.T. Lee. On k-nearest neighbor Voronoï diagrams in the plane. *IEEE Transactions on Computers*, C-31:478–487, 1982.

[Meh84] K. Mehlhorn. *Data Structures and Algorithms 3: Multidimensional Searching and Computational Geometry*. Springer-Verlag, 1984.

[MMO91] K. Mehlhorn, S. Meiser, and C. Ó'Dúnlaing. On the construction of abstract Voronoï diagrams. *Discrete and Computational Geometry*, 6:211–224, 1991.

[MS91] K. Mulmuley and S. Sen. Dynamic point location in arrangements of hyperplanes. In *7th ACM Symposium on Computational Geometry in North Conway*, pages 132–142, 1991.

[MSW92] K. Mehlhorn, M. Sharir, and E. Welzl. Tail estimates for the space complexity of randomized incremental algorithms. In *ACM-SIAM Symposium on Discrete Algorithms*, January 1992.

[Mul88] K. Mulmuley. A fast planar partition algorithm, i. In *IEEE Symposium on Foundations of Computer Science*, pages 580–589, 1988.

[Mul89a] K. Mulmuley. A fast planar partition algorithm, ii. In *5th ACM Symposium on Computational Geometry in Saarbrücken*, pages 33–43, 1989.

[Mul89b] K. Mulmuley. On obstruction in relation to a fixed viewpoint. In *IEEE Symposium on Foundations of Computer Science*, pages 592–597, 1989.

[Mul91a] K. Mulmuley. On levels in arrangements and Voronoï diagrams. *Discrete and Computational Geometry*, 6:307–338, 1991.

[Mul91b] K. Mulmuley. Randomized multidimensional search trees : dynamic sampling. In *7th ACM Symposium on Computational Geometry in North Conway*, pages 121–131, 1991.

[Mul91c] K. Mulmuley. Randomized multidimensional search trees: lazy balancing and dynamic shuffling. In *IEEE Symposium on Foundations of Computer Science*, pages 180–196, 1991.

[Ove83] M.H. Overmars. *The design of dynamic data structures. LNCS 156*, Springer-Verlag, 1983.

[PS85] F.P. Preparata and M.I. Shamos. *Computational Geometry : an Introduction*. Springer-Verlag, 1985.

[Raj91] V.T. Rajan. Optimality of the Delaunay triangulation in $\mathbb{R}^d$. In *7th ACM Symposium on Computational Geometry in North Conway*, pages 357–363, June 1991.

[Sch91] O. Schwarzkopf. Dynamic maintenance of geometric structure made easy. In *IEEE Symposium on Foundations of Computer Science*, October 1991. Full paper available as Technical Report B 91-05 Universität Berlin.

[Sei81] R. Seidel. *A Convex Hull Algorithm Optimal for Point Sites in Even Dimensions.* Technical Report 14, Departement of Computer Science, University British Columbia, Vancouver, BC, 1981.

[Sei86] R. Seidel. Constructing higher-dimensional convex hulls at logarithmic cost per face. In *ACM Symposium on Theory of Computing*, pages 404–413, 1986.

[Sei90] R. Seidel. Linear programming and convex hulls made easy. In *6th ACM Symposium on Computational Geometry in Berkeley*, pages 211–215, June 1990.

[Sei91a] R. Seidel. Backwards analysis of randomized geometric algorithms. June 1991. Manuscript.

[Sei91b] R. Seidel. A simple and fast randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Computational Geometry Theory and Applications*, 1, 1991.

[SH75] M.I. Shamos and D. Hoey. Closest-point problems. In *IEEE Symposium on Foundations of Computer Science*, pages 151–162, October 1975.

[Sha78] M.I. Shamos. *Computational Geometry.* PhD thesis, Department of Computer Science, Yale University, (USA), 1978.

[vEBKZ77] P. van Emde Boas, R. Kaas, and E. Zijlstra. Design and implementation of an efficient priority queue. *Mathematical Sysyems Theory*, 10:99–127, 1977.

[Wel85] E. Welzl. Constructing the visibility graph for n line segments in $o(n^2)$ time. *Information Processing Letters*, 20:167–171, 1985.

[Yap87] C.K. Yap. An $o(n \log n)$ algorithm for the Voronoï diagram of a set of simple curve segments. *Discrete and Computational Geometry*, 2:365–393, 1987.

Liste des Figures

Table des Matières

Résumé

La Géométrie algorithmique a pour but de concevoir et d'analyser des algorithmes pour résoudre des problèmes géométriques. C'est un domaine récent de l'Informatique théorique, qui s'est très rapidement développé depuis son apparition dans la thèse de M.I. Shamos en 1978.

La randomisation permet d'éviter le recours à des structures compliquées, et s'avère très efficace, tant du point de vue de la complexité théorique, que des résultats pratiques.

Nous nous sommes intéressés plus particulièrement à la conception d'algorithmes dynamiques : en pratique, il est fréquent que l'acquisition des données d'un problème soit progressive. Il n'est évidemment pas question de recalculer le résultat à chaque nouvelle donnée, doù la nécessité d'utiliser des schémas (semi-) dynamiques.

Nous introduisons une structure de données très générale, le Graphe d'Influence, qui permet de construire de nombreuses structures géométriques : diagrammes de Voronoï, arrangements de segments...

Nous étudions les algorithmes, à la fois du point de vue de la complexité théorique, de leur mise en œuvre pratique et de l'efficacité des programmes.

Mots-clés

Géométrie algorithmique, complexité, algorithmes randomisés,
structures de données géométriques, algorithmes dynamiques,
diagrammes de Voronoï, arrangements, enveloppes convexes.