Hiérarchisation et facettisation de la représentation par segments d'un graphe planaire

Jean Michel Moreau

ECOLE NATIONALE SUPERIEURE DES MINES DE SAINT-ETIENNE

Numéro d'ordre 46 ID

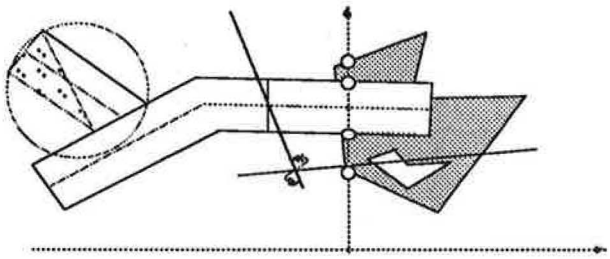
THESE

présentée par Jean Michel MOREAU

pour obtenir le grade de DOCTEUR DE L'UNIVERSITE DE SAINT-ETIENNE
ET DE L'ECOLE NATIONALE SUPERIEURE
DES MINES DE SAINT-ETIENNE
(Spécialité: Informatique)

HIERARCHISATION ET FACETISATION DE LA REPRESENTATION PAR SEGMENTS D'UN GRAPHE PLANAIRE DANS LE CADRE D'UNE ARITHMETIQUE MIXTE

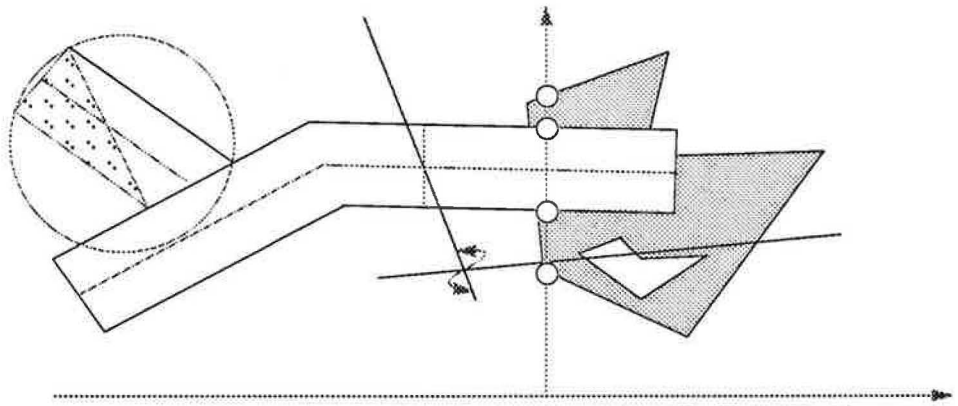
- Soutenue à Saint-Etienne le 12 octobre 1990 -



COMPOSITION DU JURY:

Monsieur	Bernard CHAZELLE	Rapporteurs
Monsieur	Bernard LACOLLE	
Monsieur	Jean AZEMA	Examinateurs
Monsieur	Yves BOUDAUD	
Monsieur	Pere BRUNET i CROZA	
Monsieur	Bernard PEROCHE	
Monsieur	Claude PUECH	

Hierarchisation et Facetisation de la Representation par Segments d'un Graphe Planaire dans le Cadre d'une Arithmetique Mixte.



J.M. Moreau
E.M.S.E.
1990.

TABLE DES MATIERES

Liste des figures et algorithmes *vii*

Remerciements *xi*

Plan *xv*

Préface *xvii*

I. TRAVAIL DE BASE ET BASE DE TRAVAIL **1**

I.A. La chaîne de simulation 2

I.B. Données 2

I.B.1 Domaines 2

1.B.2. Altimétries 3

Filtres et extrema 3

I.B.3 Les lignes de relief 4

I.B.4. Cadastre 5

Aspect et forme 5

Champs 5

Bâtiments 6

Arbres 7

Routes, rivières, etc... 7

I.B.5. Priorités 8

I.B.6. Sémantique des objets 8

I.C. Restructuration des données 9

I.D. Facétisation 13

I.E. Quelques illustrations 14

I.F. Quelques difficultés 17

I.G. Prospective 19

II. DE L'ART DE COUPER LES SEGMENTS EN QUATRE 21

II.A. Intersection de deux droites coplanaires 23
II.A.1 Le coût du calcul d'intersection de deux droites 24
II.B. Intersections de deux segments 28
II.B.1 Un peu d'ordre 28
II.B.2 Paramètres contre puissances 28
Un point commun 29
La pente fatale 31
Quelques commentaires... 32
Distance et puissance d'un point par rapport à une droite 34
Détermination du point d'intersection 35
Marchandages... 35
Présentation générale de l'algorithme 36
Analyse de l'algorithme 37
La méthode paramétrique revue et corrigée 38
Justification a posteriori 39
II.B.3 Où l'on reparle de précision... 40

III. IMPRECISIONS ARBITRAIRES 43

III.A. ... Précisions 45
III.A.1 Genèse 45
Optique 45
Choix 46
Notations 47
III.A.2 Représentation des pseudo-réels et précision 47
Expansion binaire 47
Déphasages 48
Précision et nombre de bits 49
Précision et taille des nombres 49
III.B. Imprécision en géométrie algorithmique 50
III.B.1 Où l'on reparle d'intersections... 50
III.B.2 Déplacements et dépendances 52
III.B.3 La deuxième génération 56
III.B.4 Incohérences 56
III.C. Tour d'horizon des solutions connues 58
III.C.1 Epsilons 59
Epsilon brut 59
Taille d'attributs minimum 60

epsilon-convexité 60
Géométrie 'à-epsilon-près' 61
III.C.2 Géométries robustes 62
Résolution finie 62
Déterminisme et historique 63
Valence et cohérence 64
III.C.3 Stabilité numérique et perturbations 65
Simulation de simplicité 65
Stabilité numérique 67
III.C.4 Solutions structurelles 68
Diagramme de Voronoi 68
Triangulation de Delaunay 69
III.C.5 Solutions exactes 71
Recalage sur une grille entière 71

IV. PRECISION ARBITRAIRE 73

IV.A. Caractéristiques d'une arithmétique rationnelle 75
IV.A.1 Représentation des rationnels 76
Type et Base 76
Représentation des rationnels 77
Débordements 79
Gestion de la mémoire 80
IV.A.2 Opérations élémentaires 80
Arithmétique des entiers 80
Arithmétique des rationnels 93
Un modèle deux fois plus compact 94
IV.B. Spécificités d'une solution adaptative 97
IV.B.1 Limites naturelles 97
IV.B.2 Limites structurelles 97
Structure de graphe planaire 98
IV.B.3 Opérations 99
Opérations élémentaires 99
Primitives 99
Analyse rapide des techniques relatives à ces primitives 99
Opération globale 101
IV.C. Le monde des non-epsilons 102
IV.C.1 Réticence et mixité 102
IV.C.2 Calculs de fiabilité 104
Principes de base 105

Sens et dégénérescences *106*
Seuils et nombres positifs *106*
Modularité et résultats intermédiaires *106*
Seuil global ou seuil local? *107*
IV.C.3 Structures mixtes pour la réticence 108
Générations 108
Réflexes et valeurs par défaut 109
Les pièges de l'égalité! 109
Classes d'équivalences et saules 111
Simulation naturelle de balayage 113
Performances 114
Philosophie 116
3-D 116
Annexe *117*

V. CONTRAINTES, DEPENDANCES, HIERARCHIES ET PRIORITES DANS UN GRAPHE 119

V.A. Le point de vue des arêtes *121*
V.A.1 Définitions 121
V.A.2 Transformation d'un pré-graphe en rsgp 122
Algorithme de Bentley-Ottman amélioré 125
Mise à jour des segments 129
Répétitions 130
V.A.3 Résolution de dépendances 137
Dépendance par priorité 138
Arbre de dépendance 138
V.B. Le point de vue des faces *144*
V.B.1 Décomposition de faces (clipping) 144
Modélisation de la multiplicité 146
Technique de décomposition 151
V.B.2 Hiérarchies 160
Illégalité 168
V.C. Contraintes géométriques et dépendances sémantiques *168*
V.C.1 Une nouvelle génération d'objets 168
Permis de construire 169
r-dilatés tri-dimensionnels 172
Où l'on reparle d'intersections... 177
Dépendances sémantiques 180
Radicaux 187

VI. FACETISATION AVEC CONTRAINTES 189

VI.A. Diagrammes de proximité 191

VI.A.1 Diagrammes de Voronoi 191

Définitions et propriétés générales 191

Construction par fusion 193

VI.A.2 Triangulation de Delaunay 196

Définitions et propriétés générales 197

Triangulation d'une rsgp 198

Dénombrement 198

Dualité Voronoi-Delaunay 200

Construction par fusion 202

VI.A.3 Constructions incrémentales 206

Construction incrémentale de l'enveloppe convexe d'un nuage de points 206

Construction incrémentale de DV(Sn) 208

Encore un peu de dénombrement... 214

Polygones monotones et convexes 215

Construction incrémentale de TD(Sn) 215

VI.B. Triangulation de Delaunay d'une rsgp 216

Quelques définitions 216

VI.B.1 Triangulation de Delaunay d'une rsgp non-hiérarchisée 217

Construction d'une triangulation contrainte par fusion 218

Construction incrémentale d'une triangulation contrainte 223

Diagramme de Voronoi et triangulation de Delaunay avec contraintes 230

VI.B.2 Triangulation de Delaunay d'une rsgp hiérarchique 231

Conclusion 233

Bibliographie Générale 235

Publications personnelles 239

Index 241

Liste des Figures et algorithmes

1. *Figure I.1. Interpolation bi-linéaire d'altitude.* 3
2. *Figure I.2. Influence d'une ligne de relief.* 4
3. *Figure I.3. La maison dans la montagne...* 6
4. *Figure I.4. s-polygone et projections d'un point sur les génératrices.* 7
5. *Figure I.5. Intersection d'une ligne de relief et d'un s-polygone expansé.* 10
6. *Figure I.6. Réseau de dépendances altimétriques.* 11
7. *Figure I.7. Evolution de quelques données.* 12
8. *Figure I.8. Triangulation à partir des éléments de f-I.7.* 14
9. *Clichés 1, 2* 15
10. *Clichés 3, 4* 16

11. *Figure II.1. Arbre de décision et coûts relatifs à l'intersection de 2 droites.* 27
12. *Figure II.2. L'ordre lexicographique réduit le nombre de configurations.* 29
13. *Figure II.3. L'ordre lexicographique élimine certain cas d'intersection.* 29
14. *Figure II.4. L'ordre lexicographique simplifie certains cas...* 29
15. *Figure II.5. Base commune de l'arbre de décision.* 30
16. *Figure II.6. Arbre de décision et coûts de l'algorithme 'de la pente'.* 33
17. *Figure II.7. Utilisation de la puissance d'un point...* 34
18. *Figure II.8. Arbre de décision et table de coûts relatifs...* 37
19. *Figure II.9. Table de projection des gains en temps de calcul...* 40

20. *Figure III.1. Intersection simple.* 52
21. *Figure III.2. Pseudo-réel et ses deux plus proches voisins.* 53
22. *Figure III.3. Grille d'espacement des pseudo-réels.* 55
23. *Figure III.4. Ω est "arrondi" en Ω.* 55
24. *Figure III.5. Conflit de dépendance.* 55
25. *Figure III.6. Problèmes de précision à la deuxième génération.* 57
26. *Figure III.7. Segments pour l'algorithme de Bentley-Ottman.* 57
27. *Figure III.8. Deux états rapprochés de la structure de balayage.* 57
28. *Figure III.9. Imprécision et incohérence topologique.* 58
29. *Figure III.10. Taille d'attribut minimum pour le segment [A,B].* 61
30. *Figure III.11. Crochet sur un segment.* 63
31. *Figure III.12. Transformation incrémentale d'un diagramme de Voronoi.* 69

32. *Figure IV.1. Représentation d'un rationnel.* 77
33. *Figure IV.2. Multiplication de deux entiers.* 83
34. *Figure IV.3. Retournement du chaînage relatif à u.* 85
35. *Figure IV.4. Une division est la répétition de divisions réduites.* 87
36. *Figure IV.5. Une liste pour le dividende courant, le reste et le quotient.* 89
37. *Figure IV.6. Modèle de représentation en anneau des entiers.* 95
38. *Figure IV.7. Division du plan autour de s.* 98
39. *Figure IV.8. Angle orienté déterminé par trois points.* 101
40. *Figure IV.9. Domaines de validité de l'arithmétique 'mixte'.* 103
41. *Figure IV.10. Arbre binaire et structure sous-jacente.* 111
42. *Figure IV.11. Arbre binaire en forme de saule.* 113

43. **Figure IV.12.** *Bande de largeur 2ε autour d'un segment.* 115
44. *Table de valeurs de seuils de fiabilité pour des données allant jusqu'à 100 km.* 117

45. **Figure V.1.** *Représentation topologique plane d'un graphe.* 121
46. **Figure V.2.** *Il est nécessaire de découper les segments intersectés.* 123
47. **Figure V.3.** *Segments pour l'algorithme amélioré de Bentley-Ottman.* 129
48. **Figure V.4.** *Cas pathologique pour le test de deux segments.* 131
49. **Figure V.5.** *Limitation du nombre de tests d'intersection.* 131
50. **Figure V.6.** *Adjacence au voisinage d'une intersection.* 133
51. **Figure V.7.** *Deux cas d'inversion.* 135
52. **Figure V.8.** *Rectification concernant le voisin inférieur et la suite s,t.* 136
53. **Figure V.9.** *Relations de dépendance dans un pré-graphe.* 137
54. **Figure V.10.** *Cascade de dépendances.* 139
55. **Figure V.11.** *Relations de dépendance dans l'arbre binaire.* 139
56. **Figure V.12.** *Descente de cascade.* 140
57. **Figure V.13.** *Intersection vide ou non?* 145
58. **Figure V.14.** *A quel polygone appartient la face hachurée?* 145
59. **Figure V.15.** *Décomposition ensembliste ordonnée.* 145
60. **Figure V.16.** *Arête canonique et arêtes polygonales de même classe.* 146
61. **Figure V.17.** *Classes et listes d'objets.* 148
62. **Figure V.18.** *Arêtes polygonales et arêtes canoniques.* 148
63. **Figure V.19.** *Les champs d, g et i et l'orientation des arêtes.* 148
64. **Figure V.20.** *Intersection et sens de parcours.* 151
65. **Figure V.21.** *Intérieur et extérieur.* 152
66. **Figure V.22.** *Décomposition et priorité.* 153 et 155
67. **Figure V.23.** *Décomposition de trois polygones.* 156
68. **Figure V.24.** *Deux cas où le polygone P disparaît.* 159
69. **Figure V.25.** *Disparition inopportune d'une face.* 159
70. **Figure V.26.** *Priorités et inclusions.* 161
71. **Figure V.27.** *Hiérarchie, inclusion et priorité.* 163
72. **Figure V.28.** *Cas limite dans une hiérarchie.* 163
73. **Figure V.29.** *Transformation d'inclusion-adjacence en adjacence pure...* 163
74. **Figure V.30.** *Orientation d'arêtes de polygones...* 163
75. **Figure V.31.** *Ordre vertical de deux arêtes jumelles.* 164
76. **Figure V.32.** *Arêtes canoniques de demi-plans gauche et droit.* 164
77. **Figure V.33.** *Construction de hiérarchie par balayage.* 165
78. **Figure V.34.** *Détermination de la face d'inclusion.* 165
79. **Figure V.35.** *Ordre x-polaire contre ordre lexicographique.* 166
80. **Figure V.36.** *Discrimination impossible entre les arêtes horizontales.* 167
81. **Figure V.37.** *r-dilaté d'une chaîne de segments.* 169
82. **Figure V.38.** *Couronne dégénérée.* 169
83. **Figure V.39.** *Géométrie d'un virage à droite.* 171
84. **Figure V.40.** *Représentation tri-dimensionnelle d'un r-dilaté de segment.* 173
85. **Figure V.41.** *Palier dans un r-dilaté tri-dimensionnel.* 173
86. **Figure V.42.** *Aménagement d'un palier dans l'espace.* 174
87. **Figure V.43.** *Aménagement d'un palier dans le plan.* 175
88. *Médaillon* 176
89. **Figure V.44.** *Segment de contrainte et r-3d.* 177
90. **Figure V.45.** *Projection de l'intersection...* 178
91. **Figure V.46.** *Chaîne de contrainte un peu torturée...* 178
92. **Figure V.47.** *Traitement simplifié du cas précédent.* 179
93. **Figure V.48.** *Intersection de deux r-dilatés.* 179
94. **Figure V.49.** *Divers arrêts de r-dilatés suivant les circonstances.* 180
95. **Figure V.50.** *La géométrie des intersections de r-dilatés est complexe.* 181
96. **Figure V.51.** *Intersection de trois r-dilatés en un point.* 181
97. **Figure V.52.** *Bordure mal définie.* 182
98. **Figure V.53.** *Rond-point pour la figure V.51.* 185

99.	***Figure V.54.** Deux techniques de subdivision.*	186
100.	***Figure V.55.** Sommets zombies.*	186
101.	***Figure VI.1.** Diagramme de Voronoi sur deux (a) et trois (b) sites.*	193
102.	***Figure VI.2.** Diagramme de Voronoi sur 9 sites.*	193
103.	***Figure VI.3.** Fusion de deux sous-diagrammes de Voronoi.*	194
104.	***Figure VI.4.** Graphes planaires et triangulations.*	197
105.	***Figure VI.5.** Rsgp (a) et triangulation 'par défaut' (b).*	198
106.	***Figure VI.6.** Triangulation d'un polygone 'plein'.*	199
107.	***Figure VI.7.** Triangulation de Delaunay à partir du diagramme de f-VI.2.*	201
108.	***Figure VI.8.** Equivalence des critères du cercle et de l'angle minimum.*	201
109.	***Figure VI.9.** Principe de base de la fusion de deux sous-triangulations.*	203
110.	***Figure VI.10.** Exemple complet de fusion...*	205
111.	***Figure VI.11.** Recherche des deux segments d'appui depuis p_i.*	207
112.	***Figure VI.12.** Insertion de p_4 dans le diagramme f-VI.1.(b).*	209
113.	***Figure VI.13.** Insertion du cinquième point.*	210
114.	***Figure VI.14.** Insertion du sixième point.*	210
115.	***Figure VI.15.** Insertion ultime.*	211
116.	***Figure VI.16.** Rsgp, triangulation de Delaunay...*	217
117.	***Figure VI.17.** Bandes initiales et segments d'encadrement.*	219
118.	***Figure VI.18.** Fusion contrainte.*	221
119.	***Figure VI. 19.** Illustration du lemme 1.*	224
120.	***Figure VI.20.** Arêtes de contrainte bloquantes.*	225
121.	***Figure VI.21.** Situation impossible.*	226
122.	***Figure VI.22.** Triangulation d'une rsgp.*	227
123.	***Figure VI.23.** Insertion de p_5.*	227
124.	***Figure VI.24.** Insertions de p_6.*	228
125.	***Figure VI.25.** Insertions de p_7.*	228
126.	***Figure VI.26.** Insertion de l'avant dernier sommet.*	228
127.	***Figure VI.27** Insertion du dernier sommet.*	228
128.	***Figure VI.28.** Diagramme de Voronoi borné de la rsgp précédente.*	230

II	*Algorithme de la pente*	*32-33*
	Algorithme des puissances	*36-37*
III	*Addition de deux naturels*	*81*
	Soustraction de deux naturels	*82*
	Multiplication de deux naturels	*84*
	Division de u par v: cas où v a un seul chiffre	*86*
	Division de deux naturels u et v: cas général	*92*
IV	*Détermination de l'ordre entre xA et xB*	*102*
V	*Algorithme de Bentley-Ottman amélioré*	*127*
	Algorithme de gestion des listes de test	*135*
	Résolution de dépendances dans un arbre binaire	*142*
	Décomposition par priorités décroissantes	*160*
VI	*Fusion de deux diagrammes de Voronoi*	*196*
	Fusion de deux triangulations de Delaunay	*204*
	Construction incrémentale d'un diagramme de Voronoi	*213*
	Construction incrémentale d'une triangulation de Delaunay	*215*
	Fusion de deux triangulations contraintes de Delaunay	*222*
	Construction incrémentale d'une triangulation de Delaunay d'une rsgp	*229*

A mes parents, qui m'ont fait comme j'étais,

A Alice et Hélène, qui m'ont fait comme je suis.

A la Camarde, je dis rien... De toute façon, elle finira les restes sans être invitée...

Remerciements

Je voudrais remercier tout spécialement le Professeur Bernard Chazelle, de l'Université de Princeton, dont on imagine la charge de travail, pour avoir accepté spontanément d'être rapporteur de cette thèse. Je ne pouvais espérer plus beau cadeau...

Bernard Lacolle, Maître de Conférence à l'I.M.A.G., lui aussi particulièrement pris par ses activités de recherche à Grenoble, a accepté, d'aussi bonne grâce, de se soumettre à la rude tâche de rapporteur. Qu'il en soit chaleureusement remercié ici.

Je voudrais remercier Claude Puech, Professeur à l'E.N.S., pour avoir accepté de présider ce jury en plus de ses multiples activités et responsabilités en France et aux Etats-Unis, et pour m'avoir signalé un oubli et un point théorique d'intérêt dans le dernier chapitre (ainsi que dans les remerciements...!!!).

Merci à Bernard Péroche, pour avoir accepté de diriger ces recherches en plus de beaucoup d'autres activités déjà accaparantes, d'avoir incité de nombreuses réflexions et solutions et de m'avoir procuré l'immense joie d'en arriver là!

Merci aussi à Yves Boudaud, Chef du projet SINDBAD à SOGITEC-Rennes, pour avoir guidé ce projet et avoir incité le contrat de recherche avec l'E.M.S.E. qui a permis d'initier cette thèse. Merci aussi pour les nombreuses idées qui sont venues éclairer de bien délicates situations et merci enfin pour sa participation à ce jury.

Merci encore à Pere Brunet i Croza, Professeur à l'*Universitat Politecnica de Catalunya*, Barcelone, pour avoir accepté de faire partie de ce jury avec tant d'enthousiasme.

Merci à Jean Azéma, Maître de Conférence à l'Université Jean Monnet de Saint-Etienne, pour son enseignement en algorithmique, son intérêt toujours marqué pour le travail en devenir, pour ses critiques et son aide dans la relecture du manuscrit et pour sa présence dans ce jury.

Que Franck Rousee, de SOGITEC-Rennes, à qui ce projet doit aussi beaucoup de ses idées novatrices, soit remercié pour avoir souvent été le point de rencontre de vues qui n'étaient pas toujours convergentes et pour les longs échanges que nous avons eus.

Je voudrais avoir ici une attention toute spéciale envers mon ami Jean-Louis 'The klingon' Beau, dont la patience, la méthodologie et la compétence m'ont plus appris que toutes ces années d'erreurs de syntaxe et de links erroïques; grâce à lui, j'ai pu voir ce que pouvait être un algorithme *lisible*, commenté, documenté et compréhensible par quelqu'un d'autre... C'est aussi à lui que je dois les clichés du chapitre I; qu'il en soit remercié encore ici (et Marie-Claude pour avoir supporté en silence... et pour l'accueil si chaleureux).

Merci aussi à tous les autres *cetigos*, Dominique, Isabelle, Gilles, Jean-Luc, Iris Quat-dé, etc. Votre présence et votre gentillesse ont été un réconfort de tous les jours, sauf quand le core dumpait, le soir au fond des bois...

Et puis[1], il y a les mercis à tous ceux qui sont tellement proches qu'ils pensent qu'on oublie que, sans eux, y'aurait plus personne, en quelque sorte, c'est une image... Merci à tous mes INFAmes camarades de l'harmonie des Mineurs: merci à vous, Jean-Pierre, Jean-Paul, Daniel, Michel (C) aux machines qui-font-rien-qu'à-m'embêter (ah bon, il fallait juste brancher l'alim?), Mohand Ouraba aux frontières du représentable, Elmi à la configuration impossible, Pascale et Elizabeth aux jumelles, Marc (Roelens-Phelbs) à la résolution d'équation du dix huitième degré impossible (Marc, quand tu liras ce message, la solution s'auto-détruira et le Département Infa niera toute implication si es fait prisonnier dans un espace de dimension supérieure ou égale à un!) et à la pollo (de Gabriel aux desserts cornichonnés sidérants), Philippe à la trousse d'urgence, au Macintosh-en-un et au pied de l'échaffaudage (mais, c'est tout c'que t'as trouvé comme habits?), Gilles au t...ennis, aux cônes lumineux et à la joie de vivre contagieuse, Marie-Line à *la* baguette (de chef d'orchestre, qu'est-ce que tu vas imaginer?), aux-choses-à-remplir-que-j'y-comprends-rien et aux petits soins, Franck-aux-trois-cinq-cents ("Y'a pas quelqu'un qui pourrait me filer 500 Mégas, 500 jours de calcul non-stop et 500 Kilo-Francs, c'est pour LE film?"), Geneviève au langage évolué et à la recherche de publis plus vite que son ombre, Zhi-Gang aux baguettes déformées librement, Ghassan au cèdre taciturne, Christine aux plâtres grecs, Christian B. au système expert, Christian dB. au PostScript et au débogueur de debugger avec bugs et bunnies, Pierre O.K. et sous la pression d'Archimède, Paul à 'les normes x400' (enfin, j'suis pas allé voir...), /bin/mail/Bernard/Kaddour au sfooler pou, You aux boîtes partagées sans compter, Jacqueline au lancer de rayon aux p'tits misés, Didier à l'interface humaine et aux petits plats divins, Antoine au serveur complice, Clément au schtroumph Pascalien parce qu'il n'aime pas le C, Jori au compilateur... d'histoires, Ehoud à l'hélicon (ma mère...) et au *fluent* Unix, Yolande aux calculs algébriques, Roland aux graphes repétables (?), Dominique au zozotement parenthésé et aux 'mph, ça doit pas être difficile' ("il vaut mieux que tu me répètes dans R^2, au cazoù j'aurais pas tout compris dans R^n"), Annie aux transputers, aux architectures pleine de machines et au kleenex les jours de grandes marées, et *last but not least*, J.J-la-science-f(r)iction à la répartie qui tue (enfin, trois heures après, quand on l'a comprise, avant, on fait juste semblant d'être mort).

Un merci, encore plus spécial, à Michel (B), pour m'avoir montré la voie RS232 un jour de désespoir à ne pas sortir un maître-auxilaire de son Alma Maternelle et pour le million et demi de réponses à des questions dont je n'imaginais même pas l'existence, à Jean-Marie pour avoir mis des panneaux indicateurs au cas où je n'aurais pas vu la sortie d'urgence, à Michel (J) pour m'avoir fait retourner sur le premier coefficient de forme qui passait, à Michel (R), Bésicovitch, Blachke et N.S. 1500 pour avoir tramé hexagonalement mon changement de destinée, et à l'Education Nationale pour m'avoir *tellement* aidé à ne pas résister.

Merci à Ritchie et Kernighan, à Van Newman, à Boole, à Ada Byron, à Berkeley System V et à mon Mac favori. Merci à Fred et Coramine pour quand l'hypo glycémait... Merci à Murphy pour "if you're feeling good, dont worry; you'll get over it".

[1] La suite de cette partie est particulièrement ésotérique et chargée de violence. Nous prions le lecteur sensible de passer au chapitre suivant et, en tout cas, d'éloigner les enfants.

Merci aussi à Marie-José & Jean-Jacques, Raymond, Jean de Guyane, Pierrot & Colette, Jacqueline (S), Michèle & Anaïs, Christiane & Frater, Georges & Nina, Rafaële et Gilles, Adriana & Mario, Jean-Christophe et Claude, Guy, René et Marguerite, Chris & Mary, Sally, Patrick, Marcel & Jean, pour le bonheur de leur amitié et pour avoir réprimé leurs baîllements quand j'étais lancé dans de si 'passionnantes' envolées.

Merci à toute ma famille pour avoir accepté que je sois perdu pour l'humanité pendant quatre ans, à Prisca pour ces longues promenades où tu m'as raconté ta vie de chien, Titou pour les coups de pattes à la souris.

Malheureusement, les mercis ne suffisent pas à se faire pardonner les soirées, les journées, les week-ends, les fêtes, les ponts, les congés et parfois les vacances, passés romantiquement à écouter le clapotis toujours recommencé... des touches sur un clavier. Rassure-toi Alice, toi aussi ~~petite~~ grande Hélène et toi, Gilles: votre enfer est fini; il n'y a plus de Thèse d'Etat, juste des tas de thèses...

Merci, enfin, à G. Mahler, K. Ferrier, Ludwig Van B., B. Dylan, The Beatles, The Police, Eurythmics, M. Knopfler, J. Clegg, Sting, Ry Cooder, R. Newman, G. Brassens, Renaud, W. Allen, M. Cimino, Spike Lee, le numéro 6, Tony, Angela, Samantha, Jonathan & Mona, P. Desproges, Coluche, Monty Python, Andy Capp, Asimov (l'humoriste), *The* Newcastle Brown Ale et le Jurançon pour avoir fait tant de bien là où ça fait:

```
$    cd /
$    rm *
     Oops
$    Oops not found
$    not found not found
```

krouik...

Plan

Cette thèse est organisée en deux parties complètement disjointes. La première partie est constituée du chapitre I, consacré à la présentation du travail qui a servi de support d'implantation aux idées développées. On y trouvera une description générale du travail de base effectué en deux ans sur une application de modélisation hiérarchisée de terrain pour un simulateur de vol.

La seconde partie couvre les chapitres II à VI et présente les concepts théoriques qui se sont dégagés de l'étude pratique précédente. Il faut y voir une tentative de formalisation des problèmes que l'on rencontre dans la structuration de données abstraites, comme un graphe hiérarchique.

Le chapitre II, *L'art de couper les segments en quatre*, présente, comme premier contact avec les difficultés liées aux calculs exacts, une méthodologie permettant d'optimiser de manière notoire les procédures d'intersection de segments dans le plan. Il n'y a derrière cette opération aucune difficulté théorique, mais on constate vite que la méthode traditionnelle est loin d'être optimale. C'est à partir de cette procédure qu'a pu germer la notion d''arithmétique réticente'.

Le chapitre III, *Imprécisions arbitraires*, traite des difficultés inhérentes à l'emploi de calculs de précision finie en géométrie algorithmique. On y trouve les principales sources d'échec et une présentation des solutions couramment adoptées pour pallier ce problème.

Le chapitre IV, *Précision Arbitraire*, introduit une méthode mixte de résolution des problèmes d'imprécision en suggérant que l'on fasse les calculs à deux niveaux: tant que la précision de la machine est suffisante, on effectue les calculs en précision résidente et dès que celle-ci devient incapable de permettre les décisions, on passe aux calculs en arithmétique exacte.

Ceci permet, si l'on prend la précaution de toujours créer les objets à l'aide de l'arithmétique exacte, de garder une cohérence totale dans la topologie des données tout en limitant de manière très conséquente le volume de calculs en arithmétique exacte.

Le chapitre V, *Contraintes, Dépendances, Hiérarchies et Priorités dans un Graphe*, présente les opérations nécessaires à la gestion de graphes hiérarchisés et dotés d'une relation d'ordre sur les objets.

Le chapitre VI, *Facétisation avec contraintes*, donne une description d'une technique incrémentale de triangulation de Delaunay contrainte de complexité optimale. L'accent est mis sur les différentes définitions que l'on peut donner de cette opération et la solution apportée représente une simplification de la célèbre méthode 'diviser pour résoudre'.

Research is what I am doing when
I don't know what I am doing...

Wernher von Braun

PREFACE

En informatique, on rencontre deux types de problèmes: ceux qui conduisent à des solutions simples et ceux qui conduisent à des solutions complexes. L'algorithmique, à mon sens, est ce chaînon qui doit œuvrer pour rendre toute solution complexe, simple. La géométrie algorithmique se doit encore plus de répondre à cette attente, car son domaine d'application est souvent notre environnement physique quotidien, dans lequel il n'est pas facile de se contenter de solutions qui ne soient pas de 'bon sens'.

Il paraissait, par exemple, dur d'accepter qu'il n'existe pas d'algorithme simple pour construire l'enveloppe convexe d'un polygone. La géométrie algorithmique a franchi des barrières virtuellement insurmontables, il y a dix ans, et c'est souvent parce que des chercheurs ont réussi à simplifier certains concepts au maximum, dès qu'ils les maîtrisaient mieux, afin que d'autres concepts plus complexes puissent, à leur tour, en bénéficier. La simplicité n'est, en fait, jamais synonyme de facilité!

Il y a une sorte de 'loi de la matière' informatique: toute question simple *devrait* obtenir la réponse la plus simple possible. Tant que la réponse paraît plus compliquée que la question, il faut remettre l'ouvrage sur le métier et de nouveau chercher plus simple.

Certains parmi nous ont pour vocation de prouver qu'un problème réputé difficile admet une solution. Ceux-là permettent des miracles. D'autres ont la vocation de dégager tous les éléments qu'ils peuvent pour faire progresser la solution trouvée sur la pente de plus forte simplicité.

J'ai toujours cru en cette méthodologie et je pense que cette thèse en est une manifestation. Elle contient, parmi les résultats présentés, certains dont on peut penser qu'ils sont des affinements de solution déjà publiées.

A ce titre, le dernier chapitre, qui présente aussi des solutions nouvelles à un problème ancien, est une tentative d'unification de trois solutions autour d'une méthode incrémentale, comme il existait déjà une unification de ces trois solutions autour de la célèbre méthode 'diviser pour résoudre'.

Les responsables du projet sous-tendant cette thèse ont donné beaucoup de leurs idées d'hommes de terrain pour permettre au chercheur que je suis de trouver des solutions pratiques. Ceci a été fait dans un premier temps, sous forme d'une application opérationnelle et validée en grandeur réelle. La tâche qui m'incombait était ensuite de généraliser les résultats obtenus par des méthodes plus universelles, en bref, de prendre du recul par rapport aux algorithmes trouvés.

Il y a naturellement deux raisons à cela: la première est liée à la confidentialité des méthodes utilisées. Le projet a des concurrents et il serait un peu dangereux de trop laisser transparaître des techniques mises au point. Je tiens d'ailleurs à rendre hommage ici à la compréhension qu'ont montré les cadres de la SOGITEC concernant la publication de cette thèse!

La deuxième raison est que le contenu d'une thèse ne peut pas être limité à un codage d'algorithmes, mais nécessite une réflexion plus générale, qui permette à chacun de percevoir les modèles de conception sous-jacents.

L'effort a donc porté sur une formalisation des concepts qui se sont présentés dans l'approche initiale, le 'premier jet'. La plupart des algorithmes formels présentés dans cette thèse ont eu une origine plus pragmatique, parfois non optimale, ou sont des prolongements de méthodes que l'expérience a montré trop limitatives. Certaines réflexions permettent d'envisager une refonte totale des techniques appliquées jusqu'ici et une généralisation d'assez grande envergure.

Que les contingences matérielles et industrielles permettent ou non de faire évoluer l'existant vers le suggéré dépasse le cadre de réflexion qui est celui de cette thèse. Il m'a semblé que les idées auxquelles celle-ci aboutit méritaient d'être mises en valeur pour elles-mêmes et c'est dans cet esprit qu'elles ont été décrites.

1. Travail de Base et Base de Travail

Ce premier chapitre a pour but de décrire le cadre de travail qui a inspiré le reste de la présente thèse. Les algorithmes proposés et les méthodologies que l'on trouvera au cours des chapitres à venir, sont tous nés des réflexions qu'a inspirées le travail décrit ici. Deux années de travail sont entrées dans le processus et ont permis d'apporter le chaînon manquant à une chaîne de simulation de vol.

Toutes les idées de cette thèse ont d'abord été codées afin d'obtenir une solution le plus rapidement possible, dans un contexte industriel, à des problèmes assez volumineux. Seulement après a-t-il été possible de commencer à tirer les enseignements naturels des problèmes rencontrés. Tous les chapitres qui suivent celui-ci ont résolument pour principe de tenter de généraliser les notions abordées 'sur le terrain' et d'apporter des éléments de réponse théorique aux problèmes rencontrés. Il existe une certaine distance entre les algorithmes qui ont été implantés et les solutions beaucoup plus générales et souvent mieux adaptées que proposent les chapitres 2 à 6...

On trouvera dans le chapitre présent quelques illustrations qui sont de simples témoins du travail accompli. Elles permettront peut-être au lecteur le moins gagné à la géométrie algorithmique de voir que ce domaine permet de réaliser des applications grandeur nature et qui marchent (ou devrais-je dire volent ?) ...

I.A. LA CHAINE DE SIMULATION

Le travail qui sous-tend cette thèse est parti d'un contrat entre l'Ecole Nationale Supérieure des Mines de Saint-Etienne et la société d'électronique SOGITEC (Rennes et Boulogne Billancourt), dans le cadre du projet SINDBAD, qui vise à créer les bases de données d'images de synthèse nécessaires aux simulateurs.

SOGITEC ayant développé depuis plus de dix ans une machine de visualisation extrêmement sophistiquée, la G.I. 10 000, permettant l'affichage par tampon de profondeur (*z-buffer*) de 10 000 polygones tridimensionnels en un cycle, les besoins se sont fait sentir de disposer d'un gestionnaire de terrains perfectionné.

Les données proviennent de différentes sources (géographiques, fichiers D.M.A., etc...) et de différents modules, comme, par exemple le modeleur 3D qui permet la réalisation de volumes (maisons, bâtiments, etc) et le modeleur de terrain qui permet la 'fabrication' de l'apparence de la zone représentée (relief, textures, paysages).

La cohérence de ces données est ensuite vérifiée et, si tout se passe bien, celles-ci sont transmises au module de facétisation qui doit tout rendre sous forme de triangles.

Un module d'édition de facettes permet d'éventuelles rectifications de détail afin d'obtenir le résultat le plus proche de ce que désire l'utilisateur. Toutes les opérations qui viennent d'être décrites se passent en dehors de la simulation et ne doivent pas, heureusement, être exécutées en temps réel... Celles-ci sont effectuées sur des Iris 4D (Silicon Graphics) et le module de facétisation a été conçu sur une station Sun 3115.

Le simulateur est composé d'une cabine sphérique sur les parois de laquelle les images projetées par la G.I. défilent *en temps réel*. Ceci veut dire que l'on doit être capable d'extraire les informations extrêmement rapidement des bases de données afin de rester en phase avec la vitesse de déplacement simulée de l'avion. Chaque fois qu'il est nécessaire, on doit aller chercher, en temps réel, un *bloc d'extraction* dans la base de données afin de permettre la visualisation de la zone au-dessus de laquelle parvient l'avion.

L'application peut aussi bien servir à des simulations au sol (conduite automobile, déplacements dans une zone urbaine, etc), comme le montrent certaines des illustrations de ce chapitre.

I.B. DONNEES

I.B.1 Domaines

Comme les bases de données utilisées sont immenses (certaines couvrent plusieurs dizaines de milliers de km^2), il est nécessaire de segmenter les zones à traiter en 'domaines'.

Fondamentalement, un domaine est un rectangle destiné à contenir toutes les données que le module de facétisation (qui sera naturellement privilégié ici) devra traiter.

Aucune donnée ne peut, à l'heure actuelle, être autorisée à sortir d'un domaine. L'origine de chaque domaine se trouve dans le coin inférieur gauche.

Les domaines sont en général bordés par huit domaines voisins, sauf les domaines 'du bout du monde'. Toute donnée qui serait partagée par la frontière de deux domaines adjacents doit être vérifiée de part et d'autre, dans une phase de post-traitement de la facétisation.

1.B.2. Altimétries

Sur l'ensemble des domaines est plaquée une grille *altimétrique* rectangulaire qui les recouvre tous, et est structurée en tableau de cellules de taille fixe. Chaque carreau qu'elle contient porte une altitude aux quatre coins. Ces altitudes proviennent de relevés divers et traduisent le plus fidèlement possible le relief sous-jacent.

Pour connaître l'altitude de tout point, il suffit de localiser le carreau de grille dans lequel il se trouve et d'interpoler son altitude sur les quatre connues autour de lui (sauf sur les lignes de la maille où l'interpolation se fait sur un côté seulement et aux sommets de maille où le calcul est direct). La figure I.1 donne un exemple de ce principe.

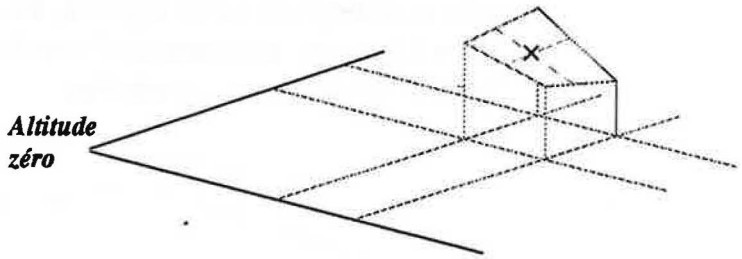


Figure I.1. Interpolation bi-linéaire d'altitude.

Sur cette figure, le plan de référence est indiqué comme ayant l'altitude zéro. C'est le plan qui sera toujours noté ***P*** dans la suite de cette thèse, et qui contient la projection bi-dimensionnelle de tous les objets contenus dans un domaine. On trouve un carreau de grille altimétrique à la croisée des chemins de quatre génératrices de la grille. Chaque coin du carreau a une altitude propre ce qui permet de visualiser un carreau de surface 'altimétrique' dans l'espace. Les côtés de ce carreau de surface sont des segments de droite et chaque point d'un côté a une altitude interpolée sur celles des deux coins de carreau que le côté relie. L'altitude de tout point situé à l'intérieur d'un carreau de surface altimétrique est interpolé sur les deux directions génératrices. Cette opération porte le nom d'interpolation bi-linéaire.

Le côté du domaine le plus proche de l'observateur sur la figure I.1 portera en général l'axe des abscisses et le second l'axe des ordonnées.

Filtres et extrema

Afin de ne pas avoir trop de données, il est nécessaire de 'filtrer' les altimétries: pour prendre un exemple simple, si les données proviennent de relevés aériens, on risque d'avoir un échantillonnage d'altitude très régulier, disons tous les soixante mètres. Si la zone représentée est

montagneuse, ceci est à la limite sous-échantillonné, mais si elle représente une plaine ou un lac, il y a de grandes chances pour que les informations soient *vraiment* redondantes.

Un module de filtrage permet donc de réduire l'échantillonnage des altimétries, lorsqu'il est besoin.

Les premières données, les plus simples aussi, sont constituées par les *extrema* altimétriques. Ce sont des points isolés ou qui se trouvent sur la frontière d'autres objets. Ils correspondent donc à des points remarquables du relief, ou servent à marquer ou à 'forcer' certains traits de relief car leur altitude peut, en pratique, être altérée par l'utilisateur.

I.B.3 Les lignes de relief

Lorsqu'on désire, pour des raisons géographiques ou liées à la représentation des terrains (pour établir, par exemple, des frontières franches entre deux faces de terrain), forcer le relief naturel à suivre d'autres pentes, on peut recourir aux lignes de relief.

Un tel objet est constitué d'un ensemble de segments consécutifs; chaque segment a deux extrémités qui sont, en fait, des extrema altimétriques. L'altitude de tout point à la verticale d'un segment de relief est interpolée sur ce segment. Pour prendre un exemple simple, la figure I.2 montre comment une ligne de relief permet de 'raboter' ou de 'relever' le terrain sur son passage.

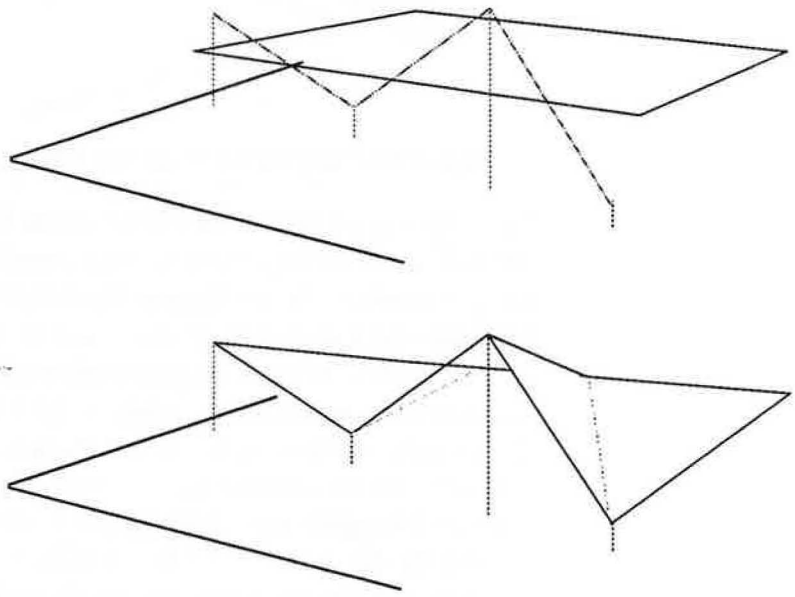


Figure I.2. Influence d'une ligne de relief.

Sur le haut de la figure on trouve une ligne de relief constituée de trois segments consécutifs le long de l'axe des x, perpendiculairement à l'axe des ordonnées. On a aussi représenté un 'carreau de terrain', symbolisé par un section pratiquement plane. A la verticale du passage de cette ligne de relief, l'altitude des points du carreau de terrain est abaissée ou relevée, suivant les circonstances, alors que les autres bords du terrain restent inchangés.

A partir de maintenant, le terme 'à la verticale' doit se comprendre comme 'parallèlement à la direction de projection sur le plan de référence': cette direction est celle des cotes (altitudes), ou axe des z.

Chaque chaîne de relief (qui peut être réduite à un segment, mais reste toujours connexe) permet de réaliser des travaux de gros-œuvre (aplanissement de terrain, vallées en U, remblais, déblais, etc).

I.B.4. Cadastre

Aspect et forme

La modélisation de paysages exige que l'on puisse représenter les terrains à la fois du point de vue de leur géométrie et d'autre part du point de vue de leur aspect physique.

Chaque objet qui va être présenté à partir de maintenant aura une texture propre. Cette texture permet d'identifier le type d'objet modélisé (champs, forêts, routes, étendues d'eau, rivières, bâtiments). Lorsqu'il n'y a *rien* quelque part (!), le domaine sous-jacent est apparent et sa texture correspond à ce que l'on appellera la *friche*. Cette texture est la seule qui soit définie par défaut. Elle dépend évidemment du contexte: la friche autour d'une île a vraisemblablement la texture d'une étendue d'eau, alors que la friche autour d'une oasis a plutôt la texture d'un sol désertique, etc.

Les textures sont connues de l'application sous forme de 'bibliothèques' et il est essentiel que les faces obtenues à la fin de la facétisation réfèrent toujours exactement à l'élément de bibliothèque auquel référait initialement l'objet donné en entrée.

Tout domaine, en plus des extrema et des chaînes de relief, contient une série d'objets polygonaux simples pour matérialiser les formes les plus générales des éléments constituant un paysage.

Les objets polygonaux ne constituent pas forcément un pavage du domaine courant, car ils ont le droit de se chevaucher afin de permettre une interface plus conviviale. Malgré tout, le domaine contient tous les objets polygonaux, et le complémentaire de l'union de ceux-ci est simplement la friche.

Les lois qui régissent l'intersection des objets polygonaux seront présentées un peu plus loin.

Champs

Les champs (ce terme désigne en fait un polygone fermé et simple autorisé à contenir d'autres éléments) constituent la majeure partie des zones non urbanisées. Un champ est représenté par un polygone tridimensionnel, que l'on désignera par le nom de *f*-polygone (*flexible polygons*). Chaque sommet de la frontière d'un *f*-polygone est un point fixe, dont l'altitude est calculée sur la grille altimétrique.

La texture de chaque *f*-polygone permet de refléter le type d'activité qui l'accompagne (culture, végétation, place publique, mer, etc).

Bâtiments

Les constructions humaines (maisons, bâtiments, ensembles d'habitation ou zones d'activité) sont créées à l'aide du modeleur 3D. En ce qui concerne le module de facétisation, ces objets sont des polygones, que nous appelerons *h*-polygones (*hard polygons*), dotés de propriétés très particulières et restrictives.

En premier lieu, ils ont, comme les *f*-polygones, des sommets de frontière fixes. Mais il se comportent comme des trous: les faces qui sont attachées aux objets de cette classe, proviennent toutes du modeleur 3D et ne sont pas manipulées par le module de facétisation. L'intérieur d'un *h*-polygone est donc résolument vide: il ne peut rien contenir. Sa frontière ne doit pas être remise en cause par d'autres objets car les facettes volumiques délivrées par le modeleur 3D ne peuvent pas prévoir d'autres points sur la base que ceux qui ont été calculés à la construction, bien avant la facétisation d'un domaine.

Ceci est lié au fait qu'un même objet 'dur' peut servir à plusieurs endroits dans un ou plusieurs domaines. Si l'on accepte des intersections entre ces objets et d'autres d'un domaine, on s'expose à l'apparition de T dans le voisinage des points d'intersection indésirables: un tel T provoque des incohérences entre les sommets de facettes prévus (en sortie du modeleur terrain) et ceux 'livrés' par le module de facétisation.

L'altitude des *h*-polygones ne se calcule pas comme celles des *f*-polygones. Pour comprendre pourquoi, examinons la figure I.3.

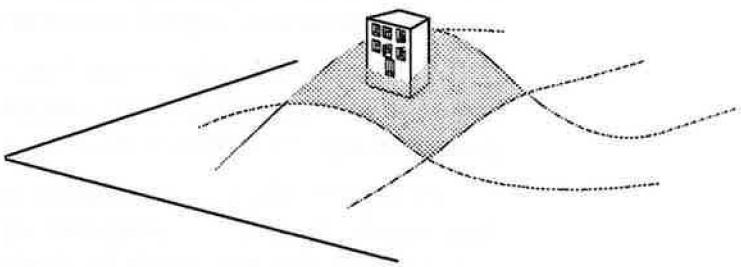


Figure I.3. La maison dans la montagne...

On désire *implanter* (c'est le mot) un bâtiment sur un pan de colline. Les règles habituelles de la maçonnerie imposent que les plans définissant le volume du bâtiment utilisent les trois directions principales de la base de l'espace, alors que les lois qui régissent la géologie de la colline n'ont que faire du fil à plomb! Une solution consiste à 'enfoncer' la maison suffisamment profondément pour être certain que les quatre sommets les plus bas du bâtiment de la figure soient bien tous en-dessous du point localement le plus haut de la colline.

Une autre solution, qui est celle retenue ici, est de considérer un point spécifique, dit point de *pose*, qui soit cohérent avec l'altitude moyenne au voisinage du bâtiment, et qui impose son altitude à tous les sommets définissants la frontière du *h*-polygone de base. Ainsi, le terrain sous-jacent (la colline) se 'moule' sur le contour du polygone de base du bâtiment.

Arbres

Il existe des *h*-polygones spéciaux qui n'ont pas de socle polygonal, mais seulement un point de pose. Les arbres en sont un exemple. Mais il est aussi possible qu'un même objet soit considéré avec ou sans son socle polygonal: quand une scène est vue de très loin, le niveau de détail qui l'accompagne est assez grossier, et il suffit que les maisons, par exemple, soient symbolisées uniquement par leur point de pose.

En tout état de cause, les *h*-polygones sans socle se rangent résolument dans leur classe mère, mais leur influence sur la géographie du domaine est identique à celle des extrema altimétriques. Les facettes relatives à ces objets, si elles existent, proviennent du modeleur 3D, ce qui explique leur classification parmi les *h*-polygones.

Routes, rivières, etc...

Pour représenter les routes et les rivières, en général les objets de largeur presque invariante et de très grande longueur, il a été choisi d'utiliser un dernier type de polygones, d'une gestion particulièrement délicate: ce sont les *s*-polygones (*snake polygons*). Ces objets sont définis par des segments de contrôle dont les extrémités ont des altitudes calculées sur la grille altimétrique et par une largeur fixe (s). On 'expanse' ensuite ces segments de façon à obtenir un polygone (quand la construction est possible) de largeur constante $2s$. On obtient ainsi une bande polygonale de points qui, en projection sur le plan de référence, sont tous situés exactement à la distance s d'un point au moins du segment de définition auquel il est attaché.

Si l'on tire une perpendiculaire en un point d'un segment de définition de *s*-polygone hors d'un virage, on trouve deux points à égale distance, dont l'altitude est celle du point choisi. Les arêtes constituant la frontière de ces objets s'appellent des *génératrices*. La figure I.4 montre un exemple de *s*-polygone.

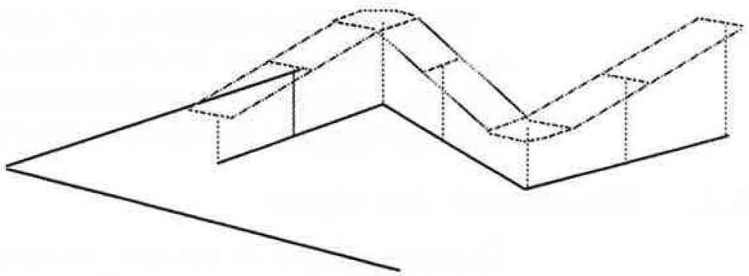


Figure I.4. *s*-polygone et projections d'un point sur les génératrices.

La figure suggère que pour résoudre les problèmes posés par les 'virages', on fasse appel à des paliers horizontaux; ceux-ci nécessitent que l'on modifie quelque peu l'altitude des *s*-polygones autour des points de rotation et de tronquer les arcs de cercle représentant les points à distance s de ceux-ci en figures 'vectorisées'. Il est important de se rappeler que les *s*-polygones ont des sections droites de très grande longueur (entre 50 et 1000 mètres pour donner un ordre de grandeur), sont très fins (quelques mètres seulement) et donc que les modifications suggérées par la figure sont assez peu visibles.

Les virages des *s*-polygones sont problématiques, mais leur intersection avec d'autres objets est encore plus complexe, comme on le verra sur certaines figures plus loin.

I.B.5. Priorités

Pour que tous ces objets, dont on a déjà dit qu'ils provenaient de source *a priori* déconnectées, puissent coexister en harmonie, il est nécessaire d'établir entre eux des *priorités*. On associe donc à chaque objet un vecteur numérique dont la première composante est la classe de l'objet. Par delà l'ordre dans lequel ces classes sont rangées, leur priorité relative dépend de l'opération en cours. Ceci sera explicité plus loin.

La deuxième composante du vecteur de priorité est le rang dans la classe. Deux objets d'une même classe ont des priorités différentes. Par exemple, une route est forcément moins prioritaire qu'une voie ferrée, une route nationale qu'une autoroute, etc.

Pour le cas où plusieurs objets de même classe et de même rang se couperaient (ceci est parfois dû au fait que l'utilisateur a laissé ce détail dans l'ombre), un autre champ, qui contient le numéro d'ordre dans la liste des données, permet de lever toute ambiguïté et d'affirmer que deux objets de la base ne peuvent jamais avoir le même vecteur de priorité. Ainsi, on a une relation d'ordre strict totale sur l'ensemble des objets d'un domaine, et donc sur les ensembles de segments ou de sommets qui les composent. Par conséquent, si deux segments s'intersectent et ont le même vecteur de priorité, ils proviennent de la même frontière polygonale, ce qui normalement interdit.

Ceci permet aussi de codifier les autorisations et les interdictions relatives aux objets: on pourra interdire que telle classe d'objet (ou tel objet de tel rang dans une classe) chevauche, soit adjacent ou occulte tel autre objet. On aura ainsi un code complet des opérations licites, gênantes ou inhibantes.

Toute opération inhibante est signalée à l'utilisateur afin qu'il y remédie après l'arrêt forcé de l'exécution. Les opérations gênantes (illicites, mais pas inhibantes) sont aussi signalées afin que l'utilisateur statue sur leurs cas, mais l'exécution n'est pas interrompue.

I.B.6. Sémantique des objets

Ce terme désigne le fait que, suivant leurs classes, les objets d'un domaine interagissent de manière contrôlée par un ensemble de règles liées au sens de l'opération visée pour les deux objets.

Par exemple, **si** une route rencontre le bord du domaine **et** si cette éventualité coïncide avec une extrémité finale de chaîne de définition, **alors** il faut prolonger le profil de la route jusqu'à épouser parfaitement le bord de domaine correspondant. En revanche, **si** la même route rencontre le bord du domaine en un point constituant un virage, **alors** il peut être nécessaire de vérifier si le domaine voisin peut accepter ce phénomène et de terminer la route de manière différente.

De même, si une extrémité de route se termine en plein milieu de la friche ou d'un champ, on arrête le *s*-polygone perpendiculairement au dernier segment de définition, alors qu'en revanche, on peut désirer l'arrêter différemment si le dernier point de définition de la route coïncide avec un point de définition d'un contour polygonal.

Il y énormément de cas à considérer, et il n'est pas question de les lister ici. On peut se permettre de parler de sémantique parce que l'analyse de chaque cas est faite en fonction du sens que revêt chaque opération, selon des règles qui varient elles aussi en fonction des circonstances dans lesquelles elles sont appliquées.

Prenons, par exemple, le cas de l'intersection d'une ligne de relief et d'un *s*-polygone: si l'intersection a lieu avant l'expansion de ce dernier, la ligne de relief plaque son altimétrie sur le segment de définition du *s*-polygone qu'elle intersecte. En revanche, si une telle intersection est constatée après que les *s*-polygones ont été expansés, les lignes de relief ne doivent plus (pour des raisons qui seront explicitées plus loin) rencontrer les frontières de ces objets: on doit inhiber l'opération et la déclarer gênante.

De même, l'adjacence de deux objets, l'inclusion d'un objet dans un autre, peuvent être parfois licites, parfois illicites, suivant le contexte dans lequel celles-ci ont lieu.

I.C. RESTRUCTURATION DES DONNEES

Le premier travail à effectuer est de retrouver une organisation logique dans cet ensemble non structuré de données. On doit procéder dans l'ordre suivant:

1. organiser les données polygonales et les données de relief par listes de sommets et d'arêtes;
2. ranger chaque liste parmi les éléments de sa classe, à la place indiquée par son rang;
3. intersecter les segments de définition des *s*-polygones et les lignes de relief;
4. expanser les *s*-polygones;
5. découvrir toute intersection entre deux segments ou un segment et un sommet isolé, toutes catégories confondues;
6. reconstruire les faces de manière cohérente en interdisant tout chevauchement;
7. retrouver toutes les inclusions entre les objets du domaine, celui étant, en principe, l'objet maximal au sens de l'inclusion.
8. calculer les altitudes définitives.

Donnons quelques explications concernant certaines de ces étapes. L'étape 3 est liée au fait que l'influence d'une ligne de relief pourrait avoir des conséquences fâcheuses sur le profil d'un *s*-polygone. Prenons le cas de la figure I.5.

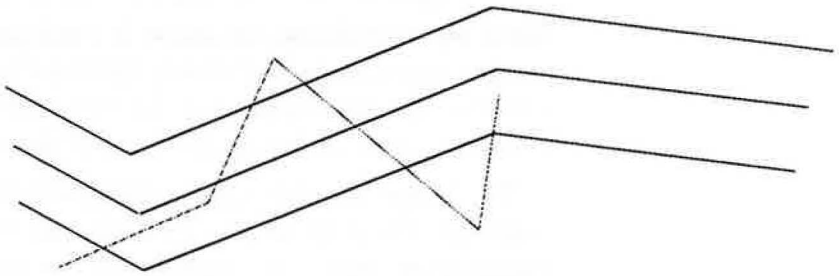


Figure I.5. Intersection d'une ligne de relief et d'un *s*-polygone expansé.

Autant l'influence des deux points d'intersection entre le segment de définition central du *s*-polygone et de la chaîne de relief paraît cohérente, autant le premier et le dernier segments de la chaîne risquent de faire perdre la planéité des tronçons de *s*-polygone qu'ils rencontrent. On décide donc, pour éliminer tous les problèmes, de procéder à une pré-intersection entre chaînes de relief et segments de définition de *s*-polygones seulement, de projeter un point sur chaque génératrice correspondante afin de lui faire partager l'altitude interpolée du même point d'intersection, et d'ignorer, après expansion des *s*-polygones, toute intersection entre les chaînes de relief et les segments de frontière expansée des *s*-polygones. Ainsi, le rôle des lignes de relief est limité à un voisinage local de ces intersections et les résultats obtenus sont bien plus contrôlables que dans une situation plus laxiste.

L'étape 5 est primordiale, car c'est elle qui permet de retrouver entièrement toutes les arêtes utiles du domaine. L'utilisateur a le droit légitime de faire chevaucher des surfaces destinées à être seulement adjacentes, de manière à être sûr qu'elles partageront réellement des arêtes, quelle que soit la complexité des objets. Par exemple, puisqu'il ne contrôle pas vraiment le profil expansé des *s*-polygones dont il ne peut, d'ailleurs, donner que les segments de définition et la largeur, il est naturel, pour lui, d'attacher les frontières de champs qui bordent une route aux sommets de définition des *s*-polygones. Ceci provoque naturellement, après expansion de ces derniers, des intersections non prévisibles et des chevauchements de surfaces polygonales qu'il faut démêler en appliquant des règles précises.

Ceci est l'objet de l'étape 6. La priorité qui accompagne les objets est naturellement un facteur de décision primordial dans cette opération. D'une manière générique, un objet n'a pas le droit de franchir la frontière d'un objet plus prioritaire et sa frontière doit être détournée au point d'intersection de manière à épouser la frontière de l'objet plus prioritaire (s'il en a jamais été à l'extérieur!) jusqu'à retrouver un second point d'intersection, donc de sortie du polygone plus prioritaire.

La septième étape consiste à entériner toute inclusion d'un objet dans un autre et à construire une structure modélisant la hiérarchie complète des objets d'un domaine. Cette étape permet de distribuer définitivement les attributs propres à chaque face (bibliothèques de textures en particulier) selon la nouvelle 'géométrie' de chacune.

La huitième étape peut surprendre, pourtant elle est vitale, comme va le montrer la figure I.6.

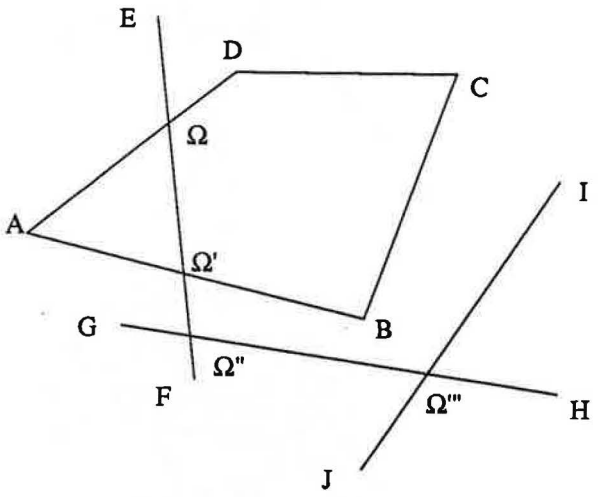


Figure I.6. Réseau de dépendances altimétriques.

Sur cette figure, on trouve un *f*-polygone ABCD et trois segments de relief [E,F], [G,H], [I,J], par ordre croissant de priorité. Les points d'intersection découverts pendant l'étape 2 sur la frontière de ABCD dépendent, en altitude, de [E,F] qui est plus prioritaire qu'un *f*-polygone.

Ces points peuvent être interpolés sur E mais pas F, car Ω' dépend de Ω" qui dépend, à son tour, du segment plus prioritaire [G,H]; de même Ω", peut être interpolé sur G, mais pas sur F car l'altitude de Ω" dépend, sur le côté droit, de Ω'", qui est lui-même interpolé sur [I,J]: on trouve un réseau de dépendance altimétrique, qui, en l'occurrence, se résout uniquement si l'altitude de Ω'" est calculée, puis celle de Ω" et enfin celles de Ω' et Ω.

On pourrait croire qu'une telle situation est très rare, mais il n'en est rien car les objets manipulés (en particulier les routes qui créent de longues chaînes de dépendances) sont très souples et provoquent énormément de situations bien plus inextricables que la précédente.

La figure I.7 illustre l'évolution de quelques données pendant les étapes précédentes. En haut à gauche, on trouve un *f*-polygone ABCD, deux *s*-polygones EBCF et DCF, et enfin un segment de relief HI.

A droite, on teste l'intersection de ce dernier avec les segments de définition des *s*-polygones. En-dessous, à gauche, on expanse les *s*-polygones et le segment de relief est considéré comme 'mort'.

Chaque sommet de définition d'un *s-polygone* donne naissance à deux projections à une distance *s*; les virages se traitent, en première approximation, par prolongement et rencontre des deux génératrices de deux tronçons consécutifs.

A droite, on en est à l'étape d'intersection. On découvre un grand nombre de points de rencontre entre les objets (tous ne sont pas indiqués sur la figure).

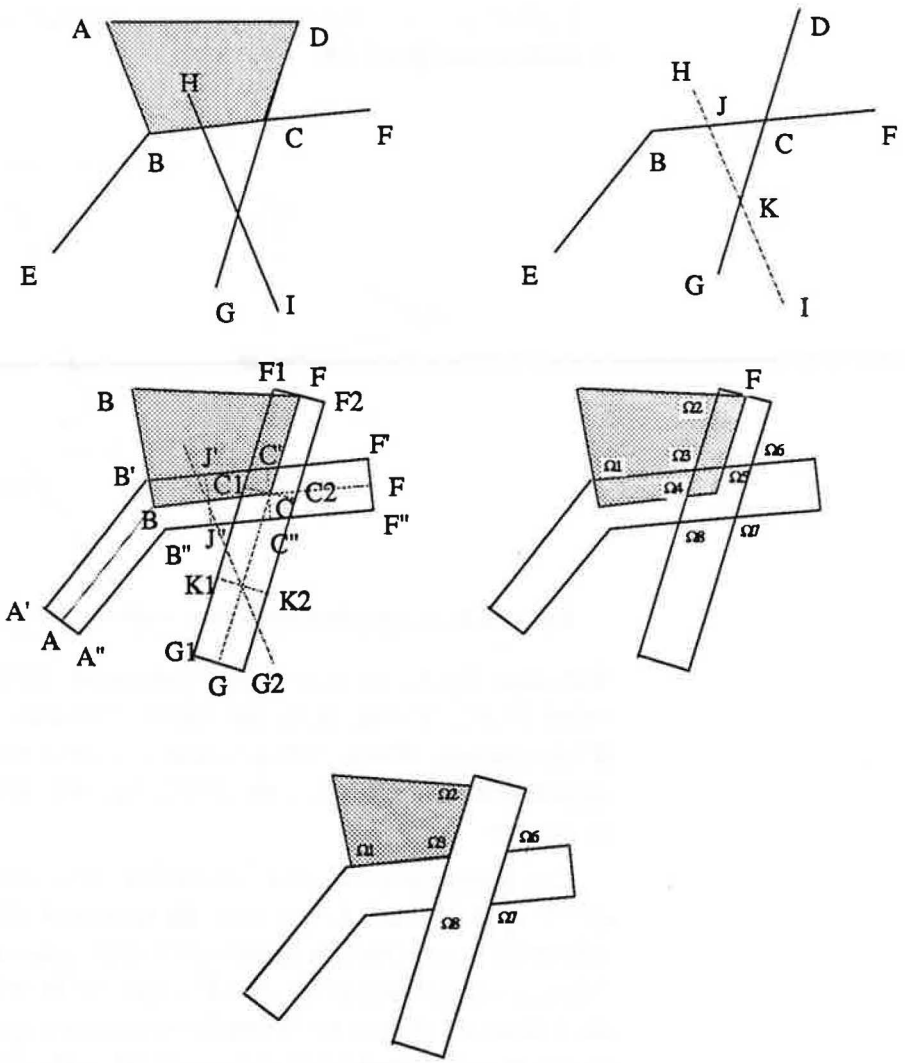


Figure I.7. Evolution de quelques données.

Sur le dernier graphique, on entérine les priorités entre les objets: on voit que le *s*-polygone en un seul tenant est le plus prioritaire, puis vient l'autre et enfin le *f*-polygone qui est réduit et est maintenant adjacent aux deux *s*-polygones sur deux arêtes.

Cette opération de reconstruction prioritaire des faces est violente: le *s*-polygone le moins prioritaire se voit scindé en deux composantes connexes totalement disjointes, dont il ne faudra pas oublier, au moment de la triangulation, qu'elles partagent la même bibliothèque d'attributs.

L'étape 7, de construction de la hiérarchie, n'est pas représentée ici, car la hiérarchie est très simple: tous les objets sont inclus dans le domaine au même niveau.

L'étape 8 nécessite ici un grand soin, car beaucoup de points sont en dépendance éloignée de sommets originaux et constituent des chaînes de dépendance qu'il faut résoudre de manière rigoureuse, sans jamais 'tourner en rond'.

Toutes ces opérations sont analysées sous un mode théorique plus général dans les chapitres qui suivent celui-ci.

I.D. FACETISATION

Après que les données ont été rendues cohérentes et organisées en structure hiérarchique, il faut encore effectuer l'opération la plus délicate de toutes: la décomposition de toutes les faces du terrain représenté en triangles licites.

En effet, ce qui fait l'efficacité d'une machine de visualisation est sa capacité à remplir vite les faces polygonales avec les textures appropriées, mais aussi à les afficher de manière à représenter fidèlement la perspective recherchée: puisque tous les polygones manipulés jusqu'ici sont tri-dimensionnels, il faut allumer chaque pixel de l'écran avec la couleur correspondant au polygone tri-dimensionnel vu en cet endroit, c'est-à-dire avec la couleur propre au point de texture correspondant du polygone le plus proche de l'observateur, pour l'angle de visée choisi.

Il est bien connu que ces deux problèmes sont d'autant plus facilement résolus que les polygones à afficher sont simples: des convexes au minimum, des triangles dans le cas idéal.

L'idée est donc de trianguler chaque polygone du terrain afin de faciliter la tâche de visualisation. Mais la difficulté réside dans le fait que les triangles doivent remplir deux conditions: ne pas être trop grands ni trop petits et surtout correspondre à des éléments cohérents du terrain représenté.

Pour prendre un exemple simple: un triangle dont les trois sommets seraient sur trois champs différents est inacceptable, de même qu'un triangle dont une arête traverserait la route plus prioritaire de f-1.7 pour joindre deux sommets de la moins prioritaire situés de part et d'autre.

En un mot comme en quinze, on interdit tout triangle qui coupe une frontière quelconque ou une chaîne de relief, ou qui soit partagé par plusieurs faces.

Comme on demande aussi que les triangles soient les plus 'ramassés' possible pour des raisons propres aux algorithmes de rendu et d'antialiassage, on est amené à choisir un type de triangulation particulière, appelée triangulation de *Delaunay*, dont une des propriétés, et non des moindres, est de rendre aussi grand que possible le plus petit de tous les angles de tous les triangles qu'elle contient: de proche en proche, ceci étant vrai de toute sous-triangulation de la triangulation de Delaunay, on s'aperçoit qu'une triangulation de ce type est unique et permet de choisir les triangles les plus équi-angulaires qu'il soit possible de former parmi tous les triangles potentiels.

Le couplage des deux contraintes (maximisation d'angle minimum et interdiction de franchir des segments spécifiés à l'avance) porte le nom de triangulation de Delaunay *contrainte* (par les segments spécifiés).

Comme l'ensemble des données a été hiérarchisé, la triangulation contrainte demandée doit, en plus, discerner la face dans laquelle un triangle est localisé. La figure I.8 illustre la triangulation des données de f-I.7 relativement au domaine rectangulaire dans lequel elles sont incluses.

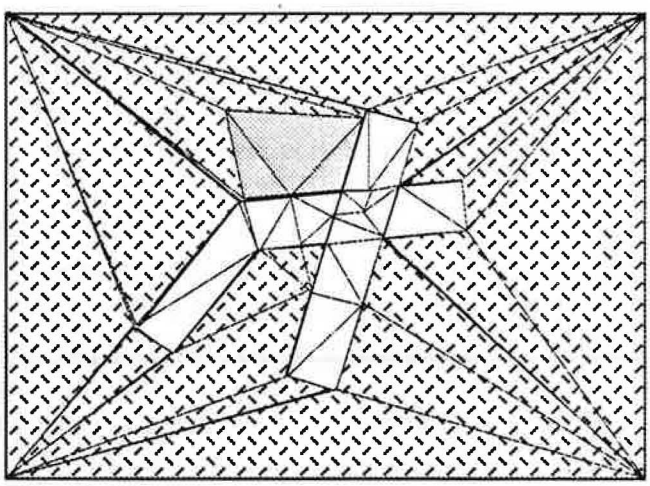


Figure I.8. Triangulation à partir des éléments de *f*-I.7.

On constate l'apparition, malgré toutes ces précautions, de très petites facettes au voisinage du virage de la route la moins prioritaire, et des facettes à l'intérieur de la seconde route, toutes liées à des intersections détectées, mais par forcément désirables. De nombreux problèmes de ce genre se présentent dans une triangulation automatique. Certains éléments de réponse à ces questions seront apportés dans les développements de cette thèse.

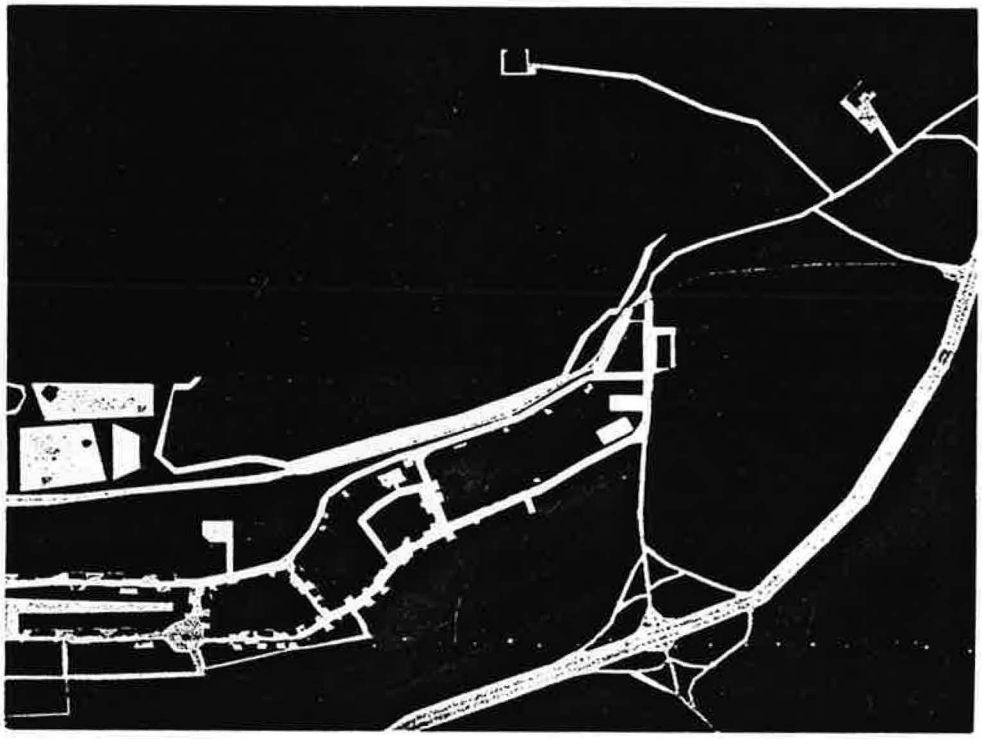
I.E. QUELQUES ILLUSTRATIONS

Quatre photographies permettront d'illustrer le travail présenté ici et qui a sous-tendu les développements à venir. Je voudrais, à cette occasion, remercier chaleureusement Jean-Louis Beau, de SOGITEC-Boulogne, pour avoir fort gentiment pensé à mettre de côté et à m'adresser ces clichés. Ils sont extraits d'une pré-étude de base de données urbaine.

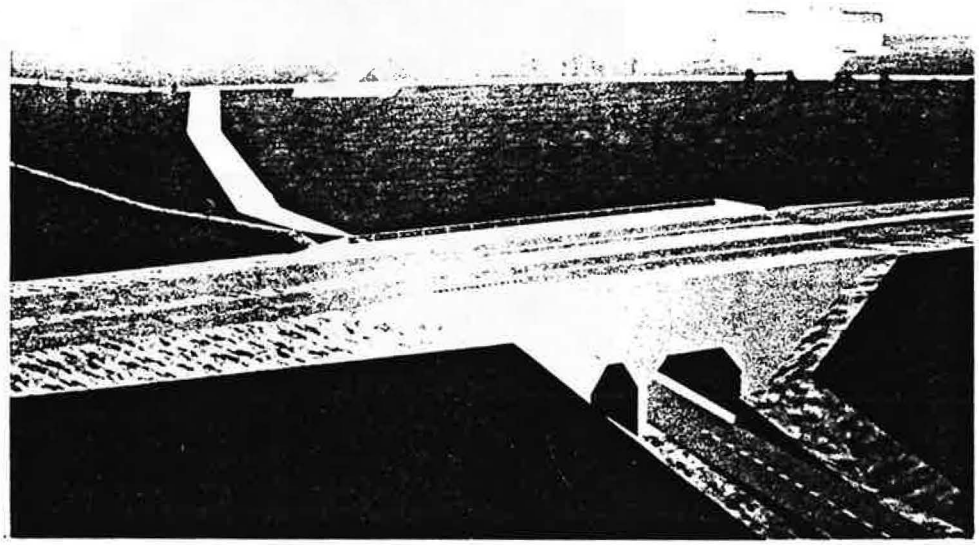
Le lecteur qui serait habitué à des images de synthèse beaucoup plus 'réalistes' doit simplement se rappeler que la vitesse d'affichage de ces images correspond à du temps réel et qu'un simulateur de vol ou de conduite en ville n'a guère le temps d'attendre que les calculs relatifs aux algorithmes de rendu sophistiqués soient terminés: l'avion serait déjà en vrille, et la voiture aussi, autour d'un platane, bien que réaliste à s'y méprendre et joliment antialiassé... Les post-traitements apportés aux images sont déjà remarquables, vu le peu de temps disponible: je le dis d'autant plus objectivement que je n'y suis pour rien...

La première photographie est une vue aérienne du site. On remarque la gestion des priorités entre les différents objets: autoroute, route nationale et chemin de terre. Le réseau routier est géré par des priorités propres et celles-ci permettent d'attribuer des textures spécifiques (bandes centrales) ou de déterminer des priorités comme on les entend couramment pour des voies de communication.

On remarque aussi, sur la gauche, plusieurs niveaux d'imbrication d'inclusion (berge $\supset$ champ $\supset$ eau) et, sur la droite, certains endroits ou les triangles choisis dans la triangulation sont assez visibles.

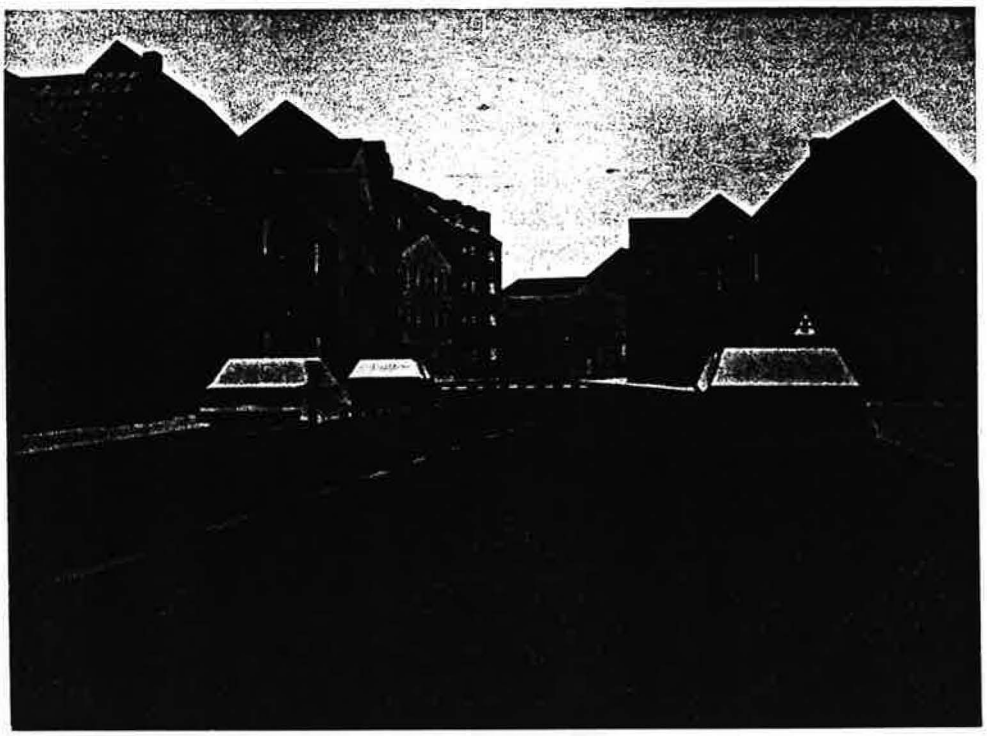


Cliché 1. Vue aérienne.



Cliché 2. Vue rapprochée de la scène précédente.

Cliché 3. Le village.



Cliché 4. Une rue. Bonjour chez vous...

Sur la deuxième photographie, on s'est rapproché sensiblement de la jonction la plus à droite du cliché précédent. On distingue mieux la différence entre les textures du chemin de terre, des routes secondaires, de l'autoroute, de la friche et des talus.

Les arbres sont des *h*-polygones sans socle avec point de pose et les maisons des *h*-polygones avec socle et point de pose. Les lignes de relief sont aussi évidentes dans l'apparence des talus au premier plan.

Un troisième niveau de détail est montré sur la reproduction suivante, qui représente une scène du village dont on distingue la place centrale sur la première vue. La texture des voies devient plus visible. On voit nettement que les maisons sont 'posées' sur le sol.

On trouve pratiquement tous les types d'objets sur cette vue, sauf peut-être les chaînes de relief... On voit notamment que l'application gère bien les adjacences d'objets dont on sait, par ce qui précède, qu'ils sont définis par chevauchement. La petite place giratoire est un exemple d'objets avec point de pose (les panonceaux) à l'intérieur d'un *f*-polygone bordant un *s*-polygone en anneau sur lequel débouche au moins deux *s*-polygones de largeur différente.

La dernière photographie montre un gros plan d'une rue de la ville avec des véhicules.

I.F. QUELQUES DIFFICULTES

L'une des premières difficultés rencontrées dans ce travail a été la résolution des problèmes de précision. Pratiquement, le premier essai a abouti à un échec, dû à l'imprécision des machines.

Comme la dynamique des données va du millimètre à la dizaine de kilomètres, il est vite devenu important de trouver une solution fiable, garantissant une cohérence totale avec la topologie globale des données, et efficace, c'est-à-dire donnant des temps d'exécution acceptables.

Pour donner un ordre de grandeur, la facétisation au niveau de détail le plus fin d'une base de données comme celle de la première vue s'accompagnait d'un temps de calcul allant jusqu'à plusieurs dizaines de minutes en arithmétique exacte; les techniques suggérées dans cette thèse ont permis de faire redescendre le temps d'exécution en-dessous de cinq minutes.

A titre d'indication plus précise, on peut donner deux temps d'exécution assez typiques, sur Iris 4D20. Les deux domaines à facétiser ont une superficie de 1km^2. Le premier contient 22 *f*-polygones (186 sommets), 6 *s*-polygones (27 sommets), 19 *h*-polygones (35 sommets) et 2 lignes de relief (29 sommets); sa facétisation a pris 43 secondes, pour l'obtention de 334 facettes triangulaires.

Le second domaine présente une densité d'objets plus forte: il contient 49 *f*-polygones (400 sommets), 17 *s*-polygones (92 sommets), 40 *h*-polygones (215 sommets) et 0 lignes de relief; sa facétisation a pris 6'43", pour l'obtention de 1120 facettes triangulaires.

Le deuxième problème délicat à résoudre est la gestion des priorités et des dépendances entre les données. Il fallait arriver à travailler sur des classes d'objets et des sous-ensembles de ces classes. Les fonctions sont notamment écrites de manière à travailler sur des sous-ensembles d'objets qu'elles traitent différemment suivant leurs particularités sémantiques et les circonstances.

Pour donner un exemple précis, le module d'intersection est capable de gérer plusieurs phases (intersection avant expansion des *s*-polygones, intersection après expansion, etc) et, dans chaque phase, 'réagit' différemment suivant les types d'objets qui s'intersectent.

La priorité des objets est une notion variable dans le temps: la frontière du domaine est infranchissable lors de l'intersection, mais doit complètement céder le terrain à tout polygone rencontré lors de la reconstruction des faces.

Ainsi, un segment de relief est plus prioritaire qu'un segment de définition de *s*-polygone dans la première phase d'intersection, mais c'est le contraire dans la deuxième.

Chaque objet est susceptible d'avoir des comportements évolutifs: alors qu'il semblait interdit, au départ d'avoir des objets inclus dans l'intérieur des *s*-polygones, la présence d'îlots, de signalisations ou ... de véhicules, demande l'évolution de cette règle. D'une manière générale, les modules de gestion des priorités et des dépendances ont dû être conçus de manière à permettre les évolutions et la relaxation des contraintes.

La principale difficulté, dans ce domaine, est certainement venue du fait qu'il n'était pas possible, pour des raisons matérielles diverses, d'utiliser une méthodologie d'intelligence artificielle pure. Le lecteur avisé reconnaîtra certainement, ici et là, des éléments de solution qui ne sont pas loin d'appartenir à ce champ d'étude. Je suis personnellement persuadé qu'au stade d'évolution où l'on est arrivé, seules des techniques propres à l'intelligence artificielle permettront de faire encore progresser les compétences acquises.

En dernier lieu, la Grande Difficulté est venue de la compréhension du comportement des *s*-polygones. Dans la version actuelle, ceux-ci sont encore soumis à des contraintes prophylactiques sévères: on empêche qu'ils débordent du domaine car la gestion de domaines multiples est encore impossible. Les virages sont encore traités de manière rudimentaire et le redécoupage des tronçons trop longs permettant, en principe, d'éviter les facettes trop effilées, n'est pas vraiment adaptatif, car on éprouve les plus grandes difficultés à contrôler de manière efficace la taille des facettes résultantes, *avant facétisation.*

Certaines solutions proposées dans cette thèse correspondent à des réflexions sur les problèmes constatés avec l'expérience. Le volume du travail qui était demandé nécessitait de régler un nombre assez important de problèmes *en priorité*. J'ai maintenant la conviction que l'on ne fait qu'entrer dans le vif du sujet!...

I.G. PROSPECTIVE

Le projet SINDBAD, dans son entier, est opérationnel dans plusieurs sites, notamment aux Etats-Unis. Il dépasse certes, par son envergure, le simple domaine d'étude présenté ici. Malgré tout, le module de facétisation est un organe vital de l'ensemble, comme le sont d'ailleurs tous les autres: le projet n'aurait jamais vu le jour si la G.I., le modeleur 3D, l'éditeur de faces, etc... n'existaient pas!

L'évolution naturelle du projet semble pouvoir prendre deux directions principales: d'une part, il devient nécessaire de pouvoir conférer plus de sémantique aux objets et d'autre part, le besoin de rendre plus locale l'action de la facétisation devient impérieux.

SINDBAD est, sauf erreur de ma part, le seul outil de création de bases de données d'images capable d'accepter la définition des objets qu'il manipule par un système de règles évoluées. Les autres applications permettent seulement la description figée des objets définitifs.

Ceci leur donne un 'avantage' que n'a pas SINDBAD: si une facette doit être retouchée pour une raison quelconque, elle peut l'être dans une application concurrente sans risquer de perturber la logique de l'ensemble, à condition de modifier 'à la main' le voisinage perturbé par les modifications apportées. Ici, toute modification, pour être entérinée, doit être validée au niveau sémantique le plus bas, afin que les dépendances soient mises à jour *automatiquement* et que l'ensemble retrouve toute la cohérence nécessaire. Le terme 'avantage' est placé entre guillemets, car il devrait être évident après les explications données plus haut, que c'est la sophistication même de SINDBAD qui, pour l'instant, l'empêche de disposer des moyens plus rudimentaires d'un créateur de bases de données pour terrains dépourvu de sémantique!

Il devient alors impératif de permettre une refonte d'une partie des données sans avoir à reconstruire les zones qui, de toute évidence, ne doivent pas être remaniées. Ceci nécessite le développement de techniques de triangulations contraintes limitées à des zones spécifiques, abordées dans la dernière section de cette thèse, et de méthodes plus fines de modélisation hiérarchisée. Ces points sont primordiaux et sont, sans doute aucun, des axes de recherche très prometteurs.

II. De l'Art de Couper les Segments en Quatre

La détermination de l'intersection de deux segments est un phénomène beaucoup plus complexe que le calcul du point de rencontre de deux droites infinies dans le plan. La principale raison est que les positions relatives de deux segments sont en bien plus grand nombre que celles de leurs supports.

On dénombre six classes principales de situations:

- les deux segments sont totalement disjoints,
- ils s'intersectent en un point distinct de leurs quatre extrémités,
- ils se coupent en un point confondu avec une de leurs extrémités,
- ils sont exactement superposables,
- ils se chevauchent partiellement,
- l'un des deux est entièrement inclus dans l'autre.

La reconstruction d'un graphe planaire à partir d'un ensemble de points, de segments et éventuellement de polygones nécessite de déterminer sans ambiguïté la position relative de chaque couple de segments donnés initialement.

Il est indispensable que la procédure de détection d'intersection soit la plus économe possible, du point de vue du nombre d'opérations et notamment de multiplications, et ce d'autant plus que les problèmes de précision imposent le recours à une arithmétique exacte, fort coûteuse.

Bien qu'élémentaire, l'analyse des techniques d'intersection de deux segments mérite qu'on lui consacre un chapitre, ne serait-ce que pour rendre encore plus pertinents les choix opérés dans le traitement des problèmes de précision.

. REFERENCES BIBLIOGRAPHIQUES POUR LE CHAPITRE II .

A/H/U 74 A. Aho, J. Hopcroft, J. Ullman, *The Design and Analysis of Computer Algorithms*, Addison Wesley Publishing Company, 1974.

B/OT 79 J.L. Bentley, T. Ottman, *Algorithms for Reporting and Counting Geomeric Intersections*, IEEE Trans. Comp. C-28, 9, pp. 643-647, Sept. 1979.

II.A. INTERSECTION DE DEUX DROITES COPLANAIRES

Les positions relatives que sont susceptibles d'occuper deux droites d'un même plan sont au nombre de trois: elles peuvent être *strictement parallèles*, *sécantes*, ou *confondues*.

Si χ est le cardinal de l'ensemble $D \cap D'$ des points communs aux deux droites, on peut caractériser la position relative des deux droites coplanaires[1] en utilisant l'algorithme élémentaire suivant:

```
si (χ=∅)
   D et D' sont strictement parallèles;
sinon
  si (χ≠1)
     D et D' sont confondues;
  sinon
     D et D' sont sécantes;
```

La logique élémentaire de cet algorithme induit celle de l'algorithme communément accepté pour la détermination de l'intersection de deux segments. Malheureusement, cette logique ne permet pas de trouver l'algorithme le plus économique.

Les coordonnées (x,y) d'un point quelconque de la droite D passant par deux points distincts A et B de coordonnées respectives (x_A,y_A) et (x_B,y_B), vérifient le système:

$$\begin{cases} x = x_A + \mu(x_B - x_A) \\ y = y_A + \mu(y_B - y_A) \\ \mu \in \mathbf{R} =]-\infty,+\infty[\end{cases} \qquad \text{(II-1)}$$

Si l'on élimine μ entre les deux premières équations, on trouve l'équation paramétrique bien connue de la droite D:

$$(x_B - x_A)y - (y_B - y_A)x + x_A y_B - x_B y_A = 0.$$

Pour deux droites D_1 et D_2 coplanaires, passant respectivement par les couples de points (A_1,B_1) et (A_2,B_2), on trouve deux ensembles de relations de type (II-1):

$$\begin{cases} x = x_{A_1} + \mu_1(x_{B_1} - x_{A_1}) \\ y = y_{A_1} + \mu_1(y_{B_1} - y_{A_1}) \\ \mu_1 \in \mathbb{R} \end{cases} \qquad \begin{cases} x = x_{A_2} + \mu_2(x_{B_2} - x_{A_2}) \\ y = y_{A_2} + \mu_2(y_{B_2} - y_{A_2}) \\ \mu_2 \in \mathbb{R} \end{cases}$$

qui se ramènent, après substitution, à un système de deux équations linéaires à deux inconnues réelles μ_1 et μ_2:

$$\begin{cases} \mu_1(x_{B_1} - x_{A_1}) - \mu_2(x_{B_2} - x_{A_2}) = x_{A_2} - x_{A_1} \\ \mu_1(y_{B_1} - y_{A_1}) - \mu_2(y_{B_2} - y_{A_2}) = y_{A_2} - y_{A_1} \\ \mu_1, \mu_2 \in \mathbb{R} \end{cases} \qquad \text{(II-2)}$$

[1] Ce terme sera implicite dans l'ensemble du chapitre et ne sera pas forcément répété.

Pour déterminer la position relative des deux droites, on calcule alors les quantités Δ (déterminant principal du système) et $\Delta\mu_1$, $\Delta\mu_2$ (déterminants secondaires), définies par les relations:

$$\Delta = \begin{vmatrix} x_{B_1}-x_{A_1} & x_{A_2}-x_{B_2} \\ y_{B_1}-y_{A_1} & y_{A_2}-y_{B_2} \end{vmatrix} = (x_{B_1}-x_{A_1})(y_{A_2}-y_{B_2}) - (x_{A_2}-x_{B_2})(y_{B_1}-y_{A_1})$$

(II-3)

$$\Delta\mu_1 = \begin{vmatrix} x_{A_2}-x_{A_1} & x_{A_2}-x_{B_2} \\ y_{A_2}-y_{A_1} & y_{A_2}-y_{B_2} \end{vmatrix} = (x_{A_2}-x_{A_1})(y_{A_2}-y_{B_2}) - (x_{A_2}-x_{B_2})(y_{A_2}-y_{A_1})$$

$$\Delta\mu_2 = \begin{vmatrix} x_{B_1}-x_{A_1} & x_{A_2}-x_{A_1} \\ y_{B_1}-y_{A_1} & y_{A_2}-y_{A_1} \end{vmatrix} = (x_{B_1}-x_{A_1})(y_{A_2}-y_{A_1}) - (x_{A_2}-x_{A_1})(y_{B_1}-y_{A_1})$$

Les deux cas suivants se présentent:

Δ est nul. Les deux droites ont la même direction (*pente*) et sont donc strictement parallèles ou confondues.

Si $\Delta\mu_1$, $\Delta\mu_2$ sont tous deux nuls, chaque équation s'obtient en multipliant l'autre par un réel; autrement dit les deux droites sont confondues.

Si, au contraire, l'une de ces deux quantités n'est pas nulle les droites sont strictement parallèles.

Δ n'est pas nul. Les deux droites se coupent en un point unique $\Omega(x_\Omega,y_\Omega)$. Le système (II-2) admet comme solution unique le couple de réels

$$\mu_1 = \frac{\Delta\mu_1}{\Delta} \text{ et } \mu_2 = \frac{\Delta\mu_2}{\Delta} \text{ tels que}$$

(II-4)

$$x_{A_1} + \mu_1(x_{B_1} - x_{A_1}) = x_{A_2} + \mu_2(x_{B_2} - x_{A_2}) = x_\Omega$$
$$y_{A_1} + \mu_1(y_{B_1} - y_{A_1}) = y_{A_2} + \mu_2(y_{B_2} - y_{A_2}) = y_\Omega$$

Au vu de ces résultats, dès lors que Δ est non nul, les coordonnées de Ω peuvent être obtenues par le simple calcul de $\Delta\mu_1$. Il est donc *inutile d'évaluer* $\Delta\mu_2$ pour déterminer la position relative de deux droites et/ou les coordonnées de leur point d'intersection éventuel, ceci *quelles que soient les droites choisies.*

II.A.1 Le coût du calcul d'intersection de deux droites

Les calculs précédents font exclusivement intervenir des opérations élémentaires sur l'ensemble des nombres pseudo-réels des machines (additions, soustractions, multiplications, divisions et tests conditionnels).

Il est intéressant de chiffrer le coût d'un algorithme en fonction du nombre moyen d'opérations nécessaires à son exécution. Les calculs du module précédent peuvent se récrire sous une forme plus compacte, faisant apparaître d'une part les opérations requises et leur nombre, et d'autre part les répétitions éventuelles dans les calculs; ces répétitions seront entérinées dans une implantation par l'utilisation de variables pour valeurs intermédiaires qui permettent de ne calculer qu'une seule fois des expressions ou valeurs rencontrées à plusieurs endroits!

Dans la modélisation suivante, les symboles • représentent les coordonnées initiales des points; chaque opération arithmétique est représentée par son initiale (S=soustraction, A=addition, M=multiplication, D=division), est accompagnée d'un numéro d'ordre dans sa propre classe et est exprimée sous forme polonaise inverse: la deuxième opération du genre $x_{A1}+x_{A2}$ serait représentée par [• • A2]. Enfin, le résultat d'une opération déjà effectuée est représenté par l'identificateur correspondant, cette-fois-ci débarrassé de ses opérandes, présents dans sa définition originelle. Ainsi, les calculs deviennent:

Δ = [[• • S 1] [• • S 2] M1] [[• • S 3] [• • S 4] M2] S5

$\Delta\mu_1$ = [[• • S 6] S2 M3] [S3 [• • S 7] M4] S8

μ_1 = [$\Delta\mu_1$ Δ D1]

x_Ω = [• [μ_1 S1 M5] A1]

y_Ω = [• [μ_1 S4 M6] A2]

A titre d'exemple, le calcul de x_Ω implique la multiplication n°5 dont les opérandes sont le résultat μ_1 de la division D1 et celui de S1 (effectuée lors du calcul de Δ).

On peut résumer ces considérations dans le tableau suivant où sont comptabilisés tous les coûts impliqués dans le calcul des coordonnées du point Ω lorsqu'il existe, par classes d'opérations:

	¿ +- */
Calcul de Δ	0, 5, 2
$\Delta=\emptyset$?	1, 0, 0
Calcul de $\Delta\mu_1$	0, 3, 2
Calcul de μ_1	0, 0, 1
Calcul de x_Ω	0, 1, 1
Calcul de y_Ω	0, 1, 1
Total:	1,10, 7

Il va s'avérer utile de trouver une mesure commune aux algorithmes proposés. Cette mesure se trouve facilement dans le décompte moyen du nombre d'additions (ou soustractions), multiplications (ou divisions) et de tests.

Une telle classification est naturellement dictée par la complexité intrinsèque des opérations [A/H/U 74, KII] et par le fait que les tests sont, parmi toutes les opérations, celles qui permettent entre autres de 'baliser' les algorithmes et d'effectuer des études comparatives.

Afin de 'noter' les algorithmes étudiés, il est pratique d'utiliser un modèle simplifié d'*arbre de décision* [A/H/U 74]. Il s'agit, tout au moins pour les besoins présents, d'un arbre binaire dans lequel les conventions suivantes sont adoptées:

- tout nœud terminal (*feuille*) porte un numéro distinct; il contient aussi le résultat de l'algorithme à cet endroit: un échec (il n'y a pas d'intersection) se traduit par le symbole { }, un succès se traduit par la lettre symbolisant le point d'intersection trouvé;

  Dans le cas de deux droites infinies, quand elle existe, l'intersection est notée Ω et est dite *normale*; dans le cas de deux segments [A,B] et [C,D], l'intersection est encore dite normale et notée Ω si le point de rencontre est distinct des quatre extrémités; mais dans le cas contraire elle est symbolisée par la lettre représentant l'extrémité où se rencontrent les deux segments, et est qualifiée de *remarquable*.

- tout nœud non terminal indique un choix, exprimé par une condition logique; en outre, un tel nœud possède exactement deux *fils*: le fils gauche indique toujours une réponse *positive* et le fils droit, une réponse *négative* au test contenu dans le nœud.

- chaque opération élémentaire est supposée porter sur des réels (ou de manière mixte sur des entiers et des réels) et se voit attribuer une *unité de coût* (udc) maximale: on considère donc que la machine travaille toujours aussi 'lentement' et l'utilisation d'entiers ne sera pas considérée comme une optimisation.

- chaque nœud synthétise le coût total des opérations impliquées depuis le test précédent (sauf pour la racine) jusqu'à l'expression du test qui lui est relatif. Par exemple, la notation

  <expr1> = <expr2> ∴ T,A,M

  exprime le fait que le test courant fait intervenir les expressions expr1 et expr2 et que les calculs se montent à un total (y compris le test) de T udc pour les tests, A udc pour les additions et de M udc pour les multiplications.

  Bien évidemment, si expr1 et expr2 font intervenir d'autres expressions calculées, ces coûts sont aussi comptabilisés dans la valeur relative au nœud.

- chaque feuille synthétise le coût opératoire cumulé depuis la racine de l'arbre jusqu'à elle. En général, chaque apparition du symbole Σ permet de connaître le coût opératoire impliqué dans le trajet depuis la racine jusqu'au nœud où il se trouve.

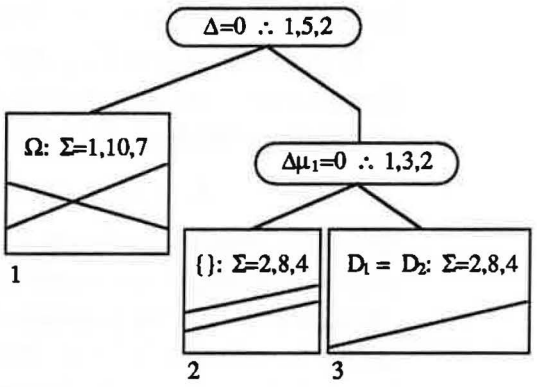
- tout test est considéré sous l'angle du 'pire des cas': le coût de son échec et de son succès sont identiques, même s'il est possible dans certaines circonstances d'optimiser un test en le morcellant en plusieurs sous-tests dont certains ne sont pas toujours exécutés. Ce qui est souhaitable n'est pas l'optimisation liée au langage[2] ou à une astuce ponctuelle, mais bien l'optimalité d'une méthode en regard des autres, quels que soient les langages, circonstances ou autres considérations non structurelles.

[2] Il est bien connu que le langage C permet de ne pas faire effectuer tous les tests d'une condition multiple comme **if** (cond1 || cond2) ..., alors que le Pascal effectue tous les tests, quelle que soit la situation.

- deux critères de comparaison sont retenus:

  le coût *moyen* pour chaque opération élémentaire; ce dernier se calcule tout naturellement en faisant la somme, pour une opération donnée, des coûts relatifs à chaque feuille de l'arbre puis en divisant cette valeur par le nombre de feuilles.
  le coût *normal* (opération par opération) de la branche conduisant à la solution dite normale.

Comme exemple, l'arbre binaire relatif à l'algorithme pour la résolution du problème de l'intersection de deux droites infinies est présenté à la figure II.1, accompagné des coûts moyens et du coût normal; ces résultats serviront de point de repère par la suite.



COUTS MOYENS		COUT NORMAL
(1+2+2)/3 ≈ 1,667	TESTS	1
(10+8+8)/3 ≈ 8,667	ADDITIONS	10
(7+4+4)/3 = 5	MULTIPLICATIONS	7

Figure II.1. Arbre de décision et coûts relatifs à l'intersection de 2 droites.

Le lecteur se pose peut-être la question: quelles seront les performances d'un algorithme de détection d'intersection pour deux segments, comparées à ces résultats? Etonnement, sauf en ce qui concerne les tests, l'algorithme final aura d'assez bonnes performances normales, et ... de bien meilleures performances moyennes!

L'algorithme analysé jusqu'ici est bien sûr construit sur une vérité digne de Monsieur de La Palisse: si l'on sait que deux droites coplanaires ne sont pas parallèles, elles se coupent nécessairement!

Si l'on applique naïvement ce précepte, il paraît donc naturel de tester, avant toute chose, si deux segments coplanaires sont parallèles, puis d'analyser les conséquences de la réponse...

Le problème est que la relation entre le parallélisme des deux segments et leur capacité à s'intersecter *normalement* (en un point intérieur aux deux segments) est assez faible et relativement coûteuse à exploiter.

II.B. INTERSECTIONS DE DEUX SEGMENTS

Etant donné un segment [A,B] dans le plan, on appelle *support* de ce segment la droite infinie, notée (AB), qui le porte. Les *positions relatives* que peuvent occuper deux segments coplanaires sont malheureusement plus complexes à qualifier qu'auparavant: que leurs supports soient confondus ou sécants, les deux segments peuvent encore n'avoir aucun point en commun.

II.B.1 Un peu d'ordre

On peut instaurer une relation d'ordre total, appelé *ordre lexicographique* sur l'ensemble des points et des segments du plan. L'idée est semblable à celle utilisée pour la classification de noms dans un dictionnaire, d'où le terme 'lexicographique'.

Soient $A(x_A,y_A)$ et $C(x_C,y_C)$ deux points du plan. On note $A <_L C$ et l'on dit que A est (strictement) *à gauche lexicographiquement* de C si les coordonnées de A et C vérifient:

(II-5) $$A <_L C \Leftrightarrow (x_A < x_C \vee (x_A = x_C \wedge y_A < y_C)),$$

où "<" symbolise la relation d'ordre (strict) naturelle sur l'ensemble des réels. On définit d'une manière identique la relation 'à droite lexicographiquement'. Si l'on souhaite inclure l'égalité entre les deux points on utilise la relation '≤' sur les réels et l'on dit que 'A est à gauche lexicographiquement de C au sens large'.

On peut étendre la définition précédente à deux segments [A,B] et [C,D] en posant:

(II-6) $$[A,B] <_L [C,D] \Leftrightarrow (A <_L C \vee (A = C \wedge B <_L D))$$

On vérifie aisément que ces relations sont des relations d'ordre total. Dire qu'un couple (U,V), où U et V sont tous deux des points ou des segments du plan, est *donné dans l'ordre lexicographique* signifie que $U <_L V$, la relation ayant un sens non ambigu dans chaque cas.

II.B.2 Paramètres contre puissances

Soient deux segments [A,B] et [C,D] du même plan. On suppose que $A <_L B$, $C <_L D$ d'une part, et $[A,B] <_L [C,D]$ de l'autre. Dans ce qui suit, tout segment est supposé *non dégénéré*, c'est-à-dire que ses extrémités sont distinctes.

L'ordre imposé aux segments permet de limiter de manière significative le nombre de configurations possibles sans en supprimer; par exemple, sur la figure II.2, les situations de gauche sont interdites car relatives à des points (et/ou des segments) ne vérifiant pas la relation d'ordre lexicographique imposée, mais sont traitées sous la forme qu'elles revêtent dans la colonne de droite.

Un autre bénéfice de ces contraintes est présenté à la figure II.3, où l'on constate que, quand B=C, le point D ne peut se situer à gauche lexicographiquement de la droite verticale passant par B et C et donc que les deux segments n'ont que cette extrémité en commun.

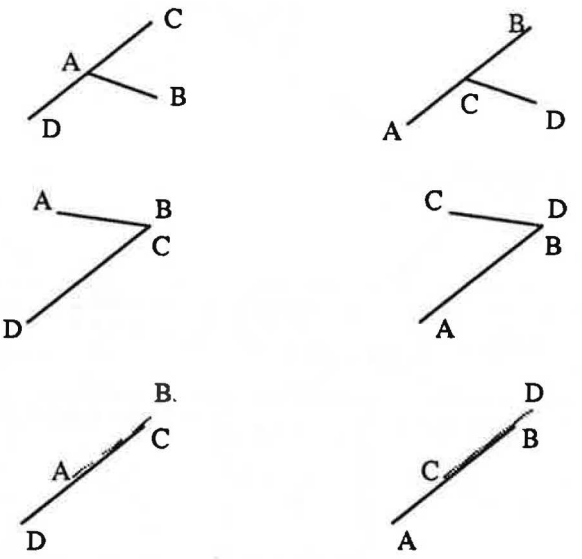


Figure II.2. L'ordre lexicographique réduit le nombre de configurations.

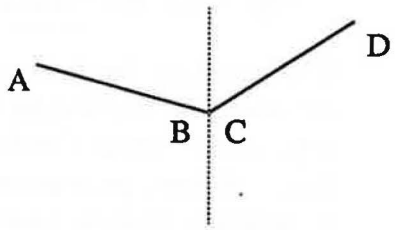


Figure II.3. L'ordre lexicographique élimine certain cas d'intersection.

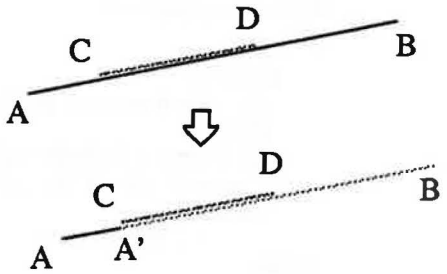


Figure II.4. L'ordre lexicographique simplifie certains cas: les deux points d'intersection pourront être traités indépendamment.

Ces restrictions ont enfin pour effet de simplifier le traitement du cas où l'un des deux segments est entièrement inclus dans l'autre (figure II.4). Le segment original [A,B] est décomposé en [A,A'] et [A',B] (A'=C); ce découpage respecte l'ordre lexicographique et le second point d'intersection est traité lorsque [C,D] et [A',B] sont confrontés.

Un point commun

Les deux algorithmes décrits dans la fin ce chapitre partagent un même souci: puisque certaines solutions remarquables existent et font intervenir les extrémités, et puisque l'on peut aisément faire le tour de ces cas particuliers en n'utilisant que des tests, mieux vaut se débarrasser de cette tâche en premier.

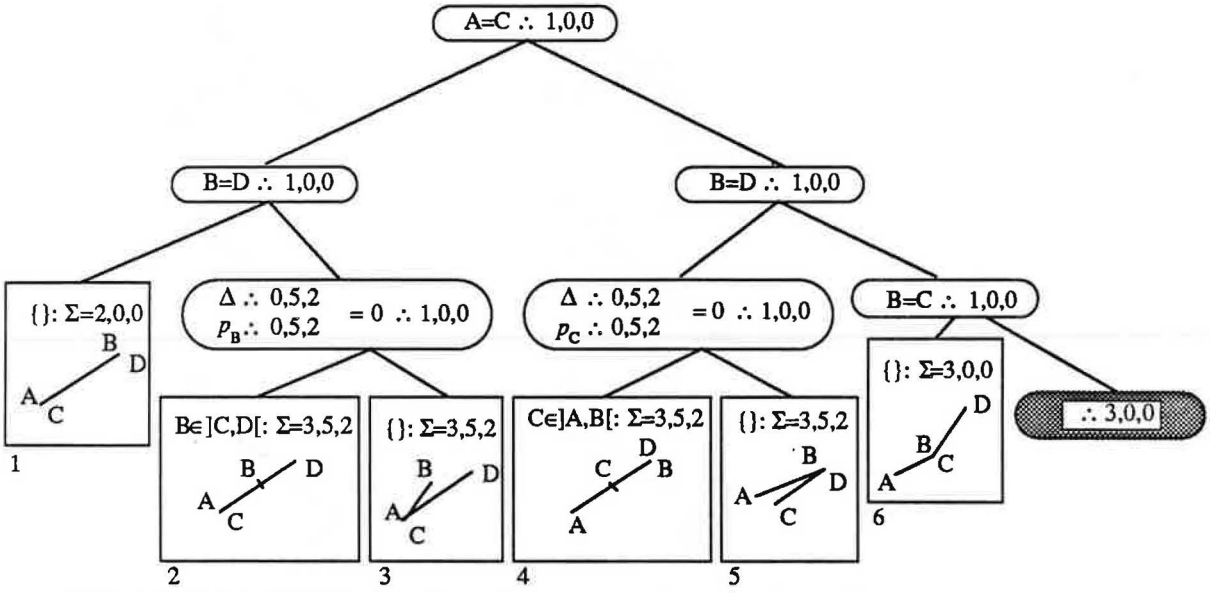


Figure II.5. Base commune de l'arbre de décision.

C'est pourquoi les deux algorithmes présentés ont la même base d'arbre de décision, représenté sur la figure ci-dessus: le nœud en grisé représente le point d'embranchement de la suite de l'algorithme et il est bien vrai que, pour parvenir à cet endroit, 3 tests sont nécessaires, à l'exclusion de toute autre opération. A part deux d'entre eux, tous les autres nœuds internes de cet arbre portent sur des cas de coïncidence entre extrémités. Cet arbre partiel matérialise graphiquement le début d'algorithme suivant:

```
si (A=C)
   si (B=D)
     pas d'intersection, [A,B] et [C,D] sont confondus;
    sinon
     si (B est sur la droite (CD))
       on a: [C,D] = [C,B] ∪ [B,D] et A=C;
     sinon
       pas d'intersection;
sinon
   si (B=D)
     si (C est sur la droite (AB))
       on a: [A,B] = [A,C] ∪ [C,B] et B=D;
     sinon
       pas d'intersection;
    sinon
     si (B=C)
       pas d'intersection;
     sinon
       ...
```

Aux lignes '**si** (X est sur la droite (YZ))' de l'algorithme correspondent deux nœuds semblables dans l'arbre *f*-II.5. Les tests relatifs à ces nœuds expriment de deux façons différentes, qui seront explicitées un peu plus loin, les deux conditions d'appartenance à un support présentes dans l'algorithme. Le coût des deux tests est (1;5;2).

Les points de suspension de l'algorithme coïncident avec le nœud grisé du graphique et c'est là où l'étude va maintenant reprendre.

La pente fatale

L'algorithme présenté en premier portera le nom d'*algorithme de la pente* (ou éventuellement méthode paramétrique) car il s'inspire directement de la logique de l'algorithme précédent.

Les paramètres μ_1 et μ_2 utilisés dans la modélisation paramétrique de l'intersection de deux segments $[A_1,B_1]=[A,B]$ et $[A_2,B_2]=[C,D]$ sont des réels de l'intervalle [0,1], liés par le système:

$$\begin{cases} \mu_1(x_{B_1}-x_{A_1})-\mu_2(x_{B_2}-x_{A_2}) = x_{A_2}-x_{A_1} \\ \mu_1(y_{B_1}-y_{A_1})-\mu_2(y_{B_2}-y_{A_2}) = y_{A_2}-y_{A_1} \\ \mu_1, \mu_2 \in [0,1] \end{cases} \quad \text{(II-7)}$$

Les déterminants principaux et secondaires – Δ, $\Delta\mu_1$ et $\Delta\mu_2$ – de ce système sont définis comme en (II-3).

Revenons à la figure II.5: les deux tests laissés dans l'ombre font intervenir le déterminant principal. Le fait qu'il soit nul exprime bien dans un cas que le point B appartient à la droite (CD) et dans l'autre que le point D appartient à la droite (AB). Les coûts mentionnés correspondent bien aux évaluations déjà faites précédemment.

Lorsque l'algorithme nécessite, une fois l'évaluation du déterminant principal faite, le calcul des deux déterminants mineurs, on doit effectuer 3 additions et 2 multiplications pour le premier et enfin 1 addition et 1 multiplication pour le dernier.

Notons à l'avance que, si $\Delta \neq 0$, la condition nécessaire et suffisante d'existence d'une solution (i.e. $0 \leq \mu_1 \leq 1 \wedge 0 \leq \mu_2 \leq 1$) peut se traduire plus économiquement (car sans division!) par la condition

$$(0 \leq |\Delta\mu_1| \leq \Delta \wedge 0 \leq |\Delta\mu_2| \leq \Delta)$$

qui elle-même est traitée de manière optimale par la séquence suivante

si $(\Delta > 0)$
  si $((0 \leq \Delta\mu_1 \leq \Delta) \wedge (0 \leq \Delta\mu_2 \leq \Delta))$ ... **sinon** ...
sinon
  si $((\Delta \leq \Delta\mu_1 \leq 0) \wedge (\Delta \leq \Delta\mu_2 \leq 0))$ ... **sinon** ...

où l'on dénombre cinq tests, 'dans le pire des cas'. On considérera toujours cette éventualité, de sorte que les deux algorithmes aient le nombre minimal de feuilles et que tous les cas soient envisagés.

Le lecteur intéressé pourra vérifier que le fait de scinder la condition initiale en deux nœuds en cascade (comme pourrait le suggérer le **et** logique) entraînerait nécessairement un accroissement des coûts. Quoi qu'il en soit, les coûts relatifs à la réalisation de cette condition se montent à (5;4;4).

L'arbre de la figure II.6 représente l'analyse que l'on peut entreprendre pour classifier toutes les situations rencontrées, en aval de celui de *f*-II.5. On peut dénombrer 8 feuilles, parmi lesquelles les quatre feuilles 7, 10, 11 et 12 correspondent à des configurations remarquables. Ces situations sont des éléments clés dans la caractérisation géométrique du problème et constituent autant de points stables de comparaison d'une méthode à l'autre.

La logique de l'algorithme peut se décomposer en deux grandes directions, comme le confirment les commentaires suivants:

- Si Δ est nul, les deux segments peuvent être strictement parallèles ou partager le même support;
  si $\Delta\mu_1$ est nul, les deux segments partagent le même support: on sait déjà que C est à droite lexicographiquement de A; pour qu'il appartienne au segment ouvert]A,B[il faut et suffit que le point C soit strictement à gauche de B; quant à D, si jamais il se trouve aussi à l'intérieur de]A,B[, on sait que son cas sera traité ultérieurement.
- Si Δ est différent de 0, on calcule immédiatement les deux déterminants mineurs et l'on passe aux tests éliminatoires relatifs aux valeurs absolues de μ_1 et μ_2. Seule la branche gauche apporte une chance de succès.
  Il reste les trois solutions remarquables à détecter: celle où B est inclus dans]C,D[(μ_1=1 $\Leftrightarrow$ $\Delta\mu_1$=Δ), celle où C est inclus dans]A,B[(μ_2=0 $\Leftrightarrow$ $\Delta\mu_2$=0) et celle où D est inclus dans]A,B[(μ_2=1 $\Leftrightarrow$ $\Delta\mu_2$=Δ).
  Quand tous les cas particuliers ont été éliminés, on obtient la solution normale, correspondant à la feuille 13. A l'entrée de cette feuille, les coûts sont (12;9;6). Il reste alors (cf (II-4)) les calculs de μ_1 (1 division), x_Ω et y_Ω (1 addition et 1 multiplication chacun). Ceci conduit bien au coût total de (12;11;9) qui accompagne cette feuille.

Quelques commentaires...

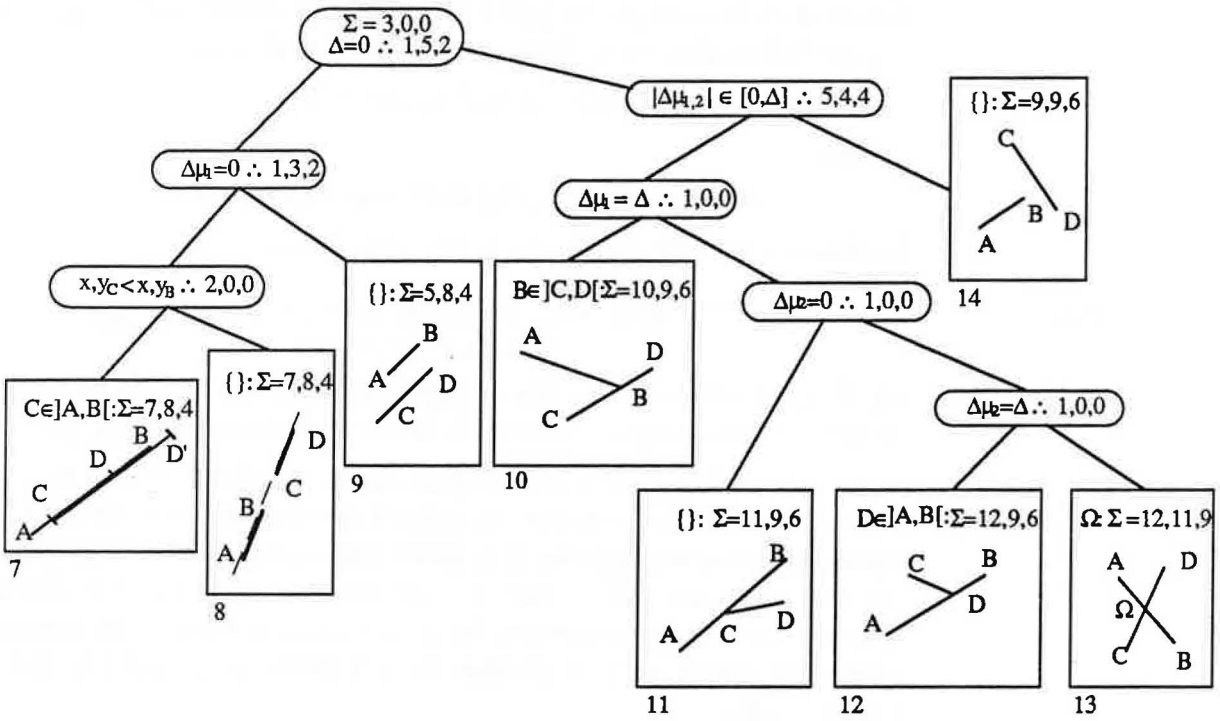
D'une manière évidente, l'algorithme proposé ici a un coût normal optimal si l'on excepte le coût relatif aux tests: il est vrai que l'on utiliserait moins de tests si l'on calculait tout de suite les trois déterminants, μ_1 et les coordonnées de Ω, quand ce point existe: on obtiendrait alors un coût normal de (4,11,9).

Mais inversement, imputer aux six feuilles de l'arbre de base un coût obligatoire de (4,11,9) serait pour le moins maladroit et se répercuterait lourdement sur le coût moyen! Comme de toute façon les deux algorithmes considérés devront adopter ce compromis, il vaut bien mieux faire comme si ces quatre tests de base faisaient partie de la famille!

Somme toute, le coût moyen de cet algorithme ne fait pas mauvaise figure quand on le compare à celui de l'algorithme d'intersection de deux droites. Il s'agit certainement de l'algorithme le plus employé pour résoudre le problème considéré, sous cette forme ou sous une autre.

On se souvient que la logique de cet algorithme provient de celle qui gouvernait la détection de l'intersection de deux droites: si elles ont des pentes différentes (Δ=0), elles sont sécantes! Or la complexité des cas susceptibles de se présenter et leur nombre, que les deux segments soient ou *ne soient pas* parallèles (Δ=0 ou non) affaiblissent, **statistiquement**, l'espoir d'information décisive en retour du test sur la valeur de Δ.

Qui plus est, quoi de plus inefficace, du point de vue de la précision, que de se poser des questions sur le parallélisme de deux segments lorsque ceux-ci sont éventuellement 'séparés' par une distante suffisante pour leur interdire de pouvoir se couper de toute façon? N'est-ce pas là, au contraire, le meilleur moyen pour rajouter des problèmes de précision?



COUTS MOYENS		COUT NORMAL
90/14 ≈ 6,428	TESTS	12
91/14 ≈ 6,5	ADDITIONS	11
53/14 ≈ 3,785	MULTIPLICATIONS	9

Figure II.6. Arbre de décision et coûts de l'algorithme 'de la pente'.

Il sera déjà assez difficile d'obtenir une précision satisfaisante dans la caractérisation du point d'intersection pour se permettre de rajouter des imprécisions relatives à des quantités dont l'influence directe sur la solution n'est, statistiquement, que bien relative.

Il est d'autre part manifeste que les feuilles 7 et 11 reflètent une certaine redondance: dans les deux cas, le point C est inclus dans le segment ouvert]A,B[. N'est-il pas possible de les regrouper, puisque, même si D est inclus dans le segment ouvert, ce cas n'est pas traité ici?

Il serait utile d'exploiter l'information concernant l'influence de chaque extrémité sur la configuration générale. Pour prendre un exemple simple, il *semblerait* bien plus décisif de découvrir qu'un segment est entièrement inclus dans l'un des deux demi-plans que détermine le support du second (ceci éliminant toute possibilité d'intersection), que d'apprendre que les deux segments ne sont pas parallèles!

C'est cette tâche qui sera confiée à l'algorithme suivant, dans l'espoir que, en repoussant le plus loin possible les calculs liés jusqu'ici au parallélisme, voire en les inhibant, les nœuds *lourds* (en calculs) vont descendre vers les feuilles de l'arbre, un mouvement susceptible d'alléger sensiblement le coût moyen.

Distance et puissance d'un point par rapport à une droite

Soient A $<_L$ B deux points distincts du plan, de coordonnées (x_A,y_A) et (x_B,y_B). L'équation de la droite passant par A et B étant

$$(y-y_A)(x_B-x_A) - (x-x_A)(y_B-y_A) = 0,$$

posons

$$p_{M/(AB)} = (y_M-y_A)(x_B-x_A) - (x_M-x_A)(y_B-y_A).$$

La *distance* du point M à la droite (AB) est égale à

(II-8)
$$d(M,(AB)) = \frac{|p_{M/(AB)}|}{\sqrt{(x_B-x_A)^2+(y_B-y_A)^2}} = K_{AB}.|p_{M/(AB)}|$$

où K_{AB} est une constante liée au segment [A,B]. On appelle *puissance du point* $M(x_M,y_M)$ *par rapport à la droite* (AB) la quantité $p_{M/(AB)}$.

La droite (AB) définit trois régions: un demi-plan formé de tous les points de puissance positive, un second demi-plan formé de tous les points de puissance négative et la droite elle-même, dont tous les points ont une puissance nulle. Grâce à la condition A $<_L$ B, on peut décider sans aucune ambiguïté que tous les points ayant une puissance positive (négative, nulle) sont 'à gauche de' ('à droite de', 'sur') la droite orientée $(\overrightarrow{AB})$.

Si M et N sont deux points du plan de la droite D, on dira que M et N sont *de part et d'autre* de D si et seulement si les puissances $p_{M/D}$ et $p_{N/D}$ sont non nulles et de signes contraires.

Selon cette terminologie, les deux segments [A,B] et [C,D] se coupent en un point Ω distinct de leur quatre extrémités si et seulement si A, B sont situés de part et d'autre de la droite (CD) et si C, D sont situés de part et d'autre de la droite (AB).

Chaque cas particulier où une extrémité au moins constitue le point de rencontre des deux segments se caractérise par une valeur nulle d'une des puissances relatives aux extrémités A, B, C et D.

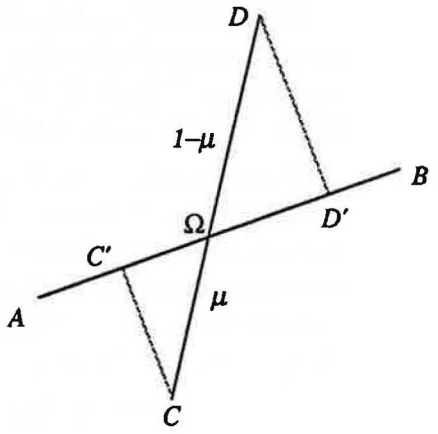


Figure II.7. Utilisation de la puissance d'un point par rapport à une droite pour le calcul de l'intersection de deux segments.

Détermination du point d'intersection

Soit Ω le point de rencontre unique des segments [A,B] et [C,D]. Convenons de noter respectivement p_A, p_B les puissances de A, B par rapport à (CD) et p_C, p_D celles de C, D par rapport à (AB).

Supposons que ces quantités aient été évaluées et que l'on désire déterminer le paramètre $\mu = \mu_2$, permettant de localiser Ω sur le segment [C,D] (voir figure II.7). Si C' et D' sont les projections de C et D sur la droite (AB) et si CC' et DD' représentent les longueurs des segments [C,C'] et [D,D'], on peut écrire:

$$\frac{\mu}{CC'} = \frac{1-\mu}{DD'} = \frac{1}{CC'+DD'} \Leftrightarrow \mu = \frac{CC'}{CC'+DD'} = \frac{|p_C|}{|p_C|+|p_D|}$$

(II-8)
$$\mu = \frac{p_C}{p_C-p_D} \in [0,1];\ \mu = 0 \Leftrightarrow p_C=0;\ \mu = 1 \Leftrightarrow p_D=0$$

puisque, pour qu'il y ait intersection en un point, p_C et p_D ne peuvent être nulles en même temps ni avoir le même signe. Des expressions similaires lient μ_1, p_A et p_B.

Marchandages...

Les formules suivantes font apparaître les répétitions que l'on peut mettre à profit dans le calcul des quatre puissances et les coûts successifs:

$$p_C = (x_B-x_A)(y_C-y_A) - (x_C-x_A)(y_B-y_A) \qquad (0;5;2)$$
$$p_D = \boxed{x_B-x_A}(y_D-y_A) - (x_D-x_A)\boxed{y_B-y_A} \qquad (0;3;2)$$
$$p_A = (x_D-x_C)\boxed{y_A-y_C} - \boxed{x_A-x_C}(y_D-y_C) \qquad (0;3;2)$$
$$p_B = \boxed{x_D-x_C}(y_B-y_C) - (x_B-x_C)\boxed{y_D-y_C} \qquad (0;3;2)$$

Le coût global est (0;14;8)! En fait, p_B peut s'obtenir plus simplement:

$$p_B = p_A - p_D + p_C \qquad (0;2;0)$$

mais on constate que le calcul des trois premières puissances présente déjà un coût trop important. Il pourrait s'avérer 'payant' de ne pas chercher à calculer *directement* celles-ci, mais de les obtenir par le biais de combinaisons linéaires. Il a déjà été dit que l'on pouvait obtenir le second paramètre, μ_2, à partir des seules deux premières puissances, p_C et p_D. Ceci conduit à imaginer les calculs suivants:

$$p_C = (x_B-x_A)(y_C-y_A) - (x_C-x_A)(y_B-y_A) \qquad (0;5;2)$$
$$p_C-p_D = \boxed{x_B-x_A}(y_C-y_D) - (x_C-x_D)\boxed{y_B-y_A} \qquad (0;3;2)$$

Le coût de ces opérations est (0;8;4). On remarque que les tests p_C=0 et p_D=0 se déduisent sans nouveaux calculs. De même, si p_C et p_D sont non nulles, deux tests suffisent pour comparer leurs signes:

si (p_C>0)
  si (p_C>(p_C-p_D)) p_D>0 ⇒ p_C et p_D ont le même signe;
  sinon p_C et p_D sont de signes contraires;
sinon
  si (p_C<(p_C-p_D)) p_D<0 ⇒ p_C et p_D ont le même signe;
  sinon p_C et p_D sont de signes contraires;

On calcule maintenant l'opposé de p_A pour un coût très modique:

$$-p_A = \boxed{x_C-x_A}\boxed{y_C-y_D} - \boxed{x_C-x_D}\boxed{y_C-y_A} \qquad (0;1;2)$$

Le coût total des opérations à ce stade est (0;9;6), c'est-à-dire exactement le même que celui nécessaire au calcul des trois déterminants.

Il a déjà été observé que la quatrième puissance pouvait se calculer à partir des trois premières. Cette relation permet de tester si p_B est nulle, sans évaluer cette quantité:

$$(p_B = p_A - p_D + p_C) \Rightarrow (p_B = 0 \Leftrightarrow \boxed{-p_A} = \boxed{p_C - p_D}) \qquad (1;0;0)$$

De même, deux tests suffisent pour comparer les signes de p_A et p_B, dans le cas où $p_A \neq 0$. Le problème équivaut à comparer les signes de $(-p_A)$ et $(-p_B)$ et donc ceux de $(-p_A)$ et $(-p_A - p_C + p_D)$. Il suffit alors de raisonner sur le signe de $(-p_A)$:

si $(-p_A>0)$
  si $(-p_A>(p_C-p_D))$: p_A et p_B ont le même signe;
  sinon: p_A et p_B sont de signes contraires;
sinon
  si $(-p_A<(p_C-p_D))$: p_A et p_B ont le même signe;
  sinon: p_A et p_B sont de signes contraires;

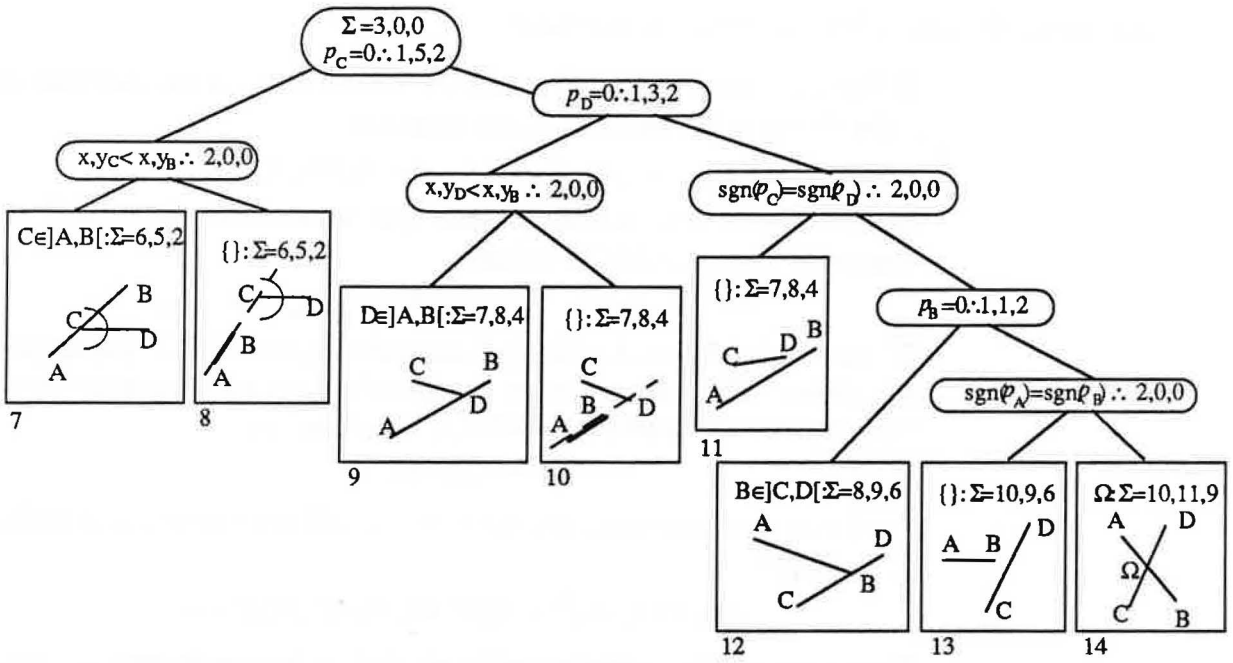
Présentation générale de l'algorithme

Dans l'algorithme présent, les deux nœuds centraux de l'arbre *f*-II.5 contiennent des tests relatifs à la puissance de B par rapport à (CD) et à celle de C par rapport à (AB), de coût (1;5;2) chacun.

Dans le premier cas, A=C et B≠D et la condition pour que B soit le point de rencontre des deux segments est $p_B = 0$: [A,B] et [C,D] sont alors colinéaires et B ne peut se situer hors du segment [C,D] sans que l'ordre lexicographique imposé soit violé. Dans le second cas, on a A≠C et B=D et un raisonnement 'symétrique' peut être mené.

La figure II.8 représente l'arbre de décision de l'algorithme utilisant la notion de puissance et les lignes suivantes sont destinées à apporter quelques commentaires pour son interprétation:

- si p_C=0, les deux segments ne peuvent se couper en C que si ce point est situé à l'intérieur du segment ouvert]A,B[, d'où les deux tests; les arcs de cercle représentés aux feuilles 7 et 8 symbolisent le fait que D peut occuper toute position dans laquelle il se situe à droite lexicographiquement de C. Il peut en particulier être sur le support de [A,B]; dans ce cas, soit il est extérieur à]A,B[et B doit aussi être inséré dans]C,D[, soit il est à l'intérieur et D doit être inséré dans [A,B]: les deux cas sont traités à une autre occasion.
- sinon, si p_D=0, les deux segments ne peuvent se couper en D que si ce point est situé à l'intérieur du segment ouvert]A,B[, d'où les deux tests.
  Sur la branche la plus à droite, on est maintenant sûr que C et D ne sont pas sur le support de [A,B]; on teste si C et D sont de part et d'autre de [A,B] en observant si p_C et p_D sont de même signe; seule la réponse positive est susceptible de rapporter une intersection.
  Si p_B=0, comme il a été prouvé que A et B sont de part et d'autre de (CD), B ne peut être qu'à l'intérieur de [A,B]!
  Sinon, on teste si A et B sont de part et d'autre de [C,D]. Dans le cas positif (feuille n°13), il ne peut y avoir d'intersection; dans le cas négatif, on obtient la solution normale dans la feuille n° 14. Les calculs relatifs à cette feuille impliquent une division pour $\mu = \frac{p_C}{p_C - p_D}$, puis trois additions et trois multiplications pour le calcul des coordonnées de Ω.



COUTS MOYENS		COUT NORMAL
78/14 ≈ 5,571	TESTS	10
83/14 ≈ 5,928	ADDITIONS	11
45/14 ≈ 3,214	MULTIPLICATIONS	9

Figure II.8. Arbre de décision et table de coûts relatifs à l'algorithme utilisant la notion de puissance.

Analyse de l'algorithme

Les efforts se montrent payants: les coûts moyens obtenus sont tous nettement inférieurs à ceux de l'algorithme de la pente, le coût normal est aussi bon – pour les additions et les multiplications – que son homologue précédent et le nombre de tests qu'il révèle est même inférieur!

Un des points forts du dernier algorithme est qu'il exploite au mieux le potentiel d'information contenu dans les extrémités: ainsi, les cas où C est inclus dans]A,B[sont bien réunis dans la seule feuille numéro 7.

Comptons le nombre de feuilles relatives à une solution remarquable, celles relatives à une solution vide dans chaque arbre et comparons les coûts moyens respectifs:

	paramètres	*puissances*
sol. remarq.	4 feuilles (10; 8,75; 5,5)	3 feuilles (7; 7,$\overline{3}$; 4)
pas de sol.	3 feuilles (7; 8,$\overline{3}$; 4,$\overline{6}$)	4 feuilles (7,5; 7,5; 4)

Les résultats sont clairs: l'arbre relatif aux puissances révèle un échange entre feuilles, qui se traduit explicitement par une diminution de tous les coûts sauf un! Ceci incite à se demander si les décisions du premier arbre ne sont pas bonnes ... mais prises aux mauvais endroits!

La méthode paramétrique revue et corrigée

Si l'on analyse les quantités manipulées dans la deuxième méthode *du point de vue* de la première, on constate que

$$\Delta\mu_2 = (x_B-x_A)(y_C-y_A) - (x_C-x_A)(y_B-y_A) = p_C$$

$\Delta\mu_2$ ne mesure donc pas autre chose que la puissance du point C par rapport à la droite (AB)! De même,

$$\Delta = (x_B-x_A)(y_C-y_D) - (x_C-x_D)(y_B-y_A) = p_C - p_D$$

D'une certaine façon, Δ est la 'puissance du segment' [C,D] par rapport à la droite (AB), autre manière d'interpréter le parallélisme des deux supports. Symétriquement, comme $p_C - p_D = p_B - p_A$,

$$\Delta = p_B - p_A$$

Δ est aussi la 'puissance du segment' [A,B] par rapport à la droite (CD). Enfin

$$\Delta\mu_1 = (x_C-x_A)(y_C-y_D) - (x_C-x_D)(y_C-y_A) = -p_A$$

Toutes ces relations corroborent les résultats déjà trouvés pour μ_1 et μ_2:

$$\mu_1=\frac{\Delta\mu_1}{\Delta}=\frac{-p_A}{p_B - p_A} \quad \mu_2=\frac{\Delta\mu_2}{\Delta}=\frac{p_C}{p_C - p_D}$$

Il n'est donc plus surprenant de trouver les mêmes coûts en additions et multiplications dans les calculs de solution normale! Plus intriguante est la signification que revêt chaque test du deuxième algorithme, du point de vue paramétrique.

Il est clair, compte tenu des relations précédentes, que les deux premiers tests de l'arbre *f*-II.8 peuvent s'écrire $\Delta\mu_2=0$ ($p_C=0$) et $\Delta\mu_2=\Delta$ ($p_D=0$): plutôt que de tester si les deux segments sont parallèles, on regarde si les points C ou D appartiennent à (AB); dans ces deux cas, on prouve l'intersection en vérifiant simplement que les coordonnées du point considéré sont bien celles d'un point de]A,B[.

Les deux tests sur les puissances ayant échoué, si l'on prouve que p_C et p_D, qui ne peuvent plus être nulles, ont des signes contraires, donc si C et D sont de part et d'autre de (AB), il est clair que $\Delta = p_C-p_D$ n'est pas nul: les segments ne peuvent *implicitement* plus être parallèles et la division par Δ sera toujours possible. De plus, toutes les conditions vérifiées jusqu'ici imposent $0<\mu_2<1$ à partir de ce niveau.

Le prochain test consiste à vérifier si $p_B=0$: ceci se traduit par $\Delta\mu_1=\Delta$, ou $\mu_1=1$, i.e. B inclus dans]C,D[;

Il ne reste plus, enfin, à vérifier que A et B sont situés de part et d'autre de (CD): c'est cette condition qui contraint de manière implicite à ne considérer que les valeurs entre 0 et 1 du paramètre μ_1.

Ce qui est satisfaisant dans cette démarche est qu'elle prouve la faible corrélation entre la pente des segments et leur capacité à s'intersecter: à aucun moment la méthode paramétrique modifiée 'sémantiquement' par la méthode des puissances n'a-t-elle besoin de vérifier si les supports des deux segments sont parallèles, donc si 'le déterminant est nul'. Un comble, pour une méthode fondée sur la résolution d'un système!

Une question s'impose maintenant: que sont devenus, dans l'arbre *f*-II.8, les cas de segments parallèles? Les segments strictement parallèles sont traités *dans l'ensemble* des cas de la feuille 11, les segments disjoints et de même support le sont dans celui de la feuille 8; enfin les segments 'à cheval' sont traités dans l'ensemble des cas de la feuille 7.

Justification a posteriori

Que dire des gains ainsi opérés et de la pertinence de l'algorithme suggéré? Et avant tout, en quoi est-il si important de pouvoir détecter les configurations remarquables?

Si l'on travaille avec une précision arbitraire, on évite au maximum, pour des raisons évidentes, la duplication des objets que l'on manipule. Si Ω est le point d'intersection de deux segments, il est primordial de détecter que ce point *pourrait* être une extrémité de segment et non de le *vérifier a posteriori*!

La technique consistant à calculer les trois déterminants, Ω quand cela est possible, puis de vérifier que Ω est une des extrémités serait stupide. Ceci explique le soin que l'on peut souhaiter mettre dans la recherche des solutions remarquables.

Pour ce qui est de l'efficacité, il est clair que si le type de tâche demandée est l'intersection de *quelques* segments épars, il n'est pas vraiment utile de se poser beaucoup de questions d'optimisation.

En revanche, si la tâche consiste à intersecter *beaucoup* de segments dont les extrémités se trouvent uniformément distribuées dans une section de plan, les optimisations proposées vont commencer à être bénéfiques. Si l'on travaille sur un ensemble structuré et constitué de beaucoup d'arêtes utilisant des points fixes comme extrémités, comme c'est le cas dans la construction d'un terrain polygonal plaqué sur une grille altimétrique, les bénéfices vont devenir très sensibles.

Pour prendre un exemple simple et plutôt minimisant, uniquement destiné à permettre de donner un ordre de grandeur, considérons l'intersection d'un ensemble de n segments du plan traitée par un algorithme de balayage comme celui de [B/OT 79]. On sait qu'une telle méthode nécessite, lorsque le nombre d'intersections est assez faible, un nombre de tests d'intersection entre segments de l'ordre de $O(n\log n)$. Supposons enfin que toutes les configurations possibles soient équiprobables. Pour n=1000, si l'on note $\delta_?$=6/7, δ_+=4/7 et δ_*=4/7 les gains moyens entre les deux méthodes, le nombre moyen de tests, d'additions et de multiplications économisés par le second algorithme est

$$n\log_2 n(\delta_?;\delta_+;\delta_*) \approx (8\,542;5\,694;5\,694). \qquad \text{(II-9)}$$

Quelle économie en temps ces résultats représentent-ils? On peut légitimement considérer qu'une arithmétique à précision arbitraire requiert au moins 10 fois plus de temps de calcul qu'une arithmétique flottante[3].

[3] Il n'y a pas que les opérations élémentaires qui entrent en ligne de compte: il faut aussi allouer et désallouer de la mémoire, ce qui prend un temps non négligeable.

Bien qu'il soit très délicat de chiffrer de manière figée les performances d'une implantation particulière d'arithmétique rationnelle sur un système donné, on peut considérer une table de projection à deux dimensions comme celle de la figure II.9. Dans celle-ci le paramètre horizontal est le nombre de segments, le paramètre vertical est le temps d'exécution d'une instruction (addition ou multiplication) et chaque cellule contient le résultat de (II-9) pour un type d'opération, correspondant au gain de temps entre les deux algorithmes en micro-secondes, millisecondes, secondes, minutes ou heures, suivant le cas.

+ ou *	10	100	1000	10000
1 s	18,982 s	6' 20"	1h 30'	21h
1/10 s	1,898 s	37,965 s	9' 29"	2h 06'
1/100 s	0,190 s	3,796 s	56,947 s	12' 40"
1/1000 s	18,982 ms	0,380 s	5,695 s	1' 15"
1/10000 s	1,898 ms	0,038 s	0,569 s	7,593 s
1/100000 s	0,190 ms	3,796 ms	0,057 s	0,759 s
1/1000000 s	18,982 µs	0,380 ms	5,695 ms	0,076 s

Figure II.9. Table de projection des gains en temps de calcul pour 10, 100, 1000, etc. segments.

Pour revenir sur le cas présenté plus haut, sur un système où une addition en arithmétique rationnelle prend un centième de seconde, l'ordre de grandeur du gain de temps pour mille segments est la minute.

Une telle table permet de rapporter les gains à l'échelle du système sur lequel on travaille et de juger si les effets des optimisations proposées ont un quelconque sens.

L'heureux utilisateur d'un système comme celui de la dernière ligne de la table *f*-II.9 n'a que faire de ces optimisations, même pour dix mille segments: sur son système, l'exécution de $10^3 \log_2 10^3$ opérations ne prendrait pas plus de 0,13 secondes de toute façon!

II.B.3 Où l'on reparle de précision...

Les idées développées dans ce chapitre ne doivent pas être prises en dehors du contexte de travail dans lequel elles ont émergé. Il s'agissait de trouver les solutions les moins coûteuses à des problèmes où l'on ne pouvait tolérer d'imprécision.

Force est de constater que la méthode des puissances part d'un principe élémentaire: toute extrémité a une influence sur le problème, il suffit de savoir exploiter cette information à bon escient. On est en droit de se poser plusieurs questions: quelle fiabilité peut-on espérer d'une telle méthode? En quoi serait-elle plus fiable que la méthode paramétrique? En quoi serait-elle plus 'orientée précision' qu'une autre?

Du point de vue de la précision, il n'y a aucune raison *a priori* de penser qu'une méthode soit supérieure à l'autre. La méthode paramétrique pose *fondamentalement* les même problèmes de précision que la méthode des puissances: on a pu constater que les objets manipulés par les deux méthodes étaient parfaitement identiques.

La seule grande différence est l'angle sous lequel ces objets sont considérés. Tous les tests de l'arbre f-II.8 peuvent s'exprimer en termes de distance ou de distance signée: la distance de M à la droite D est-elle nulle? La puissance de M par rapport à la droite D a-t-elle le même signe que celle de N par rapport à D? On verra que les techniques d'optimisation suggérées sont recommandées pour une classe de calcul incluant distances et puissances. L'élaboration de ces techniques s'est faite pour beaucoup à partir de l'expérience des puissances.

Bien sûr, les mêmes techniques d'optimisation peuvent être appliquées à la méthode paramétrique. Globalement, c'est l'arbre f-II.8 (et sa base f-II.5) qui correspond le mieux à l'esprit dans lequel les problèmes de précision sont traités. Mais rien n'interdit qu'on utilise les mêmes heuristiques dans des contextes différents.

III. Imprécisions Arbitraires

Les problèmes de précision et donc d'imprécision sont parmi les plus tenaces qui soient; il est même pratiquement impossible de s'en débarrasser complètement.

Ce chapitre présente une analyse des phénomènes d'imprécision dans le cadre de la géométrie algorithmique en relation avec la représentation des nombres et les modes de calculs en machine. Le support logistique est fréquemment l'intersection de segments dans le plan car c'est celui qui est employé par beaucoup d'auteurs.

En dernière partie on trouvera un tour d'horizon des méthodes récemment publiées pour tenter de résoudre les problèmes de cohérence et de précision et une préparation à la nouvelle méthode proposée au chapitre suivant.

Au cours de la présentation d'exemples, un certain nombre de résultats seront obtenus qui, en dehors de leur relation au sujet du présent chapitre, permettront de justifier les techniques proposées au suivant. Ceci explique pourquoi certaines listes de résultats sont parfois plus fournies que ne serait nécessaire pour les simples besoin des explications et pourquoi certaines formules ou certains calculs sont répertoriés en marge des explications données, bien que cette numérotation ne soit pas explicitement utilisée pour l'instant. Toutes ces informations seront importantes dans le chapitre IV.

. REFERENCES BIBLIOGRAPHIQUES POUR LE CHAPITRE III .

B/OT 79 J.L. Bentley, T. Ottman, *Algorithms for Reporting and Counting Geomeric Intersections*, IEEE Trans. Comp. C-28, 9, pp. 643-647, Sept. 1979.

BRO 77 W.S. Brown, *A realistic Model of floating-point Computation*, in Mathematical Software III, Ed. John R. Rice, Academic Press, pp. 343-360, 1977.

EDE 87 H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, Springer Verlag, New York, 1987.

E/M 88 H. Edelsbrunner, E.P. Mücke, *Simulation of Simplicit: A Technique to Cope with Degenerate Cases in Geometric Algorithms*, Proceedings of the 4th ACM Symposium on Computational Geometry, pp. 118-133, 1988.

G/S 85 L. Guibas, J. Stolfi, *Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams*, Proceedings of the 15th ACM Conference on the Theory of Computing, pp. 221-234, 1983. Version révisée, mêmes auteurs, même titre, ACM Transactions on Graphics, Vol 4. No. 2, April 1985, pp. 74-123.

G/S/S 89 L. Guibas, D. Salesin, J. Stolfi, *Epsilon Geometry: Building Robust Algorithms from Imprecise Computations*, Proceedings of the 5th ACM Symposium on Computational Geometry, pp. 208-217, 1989.

G/Y 86 D.H. Greene, F. F. Yao, *Finite-resolution Computational Geometry*, 27th Annual Symposium of the Foundations of Computer Science, pp. 143-152, 1986.

H/H/K 88 C. Hoffmann, J. Hopcroft, M. Karasick, *Towards Implementing Robust Geometric Computations*, Proceedings of th 4th ACM Symposium on Computational Geometry, pp. 106-115, 1988.

KAR 88 M. Karasick, *On the Representation and Manipulation of Rigid Solids*, Ph.D. Disseration, McGill University, 1988.

K/L/N 89 M. Karasick, D. Lieber, L. Nackman, *Efficient Delaunay Triangulation using Rational Arithmetic*, IBM Research Report RC 14455, 1989.

KII D.E. Knuth, *Seminumerical Algorithms*, Addison Wesley, Reading, Mass., 1969.

L/S 80 D.T. Lee, B.J. Schacter, *Two Algorithms for Constructing a Delaunay Triangulation*, Int. J. Comput. Inf. Sci., 9(3), pp. 219-242, 1980.

MIC 87 D. Michelucci, *Les représentations par les frontières quelques constructions; difficultés rencontrées*, Thèse de 3ème cycle, Ecole des Mines de Saint-Etienne, 1987.

MIL 88 V.J. Milenkovic, *Verifiable Implementations of Geometric Algorithms Using Finite Precision Arithmetic*, Ph.D. Dissertation, CMU-CS-168, Carnegie-Mellon, 1988. Ou encore V.J. Milenkovic, *Verifiable Implementations of Geometric Algorithms Using Finite Precision Arithmetic*, Artificial Intelligence, 37 377-401, 1988.

MIL 89a V.J. Milenkovic, *Double Precision Geometry: A General Technique for Calculating Line and Segment Intersections Using Rounded Arithmetic*, 30th Annual Symposium on the Foundations of Computer Science, 1989.

MIL 89b V.J. Milenkovic, *Robust Geometric Computations for Vision and Robotics*, DARPA, 1989.

M/L 89 V.J. Milenkovic, Z. Li, *Constructing Strongly Convex Hulls Using Exact or Rounded Arithmetic*, ACM-SIAM Symposium on Discrete Algorithms, 1989.

O/T/U 87 T. Ottman, G. Thiemt, C. Ullrich, *Numerical Stability of Geometric Algorithms*, Proceedings of the Third ACM Symposium on Computational Geometry, pp. 119-125, 1987.

PER 88 B. Péroche, *Modèles de partitionnement par diagramme de Voronoi: aspects algorithmiques*, E.N.S.M.S.E. 88-7, 1988.

P/S 85 F.P. Preparata, M.I. Shamos, *Computational Geometry - an Introduction*, Springer Verlag, New York, 1985.

S/I 89a K. Sugihara, M. Iri, *Two Design Principles of Geometric Algorithms in Finite-precision Arithmetic*, Appl. Math. Letter., Vol 2, No. 2, pp. 203-206, 1989.

S/I 89b K. Sugihara, M. Iri, *Construction of the Voronoi Diagram for One Million Generators in Single-precision Arithmetic*, Proceedings of the First Canadian Conference on Computational Geometry, 1989.

S/I 89c K. Sugihara, M. Iri, *A Solid Modelling System Free from Topological Inconsistency*, Research Memorandum RM 89-03, University of Tokyo, 1989.

S/S 85 M. Segal, C. H. Séquin, *Consistent Calculations for solids modeling*, Proceedings of the First ACM Symposium on Computational Geometry, pp. 29-38, 1985.

*YAP 87 C-K. Yap, *Symbolic Treatment of Geometric Degeneracies*, in Proceedings of the 13th IFIP Conference on System Modelling and Optimization, Chuo University, Tokyo, 1987. (← [YAP 88])

YAP 88 C-K. Yap, *A Geometric Consistency Theorem for a Symbolic Perturbation Scheme*, Proceedings of the 4th ACM Symposium on Computational Geometry, pp. 134-142, 1988. (→ [YAP 87])

III.A. ... PRECISIONS

III.A.1 Genèse

Quelle que soit la méthode de géométrie algorithmique que l'on décide d'implanter, la première situation à laquelle on la confronte risque fort de faire apparaître un problème de précision... C'est un état de fait qui, bien qu'imputable aux machines elles-mêmes, conduit beaucoup de gens à penser, un peu hâtivement, que cette discipline est le lieu géométrique des idées les plus éloignées de la technique qu'ils vont finir par adopter pour leur problème[1]. Heureusement, grâce à une longue série de recherches dans cette direction, cette croyance commence à reculer. La méthode proposée dans cette thèse a été validée sur une application en grandeur réelle où la moindre imprécision risquait de mettre en danger la cohérence de l'ensemble des données.

Le chapitre III se décompose en trois parties, dont celle-ci qui présente les outils de réflexions élémentaires de la représentation des nombres en machine; la deuxième section présente quelques manifestations ponctuelles de l'imprécision en géométrie algorithmique. La troisième section tente de faire un tour d'horizon des méthodes proposées jusqu'ici pour résoudre les problèmes évoqués au début. Il est indispensable de faire un tour d'horizon de toutes ces méthodes pour essayer de dégager une impression générale et surtout pour entrebâiller la porte vers la méthode décrite dans le prochain chapitre.

Aucune méthode ne peut prétendre être sortie du néant, et celle qui est proposée ici encore moins. Des raisons matérielles font que la technique suggérée par l'auteur a été accouchée au terme d'un long travail d'ermite, loin des réflexions qui se menaient, notamment aux Etats-Unis, sur le même sujet, au même moment.

La lecture des articles cités dans la troisième section a évidemment inspiré une nouvelle lecture/écriture des résultats présentés ici, mais, coïncidence heureuse, son essence la place en regard – comme à mi-chemin – des autres, toutes se complétant harmonieusement.

Il est aussi réconfortant de voir que le problème attire tant de réflexions diverses et que chacune apporte une vision différente.

Optique

Les problèmes liés à l'imprécision des machines sont des plus banals qui soient, mais leur analyse nécessite souvent beaucoup de connaissances en analyse numérique. Il serait déplacé de vouloir récrire ici ce que d'autres ouvrages exposent bien mieux.

Le plus simple, alors, est de présenter le minimum d'explications dans un minimum d'espace afin que le lecteur puisse apprécier les différentes analyses décrites plus loin. L'optique la plus naturelle est la 'découverte' des propriétés voulues par le truchement d'exemples ou de contre-exemples successifs.

---

[1] Rappelons, à ce propos, le célèbre fortunixme: "To err is human, but to really foul things up, you need a computer".

Choix

Il fallait un support pour présenter les idées les plus importantes relatives aux problèmes de précision: quoi de plus naturel que de donner des exemples *in situ* de comportements indésirables dus à la précision limitée des machines? Comment faire, d'autre part, pour que les exemples donnés ne soient ni des cas d'école, ni trop simplistes, et surtout, qu'ils soient révélateurs de l'essence des problèmes rencontrés sans être de savantes constructions érigées dans le but de faire échouer tel ou tel calcul ou prouver tel ou tel fait?

Beaucoup d'entre nous avons l'intuition de la facilité avec laquelle les imprécisions entachent l'exactitude d'une cascade de calculs au sein d'un ensemble complexe, et de la difficulté à en simuler de suffisamment révélatrices. L'optique prise dans la première section est de se donner les moyens de cette analyse: tous les exemples se veulent reproductibles et facilement analysables. Tous sont exprimés dans le contexte d'une précision double 'ordinaire' (type double de C sur 8 octets), sauf les deux premiers qui le sont avec une double précision 'étendue' (type double étendu de C sur 10 octets).

Le choix de la double précision dans les exemples présentés au début et dans l'analyse de la solution proposée a été imposé par la dynamique des données relatives aux problèmes traités; quant aux deux exemples isolés (traités en double précision étendue), il est vraisemblable qu'avec un peu plus de perspicacité la faille recherchée eût été trouvée grâce à des exemples plus fouillés en double précision ordinaire, mais que ces derniers eussent été bien plus 'tirés par les cheveux' que les deux calculs présentés en (III-1) et (III-2).

Aucun des exemples choisis ne nécessite le recours à des grands nombres, contrairement à la croyance qui veut que ceux-ci sont les seuls pour lesquels des problèmes de précision se présentent. En aucun cas leur choix ne doit laisser à penser qu'il s'agit-là d'exceptions rarissimes: il suffit de constater leur banalité pour s'apercevoir qu'aucune démarche ne peut prétendre en être exempte.

Plusieurs machines ont servi de support matériel aux idées présentées dans ce chapitre. Le lecteur peut être assuré que chaque exemple donné a été simulé. Il se peut qu'il essaie ces exemples sur 'sa machine' et qu'ils 'réussissent' (i.e. qu'ils échouent!) ou non: la seule chose fondamentale est que, dans l'absolu, les explications fournies en regard des exemples donnent une explication satisfaisante de l'échec éventuel de calculs de même type sur *une* machine, sous une forme plus ou moins cachée, un jour ou l'autre.

Maintenant que ceci est dit, les machines sur lesquelles le travail a été conçu et/ou implanté sont des stations de travail Sun 3115 et (Silicon Graphics) Iris 3130 ou 4D. Les dynamiques et les caractéristiques de taille fournies sont à prendre en référence aux stations Sun, car ce sont les machines qui ont servi de souche. Les exemples d'imprécision de ce chapitre ont tous été testés sur Macintosh et les résultats sont de simples copies de sorties d'algorithmes en C.

Notations

Pour éviter toute confusion, convenons à partir d'ici que le préfixe *pseudo-* désignera toujours une entité numérique **machine** (entier naturel ou relatif, décimal, rationnel, réel) qui sera toujours notée en ombré ($\mathbb{A}$, $\mathbb{B}$, ...) à laquelle correspond un élément d'un ensemble numérique **réel** –$\boldsymbol{N}$, $\boldsymbol{Z}$, $\boldsymbol{D}$, $\boldsymbol{Q}$, $\boldsymbol{R}$ – noté normalement (A,B, ...). Pour pousser le raisonnement jusqu'au bout, un pseudo-réel *est* un pseudo-décimal et ces deux termes seront indifféremment utilisés pour ce que l'on nomme habituellement les 'réels-machine'. Enfin, tout opérateur arithmétique sur la machine sera aussi représenté en ombré:

$$(+ \; - \; * \; / \; < \; \leq \; = \; > \; \geq \; \neq \; |\cdot|).$$

III.A.2 Représentation des pseudo-réels et précision

Les quelques éléments rassemblés dans cette section pour permettre une meilleur compréhension des problèmes de précision, peuvent se retrouver, sous une forme infiniment plus détaillée dans [KII], mieux connu des intimes sous le nom de '*The* Volume 2'. Une autre précieuse source de renseignements est [BRO 77].

On considérera par la suite deux types de pseudo-réels: les flottants et les doubles. Ces termes sont à interpréter comme des raccourcis pour 'pseudo-réels en simple et double précision'. Les doubles sont les pseudo-réels de choix pour les calculs en haute précision.

Expansion binaire

Comment la donnée 14.625 est-elle représentée en machine? On commence par séparer sa partie entière de sa partie décimale. La partie entière est décomposée en puissances croissantes et positives de 2 (ici $14_{10} = 1{\cdot}2^3 + 1{\cdot}2^2 + 1{\cdot}2^1 = 1110_2$). Appelons *partie fractionnaire* le nombre obtenu en soustrayant à un décimal sa partie entière. Cette partie fractionnaire est décomposée en puissances décroissantes strictement négatives de 2 selon le schéma suivant:

$$\begin{aligned} 0.625*2 &= \boxed{1}.250 \;\rightarrow\; 1{\cdot}2^{-1} \\ 0.250*2 &= \boxed{0}.5 \;\rightarrow\; 0{\cdot}2^{-2} \\ 0.5*2 &= \boxed{1}.0 \;\rightarrow\; 1{\cdot}2^{-3} \end{aligned}$$

Le processus s'arrête ici, car la dernière partie fractionnaire calculée est nulle. En définitive, $14.625_{10} = 1110.101_2$. Comme l'algorithme révèle un décimal pur, la précision relative à ce nombre est maximale.

On constate aisément que c'est toujours le cas pour les nombres décimaux possédant une expression rationnelle irréductible dont le dénominateur est égal à une puissance de 2, sous réserve que le codage des pseudo-réels permette de pousser la précision jusque là. En effet, si le processus s'arrête, disons à la puissance 2^{-j}, il existe une suite de chiffres binaires $a_k \in \{0,1\}$ et un entier A telle que la partie décimale obtenue peut s'écrire

$$\sum_{k=0}^{j} a_k 2^{-k} = \frac{A}{2^k}$$

A contrario, dès que le dénominateur de la représentation rationnelle irréductible de la partie fractionnaire considérée contient un facteur premier différent de 2, le processus d'expansion binaire décrit plus haut ne peut s'arrêter. Si l'on prend l'exemple de 7.8, l'expansion donne

$$\begin{aligned} 0.8*2 &= \boxed{1}.6 \rightarrow 1\cdot 2^{-1} \\ 0.6*2 &= \boxed{1}.2 \rightarrow 1\cdot 2^{-2} \\ 0.2*2 &= \boxed{0}.4 \rightarrow 0\cdot 2^{-3} \\ 0.4*2 &= \boxed{0}.8 \rightarrow 0\cdot 2^{-4} \end{aligned}$$

On constate l'apparition d'une période (1100), analogue à celle obtenue dans l'algorithme d'expansion décimale d'un rationnel irréductible non décimal. Le processus est donc infini périodique dans ce cas.

Il faut trouver un moyen d'arrêter l'algorithme de manière sûre et uniforme. Une première réflexion que l'on peut faire est que, si le nombre à coder provient d'autres calculs, la partie fractionnaire résiduelle va peut-être devenir nulle *au sens de la machine* au bout d'un certain rang. Il s'agit là de phénomènes liés à la précision interne des calculs, et dont la genèse deviendra claire dans quelques paragraphes.

Supposons, au contraire, que le test de nullité soit infaillible et que le nombre à coder soit tel que l'algorithme ne puisse s'arrêter par la seule vertu de cet test. Chaque type de données a une taille fixe et les pseudo-réels doubles n'échappent pas à la règle: un bit est réservé pour le signe, e bits sont réservés à l'exposant et m bits à la mantisse, $m+e+1$ étant égal à 8 fois la taille du type en octets. Le nombre de bits réservés à la mantisse donne une condition de terminaison naturelle et universelle du processus d'expansion binaire, lorsque le test relatif à la nullité de la partie fractionnaire résiduelle ne suffit pas.

Déphasages

Les choses se compliquent lorsque l'on constate que les algorithmes relatifs aux opérations arithmétiques ne sont pas nécessairement 'en phase' avec le précédent. Ceci peut être mis en évidence en comparant, sur une même machine, les expansions binaires de U=7.8, V=78.0/10.0 (U ≡ V, bien sûr), et U-V=78.0/10.0-7.8 ≡ 0.0:

```
          U    111.11001100110011001100110011001100110011001100110011001100110011001
          V    111.11001100110011001100110011001100110011001100110011001100110011101
(III-1)  U-V     0.00000000000000000000000000000000000000000000000000000000000000001
```

Le test d'égalité entre les quantités des deux premières lignes prend la valeur FAUX sur la machine, alors qu'une demande d'affichage de la différence entre elles donnera *apparemment* 0.0: la tolérance d'erreur n'est pas la même pour une opération d'affichage et un test logique.

De même, si l'on prend U=(5.5+4.7), V=10.2, on devrait trouver U ≡ V, U/V ≡ 1 et U-V ≡ 0:

```
          U    10.00110011001100110011001100110011001100110011001100110011001101
          V    10.00110011001100110011001100110011001100110011001100110011001100011
          U-V   0.00000000000000000000000000000000000000000000000000000000000000001
(III-2)   U/V   1.00000000000000000000000000000000000000000000000000000000000000001
```

Clairement, les tests `(U=V)`, `(U/V=1.0)` et `(U-V=0.0)` sont faux en machine, contre toute attente! De tels exemples ne sont pas des exceptions: le fait que le codage des pseudo-réels soit fini implique que les termes les moins significatifs ne soient pratiquement jamais cadrés, d'une évaluation à une autre. Notons aussi qu'en double précision ordinaire les calculs (III-1) et (III-2) n'ont provoqué aucune anomalie sur la machine de référence, ce qui ne signifie pas qu'aucune ne pourrait se produire...

Précision et nombre de bits

Peut-on faire un lien entre la précision et le nombre de chiffres exacts d'un résultat? Ainsi formulée, la question n'a pas grand sens. Le nombre de chiffres exacts d'un résultat peut être totalement déconnecté du problème de précision: par exemple, 0.999999 est une valeur approchée de 1.000000 bien plus 'serrée' ($1 \cdot 10^{-6}$) – bien que n'ayant 'aucun chiffre commun' avec 1.0 – que 1.000345 ($3.45 \cdot 10^{-4}$) – qui en a 4!

En fait, ce que l'on souhaite savoir c'est s'il existe une relation entre la précision binaire et la précision décimale. Plus formellement, soit k un entier naturel tel que 10^{-k} est la précision décimale souhaitée, on cherche les entiers naturels n tels que:

$$\begin{array}{lrcl} & 2^{-n} & < & 10^{-k} \\ \Leftrightarrow & -n \cdot \ln 2 & < & -k \cdot \ln 10 \\ \Leftrightarrow & n \cdot \ln 2 & > & k \cdot \ln 10 \\ \Leftrightarrow & n & > & k \cdot \log_2 10 \end{array} \qquad \text{(III-3)}$$

ou inversement $k < n \cdot \log_{10} 2$

De (III-3) on déduit que la précision décimale devient supérieure à 10^{-k} dès que $n \geq \lceil k \log_2 10 \rceil$. Par exemple, et en dehors d'autres considérations à venir, pour obtenir une précision supérieure à 10^{-8}, la taille de la mantisse des pseudo-réels doit être supérieure à $\lceil 8 \cdot \log_2 10 \rceil = 27$ bits.

Précision et taille des nombres

Il est maintenant nécessaire de se pencher sur la perte de précision dûe à la variation de taille des nombres représentés. Pour simplifier, admettons que les pseudo-réels traités soient positifs ou nuls et que les exposants de la notation flottante soient traités de manière identiques, qu'ils soient positifs ou nuls[2]. Cette dernière convention revient à dire que l'on dispose d'un bit magique permettant de savoir si l'exposant est positif ou négatif (dans la réalité, l'intervalle des exposants est simplement partagé en deux: une moitié pour les exposants négatifs et l'autre pour les positifs). On suppose aussi pour simplifier que les pseudo-réels sont normalisés, c'est-à-dire que l'on choisit, parmi toutes les représentations en virgule flottante celle dans laquelle le premier chiffre de la mantisse est forcément non nul; ainsi 0.000111_2 et 10.11 sont normalisés respectivement en $1.11 \cdot 2^{-4}$ et $1.011 \cdot 10^1$. On admet enfin que, quelle que soit la taille du type choisi, le bit de plus fort poids de la mantisse doit toujours être non nul.

[2] Le raisonnement pour les pseudo-réels signés diffère légèrement, mais ces détails n'ont aucun intérêt ici.

Si l'on considère maintenant que les pseudo-réels sont codés sur un octet, un nombre typique est de la forme | 1 | 1 0 | 1 1 1 1 1 |: le bit de poids fort représente le signe, les deux bits suivants représentent l'exposant exprimé en binaire et les 5 derniers bits sont pour la mantisse. Si l'exposant est positif, le nombre de cet exemple vaut donc $11111 \cdot 2^2 = 1111100_2$, soit 124.0_{10}. Si ce codage est exactement celui d'un entier, la précision binaire relative à ce nombre est maximale. Dans le cas contraire, autrement dit si l'expansion binaire a dû être tronquée par manque de place, la précision binaire ne peut être meilleure que 2^2. Si l'exposant est négatif, le nombre représenté est $11111 \cdot 2^{-2} = 111.11_2$, soit 7.75_{10}. Si le nombre originel était décimal, la précision est encore maximale, sinon la précision binaire est de l'ordre de 2^{-2} et la précision décimale de l'ordre de $2 \cdot \log_{10} 2 \approx 0.6$.

On constate donc que la précision est d'autant plus grande que la valeur absolue du nombre est petite, ou plutôt, que plus l'exposant est grand et plus la place réservée aux chiffres significatifs de la partie fractionnaire – et donc la précision binaire – est réduite.

En guise de synthèse provisoire, on peut dire que tout calcul n'incluant pas spécifiquement des entiers est *a priori* entaché d'erreur, et que le seul élément d'information stable que l'on puisse mettre en exergue d'un calcul réel de résultat x est qu'il existe un intervalle réel minimal $[x-2^{-k}, x+2^{-k}[$ contenant le pseudo-réel $\mathbf{x}$, représentant x en machine. L'amplitude de cet intervalle dépend de la taille du type choisi et permet de trouver l'ordre de grandeur de la précision décimale relative aux opérations d'une machine, pour un pseudo-réel référence, sachant que cette précision décroît globalement avec la valeur absolue d'un tel nombre.

III.B. IMPRECISION EN GEOMETRIE ALGORITHMIQUE

Avant de se lancer dans l'analyse des problèmes de précision et dans l'étude de leurs solutions, il est préférable d'en observer quelques manifestations dans le cadre de la géométrie algorithmique. Ces exemples permettront aussi, chemin faisant, d'accumuler des informations sur les caractéristiques de codage des pseudo-réels et la précision numérique qui leur est liée.

III.B.1 Où l'on reparle d'intersections...

On se rappelle que le calcul de l'intersection de deux segments fait intervenir, d'une manière directe ou non, le calcul du rapport de deux déterminants. Le cas où le point d'intersection coïncide avec une et une seule extrémité de segment peut être caractérisé par le fait que le déterminant principal du système est non nul et l'un des deux paramètres, lui-même rapport de deux déterminants, est nul ou égal à 1.0.

Le calcul (III-2) où une quantité réellement égale à 1.0 ne l'était pas en machine devrait suffire à prouver que toute séquence du genre

si (μ_1 = 1.0)
{...}

est susceptible de ne jamais être exécutée. Pour dissiper le doute que pourrait inspirer la présence de calculs en double précision étendue dans cet exemple, considérons le cas de figure trivial suivant, en double précision ordinaire: les extrémités des deux segments à tester sont

A(2.0, 4.0), B(4.0, 6.0), C(3.0, 5.0) et D(6.0, 4.0)

Un calcul rapide (dans $\boldsymbol{R}$, de grâce!) montre que C est à la fois le le point d'intersection des deux segments et le milieu du segment [A,B]. Tous les résultats obtenus en simulation sur la machine de référence en double précision confirment d'ailleurs pleinement ces affirmations:

Valeur affichée	μ_1: 0.500000	μ_2: 0.000000
Test	μ_1 = 0.5	μ_2 = 0.0
Expansion binaire	μ_1: 0.1_2 ($=0.5_{10}$)	μ_2: 0.0_2 ($=0.0_{10}$)

Malgré tout, notons que toutes les données sont des entiers déguisés! Que se passe-t-il si l'on divise systématiquement toutes les abscisses par 10? Les coordonnées des points deviennent:

A(0.2, 4.0), B(0.4, 6.0), C(0.3, 5.0) et D(0.6, 4.0)

Le point C matérialise donc encore l'intersection des deux segments et le milieu géométrique de [A,B]. Cette fois-ci, les résultats de la simulation menée sur la même machine sont incohérents:

Valeur affichée	μ_1: 0.500000	μ_2: 0.000000
Test	$\mu_1 \neq 0.5$	$\mu_2 \neq 0.0$

En première réflexion, on voit que le réflexe de faire afficher la valeur d'un paramètre n'est pas de grande utilité, amère remarque que plus d'un programmeur s'est déjà faite. Les tests étant négatifs, il faut se pencher sur les expansions binaires des deux paramètres pour interpréter le phénomène:

μ_1	0.01111111111111111111111111111111111111111111111111111111
μ_2	0.00000000000000000000000000000000000000000000000000000101

L'expansion de μ_2 montre que ce paramètre est très proche mais différent de 0.0. Qu'en est-il de μ_1? Un résultat bien connu d'algèbre veut que, si (u_n), $n \in \mathbf{N}$, est une suite géométrique de premier terme u_0 et de raison $q \neq 1$, on a

$$s_n = \sum_{i=0}^{i=n} u_i = u_0 \frac{1-q^{n+1}}{1-q} \text{ et, si } |q|<1, \lim_{n\to\infty} s_n = \frac{u_0}{1-q}$$

$$u_0=1 \text{ et } q={}^1/_2 \Rightarrow \lim_{n\to\infty} s_n - 1 = {}^1/_2+{}^1/_{2^2}+{}^1/_{2^3}+\ldots = \frac{1}{1-{}^1/_2} = 2$$

ce qui signifie que l'expansion binaire infinie $0.\underline{1}_2$ est égale à 1_{10}, et donc $0.0\underline{1}_2$, qui vaut la moitié, est égal à 0.5.

Ainsi, μ_1 et μ_2 sont extrêmement proches de leurs valeurs réelles, mais par suite d'imprécision paramétrique, C n'est ni le milieu de [A,B], ni le point de rencontre des deux segments. L'erreur commise sur μ_2 aurait pu être en excès de 1.0 et la détection de l'intersection aurait été mise en échec.

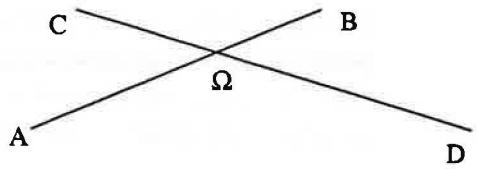


Figure III.1. Intersection simple.

III.B.2 Déplacements et dépendances

Admettons, malgré les problèmes de précision, qu'on souhaite quand même calculer les intersections d'un ensemble de segments. On se place maintenant dans le cas de données entières – donc de représentations parfaitement exactes – et d'une solution normale (voir chapitre II et se référer à la figure III.1). Les coordonnées de Ω sont *a priori* le résultat de divisions, de sorte que, dans le plan euclidien, elles sont rationnelles et ont fort peu de chances d'être décimales: dans tous les cas, les coordonnées de Ω en machine sont des pseudo-réels approchant au mieux ces rationnels.

Si cette opération est la seule que subissent les deux segments originaux, il est vraisemblable que les choses en resteront là: [A,B] et [C,D] sont transformés en quatre nouveaux segments [A,Ω], [Ω,B], [C,Ω] et [Ω,D] où Ω est le point du plan de coordonnées égales aux pseudo-réels calculés par l'algorithme d'intersection.

Dans certains cas, statistiquement assez rares, les coordonnées de Ω représentent en machine les mêmes valeurs que les coordonnées réelles de Ω et les deux points n'en font qu'un.

Pour comprendre ce qui se passe dans la majorité des cas où Ω≠Ω essayons d'imaginer une représentation géométrique du phénomène de précision raisonnablement proche de la réalité. Quel que soit le pseudo-réel x que l'on supposera fixé, il existe un ensemble fini de pseudo-réels z strictement positifs tels que:

$$x+z=x$$

L'existence de ces pseudo-réels et leur multiplicité sont dues à la précision limitée inhérente aux machines, et leur nombre est fini comme l'est de toute façon celui des pseudo-réels. De par l'existence de cet ensemble, il est possible de déduire qu'il existe un plus petit pseudo-réel, noté δ^+, tel que

$$x+\delta^+\neq x, \text{ soit } \delta^+ = \min\{z,\ z>0,\ x+z\neq x\}$$

Cette quantité, déjà dépendante de x, n'est pas la même en simple et en double précision, car elle est tributaire de la taille de la mantisse. Par exemple, sur le système utilisé dans le développement du facétiseur, si l'on prend $x=1.0$, on trouve en simple et double précision:

	flottants	*doubles*
$\min\{z,\ 1.0+z\neq 1.0\}$	$5.9046\cdot10^{-08}$	$1.11022\cdot10^{-16}$

Un raisonnement analogue permet de définir δ^- pour le même x par

$$\delta^- = \min\{z,\ z>0,\ x-z\neq x\}$$

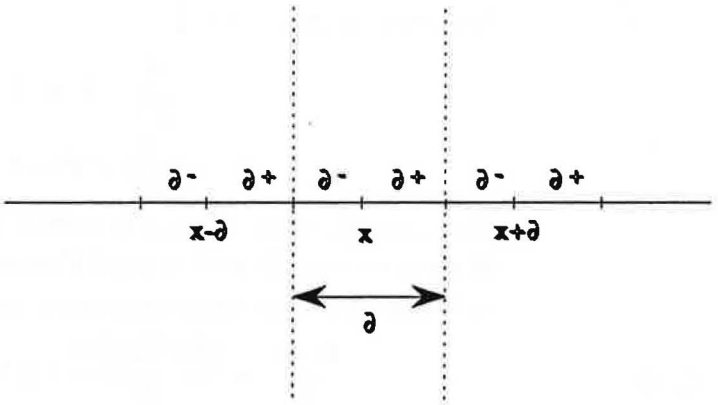


Figure III.2. Pseudo-réel et ses deux plus proches voisins.

Sur le même système et dans les mêmes conditions, on trouve en simple et double précision:

	flottants	*doubles*
$\min\{z, 1.0-z\neq 1.0\}$	$1.1921\cdot10^{-07}$	$2.22045\cdot10^{-16}$.

Si l'on pose

$$\delta = \delta^+ + \delta^-$$

δ permet de connaître, pour tout pseudo-réel x, les deux pseudo-réels qui lui sont les plus proches, comme le montre la figure III.2. Si l'on connaît δ, la valeur décimale de δ, quel que soit le point P(x,y) du plan il existe un quadruplet unique (P_{00}, P_{01}, P_{10}, P_{11}) de point décimaux, c'est-à-dire de points à coordonnées décimales, tel que

$$\forall\, i,j \in [0,1],\ 0 \le d(P, P_{ij}) < \delta$$

où d représente la distance euclidienne dans le plan.

En fait, les deux δ calculés pour x et y, coordonnées du point P, ne sont *a priori* pas les mêmes: leurs valeurs dépendent en réalité de la taille des valeurs absolues de x et de y. On peut se donner une d'idée de leurs variations en observant les valeurs de δ^+ et δ^- obtenues pour une série 'échantillonnée' irrégulièrement de puissances de 2:

(III-4)

i	2^i	δ^-	δ^+
0	1	$1.11022\cdot10^{-16}$	$2.22045\cdot10^{-16}$
1	2	$2.22045\cdot10^{-16}$	$4.44089\cdot10^{-16}$
2	4	$4.44089\cdot10^{-16}$	$8.88178\cdot10^{-16}$
10	1024	$1.13687\cdot10^{-13}$	$2.27374\cdot10^{-13}$
20	1048576	$1.16415\cdot10^{-10}$	$2.32831\cdot10^{-10}$
30	$1.07374\cdot10^{9}$	$1.19209\cdot10^{-7}$	$2.38419\cdot10^{-7}$
40	$1.09951\cdot10^{12}$	0.0001220703	0.0002441406
50	$1.1259\cdot10^{15}$	0.125	0.25
51	$2.2518\cdot10^{15}$	0.25	0.5
52	$4.5036\cdot10^{15}$	0.5	1.0
53	$9.0072\cdot10^{15}$	1.0	2.0
54	$1.80144\cdot10^{16}$	2.0	4.0
60	$1.15292\cdot10^{18}$	128	256
80	$1.20893\cdot10^{24}$	$1.34218\cdot10^{8}$	$2.68435\cdot10^{8}$
100	$1.26765\cdot10^{30}$	$1.40737\cdot10^{14}$	$2.81475\cdot10^{14}$

On constate que, si $\mathbb{x} = 2^i$,

$$\frac{\mathbb{x}}{2^{53}} = \delta^- \Leftrightarrow \delta^- = 2^{i-53}$$

$$\frac{\mathbb{x}}{2^{52}} = \delta^+ \Leftrightarrow \delta^+ = 2^{i-52}$$

Plus formellement, si p est le nombre de bits utilisés dans la mantisse, si $\mathbb{x}$ est un pseudo-réel et i est l'unique entier tel que $2^i \leq \mathbb{x} < 2^{i+1}$, un minorant de l'écart maximum relatif autour de x est donné par,

$$\frac{\delta^+ - \delta^-}{2^i} = \frac{2^{i-p+1} + 2^{i+p}}{2^i} = 2^{-p+1} + 2^{-p} \geq 2 \cdot 2^{-p} = 2^{1-p} \quad \text{(III-5)}$$

L'écart relatif entre deux pseudo-réels est donc toujours supérieur à la quantité 2^{1-p}, appelé *ulp* en anglais (unit of last place). Cette quantité joue un rôle très important dans l'estimation des erreurs.

Comme tous les pseudo-réels de même exposant sont régulièrement espacés sur une échelle absolue, on voit que l'écart relatif entre ces nombres décroît lorsque leur valeur absolue croît. A l'inverse, la taille de l'intervalle d'*arrondi* autour d'un pseudo-réel est, par (III-5), directement proportionnelle à sa valeur absolue. On obtient ainsi une grille composée de quatre quadrants, dans lesquels chaque carreau, hormis le plus petit, a des dimensions égales au double de celles du carreau qui le précédait sur chaque axe. La figure III.3. représente une partie du premier quadrant de cette grille.

Pour revenir au cas qui nous intéresse, le point d'intersection Ω des deux segments mentionnés plus haut se trouve à l'intérieur d'un carreau de la grille dont les quatre sommets sont constitués des points pseudo-décimaux les plus proches. La figure III.4 matérialise cette propriété, en première approximation graphique. D'une manière ou d'une autre, la machine associe à Ω un des quatre sommets, noté $\mathbb{\Omega}$ sur la figure, et lui substitue celui-ci. On emploie souvent le terme d'*arrondi* pour désigner ce phénomène, par analogie avec l'arrondi effectué sur les deux coordonnées de Ω.

Dans le cas où $\Omega \neq \mathbb{\Omega}$ et où l'on accepte l'approximation implicite faite par la machine, la seule relation que les quatre nouveaux segments [A,$\mathbb{\Omega}$], [B,$\mathbb{\Omega}$], [C,$\mathbb{\Omega}$] et [D,$\mathbb{\Omega}$] peuvent prétendre avoir avec les deux segments originaux [A,B] et [C,D] est qu'ils possèdent tous une et une seule des quatre extrémités A, B, C et D initiales. Leurs pentes et celles des segments initiaux sont différentes, comme le sont les coefficients des équations paramétriques de leurs supports. Tout test d'appartenance concernant les segments originaux est nécessairement entaché d'erreur s'il est traité relativement aux nouveaux et vice versa.

Dans une application géométrique, l'utilisateur est souvent amené à construire des objets polygonaux faisant explicitement référence à d'autres objets. Prenons un exemple simple mais significatif, car extrêmement fréquent. Soit [A,B] un segment et C un point supposé être le milieu de [A,B] (Figure III.5).

Figure III.3. Grille d'espacement des pseudo-réels.

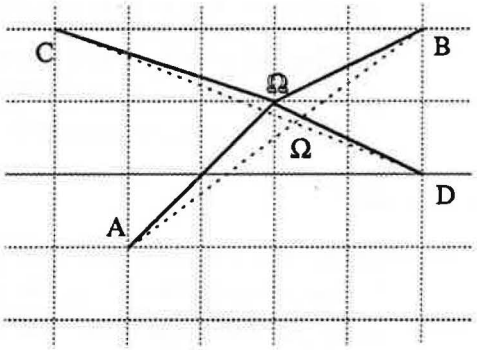


Figure III.4. Ω est "arrondi" en Ω.

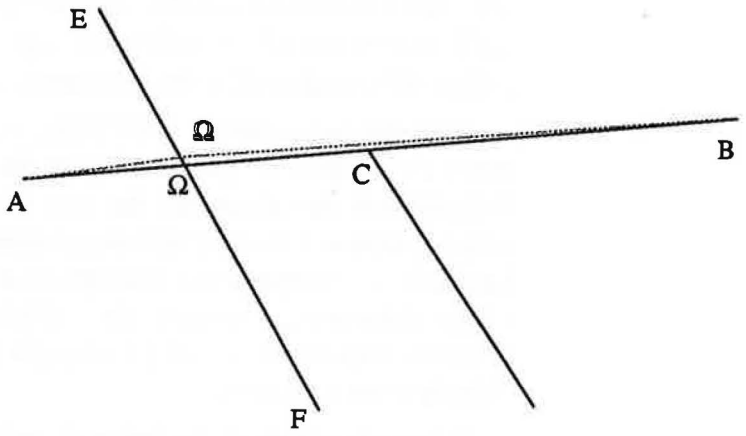


Figure III.5. Conflit de dépendance.

Si le segment [A,B] rencontre un autre segment [E,F], avant C, le point réel Ω est approché par le pseudo-point Ω, et le segment résultant [Ω,B] a très peu de chances de contenir C, pour les raisons évoquées au-dessus, en tout cas si la résolution des dépendances se fait en amont de la détection des intersections!

Enfin, la substitution de Ω à Ω peut provoquer des intersections qui n'existeraient pas sinon. On peut facilement imaginer, sur la figure précédente, que le segment [Ω,B] rencontre un dernier segment que ne coupe pas [Ω,B]!

III.B.3 La deuxième génération

Revenons au cas initial où deux segments [A,B] et [C,D] se coupent en un point Ω, approché par $\mathfrak{Q}$. Si maintenant l'un des quatre nouveaux segments, [$\mathfrak{Q}$,D], est coupé par un cinquième segment [A',B'] (Figure III.6), les choses se compliquent... Le nouveau point d'intersection, $\mathfrak{Q}$' n'a pas la même provenance que les extrémités initiales, ni que $\mathfrak{Q}$: les extrémités initiales sont de la génération 0 (on peut penser que pour de tels points, en général, la précision de la machine est optimale), $\mathfrak{Q}$ est de la génération 1 (car issu d'une paire {(0,0);(0,0)}), $\mathfrak{Q}$' est de la génération $1^1/_4$ (car issu d'une paire {(0,1);(0,0)}) et l'on trouve des éléments de génération $1^1/_2$, $1^3/_4$ et $1^4/_4$=2: ces derniers proviennent d'une paire {(1,1);(1,1)}. En règle générale, l'intersection normale de deux segments dont les extrémités sont toutes de la génération entière *m* définit un point de la génération entière *m+1*.

Chaque génération s'accompagne d'une perte de précision par rapport à la précédente et ce phénomène est une des manifestations de ce qu'il est convenu d'appeler la 'propagation d'erreur'.

III.B.4 Incohérences

[MIC 87] étudie différents problèmes d'incohérence liés à l'imprécision, et notamment l'influence des erreurs de calcul sur un algorithme de type 'balayage', comme celui présenté dans [B/OT 79]. Dans cet algorithme, on dispose d'un ensemble de segments du plan dont les extrémités sont données dans l'ordre lexicographique (cf. chapitre II) et qui sont eux-mêmes rangés suivant l'ordre lexicographique. La figure III.7 montre un tel ensemble de segments. Les lettres désignant les extrémités indiquent l'ordre qui règne sur celles-ci.

Pour chaque segment, l'extrémité la plus à gauche lexicographiquement est appelée *origine* et l'autre *fin*. Soit E_x l'ensemble constitué initialement des abscisses des extrémités de segments, rangées par ordre croissant. Les éléments de cet ensemble sont appelés *événements*. La droite de balayage est une structure évolutive, représentée graphiquement par une droite verticale se déplaçant parallèlement à elle-même d'événement en événement. La figure III.8 montre deux états rapprochés de cette structure.

A la verticale de A, la droite de balayage ne coupe que le segment [A,H], qui est alors le seul *segment actif*. La droite est translatée jusqu'en B: à ce stade, [BG] devient le deuxième segment actif, on lui reconnaît un voisin immédiatement inférieur, à savoir le segment actif [A,H], et comme les deux se coupent, leur point d'intersection $\mathfrak{Q}_{AH,BG}$ est calculé et son abscisse est insérée à la place adéquate dans E_x: si celle-ci n'était pas déjà dans cet ensemble, elle constitue un nouvel événement qui sera traité ultérieurement.

Les segments [A,H] et [B,G] sont décomposés en quatre sous-segments dont deux restent actifs ([A,$\Omega_{AH,BG}$], [B,$\mathfrak{Q}_{AH,BG}$]) et deux sont inactifs car ils ne coupent pas la droite de balayage.

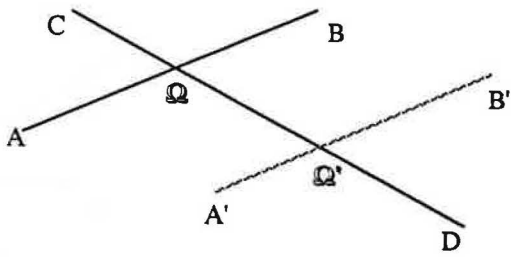


Figure III.6. Problèmes de précision à la deuxième génération.

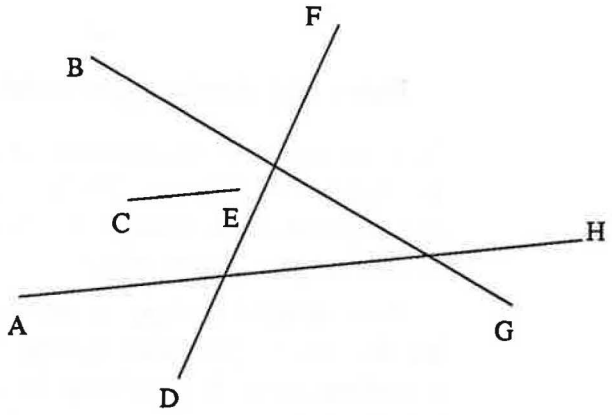


Figure III.7. Segments pour l'algorithme de Bentley-Ottman.

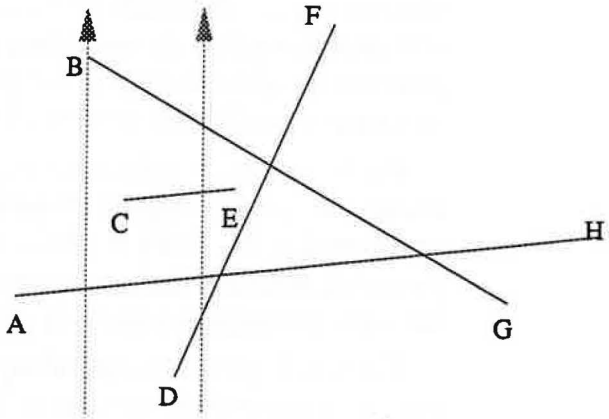


Figure III.8. Deux états rapprochés de la structure de balayage.

L'algorithme tire partie de la cohérence entre l'ordre global sur les segments et l'ordre local à la verticale d'une abscisse donnée en exploitant un invariant et deux types d'action complémentaires:

- INVARIANT: Sur la droite de balayage, deux segments ne peuvent se couper que s'ils sont adjacents selon l'ordre vertical.
- ACTION 1: Lorsque l'on arrive à la verticale d'une origine on insère une arête dans la structure de balayage et l'on teste les intersections éventuelles de l'arête insérée avec ses deux voisins immédiats au-dessus et au-dessous, quand ils existent.
- ACTION 2: Lorsque l'on parvient à la verticale d'une fin on supprime l'arête correspondante de la barre de balayage. L'arête supprimée ([C,E] pour la deuxième droite de balayage de *f*-III.8) rend éventuellement adjacentes les deux arêtes qui étaient jusqu'alors ses deux voisins immédiats dans l'ordre vertical, s'ils existent ([$\Omega_{DF,AH}$,F] et [B,G] sur la même figure): il suffit alors de de tester leur éventuelle intersection.

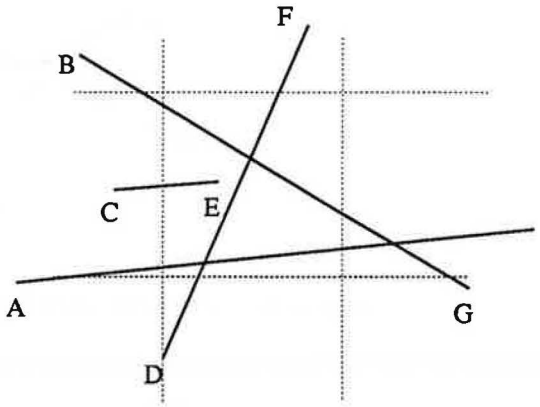


Figure III.9. Imprécision et incohérence topologique.

Si, à un moment quelconque, la précision devient insuffisante pour déterminer *correctement* l'ordre vertical sur la barre de balayage, la logique séquentielle de traitement des segments ou la logique d'intersection de ceux-ci peut être invalidée.

Pour illustrer ce type de comportement, analysons la figure III.9 où les droites en pointillés symbolisent la grille de précision évoquée précédemment. A l'intérieur du carreau central de côté δ, les événements sont traités, à cause de l'imprécision des calculs, de manière 'aveugle', du point de vue topologique: il est impossible de savoir sûrement si les segments [C,E] et [D,F] s'intersectent; d'ailleurs ceci n'a plus beaucoup de sens dans ce contexte. Il serait encore moins possible d'ordonner de manière cohérente les différents points d'intersection éventuellement trouvés à l'intérieur du carreau central.

En résumé toute cohérence topologique concernant les segments à l'intérieur de cette région disparaît. Ainsi, les positions relatives des segments [A,H], [D,F] et [B,G] risquent d'être incohérentes à l'intérieur du carreau central, et ceci a de grandes chances de provoquer une incohérence majeure lorsque la barre de balayage ressort de cette zone.

En fait, le problème est même plus grave: l'existence d'un seuil de précision laisse à penser que si des événements sont séparés par des distances sur l'axe des abscisses inférieures à ce seuil, ils seront nécessairement confondus et donc vraisemblablement traités dans un ordre incohérent avec la topologie globale.

III.C. TOUR D'HORIZON DES SOLUTIONS CONNUES

Certains auteurs ont déjà préconisé des solutions aux problèmes d'imprécision. On trouvera dans cette section une description des principales méthodes recensées à ce jour. Comme l'étude du sujet connaît une croissance quasi-exponentielle depuis quelques années, une synthèse exhaustive de tous les articles publiés serait fastidieuse. De même, une analyse comparative de ces méthodes est délicate à mener, souvent à cause de leur spécificité (telle méthode est conçue en fonction de tel phénomène), des différences de niveaux d'approche (solutions théoriques complexes contre méthodologies simples mais non générales) ou de la complexité des techniques impliquées.

A la place, il a été choisi de présenter les idées directrices de chaque méthode, afin de donner un aperçu des différents angles sous lesquels le même phénomène peut être étudié. Le lecteur intéressé trouvera en regard de chaque méthode les références bibliographiques de l'article original ou de la thèse dont elle provient.

Heureusement, une analyse critique comparative n'est pas nécessaire ici: la nouvelle méthode proposée dans le prochain chapitre procède d'une démarche suffisamment différente de toutes celles décrites dans cette section pour que les points de repère qui y auront été placés puissent servir à l'en différencier naturellement.

III.C.1 Epsilons

Dans toutes les méthodes de cette classe, on considère qu'il existe un seuil numérique ou *epsilon* en-dessous duquel aucune décision-machine n'est plus fiable et doit être remplacée par une décision empirique, justifiée par des motivations diverses. L'epsilon choisi dépend souvent des caractéristiques de la machine ou du problème rencontré.

Epsilon brut

De toutes les solutions connues, celle-ci est à la fois la plus simple et la moins efficace. Elle a une origine 'populaire' dans le sens où elle vient à l'esprit très rapidement, en fait dès que l'on rencontre la première situation aberrante due à une imprécision.

Pour revenir à une problématique de géométrie algorithmique, supposons que deux points A et B soient si proches que leur pseudo-distance (la distance euclidienne d(A,B) évaluée sur la machine) est inférieure à une valeur seuil. Alors A et B sont déclarés confondus et sont, à partir de cette décision, traités indifféremment.

Cette première mouture est un peu brutale: on peut souhaiter rendre la loi plus 'physique', en faisant intervenir ce que l'on a déjà vu sur la relation entre la précision et la taille de A lui-même (ou de B, mais les deux points sont supposés être très proches). On propose alors la version suivante: si |A-B| représente la distance entre A et B, |A| représente le module du point A (i.e. sa distance à l'origine); on accepte de dire que A et B sont confondus dès lors que

$$|A-B| \leq \varepsilon * |A|$$

Si cette approche est possible *dans certains cas*, elle devient hasardeuse dès que l'on fait référence à des notions de cohérence topologique: on est très vite amené à ranger dans la même 'classe d'imprécision' des nuages de points entiers alors que l'ordre dans lequel ils sont réellement situés les uns par rapport aux autres a une influence sur la cohérence globale.

L'introduction d'un epsilon a ceci de dangereux qu'il n'est jamais possible de prévoir quelles en sont les conséquences: celles-ci sont souvent plus graves, car structurelles et 'contagieuses', que les erreurs de calcul pour lesquelles l'epsilon a été introduit, qui ne sont, elles, que factuelles et par essence très ponctuelles.

Taille d'attributs minimum [S/S 85]

L'idée proposée par M. Segal et H. Séquin est de se donner une zone de 'no man's land' autour de tout objet (segment, face, ...), de sorte que les calculs relatifs à cette zone privilégient les attributs de l'objet de référence à défaut de tout autre.

A titre d'exemple, soient un segment $s = [A,B]$ du plan $\boldsymbol{P}$ et $D(z,\rho)$ le disque de rayon ρ centré en un point z de $\boldsymbol{P}$. Soit enfin

$$s_\varepsilon = \{z \in \boldsymbol{P} | D(z,\varepsilon) \cap s \neq \emptyset\}$$

l'ensemble, représenté à la figure III.10, constitué de tous les points du plan qui sont le centre d'un disque de rayon ε contenant au moins un point de s. (Cet ensemble porte parfois le nom d'*ε-dilaté* de s).

Si un point (ici l'extrémité C du segment [C,D]) se trouve à l'intérieur de cette zone, il est considéré comme appartenant à [A,B] et ses coordonnées sont 'recalées' de manière adéquate; sinon il sera toujours traité comme s'il était séparé de [A,B] par une distance supérieure à une valeur μ, quelle que soit la transformation appliquée aux deux objets. ε et μ sont des paramètres calculés en fonction des caractéristiques de l'application et de la machine.

Cette approche permet de créer ce que les auteurs appellent une *consolidation* de la scène étudiée. Son principal avantage est de permettre des opérations mathématiques constructives (comme celles propres à un modeleur) en s'assurant de garder une cohérence totale avec la topologie propre des objets initiaux. On reconnaît dans cette méthode une amélioration sur la méthode précédente: on gagne en cohérence, mais on ne peut toujours pas assurer que l'on ne créera pas de situations parasites, absentes de la topologie des données originales. On constate aussi que l'existence d'une région tampon autour de chaque attribut d'objet se traduit par des opérations de composition ensembliste de ces zones lorsque deux éléments s'intersectent et donc que la taille des intervalles élémentaires propres aux éléments originaux est susceptible de croître de génération en génération.

epsilon-convexité [M/L 89]

L'idée maîtresse de Milenkovic et Li est de s'attacher à ce que certaines propriétés 'vitales' d'un objet soumis à l'action d'un ensemble prédéfini d'opérations primitives, soient constamment préservées.

Prenons l'exemple de l'enveloppe convexe d'un nuage de n points du plan $\boldsymbol{P}$. La propriété principale de cet ensemble est qu'il s'agit du plus petit convexe du plan contenant ces n points. Les auteurs exposent un algorithme en $O(n\log n)$ permettant de construire un ensemble convexe formé de points initiaux tel que, si l'un quelconque de ces points est déplacé dans n'importe quelle direction d'une distance inférieure ou égale à ε, l'ensemble obtenu reste convexe.

Ce genre de technique représente certainement une évolution positive sur les méthodes d'epsilon naïves, mais elle nécessite une adaptation à chaque cas de propriété que l'on souhaite conserver. De plus, un grand effort doit être mis dans la construction d'une solution pour chaque cas et dans la preuve formelle de sa validité.

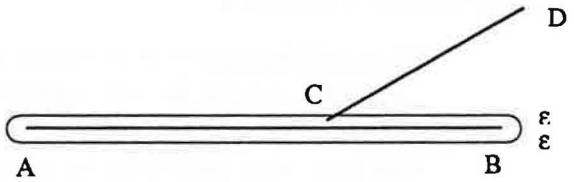


Figure III.10. Taille d'attribut minimum pour le segment [A,B].

Géométrie 'à-epsilon-près' [G/S/S 89]

Guibas, Salesin et Stolfi proposent une analyse fine des conditions mathématiques à imposer pour qu'une géométrie 'à epsilon près' soit cohérente, quelles que soient les opérations effectuées et les objets manipulés. La tâche est vaste et l'article cité en référence donne quelques directions fortement liées à l'analyse d'erreur.

On suppose que l'on est dans un espace $\boldsymbol{E}$ muni de la métrique $\|\cdot,\cdot\|$. Si P est un prédicat défini sur l'espace et $X \in \boldsymbol{E}$, P(X) prend une valeur parmi {vrai, faux}. On définit alors:

ε-P(X): P(X') est vrai pour un $X' \in \boldsymbol{E}$ tel que $\|X,X'\| \leq \varepsilon$.

Intuitivement cette définition implique que X est alors "au plus éloigné de ε de satisfaire P(X)". L'ensemble de vérité de ε-P(X) est celui de P(X) 'dilaté' (voir plus haut) par une boule de rayon ε, et 0-P(X) est identique à P(X).

En dimension 1, la procédure implantant P(X) retourne en fait une partition de la droite décrite par ε en trois intervalles F, I, V tels que le prédicat ε-P(X) est faux si $\varepsilon \in$ F, vrai si $\varepsilon \in$ V et *incertain* si $\varepsilon \in$ I. Puisque les pseudo-réels sont en nombre fini, cette partition peut être représentée par une paire de pseudo-réels l = (lmin, lmax) tels que ε-P(X) est faux si $\varepsilon <$ lmin, vrai si $\varepsilon \geq$ lmax et incertain si lmin$\leq\varepsilon <$lmax.

Les intervalles précédents sont alors calculés *via* la précision de l'arithmétique de la machine et chaque opération combine un ensemble minimum de prédicats élémentaires. Par exemple, pour savoir si deux points p et q sont superposés, on construit le prédicat ε-*Identiques*(p,q) qui sera vrai si et seulement si $\varepsilon \geq {}^1/_2\|p,q\|$. La procédure *Identiques* correspondante sera chargée de calculer la distance $\|p,q\|$ et de retourner une paire (lmin, lmax) telle que lmin $\leq {}^1/_2\|p,q\| \leq$ lmax.

L'intérêt de cette idée est double: l'article fournit une formalisation de la notion d'epsilon et le moyen de calculer les intervalles de fiabilité de primitives classiques en géométrie (co-linéarité de vecteurs, orientation d'un triangle, inclusion dans un objet, etc.) en fonction de calculs d'erreur bien connus. Bien que beaucoup de solutions antérieures aient exploité certaines de ces techniques, les auteurs sont les premiers à formaliser de manière si claire les phénomènes impliqués.

Notons en dernier que l'analyse proposée autorise une 'gestion' cohérente des imprécisions, mais ne permet pas plus que les précédentes d'éviter les déviances topologiques consécutives à l'imprécision.

III.C.2 Géométries robustes

Plusieurs auteurs ont travaillé sur ce sujet. L'idée maîtresse de ces méthodes est que la topologie propre des objets (qui est indépendante de la précision) est le seul motif de décision, lorsque la précision vient à manquer, qui permette de conserver une cohérence globale en fin d'opération, même si la cohérence locale a dû en souffrir.

Résolution finie [G/Y 86]

L'idée principale ici est de travailler sur une grille discrète de valeurs des coordonnées et de mettre en place une 'interface' robuste entre le domaine continu de définition des objets et cette grille discrète.

Greene et Yao proposent, dans le cas de l'intersection de segments (mais leur idée peut être généralisée), de déformer les segments par le biais de 'crochets'. Sur la figure III.11, un segment présente un tel crochet à proximité de son intersection avec un autre segment, passant par le point de la grille pour simplifier le dessin.

En quelque sorte, le segment déformé est 'accroché' à un point de la grille discrète sous l'action d'un autre segment, et l'action se prolonge sur un voisinage de ce point d'accrochage jusqu'à ce que l'on retombe sur le segment original d'extrémités fixes.

Si un segment est traversé par plusieurs autres, il reçoit autant de crochets que nécessaire, chaque crochet étant lui-même susceptible d'être 'accroché' par d'autres segments et/ou crochets. L'idée est que l'objet final, résultant du segment original et de ses crochets, est donné par le chemin le plus court reliant les deux extrémités initiales du segment et respectant les contraintes imposées par les crochets: il suffit de voir les crochets comme autant de plots de contournement positionnés à des points de la grille.

Si le segment a une longueur de l unités de grille, chaque crochet ajoute $O(\log l)$ sommets de grille au 'segment' et il sera nécessaire de recourir à un algorithme exact pour déterminer les points de la grille les plus proches après déformation par un crochet.

Les auteurs montrent que si k est le nombre de crochets sur un segment de longueur l, il est possible de construire le segment déformé en $O(k \log l)$ et que si l'on s'en tient à un nombre limité de règles de création des crochets, la cohérence globale des segments et leur topologie sont préservées dans l'ensemble de segments déformés obtenus. Il est seulement nécessaire de recourir à des algorithmes spécifiques de construction des crochets et de reconstitution des objets finaux.

Un des grands avantages de cette méthode est de montrer qu'un problème aussi complexe que celui de l'intersection cohérente d'un ensemble de segments peut être résolu en utilisant une précision limitée. Un de ses grands désavantages est que, bien que les points d'intersection soient tous *in fine* sur la grille, les calculs intermédiaires doivent être traités par le biais d'une arithmétique exacte. Dernier point important, si la création des crochets permet de résoudre les problèmes de cohérence topologique, elle n'en introduit pas moins, et encore plus que

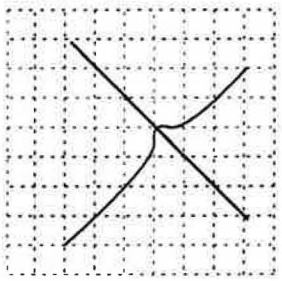


Figure III.11. Crochet sur un segment.

d'autres méthodes, des intersections parasites, absentes des données initiales, puisque les segments originaux ne sont pas subdivisés en sous-segments de même support, mais deviennent des collections continues de segments à supports de pentes variables.

Déterminisme et historique [MIL 88], [MIL 89a], [MIL 89b]

L'auteur a consacré une thèse et de nombreux articles à ce sujet. Parmi toutes celles qu'il présente, on peut dégager l'idée unificatrice de *cohérence chronologique*, qui préside à l'essentiel.

Si l'on se place dans une situation où l'on s'en remet à la précision finie d'une machine (calculs sur N bits), tout algorithme traditionnel va se mesurer à des décisions qui sont parfois erronées, parfois exactes et parfois incertaines. L'important est de définir le problème suffisamment clairement pour que toute décision prise, à un moment quelconque, puisse être en accord avec l'ensemble des décisions prises auparavant et conduire à une solution cohérente. Ce faisant, on n'écarte pas les décisions éventuellement erronées, mais on s'assure que l'ensemble des décisions prises ne pourra jamais violer un ensemble figé de propriétés logiques, topologiques ou géométriques.

Par exemple si, dans le processus de détection de l'intersection d'un ensemble de segments, on est amené à privilégier, à juste titre ou non, une relation d'ordre particulière entre deux segments, cette décision est irrévocable par la suite et tout sera fait pour l'entériner de manière que la topologie globale des segments soit cohérente, à défaut d'être la réplique exacte de la topologie réelle de ceux-ci.

Milenkovic remplace la notion de segment par celle d'objet pseudo-linéaire ou *courbe cachée* et présente une approche dans laquelle, un peu comme dans la méthode de résolution finie, les segments vont s'incurver au gré des intersections. Un point important est que l'on peut parvenir à faire en sorte que la 'courbe' obtenue en incurvant les segments autour de points de la grille entière sous-jacente, approche toujours les segments originaux avec la précision de N bits disponible.

Pour reprendre un exemple de l'auteur, si $P(x_P, y_P)$ a été reconnu en-dessous du segment [A,B] et $Q(x_Q,y_Q)$ est tel que $x_Q \geq x_P$, $y_Q \leq y_P$, le choix opéré pour P induit celui à faire pour Q: ce point doit nécessairement être en-dessous de [A,B].

La limitation de précision peut éventuellement avoir occasionné une erreur de choix pour P, mais P et Q auront des positions cohérentes relativement à [A,B]; l'auteur montre qu'il existe une courbe *théorique* relativement à laquelle P et Q occupent réellement les mêmes positions que celles 'choisies' vis à vis du segment [A,B]; de plus, cette courbe théorique cachée ne s'écarte jamais de plus d'une quantité α (caractéristique de la précision disponible) du segment [A,B].

Pour préserver la cohérence topologique *coûte que coûte*, l'auteur utilise une 'base de données' relative aux propriétés de chaque segment traité dont le but est de garder un historique explicite des choix qui ont été effectués à son sujet ou des positions relatives d'autres segments ou extrémités qui ont pu lui être comparés (e.g. C est à gauche de A et à droite de B, D est sur [A,B], ...). Toutes ces informations (qui sont nécessairement en nombre fini) sont 'vraies' modulo la précision disponible et sont établies par des relations de majoration ou de minoration liées à des calculs d'erreurs maximum à l'intérieur de cette précision. Tout nouveau choix concernant un segment déjà 'fiché' doit prendre en compte d'éventuelles informations contenues dans son historique.

La logique employée est très simple, une fois que les propriétés topologiques (ou logiques ou géométriques) qui doivent être respectées sont explicitées. Cette méthode infléchit les segments constituants les objets de façon à les faire passer par les points représentables en machine les plus proches, lorsqu'il est besoin. Comme toutes les précédentes, cette méthode implique des solutions parasites, mais cette fois-ci ces solutions sont explicitement considérées comme des consolidations locales d'une cohérence topologique globale.

Valence et cohérence [KAR 88], [H/H/K 88]

Karasick consacre lui aussi une thèse à la construction robuste de solides. En astreignant le degré des sommets de chaque solide à être égal à trois (i.e. chaque sommet est le point de rencontre de trois faces) et en contraignant la topologie des éléments constitutifs de ceux-ci, l'auteur montre que l'on peut construire un modèle géométrique dans lequel tous les objets sont cohérents sous l'influence d'un ensemble d'opérations bien définies.

Karasick utilise la représentation des solides par leurs frontières et propose des méthodes de résolution de conflits liés à l'imprécision par le biais d'un certain nombre de règles d'incidence d'inspiration empirique, permettant d'assurer une parfaite cohésion des solides construits par le modèle. La validation de ce dernier sur toutes les opérations géométriques (translations, rotations, etc) représentables en machine semble particulièrement positive: seules les rotations d'angle très petits (entre 10^{-5} et 10^{-10} degrés seulement!) sont signalées comme introduisant des solutions topologiquement incorrectes.

La technique préconisée ici est sous-tendue par l'expérience, l'auteur présentant des résultats au-delà de l'espérance de beaucoup de modeleurs réputés performants. La grande difficulté à laquelle se heurte sa technique reste, malgré tout, la justification théorique des hypothèses faites et le passage de l'expérience à la formalisation systématique.

III.C.3 Stabilité numérique et perturbations

Les méthodes de cette section relèvent toutes du domaine de l'analyse numérique. Elles permettent de trouver des réponses théoriques aux problèmes consécutifs à l'imprécision, mais leur complexité et le coût de leur implantation les rendent difficilement exploitables. Néanmoins, ce sont de telles méthodes, et bien plus souvent qu'on ne le croît, qui permettent de faire avancer le domaine de manière significative.

Simulation de simplicité [E/M 88], ([YAP 87], [YAP 88])

L'idée d'Edelsbrunner et Mücke est très simple: pour circonvenir les cas de dégénérescence dans les algorithmes de géométrie, on impose une perturbation aux données de sorte qu'elles ne puissent jamais amener à un cas dégénéré. Cette technique porte le nom de 'simulation de simplicité', le terme simplicité signifiant 'non-dégénérescence' en référence à la célèbre méthode du simplexe.

S'inspirant de méthodes d'analyse numérique, les auteurs proposent de définir comme *domaine de dégénérescence*, pour chaque problème traité, un ensemble d'éléments pour lesquels la solution générale du problème inclut un cas particulier. A titre d'exemple simple, la phrase 'soit un triangle non-aplati du plan' impose que les trois sommets définissant le triangle ne soient pas alignés (ou confondus, mais ceci est un cas particulier de la contrainte). Les dégénérescences, dans ce cas, sont représentées par toutes les situations où les trois sommets sont sur une même droite. Un ensemble d'objets est dit 'en position générale' s'il ne contient pas de dégénérescence.

Dans le modèle décrit par les auteurs, si π est une primitive topologique, elle prend en entrée un ensemble de points, retourne la valeur 0 si l'ensemble de données est dégénéré et retourne -1 ou 1 pour permettre de discriminer les cas de non-dégénérescence. Ainsi, $\pi^{-1}(0)$ représente l'ensemble des dégénérescences de la primitive π. Si cette primitive porte sur $d \geq 1$ objets, une *perturbation* sera une opération sur les d objets, considérés en position non-générale, qui leur interdit de contenir une quelconque dégénérescence.

Pour poursuivre l'analogie avec le triangle non-aplati, une idée serait de fixer les deux premiers sommets, de *faire en sorte* qu'ils soient bien à une distance minimum epsilon l'un de l'autre et définissent ainsi une droite Δ, et de contraindre le troisième à se situer dans un des deux demi-plans définis par Δ. Ces contraintes doivent être imposées par le calcul: si par exemple, la primitive π considérée a pour but de déterminer si un point est situé à l'intérieur d'un triangle, les coordonnées du triangle subissent une perturbation semblable à celle décrite précédemment, de façon à ce que le cas particulier d'un triangle aplati ne puisse jamais être rencontré par la machine: cette perturbation consiste elle-même à ce que toute distance calculée soit égale à la distance classique plus un ou plusieurs termes algébriques dont chacun traduit l'influence d'un point particulier.

Ces perturbations ne peuvent être faites qu'en fonction de chaque primitive considérée et de manière adaptative: l'ensemble ε-perturbé doit

tendre vers l'ensemble original lorsque ε tend vers 0, et posséder toutes les propriétés topologiques de ce dernier: on imagine mal une perturbation qui modifierait les relations métriques à l'intérieur d'un triangle. Enfin, les perturbations provoquées volontairement doivent être 'transparentes' à l'utilisateur, dans le sens où il est préférable qu'elles soient pré-définies sur un ensemble recensé de primitives bien connues, afin d'affranchir l'utilisateur des considérations mathématiques sous-tendant la solution proposée.

Un exemple convaincant cité par les auteurs est le cas de la détermination de non-alignement de trois points par le test du signe d'un déterminant: soient $P = \{p_1, p_2, \ldots, p_n\}$ n points d'un même plan. Il y a dégénérescence si trois points $p_i(x_i,y_i)$, $p_j(x_j,y_j)$ et $p_k(x_k,y_k)$ sont alignés et donc si un déterminant du type

$$D = \begin{vmatrix} 1 & 1 & 1 \\ x_i & x_j & x_k \\ y_i & y_j & y_k \end{vmatrix}$$

est nul. Si chaque point est repéré par un indice distinct, on peut effectuer la perturbation:

$$V_i(x_i, y_i) \rightarrow V_i^{\varepsilon}(x_i + \varepsilon^{2^{2i-1}}, y_i + \varepsilon^{2^{2i-1}})$$

D devient alors un polynôme en ε, D(ε), qui s'écrit, par puissances décroissantes d'exposants (avec i>j>k):

$$D(\varepsilon) = D - \varepsilon^{2^{2i-2}} \begin{vmatrix} 1 & 1 \\ x_j & x_k \end{vmatrix} + \varepsilon^{2^{2i-1}} \begin{vmatrix} 1 & 1 \\ y_j & y_k \end{vmatrix} + \varepsilon^{2^{2j-2}} \begin{vmatrix} 1 & 1 \\ x_i & x_k \end{vmatrix} + \ldots$$

Il apparaît clairement que si ε est nul, D(ε) devient égal à D. La simulation de simplicité consiste alors à évaluer l'un après l'autre les coefficients du polynôme D(ε) jusqu'à en trouver un non nul: le signe affecté à celui-ci donnera le signe de D puisque les termes consécutifs sont nécessairement d'ordre inférieur.

On peut apporter plusieurs critiques à la méthode: elle nécessite un volume de calculs imposant, une analyse très poussée de chaque contexte de calcul et est tributaire de l'existence sur la machine de référence d'une arithmétique exacte, voire d'un module de précision infinie pour l'évaluation de formules algébriques engendrées par la simulation.

Enfin, elle ne peut pas d'être d'un grand secours lorsque l'on doit accepter (et reconnaître) les dégénérescences aussi bien que les situations générales, ce qui est malheureusement le cas la plupart du temps. Néanmoins une telle technique permet de regarder le phénomène de l'imprécision sous un autre angle, fort instructif.

Le lecteur intéressé pourra trouver dans les deux références bibliographiques citées entre parenthèses une généralisation des techniques précédentes ainsi qu'une preuve de cohérence: Chee-Keng Yap propose le recours à une 'boîte magique' pour l'évaluation des polynômes décrits précédemment qui, en plus de donner une valeur d'évaluation, retourne un signe (+ ou -) et inhibe donc les situations de décision par valeur nulle.

Stabilité numérique [O/T/U 87]

T. Ottman, G. Thiemt et C. Ullrich ont proposé une technique permettant des évaluations exactes pour un certain nombre d'opérations.

Soit un opérateur arithmétique réel $\perp \in \{+,-,*,/\}$, $\mathbf{\perp}$ l'opérateur équivalent sur une machine de précision finie, et $\square$ l'opération d'arrondi sur celle-ci. Si l'on peut affirmer que, quels que soient les pseudo-réels $\mathbf{x}$ et $\mathbf{y}$ représentant en machine les réels x et y, l'hypothèse

$$\mathbf{x} \mathbin{\mathbf{\perp}} \mathbf{y} = \square\ (x \perp y)$$

peut être faite, et si de plus il existe un module (extérieur ou non) permettant de calculer *exactement* le produit scalaire de deux vecteurs de longueur fixée de tels pseudo-réels, les auteurs montrent que toute expression arithmétique simple (faisant intervenir les opérateurs arithmétiques +,-,*,/) peut être évaluée *exactement* par le biais du produit scalaire.

En fait, il suffit de démontrer que toute expression arithmétique 'représentable en machine' peut se décomposer en sous-expressions elles-mêmes calculables à l'aide du produit scalaire et portant sur des pseudo-réels (i.e. représentables en machine).

Encore une fois, un exemple donné par les auteurs permet de mieux comprendre cette idée. Soient $\mathbf{x}$ et $\mathbf{y}$ deux pseudo-réels représentant exactement en machine les réels x et y. On pose

$$\xi = \square\ (x * y) \text{ et } \psi = x * y - \xi$$

Alors, $x * y \equiv \xi + \psi$ et l'on peut montrer que ξ et ψ sont des pseudo-réels (représentables en machine) qui peuvent être calculés à l'aide du produit scalaire. Pour montrer ce dernier point, les auteurs font remarquer que l'on peut écrire d'une part

$$\boldsymbol{\xi} = \square\ (x * y) \equiv \mathbf{x} * \mathbf{y}$$

et d'autre part

$$\boldsymbol{\psi} = \square\ (\psi) = \square\ (x * y - \xi) = \square\ \left(\begin{pmatrix}\mathbf{x}\\ \boldsymbol{\xi}\end{pmatrix} \cdot \begin{pmatrix}\mathbf{y}\\ -1.0\end{pmatrix}\right) \equiv \begin{pmatrix}x\\ \xi\end{pmatrix} \cdot \begin{pmatrix}y\\ -1\end{pmatrix}$$

ou · dénote le produit scalaire.

L'intérêt de ce genre d'approche est de montrer que l'on peut obtenir une arithmétique exacte à partir d'un seul type d'évaluation en précision arbitraire.

Le revers de la médaille est la complexité inhérente à ce type d'évaluation: d'une part, le module d'évaluation du produit scalaire sera relativement complexe à intégrer (on pense au calcul formel) et d'autre part, les transformations des expressions arithmétiques nécessiteront un phénomène de 'transcodage' assez coûteux. Mais il existe des environnements dans lesquels cette solution est tout à fait envisageable.

Dernier point, cette technique ne peut pas facilement s'adapter aux cas où les calculs font intervenir des expressions algébriques plus complexes, mais ceci est aussi le lot pratiquement de toutes les méthodes existantes!

III.C.4 Solutions structurelles

Les deux principales méthodes de cette classe ont été conçues *explicitement* pour répondre à des besoins spécifiques, en l'occurrence aider à la résolution d'un problème recensé. Il se trouve que les deux articles choisis traitent de deux problèmes bien connus pour être 'duals' l'un de l'autre, mais il n'y a là aucune malice: il s'agit certainement de deux problèmes centraux de la théorie de la géométrie algorithmique. Les deux méthodes prouvent que l'on peut fort bien arriver à implanter des solutions réputées sensibles aux erreurs de calcul en adaptant les techniques d'évaluation à la structure de la solution.

Les deux problèmes utilisés comme support de travail seront présentés, ainsi que leurs solution, avec plus de détail dans le chapitre sur la facétisation. Le lecteur intéressé peut aussi consulter [P/S 85], [EDE 87] ou [PER 88].

Notons au passage que les deux idées présentées ici sont certainement celles qui ont donné, à ma connaissance et jusqu'ici, une des confirmations les plus 'tangibles' de l'intérêt des recherches présentées dans cette section.

Diagramme de Voronoi [S/I 89a], [S/I 89b], ([S/I 89c])

K. Sugihara et M. Iri ont étudié en détail une implantation robuste du diagramme de Voronoi de n points du plan. Leur technique les a conduit à présenter un algorithme permettant le tracé d'un tel diagramme pour un million de germes!

Le principe directeur de leurs travaux est de considérer que si, lors de la construction incrémentale d'un diagramme de Voronoi, la précision finie d'une machine peut parfois interdire la préservation de propriétés non topologiques (les polygones de Voronoi sont convexes, deux arêtes de Voronoi ne peuvent se couper qu'en leurs extrémités), elle permet toujours de préserver les propriétés topologiques

- P1 Un diagramme de Voronoi est un graphe planaire,
- P2 Tout sommet de Voronoi est *normalement* de degré trois,
- P3 Deux polygones de Voronoi ont au plus une arête en commun,
- P4 Il y a autant de polygones de Voronoi que de germes.

La principale tâche *constructive* à accomplir consiste donc à s'assurer, lorsqu'un i+1-ième germe est introduit dans le diagramme de Voronoi de i points déjà construit, que les propriétés P1 à P4 soient toutes respectées. La construction obtenue n'est peut-être pas unique, mais les calculs effectués peuvent servir à déterminer quelle solution est la plus proche du diagramme de Voronoi *réel*.

Du point de vue de la *destruction*, on sait que l'insertion d'un germe à l'intérieur d'un des polygones bornés (i.e. correspondant à un germe situé dans l'intérieur de l'enveloppe convexe) de Voronoi du diagramme relatif aux i premiers points provoque l'apparition d'un cycle 'autour du nouveau germe' (voir figure II.12, où le diagramme en pointillés déjà construit pour A, B, C et D est transformé par l'ajout du nouveau point P).

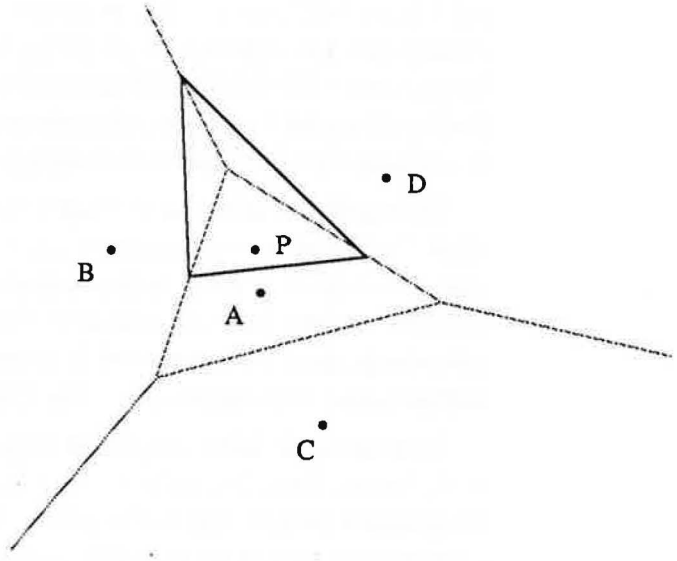


Figure III.12. Transformation incrémentale d'un diagramme de Voronoi.

Consécutivement à l'ajout du point P sur le diagramme *f*-III.12, les médiatrices de [A,B], [A,D] et [B,D] sont transformées par l'introduction des médiatrices de [A,P], [P,D] et [P,B]. Le cycle ainsi formé enferme en son intérieur une structure correspondant à la partie de l'ancien diagramme de Voronoi qui doit être détruite. On peut démontrer que cette structure est toujours un *arbre*.

Une des approches possibles consiste alors à 'vérifier' que l'algorithme construit effectivement un cycle et détruit dans le diagramme précédent une structure isomorphe à un tel arbre, dans les cas où la précision ne semble pas permettre de choisir entre plusieurs éventualités.

Une des conséquences intéressantes de cette méthodologie est qu'elle permet de simuler l'imprécision: les auteurs ont réussi à introduire une perturbation dans les données en *sortie* de leur algorithme, de sorte qu'ils ont pu simuler divers degrés d'imprécision et mesurer de manière pertinente la robustesse de leur algorithme.

Les travaux de ces deux auteurs débouchent sur une étude plus globale de la construction de modeleurs exempts de problèmes d'incohérences (référence entre parenthèses).

Triangulation de Delaunay [K/L/N 89]

Karasick, Lieber et Nackman proposent une méthode de construction d'une triangulation de Delaunay à partir d'une arithmétique rationnelle. L'idée est de rendre les évaluations les moins coûteuses possible en travaillant à la fois sur une optimisation de l'évaluation des expressions symboliques (expansion de déterminants 3*3 ou 4*4 par cofacteurs exploitant les particularités numériques des calculs impliqués), en utilisant une analyse adaptative du signe des déterminants considérés et en développant une arithmétique rationnelle modulaire (fondée sur une implantation en C++).

Les auteurs ont commencé par implanter l'algorithme de [G/S 85], lui-même reprenant les termes d'un algorithme en *diviser pour résoudre*

dû à [L/S 80][3], en double précision et en utilisant une arithmétique rationnelle. Les différences de temps de calcul se sont révélées tellement importantes (de 0.1 à 1200 secondes pour le même échantillon de 10 points aléatoires!) que ces chercheurs ont ressenti le besoin d'analyser les raisons de telles contre-performances.

De manière schématique, l'algorithme utilisé fait intervenir deux tests dont l'écriture symbolique revient à l'évaluation du signe de déterminants 3*3 ou 4*4. Leur idée est alors de réduire les coûts de calcul en rendant entiers les déterminants, dont les éléments sont *a priori* des rationnels, puis en examinant le comportement opératoire de plusieurs techniques d'expansions par cofacteurs.

Une seconde idée, originale et puissante dans ce contexte, est de remplacer, dans les calculs de déterminants, les éléments par des intervalles entiers (de taille arbitraire) et d'utiliser une arithmétique exacte pour entiers de longueur quelconque. Si le déterminant n'est pas nul, on parvient assez rapidement à encadrer sa valeur par un intervalle ne contenant pas zéro, ce qui permet de déterminer son signe!

Cette analyse par intervalles se montre très payante, et par le biais d'autres améliorations liées au codage propre de leur solution, les auteurs parviennent à abaisser les temps de calcul jusqu'à 3 secondes![4]

Notons, et ce point sera très important par la suite, que seuls les déterminants 'dégénérés' (de valeur nulle) ou très proches de zéro semblent poser des problèmes cruciaux dans la solution proposée ici: si la valeur du déterminant est très proche de zéro, il faut pousser l'encadrement très loin, et donc éventuellement manipuler des intervalles provenant de rationnels de dénominateurs et numérateurs de très grande taille! On verra dans le prochain chapitre que c'est justement ce point qu'il faut approfondir si l'on souhaite améliorer encore les performances.

L'article cité en référence est très explicite sur les méthodes à appliquer pour parvenir à de telles fins. On le lit avec d'autant plus d'intérêt qu'il contient, en quelque sorte, la réponse du berger à la bergère: c'est en analysant les raisons de la présence de valeurs intermédiaires extrêmement grandes que ces trois chercheurs ont pu découvrir une condition superflue dans les algorithmes théoriques originaux!

L'un des points forts de leur travail est de montrer qu'il existe des techniques particulièrement bien adaptées à l'implantation de solutions théoriques dont on connaît par ailleurs l'optimalité. De toutes celles présentées jusqu'ici, les idées de Karasick, Lieber et Nackman sont les plus en phase avec celles qui seront développées dans le prochain chapitre. Il apparaîtra alors clairement que cette communion de pensée permet d'envisager une fusion des deux méthodes pour d'encore meilleurs résultats.

---

[3] Le lecteur trouvera une synthèse de cet algorithme dans le chapitre VI, consacré à la facétisation.

[4] Remarquons à cette occasion qu'il est peu réaliste d'imaginer une arithmétique rationnelle qui 'descende' en-dessous de 30 fois les performances de l'arithmétique des doubles de la machine!

III.C.5 Solutions exactes

On peut effectivement se demander à ce stade pourquoi ne pas opter pour une solution dans laquelle les calculs sont effectués à l'aide d'une arithmétique 'exacte'. Le cas d'une solution rationnelle est traité à part dans le chapitre IV, car elle sert de toile de fond à la solution sous-tendant le présent travail. Malgré tout, on sait par expérience que le choix d'une arithmétique exacte impose des temps de calcul prohibitifs et que la complexité des algorithmes mis en œuvre dans ce contexte est trop souvent dominée par celle des primitives élémentaires.

Les trois auteurs précédents signalent que, lors de leur première tentative d'implantation en rationnels, certaines valeurs intermédiaires obtenues pendant l'exécution de l'algorithme de triangulation appliqué à dix points pris au hasard dans le disque *unité*, comportaient jusqu'à 81 chiffres en base 2^{16}!

On comprend aisément que de telles statistiques découragent les meilleures volontés. Néanmoins, il existe des moyens de réduire considérablement le coût de telles entreprises comme en témoigne le dernier exemple de solution de cette section.

Recalage sur une grille entière [MIC 87]

D. Michelucci se place dans le cas où les données sont supposées appartenir (ou avoir été 'recalées' sur) une grille entière de taille suffisamment grande. Tous les calculs impliqués dans la détection de l'intersection de deux segments concernent alors des pseudo-rationnels bornés: si la taille de la grille est bien choisie, les numérateurs et dénominateurs des rationnels issus des calculs des trois déterminants et des deux paramètres ne seront jamais supérieurs au plus grand entier représentable en machine.

En effet si G est la taille de la grille, les coefficients de l'équation paramétrique du support de tout segment de droite dont les extrémités sont des points de cette grille, sont des entiers A, B et C, liés par une relation du type:

$$Ax+By+C=0$$

Comme les points définissant les segments sont forcément à l'intérieur de la grille, on est assuré que

$$|A|, |B| \leq G \text{ et } |C| \leq 2G^2 \text{ (cf II.A.)}$$

De même, les quantités Δ, $\Delta\mu_1$ et $\Delta\mu_2$ sont nécessairement inférieures à $2G^2$ en valeur absolue (cf. (II-3)). Il suffit donc de choisir G tel que $2G^2$ soit inférieur ou égal au plus grand entier représentable en machine. Si les entiers sont codés sur n bits, ce nombre est $N = 2^n-1$, et l'on doit choisir G de sorte que

$$G \leq \sqrt{2^{n-1}-\frac{1}{2}}$$

(III-6)
$$n = 32 \Leftrightarrow G < 32768$$

Ce majorant n'est lui-même valable que pour les intersections entre segments originaux: il doit être révisé à la baisse dès qu'un segment de génération nulle est comparé à un segment d'une autre génération.

La solution proposée par D. Michelucci est en fait une technique qui permet de se ramener à des calculs sur des pseudo-entiers uniquement. Elle contient donc sa propre critique: si la dynamique des données est trop grande, elle ne sera plus applicable. Malgré tout, lorsqu'elle est applicable, la méthode du recalage est moins délicate à utiliser qu'une solution faisant intervenir une arithmétique de précision infinie, car la taille des quantités manipulées permet de profiter, dans les algorithmes de base, des opérations naturelles relatives aux pseudo-entiers.

Notons enfin que Michelucci prolonge son étude à la modélisation d'objets par leurs frontières, propose une nouvelle structure de données unifiante et minimale pour décrire un solide et fait une analyse détaillée des conditions dans lesquelles une telle représentation pourrait être effectivement (à défaut d'efficacement, pour l'instant) implantée, moyennant l'existence d'une arithmétique *algébrique*. Ce domaine est encore à l'état de la recherche pure, mais il est presque inévitable que les solutions préconisées dans la thèse citée ici seront largement envisageables *asymptotiquement* d'ici quelques générations: le tout est de savoir si ce terme doit être appliqué aux machines ou aux humains!

IV. Precision arbitraire

Presque toutes les solutions présentées dans le chapitre précédent partent du même principe: l'arithmétique exacte étant très pénalisante, essayons de nous contenter d'une arithmétique à précision finie en conservant le maximum de cohérence avec la solution exacte.

On a pu voir que ce présupposé conduit malheureusement à introduire dans les problèmes des solutions parasites ou à entériner des déviances locales de la topologie originale allant parfois jusqu'à fausser complètement celle de la solution globale, dans les situations les plus délicates.

On a pu constater aussi que les solutions proposées achoppent pratiquement toujours sur la levée d'indéterminations relatives à des tests autour de valeurs de dégénérescence: il se trouve que ce sont les évaluations les plus instables sur lesquelles reposent les poids de décision les plus forts.

N'est-il pas possible d'envisager une solution dans laquelle toutes les décisions sont prises sur la foi d'une première évaluation logique en précision finie et sur celle d'une précision infinie (arbitraire) si et seulement si l'issue semble incertaine selon un critère 'universel' de référence?

Est-il en d'autres termes possible de définir une mesure de fiabilité globale qui soit éventuellement remise en cause localement, mais qui assure que tous les tests intermédiaires sont à la fois fiables du point de vue de la précision absolue et du point de vue de la précision disponible?

Ce chapitre apportera des réponses à toutes ces questions en proposant l'analyse d'une solution à précision adaptative et de son coût.

. REFERENCES BIBLIOGRAPHIQUES POUR LE CHAPITRE IV .

A/H/U 74 A. Aho, J. Hopcroft, J. Ullman, *The Design and Analysis of Computer Algorithms*, Addison Wesley Publishing Company, 1974.

A/H/U 87 A. Aho, J. Hopcroft, J. Ullman, *Structures de données et Algorithmes*, InterEditions, Paris, traduction de J.M. Moreau, 1987. Version originale: *Data Structures and Algorithms*, Addison Wesley, Reading, Mass.,

G/S1983. L. Guibas, J. Stolfi, *Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams*, Proceedings of the 15th ACM Conference on the Theory of Computing, pp. 221-234, 1983.

G/S/S 89 L. Guibas, D. Salesin, J. Stolfi, *Epsilon Geometry: Building Robust Algorithms from Imprecise Computations*, Proceedings of the 5th ACM Symposium on Computational Geometry, pp. 208-217, 1989.

K/L/N 89 M. Karasick, D. Lieber, L. Nackman, *Efficient Delaunay Triangulation using Rational Arithmetic*, IBM Research Report RC 14455, 1989.

K II D.E. Knuth, *Seminumerical Algorithms*, Addison Wesley, Reading, Mass., 1969.

SED 90 R. Sedgewick, *Algorithms in C*, Addison Wesley Publishing Company Inc., 1990. Version française: *Algorithmes en C*, InterEditions, Paris, traduction de J.M. Moreau, en cours (1990).

Le but de ce chapitre est d'apporter une solution novatrice au problème de l'imprécision, dont la philosophie se situe à mi-chemin entre celles des solutions de type 'epsilon' et celles des solutions exactes. Une solution de cette dernière catégorie considère que les données sont aussi précises que possible et fournit des résultats pour lesquels la précision est absolue: ceci restreint donc ces méthodes à des problèmes dans lesquels les calculs ne font pas explicitement intervenir des expressions irrationnelles, ou, si c'est le cas, de manière telle que leur représentation puisse être manipulée sans ambiguïté (représentation des racines d'une équation du second degré par fractions continues, par exemple).

Un type de solution qui sera particulièrement utile par la suite est fondée sur l'existence d'un module d'arithmétique rationnelle. L'écriture d'une telle 'bibliothèque' nécessite quelque soin, tant en ce qui concerne la représentation des objets manipulés que la conception d'algorithmes efficaces pour ces manipulations: la première section de ce chapitre est consacrée à ce sujet. La section suivante analyse le spécificités de la solution adaptative présentée en détail dans la troisième et dernière section de ce chapitre.

IV.A. CARACTERISTIQUES D'UNE ARITHMETIQUE RATIONNELLE

Un certain nombre de résultats utilisés dans cette section sont présentés en détail dans [K II] et dans [A/H/U 74]. Les programmeurs émérites connaissent tous l'ouvrage de D.E. Knuth 'The Art of Computer Programming'. Le tome 2 ([K II]) contient tout ce qu'il faut savoir sur les algorithmes numériques et fournit notamment une base de travail très documentée sur la réalisation de modules de calcul en arithmétique entière et rationnelle. Les informations contenues dans ces chapitres sont tellement rigoureuses que l'auteur a lancé depuis longtemps le défi amical de renvoyer $10 à tout lecteur qui y relèverait une erreur.

Devant un ouvrage d'une telle envergure, on se sent plutôt tenté de passer son chemin, en profitant des leçons prodiguées par le 'Maître'. Et puis, on s'aperçoit que s'il ne contient pas d'erreurs à $10, l'ouvrage présente une version des faits qui, parce qu'elle est tournée vers l'architecture d'une machine idéale mais figée, est quelque peu déphasée par rapport aux langages évolués disponibles maintenant. Ceci n'est pas une tare rédhibitoire, dans le sens où le contenu des explications (le 'fond') est souvent totalement indépendant de ce genre de considérations ('la forme').

En revanche, il peut s'avérer relativement agréable à la personne intéressée par ces problèmes, de trouver une présentation différente, plus orientée vers l'algorithmique que vers la machine. La présente section voudrait être ce trait d'union, sans vouloir faire oublier au lecteur que les bases de ce qu'il cherche se trouvent toujours et avec autant d'actualité dans le travail gigantesque et incontournable de Knuth.

L'optique qui sera souvent prise ici consistera à 'reconduire' les analyses de D.E. Knuth en profitant de l'occasion pour proposer des méthodes parfois différentes des siennes, en tout cas plus directement adaptées à la représentation des nombres par listes chaînées.

IV.A.1 Représentation des rationnels

A partir de cette section, certains nombres seront assimilables à leur représentation en machine: ainsi un entier quelconque (relatif ou naturel) sera représentable en machine quelle que soit sa taille (dans la limite de la mémoire vive de la machine, bien entendu!...). Pour distinguer les deux entités, on conviendra que les notations entier, rationnel sont toutes relatives à des entités représentées en machine. Les variables ou nombres correspondants auront la même apparence typographique.

Type et Base

Pour travailler en rationnel, il est indispensable de choisir une base et un type pour représenter les chiffres de celles-ci. On choisira ici le type 'entier non signé' de la machine car il permet la meilleure rentabilisation de l'espace mémoire. La taille de ce type détermine la base dans laquelle s'effectuent les calculs.

Soit B cette base. Les chiffres à représenter couvrent donc l'intervalle [0, B-1] et tout naturel n peut s'exprimer, en base B, sous forme d'une somme

$$n = \lambda_0 + \lambda_1 B + \lambda_2 B^2 + \ldots + \lambda_m B^m = \sum_{i=0}^{m} \lambda_i^i \cdot B^i$$

où les λ_i sont des chiffres en base B. Par commodité, ce nombre, dont on dira qu'il est de *longueur* m+1, sera aussi noté:

$$(\lambda_m, \lambda_{m-1}, \ldots, \lambda_2, \lambda_1, \lambda_0)_B$$

ou plus simplement: $(\lambda_m, \lambda_{m-1}, \ldots, \lambda_2, \lambda_1, \lambda_0)$

comme la base sera supposée fixe dans tout ce chapitre. Cette notation recouvre exactement la représentation décimale habituelle.

Par la suite, il sera toujours considéré, sans que cela soit répété, que le chiffre le plus significatif de tout nombre (donc ici λ_m) est obligatoirement différent de 0.

La détermination de la base nécessite une petite réflexion: il faut à la fois la choisir suffisamment grande pour minimiser le nombre de chiffres dans la représentation des données mais pas grande au point d'imposer la vérification permanente que tous les calculs se font bien dans la dynamique impartie au type leur servant de support. On verra lors de l'étude des opérations arithmétiques que c'est la multiplication qui provoque le calcul de retenue le plus élevé, sous la forme $u*v+w+r_u$ où u, v, w et r_u sont des chiffres (cf. (IV-2)). Si ns est le nombre de bits d'un entier non signé, le plus grand entier de ce type représentable en machine est $2^{ns} - 1$. On doit alors respecter l'inégalité

$$\begin{aligned} & (B-1)^2 + 2(B-1) \le 2^{ns} - 1 \\ \Leftrightarrow \quad & B^2 - 1 \le 2^{ns} - 1 \\ \Rightarrow \quad & B = 2^{\lfloor ns/2 \rfloor} \end{aligned} \qquad \text{(IV-1)}$$

Si le nombre de bits d'un entier non signé est de $ns = 32$, la base peut être prise comme étant égale à $B = 2^{16} = 65536$.

Il est aussi important de connaître à l'avance la taille moyenne des nombres que l'on va être amené à manipuler. On peut se faire une idée de l'ordre de grandeur des entiers représentés par quelques chiffres dans la base B = 65536:

```
1 chiffre:    0              →    65 536
2 chiffres:   65 537         →    4 294 967 296
3 chiffres:   4 294 967 297  →    ≅ 2.8147·10^14
```

Pour donner un élément de comparaison, des données dont la taille varie du millimètre (unité principale) à la centaine de kilomètres ($1 \cdot 10^9$ millimètres) sont représentées sur 2 chiffres dans la base considérée. Les résultats intermédiaires peuvent bien sûr être remarquablement plus volumineux.

Représentation des rationnels

Un rationnel, qui est composé d'un numérateur entier et d'un dénominateur entier non nul, dont on suppose qu'ils sont premiers entre eux, est représenté par une structure de données composée d'un signe (sur un bit si cela a un sens sur la machine hôte) et de deux listes chaînées de chiffres en base B choisie:

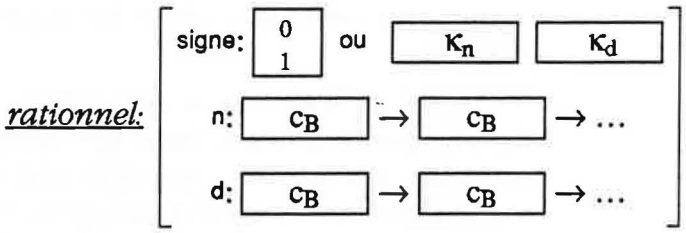


Figure IV.1. Représentation d'un rationnel.

Bien que ceci ne soit en rien obligatoire, il s'avère très profitable de considérer que les numérateurs et les dénominateurs des rationnels ne pourront pas avoir plus qu'un certain nombre de chiffres, assez élevé comme on le verra. Cette restriction permet d'améliorer considérablement l'exécution des algorithmes de comparaison entre entiers et, *a fortiori*, entre rationnels. Si l'on opte pour cette contrainte, on considère que le champ réservé au signe du rationnel est en fait subdivisé en deux sous-champs de même taille, par exemple de la moitié de la taille d'un entier non-signé.

Le sous-champ correspondant aux bits de poids faibles (κ_d sur la figure IV.1) est le codage sur $^{ns}/_2$ bits du nombre de chiffres du dénominateur, dont on sait qu'on peut s'arranger pour qu'il soit toujours strictement positif. Le sous-champ correspondant aux bits de poids fort (κ_n sur la figure) représente le codage sur $^{ns}/_2$ bits du nombre signé de chiffres du numérateur: si ce nombre est positif, le rationnel est supérieur à 0, si ce nombre est négatif, le rationnel est strictement négatif et si ce nombre vaut 0, le rationnel est nul. Comme ce nombre est signé et codé sur $^{ns}/_2$ bits, le plus grand nombre de chiffres que peut contenir le numérateur d'un rationnel est:

$$2^{2^{\frac{ns}{2}-1}-1}$$

soit $2^{15} - 1 = 32767$ chiffres si $ns = 32$ bits.

On peut mesurer l'ordre de grandeur de tels nombres de la façon suivante: le nombre constitué de k fois le chiffre (B-1) en base B est égal à $B^k - 1$. Si $B = 2^{16}$, l'entier maximal recherché est égal à $(2^{16})^{32767} - 1$. Pour exprimer l'ordre de grandeur de ce nombre en base 10, on doit trouver le nombre x tel que

$$(2^{16})^{32767} \cong 10^x \Rightarrow x = 16 \cdot 32767 \cdot \log_{10} 2 \cong 157821.$$

On voit que cette restriction laisse une certaine marge de manœuvre, bien que certaines applications, vraisemblablement liées à des données de taille gigantesque, puissent effectivement donner lieu à la création de valeurs intermédiaires de cet ordre. Par sécurité, on peut inclure dans la bibliothèque rationnelle, une routine de contrôle de débordement de nombre de chiffres.

D'un autre côté, si cette restriction semble injustifiée, on peut toujours utiliser un entier pour garder trace de l'information concernant le nombre de chiffres du numérateur et du dénominateur des rationnels, car si l'application doit se mesurer à de telles quantités, il vaut mieux optimiser coûte que coûte les algorithmes de comparaisons entre entiers.

Quoi qu'il en soit, si l'on accepte la 'restriction' proposée, il ne coûte plus beaucoup, dans une optique d'uniformisation des algorithmes à venir, de considérer que chacun des deux sous-champs précédents est la représentation sur $^{ns}/_2$ bits d'un entier relatif: ceci autorise le même traitement pour les deux sous-champs et 'aligne' les nombres de chiffres du numérateur et du dénominateur.

L'hypothèse implicite faite dans la représentation décrite ci-dessus est que les entiers n'ont pas de signe: on considère en fait que tout entier relatif est un rationnel de dénominateur égal à 1. Comme ce rationnel s'accompagne, lui, d'un signe, l'entier naturel sous-jacent redevient par ce biais relatif. Si les besoins particuliers d'une application nécessitent que l'on représente, de manière autonome, des entiers relatifs, il suffit d'adjoindre à leur représentation propre un bit de signe.

Cette hypothèse permet surtout de simplifier le mode de représentation des entiers: comme son signe se trouve représenté à part, il est possible de coder tout entier *sans ambiguïté* dans le sens croissant de ses exposants. Pour prendre un exemple simple en base 10, le nombre 12340 sera codé $\boxed{0} \rightarrow \boxed{4} \rightarrow \boxed{3} \rightarrow \boxed{2} \rightarrow \boxed{1}$, alors que si le signe était porté, comme cela se fait dans certaines représentations, par le chiffre en tête de liste, le nombre -12340 aurait la même représentation que son opposé et il faudrait recourir au sens inverse de chaînage! Le codage des nombres 'des unités vers les puissances supérieures' permet avant tout d'améliorer de manière très significative la performance des opérations arithmétiques élémentaires, qui nécessitent un 'renversement' systématique des opérandes quand l'ordre de chaînage est inversé.

Débordements

On peut se demander s'il ne serait pas possible de choisir une base qui soit exactement le plus grand entier non signé représentable en machine: après tout, l'avantage de ce choix serait de permettre de diviser par deux le nombre moyen de chiffres manipulés. Mais ceci se ferait au prix d'une attention bien trop grevante.

Il est tout à fait possible de considérer la somme de deux entiers non signés, tous deux inclus dans l'intervalle [0, $M=2^{ns}-1$]. Le résultat, *a priori* composé d'un ou deux chiffres du même intervalle, entraîne donc une éventuelle 'retenue'. Mais, sur une machine, le résultat sera donné *modulo* M. Par exemple, sur un octet, le plus grand nombre représentable est $2^8-1 = 255$. Si on tente d'additionner 255 et 1 sur un octet, on trouve un résultat égal à 0. Dans un ordre d'idée parallèle, la différence 0-1 donne 255.

Plus formellement, si a et b sont deux entiers quelconques, il existe un entier $c_{a+b} \in [0, 2^{ns}-1]$ tel que $a + b \equiv c_{a+b} \bmod (2^{ns})$. Ainsi, dans les deux cas précédents:

$$255+1 = 256 \equiv 0 \bmod (2^8), \text{ et}$$
$$0 - 1 = -1 \equiv -1 \bmod (2^8) \equiv 255 \bmod (2^8)$$

Appelons les deux types de comportement précédents des *débordements*. On dira aussi qu'un résultat est *représentable* s'il ne donne lieu à aucun débordement, dans le contexte décrit plus haut. Ceci revient donc à dire que c_{a+b} et a+b sont les mêmes nombres. Sinon, a+b déborde et on trouve une retenue égale à 1; on peut convenir que les deux cas précédents se notent:

$$a + b < 2^{ns} \Rightarrow a+b \equiv (c_{a+b} = a+b, 0)$$
$$a + b \geq 2^{ns} \Rightarrow a+b \equiv (c_{a+b}, 1)$$

Pour circonvenir les débordements, il est nécessaire de se livrer à la petite gymnastique calculatoire suivante: soient a et b les deux nombres à additionner; si b est nul, le résultat est trivial. Sinon, posons

$$a' = (2^{ns}-1) - a; \; b' = b - 1.$$

Dans l'ensemble des relatifs, on peut toujours écrire

$$a + b = b' - a' + 2^{ns}$$

Si (a' = b'), a + b donne (0, 1). Si (a' < b'), la quantité (b' - a') est positive, il y a retenue et donc a + b donne (b' - a', 1). Si, en dernier lieu, (a' > b'), la quantité (b' - a') est négative et la somme a + b peut être calculée sous la forme directe ou sous la forme (b' - a'), en utilisant le mécanisme d'*underflow* naturel sur la machine.

L'algorithme relatif au contrôle du débordement dans la somme de deux chiffres en base B est assez lourd, celui relatif aux débordements dans la multiplication de deux chiffres est beaucoup plus délicat. Ces deux opérations sont centrales dans l'arithmétique des entiers et leur lourdeur se répercute sur la performance des opérations élémentaires: ce genre de technique n'est conseillé que lorsque l'espace mémoire est tellement compté que les baisses de performances sont peu importantes.

Gestion de la mémoire

L'un des facteurs les plus importants dans l'implantation d'une arithmétique rationnelle est la gestion de la place nécessaire à son fonctionnement. Les opérations élémentaires seront décrites succinctement plus loin, mais on peut déjà deviner que chacune nécessite la réservation de la place requise pour deux opérandes, les créations/destructions de rangées de chiffres pour les résultats intermédiaires et la réservation de la place nécessaire au résultat.

De plus, beaucoup d'opérations créent des résultats qui ne sont utiles qu'en tant que valeurs de décision et doivent être détruits aussitôt même qu'ils sont créés (cf. la comparaison de deux rationnels). Enfin, l'expérience et une réflexion rapide montrent que ce n'est pas la gestion des rationnels qui est la plus lourde, mais, de loin, celle des entiers.

Comme chaque mouvement de création et de libération est relativement coûteux, un moyen efficace pour abaisser les temps d'exécution, à ce niveau, est de créer un 'pool' de chiffres en base B. Il s'agit d'une liste de réserve, qui évolue au gré des libérations et des créations de chiffres pendant les calculs intermédiaires et qu'on laisse grossir jusqu'à une taille maximum, au-dessus de laquelle on procède à une libération-système: cette dernière a pour tâche de faire redescendre la liste de réserve jusqu'à environ la moitié de la taille maximum choisie. Ceci permet de remplacer les séquences d'appels-système 'libération-puis-création', qui sont statistiquement très fréquentes, par une tâche-client consistant, quand une liste de chiffres est libérée, à aller l'insérer en tête de la liste de réserve, de vérifier que cette liste n'est pas trop volumineuse et éventuellement de la réduire; en cas de création cette technique permet d'obtenir beaucoup plus rapidement une zone libre.

IV.A.2 Opérations élémentaires

Les algorithmes décrivant les opérations arithmétiques élémentaires dans [K II] sont proposés dans un cadre assez rigide de représentation des nombres par tableaux. Rares sont les ouvrages ayant retracé ces derniers à la lumière des langages évolués d'aujourd'hui. Le modèle choisi ici fait appel à des structures de listes, beaucoup plus souples mais aussi bien plus délicates à gérer. Cette sous-section permettra de donner aux algorithmes spécifiques de l'arithmétique rationnelle un nouvel éclairage en tentant de capturer l'optimalité des originaux.

Arithmétique des entiers

(≤) Si les signes des entiers ne permettent pas la discrimination, on compare leurs listes de chiffres en les parcourant depuis les unités: si l'une a moins de chiffres que l'autre, elle représente un plus petit nombre; si les listes ont le même nombre de chiffres, le plus petit des deux nombres est celui dans la liste duquel on a relevé un chiffre inférieur au chiffre correspondant du second, pour la dernière fois, dans le parcours depuis les unités. Par exemple, 1547 est inférieur à 1637 car la dernière 'différence' constatée porte sur '5' et '6'.

Si la représentation donne explicitement le nombre de chiffres, on ne doit recourir à la technique précédente que si les deux entiers ont le même signe et le même nombre de chiffres. On se simplifie d'autant la tâche si la convention adoptée est que les sous-champs κ_d et κ_n décrits plus hauts ont le même format. Il est enfin pratique de faire en sorte que la fonction de comparaison retourne la valeur, parmi {<,=,>}, correspondant au résultat du test.

(+) Pour additionner deux entiers naturels u et v de m et n chiffres respectivement, il suffit de parcourir les listes de leur représentation en faisant 'suivre' une retenue en base B.

Dans tous les algorithmes concernant la bibliothèque rationnelle, la fonction Chiffre(c) crée un élément de liste chaînée dont la valeur est le chiffre c en base B passé en paramètre et dont le successeur est automatiquement NIL. 'suivant(x)' donne accès à l'élément suivant x, ou est NIL si x est le dernier élément; 'valeur' permet de consulter ou de mettre à jour la valeur contenue dans un élément de liste.

Dans l'algorithme qui suit, u et v symbolisent les éléments courants des listes chaînées représentant les deux entiers à additionner et s symbolise l'élément courant de la liste chaînée du résultat, notée somme; les pointeurs s et spréc permettent le chaînage naturel des chiffres successifs du résultat (consulter [SED 90], par exemple, pour la gestion de listes linéaires). La variable nbchiffres permet la mise à jour du nombre de chiffres du résultat.

```
retenue, nbchiffres ← 0;
u, v ← début de listes des deux opérandes u et v;
somme, s, spréc ← NIL;
tant que ((u ≠ NIL) ∨ (v ≠ NIL))
    m1 ← (u ≠ NIL)? valeur(u) : 0 ; m2 ← (v ≠ NIL)? valeur(v) : 0 ;
    m3 ← m1+m2+retenue;
    si (m3 ≥ B)
        retenue ← 1;
        m3 ← m3 - B;
    sinon
        retenue ← 0;
    nbchiffres++;
    s ← Chiffre(m3);
    si (somme = NIL) somme ← s;
    sinon suivant(spréc) ← s;
    spréc ← s;
    u ← suivant(u);
    v ← suivant(v);
si (retenue ≠ 0)
    suivant(s) ← Chiffre(retenue));
    nbchiffres++;
```

Algorithme IV-i: Addition de deux naturels u et v.

Si les deux naturels à additionner comportent n et m chiffres en base B, une telle opération a un coût de $O(\max(m, n))$. La somme de deux entiers se déduit de la somme et de la soustraction de deux naturels, puisque la forme de cette opération dépend du signe des opérandes et de la comparaison de leurs valeurs absolues.

(-) Pour la soustraction u - v, on suppose que le premier naturel est supérieur ou égal au second. Avec les mêmes conventions que précédemment, la variable d donne l'élément courant de la liste chaînée différence donnant le résultat. Il faut faire attention à deux points:

L'évaluation de la quantité m_1-m_2-retenue ne doit pas entraîner un résultat 'négatif'. Pour cela, il suffit de décomposer cette opération en deux temps: on évalue m_2+retenue, et si m_1 est supérieur ou égal à cette quantité, on effectue l'opération directement, sinon il y a retenue.
Le résultat de la soustraction peut contenir entre 0 et *m* chiffres! Par exemple, le résultat de la soustraction 10000-9988 en base 10, pourrait bien être 00012, si l'on suivait une logique trop proche de celle de l'addition: si l'on tombe sur un chiffre nul dans le résultat, on doit attendre le prochain chiffre non nul pour décider comment agir; si aucun chiffre non nul n'est de nouveau rencontré, le dernier chiffre créé est le dernier chiffre du nombre et tout va bien; sinon, il suffit d'avoir compté le nombre z de zéros qui se sont accumulés depuis le dernier chiffre non nul, de créer cette sous-chaîne de z zéros suivis du nouveau chiffre trouvé et de marquer cette nouvelle position comme le nouveau dernier chiffre non nul rencontré. L'instruction contenant l'appel de la procédure ChaînerZéros est une écriture symbolique raccourcie de l'opération consistant à ajouter z zéros au bout d'une liste (éventuellement vide), à mettre à jour (éventuellement) la tête de liste, à retourner l'élément courant et à reconnaître son prédécesseur!

```
retenue, nbchiffres, z ← 0;
u, v ← début de listes des deux opérandes u et v;
différence,dernierchiffre, dpréc ← NIL;
tant que (u ≠ NIL)
    m1 ← valeur(u);
    m2 ← (v ≠ NIL)? valeur(v) : 0;
    m3 ← m2 + retenue;
    si (m1 ≥ m3)
        retenue ← 0;
        m3 ← m1 - m3;                          ¡m3 ← m1-m2-retenue¡
    sinon
        retenue ← 1;
        m3 ← B+ m1 - m3;                       ¡m3 ← B+m1-m2-retenue¡
    si (m3 ≠ 0)
        si (z>0)
            différence, d, dpréc ← ChaînerZéros( z);   ¡ajout de z zéros¡
            nbchiffres ← nbchiffres+z;
            z ← 0;
        d ← Chiffre(m3);                       ¡insertion du chiffre non nul¡
        nbchiffres++;
        si (différence = NIL) différence ← d;
        sinon suivant(dpréc) ← d;
        dpréc ← d;
    sinon z ← z+1;
    u ← suivant(u);
    v ← suivant(v);
```

Algorithme IV-ii: Soustraction de deux naturels u et v, u ≥ v.

Pour additionner deux relatifs, on doit comparer les valeurs absolues des deux nombres et traiter les quatre cas de relations possibles. La soustraction a - b se traite facilement en effectuant à la place a + opposé(b), l'opposé d'un relatif s'obtenant trivialement. La complexité de toutes ces opérations est, bien entendu, identique à celle de l'addition de deux naturels, les comparaisons implicites étant proportionnelles au nombre de chiffres des deux entiers.

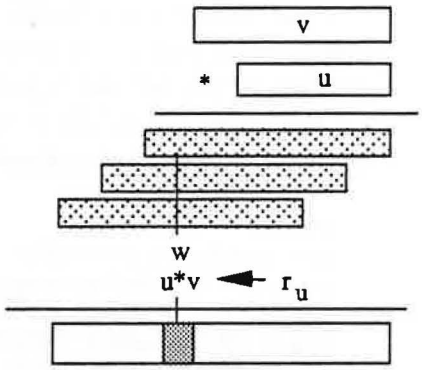


Figure IV.2. Multiplication de deux entiers.

(*) Le mécanisme de multiplication de deux naturels est plus simplement décrit par le schéma fonctionnel de la figure IV.2. Sur cette figure symbolisant la multiplication des entiers u et v, l'opération est représentée au stade où s'effectue le produit des chiffres u et v. Les rectangles grisés représentent les résultats successifs des multiplications des chiffres précédant u, avec v. Le produit u*v 'hérite' donc la somme w modulo B des résultats précédemment obtenus 'à la verticale' du chiffre correspondant à ce produit.

D'autre part, comme d'habitude, une retenue r_u est propagée depuis la droite, toujours à l'actif de u qui est le multiplicande courant. Le résultat de la somme

(IV-2) $$w+u*v+r_u$$

donnera le nouveau w calculé et une nouvelle retenue.

Le plus petit nombre de k chiffres en base B est B^{k-1}. Ceci permet d'affirmer que le produit d'un naturel de m chiffres par un naturel de n chiffres possède au moins $m+n-1$ chiffres. De même, le plus grand nombre de k chiffres en base B est B^k-1. On en déduit que le produit des deux nombres précédents a au plus $(m+n)$ chiffres.

Si l'on est sûr que les deux nombres sont non nuls, ce qui doit obligatoirement être vérifié en premier, on peut dès le départ construire une liste de $(m+n)$ 0 en base B: ceci permettra que toute rencontre de zéros dans le multiplicande soit entérinée d'avance. Le dernier élément de cette liste est en fait un élément spécial, propre à la librairie rationnelle: il sert à éviter les mouvements de mémoire inutiles et, pour cela, a été créé une fois pour toutes.

Que le dernier chiffre du résultat soit différent de zéro ou non, cet élément 'bagué' est restitué à la bibliothèque en fin d'algorithme, mais dans le premier cas, la valeur qu'il contient sert à créer un dernier chiffre à insérer en fin de liste et le nombre de chiffres est égal à $(m+n)$. Dans l'autre cas, on rompt le dernier chaînon de la liste du produit et le nombre de chiffres est égal à $(m+n-1)$.

Dans l'algorithme suivant pour la multiplication de deux entiers naturels, le pointeur v parcourt le multiplicande v et u parcourt le multiplicateur u. Si v se trouve à la position i dans le premier nombre et u à la position j dans le second, p, qui parcourt la liste produit, se trouve alors à la position $(i+j)$ dans celle-ci.

Pour réaliser ceci, il suffit, à chaque retour de boucle sur le premier opérande, d'avancer la position de départ de p d'une position vers la gauche. Enfin, la mise à jour de la variable nbchiffres n'est faite qu'à la fin puisqu'on sait qu'elle ressort de l'opération avec la valeur $(m+n)$ ou $(m+n-1)$, m et n étant le nombre de chiffres respectif de u et v.

```
retenue ← 0;
u ← début de liste du multiplicateur u;
produit ← liste chaînée remplie de (m+n) zéros;
avt_dernierp: avant dernier élément de la liste du produit;
bague: élément spécial marquant la fin de la liste du produit;
p ← début de liste de produit;
tant que (u ≠ NIL)
    m2 ← valeur(u);
    si (m2 ≠ 0)
        v ← début de liste du multiplicande v;
        w ← p;
        tant que (v ≠ NIL)
            m1 ← valeur(v);
            m3 ← m1*m2+w+retenue;
            retenue ← ⌊m3/B⌋;
            valeur(w) ← m3-retenue*B;
            w ← suivant(w);
            v ← suivant(v);
    u ← suivant(u);
    p ← suivant(p);
si (valeur(bague ≠ 0))
    suivant(avant_dernierp) ← Chiffre(valeur(bague));
    nbchiffres ← m+n;
sinon
    suivant(avant_dernierp) ← NIL;
    nbchiffres ← m+n-1;
restituer bague à la bibliothèque;
```

Algorithme IV-III: Multiplication de deux naturels non nuls u et v.

On a vu lors du calcul de la base que la quantité '$m_1*m_2+w+retenue$' avait été prise en compte pour qu'aucun débordement ne puisse avoir lieu. La complexité de cet algorithme, qui fait intervenir deux boucles imbriquées, est naturellement $O(mn)$ et est identique à celle relative à la multiplication de deux relatifs puisque cette opération ne nécessite en plus que de considérer le produit des signes des deux nombres.

Remarque: Il existe plusieurs techniques de multiplication des entiers longs dont la complexité est asymptotiquement meilleure que la méthode présentée ci-dessus. Certaines de ces méthodes s'inspirent du principe 'diviser pour résoudre' qui permet de subdiviser un problème en deux sous-problèmes moins volumineux que l'on résout récursivement et dont la fusion donne la solution complète.

Il est aussi connu que ces méthodes ne deviennent réellement plus efficaces que pour des nombres dépassant une longueur de 500 chiffres. Malgré tout, il n'a pas été utile de recourir à de telles méthodes pour les besoins du travail présenté dans cette thèse.

Le lecteur intéressé trouvera les références bibliographiques et quelques explications concernant certaines de ces méthodes dans [A/H/U 87, pages 315-317, et page 354].

(/) La division euclidienne des deux naturels u et v, de $m+n$ et n chiffres respectivement permet de trouver deux naturels uniques, q et $r \in [0, v[$, de $m+1$ et n chiffres, tels que $u = vq + r$. Si l'on multiplie les deux naturels par la même quantité D, le quotient est le même, mais le reste est multiplié par D: cette propriété sera exploitée un peu plus loin

On traite à part deux cas particuliers mutuellement exclusifs en les détectant à l'aide de la procédure de comparaison déjà décrite: si u est inférieur à v, q=0 et r=1 et d'autre part, si u est égal à v, q=1 et r=0.

Une fois ces cas éliminés, u ne peut plus qu'être strictement supérieur à v. Un dernier cas particulier reste à traiter: celui où v a un seul chiffre. Si u est aussi réduit à un seul chiffre, la division est encore directe. En revanche si u a plusieurs chiffres, il suffit de parcourir la liste des chiffres de ce nombre depuis le plus significatif et de diviser de proche en proche chaque chiffre par v. L'idéal est donc de renverser u et de reconstruire le chaînage original de ce nombre au fur et à mesure que la division est effectuée. Un seule précaution reste à prendre: le premier chiffre du quotient obtenu peut fort bien être nul et il faut donc traiter ce cas à part, en début d'algorithme.

Dans l'algorithme proposé, Inversion(x) est une procédure chargée de renverser le chaînage de la liste représentant x; le chaînage de la liste du quotient est effectué 'à reculons' puisque les chiffres sont trouvés depuis les exposants les plus forts jusqu'aux unités.

Comme le chaînage de u a été inversé, il est indispensable de procéder, simultanément, au retournement définitif de u. Ceci est fait par l'appel (upréc, u) ← Suiv+Inv(upréc, u) qui a pour but de décaler l'élément u vers la position suivante dans la liste à laquelle il appartient, et de renverser son chaînage: upréc devient u, u devient suivant(u) et le suivant de l'ancien u devient l'ancien upréc, comme le montre la figure IV.3: il y a toujours rupture initiale entre l'élément u et celui qui le précédait, dans u, avant le rechaînage: ceci nécessite de toujours garder trace de cet élément, upréc, dans l'algorithme. Lorsque la condition (u = NIL) est atteinte, c'est en fait upréc qui pointe sur le début de liste de u.

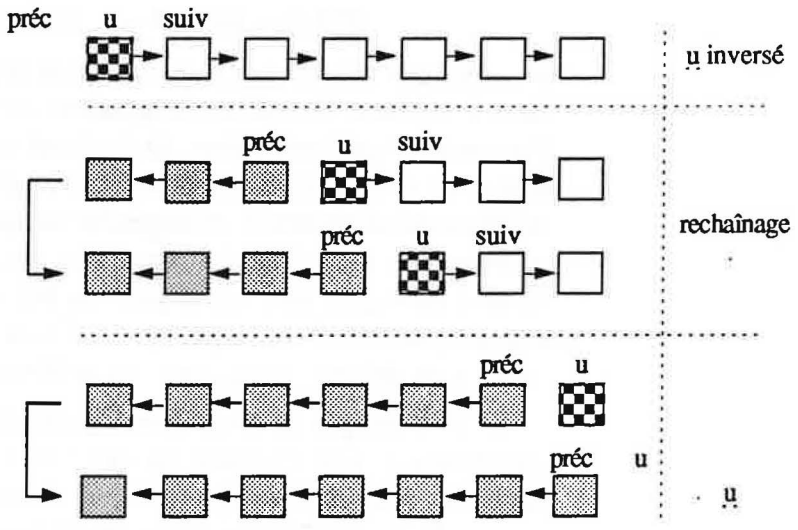


Figure IV.3. Retournement du chaînage relatif à u.

```
u, v ← début de liste des deux opérandes;
si (v ≤ Base)
  V ← valeur(v);
  si (u ≤ Base)
    U ← valeur(u);
    retourner(q ← Chiffre(⌊U/V⌋), r ← Chiffre(U-⌊U/V⌋*V));
  u ← début de liste de Inversion(u);
  reste ← 0;
  upréc ← NIL;
  répéter                                        ¡éliminer 1er chiffre nul¡
    quotient ← ⌊((dividende ← valeur(u))+reste)/V⌋;
    reste ← dividende-quotient*V;
    (upréc, u) ← Suiv+Inv(upréc, u);
  jusqu'à (quotient ≠ 0);
  q ← Chiffre(quotient);
  qpréc ← NIL;
  tant que (u ≠ NIL)
    quotient ← ⌊((dividende ← valeur(u))+reste)/V⌋;
    reste ← dividende-quotient*V;
    q ← Chiffre(quotient);
    suivant(q) ← qpréc;
    qpréc ← q;
    (upréc, u) ← Suiv+Inv(upréc, u);
  si (reste = 0)
    retourner(q, NIL);
  retourner(q, r ← Chiffre(reste));
```

Algorithme IV-iv: Division de u par v: cas où v a un seul chiffre.

Reste maintenant le cas général, où u est strictement supérieur à v, et où ces deux nombres ont deux chiffres, ou plus.

Comme cette opération nécessite un certain savoir-faire quand on l'effectue à la main, ce que l'on va essayer de faire ici est de dégager un moyen pour uniformiser tous les cas. Si l'on compare, dans un intérêt de dénombrement simple, les deux divisions:

$$2345 = 97*24+17,\ 24\,27 = 101*24+3$$

on constate, bien que le dividende et le diviseur ait le même nombre de chiffres dans les deux cas, qu'on ne pas en dire autant du quotient et du reste. En revanche, si l'on adopte la convention d'écriture suivante:

$$02345 = 097*24+17,\ 0242\,7 = 101*24+03$$

on voit cette fois-ci que, dans les deux cas, le dividende a (3+2) chiffres, le diviseur 2 chiffres, le quotient 3 chiffres et le reste 2 chiffres. Par souci d'uniformisation, la division euclidienne sera dite de type $(m+n, n)$ si elle porte sur deux entiers u et v de $(m+n)$ chiffres et n chiffres respectivement, et engendre un quotient q de $(m+1)$ chiffres et un reste r de n chiffres, étant entendu que le chiffre le plus significatif de u, q ou r peut être nul (r peut en fait être nul!). L'algorithme doit nécessairement s'assurer, en phase finale, que les nombres fournis en sortie sont débarrassés de tous ces chiffres parasites.

Si l'on adopte la convention précédente, on sait par expérience qu'effectuer une division de type $(m+n)$ revient à répéter $(m+1)$ divisions réduites de type $(n+1, n)$ (voir la figure IV. 4). Il est donc naturel d'essayer de résoudre ce problème en premier.

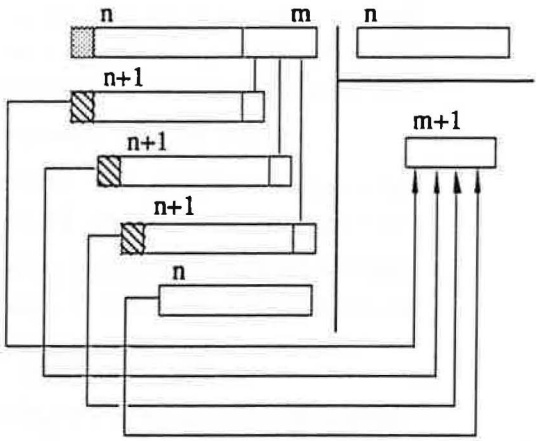


Figure IV.4. Une division est la répétition de divisions réduites.

En partant de la technique utilisée dans le calcul d'une division réduite posée 'à la main', on tente de trouver, à chaque étape, une valeur d'essai pour le quotient, notée $\hat{q}$, qui soit toujours supérieure ou égale au véritable quotient, et qui s'en éloigne le moins possible.

La connaissance des deux premiers chiffres (u_{N+1} et u_N) du dividende et du chiffre le plus significatif (v_N) du diviseur suffisent à 'estimer' une assez bonne valeur d'essai, donnée par:

$$\hat{q} = \left\lfloor \frac{u_{N+1}B+u_N}{v_N} \right\rfloor$$

Comme u_N peut prendre toute valeur entre 0 et B-1, $\hat{q}$ peut varier, pour un même u_{N+1}, entre $\left\lfloor \frac{u_{N+1}B}{v_N} \right\rfloor$ et $\left\lfloor \frac{u_{N+1}B}{v_N} + \frac{B-1}{v_N} \right\rfloor$.

La meilleure contrainte naturelle que l'on pourrait souhaiter trouver est que v_N ait une valeur aussi grande que possible: il est facile de voir que si ($v_N > \lfloor B/2 \rfloor$) les variations de $\hat{q}$ sont minimales. Si l'on multiplie le diviseur par la quantité

$$D = \lfloor B/(v_N+1) \rfloor$$

(opération qualifiée de *normalisation*[1]), on obtient un nouvel entier $\boxed{v}$ de même longueur que le précédent et dont le chiffre le plus significatif, $\boxed{v_N}$, vérifie la condition précédente. Pour ne pas changer la nature de la division, le dividende est multiplié par la même quantité D. D'un point de vue algorithmique, cette normalisation du dividende revient à partir du dividende de *m+n* chiffres original, à lui ajouter un 0 en tête, puis à multiplier, chiffre à chiffre, ce nombre par D.

On désigne par $\boxed{\alpha}$, $\boxed{\beta}$ et $\boxed{\gamma}$ et par $\boxed{v_N}$ et $\boxed{v_{N-1}}$ les chiffres les plus significatifs du dividende et du diviseur normalisés. Si l'on choisit $\hat{q}$, le quotient partiel d'essai, de la manière suivante

si ($\alpha = \boxed{v_N}$) $\hat{q} \leftarrow$ B-1

sinon $\hat{q} \leftarrow (\boxed{\alpha}B+\boxed{\beta})$ **div** $\boxed{v_N}$

[1] Dans toute cette section, la notation $\boxed{x}$ signifiera que la quantité x est normalisée.

dont on peut voir qu'elle équivaut à toujours choisir pour $\hat{q}$ la plus petite des deux quantités en lice, on s'assure que le cette quantité reste bien un chiffre en toute circonstance et l'on peut montrer que

(IV-3) $$\hat{q} - q \in [0, 2]$$ (Théorèmes A et B, [K II, pp. 256-7])

Cela signifie que $\hat{q}$ est toujours supérieur au quotient réel et au maximum de deux unités. En utilisant $\boxed{\gamma}$, le troisième chiffre significatif du dividende courant, on peut affiner le choix de $\hat{q}$ en décrémentant sa valeur d'une unité si l'on constate que

$$\hat{q}(\boxed{v_N}B+\boxed{v_{N-1}}) > (\boxed{\alpha}B^2 + \boxed{\beta}B + \boxed{\gamma})$$

Malheureusement, cette inégalité ne peut se vérifier sans débordement sous cette forme (qui permettrait le pré-calcul de $\{\boxed{v_N}B+\boxed{v_{N-1}}\}$ pour toute la division); il est nécessaire de la vérifier sous la forme suivante, toujours représentable mais moins directe et plus coûteuse

(IV-4) $$\hat{q}\boxed{v_{N-1}} > (\boxed{\alpha}B + \boxed{\beta} - \hat{q}\boxed{v_N})B + \boxed{\gamma}$$

Dans certains cas, une fois cette condition vérifiée, le quotient d'essai doit encore être décrémenté. On ne peut s'en apercevoir qu'après évaluation de la différence

(IV-5) $$reste_{courant} \leftarrow reste_{courant} - \hat{q}*diviseur$$

qui apporte une ultime retenue non nulle: il suffit alors de ne pas prendre celle-ci en compte et de rajouter chiffre à chiffre la valeur du diviseur au reste courant.

Le choix de $\hat{q}$ amène à une petite réflexion: puisque l'on décide de prendre B-1 chaque fois que $\alpha = \boxed{v_N}$, que se passe-t-il lorsque l'on divise 1230 par 123, en base 10 par exemple? Apparemment, ce choix empêche de trouver 10 comme résultat! Ceci est faux, comme le montrent les résultats suivants: $D=\lfloor 10/(1+1)\rfloor=5$ et $\boxed{v}=615$, $\boxed{v_N}=6$ et $\boxed{u}=0615$, car le nouveau dividende doit avoir un chiffre de plus que l'ancien. Comme $\boxed{\alpha}\neq\boxed{v_N}$ on choisit $\hat{q}=\lfloor(0.10+6)/6\rfloor=1$.

Puisque le résultat (IV-3) est indépendant de la base et que la seule contrainte relative à v porte sur le chiffre le plus significatif, il peut se reporter de proche en proche et suggère une méthode de division efficace et assez facilement implantable lorsque les nombres sont représentés par des tableaux. Comme ils sont représentés ici par des listes chaînées, il est nécessaire de pousser encore un peu plus loin l'analyse.

Si l'on s'arrange au départ pour placer le dividende normalisé $\boxed{u}$ (u' sur la figure IV.5) dans une liste annexe de $(m+n+1)$ éléments, celle-ci peut couvrir entièrement les besoins des calculs. Comme chaque division réduite s'accompagne de la 'perte' d'un chiffre du reste courant, on peut récupérer l'élément de liste correspondant pour le chiffre courant du quotient. A chaque étape, le nouveau quotient partiel est placé dans un élément ainsi 'recyclé' et ce dernier se voit attribuer comme successeur définitif, l'élément qpréc, initialement mis à NIL, contenant le quotient partiel obtenu à l'étape précédente.

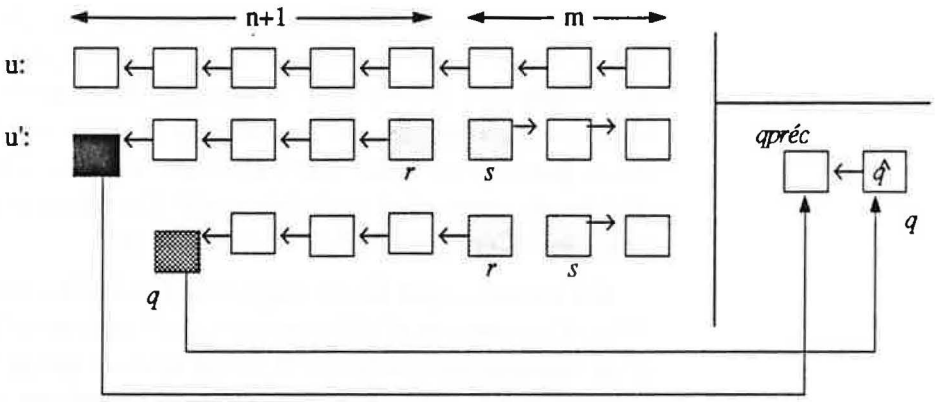


Figure IV.5. Une liste pour le dividende courant, le reste et le quotient.

Pour revenir à la construction de la liste annexe pour u', l'astuce est de scinder la liste initiale en deux sous-listes: la liste de gauche contient $(n+1)$ éléments, est chaînée dans le sens naturel du chaînage des naturels et débute par l'élément r. A la fin de l'algorithme, r indiquera le début de la liste chaînée *dans le bon sens* matérialisant le reste. La liste de droite contient m éléments, est chaînée dans le sens opposé et débute par l'élément s. A la fin de l'algorithme, s pointera sur NIL.

La construction de cette liste peut se faire en une seule passe dès que l'on connaît le nombre de chiffres des deux nombres initiaux, en effectuant la multiplication chiffre à chiffre de u par D en deux temps: tant que l'on n'a pas atteint le m-ième chiffre de u, on chaîne chaque élément obtenu pour le produit vers son prédécesseur chronologique; on positionne le pointeur s sur le m-ième élément et le pointeur r sur le $(m+1)$-ième élément, puis on termine l'opération en chaînant, après r, tout élément sauf le dernier sur son suivant chronologique.

Pendant cette opération, on a aussi positionné trois pointeurs supplémentaires, p_α p_β et p_γ, qui permettront de connaître à tout moment les trois chiffres les plus significatifs du reste courant. A chaque étape, on pourra profiter de l'opération (IV-5) pour faire avancer ces trois pointeurs d'un cran vers la droite, grâce à un jeu de variables positionnelles. De même, à la fin de chaque étape, il suffira d'inverser le chaînage en s et de faire avancer r et s d'un cran vers la droite.

Tout serait parfait si la normalisation n'imposait pas, en fin d'algorithme, de rediviser reste et diviseur par D, alors que ces deux listes sont chaînées dans le mauvais sens pour une telle opération. On pourrait recourir à l'algorithme (IV-iv) de division par un chiffre, mais on peut encore faire mieux.

Tout bien considéré, la normalisation ne sert qu'une cause: permettre de trouver les valeurs d'essai les plus proches des quotients partiels réels. On peut très bien concevoir une nouvelle méthode dans laquelle on calcule effectivement D, $\boxed{v_N}$ et $\boxed{v_{N-1}}$ une fois pour toutes, sans chercher effectivement à garder le diviseur normalisé. Si l'on trouve un moyen d'exploiter ces données pour calculer $\hat{q}$, la soustraction (IV-5) de chaque étape peut tout aussi bien être effectuée avec le diviseur original, ce qui a au moins l'avantage de produire le vrai reste à la fin.

Comment concilier ces deux points de vue? Si l'on suppose que la liste u' de la figure IV.5 est en fait une copie pure et simple de u, il est nécessaire de connaître, pour chaque détermination *initiale* de $\hat{q}$, les valeurs courantes des deux premiers chiffres du dividende normalisé: est-il possible de connaître ces deux chiffres sans avoir à calculer le dividende normalisé explicitement? La réponse est oui, grâce à une propriété remarquable, due au choix de D.

On constate que D est toujours, par définition, inférieur ou égal à $\lfloor B/2 \rfloor$. Ceci permet d'affirmer que, dans une multiplication quelconque d'un nombre par le chiffre D, toute retenue est au plus égale à $\lfloor B/2 \rfloor$: au premier rang, il n'y a pas de retenue antérieure à reporter, et il suffit d'écrire que la retenue future est au maximum

$$D(B-1) \textbf{ div } B < DB \textbf{ div } B = D \leq \lfloor B/2 \rfloor$$

Au second rang, la nouvelle retenue à calculer provient, au pire à cause de la retenue précédente, de

$$r_A = [D(B-1)+B/2] \textbf{ div } B \leq [B/2(B-1)+B/2] \textbf{ div } B = \lfloor B/2 \rfloor$$

Ceci montre bien la propriété annoncée. Quelle est la portée d'une telle retenue? Comme le produit d'un chiffre quelconque x par D est inférieur ou égal à $\lfloor B/2 \rfloor(B-1)$, la retenue 'engendrée' par un tel produit est au plus, indépendamment de la retenue provenant de calculs antérieurs

$$R_x = (Dx) \textbf{ div } B \leq \lfloor B/2 \rfloor(B-1) \textbf{ div } B < (B-1)/2$$

Si l'on regarde maintenant la retenue obtenue après ajout d'une éventuelle retenue antérieure r_A, on obtient

$$R'_x = (Dx + r_A) \textbf{ div } B \leq [B/2(B-1)+B/2] \textbf{ div } B \leq B/2$$

Comme $R'_x \geq R_x$, la différence entre les deux retenues est, au pire:

$$R'_x - R_x \leq B/2 - (B-1)/2 = 1/2$$

Autrement dit, dans la multiplication d'un entier par le chiffre $D \leq \lfloor B/2 \rfloor$, aucune retenue ne peut être reportée sur plus d'une position. En particulier, si l'on commence le calcul de D*u au *troisième* chiffre significatif de u, le produit relatif aux deux premiers chiffres est forcément exact, le troisième ne l'étant 'qu'à la retenue près'.

Ce résultat est crucial, puisqu'il implique que l'on peut calculer le dividende normalisé morceau par morceau: bien qu'il soit nécessaire, au départ, de calculer les trois produits relatifs aux chiffres les plus significatifs de u, dès l'étape suivante, il n'y a plus qu'un seul produit à calculer: $\boxed{\alpha}$ devient l'ancien $\boxed{\beta}$, dont on sait que sa valeur est exacte, sans plus de calculs; il faut ensuite recalculer $\boxed{\gamma}$ pour obtenir la retenue à reporter sur $\boxed{\beta}$, c'est-à-dire l'ancien $\boxed{\gamma}$, pour lequel la valeur de retenue n'était pas disponible. Les nouvelles valeurs de $\boxed{\alpha}$ et $\boxed{\beta}$ sont exactes et celle de $\boxed{\gamma}$ est, de nouveau, exacte à la retenue près.

De plus, en ce qui concerne la 'fausse' normalisation du diviseur, seule est nécessaire la connaissance du chiffre le plus significatif du diviseur normalisé: ce qui vient d'être dit signifie qu'il n'est pas utile de multiplier plus que les deux chiffres les plus significatifs du diviseur original par D!

Enfin, du fait de l'identité entre le quotient réel et le quotient obtenu par normalisation, il est clair que le test (IV-4) d'affinement des valeurs d'essai peut maintenant être effectué à l'aide des chiffres les plus significatifs du dividende et du diviseur originaux, d'où le nouveau test

(IV-6) $$\hat{q}v_{N-1} > (\alpha B + \beta - \hat{q}v_N B + \gamma)$$

(α, β, γ) symbolisent les trois chiffres les plus significatifs du 'vrai' reste courant à chaque étape. En général, le fait que les opérandes normalisés soient remplacés par les opérandes originaux ne change en rien la logique de ce qui a pu être dit sur l'algorithme.

Par simple comparaison avec celui de [K II], l'algorithme présenté ici montre un certain nombre de différences:

- le dividende est multiplié par D une fois, mais ni le reste ni le diviseur ne doivent être redivisés par D à la fin;
- seuls le nombre constitué des deux chiffres les plus significatifs du diviseur est multiplié par D;
- on effectue les mêmes tests et opérations que dans l'algorithme de Knuth, mais rapportés aux opérandes initiaux;
- il est nécessaire, comme c'est souvent le cas dans la gestion de listes chaînées, de disposer d'un petit peu plus de variables positionnelles pour permettre de parcourir les listes dans les deux sens.

Pour optimiser l'espace utilisé dans l'implantation de la division, il faut encore se pencher sur la structure des résultats obtenus. On a vu que le quotient peut commencer par un zéro qui provient en fait de l'adjonction d'un chiffre supplémentaire en tête du dividende. On peut tout à fait envisager de toujours représenter ce chiffre par un élément 'bagué': il s'agit d'un chiffre créé une fois pour toutes, par exemple lors d'une initialisation générale de la librairie rationnelle, et destiné à matérialiser le chiffre le plus significatif du dividende. Si son contenu final est non nul, on crée un véritable élément de liste, par lequel finira le quotient, pour recevoir le chiffre correspondant; sinon, il suffit de restituer cet élément à la bibliothèque sans autre forme de procès. Le cas du reste est plus délicat, dans le sens où celui-ci peut prendre toute valeur entre 0 et (v-1), ce qui veut dire que l'algorithme peut aboutir sur un reste ne contenant que des zéros ou simplement plusieurs zéros en fin de liste du reste. On a déjà vu, dans le cadre de la soustraction, comment localiser le dernier chiffre non nul d'un résultat et utiliser cette position pour retomber sur un nombre conforme. Comme il est impossible de connaître à l'avance le nombre de zéros obtenus en tête du reste (ou de la différence), on trouve là une justification naturelle de l'utilisation d'une 'réserve' élastique de chiffres, prête à amortir les coûts de séquences création/destruction répétitive, pour ces deux opérations.

L'algorithme de division euclidienne de $\underline{u}$ par $\underline{v}$, représentés par des listes linéaires de longueurs respectives $m+n$ et n, donnant le quotient et le reste sous forme de listes chaînées q et r de longueurs respectives $m+1$ et n peut maintenant être décrit. Pour raccourcir son écriture, on suppose que l'on a déjà connaissance des longueurs des deux nombres, de D de v_N et de v_{N-1}. On débute par la construction de la copie de $\underline{u}$, sous forme de deux sous-listes de chaînages opposés:

```
retenue, compteur ← 0;
p, qprec, r, s, s_suiv, u', u'préc ← NIL;
u,v ← début de listes du dividende et du diviseur;
[v_N], [v_{N-1}] ← D*(B*v_N+v_{N-1});          ¡fausse normalisation de v¡
pour (compteur ← 1; compteur≤m+n+1; compteur++)   ¡copie de u¡
    si (compteur≤m+n)
      u' ← Chiffre(valeur(u));
    sinon u' ← Chiffre(0);                      ¡n+m+1-ième chiffre¡
    si (compteur≤m)
      suivant(u') ← u'préc;
    si (compteur = m)
        s ← u';                                 ¡marquage de s¡
        u'préc ← NIL;                           ¡départ de chaînage inverse¡
    sinon
        suivant(u'préc) ← u';
        si (compteur = m+1)
          r ← u';                               ¡marquage de r¡
    u'préc ← u';
    si (compteur = m+n-1)
      p_γ ← u;
    sinon
      si (compteur = m+n)
        p_β ← u;
      sinon si (compteur = m+n+1)
          p_α ← u';                             ¡t serait NIL pour p_α¡
          q ← u';                               ¡départ du quotient¡
    t ← suivant(t);
si ([γ] ← D*val(p_γ)≥ B)
    retenue ← 1; [γ] ← [γ]-B;
si ([β] ← D*val(p_β)+retenue≥ B)
    retenue ← 1; [β] ← [β]-B;
[α] ← D*val(p_α)+retenue;                       ¡plus de retenue possible¡
tant que (s ≠ NIL)                              ¡boucle principale¡
    si ([α] = [v_N]) q̂ ← B-1;
    sinon           q̂ ← ([α]B+[β]) div [v_N];
    si (q̂*v_{N-1} > (αB + β - q̂*v_N)*B + γ) q̂- -;
    r ← (r-q̂*v); ¡chiffre à chiffre depuis r et v: pendant cette opération,
                  faire avancer p_α, p_β, p_γ d'un cran vers la droite¡
    si (la dernière retenue est négative)
        r ← r+v;                                ¡chiffre à chiffre depuis r et v¡
        q̂- -;
    si ((q̂ ≠ 0) ∨ (qprec ≠ NIL))
        valeur(q) ← q̂;
        suivant(q) ← qprec;
        qprec ← q;
    sinon
      libérer q;                    ¡1er chiffre du quotient ne doit pas être nul¡
    q ← p_α;                        ¡q toujours donné par le chiffre le + significatif¡
    s_suiv ← suivant(s);
    suivant(s) ← r;                             ¡inversion du chaînage en s¡
    r ← s;
    s ← s_suiv;
r ← r privé des zéros de tête;                  ¡r peut être NIL après ceci¡
retourner(q, r);
```

Algorithme IV-5: Division de deux naturels u et v, cas général.

La complexité de cet algorithme est aussi proportionnelle au produit des longueurs des deux nombres. La division de deux relatifs se déduit trivialement de l'étude précédente.

(pgcd) Une fois acquise la division, on peut aussi construire l'opération pgcd qui donne le plus grand naturel divisant exactement à la fois les naturels u et v en appliquant l'algorithme d'Euclide ou une méthode dite 'binaire' présentée en détail dans [K II, pp. 321-323]. Le pgcd permet de construire l'opération IRR qui retourne la fraction irréductible représentant le rationnel r: cette dernière est importante pour toujours obtenir des représentants irréductibles des rationnels manipulés ou donnés en entrée.

Arithmétique des rationnels

Une fois définies les opérations sur les entiers, les opérations arithmétiques sur les rationnels se déduisent assez simplement.

Si les rationnels $r_1 = \frac{n_1}{d_1}$ et $r_2 = \frac{n_2}{d_2}$ ont le même signe, on les compare grâce aux produits croisés numérateur-dénominateur $n_1 * d_2$, $d_1 * n_2$. On peut éviter, dans de nombreux cas, de calculer *effectivement* ces deux produits, lorsque la représentation des entiers inclut explicitement leur taille. Supposons que tn_i et td_i i=1,2, indiquent la taille, respectivement, des numérateurs et des dénominateurs des deux rationnels: si l'on trouve $|tn_1 \cdot td_2 - tn_2 \cdot td_1| > 1$, on en déduit qu'un des produits possède, au plus, un chiffre de moins que l'autre et lui est nécessairement inférieur; le signe permet ensuite de déterminer l'ordre relatif des deux rationnels.

On ne peut malheureusement rien dire si cette relation n'est pas vérifiée et il faut recourir au calcul effectif des produits croisés entiers mentionnés plus haut; le coût d'une comparaison entre rationnels est donc, dans le pire des cas, proportionnel à la somme des chiffres des couples croisés numérateur-dénominateur, et non plus constant ($O(1)$) comme celui de la comparaison de deux pseudo-réels en précision finie!

Remarque: On retrouve ici une des 'bêtes noires' de l'implantation de l'algorithme de triangulation de Delaunay en arithmétique rationnelle du chapitre précédent: le coût des dégénérescences du type ($\Delta \neq 0$) est bien le plus élevé puisque, dans la vision que ces auteurs ont des choses, les approximations successives des déterminants ne peuvent 'converger' rapidement que si les intervalles successifs obtenus ne contiennent pas la valeur 0, les comparaisons devenant alors gargantuesques.

Pour éviter que les entiers représentant les numérateurs et les dénominateurs n'aient pas trop tendance à croître en taille de manière incontrôlée, il est judicieux de toujours ramener les rationnels considérés à leur expression irréductible. La fonction IRR, citée plus haut, permet d'effectuer cette transformation. Si la fraction n/d est un représentant du rationnel r positif, et si π est le pgcd de ces deux naturels, le représentant irréductible de r est donné par

$$IRR(r) = \frac{n/\pi}{d/\pi}$$

Une remarque de [K/L/N 89], déjà cité, concerne le fait que le recours au pgcd n'est pas toujours indispensable, voire peut s'avérer un élément freinant dans l'exécution de calculs en arithmétique exacte. Si cette remarque peut être justifiée dans le cas de calculs effectués sur des données aléatoires, elle est moins fondée quand les données proviennent de fichiers 'naturels'. Statistiquement, dans ce genre de situations, la simplification des fractions rationnelles permet souvent de réduire la taille des entiers manipulés.

La somme et la différence des rationnels $r_1 = \frac{n_1}{d_1}$ et $r_2 = \frac{n_2}{d_2}$ s'expriment sous forme d'un troisième rationnel

(IV-7) $$r_{s,d} = \frac{n_1 * d_2 \pm n_2 * d_1}{d_1 * d_2}$$

Le rationnel irréductible représentant le résultat de ces deux opérations peut s'obtenir à l'aide de la fonction IRR décrite plus haut. Une méthode plus complexe mais souvent efficace consiste à utiliser les pgcd à l'intérieur des calculs.

Dans un premier temps, on cherche le pgcd des deux dénominateurs, $p_D = \text{pgcd}(d_1, d_2)$. Si celui-ci est différent de 1, il va permettre de réduire une première fois le rationnel $r_{s,d}$:

(IV-8) $$r_{s,d} = \frac{p_D}{p_D^2} * \frac{n_1 * d_2' \pm n_2 * d_1'}{d_1' * d_2'} = \frac{1}{d_1' * d_2'} * \frac{n_1 * d_2' \pm n_2 * d_1'}{p_D}$$

Il se peut encore que le numérateur et le dénominateur de la fraction de droite admettent un pgcd différent de un, auquel cas on pourra encore réduire la fraction de (IV-8) et obtenir le représentant irréductible du rationnel somme ou différence. Cette technique permet d'éviter, dans les cas favorables, la création de quantités intermédiaires beaucoup plus grandes que les quatre entiers définissant les deux opérandes rationnels.

La multiplication et la division de deux rationnels s'obtiennent par application directe des définitions mathématiques de ces opérations et du pgcd.

Un modèle deux fois plus compact

Dans toute solution faisant usage d'une arithmétique rationnelle, un des coûts les plus importants est lié à la gestion de la mémoire: tous les calculs nécessitent de grands mouvements de mémoire et il est toujours très bénéfique de limiter la place prise par chaque nombre manipulé.

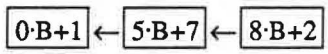
On a pu constater que les opérations sur les rationnels faisaient intervenir fréquemment le calcul d'un pgcd, qui lui-même implique un nombre important de divisions. Il est aussi patent que la division, parmi toutes les opérations arithmétiques élémentaires, présente une dissymétrie quelque peu grevante: dans le cas général (bien sûr le plus fréquent), il est nécessaire de pouvoir accéder aux deux derniers chiffres du diviseur avant de pouvoir calculer le facteur de normalisation et obtenir le chiffre le plus significatif du diviseur normalisé: pour cela, on est obligé de parcourir en entier la liste du diviseur avant de pouvoir faire quoi que ce soit.

Une deuxième critique que l'on peut apporter au modèle décrit jusqu'ici est que, finalement, les éléments matérialisant les chiffres des entiers ne servent jamais à autre chose qu'à stocker des valeurs dont la taille est la moitié de celle du type qui sert à les implanter. On a se rappelle les conséquences des problèmes de débordement dès que l'on essaie de prendre une base supérieure à celle adoptée en IV.A.

Mais, finalement, les opérations relatives aux chiffres en base B peuvent très bien se faire à l'aide de variables auxiliaires de type 'entier non signé' et les éléments matérialisant les chiffres peuvent eux-mêmes porter le contenu de *deux* chiffres: si l'on utilise un masque pour lire et écrire dans les $^{ns}/_2$ octets de poids faible et un second pour lire et écrire dans les $^{ns}/_2$ octets de poids fort, on sait que deux chiffres en base B peuvent facilement coexister dans le même entier non signé! De fait, si c et d sont des chiffres en base B, la quantité cB+d est au plus égale à B(B-1)+(B-1) = (B-1)(B+1) = B^2-1: B a justement été choisi en (IV-1) de sorte que ce nombre soit représentable par un entier non signé.

On peut alors conjuguer ces quelques remarques pour proposer un modèle beaucoup plus compact de représentation des entiers, qui a la particularité d'optimiser la place mémoire requise et de remettre sur un plan d'égalité, ou presque, les quatre opérations élémentaires.

Dans la suite, tous les entiers considérés sont supposés non nuls, (0 admet une représentation triviale de liste vide) et la longueur des listes représentant les nombres supposée explicitement connue. Les chiffres sont rangés par 'couples' dans les éléments utilisés jusqu'ici, en respectant la logique d'écriture b-adique habituelle; par exemple, le nombre $(1,5,7,8,2)_B$ est représenté sous la forme



Mais, au lieu d'être chaînés comme précédemment, sous forme de liste linéaire simple, les entiers sont chaînés 'en anneau', de façon à ce que le dernier chiffre ait comme suivant le premier chiffre du nombre.

Enfin, on ne repère pas les nombres par leur début de liste, mais par leur dernier élément: pour l'exemple choisi, on trouverait la configuration de la figure IV.6.(a), et pour un entier d'un ou deux chiffres celle de la figure IV.6.(b), où u est éventuellement nul.

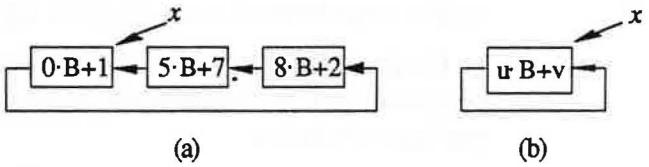


Figure IV.6. Modèle de représentation en anneau des entiers.

Comme l'exemple choisi le montre, le dernier 'couple' d'un entier ne contient deux chiffres que s'il possède un nombre pair de chiffres en base B. Si le nombre de chiffres est impair et supérieur ou égal à 3, on convient de répéter l'avant-dernier chiffre en 'dernière position'. Ainsi, dans l'exemple IV.6.(a), le dernier couple aura en fait la forme 5·B+1.

Ceci ne pose pas grand problème puisqu'aucune place supplémentaire n'est nécessaire et que la connaissance de la parité du nombre représentant la longueur de la liste relative à chaque nombre permet de lever toute ambiguïté. (Incidemment, on teste la parité de la taille t d'un entier en effectuant un test 'bit-à-bit' du type $(t \wedge 1)$, où $\wedge$ est l'opérateur unaire 'et'; ce test provoque la réponse VRAI si et seulement si le bit de poids le plus faible de t est égal à 1, c'est-à-dire si t est un nombre impair).

Dans le nouveau modèle, on accède *directement* aux deux derniers chiffres du nombre x, et à son premier chiffre en utilisant une fois la primitive 'suivant': d'une manière remarquable, l'instruction suivant(x) permet d'accéder au premier chiffre de l'entier non nul x, qu'il ait un, deux ou plusieurs chiffres!

On voit que, moyennant masquages et démasquages (nettement préférables aux opérations $(c_1,c_2) \leftrightarrow c_1 \cdot B+c_2$ pour les mêmes résultats) et une attention plus soutenue concernant le chaînage des nombres, le nouveau modèle donne des algorithmes à peine moins efficaces pour les trois premières opérations arithmétiques, une implantation évidemment plus efficace de la division – donc du pgcd – et permet surtout une diminution de moitié de l'espace mémoire utilisé pour la représentation des entiers, ce qui se traduit nécessairement par une réduction substantielle du temps d'exécution lié à la gestion de la mémoire!

La perte de place occasionnée par le stockage obligatoire de la taille des entiers est largement compensée dans le nouveau modèle: chaque rationnel s'accompagne de quatre octets supplémentaires pour la taille du dénominateur, du numérateur et de son signe, mais le nombre de ses 'chiffres' est divisé par deux: dès que le numérateur et le dénominateur ont deux chiffres, ce modèle est plus économique que le premier.

Notons enfin que ce modèle permet, statistiquement, une comparaison plus rapide des entiers, bien que toujours proportionnelle au nombre de leurs chiffres dans le pire des cas: lorsque les deux entiers de même signe ont le même nombre de chiffre et possèdent des chiffres de dernier rang (plus hautes puissances) différents, on peut les comparer en temps constant grâce à l'accès direct aux deux derniers chiffres proposé dans ce modèle. Il n'est plus nécessaire de parcourir leurs deux listes de bout en bout *que* s'ils ont même signe, même nombre de chiffres et mêmes chiffres les plus significatifs.

On peut considérer que ce modèle procure les avantages de la représentation fixe par tableaux en adjoignant la souplesse de représentation par listes chaînées.

Remarque: Pour le lecteur peu habitué aux effets néfastes de l'arithmétique rationnelle, il sera intéressant de noter qu'une application comme la facétisation de Delaunay d'un domaine d'un kilomètre carré de surface contenant une centaine de segments ou points peut donner lieu, avant de recourir à des mesures draconiennes comme celles de ce chapitre, à la création de *plusieurs millions* de chiffres en base 65536! Les méthodes suggérées sont susceptibles de diviser par cinq ou dix le temps passé par le système a trouver de la place mémoire ou à la libérer.

IV.B. SPECIFICITES D'UNE SOLUTION ADAPTATIVE

De même que seule une solution algébrique peut résoudre des problèmes d'essence algébrique, toute solution rationnelle impose des limites naturelles et structurelles.

Il est plus que raisonnable, dans ces circonstances, de ne pas prétendre concevoir une 'panacée', solution miracle qui pourrait résoudre tout problème de géométrie algorithmique sur tout domaine, mais plutôt d'essayer de qualifier les réponses qu'il serait souhaitable d'obtenir par le biais d'une solution adaptative, quitte à devoir adapter telle ou telle méthode pour un autre domaine d'application. Le terme adaptatif sera donc compris comme 'capable d'adaptation en fonction des données et des opérations souhaitées'.

Plutôt que de faire la description exhaustive de toutes les situations que ne pourra pas traiter la solution adaptative proposée, il sera plus constructif d'en dégager les spécificités.

IV.B.1 Limites naturelles

Etant donné le caractère fini du nombre de pseudo-réels représentables et de la précision relative à ces nombres, il est obligatoire de limiter le domaine de travail d'une application de géométrie algorithmique, *même* si une arithmétique exacte est disponible: l'espace d'adressage, la mémoire, etc... de la machine sont, de toute façon, finis.

Afin de pouvoir profiter des techniques mixtes d'évaluation (précision finie et arithmétique exacte), il est essentiel que ces données aient une dynamique telle qu'il soit possible d'approcher chaque nombre de l'intervalle choisi par un pseudo-réel. Si tel n'est plus le cas, il n'y a pas d'autre issue que de recourir à une solution exacte.

Pour cette raison, la solution adaptative admet des données qui, pour simplifier la présentation, seront supposées appartenir à l'intervalle réel [0, M], (M>0). Un nombre de [0, M] sera dit *modèle*. D'une certaine façon, la valeur de M dépend d'un seuil de fiabilité attendu, comme cela sera expliqué plus loin. Ce qui sera dit pourra très facilement être étendu à des nombres de l'intervalle [-N, M] (N, M > 0).

IV.B.2 Limites structurelles

Il faut maintenant limiter la portée de la solution adaptative par un choix de structures, de modèles et d'opérations: cette limitation n'est pas une surprise, car on sait par avance que tous les problèmes d'essence irrationnelle doivent être éliminés. Les restrictions proposées ici ont pour unique but de délimiter un support tangible de travail, seulement borné en fait par l'essence des opérations visées.

Une fois notre impuissance avouée devant les problèmes de nature algébrique pure, irrationnelle et transcendante (!), tout problème d'un autre type peut être modélisé dans une solution adaptative: le seul but de cette sous-section est de dégager un 'ensemble d'accueil' aussi large et adaptable que possible.

Structure de graphe planaire

La solution adaptative est censée donner en bout de chaîne la représentation par segments d'un graphe planaire (RSGP[2]) dont les éléments peuvent être décrits par la 'syntaxe' suivante:

- Un point du plan constitue un sommet du graphe, une extrémité d'arête ou un sommet de face.
- Une arête est définie par la donnée de deux points distincts du plan.
- Une chaîne est définie par un ensemble d'au moins trois points non tous confondus; (ainsi, un segment est un cas particulier de chaîne).
  Si le premier et le dernier points sont distincts, la chaîne est ouverte, sinon elle est fermée et porte le nom de cycle.
  Dans un cycle, tout sommet a exactement deux voisins, dans une chaîne ouverte ceci est vrai sauf pour les deux extrémités qui n'ont qu'un seul voisin.
- Une face est définie par un cycle dont les arêtes forment la frontière[3] et délimitent l'intérieur. 'polygone simple sans trous' sera un synonyme de face souvent employé .
- La rsgp est composé,e du point de vue géométrique, de sommets et d'arêtes et du point de vue topologique, de sommets, de segments, de chaînes et de faces.

La rsgp est supposée plongée dans un plan euclidien, ce qui permet de bénéficier de la métrique euclidienne et de la notion d'angle géométrique ou orienté, l'orientation canonique des angles étant l'orientation trigonométrique (ou directe) habituelle.

Les points, arêtes et polygones contenus dans les données appartiennent tous au même plan et sont susceptibles d'occuper toutes les positions relatives imaginables.

L'ensemble des données portera par la suite le nom de *pré-graphe*, car il constitue une sorte de version préliminaire d'un graphe planaire, mais ne vérifie pas, dans son état brut, certaines des propriétés d'une telle structure.

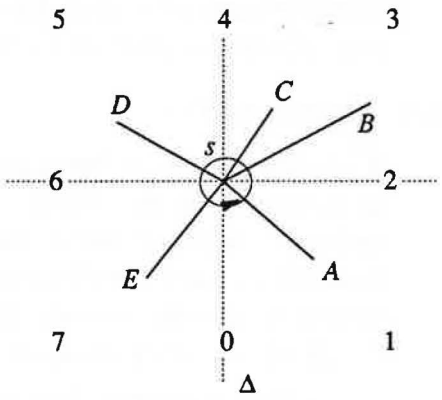


Figure IV.7. Division du plan autour de *s*.

[2] Le terme anglais consacré étant *planar straight line graph* (PSLG), cf. [P/S 85].
[3] Le lecteur est renvoyé aux ouvrages de Géométrie Différentielle de référence, comme *Differential Geometry*, J.J. Stoker, Wiley Inter-Science, 1969.

IV.B.3 Opérations

Opérations élémentaires

On considère comme opérations élémentaires de la machine les quatre opérations arithmétiques, les fonctions mathématiques élémentaires (racine carrée, exponentiation, logarithme, sinus, cosinus, tangente etc) et les opérations logiques ou bit à bit.

Primitives

Toute autre opération est appelée *primitive* et considérée comme une opération composite; la complexité d'une primitive est fonction de la taille *a priori* variable des paramètres sur lesquels elle opère.

Les primitives visées sur le pré-graphe ou le graphe planaire sont:

- la comparaison lexicographique de deux sommets ou arêtes,
- le classement d'arêtes de même origine suivant leur pente,
- la détection de l'intersection de deux arêtes du pré-graphe,
- le test d'appartenance d'un sommet à une arête ou à l'extension infinie du support de celle-ci,
- la détermination des coefficients paramétriques de l'équation du support d'une arête ou de sa médiatrice,
- l'évaluation de la distance de deux sommets, de la puissance d'un sommet à une arête, ou à son support,
- le test d'inclusion d'un sommet ou d'une arête dans une face,
- le test d'alignement de trois sommets,
- le test d'appartenance d'un sommet au disque circonscrit à trois sommets distincts non alignés,

Cette liste n'est pas limitative, dans le sens où l'on pourrait lui adjoindre d'autres éléments sans remettre en cause l'ensemble des caractéristiques qui vont être utilisées, mais l'ensemble obtenu est représentatif des opérations indispensables à l'analyse recherchée.

Analyse rapide des techniques relatives à ces primitives

Pour permettre un exposé compact des méthodes à venir, et surtout permettre de justifier certains des calculs qui y seront menés, il est important de donner quelques détails sur les primitives précédentes.

On sait déjà tout sur la première primitive. Pour illustrer la deuxième, on considère toutes les arêtes incidentes en un sommet s d'un graphe planaire (cf. Figure IV.7). En vertu de la planarité de ce graphe, si l'on considère que ces arêtes sont en fait des arcs orientés depuis s, il est impossible de trouver deux arcs non confondus, de même sens et de même pente. Si l'on appelle Δ le demi axe infini d'origine s et dirigé vers les y négatifs, il est possible de classer toutes les arêtes selon l'angle orienté qu'elles forment avec Δ, relativement à l'orientation naturelle trigonométrique.

En prenant s comme origine du plan et Δ comme origine des angles, on peut subdiviser le plan en quatre quadrants ouverts bordés par autant de demi-axes infinis, de sorte que tout point t différent de s se voie affecter un chiffre entre 0 et 7. Ce découpage, qui s'effectue naturellement en étudiant les signes des quantités $(x_t\text{-}x_s)$ et $(y_t\text{-}y_s)$, permet de limiter passablement les calculs relatifs au classement des arêtes.

Si $U(x_u,y_u)$ et $V(x_v,y_v)$ sont deux points différents de s pour lesquels les arêtes $[s, U]$ et $[s, V]$ reçoivent le même chiffre dans le processus précédent, on est certain, de par la planarité du graphe, que ces dernières sont soit confondues, soit situées dans le même quadrant. Les arêtes distinctes sont enfin classées en comparant leurs pentes

$$\frac{y_v-y_s}{x_v-x_s} \gtrless \frac{y_u-y_s}{x_u-x_s}.$$

Pour éviter les divisions, dont l'analyse de fiabilité est toujours délicate, on peut raisonner sur le signe des accroissements en x et en y selon le quadrant et tester

(IV-9) $$(y_v-y_s)\cdot(x_u-x_s) - (x_v-x_s)\cdot(y_u-y_s) \gtrless 0$$

Le chapitre II a déjà introduit les trois primitives suivantes et seule l'équation de la médiatrice d'un segment nécessite un peu d'explications. L'équation du support de [A,B] est

$$(x-x_A)(y_B-y_A)-(y-y_A)(x_B-x_A) = 0$$

Toute perpendiculaire à [A,B] a donc une équation de la forme

$$x(x_A-x_B)+y(y_B-y_A)+\varpi = 0.$$

et comme la médiatrice passe par le milieu de [A,B], on trouve

(IV-10) $$\varpi = {}^1/_2[(x_B^2-x_A^2)-(y_B^2-y_A^2)].$$

Le carré de la distance entre deux points A et B est donné par

(IV-11) $$d^2(A,B) = (x_B-x_A)^2+(y_B-y_A)^2$$

La puissance d'un point $M(x_M,y_M)$ par rapport à la droite d'équation $ux+vy+w=0$ est donnée par

(IV-12) $$p(M, \Delta_{(u,v,w)}) = ux_M+vy_M+w$$

On teste l'appartenance d'un sommet à une arête en vérifiant si ce point vérifie l'équation paramétrique relative au segment. On teste si un sommet est dans une face en évaluant sa puissance par rapport aux arêtes de la face, et l'on teste l'inclusion d'une arête dans une face en appliquant l'algorithme précédent aux deux extrémités de l'arête: toutes ces primitives font référence à des primitives précédentes.

La primitive testant l'alignement de trois sommets nécessite une réflexion plus sérieuse et joue un rôle fondamental en géométrie algorithmique. Si $\overrightarrow{BA}$ et $\overrightarrow{BC}$ sont deux vecteurs du plan, leur produit vectoriel est un vecteur normal à ce plan dont la norme est égale à l'aire du parallélogramme qu'ils définissent (voir la figure IV.8.(a)). Comme ce produit vectoriel s'exprime à l'aide du déterminant

(IV-13) $$D_{(A,B,C)} = \begin{vmatrix} 1 & 1 & 1 \\ x_A & x_B & x_C \\ y_A & y_B & y_C \end{vmatrix}$$

si la quantité $D_{(A,B,C)}$ est nulle, les trois points sont alignés, et sinon l'angle vectoriel $(\overrightarrow{BA}, \overrightarrow{BC})$ est de même signe qu'elle: ceci permet de savoir si la chaîne (A, B, C) définit un angle orienté supérieur à π (virage à gauche, déterminant positif, f-IV.8.(b)) ou inférieur (virage à droite, déterminant négatif, f-IV.8.(c)).

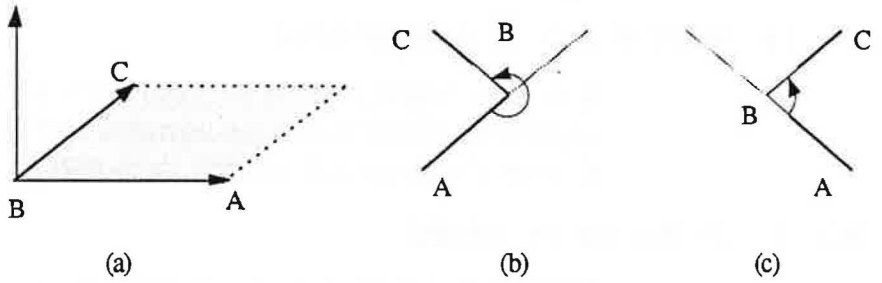


Figure IV.8. Angle orienté déterminé par trois points.

On préfère exprimer ce déterminant sous la forme présentant le moins de multiplications possible:

(IV-14) $$D_{(A,B,C)} = x_B(y_C-y_A) - x_C(y_B-y_A) + x_A(y_C-y_B)$$

La dernière primitive mentionnée est la plus 'gourmande'. La présentation originale et détaillée de cette solution se trouve dans [G/S/ 85].

A tout point du plan on peut associer un point unique de la paraboloïde de révolution d'équation $z = x^2+y^2$, en utilisant l'application

$$(x,y) \rightarrow (x, y, x^2+y^2)$$

On montre assez facilement que les quatre points A, B, C et M sont sur un même cercle du plan d'origine si et seulement si leurs images sur la paraboloïde sont coplanaires, i.e. si le déterminant

(IV-15) $$D_{(A,B,C,M)} = \begin{vmatrix} x_A & y_A & x_A^2+y_A^2 & 1 \\ x_B & y_B & x_B^2+y_B^2 & 1 \\ x_C & y_C & x_C^2+y_C^2 & 1 \\ x_M & y_M & x_M^2+y_M^2 & 1 \end{vmatrix}$$

est nul. D'autre part, on montre de la même façon que le point M est à l'extérieur du disque fermé circonscrit aux trois points A, B et C distincts et non alignés si et seulement si la quantité $D_{(A,B,C,M)}$ est négative. On voit que le déterminant (IV-15) implique des produits du type $x_A y_B(x_C^2+y_C^2)$, de degré quatre.

On verra que cette approche est très pénalisante dans le cas d'une arithmétique mixte et comment la remplacer par une technique mieux adaptée.

Opération globale

C'est une des tâches de l'application de transformer le tas d'informations livrées 'en vrac' dans les données en une rsgp et d'adjoindre à celle-ci une topologie cohérente.

Le deuxième travail consiste à organiser ce graphe topologique en une hiérarchie globale: tel point ou segment dans telle face, telle face dans telle autre et voisine de telles autres. Enfin, l'application fournit la triangulation de Delaunay du graphe hiérarchisé.

Encore une fois, le but de la solution envisagée n'est pas restrictif et son essence veut qu'elle s'adapte à la majeure partie des problèmes de la géométrie algorithmique. Toute adaptation passe par une analyse analogue à celle qui va être conduite pour l'application choisie.

IV.C. LE MONDE DES NON-EPSILONS

Il est maintenant possible de présenter la solution adaptative. Comme auparavant, l'intersection de segments servira de toile de fond, mais la solution s'applique à l'ensemble de problèmes décrits précédemment.

IV.C.1 Réticence et mixité

Imaginons que l'on doive obtenir des résultats aussi fiables que possible et que l'on dispose pour cela de deux évaluateurs: une arithmétique à précision finie livrée avec la machine hôte et donc gratuite, et une arithmétique exacte fournie par une société de service, dans laquelle chaque évaluation s'accompagne d'une taxation assez importante. On aura une certaine *réticence* à effectuer les opérations en arithmétique exacte et l'on cherchera à utiliser au mieux les capacités de l'arithmétique à précision finie disponible. Le réflexe naturel va même consister à calculer d'abord en précision finie, puis, uniquement s'ils sont trop peu concluants et s'il n'existe pas d'alternative, on prolongera les résultats déjà obtenus par des calculs en arithmétique exacte.

Formalisons la constatation précédente sur un exemple simple. Soit ε un nombre choisi à l'avance; on considère, l'algorithme IV.vi de comparaison des deux quantités pseudo-réelles x_A et x_B:

```
Comparaison(x_A, x_B):
si (x_A ≥ x_B + ε)
   x_A est vraiment supérieur à x_B;
sinon
   si (x_A ≤ x_B - ε) x_A est vraiment inférieur à x_B;
sinon
   exact-Comparaison(x_A, x_B);
```

Algorithme VI-vi: Détermination de l'ordre entre x_A et x_B.

La dernière instruction est une convention d'écriture signifiant: 'les deux nombres doivent être comparés à l'aide d'un module similaire fondé sur une arithmétique exacte'.

Le choix d'epsilon doit être réfléchi, car on sait que la précision varie en fonction de la taille des deux pseudo-réels. Mais justement, si epsilon est choisi de sorte que les deux premières instructions soient vraies *pour tous les pseudo-réels acceptés*, la troisième ne risque pas d'être invalidée et il est clair qu'aucun risque ne pourrait être pris, à aucun moment, dans la décision imposée par cet algorithme.

On a vu au chapitre précédent (cf (III-4)) que, sur une machine '32 bits', la précision relative à des pseudo-réels de l'ordre de 10^9 était d'environ 10^{-7} et qu'elle descendait à environ 10^{-4} pour des pseudo-réels de l'ordre de 10^{12}. Si l'on prend epsilon bien au-dessus de 10^{-4} (par exemple 10^{-1}), la validité de l'algorithme IV.vi ne fait aucun doute pour tout couple de données dont la taille va jusqu'à 10^{12}.

Tour ceci revient à dire que si l'on connaît la dynamique des données considérées dans une solution et si celle-ci obéit à certaines contraintes, on peut déterminer un seuil positif au-delà duquel les décisions de

l'algorithme IV.vi *peuvent* être prises sur la foi de la précision finie, en-dessous duquel elles *doivent* être prises sur celle de l'arithmétique exacte et en vertu duquel toute décision prise dans cet algorithme est identique à celle qui serait prise dans le cadre d'une arithmétique exacte.

Que gagne-t-on à utiliser ce type de méthodologie mixte? On se débarrasse complètement des problèmes de précision, en n'imposant pas un recours systématique à l'arithmétique exacte: celle-ci n'est appelée que lorsque l'arithmétique à précision finie ne peut seule suffire aux décisions. S'il est difficile de quantifier exactement le gain de temps théorique lié à cette technique, celle-ci permet, en pratique, de gagner plus de 50% du temps d'exécution de la solution exacte.

Le seuil de fiabilité choisi n'est plus 'aussi petit petit que *possible*', mais 'le plus petit permettant la validité de l'algorithme IV.vi *partout*'. Ce seuil sera effectivement assez petit, aussi petit que le permettra la dynamique des données imposées, mais sa taille devra surtout permettre de limiter un maximum le recours à l'arithmétique exacte.

Curieusement, la formulation de [G/S/S 89] se transpose ici, bien que quelque peu détournée: on ne cherche pas l'intervalle de validité d'un prédicat mais une limite supérieure de non-validité de celui-ci: en-dessous de cette limite, on ne peut plus lui faire confiance et l'on doit recourir au prédicat fondé sur l'arithmétique exacte correspondant.

Cette méthodologie présente un défaut, semblable à celui d'autres méthodes examinées au chapitre dernier: il faut analyser chaque primitive pour lui associer un epsilon. Il est éventuellement possible de déterminer un seuil universel, applicable à toutes les primitives, mais ceci ne peut être garanti qu'après une analyse fine du problème. L'existence d'un epsilon global permet évidemment d'alléger les algorithmes 'mixtes' en leur conférant une apparence uniforme.

Les notions décrites jusqu'ici inspirent la représentation *fonctionnelle* de la figure IV.9: le disque noir autour du pôle de décision A matérialise une zone à l'intérieur de laquelle toute décision doit être prise en référence à une arithmétique exacte, le disque blanc symbolise une zone dans laquelle ces mêmes décisions peuvent être prises indifféremment avec une arithmétique à précision finie ou infinie. Dans la troisième zone, en fait infinie, toute décision doit encore être prise par un autre biais: soit la dynamique des données n'est plus acceptable, soit la précision disponible introduit certaines incohérences, soit le modèle d'arithmétique exacte choisi s'avère inadéquat (problèmes d'essence irrationnelle, limitation sur la mémoire, etc). Les contraintes imposées aux données et aux types de primitives permettent d'affirmer que l'on n'atteindra *jamais* cette zone infinie de la figure IV.9.

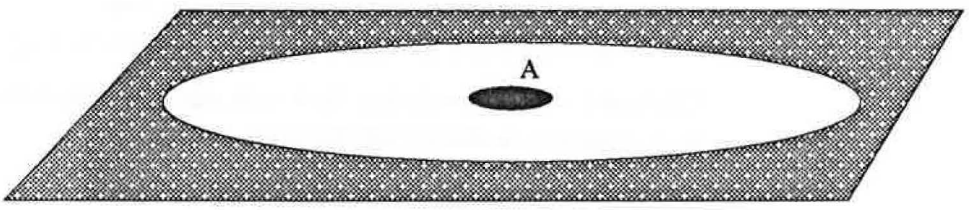


Figure IV.9. Domaines de validité de l'arithmétique 'mixte'.

IV.C.2 Calculs de fiabilité

Pour comprendre la généralisation de certaines des affirmations précédentes, il est utile d'étudier le comportement de certaines expressions dont dépendent les primitives retenues. Les techniques utilisées dans cette section sont simples, découlent des calculs d'erreur traditionnels [K II] et peuvent être comparées à celles de [G/S/S 89], sans oublier les divergences relatives aux présupposés et aux conclusions!

Dans les calculs qui suivent, les lettres a, b, ... symbolisent des nombres modèles. Pour simplifier et raccourcir la présentation, on ne cherchera que l'expression des bornes supérieures, mais celle des bornes inférieures est pratiquement identique.

Si $\perp$ est une opération élémentaire portant sur les pseudo-réels a et b représentés avec une mantisse de p places, on peut trouver un réel δ positif – qui est la précision de l'opération – tel que

(IV-16) $$a \perp b = (a \perp b)(1+\delta),\ \delta \le 2^{1-p} = \delta_0 \text{ [K II, p. 216]}$$

Cette formulation est un écho des constations du chapitre III, relatives à la précision. La formule (IV-16), adaptée à la somme, devient

$$(a \oplus b) - (a+b) = \delta\cdot(a+b) \le 2\delta M \le 2\cdot\delta_0 M = 2^{2-p}M$$

où M symbolise le plus grand nombre modèle accepté. Il est facile de voir que le seuil de fiabilité de l'addition peut être défini par la relation

$$\varepsilon_+ \ge 2^{2-p}M$$

En fait, ce que l'on souhaite pouvoir obtenir de manière formelle, c'est un algorithme comme IV.vi. Supposons que c soit un troisième nombre modèle, auquel est comparé (a+b). Dans la lignée de l'algorithme IV.vi, on souhaite connaître un seuil epsilon de fiabilité qui permette d'écrire, du point de vue de la borne supérieure, une instruction du type:

si (a+b>c+ε) action 1;

Le calcul de seuil précédent est insuffisant: il faut, en fait, évaluer le seuil relatif à l'expression (a+b-c). On procède comme précédemment, en considérant, pour simplifier les choses, qu'il n'y a que des sommes:

$$(a \oplus b \oplus c) = ((a+b)(1+\delta_1)+c)(1+\delta_2)$$

Cette fois-ci, on rencontre deux δ: un relatif à la somme (a+b) et le second relatif à la somme ((a+b)+c). Quoi qu'il en soit, ces deux δ sont obligatoirement positifs et inférieurs à δ_0; de plus, l'un est nécessairement supérieur ou égal à l'autre. Notons $\delta = \max(\delta_1, \delta_2)$, on peut récrire la relation précédente sous la forme

(IV-17) $$(a \oplus b \oplus c) \le ((a+b)(1+\delta)+c)(1+\delta)$$

$$\Leftrightarrow \quad (a \oplus b \oplus c) - (a+b+c) \le \delta[(a+b)(2+\delta) + c]$$

Comme les pseudo-réels a, b, c sont nécessairement inférieurs à M, on peut maintenant donner un majorant précis:

$$(a \oplus b \oplus c) - (a+b+c) \le M\delta(2\delta+3) \le M\delta_0(2\delta_0+3)$$

$$\Rightarrow \quad \varepsilon_+ \ge 2^{1-p}M(2^{2-p}+3)$$

Pour fixer les idées, si $M=2^{30}=1073741823$, $p=53$ (machine 32 bits), les calculs donnent $\varepsilon_+ \geq 2^{-32}(2^{-51}+3)$. On peut donc sans risque prendre ε_+ égal à $2^{-32}.(1+3)=2^{-30} \approx 10^{-9}$. Ainsi, l'instruction recherchée peut-elle être maintenant écrit sous la forme:

si (a+b-c > 10^{-9}) action 1; **sinon** *vérification rationnelle du test...*

Le choix de δ apporte une simplification dans les calculs, et il est important de le justifier: la taille des nombres modèles doit nécessairement être telle que le résultat des opérations intermédiaires ne créent jamais de dépassement de capacité; ceci permet de garder la même borne supérieure de précision. Quelle que soit l'opération, la précision qui l'accompagne ne peut être meilleure que 2^{1-p} et ceci est donc aussi vrai des deux δ_i précédents. Le fait de prendre le plus grand des deux majore le premier membre de l'équation (IV-17) et cette majoration va dans le sens recherché puisque epsilon doit encore *majorer* la différence (**a + b + c**) - (a+b+c): la dernière implication de ce genre de calculs, la seule à commencer par $\Rightarrow$, doit toujours être comprise comme "il suffit de prendre epsilon tel que...".

Principes de base

L'évaluation d'un seuil de fiabilité inclut une différence en plus des opérations propres à l'expression à seuiller quand le résultat doit être comparé à une valeur *non nulle*. Ceci étant, on commet un 'abus' de confiance pour permettre des explications plus courtes qui consiste à escamoter la partie relative à cette ultime différence dans les calculs qui vont suivre. Ceci rend les calculs plus compacts (on les dira 'simplifiés'); mais il est entendu que l'on doit, quand cela est nécessaire, faire intervenir la différence de test pour obtenir les seuils définitifs.

Le deuxième principe est *fondamental*: pour conserver une topologie parfaitement cohérente, il est nécessaire que toute action provoquant la création, la destruction ou la modification d'un élément du graphe (sommet, arête, face) soit impérativement exécutée avec la caution explicite de l'arithmétique exacte. Par exemple, si est découverte une solution normale dans l'intersection de deux segments, et ceci même en précision finie, les coordonnées du point d'intersection doivent, *elles*, être calculées explicitement avec l'arithmétique exacte.

Il reste maintenant à faire valoir quelques principes et remarques sur l'évaluation des seuils pour une primitive type.

Les calculs de comparaison de pentes, de distance et de puissance font intervenir des expressions de la forme (a+b)(c+d)+(e+f)(g+h). Chaque opération de cette expression donne naissance à un terme $(1+\delta_i)$, $\delta_i \leq \delta_0$:

(a+b)*(c+d)+(e+f)*(g+h) =

$[[(a+b)(1+\delta_1)][(c+d)(1+\delta_2)](1+\delta_3)+[(e+f)(1+\delta_4)][(g+h)(1+\delta_5)](1+\delta_6)](1+\delta_7)$

Si l'on pose $\delta = \max_{i \in [1,7]}(\delta_i)$, la quantité précédente peut être majorée par

$$(1+\delta)^4[(a+b)(c+d)+(e+f)(g+h)]$$

Puisque tous les pseudo-réels sont compris entre 0 et M, on peut encore écrire

$$(a+b)*(c+d)+(e+f)*(g+h) - (a+b)(c+d)+(e+f)(g+h) \leq$$
$$((1+\delta)^4-1)[(M+M)(M+M)+(M+M)(M+M)] = 8M^2\delta(4+6\delta+4\delta^2+\delta^3)$$
$$\Rightarrow \quad \varepsilon \geq 8M^2\delta_0(4+6\delta_0+4\delta_0^2+\delta_0^3)$$

On peut simplifier l'expression des seuils en utilisant le fait que δ_0 est en général très petit, rendant négligeables les termes δ^k, $k>1$: le seuil précédent peut être remplacé par $8M^2(4\delta_0+\lambda_0)$, avec une évaluation approchée de type $\lambda_0 \geq 2\delta_0^2(3+2\delta_0)$.

Sens et dégénérescences

Intervient ici un point fondamental, qui ramène au chapitre sur l'intersection de deux segments: on a vu qu'on pouvait obtenir un algorithme pour résoudre ce problème où les quantités sur lesquelles portaient les choix étaient, *sémantiquement*, équivalentes à des puissances.

Il est très important de remarquer qu'une solution adaptative nécessite une bonne connaissance de ce que représentent les valeurs sur lesquelles des seuils vont s'exercer: si une puissance a une valeur proche de zéro, c'est forcément que le point concerné est très proche du support du segment; la valeur de dégénérescence numérique est une transcription fidèle et proche de la dégénérescence géométrique.

Le fait que l'on compare des puissances donne une idée précise de l'influence des variations de ces quantités sur la géométrie du problème et réciproquement. Ce genre d'information est capital dans l'élaboration de primitives mixtes: plus on s'éloigne de la modélisation géométrique d'un problème et moins ses cas de dégénérescence sont facilement interprétables en termes de seuils.

Seuils et nombres positifs

Il peut s'avérer important que les nombres modèles soient effectivement positifs. Pour illustrer ceci, considérons l'exemple (IV-14) du test d'alignement, qui inclut une expression du type a(b-c)-d(e-f)+g(h-i). Les calculs de fiabilité simplifiés donnent:

$$a(b-c)-d(e-f)+g(h-i) - [a(b-c)-d(e-f)+g(h-i)] \leq$$
$$((1+\delta)^3-1)[a(b-c)-d(e-f)+g(h-i)] =$$
$$\delta(4+6\delta+4\delta^2+\delta^3)[a(b-c)-d(e-f)+g(h-i)]$$

On voit facilement que si les soustractions étaient remplacées par des sommes, la quantité entre crochets devrait être majorée par $6M^2$. En revanche, les différences permettent de trouver le meilleur majorant de valeur $3M^2$, si les nombres modèles sont bien positifs ou nuls. Une remarque analogue s'applique aux minorants.

Modularité et résultats intermédiaires

Comme dernier exemple, considérons le seuil de fiabilité simplifié relatif au calcul du déterminant (IV-15); celui-ci doit vérifier la relation

$$\varepsilon \geq 12M^4[(1+\delta_0)^7-1]$$

Ce dernier résultat laisse entrevoir qu'une telle primitive n'est peut-être pas si directement utilisable que son écriture le laisse supposer: dans le cas particulier considéré, il sera plus efficace de se ramener au calcul du centre et du rayon du disque puis de tester si la distance du point M au centre du disque est inférieure ou non au rayon; le centre du disque aura, de nouveau, une représentation approchée suffisamment voisine de la position exacte pour que l'on puisse envisager une technique plus simple de calcul de distance, dont le seuil de fiabilité sera bien plus bas que le seuil précédent!

Globalement, on a n'a pas abaissé le degré du problème (le coefficient ϖ de (IV-10) rappelle les carrés présents dans (IV-15)), mais on a réussi, en passant par un point auxiliaire (le centre du disque circonscrit aux trois points), à s'assurer que les calculs intermédiaires ne puissent diverger trop, localement, de la réalité du problème.

Ceci est une constante dans l'approche mixte: il est essentiel de subdiviser les calculs longs en étapes dans lesquelles les résultats intermédiaires permettent des 'recalages' des valeurs approchées sur les valeurs exactes, de proche en proche.

Seuil global ou seuil local?

Toutes les primitives retenues combinent des opérations semblables à celles examinées plus haut, et il suffit de considérer chacune pour en déduire un seuil de fiabilité. Une fois ces calculs effectués, on se trouve confronté à un choix: vaut-il mieux prendre le maximum de tous les seuils choisis et l'appliquer à chaque primitive rencontrée ou bien laisser chaque primitive utiliser son propre seuil, tel qu'il a été calculé?

La réponse dépend en fait de la 'dynamique' des seuils obtenus. Si les opérations de ces primitives incluent toutes ou presque le même nombre d'opérations élémentaires, il y a de grandes chances qu'une uniformisation soit possible.

A l'inverse, supposons qu'il y ait un grand nombre d'opérations du genre 'comparer deux pseudo-réels' dont on a vu que le seuil de fiabilité est très petit, et une primitive comme celle permettant de savoir si un point est inclus dans le disque circonscrit à trois autres points du même plan; le rapport des deux seuils est de l'ordre de

$$\frac{7\cdot 12M^4\delta_0}{3\cdot M\delta_0} = 28M^3$$

et la disproportion est telle qu'on aura intérêt à scinder les deux types d'action de façon à ne pas imposer à l'opération la moins dispendieuse les coûts prohibitifs de la seconde.

Il faut, autant que faire se peut, peser les gains et les pertes statistiques que l'on encourt à agir d'une façon ou de l'autre: l'uniformisation des epsilons accroît certainement les recours à l'arithmétique exacte, mais permet une écriture bien plus compacte des primitives.

On trouvera à la fin de ce chapitre une table de quelques valeurs numériques de seuils d'opération en fonction de la dynamique des données.

IV.C.3 Structures mixtes pour la réticence

Il faut maintenant se donner un environnement de travail pour permettre la conception de procédures réticentes. Le principe général est qu'il faut prévoir une duplication des informations dans les structures afin de permettre le prolongement des résultats à précision finie par des calculs en précision arbitraire.

La construction du canevas de structures qui vont modéliser le problème est fondamentale: c'est à ce stade que nombre d'opérations vont être effectuées dont le résultat devra être réutilisable par la suite. En d'autres termes, étant donnée la pénalisation accompagnant chaque calcul en arithmétique exacte, tout calcul effectué une fois doit pouvoir être réutilisé au cours des opérations à venir, si l'on peut prévoir à l'avance qu'une telle opportunité se présentera.

Autant la comparaison de deux valeurs pseudo-réelles peut être effectuée un nombre quelconque de fois sans que l'on s'en émeuve (une fois la complexité théorique d'un algorithme précisée), autant la comparaison des rationnels représentant ces deux valeurs doit être répétée le moins souvent possible, car sinon, l'exécution de l'algorithme sous-jacent risque d'être dominée par celle des routines de l'arithmétique exacte.

Générations

En général, les données sont considérées comme si elles s'accompagnaient d'une précision maximale: quels que soient les algorithmes de codage binaire, si l'on convient d'une norme d'écriture des nombres initiaux, les égalités entre données se traduisent par des égalités entre représentants en machine de celles-ci.

Si les données sont des entiers modèles, la situation est la plus simple qui soit; si ce sont des décimaux, on suppose que leur écriture ne dépasse pas la précision de la machine, ou sinon une troncature automatique est indispensable. De toute façon, c'est au programme de lecture de s'arranger pour respecter les cohérences en amont de son exercice.

Les données initiales sont dites de *génération zéro*, dans le sens où l'on peut considérer que ce sont des quantités exactes du problème. Au vu des remarques précédentes, un test d'égalité portant sur deux nombres de génération zéro donne toujours un résultat vrai si les données initiales sous-jacentes étaient véritablement égales.

Ceci peut être important pour la construction des structures de données chargées de mettre de l'ordre dans le pré-graphe: les opérations qui les concernent peuvent éventuellement s'accompagner de seuils de fiabilité beaucoup plus faibles et se comporter, à la limite, comme des opérations naturelles de la machine sans grand risque.

Tout autre nombre ou résultat de calcul dans une solution adaptative est de génération strictement supérieure: si un tel nombre est opérande, l'opération considérée s'accompagnera d'une précision seuillée par une technique de la sous-section IV.C.2.

Réflexes et valeurs par défaut

On peut donc imaginer un modèle mixte très simple, dans lequel les fonctions chargées de calcul auront une certaine autonomie et une liberté d'action quelque peu comparable à celle des méthodes d'un langage à objets.

Prenons l'exemple de la structure chargée de matérialiser un point: elle contient naturellement deux champs (x, y) pseudo-réels pour représenter l'abscisse et l'ordonnée de la projection du point dans le plan de référence, muni d'un repère cartésien fixe. Elle contient aussi deux champs (rx, ry) rationnels qui permettront, si besoin est, de connaître la valeur exacte des coordonnées du point. *A priori*, si le point est de génération zéro, il n'est pas indispensable de connaître ces valeurs exactes et l'on peut mettre les champs correspondants à NIL: ceci sera un signal explicite d'appartenance à la génération zéro.

Toute procédure ou fonction chargée d'effectuer un calcul sur les coordonnées d'un point trouve directement ces valeurs dans les champs x et y de la structure qu'elle reçoit en paramètre. Si les calculs peuvent être effectués sans recours à l'arithmétique rationnelle, la routine rend la main sans autre forme de procès. En revanche, s'il devient nécessaire de passer à une évaluation exacte, la routine doit accéder aux champs rx et ry. Si ceux-ci sont vides, elle doit envoyer d'elle-même les valeurs des champs x et y aux procédures de la bibliothèque rationnelle pour remplir les champs rx et ry du point *une fois pour toutes*: on retrouve-là l'idée de réflexe et de valeur par défaut dans un langage à objets.

Ce genre de technique permet très naturellement de minimiser l'utilisation des librairies exactes, puisque les champs de ce type ne sont remplis qu'en cas de besoin. On va voir qu'une autre amélioration très sensible est encore possible.

Les pièges de l'égalité!

Une fois les données 'décodées', il ne semble pas très sain de continuer à considérer qu'il y a, d'un côté, les 'bons' nombres (de génération 0) et, de l'autre, les 'mauvais': les routines doivent toutes se comporter de manière cohérente par rapport à la précision et deux valeurs 'très proches' ne sont *réellement* confondues *que* si l'arithmétique rationnelle l'a reconnu, ou bien s'il a été prouvé, par un raisonnement analogue au précédent, que les deux valeurs provenaient de *la* même donnée initiale.

Il est assez évident que la méthode préconisée dans ce chapitre, si elle est mal employée, peut provoquer une certaine inflation de calculs. Pour illustrer ceci, supposons que l'on ait recours à un algorithme de balayage du plan.

Le principe d'un tel algorithme est de parcourir les données selon l'ordre lexicographique, de la plus petite à la plus grande. Chaque fois qu'une nouvelle abscisse est rencontrée, on insère tous les points de même abscisse dans la structure de balayage: ceci signifie implicitement que l'on doit à chaque étape exécuter des instructions du type 'tant que l'abscisse courante est égale à x_0' ou 'pour tous les points dont l'abscisse est inférieure ou égale à x_0'.

Mais justement, ce genre d'opération est devenu complexe, du fait de l'introduction d'une certaine réticence dans les algorithmes. Typiquement, quand il rencontre un point à la verticale de x_0, un algorithme de cette trempe dit "Tiens! Cet x semble trop proche de x_0: bien que l'idée me répugne, je suis forcé de demander à l'arithmétique rationnelle de statuer".

Si l'on n'y prend garde, la majeure partie du temps va être passée à vérifier des choses dont on ne se préoccupait pas trop, auparavant. Sous prétexte de ne *jamais* passer à côté de situations trop proches pour être discernées par l'arithmétique à précision finie, on finit par empêcher la reconnaissance de toutes les situations qui, auparavant, étaient trivialement identiques: une valeur paramètre comme le x_0 précédent n'a pas la même essence structurelle que les abscisses des points considérés dans l'algorithme!

Comme c'est souvent le cas dans ce genre de situation, le salut vient de la structure des problèmes: il va falloir recourir à une construction beaucoup plus solide que d'habitude et c'est cet échafaudage qui va faire entrevoir l'étendue de ce que l'on à gagner dans l'emploi d'une arithmétique réticente.

A la lecture, il semble naturel, vues les applications attendues, d'organiser les données de manière à ce qu'elles puissent être facilement retrouvées, supprimées, dupliquées, manipulées... On va supposer que les points du pré-graphe sont organisés en arbre binaire; que cet arbre soit équilibré ou non sera de peu d'importance. Lorsqu'il s'agit de l'organisation de points dans le plan, il est avisé de s'assurer que l'on commet le moins de duplications possible. En particulier, si l'on considère deux arêtes d'un graphe ayant en commun un sommet, il est naturel de considérer que les points sont des structures autonomes sur lesquelles pointent les extrémités d'arêtes. D'une certaine façon, on peut imaginer que l'on plaque sur l'arbre binaire des points une structure sous-jacente qui reflète la structure du graphe.

Sur la figure IV.10, l'arbre binaire contient plusieurs points dont A, B et C et les traits pointillés qui relient ces trois derniers symbolisent le fait que le graphe considéré possède les deux arêtes [A,B] et [B,C] partageant le sommet B. Cette sous-structure n'est pas explicite, mais elle est implicite dans le fait que les arêtes [A,B] et [C,D] sont représentées par des structures dont les champs 'origine' et 'fin' pointent, respectivement, sur les adresses de l'arbre relatives à A, B, B et C: on peut voir dans l'arbre une matérialisation de l'ordre lexicographique qui accompagne les deux arêtes.

La constitution d'une telle structure auto-référante nécessite qu'à un moment donné on instaure un ordre total sur tous les constituants. Ceci revient à dire non seulement que les points sont ordonnés, mais les champs 'origine' et 'fin' des arêtes et les arêtes elles-mêmes aussi. Ainsi, beaucoup d'effort est placé dans l'instauration de tous ces ordres – qu'il s'agisse d'une arithmétique exacte, à précision finie ou mixte. Un principe crucial est de profiter, autant que faire se peut, de cette

structuration, afin de pouvoir l'utiliser explicitement dans d'éventuels calculs ou tests ultérieurs.

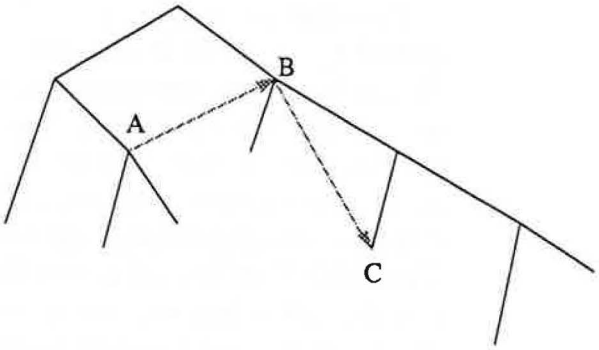


Figure IV.10. Arbre binaire et structure sous-jacente.

Classes d'équivalences et saules

L'arbre binaire contient une information qui sera exploitable une fois le graphe rendu planaire: chaque nœud de l'arbre porte un numéro d'ordre entier (son 'indice' dans un parcours infixé de l'arbre) et la comparaison de deux numéros permettra de dire sans équivoque la position relative de deux points quelconques du plan représentés par deux nœuds de l'arbre.

Mais, il faut encore faire un tout petit effort: afin de pouvoir, plus tard, reconnaître que deux points ont *exactement* la même abscisse sans recourir à une arithmétique exacte une seconde fois, il est nécessaire, en plus du numéro de nœud, de disposer d'un champ supplémentaire de 'filiation en x'. Chaque point nouvellement inséré dans l'arbre provoque une descente dans celui-ci jusqu'à l'apparition de l'une des deux éventualités suivantes: le point est vraiment nouveau (aucun autre point n'a le même couple de coordonnées) ou le point est déjà dans l'arbre.

Implicitement, ceci signifie qu'à chaque nœud ϖ visité, on compare à l'aide de primitives mixtes les coordonnées x et y du point à insérer et celles x_ϖ et y_ϖ relatives au point p_ϖ attaché à ce nœud. Si le point à insérer est à gauche lexicographiquement de p_ϖ, on descend vers le fils gauche du nœud courant, s'il est à droite, on descend vers son fils droit, et s'il est confondu, on le signale à la procédure appelante.

Il faut préciser ici qu'il existe deux types d'insertion dans l'arbre binaire: on a déjà rencontré l'insertion de points initiaux qui peut être gérée par l'intermédiaire de procédures très spécifiques en fonction de la nature privilégiée des points originels. Le deuxième type d'insertion provient notamment des points découverts par le module d'intersection: ceux-ci sont de génération supérieure ou égale à zéro, et les insertions doivent être effectuées en arithmétique mixte.

Quoi qu'il en soit, les tests effectués pour placer le point dans l'arbre binaire sont valides au sens de la précision arbitraire: il se peut que l'insertion ait été conclue après certains calculs en arithmétique exacte ou non, mais la position est exacte et si le nouveau point a la même abscisse (*réelle*) x_0 qu'un point déjà inséré, quelle que soit la méthode de vérification de cette affirmation, on peut catégoriquement placer les

deux points dans la même classe d'équivalence Cx_0 de l'arbre binaire, constituée de tous les points ayant l'abscisse x_0.

Pour traduire ceci, la structure 'point' contient un champ spécifique, appelé *x_classe*. Si le nouveau point inséré q possède une abscisse différente de toute autre déjà rencontrée, on convient de mettre l'adresse $@_q$ du nœud qui portera q dans l'arbre, comme valeur du champ *x_classe* pour la structure relative à q. Ceci traduit naturellement le fait que q, pour l'instant, est le seul élément de la classe d'équivalence Cq_x et que $@_q$ désigne le nœud représentant le premier (chronologiquement) d'une série de points qui auront (éventuellement) la même abscisse que q: dans l'arbre binaire, tous les nœuds matérialisant un élément quelconque de la classe Cq_x aura l'adresse $@_q$ dans le champ *x_classe*.

Si q se trouve avoir la même abscisse que le point p attaché au nœud couramment visité dans la descente, le champ *x_classe* de ce nœud contient, comme il a été dit précédemment, une adresse $@_{p_j}$ que l'on reporte dans le champ *x_classe* de la structure relative à q afin de perpétuer la propriété de transitivité de la classe d'équivalence. Ainsi, quelle que soit la suite $p_1, p_2, \ldots, p_n$ de points de même abscisse, il existe un point p_j parmi ceux-ci (vraisemblablement le premier à avoir été inséré dans l'arbre) tel que tous les champs *x_classe* des structures relatives aux éléments de cette suite contiennent l'adresse du nœud d'attache de p_j dans l'arbre binaire: ce point constitue le *représentant canonique* de la classe d'équivalence, et le nœud qui lui est associé dans l'arbre, est appelé le *nœud canonique* de la classe.

L'ordre dans lequel ces points sont rentrés ne change en rien les notions précédentes: par convention, on sait qu'on insère dans la direction du fils gauche tous les points ayant une abscisse égale à celles correspondant au nœud courant et une ordonnée inférieure, et dans la direction du fils droit ceux de même abscisse et d'ordonnée supérieure. Si l'arbre global n'est pas rééquilibré à chaque insertion, tout point ayant la même abscisse que p devra nécessairement être inséré dans un sous-arbre du nœud auquel p est attaché.

Sur la figure IV.11, les points ont été insérés dans l'ordre alphabétique et l'on rencontre quatre classes d'équivalence pour le x-ordre. L'arbre est principalement un arbre binaire ordonné lexicographiquement et les liens 'verticaux' reliant les représentants d'une même classe d'équivalence sont représentés en pointillés: le nom de *saule* vient tout de suite à l'idée pour désigner une telle structure.

Dans le cas d'un équilibrage dynamique de l'arbre, les choses se prêtent moins à une description fonctionnelle simple comme la précédente, mais, la relation 'q a le même champ *x_classe* que p' définissant une classe d'équivalence sur l'arbre, tout point ayant une abscisse déjà rencontrée est nécessairement comparé à un représentant de cette classe, dont le champ *x_classe* réfère directement au représentant canonique. Cette technique n'influence en rien l'équilibrage des arbres, et ne demande qu'une place proportionnelle au nombre de nœuds de l'arbre.

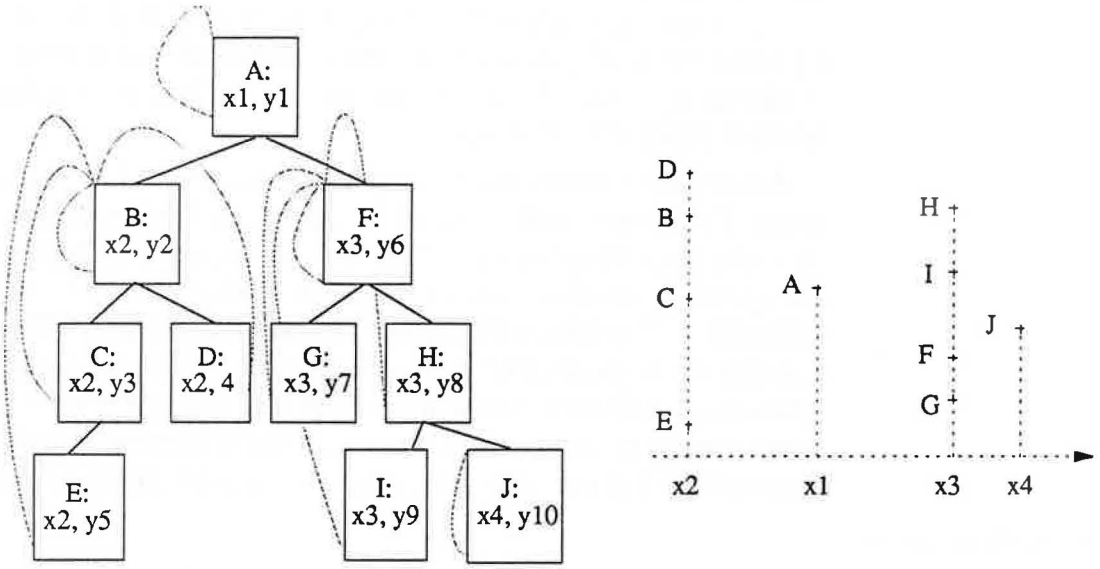


Figure IV.11. Arbre binaire en forme de saule.

Si $(@_p, n_p, @_{xp\_classe})$ est le triplet unique donnant l'adresse du nœud d'attache de p, son numéro d'ordre et l'adresse de son *x_classe* canonique dans l'arbre, on peut affirmer, une fois l'arbre constitué et le graphe rendu planaire, p et q étant deux points représentés dans l'arbre:

$$\begin{array}{lcl} p = q & \Leftrightarrow & (@_p = @_q); \\ p \leq_L q & \Leftrightarrow & (n_p \leq n_q); \\ p_x = q_x & \Leftrightarrow & (@_{xp\_classe} = @_{xq\_classe}). \end{array}$$

Bien évidemment, ces tests sont remarquablement faciles à exécuter, mais ne sont possibles que si le graphe a été rendu planaire, puisqu'il faut attendre que toutes les intersections aient été découvertes pour pouvoir prétendre connaître le numéro d'ordre définitif de chaque nœud de l'arbre.

D'autre part, il est évident que l'on a aussi intérêt à effectuer le même genre d'opérations sur les ordonnées des points pour se donner les mêmes opportunités dans la direction orthogonale.

Simulation naturelle de balayage

Une fois le graphe rendu planaire, l'arbre binaire ordonné permet naturellement le parcours lexicographique infixé de tous les points, mais le champ *x_classe* autorise en plus une simulation très simple et efficace d'un algorithme de balayage: on a vu que ce champ permet naturellement de remplacer une instruction du type 'pour tous les points à la verticale du point courant' par 'pour tous les points dont le champ *x_classe* est égal à ...'.

Imaginons l'algorithme de balayage a son début: on lance un parcours infixé de l'arbre binaire et l'on débute par le point situé le plus à gauche lexicographiquement du domaine. Le champ *x_classe* fait référence à une adresse dans l'arbre, qui est celle du nœud canonique de la classe. On sait, d'autre part que le parcours infixé va lister, dans l'ordre lexicographique, tous les points dont le champ *x_classe* fait éventuellement référence à la même adresse: tant que tous les repré-

sentants de la classe d'équivalence du point initial n'auront pas été listés, on ne changera pas d'abscisse. Il suffit donc de remonter dans le dépilement récursif jusqu'à ce qu'une nouvelle adresse apparaisse dans le champ *x_classe* du point listé, ce qui indiquera le début d'une nouvelle classe d'équivalence.

A l'origine, l'introduction de la notion de classe d'équivalence est venue d'une complication liée à la réticence de l'arithmétique. En fait, on s'aperçoit vite qu'au lieu d'être une conséquence fâcheuse, c'est le fait qu'on ne pratique pas ce genre de méthode dans le cas d'une arithmétique à précision finie qui est pénalisant: au lieu d'effectuer des tests sur des réels, on peut se ramener à des tests sur des entiers ou des adresses en mémoire! Comme de toute façon le pré-graphe doit être transformé en graphe planaire, on ne fait 'qu'enregistrer' les minutes de cette opération afin d'en bénéficier de manière très puissante plus tard.

Performances

La bonne marche d'une arithmétique réticente dépend en grande partie de la capacité du programmeur à savoir renâcler devant la tâche! Malgré tout, on se doute bien que lorsque les décisions doivent être prises, en dernier recours, sur la seule foi de l'arithmétique exacte, on a en fait été forcé d'effectuer deux fois les mêmes séries de calcul: une fois en précision finie et une autre en précision arbitraire.

Cette réflexion amène à penser que l'on peut très bien construire des situations dans lesquelles *tous les calculs* devront être effectués deux fois, ce qui laisse présager un certain ralentissement par rapport à la solution exacte pure. Certes, on peut imaginer de tels cas!

On pourrait aussi envisager des situations où tous les calculs pourraient être faits uniquement sur la foi de l'arithmétique à précision finie, auquel cas la solution adaptative serait même inutile. Bien sûr...

Heureusement, il reste tous les autres cas, où, statistiquement, les techniques décrites dans ce chapitre permettent non seulement de toujours garantir une réponse exacte et cohérente, mais en plus avec un temps de réponse bien meilleure que celui d'une solution exacte. Aucun exemple de données n'a réussi à provoquer un comportement 'hystérique' du module de facétisation de Delaunay.

Il est vraisemblable que ce n'est pas un hasard: essayons de quantifier de manière grossière la probabilité de provoquer le déclenchement de calculs rationnels dans la recherche d'une réponse (oui/non) à la question: deux segments 'aléatoires' se coupent-ils?

Soit un segment s_1 de longueur l dans un domaine de largeur maximale M. La longueur de ce segment est nécessairement majorée par $l \cdot \sqrt{2}$. Soit ε le seuil relatif à la primitive considérée. Si s_2 est un deuxième segment, on sait que les calculs devront être menés en rationnel dès qu'une des extrémités du second segment est située dans la bande de largeur 2ε s'étendant sur la restriction du support du segment au domaine (Figure IV.12). La 'forme' de cette bande est un parallélogramme, un rectangle, un trapèze, ou un hexagone lorsque le support du segment est confondu avec une diagonale du domaine.

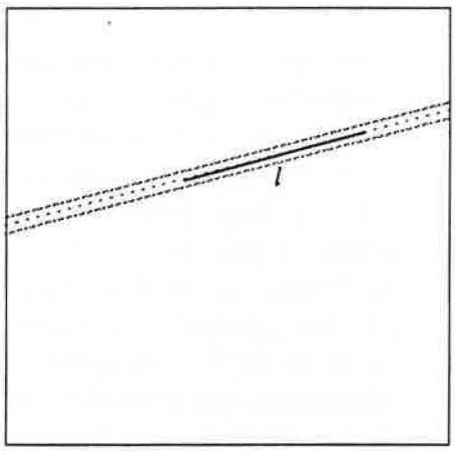


Figure IV.12. Bande de largeur 2ε autour d'un segment.

On peut montrer facilement que l'aire maximale de cette bande correspond à une position diagonale du segment, et est égale à $2^{1,5}\varepsilon M$. La probabilité pour qu'aucun calcul rationnel ne soit effectué est (hormis les cas de coïncidence d'extrémités, de probabilité nulle), selon le paradigme des puissances du chapitre III, celle pour qu'aucune puissance ne soit inférieure à epsilon. Ceci revient donc à calculer la probabilité pour que les deux extrémités du second segment soient toutes deux hors-bande: dans le cas d'une bande exactement rectangulaire, la probabilité pour qu'un sommet soit hors de la bande est minimale et vaut

$$\frac{M^2-2^{1,5}\varepsilon M}{M^2} = 1 - 2^{1,5}\frac{\varepsilon}{M}$$

La probabilité recherchée est donc le carré de celle-ci et, en définitive, la probabilité pour qu'aucun calcul rationnel ne soit effectué est *de l'ordre de*

$$p_r \approx 1 - [1 - 2^{1,5\varepsilon/M}]^2 \approx 2^{2,5.\varepsilon/M}$$

Comme le seuil epsilon trouvé pour l'évaluation d'une puissance est de l'ordre de $8M^2\delta_0 = 2^{-49}M^2$,

$$p_r \approx \frac{2^{2,5}\cdot 8\cdot M^2\delta_0}{M} = 2^{-46,5}M$$

Cette formule fait bien sentir le lien serré qui existe entre la taille des nombres, le seuil de fiabilité et la précision: si M dépasse $2^{46,5}$, le seuil est tellement important que l'on s'expose forcément à des calculs en rationnels. Si l'on prend M dans l'intervalle $[0, 2^{40}] \approx [0, 1\cdot 10^{12}]$, la probabilité obtenue est au maximum $2^{-6,5} \approx 0,01$. Les risques statistiques de provoquer des calculs rationnels dans la solution mixte et dans le cas de deux segments 'pris au hasard' sont faibles, de l'ordre de 1%, bien que non négligeables.

Il est malheureusement beaucoup plus délicat de quantifier cette probabilité dans le cas de pré-graphes issus de situations non aléatoires: il est très fréquent de vouloir placer une extrémité de segment contre un autre ou de faire mourir une arête très près à l'intérieur d'une face, sachant que celle-ci viendra effacer la partie destinée à disparaître...

Philosophie

Comment doit être écrite une primitive mixte et réticente? Elle doit être dotée d'un pouvoir de décision total quant aux modifications éventuelles des structures de données qui lui passent 'entre les mains': après avoir essayé par tous les moyens imaginables d'obtenir gain de cause avec la seule aide des structures majeures (arbres, graphe, hiérarchies...), puis avec celle de l'arithmétique à précision finie, elle se doit de limiter les calculs rationnels au strict minimum en remplissant les champs rationnels une seule fois, et seulement quand cela est inévitable, et en ne dupliquant *jamais* des calculs dont on peut prévoir qu'ils se répéteront: par exemple, une arête doit souvent être représentée sous forme de deux arcs 'jumeaux', de champs identiques, mais de sens contraire; un de ces arcs doit être considéré comme *original* et contenir des originaux des enregistrements rationnels (mais aussi pseudo-réels), tandis que l'autre est explicitement déclaré comme une *copie* du premier, ne contenant aucun champ propre (sauf un pointeur différent de NIL sur l'original): tout calcul relatif à ce dernier doit faire explicitement intervenir les champs du premier; si des calculs relatifs au second arc nécessitent l'évaluation de champs rationnels, ce sont ceux de l'original qui seront remplis.

Ceci exige également que les routines de libération des objets complexes doivent être aussi spécialisées et reconnaître les copies afin de ne libérer un objet que lorsqu'on est sûr que c'est un orignal *et que la copie n'est plus en service.*

En dernier lieu, on peut aussi remarquer que l'on obtiendra une souplesse totale si l'on fait calculer les seuils relatifs aux primitives une fois que les données sont lues: ceci permet effectivement de serrer au mieux les valeurs seuils et donc de limiter au maximum, pour chaque cas de figure, le nombre d'accès à la bibliothèque exacte.

3-D

Toutes les techniques décrites dans ce chapitre ont été conçues dans le cadre d'une application bi-dimensionnelle. Malgré tout, rien n'empêche de penser qu'elles pourraient être utilisées dans un espace à trois dimensions ou plus. Il est certain que les problèmes topologiques liés à l'application de seuils sont plus encombrants et délicats en trois qu'en deux dimensions, mais, fondamentalement, le principe de primitives seuillées peut se transposer indépendamment de la dimension de l'espace de travail. Certains travaux sont en cours à l'E.N.S.M.S.E. qui porteront en partie sur la généralisation de ces problèmes à un espace à trois dimensions, dans le cadre d'une arithmétique exacte.

ANNEXE

Afin de donner une idée de l'ordre de grandeur des seuils dans différents contextes, la table de cette annexe donne les valeurs de seuil pour trois tests couramment pratiqués dans une application de géométrie algorithmique.

Ces tests sont effectués dans les mêmes conditions sur chaque colonne. Les données sont supposées appartenir à l'intervalle [0, M], et M prend les différentes valeurs indiquées sur la première ligne de la table. On considère à chaque fois que l'unité principale est le millimètre.

Le premier test concerne la comparaison de la somme de deux réels à un troisième ($a+b \gtrless c$). La formule du seuil de fiabilité de ce test est

$$\delta_0 M(2\delta_0+3)$$

Le deuxième test compare la puissance d'un point M par rapport à une droite à zéro. Le seuil de fiabilité se calcule ici par la formule

$$2\delta_0 M^2(4+6\delta_0+4\delta_0^2+\delta_0^3)$$

Le troisième test permet de déterminer si trois points coplanaires A, B et C sont alignés ou non. Le seuil correspondant est donné par

$$3\delta_0 M^2(4+6\delta_0+4\delta_0^2+\delta_0^3).$$

On rappelle que δ_0 est égal à 2^{1-p}. On considère dans cette table que les pseudo-réels sont codés sur 64 bits, dont p=53 sont réservés à la mantisse.

1 km	5 km	8 km	10 km	100 km
6,6613E-10	3,3307E-09	5,3291E-09	6,6613E-09	6,6613E-08
0,00133227	0,03330669	0,08526513	0,13322676	13,3226763
0,00266454	0,06661338	0,17053026	0,26645353	26,6453526

Table de valeurs de seuils de fiabilité pour des données allant jusqu'à 100 km.

On constate aisément, en comparant les valeurs de seuils dans la table, que jusqu'à 10 km, il est possible de choisir un seuil global (par exemple 0.5) pour les trois tests choisis, mais que pour une dynamique 1mm $\leftrightarrow$ 100 km = 10^{08} mm et plus, on a intérêt à utiliser un seuil spécifique pour chaque type d'opération.

V. Contraintes, Dépendances, Hiérarchies et Priorités dans un Graphe

Maintenant que les problèmes liés à l'imprécision ont été bien cernés, il est possible de se concentrer sur des phénomènes relatifs à la cohérence et à l'interdépendance des éléments du graphe en construction.

Dans le chapitre précédent, une *rsgp* (représentation par segments d'un graphe planaire) a été dotée de primitives qui permettent la manipulation systématique de ses constituants. Le chapitre présent s'intéresse spécifiquement à des opérations plus fines sur une telle structure, à savoir celles qui permettent de *hiérarchiser* ses éléments (sommets, arêtes, faces) suivant des lois fixées et permettant de lever toute incohérence topologique qui découlerait d'un ensemble quelconque de données.

Par exemple, les projections de deux faces initiales peuvent recouvrir une même portion de plan: d'un point de vue ensembliste, ceci ne pose aucun problème. D'un point de vue topologique, ceci n'est pas acceptable car la planarité du graphe impose nécessairement que tout point du plan appartienne à une face unique du graphe.

Il est donc de première importance de pouvoir définir un ensemble de règles élémentaires qui permettent de transformer tout pré-graphe, aussi 'arbitraire' soit-il, en un graphe planaire unique cohérent.

Ce chapitre est divisé en trois sections. La première étudie la 'mise en planarité' du pré-graphe par la résolution des conflits d'arêtes et la gestion de dépendances à caractère numérique. La seconde présente l'opération consistant à paver la section rectangulaire dans lequel le graphe est plongé, par des faces organisées en hiérarchie et la section finale traite des phénomènes de dépendance sémantique entre les objets considérés dans le pré-graphe.

. REFERENCES BIBLIOGRAPHIQUES POUR LE CHAPITRE V .

A/W 77 P. Atherton, K. Weiler, *Hidden Surface Removal Using Polygon Area Sorting*, Computer Graphics, Vol. 11, pp. 214-222, (Proc. SIGGRAPH 77), 1977.

BER 58 C. Berge, *La théorie des graphes et ses applications*, Dunod, Paris, 1958.

B/OT 79 J.L. Bentley, T. Ottman, *Algorithms for Reporting and Counting Geomeric Intersections*, IEEE Trans. Comp. C-28, 9, pp. 643-647, Sept. 1979.

CHA 90 B. Chazelle, *A Linear-time Algorithm for the Triangulation of a Simple Polygon*, Conférence aux Journées de Géométrie Algorithmique, Valbonne, 1990.

C/E 88 B. Chazelle, H. Edelsbrunner, *An Optimal Algorithm for Intersecting Line segments in the Plane*, CS-TR 148-88, Department of Computer Science, Princeton University, 1988.

H/W 79 G.H. Hardy, E.M. Wright, *An Introduction to the Theory of Numbers*, Oxford University Press, 1979.

KM3 84 K. Mehlhorn, *Data Structures and Algorithms: Multi-dimensional Searching and Computational Geometry*, Vol. 3, Springer Verlag, Berlin, 1984.

MIC 87 D. Michelucci, *Les représentations par les frontières: quelques constructions; difficultés rencontrées*, Thèse de 3ème cycle, Ecole des Mines de Saint-Etienne, 1987.

P/S 85 F.P. Preparata, M.I. Shamos, *Computational Geometry - an Introduction*, Springer Verlag, New York, 1985.

ROB 77 J. Roberts, *Elementary Number Theory: A Problem-oriented Approach*, MIT Press, 1977.

ROG 85 D.F. Rogers, *Procedural Elements for Computer Graphics*, McGraw-Hill Book Company, New York, 1985.

T/VW 88 R.E. Tarjan, C.J. Van Wyck, *A n O(nloglogn)-time algorithm for triangulating a simple polygon*, SIAM J. Comput., Vol 17, No.1, 1988.

WIN 84 P.H. Winston, *Artificial Intelligence*, Addison Wesley, 1984. Version française, *Intelligence Artificielle*, InterEditions, traduction de J.M. Moreau, Paris, 1988.

V.A. LE POINT DE VUE DES ARETES

V.A.1 Définitions

Les définitions relatives aux graphes qui vont être données ici peuvent se retrouver dans [BER 58]. Soient S un ensemble et A un sous-ensemble de $S * S$. Un graphe est constitué par la donnée de la paire (S, A). Chaque élément de S est appelé un *sommet* (ou *nœud*) et chaque élément de A une *arête*. La paire de sommets d'une arête constitue ses *extrémités*. Si une arête admet le sommet s comme extrémité, on dit que l'arête est *incidente* en s. Le *degré* d'un sommet est le nombre d'arêtes incidentes en ce sommet. Un sommet est dit *isolé* s'il a un degré nul. Si $A = S * S$ le graphe est dit *complet.*

Le graphe est *orienté* si, pour toute arête a, la paire de ses extrémités est un couple ordonné $(s_{origine}, s_{fin})$. Les arêtes d'un graphe orienté portent le nom d'*arcs*. Une arête d'un graphe non orienté est en fait assimilable à deux arcs du même graphe orienté.

Soient s et t deux sommets d'un graphe; s'il existe une suite $\gamma=\{s_0=s, s_1, s_2, \ldots, s_k=t\}$ de sommets du graphe tels que $\{s_i, s_{i+1}\}$ est une arête du graphe pour $0 \leq i \leq k-1$, alors γ est une *chaîne* reliant s et t. Si le graphe est orienté, on remplace arête par arc dans la définition précédente, et l'on obtient celle d'un *chemin* entre s et t. Une chaîne ou un chemin est *simple* si tous ses sommets, sauf éventuellement le premier et le dernier, sont distincts. Dans le cas où leur premier et leur dernier sommets sont identiques, une chaîne est appelée un *cycle* et un chemin un *circuit.* Un graphe est dit *connexe* si, quel que soit le couple de sommets (s, t), il existe une chaîne entre s et t.

Soient $G = (S, A)$ un graphe, S un ensemble de points du plan euclidien P et b une bijection de S vers S . Si, quelle que soit l'arête $a=(s, t)$ de A, il existe un élément unique ζ de l'ensemble G de courbes simples du plan, paramétrées dans [0, 1], telle $\zeta(0)=b(s)$ et $\zeta(1)=b(t)$, on dit qu'il existe une *représentation topologique plane* G_{rtp} du graphe G (figure V.1), constituée par la donnée du couple (S, G).

$$S = \{0, 1, 2, 3, 4, 5, 6\}$$

$$A = \{(0, 2), (3, 4), (1,5)\}$$

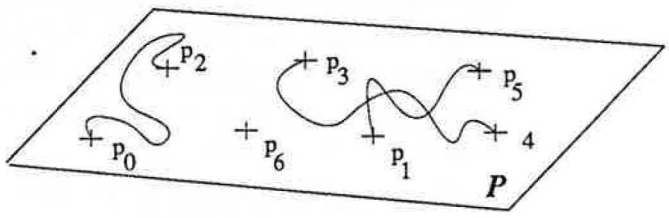


Figure V.1. Représentation topologique plane d'un graphe.

Si deux courbes simples quelconques de G_{rtp} ne se coupent, dans le pire des cas, qu'en leurs extrémités, on dit que le graphe admet une *représentation planaire* G_p.

On dit souvent aussi que le graphe est *planaire*, et, dans ce cas, on confond le graphe et sa représentation topologique plane G_p. Les composantes connexes de $\boldsymbol{P}$-G_p constituent les *faces* de G_p. Si la représentation plane du graphe planaire connexe $\boldsymbol{G}$ comporte σ sommets, α arêtes et ϕ faces, le mathématicien Euler a démontré que:

$$\sigma - \alpha + \phi = 2 \qquad \text{(V-1)}$$

Si, enfin, les courbes simples peuvent être remplacées par un ensemble A de segments de droites fermés du plan $\boldsymbol{P}$, $(\boldsymbol{S}, \boldsymbol{A})$ constitue ce qu'il est convenu d'appeler la *représentation par segments d'un graphe planaire* (*rsgp*).

Hormis la face infinie, les faces d'une rsgp sont donc des polygones bornés du plan, ses arêtes sont des segments fermés et ses sommets, des points de $\boldsymbol{P}$.

V.A.2 Transformation d'un pré-graphe en rsgp

D'après ce qui a été dit dans la section B du chapitre précédent, un pré-graphe est un ensemble de points, de segments et de polygones de $\boldsymbol{P}$ dont rien ne permet de dire qu'il constitue une rsgp.

Le module d'intersection fait le premier pas dans la transformation du pré-graphe en rsgp. Pour permettre de reconstituer la représentation planaire il est nécessaire que deux segments quelconques du pré-graphe transformé ne puisse couper un autre segment qu'en une extrémité, éventuellement. De même, un sommet du pré-graphe ne peut être inclus dans un segment ouvert de la rsgp finale.

On doit donc organiser l'intersection de deux types d'objets: segments contre segments et points contre segments: l'intersection point contre point n'a pas vraiment de sens. Comme un sommet isolé est un segment dont les extrémités sont confondues, il est simple d'unifier cette opération.

[C/E 88] propose un algorithme de complexité $O(n\log n + k)$ optimale, où n est le nombre de segments dont on cherche l'intersection et k, le nombre d'intersection de ces segments, peut être d'ordre $O(n^2)$.

Le principe fondamental qui gouverne l'algorithme de Chazelle et Edelsbrunner est une mise à jour optimale d'une structure de données (*web*) matérialisant l'écheveau de segments: chaque segment inséré dans cette structure donne lieu à un parcours de régions convexes parmi celles définies par l'ensemble des segments présents; si le nouveau segment intersecte des segments déjà insérés, les points d'intersection sont trouvés et la structure est remise à jour selon un mécanisme dont le coût amorti est optimal.

Cette structure représente, à tout moment, la topologie induite par les segments déjà insérés. Schématiquement, elle ne donne pas l'ordre total sur tous les segments, mais permet de diminuer les coûts en ne cherchant qu'à remodeler le plus localement possible la topologie globale après l'arrivée d'un nouveau segment. L'article cité en référence a permis de confirmer une conjecture très longtemps ouverte.

Pour des raisons chronologiques , l'application développée n'a pu bénéficier de cette 'chute' de borne inférieure, et c'est l'algorithme de Bentley-Ottman [B/OT 79], de complexité sub-optimale $O((n+k)\log n)$ qui a servi de support logistique à ce travail. Cet algorithme est décrit en détail dans [P/S 85] et [KM3 84] et certaines de ses caractéristiques sont présentées dans le chapitre III de cette thèse.

Un des points essentiels de l'algorithme de Bentley-Ottman est qu'il procède par un balayage du plan fondé sur l'ordre lexicographique des arêtes. Ce qui rend cette méthode non-optimale est que l'on doit 'casser' chaque segment nouvellement introduit et tous les segments qu'il rencontre en autant de sous-segments: si l'on ne garde pas trace explicitement de toutes les arêtes et sous-arêtes créées dans le processus d'intersection, la formule d'Euler, reliant le nombre de faces, d'arêtes et de sommets, n'est plus respectée. Schématiquement, un tel oubli peut être approché par l'idée que, pour chaque segment, on ne garderait trace que d'un nombre constant de points d'intersections découverts (par exemple, pour un segment donné, celles qui ont provoqué les deux valeurs extrêmes du paramètre modélisant μ). Dans ce genre de situation, on serait assez facilement en mesure de caractériser certains parmi les sommets de la face externe du graphe obtenu, comme le montre la figure V.2.

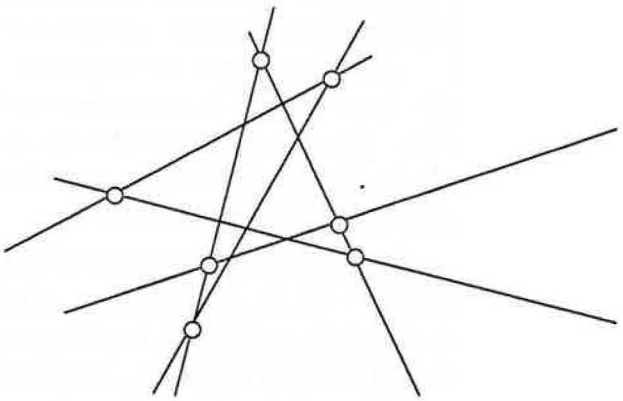


Figure V.2. Il est nécessaire de découper les segments intersectés.

A chaque insertion d'un nouveau segment, l'invariant présuppose que *tous les autres segments* forment un graphe planaire et son introduction prolonge cette propriété en localisant, à la verticale de la droite (ou *barre*) de balayage, les deux segments qui encadrent le segment inséré dans l'ordre vertical: s'il intersecte l'un ou l'autre de ses voisins, le graphe courant est de nouveau rendu planaire en construisant autant de sous-arêtes (et donc implicitement de faces) qu'il est nécessaire pour que la formule d'Euler soit valide. Un argument identique peut être tenu pour le cas d'une suppression de segment, lorsque la droite de balayage parvient à la verticale de son extrémité finale.

Pour ce qui est de la complexité, chaque intersection découverte provoque l'apparition d'un des k points de génération supérieure à zéro du graphe planaire en construction. Ce point, événement imprévisible initialement, est inséré dans la structure de file d'attente des événements jusqu'ici prévus.

La taille de cette file est, en fait, proportionnelle à $n+k$ et les opérations d'insertion ont une complexité de $O(log(n+k))$ dans le pire des cas. La complexité globale de l'algorithme est donc $O((n+k)\log(n+k))$, soit $O(n+k)\log n)$, puisque k est lui-même en $O(n^2)$.

Malgré tout, on peut arguer à juste titre du fait que, dans la réalité des cas *normaux* (dans laquelle s'inscrit assez naturellement la représentation de cartes géographiques), le pire des cas, bien que possible et à ne *jamais* exclure, n'est pas le plus fréquent. Il n'est jamais négligeable de pouvoir obtenir un algorithme dont la complexité est fonction de celle des données en sortie: c'est exactement le cas des deux algorithmes cités ici, avec une supériorité évidente pour le premier.

Mais il n'est pas non plus négligeable, lorsque la complexité de sortie a de grandes chances d'être 'raisonnablement faible', d'être encore plus explicite sur la relation entre les deux complexités: il serait souhaitable ici, de disposer d'un autre algorithme, bien sûr non optimal dans le pire des cas, mais dont la complexité serait *faiblement* optimale lorsque k est petit. Par exemple, une complexité de $O(n\log n + k\log k)$, qui se ramène à la complexité de l'algorithme de Bentley-Ottman dans le pire des cas, est bien préférable à cette dernière lorsque k est petit.

Les segments verticaux posent des problèmes, même dans le cas de l'algorithme de Bentley-Ottman original. Pour simplifier l'étude, il est pratique de considérer que la direction de balayage choisie est telle que l'on ne trouve pas de segments exactement verticaux et que les abscisses de tous les segments sont distinctes. Ces restrictions ne sont liées qu'à des impératifs d'analyse de complexité et il suffit de prolonger les techniques décrites dans les cas non-dégénérés pour trouver une solution valide dans tous les cas.

Il est possible d'affirmer que la structure de balayage ne contient jamais plus de n segments: schématiquement, l'abscisse du point d'intersection de deux segments quelconques constitue un pivot avant lequel les deux segments étaient représentés par deux demi-segments et après lequel ils sont représentés par deux nouveaux demi-segments: ces quatre demi-segments sont les quatre segments de nouvelle génération provoqués par l'intersection des deux segments originaux. S'il est vrai que le nombre total de segments obtenus, en fin du processus de balayage, est de l'ordre de $n+k$, la structure de balayage ne fait qu'entériner k remplacements de deux demi-segments par deux autres!

Imaginons la très simple variante de l'algorithme de Bentley-Ottman dans laquelle on dispose de trois structures: la première, B, est un arbre équilibré qui matérialise la droite de balayage, ne contient que des représentants éventuellement modifiés des n segments originaux et dont on verra que la gestion totale (insertions, suppressions, localisations et inversions) est en $O(n\log n)$. La deuxième structure, Ep, est une liste chaînée matérialisant la file d'attente des événements prévus (les extrémités des n segments initiaux, dont le nombre est, au plus, $2n$), dont la construction nécessite $O(n\log n)$ étapes, mais dont la seule utilisation sera une lecture linéaire. La dernière structure, E_i, est une file

de priorité représentée par un arbre équilibré qui contient tous les événements imprévus. Sa gestion globale sera de l'ordre de $O(k\log k)$.

Dans un premier temps, on constate que ce que l'on tente de faire est de marginaliser les événements imprévus, de façon à pouvoir modéliser plus naturellement une situation dans laquelle il y en a peu, voire, à la limite, aucun. Le point important est qu'il faut profiter de l'information contenue dans la découverte d'une intersection.

Dans l'algorithme traditionnel de Bentley-Ottman, on découvre une intersection lorsque deux segments, disons s et t, deviennent adjacents et se coupent. Si la structure E_i est bien conçue, elle doit non seulement contenir l'abscisse $x_{s,t}$ de cette intersection, mais en plus les adresses des deux segments dans la barre de balayage: comme les segments ne sont supprimés que lorsque l'on parvient à leur extrémité finale, quand on arrivera à la verticale de l'abscisse $x_{s,t}$ il sera possible en $O(1)$ d'aller inverser les positions des représentants de s et t dans la barre de balayage: à ce stade, rien n'est changé (pente, adresse...) les concernant, si ce n'est leurs positions relatives dans la barre, sachant qu'ils ont leur point d'intersection en commun.

Si l'événement $x_{s,t}$ ne provoque qu'une seule inversion de ce type, il suffit d'examiner le voisin directement au-dessus du segment supérieur et le nouveau segment directement en-dessous du segment inférieur. Ces tests occasionnent la découverte éventuelle de deux nouvelles intersections et le processus peut continuer comme précédemment.

Si, au contraire, l'abscisse $x_{s,t}$ s'accompagne de plusieurs événements imprévus (i.e. on a découvert à la verticale de cette abscisse plusieurs intersections de segments), on adopte la même technique pour chaque événement successif à la même abscisse: comme ceux-ci arrivent dans l'ordre lexicographique, la dernière inversion donne l'ordre vertical exact final; certains segments seront testés deux fois mais aucune intersection qui serait détectable à cette abscisse ne peut être oubliée.

Le processus d'inversion demande un temps constant si l'on a pris la précaution de munir la structure d'arbre équilibré d'un double chaînage supplémentaire, matérialisant la liste doublement chaînée de voisinage vertical. Cette liste ne sert qu'à permettre les inversions et n'est jamais utilisée pour les insertions et les suppressions dans E_p.

Algorithme de Bentley-Ottman amélioré

Pour formaliser un peu cet algorithme, donnons quelques informations sur les structures de données qu'il nécessite. Chaque arête est une structure munie d'un champ *origine* et d'un champ *fin*, qui sont des pointeurs sur des nœuds de l'arbre des points du domaine, décrit dans le chapitre précédent. On doit disposer d'un champ supplémentaire *b_elt* qui indiquera l'élément de la barre de balayage matérialisant cette arête: lorsqu'une arête est insérée dans la barre, l'adresse de l'élément dans lequel elle est placée est inscrite dans ce champ; lorsque cette arête doit être supprimée, le champ donne directement la position de l'élément de barre qui la contient. Ceci permet des opérations de localisation et d'échange en temps constant dans la barre de balayage.

La structure de balayage est un arbre binaire équilibré dans lequel tout nœud peut fournir son prédécesseur et son successeur dans l'ordre vertical, à l'aide des champs *b_sous* et *b_sur*. L'insertion d'un nouvel élément et la suppression d'un élément déjà présent dans la barre de balayage se font en $O(\log n)$, si n est le nombre de segments considérés.

La recherche de l'élément précédant ou suivant un élément d'adresse connue dans la barre se fait en $O(1)$, grâce aux champs définis précédemment. Lors de l'insertion d'un nouvel élément, on utilise les champs *fils_g* et *fils_d* pour se déplacer en profondeur dans l'arbre jusqu'à atteindre la bonne position. Cette opération, jointe au ré-équilibrage dynamique, est la seule qui s'accompagne d'une complexité logarithmique.

La mise à jour des champs *b_sous* et *b_sur* est alors immédiate par la connaissance des mêmes champs pour les deux voisins supérieur et inférieur découverts au nouvel élément lors de l'insertion, avant ré-équilibrage. Dans la formalisation de l'algorithme proposée, l'opération Insertion_Barre retourne l'adresse @ de l'élément représentant l'arête dans la barre et celles, *sup* et *sub*, des deux arêtes (éventuellement inexistantes) représentées par les voisins immédiats de cet élément sur la barre.

La structure de données représentant un événement prévisible (extrémité d'un segment original) est composée d'un drapeau booléen, *extrémité*, de valeur *origine* ou *fin*, indiquant le type d'extrémité. On a aussi besoin d'un second champ, *arête*, qui est un pointeur sur l'arête correspondante. Le drapeau booléen est simplement mis à jour lors de la construction de la structure E_p.

La structure de données représentant les événements imprévisibles (consécutifs à une intersection *normale*) est constitué d'un champ *pt* qui est un pointeur sur le nœud de l'arbre binaire des points, matérialisant l'intersection correspondante. On trouve aussi deux pointeurs, *bas* et *haut*, sur les arêtes dont les représentants sur la droite de balayage devront être inversés, le moment venu.

A ce stade, on rappelle que la structure relative à une arête permet de connaître la position de l'élément qui matérialise celle-ci dans la droite de balayage.

A chaque itération, on doit trouver quel est le prochain événement à traiter: si $évt_p$ est l'élément courant de la liste des événements prévisibles non traités et $évt_i$ est le premier élément de la file de priorité des événements imprévisibles restants, la primitive *premier*($évt_p$, $évt_i$) est chargée de retourner celui, parmi ces deux événements qui est le plus à gauche lexicographiquement, ainsi que le type d'événement (*prévisible* ou *imprévisible*), le pointeur sur l'arête concernée dans le cas *prévisible* ou les deux pointeurs sur arêtes dans le cas *imprévisible*. Dans l'algorithme présenté ci-contre, la variable *événement* capture formellement toutes les informations retournées par cette dernière primitive.

L'algorithme de Bentley-Ottman amélioré peut donc se traduire de la manière suivante:

```
Construire la structure E_p;
e_p ← premier élément de la liste des événements E_p;
e_i ← l'élément de plus haute priorité de la file de priorité E_i;
tant que (e_p ≠ NIL ∧ e_i ≠ NIL)
    événement ← premier(e_p, e_i);
    si (événement.type = prévisible)
        si (événement.extrémité = origine)
            arête ← événement.arête;
            (@, sup, sub) ← Insertion_Barre(arête );
            si ((pt ← sup ∩ arête) ≠ Ø)
              Insérer_E_i(pt, arête, sup);
            si ((pt ← sub ∩ arête) ≠ Ø)
              Insérer_E_i(pt, sub, arête);
        sinon                                   ¡suppression d'une arête¡
            Supprimer(arête.barre) dans la barre;
            sup, sub ← voisins directs de l'élément supprimé;
            si ((pt ← sup ∩ sub) ≠ Ø)
              Insérer_E_i(pt, sub, sup);
        e_p ← suivant_E_p(e_p);
    sinon                                       ¡événement prévisible¡
        haut ← événement.haut;
        bas ← événement.bas;
        Inverser(bas.barre, haut.barre) dans la barre;      ¡haut est sous bas¡
        sup ← nouveau voisin au-dessus de bas;
        sub ← nouveau voisin en-dessous de haut;
            si ((pt ← sup ∩ bas) ≠ Ø)
              Insérer_E_i(pt, bas,sup);
            si ((pt ← sub ∩ haut) ≠ Ø)
              Insérer_E_i(pt, haut, sub);
        supprimer_E_i(e_i );
        e_i ← l'élément de plus haute priorité de la file de priorité E_i;
```

Algorithme V-1: Algorithme de Bentley-Ottman amélioré.

Afin d'illustrer le fonctionnement de cet algorithme, prenons l'exemple de cinq segments [A,J], [B,G], [C,E], [D,I] et [F,H] (figure V.3), dont les extrémités sont dans le même ordre lexicographique relatif que les lettres qui les matérialisent. Cet exemple regroupe un certain nombre de situations typiques de l'algorithme original.

Sur la figure, les segments verticaux en pointillés symbolisent les événements prévisibles et ceux en continu indiquent les événements de type imprévisible. Les numéros 1, 2,...,5 n'indiquent pas l'ordre dans lequel ces événements sont découverts, mais celui dans lequel ils se succèdent sur l'axe horizontal des événements.

Dans la suite, (x) symbolisera l'événement imprévisible de numéro x, et *inv*(x) l'inversion dictée par l'événement x: si (x) est de type $\begin{matrix}UV\\YZ\end{matrix}$, les segments [U,V] et [Y,Z] doivent être inversés dans la structure de balayage lorsque *inv*(x) est rencontré. *ins*[U,V], *sup*[U,V] symbolisent l'insertion et la suppression du segment [A,B] de la barre de balayage dont les éléments seront listés du plus bas vers le plus haut. On rappelle que lorsqu'un élément de la droite de balayage est inséré ou changé de place, on doit systématiquement le tester avec ses voisins immédiatement au-dessus et en-dessous (s'ils existent) pour vérifier si une

intersection est possible. De même, lorsqu'un élément est supprimé, on confronte au test ses deux éléments immédiatement voisins, s'ils existent. Le symbole *versus* exprime ces considérations dans ce qui suit.

ins[A,J]	AJ	
ins[B,G]	BG AJ	[B,G] versus [A,J] = Ø;
ins[C,E]	BG AJ CE	[A,J] versus [C,E] = (1) $\frac{AJ}{CE}$.
inv(1) $\frac{AJ}{CE}$	BG CE AJ	[B,G] versus [C,E] = Ø.
ins[D,I]	DI BG CE AJ	[D,I] versus [B,G] = Ø.
sup[C,E]	DI BG AJ	[B,G] versus [A,J] = Ø.
ins[F,H]	FH DI BG; AJ	[F,H] versus [D,I] = (2) $\frac{FH}{DI}$.
inv(2) $\frac{FH}{DI}$	DI FH BG; AJ	[F,H] versus [B,G] = (3) $\frac{FH}{BG}$.
inv(3) $\frac{FH}{BG}$	DI BG FH AJ	[D,I] versus [B,G] = Ø, [F,H] versus [A,J] = (4) $\frac{FH}{AJ}$.
inv(4) $\frac{FH}{AJ}$	DI BG AJ FH	[B,G] versus [A,J] = Ø.
sup[B,G]	DI AJ FH	[D,I] versus [A,J] = (5) $\frac{DI}{AJ}$.
sup[F,H]	DI AJ	pas de test à faire.
inv(5) $\frac{DI}{AJ}$	AJ DI	pas de test à faire.
sup[D,I]	AJ	pas de test à faire.
sup(AJ)		Fin.

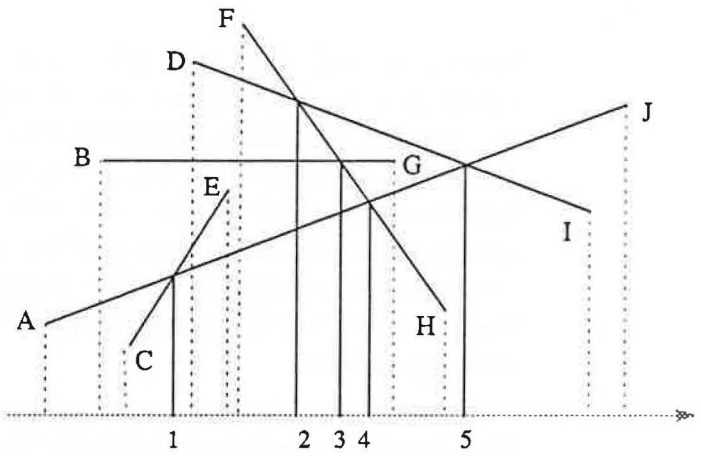


Figure V.3. Segments pour l'algorithme amélioré de Bentley-Ottman.

En ce qui concerne la complexité de l'algorithme, on commence par un tri et la construction de la liste linéaire des événements prévisibles E_p, qui s'accompagnent d'une complexité de $O(n\log n)$. Le nombre d'itérations de la boucle **tant que** est $2n+k$. On passe ainsi $2n$ fois exactement dans la section prévisible et k fois dans la section imprévisible. Le coût global relatif à la section prévisible est $O(n\log n)$ puisque chaque opération a au maximum un coût logarithmique. Chaque inversion s'exécute en temps constant, grâce au choix de structures de données. La gestion de la file de priorité des événements imprévisibles s'accompagne donc d'un coût de $O(k\log k)$. La complexité globale de l'algorithme V.i est $O(n\log n+k\log k)$.

Mise à jour des segments

Dans l'algorithme original de Bentley-Ottman, le passage de la droite de balayage à la verticale d'un événement 'intersection' provoque la suppression de deux arêtes périmées (d'extrémité coïncidant avec le point d'intersection) et l'insertion de deux nouvelles arêtes (d'origine coïncidant avec le point d'intersection). La technique suggérée par l'algorithme V.i. élimine ces actions de l'algorithme original.

On ne semble jamais subdiviser les segments originaux en sous-segments dont les extrémités sont des points de génération strictement supérieure à zéro, ce qui serait ennuyeux dans une situation pratique, car on s'attend à ce que les segments sortant du module d'intersection reflètent les intersections détectées. Ce problème n'est qu'apparent: quand une intersection normale est détectée entre les segments *courants* s et t, un sommet de génération non nulle est créé et il suffit, au moment où la barre de balayage parvient à la hauteur de l'événement correspondant de créer deux arêtes de toutes pièces: pour chacune l'origine sera donnée par l'origine courante de l'arête s ou t et l'extrémité est égale au point d'intersection; de plus, les segments inversés (dont les adresses sont celles des segments initiaux qui, eux, restent jusqu'au bout sur la droite de balayage) se voient attribuer comme origine le point d'intersection. Quand un segment est supprimé de la droite de balayage, il a comme origine le dernier point d'intersection qu'il a engendré et comme extrémité finale, celle du segment initial dont il provient.

Répétitions

Non seulement l'algorithme de Chazelle et Edelsbrunner est-il supérieur à celui de Bentley-Ottman du point de vue de la complexité, mais encore possède-t-il un atout supplémentaire: chaque segment n'est introduit qu'une seule fois et, par construction, ne peut être confronté à un autre segment quelconque qu'une seule fois: on ne peut manquer de remarquer que la méthode de Bentley-Ottman comporte intrinsèquement le risque de tester plusieurs fois l'intersection de deux mêmes segments. Pour s'en convaincre aisément, il suffit de considérer l'exemple simple de la figure V.4.

L'intersection des deux segments horizontaux est testée un première fois au départ, mais aussi à chaque fois qu'un des segments obliques est supprimé de la barre de balayage. Ceci est encore une fois très ennuyeux dans une solution utilisant une arithmétique mixte ou exacte, car si le test d'intersection de deux segments fait intervenir une fois l'arithmétique exacte, il la fait intervenir, en fait, autant de fois que le test est sollicité! La situation devient inacceptable lorsque les deux segments persécutés se coupent effectivement...

Ce défaut connu n'a jamais été éliminé, à ma connaissance, sauf par la gestion assez lourde d'un arbre équilibré des couples de segments testés. On peut néanmoins résoudre ce problème d'une manière assez simple, à l'aide de structures annexes dont la gestion s'accompagnera d'un temps amorti constant et d'une taille proportionnelle au nombre de tests entre segments effectués.

Convenons de quelques notations qui vont permettre d'alléger les explications. L'élément de la droite de balayage correspondant au segment s sera noté $\underline{s}$. Cet élément s'accompagnera d'une liste linéaire ordonnée (appelée *liste de tests*), constituée de pointeurs sur des segments, et qui sera notée $l(\underline{s})$. Ainsi, $l(\underline{s})$: a,b,c signifie "la liste de la droite de balayage correspondant au segment s est composée des références aux arêtes a, b c, dans cet ordre". '$l(\underline{s})$:' désigne une liste vide. Le chaînage de ces listes va dans les deux sens.

Chaque élément de la barre de balayage se voit attribuer une telle liste au cours du processus décrit dans l'algorithme V.i qui sert de toile de fond aux explications. On note v.s. (resp. v.i.) le voisin immédiatement au-dessus (resp. en-dessous) d'un élément de la barre de la balayage.

Partons d'une barre vide dans laquelle vont être insérés les quatre segments de la figure V.5. Ces segments sont numérotés suivant l'ordre lexicographique dans lequel ils seront insérés, et les lettres le long de l'axe horizontal désignent les événements rencontrés dans l'algorithme de balayage: $\{a,b,c,d,g,h,i,j\}$ pour les événements prévisibles et $\{e,f\}$ pour les événements imprévisibles. On va suivre le déroulement des événements en apportant quelques modifications à l'algorithme V.i, qui seront d'abord introduites 'en situation', puis justifiées dans un deuxième temps. Dans ce qui suit, on ne considère que la relation 't est le v.s. de s' puisque la relation opposée 's est le v.i. de t' s'obtient trivialement. Chaque itération de l'algorithme de Bentley-Ottman donne lieu à des relations de ce genre.

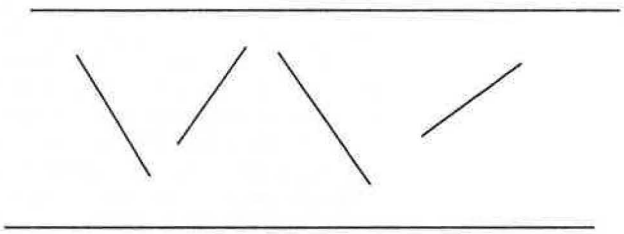


Figure V.4. Cas pathologique pour le test de deux segments.

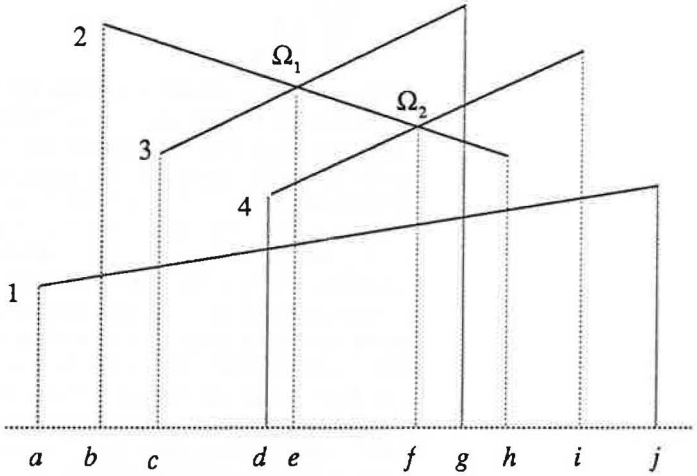


Figure V.5. Limitation du nombre de tests d'intersection.

a insertion de 1, comme 1 n'a pas de v.s. , la liste correspondante est *l*(1):.

b insertion de 2, 2 =v.s(1), ceci donne les deux listes:
l(2):
l(1): 2

c insertion de 3, comme 2=v.s.(3), cette référence est placée dans sa liste, mais 3 devient le nouveau v.s. de 1: on ajoute en tête de liste la référence correspondante, sans supprimer la référence à 2, déjà présente.
l(2):
l(3): 2
l(1): 3,2
La procédure d'intersection signale que 2 et 3 se coupent en Ω_1 et cette information est consignée, en plus de celles nécessaires à l'exécution de l'algorithme V.i, sous la forme sibylline suivante: 1, le v.i. de 3, étant le seul témoin[1]possible de cette intersection contient *éventuellement* une référence à 2: **si c'est le cas**, 2 doit se trouver cité en deuxième position de *l*(1), car 3 est en ce moment le v.i. de 2. Il est facile de vérifier que ceci est vrai, dans *l*(1). Ainsi, on garde, dans la structure relative à l'événement marqué *e* sur le graphique, l'adresse *de l'élément 2* de la liste *l*(1).

d insertion de 4, mise à jour des listes faciles car pas d'intersection
l(2):
l(3): 2
l(4): 3
l(1): 4,3,2

e croisement de 2 et 3. Outre celles concernant directement l'algorithme V.i, les informations stockées avec Ω_1 indiquent que l'élément 2 de la liste *l*(1) doit être inversé avec son voisin de gauche: ceci est effectué en temps constant grâce au double chaînage; la liste devient *l*(1): 4,2,3.
L'algorithme V.i provoque l'inversion des deux éléments de barre 2 et 3, ce qui veut dire que l'harmonie qui régnait dans la série de listes précédentes risque de souffrir quelque peu! Mais, si 2 passe en dessous de 3, 3 perd son premier élément de liste (référence ancienne à 2 qui était son v.s.) et 2 reçoit en échange 3 comme tête de liste: ceci entérine le fait que l'on sait déjà que 2 et 3 ont été trouvés adjacents puisque sécants. Dans un premier temps, on trouve naturellement les nouvelles listes suivantes:
l(3): *l*(2): 3 *l*(4): 3 *l*(1): 4,2,3

[1] Dans toutes ces explications, témoin signifiera 'témoin voisin inférieur'.

L'algorithme V.i impose la comparaison de 2 et de son nouveau voisin inférieur, 4: avant d'entériner l'intersection trouvée, remarquons que cette comparaison coïncide avec l'insertion en tête de liste de 4 d'une référence à 2: ceci est l'écho du processus correspondant dans l'algorithme V.i, qui s'éclaire maintenant sous un jour nouveau; cette action a pour principale conséquence de conserver un invariant primordial, que nous appelerons 'invariant de première référence': toute liste, sauf la plus haute, débute par une référence directe au v.s. local!
Les nouvelles listes complètes sont donc:

l(3):
l(2): 3
l(4): 2,3
l(1): 4,2,3

L'invariant de première référence est satisfait partout.
Revenons sur l'intersection Ω_2 constatée entre 2 et 4: le v.i. de 4 (le plus 'bas' des deux segments sécants) est encore une fois le seul témoin possible de cette dernière. Ici encore, **s'il a été adjacent à un moment quelconque** à 2, une référence à ce segment doit se trouver dans la liste de 4, immédiatement après celle à 3, son actuel v.s.: c'est encore vrai ici. Sinon, aucune action ne devrait être prise. Encore une fois, on garde avec Ω_2 dans l'événement noté f, l'adresse de l'élément 2 de *l*(4).

f croisement de 2 et 4: on doit inverser ces deux segments dans la barre mais aussi, comme il a été consigné dans la structure relative à l'événement f, on doit inverser l'élément 2 avec son voisin gauche dans *l*(4). On obtient alors les listes suivantes. L'inversion de 2 et 4 dans la barre ne s'accompagne d'aucune découverte d'intersection, car l'invariant de première référence est satisfait partout, dès que les transformations précédentes et celles de l'algorithme V.i sont effectuées:

l(3):
l(4): 3
l(2): 4,3 (ajout automatique de 4 dans le 'croisement')
l(1): 2,4,3

L'information d'adjacence passée de 1 et 2 n'a pas été perdue!

g suppression de 3: *l*(3) disparaît et la liste du voisin inférieur devient vide, après perte de sa tête de liste. 3 reste présent dans les listes inférieures, mais aucune action ne peut être entreprise pour l'instant. Il reste:

l(4):
l(2): 4,2,3
l(1): 2,4,3

h suppression de 2: *l*(2) doit être 'nettoyée' et il faudra vérifier le coût global de ce type d'opération. La tête de liste de *l*(1) saute et il reste:

l(4):
l(1): 4,3

i suppression de 4, il reste *l*(1): 3, mais la référence à 3 peut être supprimée si l'on sait que 3 a bien disparu de la barre. Ceci peut se faire si l'on dispose d'un drapeau booléen à cet effet dans la structure *arête*.

j suppression de 1: fin.

Le déroulement de cet algorithme n'est pas suffisant pour prouver la validité du processus quelles que soient les circonstances, mais il apporte l'idée principale du mécanisme, qu'il va falloir justifier maintenant.

Il est nécessaire de démontrer un certain nombre de propriétés qui vont permettre cette justification. Soit s un segment quelconque présent sur la barre sous la forme $\underline{s}$ et x une abscisse sur l'axe horizontal. On dit qu'un point admet s comme v.s. (resp. v.i.) en x si s est le plus proche segment en-dessous (resp. au-dessus) de s à la verticale de x. De plus, si $l(\underline{s})$: $a,b,c, \ldots$ est une liste de tests, on dira que cette liste (ou ses éléments) forme(nt) un *sous-ensemble en ordre à la verticale de* x si la suite de segments $a, b, c, \ldots$ constitue un sous-ensemble de l'ensemble général S_x des segments déjà traités à la verticale de x dans lequel tout couple est dans le même ordre que le couple correspondant de S_x.

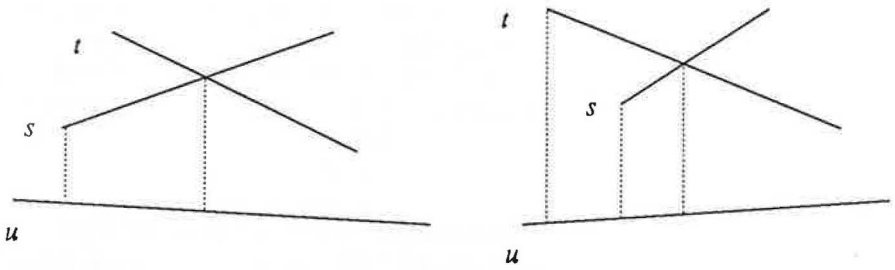


Figure V.6. Adjacence au voisinage d'une intersection.

Intuitivement, l'ordre de la liste considérée reflète exactement une partie de l'ordre dans lequel tous les segments déjà traités se succèdent sur la barre de balayage.

On suppose enfin que tous les événements ont des abscisses différentes: ceci permet d'éliminer les cas dégénérés de segments verticaux ou de plus de deux segments se coupant en un point. Ces restrictions ne sont imposées que pour faciliter la présentation des démonstrations, et peuvent être levées aisément.

Lemme 1: Si les segments s et t se coupent en Ω, il existe au maximum un plus proche segment déjà inséré sous Ω.

Démonstration: par construction.

Corollaire 1: Si les segments s et t se coupent en Ω, il existe au maximum une seule liste de tests contenant la suite s, t dans cet ordre ou dans l'ordre inverse.

En effet, s'il existe, le seul segment du lemme 1 peut éventuellement être v.i., successivement, de s et t. On voit sur la figure V.6 que seul le diagramme de gauche répond aux critères et donne la suite t, s dans la liste de tests de u, alors que l'inversion du lettrage donnerait la suite s, t..

Lemme 1: (*Invariant de l'ordre vertical*) Après chaque inversion de deux segments adjacents, toute liste de la barre est un sous-ensemble en ordre à la verticale de l'abscisse d'inversion.

Remarque: il va sans dire que ce lemme s'applique trivialement aux insertions et aux suppressions.

Démonstration: Soit Ω le point d'intersection des segments s et t. Juste avant d'arriver en Ω, la droite de balayage est supposée représenter le bon ordre des segments présents. En Ω, les positions relatives de s et t s'inversent. S'il existe un segment, parmi ceux déjà considérés, qui a été v.i. de s et t successivement et dans cet ordre, il est le seul à pouvoir prétendre posséder, dans sa liste de tests, la suite t, s (corollaire précédent). Il n'y a donc, au maximum, qu'une seule suite de cette sorte à inverser parmi les listes de tests. Ceci est fait par le processus d'inversion et comme s et t sont les seuls segments qui sont à inverser localement, la barre de balayage repart avec une cohérence totale entre les ordres partiels de ses listes de tests et l'ordre global des segments, dans un le voisinage compris entre l'abscisse de Ω et celle du prochain événement.

Lemme 3: (*Invariant de première référence*) Après chaque action élémentaire sur la droite de balayage (insertion d'un segment, suppression d'un segment, inversion de deux segments voisins), la liste de tests de tout élément, à l'exception du plus haut sur la barre, débute par une référence à son voisin immédiatement au-dessus.

Démonstration: Il faut détailler chaque opération. Quand on insère le segment u entre les segments s et t, l'état de la barre de balayage avant est:

...
$l(t)$: r, ...
$l(s)$: t, ...
...

u ne peut avoir été déjà comparé à un autre segment, puisqu'il est nouveau, donc on le compare à s et t dans l'algorithme V.i; ceci se traduit ici par l'insertion en tête de liste de $\underline{u}$ d'une référence à t et en tête de celle de $\underline{s}$ une référence à u, par construction, ce qui donne:

$l(\underline{t})$: r, ...
$l(\underline{u})$: t,r, ...
$l(\underline{s})$: u,t, ...

L'insertion conserve donc l'invariant de première référence. Considérons maintenant l'inversion de deux segments s et t. Si les deux segments ont eu un voisin inférieur commun w, le processus d'inversion provoque une inversion de s,t dans la liste de tests de w. On suppose que la droite de balayage contient, du bas vers le haut, les éléments suivants, juste avant l'inversion: $\underline{r}$, $\underline{s}$, $\underline{t}$, $\underline{u}$. r est donc le v.i. de s. Deux cas de figure se présentent: ou bien r a été adjacent à t dans le passé, ou bien non.

Traitons le second cas tout de suite: en se référant à la figure V.7.a (et en ignorant pour l'instant les segments issus de k), r est le v.i. de s juste avant l'inversion, et devient celui de t juste après: comme r n'était pas adjacent à t avant, il faut le confronter aux deux segments. Ceci se traduit simplement, en termes de liste: il est sûr que $l(\underline{r})$ ne contient pas de référence à t dans le cas où r et t n'ont jamais été reconnus adjacents, et la confrontation imposée par l'algorithme V.i. fait apparaître une référence à t en tête de liste de $l(\underline{r})$. Comme, d'autre part, la référence à s est automatiquement insérée en tête de $l(\underline{t})$ (cf. l'exemple traité plus haut) les listes de tests avant et après l'inversion ont l'allure suivante:

$l(\underline{u})$: ...	$l(\underline{u})$: ...
$l(\underline{t})$: u,...	$l(\underline{s})$: ...
$l(\underline{s})$: t,...	$l(\underline{t})$: s,u...
$l(\underline{r})$: s,...	$l(\underline{r})$: t,s...

La seule incertitude, concernant l'invariant, se situe au niveau de la liste $l(\underline{s})$ dont ne peut pas, pour l'instant, caractériser le premier élément. Si, comme précédemment, s n'a jamais été adjacent à son éventuel nouveau v.s. u, cette référence est ajoutée, en tête de $l(\underline{s})$ et l'invariant est conservé. Sinon, si $l(\underline{s})$ débute par une référence à un segment k, différent de u, cela signifie que ce segment a été un v.s. de s après u: Si ce segment a été supprimé, il peut disparaître de cette liste et l'on continue jusqu'à retomber sur le cas suivant: ce segment k est encore actif (son extrémité finale a une abscisse supérieure à celle de l'intersection de s et t); il ne peut plus être au-dessus de s car sinon, il aurait empêché l'adjacence de t et u juste avant l'intersection; il ne peut couper s et t en même temps, car sinon ces deux derniers ne pourraient pas être adjacents comme ils le sont en ce moment; reste la seule possibilité (kk' sur la figure) où k coupe s avant que s ne coupe t: la technique d'inversion a nécessairement supprimé k de la liste de s, au moment où k est passé en-dessous de s (d'après le lemme 1). La tête de liste de $l(\underline{s})$ est donc forcément u et l'invariant est encore conservé.

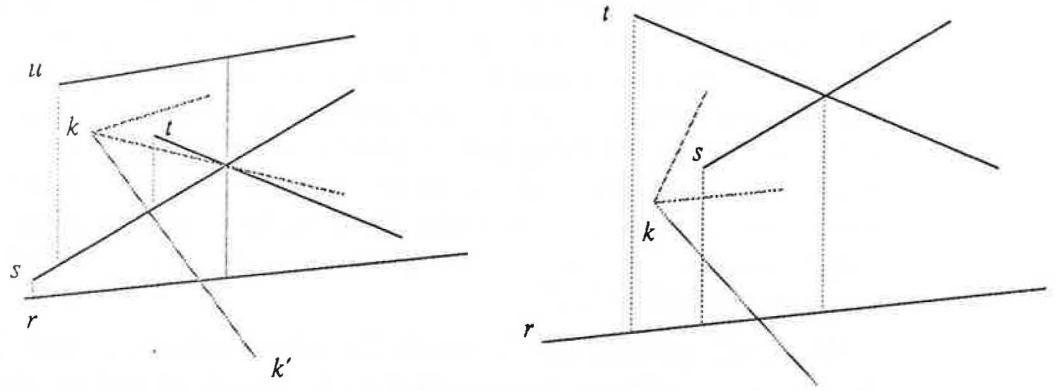
Si l'élément en tête de $l(\underline{s})$ est actif et différent de u, on peut affirmer que s et u n'ont jamais été adjacents et on doit confronter les deux segments.

Dans le cas où r a été adjacent à t (f-V.7.b), comme il est aussi adjacent à s, la suite s,t est renversée dans le processus d'inversion et se trouve maintenant en phase avec l'ordre relatif des deux segments à droite de Ω. L'invariant est nécessairement respecté dans ce cas. Les questions relatives à s sont identiques à celles qui ont obtenu une réponse précédemment.

Il reste à considérer le cas d'une suppression de segment. Les listes avant et après la suppression de s se présentent de la façon suivante:

$l(\underline{t})$: ...	$l(\underline{t})$: ...
$l(\underline{s})$: t,...	
$l(\underline{r})$: s,x...	$l(\underline{r})$: x,...

Si $x = t$, l'invariant est conservé. Sinon, si x est périmé, on le supprime de la liste et l'on recommence jusqu'à retomber sur la situation suivante: x n'est pas périmé et différent de t; on en déduit que u ne peut être situé plus à droite dans la liste, par le lemme 1, car x ne serait pas à sa place.



(a) (b)

Figure V.7. Deux cas d'inversion.

```
InsérerListeTests(s,u,t)                          ¡insertion de u entre s et t¡
    insérer la référence de t en tête de l(u);
    marquer l'arête u 'en service';
    si (s≠ NIL ∧ réf(l(s) ≠ u)
        confronter s et u;
        insérer la référence de u en tête de l(s);
SupprimerListeTests(s,u,t)                        ¡suppression de u entre s et t¡
    vider la liste de u;
    marquer l'arête u 'hors service';
    si (s ≠ NIL)
        tant que la référence en tête de l(s) est périmée, la supprimer;
        si (l(s) ≠ NIL ∨ réf(l(s) ≠ t))
            confronter s et t;
            insérer la référence de t en tête de l(s);
InverserListeTests(u,t,s,r,Z)                     ¡inversion de s et t¡
    supprimer référence de t en tête de l(s);
    si (u ≠ NIL ∧ réf(l(s) ≠ u)                   ¡confrontation avec v.s.?¡
        confronter u et s;
        insérer référence à u dans l(s);
    insérer référence à s dans l(t);
    si (r ≠ NIL)                                  ¡confrontation avec v.i.?¡
        tant que la référence en tête de l(s) est périmée, la supprimer;
        si (l(r) ≠ NIL ∨ réf(l(t) ≠ t))
            confronter t et r;
            insérer référence à t en tête de l(r);
    si (Z ≠ NIL) Inverser Z et son prédécesseur;
```

Algorithmes V-II: Gestion des listes de tests.

Donnons quelques détails sur l'implantation de la solution proposée. La structure d'arête est munie d'un drapeau booléen de valeur 'en service' ou 'hors service' pour indiquer si l'arête est sur la droite ou pas. Les listes de tests sont des listes linéaires avec accès en tête et double chaînage. L'inversion de deux éléments se fait alors en temps constant dès que l'on donne l'adresse de l'élément de droite à inverser.

Le paramètre Z est gardé avec les informations accompagnant la structure d'événement imprévisible. Ceci implique que, lorsque l'algorithme V.i détecte l'intersection de deux segments *s* et *t*, il est nécessaire de repérer, dans la liste du voisin inférieur du segment qui, entre *s* ou *t*, est le plus bas sur la barre, le deuxième élément de liste de tests: si cet élément est bien une référence à *t*, on garde la trace de l'adresse Z correspondante, sinon Z est mise à NIL. Cette opération se fait, bien entendu, en temps constant.

Cette réflexion amène à rectifier un point de détail antérieur: si u est le segment le plus proche *en-dessous* et témoin de l'intersection de s et t, mais n'est pas le voisin inférieur r au moment où l'intersection de s et t est *détectée* (voir la figure V.8), lui aussi pourrait prétendre posséder s et t successivement dans sa liste de tests: mais on sait à l'avance qu'il n'y a pas à se préoccuper de l'ordre de s et t dans la liste de u car t et s sont présents dans le bon ordre dans sa liste à la verticale de l'intersection et, après celle-ci, les deux segments s et t ne changeront plus jamais d'ordre relatif!

Il apparaît que certains éléments de listes doivent être supprimés bien après que les segments correspondants aient été supprimés: ce phénomène est justement lié au fait que l'on n'essaie jamais de détruire toutes les références à un même segment supprimé, au moment de sa suppression. En fait, on ne détruira jamais plus d'éléments de listes qu'il n'en a été créés! On sait même que grâce à cette méthode, il s'en créée moins que de nombre total d'itérations de la boucle principale de l'algorithme V.i. La complexité de l'algorithme complet, incluant les routines V.ii. est donc la même que celle du précédent.

Si l'on s'intéresse à la complexité propre des opérations concernant les listes de tests, il est clair que l'insertion d'un élément de liste se faisant toujours en tête, sa complexité est constante. L'inversion de deux éléments de liste, déclenchée par l'inversion de l'algorithme V.i, du fait des réflexions précédentes, nécessite malheureusement la suppression effective d'éventuelles références en tête de liste qui n'ont pas pu être supprimées lors de la disparition du segment correspondant.

On ne peut pas affirmer que chaque inversion (ni d'ailleurs de chaque suppression) se fait en temps constant, puisque ces deux opérations nécessitent la suppression différée de références périmées. Malgré tout, il est clair que la complexité *amortie* de ces opérations sur le nombre total d'opérations {insertion, inversion, suppression} s'effectue en temps constant.

La taille mémoire nécessaire à la création des listes de tests est proportionnelle au nombre maximal de tests effectués entre segments, c'est-à-dire au maximum $\frac{n(n-1)}{2}$ si n est le nombre de segments.

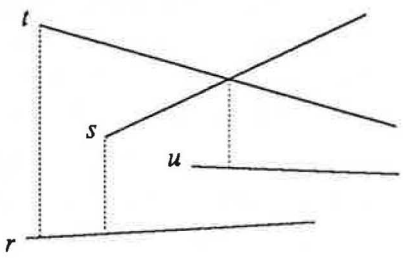


Figure V.8. Rectification concernant le voisin inférieur et la suite *s,t*.

V.A.3 Résolution de dépendances

Une fois toutes les intersections induites par le pré-graphe découvertes, on peut affirmer que l'ensemble des segments et des points résultants admet une représentation planaire.

Imaginons que chaque point original soit porteur d'une information de trois types: une partie de cette information est numérique μ, l'autre est un vecteur ϖ à plusieurs composantes permettant d'établir une relation d'ordre totale sur l'ensemble des objets; en dernier, chaque point (et donc chaque extrémité de segment de génération zéro) est en relation avec un ou plusieurs autres points de l'ensemble S initial de sommets du pré-graphe. Les lois qui régissent ces relations sont les suivantes:

- la relation qui lie deux sommets peut être réflexive ou non, symétrique ou non, mais elle n'est jamais transitive. En d'autres termes, si A est en relation avec B et B avec C, A n'est pas en relation avec C.
  De cette manière, si A est en relation avec B, l'information numérique de A sera dépendante de celle de B, si B est lui-même en relation avec un troisième élément C et ainsi de suite, l'information numérique de A ne sera connue que lorsque toutes les informations numériques de cette chaîne seront déterminées.
  Le graphe des relations liant tous les points originaux est supposé dépourvu de circuits, faute de quoi, la résolution de dépendances ne sera pas possible.

Un exemple d'une telle situation est donné par la figure V.9. qui compte cinq sommets de génération zéro. Une flèche de U vers V signifie que *U dépend de V*, ce que l'on symbolise par $U_\mu \leftarrow V_\mu$. On voit que ce pré-graphe est soumis aux dépendances suivantes, entre sommets originaux: $A_\mu \leftrightarrow A_\mu$, $B_\mu \leftarrow C_\mu$, $C_\mu \leftarrow A_\mu$, $D_\mu \leftrightarrow D_\mu$, $E_\mu \leftrightarrow E_\mu$. Par convention à partir de maintenant, il sera 'économique' de ne pas signaler les relations réflexives, ni l'indice de champ numérique, qui sera supposé fixé; ainsi, les trois 'boucles' de f-V.9 disparaissent-elles et le réseau de relations devient-il $B \leftarrow C$, $C \leftarrow A$. Résoudre ce réseau de dépendances consiste à donner une expression explicite du champ μ de chaque sommet; ici, on obtient évidemment, pour les points A, B, C, D et E dans cet ordre, A_μ, A_μ, A_μ, D_μ et E_μ, les trois quantités restantes étant connues par hypothèse.

Ce qui vient d'être dit suppose donc que les informations relatives à la dépendance entre les sommets initiaux et la valeur des champs numériques des sommets auto-dépendants soient explicitement connues.

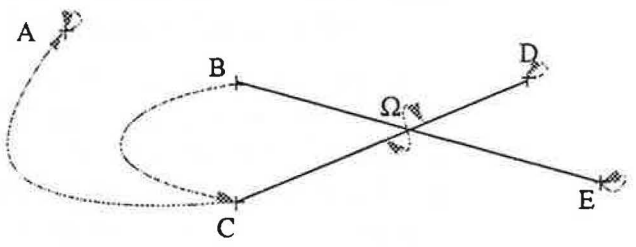


Figure V.9. Relations de dépendance dans un pré-graphe.

Dépendance par priorité

Un système de flèches accompagne, sur la figure précédente, le point d'intersection des segments [C,D] et [B,E]. Ce type de flèches n'est jamais explicite dans le pré-graphe et doit être déduit en fonction des relations d'ordre existant entre les objets qu'il contient.

Pour reprendre l'exemple *f*-V.9, si la relation d'ordre '$<_O$' sur les objets du pré-graphe permet d'affirmer que [C,D] $<_O$ [B,E], alors le champ numérique du point d'intersection Ω des deux segments s'exprime en fonction de ceux des extrémités C et D. Cette fonction, symbolisée par le système de flèches accompagnant Ω sur la figure, notée $\Omega \leftarrow \pi(C,D)$, peut être une interpolation linéaire ou prendre toute forme désirable pour la métrique du problème.

Résoudre un système de dépendances consiste à utiliser l'ensemble des relations entre sommets initiaux et la relation d'ordre définie sur les segments pour déterminer de manière non ambiguë la valeur du champ numérique de chaque sommet du graphe planaire résultant.

Tout polygone, chaîne polygonale ou ensemble de point est composé de sommets portant tous le même champ vectoriel. Il est donc possible d'affirmer que les objets O_1 et O_2 sont tels que $O_1 <_O O_2$ en comparant la relation d'ordre existant entre tout couple de sommets des deux objets. Cette relation d'ordre reflète des relations plus subtiles de priorité entre les objets sous-jacents aux éléments purement géométriques du pré-graphe. Le lecteur est renvoyé au chapitre I pour retrouver des descriptions moins formelles de ces propriétés. Un point fondamental, malgré tout, est que tout objet défini doit avoir un vecteur de priorité différent de tous les autres, afin de permettre la résolution des dépendances en toute circonstance.

Arbre de dépendance

Quand on détecte l'intersection normale Ω de deux segments [A,B] et [C,D], on doit déterminer les deux liens de dépendance à attribuer au point de rencontre. Si les relations sur les points sont $A \leftarrow P_A$, $B \leftarrow P_B$, $C \leftarrow P_C$, $D \leftarrow P_D$ et [A,B] $<_O$ [C,D], il faut s'assurer que le point d'intersection Ω vérifie $\Omega_\mu \leftarrow \pi(A,B)$ et attribuer le plus grand des deux vecteurs de priorité à Ω en cas d'intersection normale.

L'évaluation du champ numérique de Ω ne peut pas se faire tant que toutes les intersections du pré-graphe n'ont pas été détectées: si l'on rajoute à la figure *f*-V.9 un nouveau segment plus prioritaire que [C,D], les liens de dépendance deviennent ceux de la figure V.10. La valeur du champ numérique de Ω s'exprime par $\Omega_\mu = \pi(C,\Omega')$, avec $\Omega'_\mu = \pi(F,\Omega'')$, $\Omega''_\mu = \pi(\Omega',G)$, $F \leftrightarrow F$, $G \leftrightarrow G$.

Le point Ω' constitue un changement de direction de dépendance pour son voisin Ω, et le couple Ω', Ω'' forment une cascade de dépendances: Ω' et Ω'' dépendent tous deux de F et G seulement. Malgré tout, la présence de ces deux points dans le problème est inéluctable et il faut apprendre à traiter de telles situations. Un changement de direction de dépendance coïncide avec la rencontre, le long d'un segment, d'une intersection avec un segment plus prioritaire.

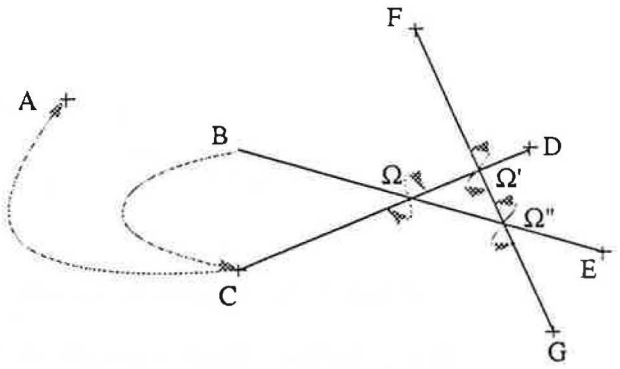


Figure V.10. Cascade de dépendances.

Chaque nœud de l'arbre binaire équilibré obtenu après l'exécution du module d'intersection comporte, en plus des informations déjà mentionnées, deux champs *lg* et *ld* (pour liens gauche et droit) éventuellement vides, qui vont permettre de modéliser parfaitement la situation décrite précédemment. Ces champs sont supposés être ordonnés et remplis selon la logique suivante: si un sommet ne dépend d'aucun autre, ses champs de dépendance sont vides; si un sommet dépend uniquement d'un autre sommet, son champ *lg* porte l'adresse, dans l'arbre, du sommet de dépendance et l'autre champ, *ld*, est vide. Si un sommet dépend de deux autres sommets, on met dans *lg* l'adresse du nœud d'arbre correspondant au sommet de dépendance le plus à gauche lexicographiquement des deux, et l'autre adresse dans le champ *ld*.

On dira qu'un sommet est *réduit* si ses deux champs de dépendance sont vides. Si, à un moment quelconque, on s'aperçoit que l'un des deux champs de dépendance d'un sommet est réduit, mais pas l'autre, on met systématiquement le champ réduit dans *lg*. Ceci permet une cohérence dans la remontée de l'algorithme de résolution des liens de dépendance. Une représentation de l'état général de l'arbre binaire correspondant à *f*-V.10 est donnée à la figure V.11.

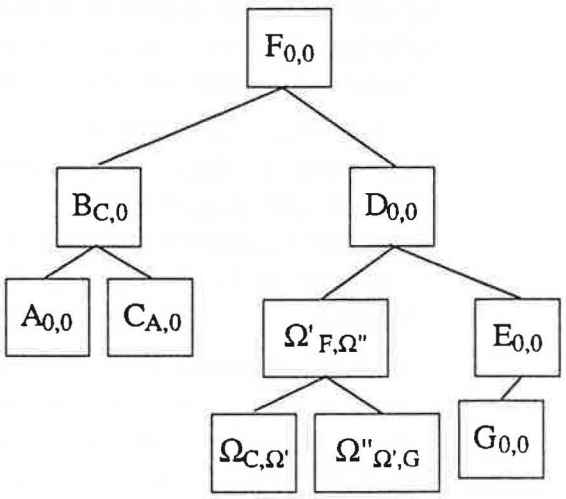


Figure V.11. Relations de dépendance dans l'arbre binaire.

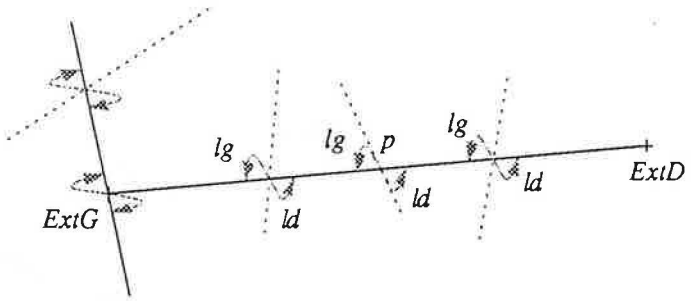


Figure V.12. Descente de cascade.

On va réduire tous les nœuds de l'arbre en procédant à un parcours infixé récursif. Soit v un nœud quelconque de l'arbre binaire des sommets du graphe planaire. Ce nœud peut être réduit (les deux liens de dépendances sont à NIL): dans ce cas, on passe au nœud suivant dans le parcours infixé. Sinon, au moins un des deux liens gauche et droit de ce nœud est différent de NIL. Considérons d'abord le cas où les deux liens sont différents de NIL. Par construction, ce nœud correspond à un sommet de génération strictement supérieure à zéro, issu d'une intersection. Ces deux liens sont donc des pointeurs sur les deux voisins du point concerné sur un segment de droite. Soit *vd* le point correspondant au lien droit. Si le lien gauche associé à *vd* est un pointeur sur le nœud v, on en déduit que les deux voisins appartenant au même segment *s*, sont des points d'intersection avec deux segments de priorité strictement inférieure à celle de *s*: il s'agit d'une partie de ce que l'on a précédemment appelé une cascade.

Par construction, une cascade est caractérisée par une suite de sommets à nœuds non réduits et de même vecteur de priorité: une telle suite de segments peut se terminer, aux deux extrémités, de deux façons: le dernier sommet rencontré correspond à un nœud réduit, auquel cas la cascade se termine par une extrémité dont le champ numérique est déterminé, ou bien il est non réduit et possède un vecteur de priorité supérieur à celui des points de la cascade. Dans ce dernier cas, le sommet de fin peut être simplement lié à un sommet isolé; cela signifie qu'une contrainte naturelle est que tout sommet lié à un sommet isolé doit être moins prioritaire que ce dernier (*loi des priorités*), faute de quoi la modélisation des dépendances risque d'être incohérente.

Dans le cas où le nœud considéré a deux liens différents de NIL, l'idée, pour le réduire, est de parcourir l'éventuelle cascade dans laquelle il se trouve jusqu'à rencontrer une extrémité gauche et une extrémité droite: si on s'arrête sur un changement de direction, il suffit de relancer récursivement l'algorithme de réduction sur le nouveau nœud atteint. L'algorithme correspondant a une partie récursive et une partie itérative.

Comme la réduction d'un nœud s'accompagne de l'évaluation du champ numérique du sommet correspondant, on attend en retour de l'algorithme de résolution l'adresse de deux nœuds afin de permettre l'application d'une fonction sur ces deux variables. On adoptera la notation générique Z(*a*,*b*,*c*) pour dire que le champ numérique du nœud *c* s'obtient par un calcul faisant intervenir ceux des nœuds *a* et *b*.

Quand on s'aperçoit que le nœud à réduire fait partie d'une cascade, on peut utiliser la propriété très importante suivante: tout nœud d'une cascade, sauf les deux extrêmes, possède un lien en direction du nœud à réduire, et ce dans les deux demi-cascades de part et d'autre de tout point intermédiaire.

Sur la figure V.12, on choisit de considérer une descente de la cascade débutant en p vers la gauche. Tout lien *lg* de cette demi-cascade est dirigé vers l'extrémité gauche de celle-ci et tout lien *ld* est dirigé vers le point de départ.

Autrement dit, pour réduire la demi-cascade gauche, il suffit de suivre les liens *lg* jusqu'à rencontrer l'extrémité recherchée; si celle-ci correspond à un nœud non réduit, on lance récursivement la réduction de celui-ci; au retour de la récursion, la procédure de réduction fournit les adresses de deux nœuds *a* et *b* permettant l'évaluation du champ numérique de l'extrémité gauche. Dans tous les cas donc, l'extrémité gauche peut être considérée comme réduite.

Une technique similaire permet de réduire l'extrémité finale de la demi-cascade située à droite du point initial. Appelons ExtG et ExtD les deux nœuds extrêmes réduits découverts dans les deux processus précédents. Le champ numérique de tout point de la cascade d'origine *p* doit évidemment être évalué à partir des nœuds ExtG et ExtD.

Comme le lien droit de tous les nœuds rencontrés dans la demi-cascade gauche et le lien gauche de tous les nœuds rencontrés dans la demi-cascade droite sont dirigés vers *p*, il est facile de réduire chacun de ces nœuds en remontant de chaque côté jusqu'à *p*. La fonction de réduction peut alors 'retourner' les nœuds ExtG et ExtD et la procédure appelante se charger de la réduction définitive du nœud *p* de départ.

La convention de remplissage des liens en respect de l'ordre lexicographique permet de s'assurer que les liens gauche et droit ont toujours une orientation cohérente le long d'une cascade, et en particulier que l'on rencontre forcément une alternance lien gauche, lien droit, quelle que soit la direction propre du support de cascade dans le plan.

Si l'on revient maintenant sur le cas d'un nœud présentant un lien à NIL et seulement un, on sait qu'on a décidé de toujours choisir *lg* pour contenir ce lien. La chaîne de lien qui part de ce nœud doit aussi respecter la loi des priorités (jamais de lien vers un élément moins prioritaire).

Le principe de réduction de ce type de nœud est semblable à la technique précédente, à ceci près que l'extrémité droite retournée étant parfois nulle, et même parfois l'extrémité gauche aussi, la fonction Z doit détecter ces cas.

On peut résumer l'algorithme général de réduction de l'arbre binaire de points. On utilise deux fonctions, RéduireArbre et RéduireNœud, dont la première provoque un parcours infixé de l'arbre et appelle récursivement la seconde. La notation '$a \sim b$' signifie que les nœuds *a* et *b* ont le même vecteur de priorité.

```
RéduireArbre(r)
    si (r ≠ NIL)
        RéduireArbre(r→filsgauche);
        si (r est non-réduit)
            (a,b) ← RéduireNœud(r);
            Z(a,b,r);
            r→lg← NIL;
            r→ld ← NIL;                                  ¡réduction confirmée¡
        RéduireArbre(r→filsdroit);
RéduireNœud(p);
    gauche ← p; suivantgauche ← p → lg;
    droit ← p; suivantdroit ← p → ld;
    si (suivantdroit = NIL)                              ¡un seul lien au plus¡
        si (suivantgauche = NIL) retourner(NIL,NIL);     ¡p est réduit¡
        si (suivantgauche non réduit)
            (a,b) ← RéduireNœud(suivantgauche);
            Z(a,b,p);                                    ¡b éventuellement NIL¡
            suivantgauche→lg← NIL;
            suivantgauche→ld ← NIL;                      ¡réduction confirmée¡
            retourner (a,b);
        Z(suivantgauche,NIL,p);          ¡p ne dépend que de suivantgauche¡
        retourner(suivantgauche,NIL);
    DernierG, DernierD ← p;              ¡à partir d'ici il y a forcément 2 liens¡
    tant que (suivantgauche non réduit)  ¡traitement de la partie gauche¡
        si (gauche ~ p)                                  ¡cascade¡
            DernierG ← gauche;
            gauche ← suivantgauche;
            suivantgauche ← suivantgauche→lg;
        sinon                                ¡pas de cascade, récursion¡
            (a,b) ← RéduireNœud(gauche);
            Z(a,b,gauche);
            gauche→lg← NIL;
            gauche→ld ← NIL;                             ¡réduction confirmée¡
        ExtG ← gauche;                   'fin de cascade ou cas 'normal'¡
    tant que (suivantdroit non réduit)   ¡traitement de la partie droite¡
        si (droit ~ p)                                   ¡cascade¡
            DernierD ← droit;
            droit ← suivantdroit;
            suivantdroit ← suivantdroit→ld;
        sinon                                ¡pas de cascade, récursion¡
            (a,b) ← RéduireNœud(droit);
            Z(a,b,droit);
            droit→lg← NIL;
            droit→ld ← NIL;                              ¡réduction confirmée¡
        ExtD ← droit;                    'fin de cascade ou cas 'normal'¡
    tant que (DernierG ≠ p)              ¡remontée de la demi-cascade gauche¡
        gauche← DernierG;
        DernierG ← DernierG → ld;                        ¡remonter vers la droite¡
        Z(ExtG,ExtD,gauche);
        gauche→lg← NIL;
        gauche→ld ← NIL;                                 ¡réduction confirmée¡
    tant que (DernierD ≠ p)              ¡remontée de la demi-cascade droite¡
        droit← DernierD;
        DernierD ← DernierD → ld;                        ¡remonter vers la droite¡
        Z(ExtG,ExtD,droit);
        droit→lg← NIL;
        droit→ld ← NIL;                                  ¡réduction confirmée¡
    retourner(ExtG,ExtD);
```

Algorithme V-III: Résolution de dépendances dans un arbre binaire.

Il est important de s'assurer que le pré-graphe ne contienne pas intrinsèquement un circuit dû à une incohérence ou une violation des règles émises précédemment.

Il est clair que la partie itérative (toutes les boucles **tant que**) de cet algorithme ne peut engendrer de boucle infinie: une cascade est avant tout une suite de sommets sur un même segment.

Si une boucle infinie apparaît, c'est dans un appel récursif sur une cascade dont les extrémités sont inscrites dans un circuit. Le moyen le plus simple pour éviter les boucles infinies est d'utiliser un drapeau booléen d'état, initialement positionné à 'non examiné' pour tous les nœuds lorsqu'ils sont créés. Dès qu'un nœud est passé en paramètre à la fonction RéduireNœud, son drapeau d'état est mis à 'examiné'.

De cette façon, lorsqu'un nœud déjà examiné est rencontré une seconde fois dans la procédure, on peut arrêter l'exécution et signaler une erreur de cohérence.

Il est maintenant nécessaire de prouver que la fonction RéduireNœud ne traite jamais plus d'une fois un nœud donné, d'autant plus que la procédure RéduireArbre a évidemment une complexité linéaire si l'on prouve que la fonction RéduireArbre, appelée par elle, traite un nombre fini de fois chaque nœud de l'arbre.

Sachant qu'on a évité les boucles liées à des vices de forme des données, il est clair que cette propriété est vérifiée si tout nœud d'une cascade est réduit définitivement après qu'il a été explicitement manipulé par la fonction RéduireNœud, et surtout ne peut pas être rappelé en paramètre dans cette fonction *à cause* d'une autre cascade!

Dès qu'un nœud est réduit il ne peut plus être passé en paramètre à la fonction de réduction de nœuds *par* la procédure RéduireArbre, de par la manière dont elle est construite (voir algorithme V.iii).

On peut donc affirmer que, lorsqu'une cascade est découverte, celle-ci est traitée en entier et ses nœuds deviennent des éléments qui peuvent être utilisés comme extrémités d'autres cascades, dans le pire des cas.

En particulier, si un nœud est intermédiaire dans une cascade traitée et se trouve aussi inclus dans une cascade transversale (de support intersectant celui de la première au point matérialisé par ce nœud), il ne peut être qu'un nœud de changement de direction pour la deuxième cascade, puisque deux objets ne peuvent avoir la même priorité et que les circuits sont éliminés.

Ainsi, ce nœud marque une extrémité de cascade et est réduit puisque la première cascade l'a été; ses liens sont à NIL et il peut être *directement* utilisé en fin de partie itérative pour la deuxième cascade, son champ numérique ayant été évalué, sans que la fonction RéduireNœud ne soit plus appelée pour ce bout de cascade.

Comme chaque nœud ne peut posséder, par construction, que deux liens au maximum, et ne peut appartenir, par définition, à plus d'une cascade, la fonction RéduireNœud traite chaque nœud, quel qu'il soit, *au plus* une fois.

V.B. LE POINT DE VUE DES FACES

Le graphe planaire obtenu à la sortie des algorithmes de la section précédente n'est pas satisfaisant à tout point de vue. Deux problèmes fondamentaux se présentent, que cette section est chargée de résoudre.

V.B.1 Décomposition de faces (clipping)

En premier, puisque les données proviennent de sources *a priori* disjointes, il est nécessaire d'imposer des contraintes naturelles comme, par exemple, l'interdiction pour certaines classes d'objets d'intersecter d'autres classes. La procédure d'intersection de segments se révèle impuissante, à elle seule, pour apporter une réponse satisfaisante à cette nouvelle contrainte.

Prenons l'exemple de la figure V.13. La frontière du polygone P rencontre celle de Q en un point unique, ainsi que celle de R. Mais, la différence fondamentale est que Q, en tant qu'ensemble, est inclus entièrement dans P, alors que R ne l'est pas.

Il est d'ailleurs nécessaire de clarifier un point de terminologie qui sera utile par la suite. Le terme *polygone* désigne en fait l'ensemble union de son *intérieur* et de sa *frontière*. Si Fr(P) et $\mathring{P}$ désignent la frontière et l'intérieur du polygone P, on peut donc écrire $P = Fr(P) \cup \mathring{P}$. Dans le cas présent, on cherche à savoir si $\mathring{P} \cap \mathring{Q}$ est vide ou non. L'interdiction d'intersecter ensembliste est souvent placée au niveau des intérieurs pour permettre les situations limites où deux objets ont éventuellement une section de frontière commune, ce qui paraît la moindre des choses.

La procédure d'intersection de segments éprouve quelques difficultés à statuer sur ce problème justement car elle ne s'applique qu'à des segments et non à des faces. Les choses se corsent si les points d'intersection sont effectivement inclus dans la liste initiale des sommets des frontières polygonales; par exemple si, en *f*-V.13, le sommet du triangle Q qui touche la frontière de P est un sommet explicite de celle-ci, la procédure ne détecte aucune intersection entre segments, puisque dans une telle situation, tous les segments testés ne se coupent au pire qu'en une extrémité, condition communément acceptée pour déclarer deux segments adjacents en un sommet et donc non sécants.

En fait, le problème le plus grave est que la procédure d'intersection s'avère totalement incapable de résoudre le problème suivant, plus pernicieux: dans le cas de figure V.14, les deux polygones sont supposés avoir le droit de se couper, à condition que tout point de leur union appartienne à un polygone et un seulement. Pour ce faire, on a recours au vecteur de priorité décrit plus haut, qui permet de décider, par exemple, que le polygone blanc est le plus 'prioritaire' des deux.

La zone hachurée correspond à l'intersection des deux polygones, et la procédure d'intersection ne peut en aucun cas décider de sa forme, ni de l'ensemble original auquel appartient chacun de ses points de manière privilégiée.

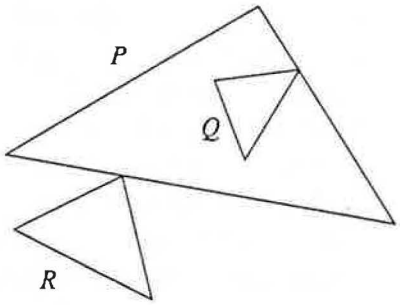


Figure V.13. Intersection vide ou non?

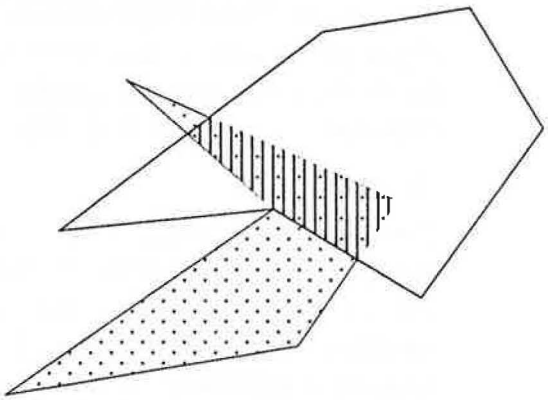


Figure V.14. A quel polygone appartient la face hachurée?

Plus grave encore, il est impossible de caractériser la partie 'utile' du polygone moins prioritaire, c'est-à-dire le complémentaire ensembliste, dans ce premier, de la région hachurée, partie utile dont on voit sur le graphique qu'elle comporte deux composantes connexes. Les opérations impliquées dans ces deux problèmes portent le nom générique de *clipping* en anglais, qui sera traduit ici par 'décomposition'.

Soient A et B deux ensembles de l'univers $\boldsymbol{U}$ liés par une relation d'ordre *stricte*, $>_P$; la *décomposition ordonnée* de A et B, notée $\div_P(A,B)$, est l'opération de $\boldsymbol{U}*\boldsymbol{U}$ vers $\boldsymbol{U}$ définie par

si $(A \cap B = \emptyset)$ $\div_P(A,B) \leftarrow (A, B)$;
sinon
  si $(A >_P B))$ $\div_P(A,B) \leftarrow (A, B\text{-}A)$;
  sinon
    si $(B >_P A)$ $\div_P(A,B) \leftarrow (A\text{-}B, B)$;
    sinon $\div_P(A,B)$ est une opération indéterminée;

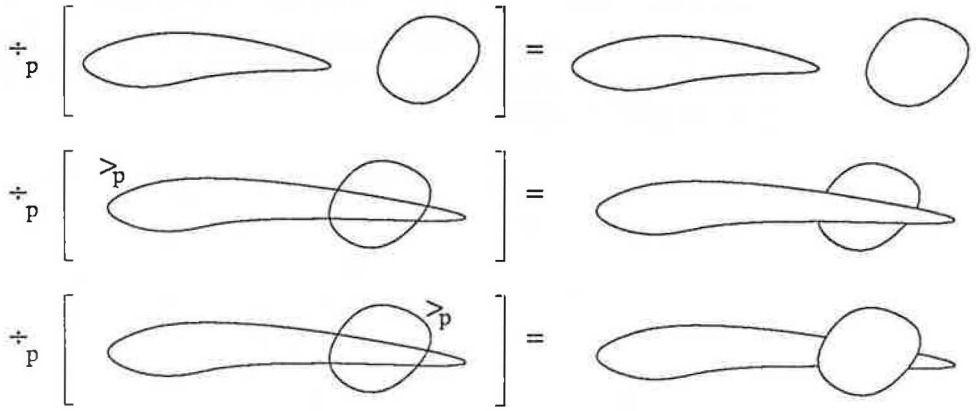


Figure V.15. Décomposition ensembliste ordonnée.

La figure V.15 illustre cette définition (le symbole $>_P$ en regard d'un ensemble signifie que celui-ci est plus prioritaire que l'autre). Clairement, l'opération de décomposition est indéterminée si l'intersection des deux ensembles est non-vide et si ceux-ci ont le même vecteur de priorité: il est impossible, dans ces conditions, d'associer à tout point de $A \cap B$, un ensemble unique d'appartenance. La définition précédente généralise la notion de *différence ensembliste* mais il est important de remarquer que le résultat de cette opération de décomposition ordonnée est un couple ordonné de sous-ensembles des ensembles originaux.

Remarque: Dans l'opération de décomposition ordonnée, seuls les objets polygonaux sont concernés: les sommets isolés ou les segments de chaînes ouvertes du graphe sont totalement neutres dans cette opération et ne seront jamais mentionnés.

Modélisation de la multiplicité

Considérons un polygone P_n à n sommets du plan euclidien $\boldsymbol{P}$. En tant que tel, ce polygone provient vraisemblablement d'une liste de sommets sous la forme $\{p_1, p_2, \ldots, p_n\}$: implicitement, la *n-ième* arête de sa frontière relie le dernier point listé au premier. On parle alors de relations d'adjacence 'modulo n'.

Puisque l'on peut naturellement parcourir la frontière dans le sens $p_1 \rightarrow p_n$, ou dans le sens $p_n \rightarrow p_1$, on dispose de deux orientations de celle-ci: on dit que l'orientation de la frontière telle que l'intérieur du polygone se trouve toujours à gauche du sens de déplacement est l'orientation *directe* ou *trigonométrique*. C'est celle-ci qui sera toujours employée ici. L'autre orientation est dite *rétrograde*.

Quand deux polygones partagent une arête, on doit pouvoir connaître, à tout moment, à la fois les *attributs* géométriques de l'arête commune (pente, extrémités) et les représentants qui l''occupent' (identificateurs des polygones partageant cette arête). Le mot représentant n'est pas choisi au hasard: on peut définir une relation d'équivalence sur l'ensemble des arêtes du graphe planaire par:

*a*R*b* **si et seulement si** (*a* **et** *b* ont les mêmes attributs géométriques)

Cette relation d'équivalence définit des classes d'équivalences et dans chacune de celle-ci il est possible d'isoler un représentant *canonique* ; son choix est laissé à l'appréciation de chacun, mais on peut imaginer que la première arête présentant les attributs d'une classe fait assez bien l'affaire, si l'on imagine que l'on a classé les polygones par ordre de priorité, par exemple.

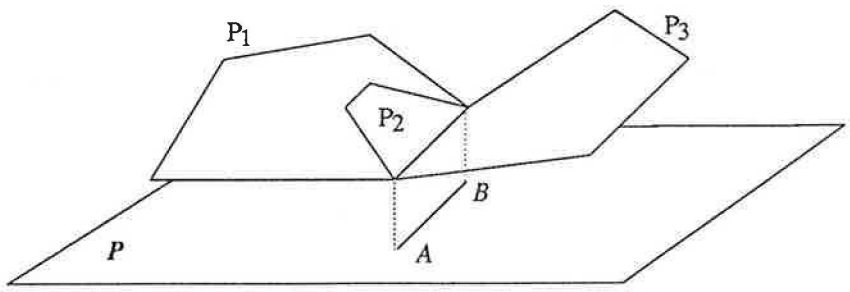


Figure V.16. Arête canonique et arêtes polygonales de même classe.

En fait, on a tout intérêt à dissocier les arêtes polygonales de cette notion d'arête canonique. Par exemple, on peut imaginer, à partir de la figure V.16, que le segment [A,B], projection confondue des trois arêtes polygonales sur le plan dans lequel est plongée la représentation par segments du graphe planaire, est ce représentant canonique, c'est-à-dire une entité dissociée permettant de retrouver, par une structuration adéquate, les informations géométriques et identificatrices relatives aux trois arêtes polygonales qui se projettent sur elle.

Du point de vue structurel, il reste à examiner un élément important, qui concerne le tri de tous les éléments du pré-graphe. Puisque les objets qu'il contient s'accompagnent d'informations géométriques (coordonnées) et logiques (classes d'objets, rang dans une classe, priorités ou dépendances), il est nécessaire de considérer plusieurs ordres entre les arêtes[2] de cet ensemble.

Plus précisément, chaque objet

- appartient à une classe et une seule,
- admet un rang dans cette classe et est le seul à y prétendre,
- est constitué d'arêtes dont on sait qu'elles sont données dans un ordre particulier (dit *naturel*), et dont on peut voir qu'il est possible de les trier, selon l'ordre lexicographique, à l'intérieur d'une même classe aussi bien qu'au sein de l'ensemble général de toutes les arêtes, toutes catégories confondues.

En dernier, on a vu au chapitre sur la précision arbitraire, que les arêtes canoniques sont aussi ordonnées selon l'angle polaire (relatif à la base du plan de projection) autour d'une de leurs extrémités.

Pour organiser tout ceci de la manière la plus souple possible, il est nécessaire de prévoir que chaque procédure ou fonction manipulant des arêtes devra éventuellement reconnaître certains attributs de celles-ci pour savoir comment agir et parfois pour distinguer différents types de comportement à différents moments!

Cette propriété sera d'une extrême importance puisque c'est elle qui permet, de fait, l'existence d'une 'sémantique' des objets: il sera possible, puisque l'on travaille sur des classes d'ensembles, de définir des comportements, à l'intérieur de fonctions génériques, qui seront directement dépendants des circonstances: par exemple, la priorité relative de deux objets peut varier dans le temps, suivant la signification qu'ils revêtent à deux moments différents – l'un peut être plus prioritaire que l'autre au moment de l'intersection (on ne doit pas sortir du domaine) et moins lors de la décomposition (le domaine cède sa frontière à qui le désire).

Mais ceci nécessite de structurer le plus souplement possible toutes les données afin de se réserver un maximum de latitude.

---

[2] Un choix tranché est fait concernant les sommets isolés: ce sont des cas limites de segments dont les extrémités sont confondues. Ceci permet évidemment de traiter indifféremment ces deux types d'éléments et donc d'unifier les algorithmes qui les manipulent.

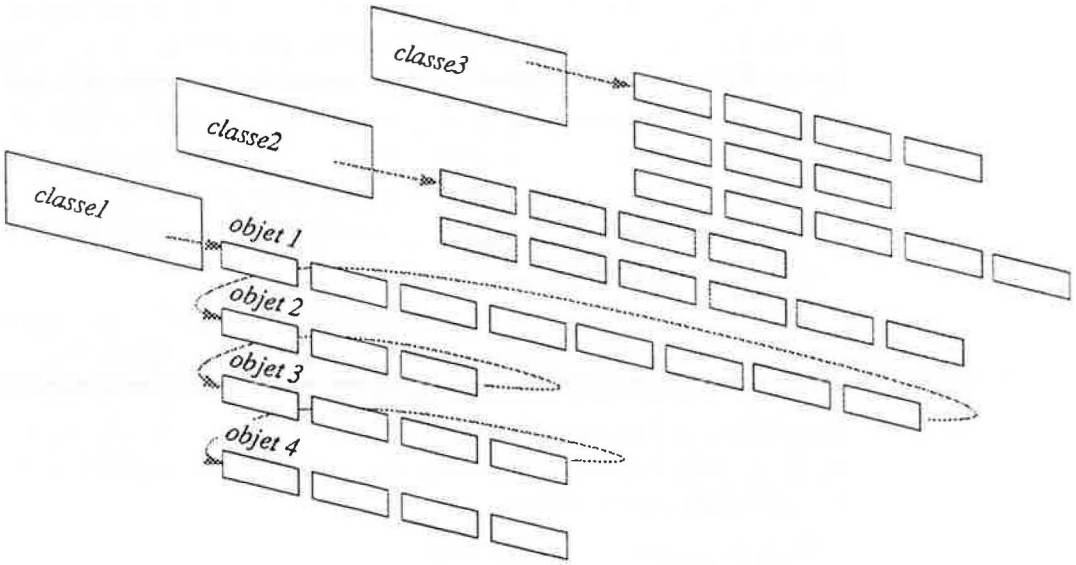


Figure V.17. Classes et listes d'objets.

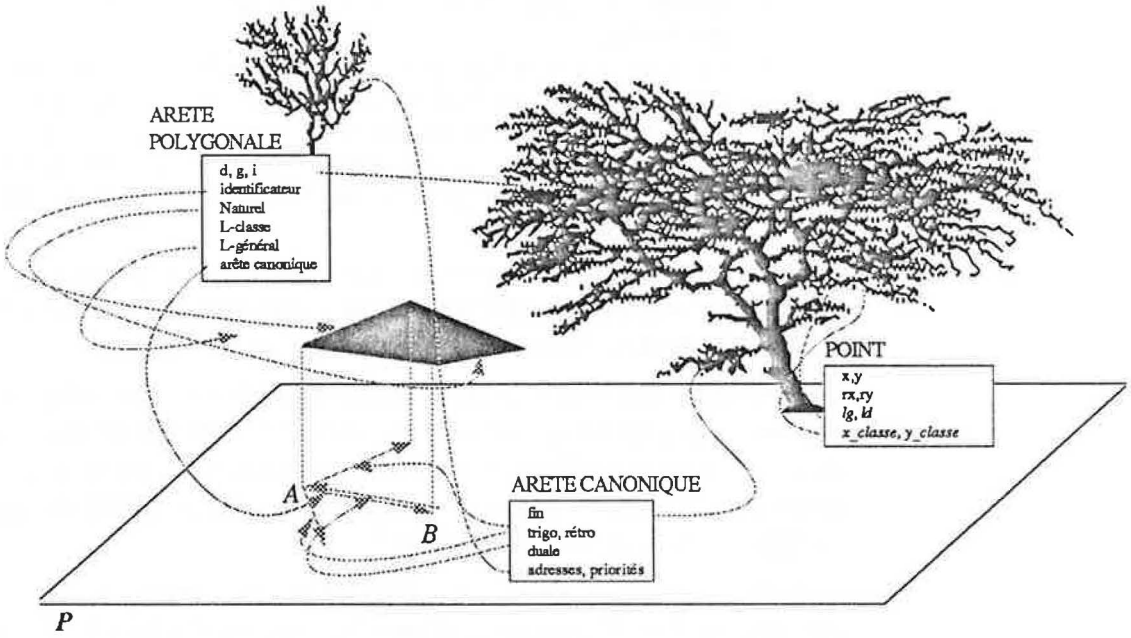


Figure V.18. Arêtes polygonales et arêtes canoniques.

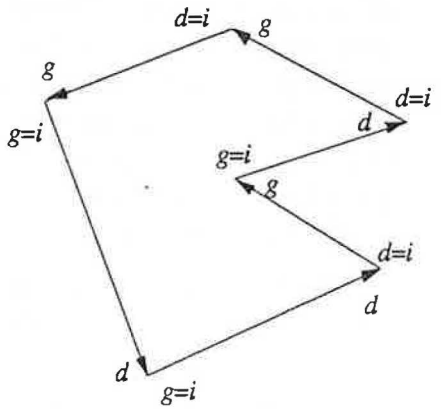


Figure V.19. Les champs *d*, *g* et *i* et l'orientation des arêtes.

La figure V.17 transcrit une première partie de l'architecture nécessaire: un vecteur de classe d'objets permet d'accéder à autant de débuts de listes d'objet; chaque liste chaînée est un ensemble structuré d'arêtes polygonales; d'une manière qui sera expliquée juste après, le dernier élément d'une liste permet d'accéder au premier élément de l'objet suivant de la même classe; le dernier élément du dernier objet d'une classe n'a pas de successeur. Les objets sont triés par ordre croissant de rang dans une même classe, et aucun n'a le même rang qu'un autre. Chaque classe a une priorité propre et les classes sont organisées dans un tableau par priorité croissante, qui donne accès au premier élément du premier objet de chaque classe.

Pour comprendre comment le chaînage est structuré, considérons maintenant la figure V.18. On y trouve un polygone de l'espace (en noir) dont une arête sert de cobaye; cette arête est projetée sur le plan du graphe pour donner une arête canonique [A,B].

La structure d'*arête polygonale présente* trois champs, *g*, *d* et *i*, qui pointent sur des nœuds de l'arbre binaire général. Les deux premiers indiquent respectivement l'extrémité la plus à gauche lexicographiquement de l'arête (son origine), le deuxième désigne son extrémité droite (sa fin) et le dernier champ permet de tester l'orientation d'une arête. Si *s* est un segment quelconque non dégénéré, et si l'on considère le parcours direct et donc orienté de la frontière polygonale (ou parcours naturel de la chaîne) à laquelle appartient l'arête *s*, le segment *s* possède une extrémité *initiale*, représentée par le champ *i*. Les deux cas d'orientation $i = g$ ou $i = d$ peuvent se présenter et il suffit de disposer de ces trois champs explicitement dans la structure d'arête pour déterminer l'orientation sans ambiguïté. La figure V.19 illustre quelques situations typiques.

La structure d'arête polygonale possède ensuite un champ *identificateur* pointant sur l'*en-tête* de liste, c'est-à-dire la première arête de la liste des arêtes constituant un objet. Cet en-tête est le même pour toutes les arêtes d'un même objet.

Le lien de chaînage *Naturel* permet de parcourir la frontière de l'objet (ou la suite des arêtes d'une chaîne ouverte) de proche en proche. Sur *f*-V.17, le dernier lien *Naturel* du premier objet de la première classe pointe sur le premier élément du deuxième objet: ceci est vrai de toutes les listes. Ainsi, puisque l'en-tête de chaque arête d'un objet est le premier élément de la liste de celui-ci, chaque élément de la liste des arêtes d'un objet a le même en-tête et l'on sait qu'on parvient à la fin d'une liste lorsque le lien *Naturel* est NIL (fin d'une classe) ou pointe sur un élément d'en-tête différent.

Chaque arête possède ensuite un chaînage *L-classe* qui permet de trier toutes les arêtes d'une même classe selon l'ordre lexicographique. Ce chaînage est extrêmement utile quand on désire exécuter une opération généraliste (comme le test d'intersection) sur un sous-ensemble de classes seulement. Le chaînage suivant, relatif au champ *L-général*, permet de trier lexicographiquement toutes les arêtes, toutes catégories et priorités confondues.

On peut déduire l'ordre *L-classe* de l'ordre *L-général*, en une passe sur toutes les arêtes, une fois le tri général effectué. Pour permettre des performances idéales, tous ces chaînages doivent en fait aller dans les deux sens.

En dernier lieu, la structure d'arête polygonale contient un pointeur vers l'arête canonique représentant toute arête polygonale.

En ce qui concerne la structure d'*arête canonique*, on voit qu'elle contient un champ, *fin*, qui pointe sur le nœud d'arbre binaire général indiquant son extrémité finale (ce champ n'est pas indispensable, puisque l'on pourrait obtenir l'information par le biais des arêtes polygonales représentées par l'arête canonique, mais il permet de diminuer les temps d'accès).

Le champ *trigo* permet d'accéder à la prochaine arête canonique autour de l'origine de l'arête canonique courante, dans le sens trigonométrique. Le sens de parcours rétrograde est aussi accessible, par le champ *rétro*.

Le champ *duale* permet de passer de l'arête canonique [A,B] incidente en A à l'arête canonique duale, [B,A], incidente en B. En combinant les deux champs précédents, on peut, en temps constant, connaître l'arête canonique qui suit [B,A], autour de B, dans le sens des aiguilles d'une montre, opération qui sera fort utile par la suite.

Un dernier champ, *arêtes*, est composé de deux sous-champs, *adresses* et *priorités*, symbolisées par un arbre de plus petite taille que le premier: cet arbre est la structure équilibrée permettant de stocker les références de toutes les arêtes polygonales représentées par l'arête canonique.

Cette structure mérite un peu d'attention. Il est clair que si plusieurs polygones partagent la même arête polygonale *ab*, ces arêtes confondues vont avoir le seul représentant canonique [*a*,*b*]. Partant de ce dernier, on peut être amené à se poser plusieurs types de questions: quelle est la priorité maximale parmi les objets partageant cette arête canonique, est-ce que cette arête canonique représente une arête polygonale de l'objet *o*, ...?

Pour répondre à ce genre de questions, il est pratique de diviser en deux le champ *arêtes*, donc d'utiliser deux structures, *adresses* et *priorités*: la première porte sur l'ordre des adresses des arêtes polygonales et permet de répondre à la première question; la seconde porte sur l'ordre de priorité des arêtes et permet de répondre à la seconde question.

Si l'on est sûr que le nombre moyen de polygones partageant des arêtes est très faible, on peut éventuellement utiliser des structures de listes chaînées; sinon on se sert d'arbres équilibrés.

Technique de décomposition

La décomposition de deux polygones simples[3] quelconques, en l'absence de priorités, a été étudiée en détail dans [A/W 77]. On trouve un survol complet de ce problème dans [ROG 85]. L'algorithme proposé ici est une généralisation de celui d'Atherton-Weiler, lorsqu'on considère un ensemble quelconque de polygones simples totalement ordonnés par une relation de priorité logique $>_P$. Dans toute la suite, deux polygones ne pourront avoir le même vecteur de priorité.

Remarques: la solution proposée dans cette sous-section ne traitera pas explicitement les polygones à trous. Il n'en reste pas moins que cette méthode pourra être adaptée aisément à ces situations. De même, dans les raisonnements qui suivent, les polygones sont supposés n'avoir qu'une composante connexe. Ceci n'est pas une contrainte nécessaire, mais seulement utile pour rendre la présentation plus concise. La généralisation à des polygones simples à plusieurs composantes connexes est quasiment immédiate.

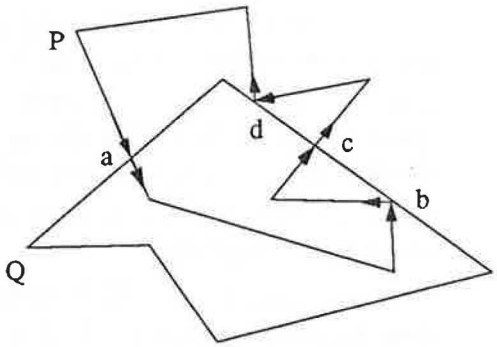


Figure V.20. Intersection et sens de parcours.

Avant de rentrer dans les détails de la solution, il est nécessaire de remarquer un phénomène important: lorsque les frontières de deux polygones s'intersectent, elles peuvent le faire de quatre façons distinctes. Suivons la frontière de Q dans le sens direct jusqu'à un point d'intersection avec le polygone P, sur la figure V.20: P peut pénétrer à l'intérieur de Q (cas a), ressortir de l'intérieur de Q (cas c) ou bien rester à l'intérieur (cas b) ou à l'extérieur (cas d).

Soient Ω un point d'intersection quelconque entre P et Q, q_1 et q_2 les deux arêtes de Fr(Q) partageant Ω, q_2 suivant q_1 dans l'ordre de parcours direct. Le champ origine de q_2 coïncide avec Ω, alors que pour q_1, ce champ est donné par l'extrémité opposée.

Si l'on imagine un 'petit bonhomme d'Ampère' se tenant sur la frontière de Q, en Ω, regardant l'arête q_2 dans le sens de parcours direct, ce témoin peut considérer qu'il observe deux arêtes canoniques incidentes en Ω: il a donc deux moyens de passer de l'une à l'autre, consistant à énumérer toutes les arêtes de P qu'il rencontre dans le sens trigonométrique autour de Ω pour l'un, et dans le sens rétrograde pour l'autre.

[3] Un polygone simple est un polygone dont toute arête frontalière ne peut en couper une autre qu'en une de ses extrémités. Tout polygone sera implicitement considéré simple.

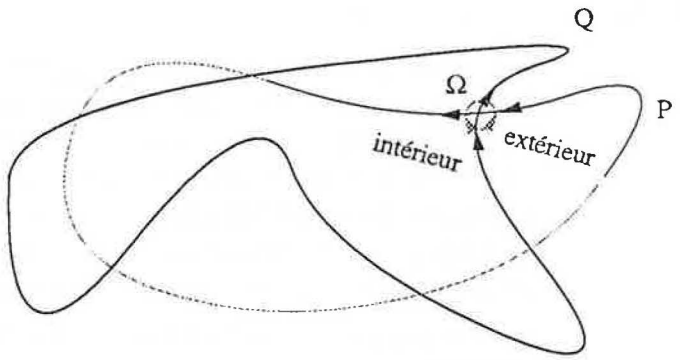


Figure V.21. Intérieur et extérieur.

De son point de vue, une arête de P incidente en Ω est *rentrante* si elle a un champ origine en Ω, et sortante sinon. De plus, une arête de Fr(P) est à l'intérieur de Q si elle est située à main gauche pour lui (sens rétrograde) et extérieure à Q si elle est située à main droite (sens direct). Consulter la figure V.21.

Lorsque P rentre à l'intérieur de Q (cas a), le bonhomme d'Ampère compte une arête rentrante de Fr(P) à main droite et une arête à main gauche. Lorsque P ressort de Q (cas c), il compte une sortante à main gauche et une arête à main droite. Lorsque P 'fait semblant' de ressortir de Q (cas b) il compte deux arêtes à main gauche et aucune à main droite, et dans la dernière situation il compte deux arêtes à main droite et aucune à main gauche.

L'idée principale de l'algorithme d'Atherton-Weiler est la suivante: pour connaître l'ensemble P-Q, on tient à jour une liste des intersections où P sort de Q, et dans les cas où cette liste n'est pas vide, il suffit de partir itérativement de chaque élément de cette liste (point de départ courant), de le retirer de la liste, et de parcourir la frontière de P jusqu'à rencontrer une intersection où P rentre dans Q (par exemple Ω dans la figure précédente); à ce stade, on 'saute' sur la frontière de Q, mais en la parcourant dans le sens rétrograde jusqu'à atteindre le point de départ courant ou bien une nouvelle intersection rentrante (qu'il faut d'ailleurs à cette occasion rayer de la liste), et l'on continue jusqu'à revenir au point de départ courant: ceci termine une composante connexe de P-Q. On recommence le processus tant que la liste des intersections sortantes n'est pas vide. La partie de P décomposée par Q (donc la partie de P commune avec Q) se trouve en utilisant une liste des intersections rentrantes et en procédant de la même façon, en sautant d'une frontière à l'autre, mais cette fois-ci, la frontière de Q étant parcourue dans le sens direct.

On voit qu'une des conditions de fonctionnement de cet algorithme est une 'table' permettant de retrouver, dans le polygone Q, l'arête adjacente à celle de Fr(P) que l'on quitte et une autre table pour le problème inverse. Ceci ne serait pas pratique dans le cas d'un grand nombre de polygones et encore moins s'ils obéissent à des lois de priorité.

On va généraliser le processus de l'algorithme d'Atherton-Weiler de façon à ce qu'il intègre toutes les conditions qui ont été imposées précédemment. Le bonhomme d'Ampère va être chargé d'un travail plus complexe qu'auparavant, pour une tâche, elle, plus réduite.

Pour le cas qui nous préoccupe, tout couple de polygones impliqués dans une décomposition est tel qu'il en existe un plus prioritaire que l'autre; cela signifie que la décomposition ne doit porter que sur la partie extérieure du polygone moins prioritaire. On peut alors parler de *décomposition d'un polygone par un autre.*

En fait, on peut très bien s'arranger, en vertu de la relation d'ordre instaurée sur l'ensemble des polygones, pour les traiter dans l'ordre décroissant de priorité: de cette manière, au moment de la décomposition d'un polygone P par un polygone Q, on sera sûr que le polygone Q aura été décomposé par tous les polygones plus prioritaires que lui; le résultat de ces opérations sera peut-être un ensemble de composantes connexes d'intérieurs disjoints et il faudra s'assurer que ces nouveaux polygones simples élémentaires auront bien été insérés dans la structuration complexe présentée plus haut.

Si l'on a ordonné les polygones par ordre décroissant de priorité, le premier polygone ne peut être décomposé par aucun autre et vérifie les hypothèses énoncées précédemment.

Afin de présenter l'algorithme recherché, exploitons l'exemple assez simple de la figure V.22. Le polygone Q de celle-ci est plus prioritaire que le polygone P. On commence par le traitement de Q. Le principe énoncé précédemment permet de s'affranchir de la décomposition de Q par des polygones plus prioritaires puisqu'il n'y en a pas, et de se concentrer uniquement sur la décomposition, par Q, 'des polygones moins prioritaires' que lui et pénétrant en son intérieur.

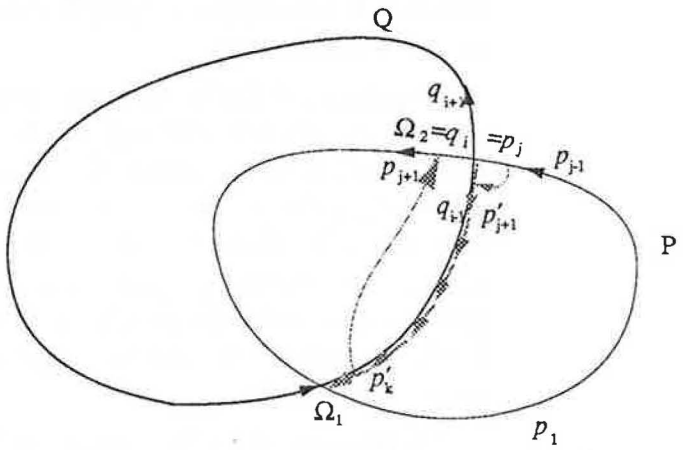


Figure V.22. Décomposition et priorité.

On commence le parcours de Fr(P) en p_1, qui marque l'origine de l'arête de départ. Il est important de noter que cette arête peut se trouver *n'importe où* sur Fr(P), ce qui nécessite certaines précautions dans l'algorithme. Soit $q_{i-1}q_i$ l'arête courante et q_iq_{i+1} l'arête suivante de Fr(Q) dans le sens de parcours naturel. (On rappelle que les notations sont à comprendre 'modulo' le nombre d'arêtes de Fr(Q)).

L'arête canonique associée à $q_{i-1}q_i$ est $[q_{i-1},q_i]$[4] et l'arête canonique duale est $[q_i,q_{i-1}]$. Soit $[q_i,Z]$ la première arête canonique rencontrée en tournant autour de q_i dans le sens des aiguilles d'une montre. Si Z est égal à q_{i+1}, le polygone Q ne subit pas d'intrusion et l'on peut avancer à l'arête suivante. Si Z est différent de q_{i+1}, il faut chercher si, parmi les arêtes polygonales que représente $[q_i,Z]$, il en existe une (de priorité inférieure, cela va sans dire...) qui soit rentrante. Si ce n'est pas le cas (Ω_1 sur la répétition de la figure V.22, en face), on passe à l'arête suivante de Fr(Q), car à cette arête sortante correspond nécessairement une arête rentrante en début de chaîne, qui sera trouvée ultérieurement. Si, en revanche, il existe une arête rentrante (Ω_2 sur la figure), c'est *peut-être* le signe qu'un polygone pénètre dans Q, mais n'oublions pas qu'une situation comme *f*-V.20 (b) est toujours possible.

Si P rentre véritablement dans Q, il existe deux arêtes $p_{j-1}p_j$ et p_jp_{j+1} de Fr(P) ($q_i=\Omega_2=p_j$ sur la figure) telle que la première soit extérieure et la seconde intérieure à Q. Après avoir trouvé une arête polygonale rentrante dans l'arbre des références de $[q_i,Z]$ ($Z=p_{j+1}$), on localise l'arête précédant celle-ci sur la frontière polygonale du polygone intrus ($p_{j-1}p_j$ sur la figure). Il suffit alors de compter le nombre de fois que l'on rencontre la frontière de Q en tournant depuis l'arête canonique courante de départ, $[p_j,p_{j-1}]$, autour de p_j dans le sens des aiguilles d'une montre, jusqu'à atteindre l'arête canonique 'but' $[p_j,p_{j+1}]$: si ce nombre est pair (ou nul), le polygone P ne traverse pas la frontière de Q comme b dans *f*-V.20 et l'on passe à l'arête de Fr(Q) suivante.

Si ce nombre est impair, la frontière de Q est bien franchie et il faut procéder à la décomposition locale de P par Q. Pour ce faire, on parcourt la section de Fr(P) sur laquelle bute Fr(Q) en Ω_2; cette section va de Ω_1 à q_i sur *f*-V.22. Mais, cette section doit être parcourue 'à reculons' en créant, à chaque arête de Fr(P) empruntée, une arête jumelle, de sens opposée, pour la fermeture de la composante connexe en construction de Q.

Si q_uq_{u+1} est l'arête de Fr(Q) couramment empruntée, l'origine de cette arête, en tant qu'élément de Fr(Q) est, bien évidemment, le sommet q_u. L'arête jumelle, futur élément de Fr(P), admet les mêmes extrémités mais son origine est en q_{u+1}. Pour parcourir la frontière de Q à reculons, il suffit d'utiliser le chaînage inverse du chaînage naturel. L'arête précédant q_uq_{u+1} est $q_{u-1}q_u$. Le seul moyen d'arrêter le processus de recul sur Fr(Q) est la rencontre d'une arête sortante de Fr(P), à main gauche, autour du pivot courant, q_u, l'origine de l'arête courante de Fr(Q).

On parcourt donc la liste des arêtes rencontrées dans le sens des aiguilles d'une montre autour de q_u, depuis l'arête canonique courante de départ, $[q_uq_{u+1}]$. Cette liste ne peut jamais être vide, puisqu'elle contient au moins $[q_{u-1}q_u]$.

---

[4] Dans cette sous-section, il sera convenu que [A,B] est l'arête canonique associée à l'arête polygonale AB et que [B,A] est l'arête canonique duale, associée à l'arête polygonale opposée BA.

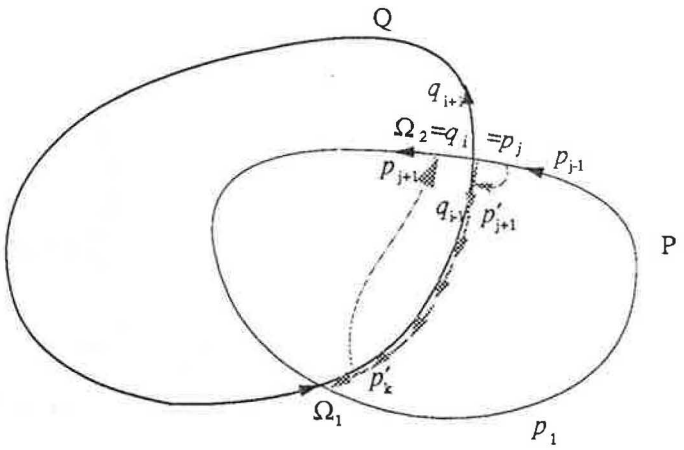


Figure V.22 (répétition). Décomposition et priorité.

S'il n'existe pas d'arête intermédiaire entre ces deux arêtes canoniques symbolisant la frontière de Q, on crée une arête jumelle, comme expliqué plus haut. Pour l'instant on considère qu'il s'agit là d'une liste annexe, chaînée naturellement, mais non raccordée à la frontière de P.

Si, en revanche, il existe une arête canonique intermédiaire entre $[q_uq_{u+1}]$ et $[q_{u-1}q_u]$, et si celle-ci représente une arête sortante de P (Ω_1 de *f*-V.22), on arrête le processus de recul sur Fr(Q) et, dans tous les autres cas, on passe à la suivante des arêtes canoniques incidentes en q_u. Appelons p'_{k-1},p'_k l'arête polygonale de Fr(P) obtenue en dernier, au moment de la rencontre de cette arête sortante de Fr(P).

Lorsqu'on 'recopie' la section de Fr(P) empruntée à Q, il faut être attentif dans le recollement des trois sections de Fr(P): celle précédant l'intrusion dans Q (jusqu'à $p_{j-1}p_j$), celle suivant Fr(Q) (entre p_j et p'_k) et celle débutant à l'intérieur de Q (à partir de p_jp_{j+1}). En fait, il suffit de chaîner dans cet ordre les trois sections pour préserver la composante connexe découverte et garder l'ancienne liste d'arêtes de frontière, qu'il faut continuer à explorer.

Notons que si le modèle permet de décrire explicitement les composantes connexes d'un polygone (un tel modèle ne contient guère qu'un étage de plus que celui décrit ici...), on peut scinder Fr(P) en deux, au niveau de p'_{k-1},p'_k, puisque le chemin débutant en p_jp_{j+1}, s'il peut encore conduire à une autre composante connexe, ne peut pas se situer dans la même que celle à laquelle appartient Fr(P) 'en amont' de p'_{k-1},p'_k.

Il faut enfin signaler que l'arête pénétrant dans Q, p_jp_{j+1}, n'est plus 'valide', dans le sens où elle est invalidée par son intrusion dans un polygone plus prioritaire: ceci peut facilement se faire par l'intermédiaire d'un drapeau booléen dans la structure d'arête polygonale, susceptible de prendre les valeurs 'valide' et 'invalide'.

Le processus précédent doit être pratiqué sur toutes les arêtes polygonales incidentes en q_i, rencontrées entre $[q_i,q_{i-1}]$ et $[q_i,q_{i+1}]$ (voir plus haut), rentrant dans Q et de plus faible priorité.

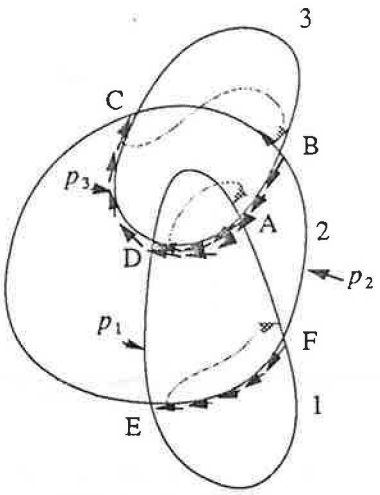


Figure V.23. Décomposition de trois polygones.

Il apparaît donc, au vu de ce qui précède, et sauf si Q est *le* plus prioritaire de tous les objets polygonaux, qu'une section de sa frontière a pu être invalidée et remplacée par une section de déviation 'jumelle' d'une section d'un polygone plus prioritaire. Comment remarque-t-on un tel cas de figure et que faut-il faire? Pour bien comprendre ce qui se passe, il sera encore utile de s'appuyer sur un exemple, illustré à la figure V.23. Les trois polygones de celle-ci 'portent' la même priorité que leur identificateur et une flèche sur chaque contour représente les points de départ des listes d'arêtes des frontières.

On commence évidemment par le polygone 3, en p_3. Rien ne se passe jusqu'à A, puisqu'en D on ne trouve qu'une arête polygonale sortante, à main gauche. En A, en revanche, on trouve une arête polygonale rentrante, à main gauche et de priorité inférieure; le processus décrit plus haut construit une déviation le long de AD pour le polygone 1 et chaîne les trois sections obtenues, en invalidant le premier élément intrus de Fr(1). En B, un phénomène similaire se produit, concernant Fr(2). On constate qu'une section de Fr(3) est 'recopiée' une *deuxième* fois et que la déviation relative à Fr(2) passe par-dessus la frontière de Fr(1), moins prioritaire de toute façon...

Le parcours du polygone 3 se termine sans encombre. On traite maintenant le polygone 2; en commençant en p_2, on parvient en premier en B: étant donné le chaînage effectué dans le processus décrit dans cette sous-section, on trouve bien que la prochaine arête, après B, de Fr(2) est effectivement non plus celle qui, auparavant, pénétrait dans le polygone 3, mais celle qui suit une section de frontière de ce dernier! On suit d'ailleurs cette frontière jusqu'à ce que le chaînage naturel fasse rencontrer une arête invalide: celle-là qui, en pénétrant dans le polygone 3, s'est vue invalidée précédemment. Tout ce que l'on peut dire, concernant la situation présente, est que l'arête suivant *naturellement* l'arête courante est 'à des kilomètres' de cette dernière, alors que l'arête suivant *logiquement* l'arête courante se trouve à main gauche.

Ceci doit donc constituer une exception importante: si la prochaine arête à atteindre sur la frontière du polygone courant est invalide,

chercher à main gauche la véritable arête frontalière: on sait bien que c'est justement la présence d'une telle arête qui a provoqué l'arrêt du parcours à reculons de Fr(3) et le chaînage avec l'arête invalidée. Que faire, maintenant? D'une part, l'arête invalidée peut tout à fait conduire à une nouvelle composante connexe et d'autre part, la composante connexe n'est pas complète! Il suffit de mettre en file d'attente l'arête invalide et de continuer la découverte de la fin de la composante connexe courante.

Cette opération entraîne la décomposition *par anticipation* de Fr(1), en arrivant en F. La frontière originelle est détournée par celle du polygone 2, comme cela a déjà été le cas plusieurs fois, mais ici, le fait que le début de liste de Fr(1) se trouve 'visiblement' à l'intérieur d'un polygone plus prioritaire est extrêmement dommageable, car rien ne pourra permettre de connaître cet état de choses au moment où débutera la décomposition de Fr(1)!

Mis à part ce problème 'prévisionnel', la découverte de la composante connexe courante de 2 se termine sans encombre en p_2 qui était le point de départ; la composante passe successivement par les point p_2, B, A, D, C, E, F et p_2: elle est complète et peut être 'détachée'. Plus précisément, sa construction a été accompagnée de la création d'un objet de type 'polygone', destiné à porter des informations propres à la hiérarchie de la section suivante, dont il nous suffit de savoir, pour l'instant qu'il existe une bijection entre les composantes connexes et ces éléments: à chacune peut être associé de manière bi-univoque un élément de type polygone. Ainsi, le champ *composante* de chaque arête de la composante connexe peut-il être rempli par l'adresse de l'unique polygone associé à l'ensemble de la composante.

On se rappelle que la rencontre d'une arête invalidée a provoqué la mise en file d'attente de celle-ci; comme la construction de la composante connexe qui a provoqué cette opération est terminée, il suffit de retirer la première arête invalide de cette file. *A priori* tout ce que l'on sait de celle-ci est qu'elle pénètre à l'intérieur d'un polygone plus prioritaire (3 en l'occurrence). Il suffit donc de suivre le chemin de Fr(2) débutant par cette arête invalidée, en invalidant toute arête de ce contour invalide, jusqu'à trouver à main gauche un nombre *impair* d'arêtes sortantes de priorité égale à celle qui a provoqué la mise en quarantaine: cela signifie qu'il est nécessaire d'inclure dans la file d'attente l'information relative au polygone franchi, ici le polygone 3. Sur la figure V.23, on voit que l'on parcourt la chaîne BC de Fr(2) avant de ressortir 'à l'air libre': on a rencontré en C une arête canonique incidente en C, représentant le polygone de priorité 3, en passant d'une arête de Fr(2) à la suivante.

De deux choses l'une: ou bien la nouvelle arête appartient au polygone dévié (2) et a un champ *composante* vierge, auquel cas, il faut commencer la découverte d'une nouvelle composante connexe de celui-ci, ou bien non, auquel cas le travail est terminé pour la section invalidée.

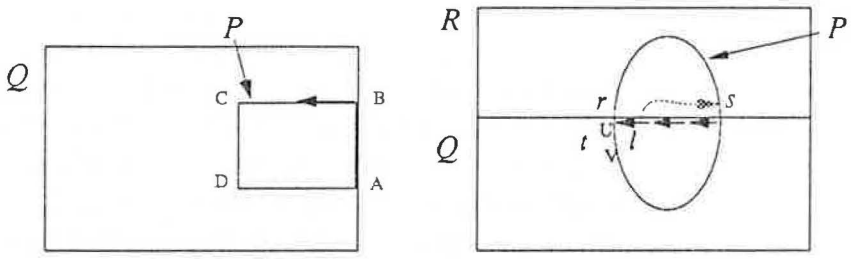
Réglons maintenant le problème concernant l'invalidation imprévisible de p_1, point de départ de la décomposition du polygone 1: pour éviter ce problème, on peut envisager de garder avec toute liste polygonale, une arête dont l'algorithme de décomposition par anticipation (appliqué à des polygones plus prioritaires) a prouvé qu'elle était nécessairement extérieure à tout polygone plus prioritaire. Par exemple, le *dernier* détournement de frontière du polygone 1, par Fr(2), a provoqué la création d'une première arête jumelle, en F: cette arête est un excellent candidat au poste d'arête valide. Ce stockage est effectué, pour un polygone donné, à chaque détournement de frontière de celui-ci, ce qui permet d'affirmer que la dernière référence gardée pour le polygone indique une section de frontière valide.

Cette dernière remarque mérite quelques explications. Une des bizarreries de la gestion de priorités dans un graphe planaire est que, du point de vue de la décomposition, un polygone P a le droit d'être complètement inclus dans un second polygone Q plus prioritaire, si leurs frontières sont disjointes, mais ne l'a plus dès que les frontières ont une arête en commun. En toute logique, la sous-section suivante devrait aussi permettre d'interdire la première situation, ou tout du moins, permettre de la gérer de manière cohérente.

Pour l'instant, examinons un cas qui permettra de saisir la portée du problème. La figure V.24.(a) présente un polygone P inclus dans un polygone Q plus prioritaire, les deux partageant l'arête AB. Lors de la décomposition de Q, qui est faite en premier, le passage par B fait apparaître une exception notoire: l'arête rentrante d'origine B de Fr(P) a bien un antécédent, comme c'est logique, mais celui-ci n'est pas à l'extérieur de Q. La situation présente incite à ne pas créer d'arête jumelle, puisque celle-ci existe déjà au sein de P sous la forme AB, et de garder cette dernière comme référence ultime de P en invalidant l'arête BC et sa précédente AB. Cette technique permet de ne jamais créer d'ambiguïté, quel que soit l'ordre de stockage des références. Dans le cas présent, il est clair que toutes les arêtes de P seront invalidées au moment du traitement propre de ce polygone car il ne ressort jamais à l'extérieur de Q.

On en déduit d'ailleurs que l'algorithme invalide, à un moment ou à un autre, toute arête sur laquelle se superpose une ou plusieurs arêtes plus prioritaires de même orientation. Il est d'autre part nécessaire de garder une trace du polygone Q qui a provoqué l'invalidation de AB: ce polygone est gardé avec la dernière référence de P, de façon à tester, au moment de la décomposition propre de P, si la chaîne débutant par AB ressort effectivement de Q.

La même technique s'applique encore à la figure V.24.(b) où le polygone P est inclus dans l'union des polygones plus prioritaires et adjacents Q et R. R est traité en premier, provoquant la création d'une chaîne jumelle et l'invalidation de l'arête s. Lorsque Q est traité, le passage par U entraîne une situation conflictuelle: n'a-t-on pas dit qu'il fallait rechercher le *prédécesseur* de l'arête entrante t? Mais celui-ci n'est autre que le segment r.



(a) (b)

Figure V.24. Deux cas où le polygone P disparaît.

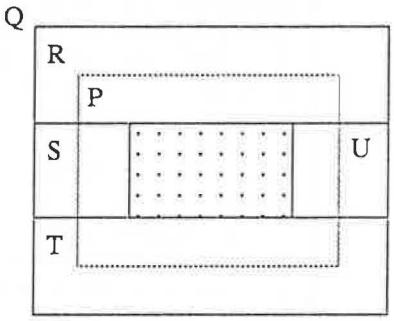


Figure V.25. Disparition inopportune d'une face.

Il faut en fait compter le nombre de fois que l'on franchit la frontière de Q entre l'arête canonique représentant *r* et celle représentant *t*, autour de U, mais aussi décompter le nombre fois où cette frontière est confondue avec une arête de *P*, aboutissant en U! Dans le cas présenté, l'arête *l* est gardée comme référence pour *P* (elle est trouvée lors de la vérification du test plus complexe exprimé ci-dessus et d'ailleurs invalidée à cette occasion), *t* est invalidée et *P* n'aura aucune arête valide, en définitive.

Comme on le voit, les cas particuliers à traiter sont nombreux, l'écriture de la procédure de décomposition ordonnée est assez complexe et nécessite une très grande attention. L'algorithme final proposé est une synthèse des actions principales engendrées.

Pourquoi faut-il effectuer une décomposition anticipée des polygones moins prioritaires, durant la décomposition d'un polygone donné? Il suffira d'un exemple pour se convaincre des conséquences qu'entraînerait la suppression de cette opération. Dans la figure V.25, le polygone P est pris en étau par quatre polygones plus prioritaires: si l'on ne 'fermait' pas, à chaque décomposition, la partie de P extérieure à chaque polygone, la frontière de P serait entièrement avalée par les quatre polygones et la face centrale disparaîtrait, au lieu d'être attribuée au résultat logique de la décomposition de P par les autres polygones!

Il est maintenant possible de donner une synthèse de l'algorithme de décomposition par priorités décroissantes; dans celle-ci, l'opération '$w \leftarrow$ arêtes(*ac*, orientation(*u*))' signifie que l'on cherche, dans la structure *arêtes* de l'arête canonique *ac* une arête polygonale de Fr(*o*) dont le sommet initial soit le sommet terminal de *u*; si une telle arête existe, il s'agit de l'arête sur laquelle on retombe lors de l'arrivée en bout de chaîne jumelle, qui n'est pas le successeur naturel de la dernière arête jumelle (dont le successeur est lui invalide) et qu'il faut rechaîner en

bout de la composante connexe courante (cf. 'exception' dans l'explication de *f*-V.23). Si une telle arête n'existe pas, la priorité des arêtes polygonales représentées par *ac* ne peut être qu'inférieure à celle de *o*, par l'invariant de décomposition, et *w* devient l'arête *rentrante* de plus haute priorité représentée par *ac*. On doit alors inspecter par priorité décroissante à partir de *w*, toutes les arêtes représentées par *ac* pour décomposer par anticipation les polygones P_w correspondants.

```
trier les objets polygonaux par priorité décroissante;
pour tout objet o, décomposer o:
    point de départ ← origine de dernière référence de o;
    Avancer dans la liste jusqu'à trouver une arête valide:
        si la première arête est invalide, invalider la section jusqu'à
        ressortir du polygone franchi, gardé aussi en référence;
    si (aucune arête n'est valide)
      passer à l'objet suivant;
    u ← arête courante valide de o;
    Traiter composante connexe courante de o:
        p ← extrémité finale de u;
        v ← suivant naturel de u;
        pour toute arête canonique ac à main gauche entre u et v, autour de p:
            si (u → canonique = ac)
              sortir de boucle et passer à l'arête suivante, v;
            sinon
                w ← arêtes(ac, orientation(u));
                si (w est une arête de o)
                    rechaîner u sur w;
                    mettre v en file d'attente;
                    sortir de cette boucle et passer à w;
                sinon                          ¡objet moins prioritaire à main gauche¡
                    pour toute arête rentrante depuis w représentée par ac
                        soit w.1 l'arête précédant w sur le polygone P_w;
                        si (w_1 n'est pas intérieure à o)
                            si (w_1 est intérieure à o)          ¡frontière franchie¡
                                construire une chaîne x → y empruntant Fr(o);
                                chaîner w.1 → x, y → a;
                            sinon                                ¡cas particulier f-V.24¡
                                invalider w.1;
                                x ← w.1;
                            invalider w;
                            garder x comme référence de P_w;
    dernière référence de o ← élément en tête de file d'attente;
```

Algorithme V-iv: Décomposition par priorités décroissantes.

Cet algorithme a une complexité quadratique, car la boucle principale traite tous les polygones arête par arête et la boucle imbriquée peut être amenée à localiser $O(n)$ arêtes canoniques autour d'un sommet.

V.B.2 Hiérarchies

Le dernier problème à résoudre dans cette section consiste à construire une hiérarchie globale dans le graphe: une fois l'opération de décomposition effectuée, les faces du graphe planaire sont telles que si l'on choisit deux faces quelconques, leurs intérieurs sont totalement disjoints ou bien l'une est entièrement incluse dans l'autre. Le premier cas est idéal car tout point de l'intérieur des deux faces appartient à une et une seule face et seuls les points des sections communes de frontière sont ambigus: on admet aisément qu'ils puissent sans grand problème appartenir aux deux faces qui sont adjacentes le long de cette section.

En revanche, le deuxième cas est plus ennuyeux, car si un point se trouve à l'intérieur d'une suite de f faces dont chacune, sauf la première, est incluse dans la précédente, il est situé à l'intérieur d'une *hiérarchie* locale de faces qu'il est indispensable de pouvoir caractériser sans ambiguïté.

En fait, le problème général à résoudre est plus vaste que cela: autant la décomposition ne concernait que les objets polygonaux, autant la mise en hiérarchie concerne *tous* les objets: tout sommet isolé ou tout segment d'une chaîne doit être maintenant placé dans la bonne face.

L'avantage de l'utilisation de l'algorithme de Bentley-Ottman (cf. [MIC 87]) est que le principe du balayage permet de connaître, à la verticale de toute abscisse, la *hiérarchie* de tous les segments, au moment de l'intersection. Dans un algorithme comme celui de Chazelle-Edelsbrunner, ceci n'est plus possible puisque l'insertion de segments se fait selon une planification spécifique assez déconnectée de l'ordre lexicographique de ces derniers.

Malgré tout, comme la décomposition complique beaucoup les choses, il va être nécessaire de faire comme si l'on ne disposait pas de l'atout de l'algorithme de Bentley-Ottman, pour comprendre ce qu'il faudrait faire dans le cas de l'algorithme de Chazelle-Edelsbrunner!

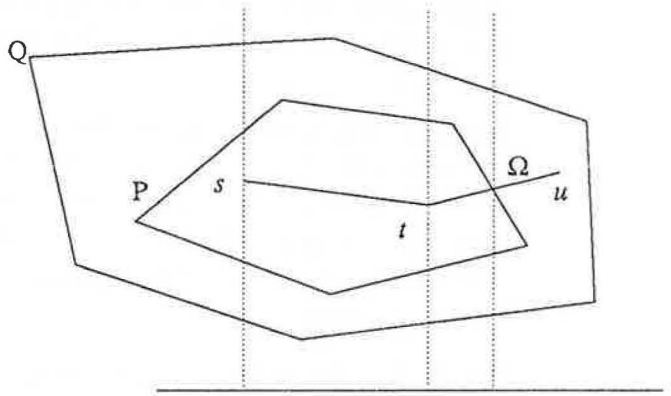


Figure V.26. Priorités et inclusions.

Quelle influence peut bien avoir l'opération de décomposition? Prenons un exemple particulièrement simple, illustré à la figure V.26. Le graphe comprend un polygone P, un polygone Q et une chaîne ouverte définie par les trois sommets s, t et u. Supposons que l'on ne prête plus attention aux problèmes de priorité: au passage à la verticale de s, la droite de balayage de l'algorithme de Bentley-Ottman présente quatre segments. La localisation du sommet s sur cette barre permet aisément d'affirmer que ce point est situé entre deux arêtes du polygone P: ceci semble vouloir dire que l'arête d'origine s est située dans le polygone P. Comme l'arête suivante tu rencontre la frontière de P, l'intersection Ω, détectée à l'avance, est un événement de type 'imprévisible'.

Quand la droite de balayage parvient à la verticale de Ω, la section $t\Omega$ est comprise entre deux segments de P et donc déclarée incluse dans ce polygone. Lorsque la droite parvient à Ω, elle ne contient plus que les deux segments relatifs à Q: le segment Ωu est 'attribué' à Q.

Si, maintenant, on se place du point de vue de la priorité, on s'aperçoit, puisque les deux polygones ont une section de frontière commune, qu'au moins une des inclusions précédentes devient fausse: si P est moins prioritaire que Q, P disparaît entièrement et toute la chaîne est incluse dans Q; si Q est moins prioritaire, c'est lui qui disparaît et la section de chaîne Ωu se trouve incluse dans le polygone qui incluait Q (éventuellement la face externe du graphe).

Ce problème mérite une seconde de réflexion: il implique aussi que des intersections entre objets peuvent être détectées, avant décomposition, qui deviennent caduques après celle-ci: c'est un problème complexe qui sera étudié dans la dernière section de ce chapitre.

Pour trouver une solution efficace, il faut rentrer un peu plus dans la structure des hiérarchies d'un graphe à priorité. Si l'on considère les deux polygones P et Q de la figure V.27.(a), la décomposition peut donner deux résultats différents, suivant que P est plus prioritaire que Q (b) ou le contraire (c). (La justification de l'interdiction, pour deux objets, d'avoir le même vecteur de priorité, prend ici toute sa force). A la limite, si les deux polygones partagent une section de frontière, on obtient les deux résultats de la figure V.28 (b) et (c).

Dans la suite, Γ représentera une représentation par segments d'un graphe planaire à priorité. Les faces de Γ sont, bien évidemment, des polygones simples. Une première propriété se déduit aisément des remarques précédentes.

Propriété 1: (*Transformations des inclusions en adjacence:*) Soient P et Q deux faces de Γ, telles que $Q \subset P$, $P <_p Q$ et $Fr(P) \cap Fr(Q) \neq \emptyset$. Après l'opération de décomposition, P est éventuellement décomposée en plusieurs composantes connexes *extérieurement adjacentes* à Q (voir la figure V.29, où l'on remarque que l'adjacence intérieure peut être limitée à un seul sommet!

Corollaire: *Après la décomposition*, aucune arête canonique ne peut représenter plus de *deux* arêtes polygonales valides, celles-ci ayant des orientations nécessairement opposées, en vertu du théorème de la courbe simple de Jordan.

Preuve: Considèrons deux polygones simples quelconques dans le plan euclidien, de frontières orientées dans le sens direct. Si ces deux polygones ont une arête en commun, on rencontre deux cas de figure: si les polygones sont *intérieurement adjacents*, les deux arêtes polygonales ont la même orientation, par application de théorème de Jordan (cf. figure V.30, à gauche); sinon, les deux polygones sont *extérieurement adjacents* et l'orientation de leurs frontières oblige les deux arêtes à montrer des orientations opposées (*f*-V.30, à droite). Il suffit ensuite de remarquer que la propriété 1 interdit le premier cas.

La propriété suivante s'applique à deux faces partageant une arête non verticale:

Propriété 2: (*ordre vertical des arêtes jumelles*) Si *a* et *a'* sont deux arêtes des faces adjacentes P et P' de Γ représentées par la même arête canonique, on peut toujours classer celle dont l'extrémité initiale coïncide avec l'extrémité finale (d=i) avant l'autre, sur une droite de balayage verticale intersectant les deux (voir figure V.31).

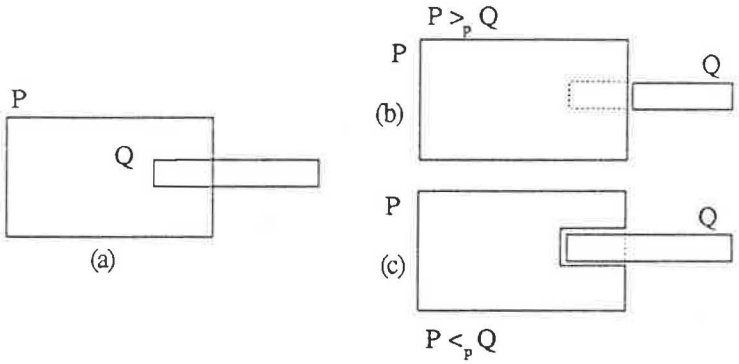


Figure V.27. Hiérarchie, inclusion et priorité.

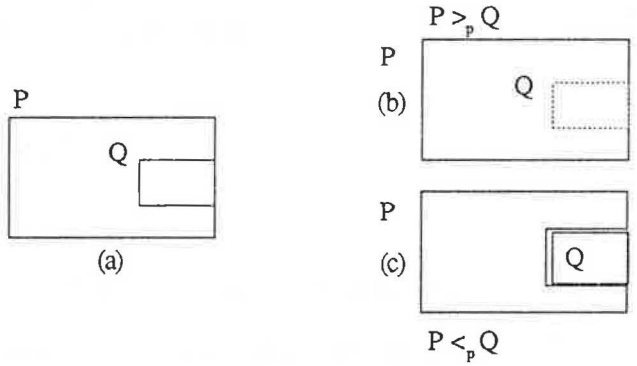


Figure V.28. Cas limite dans une hiérarchie.

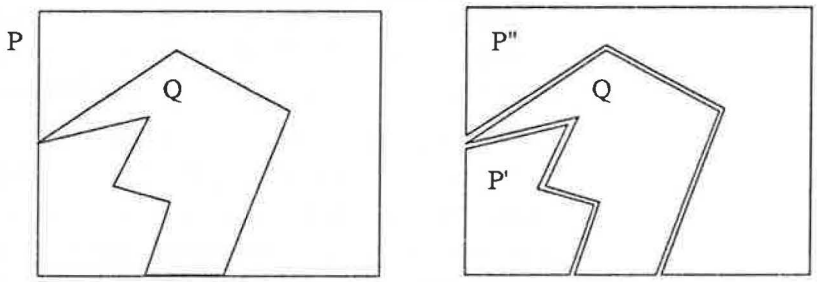


Figure V.29. Transformation d'inclusion-adjacence en adjacence pure entre faces d'un graphe à priorité ($P <_p Q, P \supset Q$).

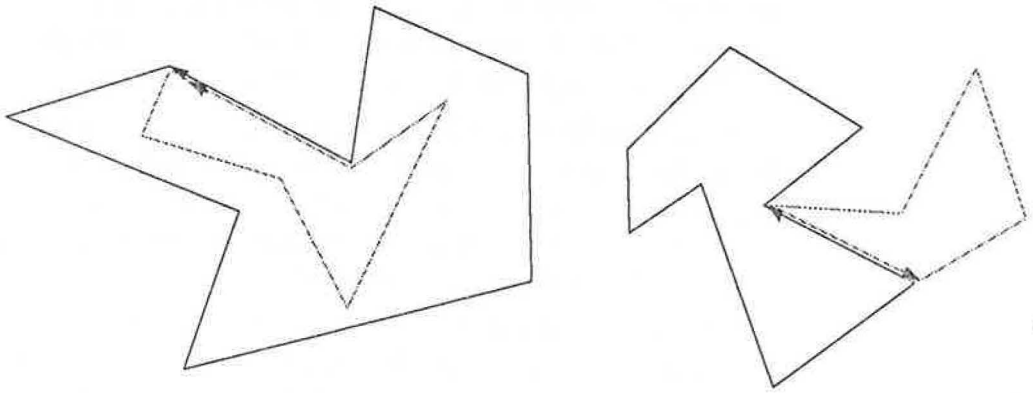


Figure V.30. Orientation d'arêtes de polygones intérieurement ou extérieurement adjacents.

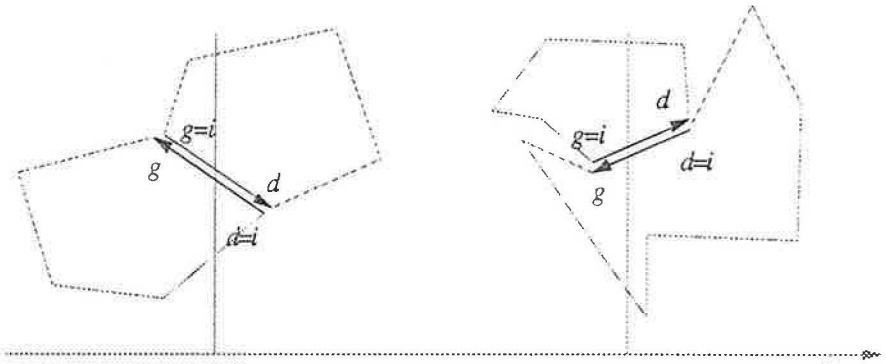


Figure V.31. Ordre vertical de deux arêtes jumelles.

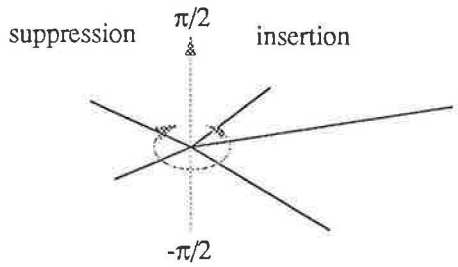


Figure V.32. Arêtes canoniques de demi-plans gauche et droit.

Cette propriété, qui est une conséquence directe de ce qui précède, permet de conclure que l'algorithme de balayage, appliqué aux arêtes de Γ, peut être spécialisé de manière à toujours chercher, quand une arête n'est pas unique, à placer les deux seules arêtes valides ayant ces deux extrémités et dans un ordre dicté par l'orientation propre de chacune.

Si l'on a utilisé, pour l'intersection, un algorithme de type Chazelle-Edelsbrunner, on doit effectuer un balayage du plan pour construire explicitement la hiérarchie des faces du graphe. Ce balayage travaille uniquement sur les arêtes canoniques représentant au moins une arête valide après décomposition. Sachant que chacune donne lieu, au maximum, à deux éléments sur la droite de balayage, on s'arrange pour que ces éléments jumeaux soient représentés par un original et une copie pointant sur ce dernier, de façon à éviter toute duplication de calculs, quelle que soit la profondeur d'imbrication de la hiérarchie des faces!

A cause de leur double chaînage, il est possible de parcourir la liste des arêtes canoniques incidentes en un sommet, depuis l'origine des angles ($-\pi/2 = 3\pi/2 \bmod (2\pi)$), dans le sens rétrograde (ceci donne les arêtes dans le demi-plan gauche) ou dans le sens direct (pour les arêtes du demi-plan droit). Voir figure V.32.

Le balayage peut alors être fondé sur l'ordre x-polaire: on parcourt les sommets du graphe planaire par classes d'abscisse (grâce à la structure de saule) et, pour un sommet donné, on commence par supprimer les arêtes canoniques 'valides' dont l'angle polaire est dans l'intervalle $]\pi/2, 3\pi/2[$ (car elles sont nécessairement caduques; en fait, bien entendu, ce sont les duales des arêtes canoniques gauches du sommet courant qui sont sur la droite de balayage!), en délaissant systématiquement les arêtes verticales qui ne sont d'aucune utilité, puis l'on insère les arêtes canoniques valides d'angle compris dans $]-\pi/2, \pi/2[$.

On peut alors appliquer une technique de 'jetons': chaque arête canonique donne lieu à un ou deux jetons permettant d'identifier les arêtes polygonales correspondantes (identificateur d'objet, pente de l'arête, y courant, extrémité finale); ces jetons sont les éléments de la droite de balayage; lorsque deux jetons d'un même polygone aboutissent au même point, ils sont supprimés de la droite lorsque celle-ci parvient à la verticale de ce point.

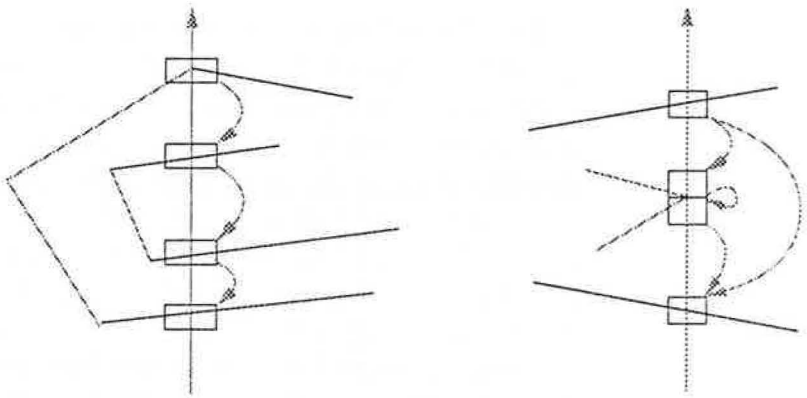


Figure V.33. Construction de hiérarchie par balayage.

Si l'on prend soin de toujours mettre à jour, à l'insertion, le champ donnant le voisin inférieur de chaque jeton (éventuellement vide, pour le plus bas sur la droite), à tout moment, le voisin inférieur de tout jeton indique la face dans laquelle l'élément (face, arête ou sommet isolé) correspondant est situé. Lorsque deux jetons collatéraux 'meurent', le voisin supérieur du plus haut des deux se voit attribuer comme voisin inférieur, celui du plus bas des deux. Toutes ces considérations sont illustrées à la figure V.33.

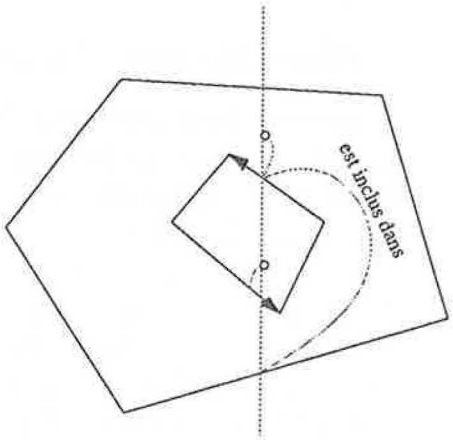


Figure V.34. Détermination de la face d'inclusion.

Il est très facile, sachant cela, d'organiser l'algorithme de construction de la hiérarchie, en se servant de la relation de voisin inférieur; pour ne donner qu'un cas de situation, si un sommet isolé a comme voisin inférieur une arête de type (g, i=d), ce sommet est nécessairement extérieur à la face correspondante et donc dans l'intérieur de la face qui le contient (voir la figure V.34). Si, au contraire, l'arête est de type (i=g, d), le sommet est inclus dans la face à laquelle appartient cette arête.

Comme tous les sommets isolés qui intersectaient des arêtes ont été avalés par celles-ci, il ne reste plus que des sommets proprement inclus dans des faces et la démarche précédente donne toujours la bonne réponse.

La complexité de cet algorithme de construction de la hiérarchie est le même que celui de la gestion de la structure de balayage, soit $O(n\log n)$, si n est le nombre d'arêtes canoniques.

La relation 'est inclus dans' est construite en même temps que la relation d'inclusion d'un élément est trouvée: on met à jour un champ pointant sur le *polygone* (face) contenant l'élément. A l'insertion d'une arête canonique portant une arête relative à une composante connexe non encore hiérarchisée, on crée un polygone et on l'insère dans l''arbre de hiérarchie' qui permet de matérialiser la structure hiérarchique du graphe planaire. Chaque nœud de cet arbre possède un fils unique et un pointeur sur un polygone 'frère': la relation père-fils traduit l'inclusion et la relation polygone-frère traduit la reconnaissance d'un même père: tous les polygones frères sont inclus dans une même face. La chaîne de fraternité est une liste chaînée. Ceci permet à tout nœud d'avoir un nombre variable de fils. Ce faisant, on prend le risque d'une gestion quadratique assez lourde si le graphe est composé de $O(n)$ faces ayant un nombre borné d'arêtes chacune; si cette situation doit être rencontrée fréquemment, il est préférable de 'binariser' cet arbre et de l'équilibrer.

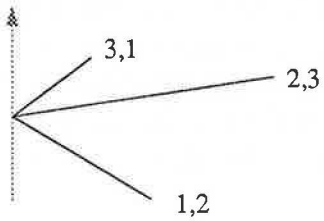


Figure V.35. Ordre x-polaire contre ordre lexicographique.

Remarque: Il est intéressant de noter que l'ordre lexicographique n'est pas celui qui permet de construire la hiérarchie: si l'on se réfère à la figure V.35, où les trois arêtes canoniques portent en premier l'ordre polaire dans lequel elles sont organisées autour du sommet d'incidence et en second l'ordre lexicographique qui les lient, on s'aperçoit que seul l'ordre polaire local respecte la cohérence d'inclusion hiérarchique recherchée: il est malgré tout notoire que l'algorithme de Bentley-Ottman, en rendant le graphe planaire, constitue une passerelle entre l'ordre lexicographique et l'ordre x-polaire!

Revenons maintenant à la problématique de l'algorithme de Bentley-Ottman: si, au moment de l'intersection, plusieurs polygones partagent une section commune de frontière, il n'y a aucun moyen pour pouvoir ordonner de manière cohérente les arêtes 'superposées', à la verticale d'une même abscisse: comment prédire l'ordre de succession des arêtes horizontales sur la droite de balayage de la figure V.36, où les deux polygones ont en commun trois arêtes? La seule information qui permette d'ordonner ces arêtes n'est disponible qu'à la verticale de l'extrémité finale des deux arêtes 'internes'!

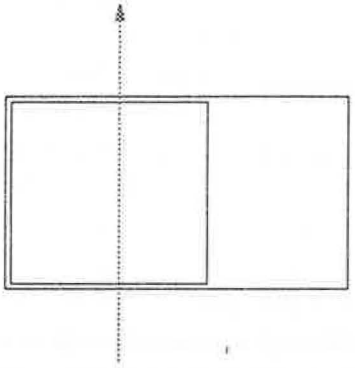


Figure V.36. Discrimination impossible entre les arêtes horizontales.

En revanche la propriété 2, énoncée plus haut, permet de retrouver *facilement* toute l'information de hiérarchie, si l'algorithme de Bentley-Ottman est spécialisé pour s'appliquer aux arêtes canoniques, c'est-à-dire aux représentants uniques des classes d'équivalence des arêtes polygonales: il suffit de garder une trace de la hiérarchie verticale des arêtes canoniques lors du balayage du plan par l'algorithme, comme le démontre la suite.

Si l'on effectue un parcours infixé de l'arbre binaire des points, on sait que l'on parcourt ces points par classes d'équivalence d'abscisses. On ne considère plus, en chaque point situé à la verticale d'une abscisse, que les arêtes canoniques du demi-plan droit. On a, d'autre part, pris soin, lors de l'algorithme d'intersection par balayage, d'accompagner ces arêtes d'un champ pointant sur l'arête canonique voisine inférieure: ce champ est mis à jour, pour toute arête canonique, au moment de son insertion; comme l'algorithme de Bentley-Ottman oblige à rechercher les deux voisins immédiats au-dessus et au-dessus des arêtes insérées, cette mise à jour est 'gratuite'.

Il sera utile d''isoler' cette ou ces deux arêtes polygonales valides de façon à pouvoir y accéder en temps constant par la suite; le plus simple est certainement de détruire, au cours de la décomposition, toute référence non valide détectée dans une structure *adresses* par exemple: le processus de décomposition a certainement laissé l'occasion de localiser et d'ordonner, pour chaque arête canonique, les deux arêtes polygonales qui restent valides, dans le meilleur des cas.

La seule chose portant à conséquence qui puisse arriver à une arête canonique, dans le processus de décomposition, est de ne plus représenter d'arête polygonale valide: on sait que ceci se produit pour des raisons de cohérence de priorité. Ainsi, si l'on remonte, à la verticale d'une abscisse et de bas en haut sur la droite de balayage dont la construction est maintenant seulement *simulée*, toute arête canonique porte zéro, une ou deux arêtes polygonales valides au maximum: il suffit de construire la relation d'inclusion en sautant les arêtes canoniques non valides et en suivant la relation de voisin inférieur déterminée par l'algorithme de Bentley-Ottman, suivant le principe indiqué précédemment, mais en la faisant cette fois-ci porter sur les arêtes polygonales valides mises en évidence pour chaque arête canonique.

D'un point de vue 'dynamique', l'inclusion d'un polygone dans une face n'est *réellement* faite qu'une fois, lors de la rencontre de l'arête non verticale de sa frontière la plus à gauche lexicographiquement. De même, l'inclusion d'une arête de chaîne se fait lors de la rencontre de son extrémité gauche et celle d'un sommet isolé, au moment où l'on atteint son nœud dans l'arbre binaire: si la liste des arêtes canoniques de ce nœud est vide, le point est bien isolé et doit être 'placé', sinon, comme il est inclus dans une arête (quelle qu'elle soit), on peut éventuellement l'ignorer. Enfin, dans le cas où une arête d'une chaîne ouverte est superposée à une arête de frontière polygonale, on peut éventuellement ignorer cette première, si l'application l'autorise.

Cette fois-ci, le processus est *linéaire* dans le nombre total d'arêtes, car on ne fait que parcourir l'arbre binaire en exploitant les résultats de l'ordre polaire local établi par l'algorithme d'intersection.

Illégalité

Un polygone moins prioritaire peut se glisser à l'intérieur d'un autre polygone: si leurs frontières sont disjointes, le processus de décomposition a laissé le premier polygone intact. C'est à la procédure de construction de la hiérarchie de référer aux 'dictionnaires' de priorité pour décider si l'inclusion est illicite ou tolérée.

V.C. CONTRAINTES GEOMETRIQUES ET DEPENDANCES SEMANTIQUES

Cette dernière section va permettre d'explorer des phénomènes que ne peuvent traiter les deux premières. On a déjà vu que l'existence d'une relation d'ordre sur les objets pouvait provoquer des intersections 'indues' dans le sens où l'un des objets provoquant une telle intersection est susceptible d'être totalement supprimé dans le processus de décomposition: dans un tel cas, l'objet plus prioritaire intersecté se retrouve avec des sommets de frontière de génération non nulle qui ne servent en fait à rien.

On va voir que de telles situations sont très fréquentes et assez ennuyeuses quand de nouveaux types d'objets, présentés maintenant, sont utilisés.

V.C.1 Une nouvelle génération d'objets

On se donne une chaîne c (*a priori* ouverte, mais éventuellement fermée) de segments du plan $\boldsymbol{P}$ et un réel positif r. On appelle *r-dilaté de la chaîne c*, l'ensemble, noté $r(c)$, défini par

$$r(c) = \{z \in \boldsymbol{P} \mid \exists\, y \in c,\ d(z, y) \leq r\} \tag{V.2}$$

où d représente la distance euclidienne dans le plan. La frontière de cet ensemble est le lieu géométrique des points situés à la distance r de la chaîne. On constate aisément que si c est réduit au point ϖ, le r-dilaté de ce point n'est rien d'autre que le disque centré en ϖ, de rayon r. De même, si r est inférieur au rayon du cercle C, le r-dilaté de C est une couronne circulaire.

Intuitivement, le r-dilaté de c s'obtient en faisant circuler un disque centré sur les points de la chaîne, et en prenant l'union de tous les dis-

ques obtenus. La figure V.37 donne un exemple de r-dilaté. On a conservé 'l'axe de définition' du r-dilaté et les disques de dilatation, afin de rendre le dessin plus compréhensible. C'est une convention qui sera souvent utilisée.

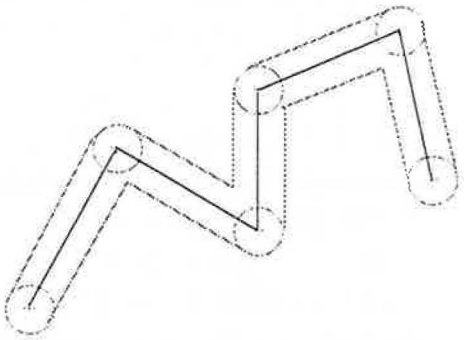


Figure V.37. r-dilaté d'une chaîne de segments.

Dans le meilleur des cas, le r-dilaté d'une chaîne ouverte est un 'polygone' simple, sans trou et à une seule composante connexe. Si les deux extrémités de la chaîne coïncident, donc si la chaîne est fermée, le r-dilaté de c est un polygone simple avec un trou, (on parlera de 'couronne' polygonale) si la frontière de $r(c)$ ne s'auto-intersecte pas... Une couronne polygonale dégénérée est représenté à la figure V.38.

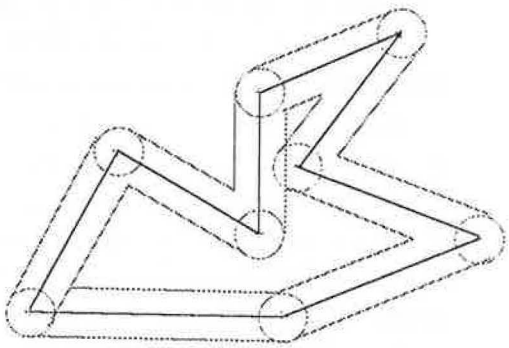


Figure V.38. Couronne dégénérée.

Notons que le paramètre r est le même pour tous les segments de définition d'un r-dilaté, mais que chaque r-dilaté peut être défini pour une valeur quelconque positive de r.

Permis de construire

On l'a vu sur la figure précédente, la création d'un r-dilaté à partir d'une chaîne et d'un r ne donne pas nécessairement un 'bon' objet. Dans l'esprit de leur création, les r-dilatés doivent représenter des polygones simples, après d'éventuelles vectorisations des parties continues. Cela signifie qu'il faut s'attacher à modéliser des objets aussi proches que possible des objets finaux, qui sont, on s'en doute maintenant, des routes, des rivières ou tout objet de largeur pratiquement constante et beaucoup plus long que large.

La démarche qui suit a pour but d'essayer de fournir un minimum de contraintes géométriques pour toujours obtenir des objets licites, mais le moins possible, afin de ne pas restreindre le champ d'investigation à de simples contingences matérielles.

Il faut, en premier, déterminer des contraintes géométriques pour s'assurer que chaque r-dilaté construit est un polygone simple, voir une couronne (non-dégénérée). La figure V.39 contient un grand nombre d'informations de premier ordre. Elle représente ce que l'on convient d'appeler un 'virage' (on peut dire qu'il s'agit d'un virage à droite si la chaîne est parcourue dans le sens JϖK). On a noté α le 'demi-angle' entre les deux segments consécutifs de chaîne, Jϖ et ϖK: l'angle complet est supposé 'géométrique', c'est-à-dire qu'il est toujours compris entre 0 et π, et donc $\alpha \in [0, \pi/2]$. Pour les virages à gauche, il suffit d'appliquer une symétrie à la figure pour obtenir des résultats analogues à ceux qui seront trouvés.

Pour le segment de définition Jϖ, on trouve deux 'génératrices', de support parallèle à celui de Jϖ: la génératrice *intérieure* passe par E et la génératrice *extérieure* passe par C et B; la projection orthogonale de E sur Jϖ est F et sur la génératrice extérieure est C. G et D jouent les mêmes rôles pour la génératrice extérieure du deuxième *tronçon*. Le point E ne peut exister que lorsque les deux génératrices intérieures successives se rencontrent *normalement*, c'est-à-dire si $\alpha \neq 0$. C'est la première contrainte géométrique qui vient à l'esprit. H est le point de rencontre des génératrices extérieures, et la condition d'existence de ce point est naturellement la même que la précédente.

Dans le cas où les deux segments de définition successifs sont colinéaires, on construit les deux tronçons en mettant bout à bout les génératrices, dont on peut affirmer qu'elles ont mêmes équations.

ϖ est le centre du cercle de rayon r passant par A et B, projections de ϖ sur les deux génératrices extérieures: A et B sont donc les deux points de tangence du cercle et des génératrices extérieures. On retrouve l'angle α, entre autres, dans les secteurs $\angle$AHI, $\angle$GϖE, $\angle$CDE. Enfin, δ est la bissectrice de $\angle$CHD et$\angle$FϖG, est un axe de symétrie du quadrilatère CHDE, et T est le point de rencontre entre cette bissectrice et CD. Par la suite, on notera indifféremment UV le segment géométrique d'extrémités U et V et la mesure de sa longueur: le contexte suffira à éviter les ambiguïtés.

On peut établir le florilège de relations suivantes, qui se révéleront utiles par la suite:

$$
\begin{aligned}
&CF = FE = EG = GD = A\varpi = B\omega = G\varpi = F\varpi = r \\
&\varpi G = \frac{EG}{\mathrm{tg}\angle E\varpi G} = \frac{r}{\mathrm{tg}\alpha} = \varpi F, \\
&AH = \frac{\varpi A}{\mathrm{tg}\angle AH\varpi} = \frac{r}{\mathrm{tg}\alpha} = {}^1\!/_2 HD = BH, \\
&ET = DE\cdot\cos\angle TDE = 2r\cdot\cos\alpha, \\
&AI = A\varpi\cdot\cos\angle IA\varpi = r\cdot\cos\alpha, \\
&AB = 2\cdot AI = 2\cdot r\cdot\cos\alpha, \\
&CD = 2\cdot AB = 4\cdot r\cdot\cos\alpha, \\
&HE = \frac{EG+GD}{\sin\angle EHD} = \frac{2r}{\sin\alpha} \\
&IE = I\varpi + \varpi E = r\cdot\sin\alpha + \frac{r}{\sin\alpha}
\end{aligned}
\qquad \text{(V-3)}
$$

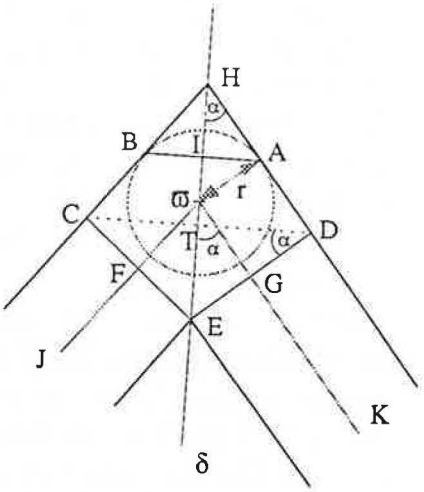


Figure V.39. Géométrie d'un virage à droite.

La relation qui permet de déterminer ϖG est très importante, car elle donne une borne de validité: supposons que le virage en K de demi-angle α' soit aussi à droite; celui-ci donne lieu à un point G', équivalent de G pour le virage en ϖ. Pour que le virage en ϖ soit possible, il est nécessaire que E ne 'déborde' pas trop loin, c'est-à-dire que ϖF reste inférieur à ϖJ et d'autre part, pour que le virage en K soit possible, il est nécessaire que la somme ϖG+G'K soit inférieure à ϖJ. Si le virage en J est dans l'autre direction, seule la première condition s'applique en ϖ. Les deux relations précédentes s'expriment ainsi:

(V-4)
$$\varpi J > \frac{r}{\mathrm{tg}\alpha} \Leftrightarrow \mathrm{Arctg}\left(\frac{\varpi J}{r}\right) < \alpha < \pi/2$$

$$\varpi G + G'K < \varpi K$$

$$\Leftrightarrow \quad r\left[\frac{1}{\mathrm{tg}\alpha} + \frac{1}{\mathrm{tg}\alpha'}\right] < \varpi K$$

(V-5)
$$\Leftrightarrow \quad \frac{1}{\mathrm{tg}\alpha} + \frac{1}{\mathrm{tg}\alpha'} < \frac{\varpi K}{r} \Leftrightarrow \alpha > \pi/2 - \mathrm{Arctg}\left[\frac{\varpi K}{r} - \frac{1}{\mathrm{tg}\alpha'}\right]$$

La construction peut ainsi s'effectuer de proche en proche en examinant les directions successives des virages. Nous ne nous intéressons ici qu'à la modélisation; les problèmes de calculs (approximation de fonctions transcendantes) seront abordés un peu plus loin.

Ce qui précède ne concerne que la validité *locale*: si une arête située à plus de deux segments de chaîne vient intersecter les génératrices du tronçon courant, la méthode ci-dessus ne peut rien faire. Il faut, en fait, replacer le problème dans un contexte plus vaste: supposons la construction d'un r-dilaté effectuée selon les conditions précédentes; pour savoir si cet objet est valide, il suffit simplement de vérifier que sa frontière ne s'auto-intersecte pas.

On sait depuis peu ([CHA 90]) que le test de simplicité d'un polygone peut être effectué en un temps proportionnel au nombre de ses côtés. Ce résultat fondamental ne s'accompagne malheureusement pas d'un algorithme 'utilisable'. L'algorithme de [T/VW 88] donne une solution pour le même problème en $O(n\log\log n)$, si n est encore le nombre de côtés du polygone. Les structures de données permettant d'atteindre une telle complexité sont encore plus délicates à manipuler et

aucune méthode pratique n'a pu être mise au point pour utiliser l'algorithme. Devant ce cas de force majeure, on est bien obligé de se résoudre à utiliser une méthode *naïve* mais efficace: si l'on permet à l'algorithme de Bentley-Ottman de traiter des segments de droite et des arcs de cercles, on obtient une méthode en $O(n\log n)$ pour détecter une infraction à la simplicité de l'objet en construction. La recherche de toutes les k infractions s'accompagne de la complexité $O(n\log n + k\log k)$.

Comment incorporer les arcs de cercles à l'algorithme d'intersection? Pour la circonstance, un arc de cercle est avant tout un sommet: une des propriétés qui prévaut dans l'algorithme de Bentley-Ottman est qu'il est inutile de chercher à intersecter deux segments qui ne sont pas adjacents. Cette remarque tient aussi dans le cas de deux cercles et d'un cercle et d'un segment, car tous ces objets sont convexes: malgré tout, 'l'intersection' d'un cercle et d'un autre objet nécessite de remplacer la notion d'adjacence par celle de 'à une distance inférieure au rayon'.

La transformation à apporter à l'algorithme original est donc minime. En quoi ceci résout-il le problème qu'on se pose? Si l'on prend l'ensemble des segments construits pour le r-dilaté considéré, on peut affirmer que ne sont pas sécants deux arêtes consécutives ni un arc de cercle et une arête ayant un 'sommet' (point de contact de type C_1) en commun. Si une arête ou un cercle intersecte un cercle ou une arête, la construction est illicite, car la frontière n'est plus simple: au moins un point de la frontière est plus proche que r d'un des segments de la chaîne. Ceci inclut les cas où les deux extrémités de la chaîne sont à une distance inférieure à r mais non confondues: ce cas peut éventuellement être détecté et 'corrigé'.

r-dilatés tri-dimensionnels

Passons maintenant à la définition de l'objet dans l'espace qui est issu d'un r-dilaté. Commençons par le cas d'un segment simple. On a vu au début de ce chapitre que tout point d'un segment pouvait se voir attribuer une valeur numérique (par exemple une altitude) dont la valeur était fonction du champ numérique associé aux deux extrémités du segment.

Cette propriété est généralisée ici de sorte que tout point du segment qui se projette sur les deux arêtes constituant les génératrices donne la valeur de son champ numérique à ses deux projections; de plus, tout point appartenant à l'une des deux calottes circulaires reçoit comme valeur de champ numérique celle de l'extrémité associée. Cette définition se traduit par la figure V.40, dans le cas où le champ numérique représente une altitude.

Bien que le champ numérique puisse représenter toute information utile, il est préférable de garder la représentation tri-dimensionnelle de la figure précédente à l'esprit.

Il paraît assez dur de définir la valeur du champ numérique dans le cas d'un objet portant sur plus d'un segment. La difficulté provient du fait que, si deux segments consécutifs de définition du r-dilaté ont un point commun dans le plan, la situation entraîne un blocage dans

l'espace: le point E de *f*-V.39 existe dans le plan, mais il est la projection de points de deux segments de l'espace: il y a de fortes chances que les champs numériques des deux génératrices de l'espace n'aient pas la même valeur.

Nous sommes coutumiers de ce phénomène dans la construction d'un escalier: le saut d'altitude à la verticale du point de virage est contrebalancé par des marches *palières*. La situation équivalente, pour un *r*-dilaté, est illustrée à la figure V.41.

Revenons un instant sur la figure V.39. A la verticale (dans l'espace) du point d'intersection dans la plan des deux génératrices intérieures, le champ numérique devrait avoir deux valeurs, ce qui n'est pas tolérable. La même situation s'applique à la verticale de l'intersection des génératrices extérieures.

Pour sortir de l'impasse, l'idée la plus simple est de considérer qu'il existe une zone, le *palier*, dans laquelle tous les points sont dans un même plan pour le champ numérique; d'autre part, on choisit de 'forcer' ce même champ pour l'un des deux tronçons (ou les deux), de sorte que le *r*-dilaté tri-dimensionnel soit constitué de trois sections 'planes', comme sur la figure V.42.

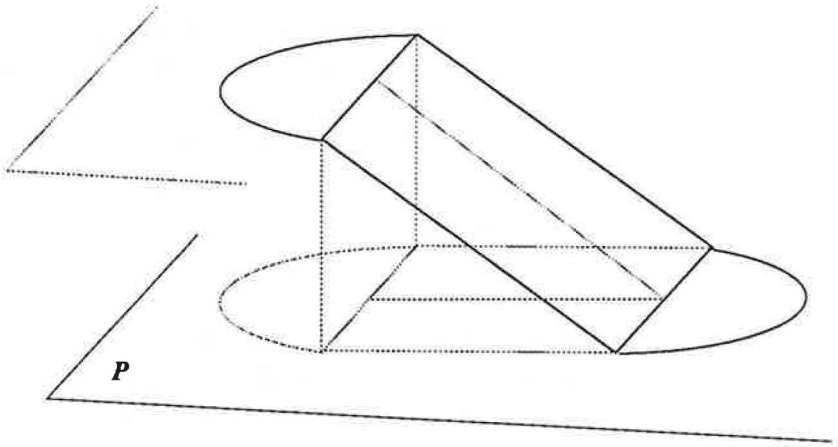


Figure V.40. Représentation tri-dimensionnelle d'un *r*-dilaté de segment.

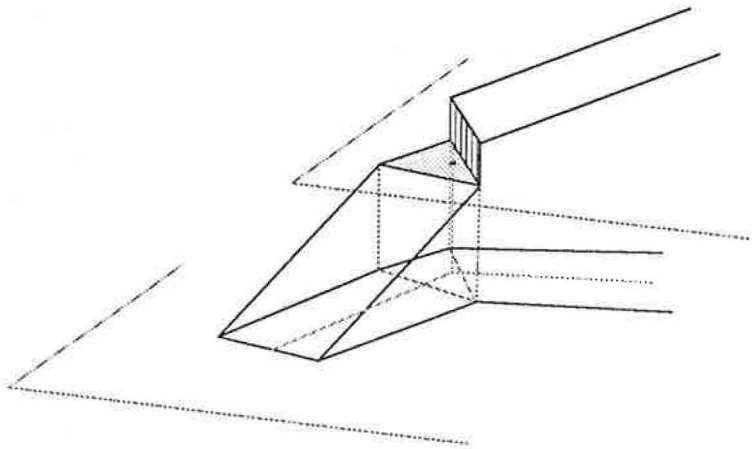


Figure V.41. Palier dans un *r*-dilaté tri-dimensionnel.

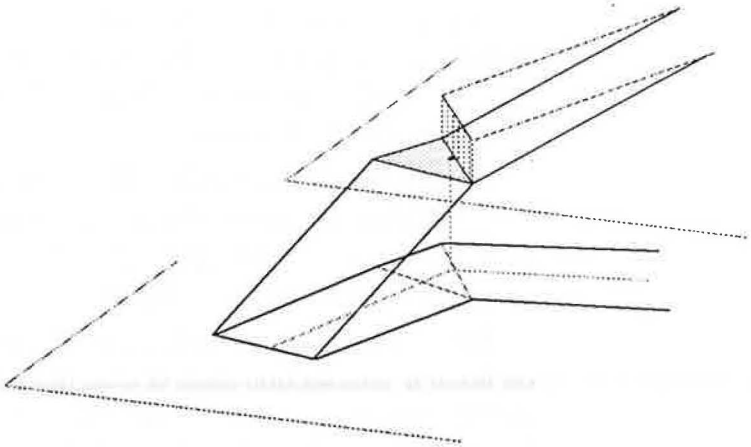


Figure V.42. Aménagement d'un palier dans l'espace.

La face 'horizontale' est un triangle sur cette figure, mais on va voir que ce n'est pas toujours le cas. Maintenant que cette contrainte géométrique est connue, essayons de formaliser un peu plus les éléments de choix.

Une des premières constatations qui s'impose est que, si l'on veut arriver à traiter de tels objets, il n'est pas déconseillé de réduire nos prétentions et d'essayer de 'vectoriser' quelque peu les parties continues, relatives aux arcs de cercle. Autant la modélisation choisie laisse toute latitude, autant la triangulation qui va suivre impose les objets les plus simples possible. L'idée maintenant est d'essayer de trouver les points les plus judicieux dans le *r*-dilaté plan, pour représenter un *r*-dilaté tri-dimensionnel aussi fidèle que possible.

Revenons à un virage à droite de *r*-dilaté dans le plan (figure V.43). Pour entériner le choix imposé précédemment, une solution consiste à utiliser le champ numérique associé à ϖ, et à l'imposer aux points A, B, C, D, E et H: tous les points de cet hexagone ont le champ numérique associé à ϖ. De cette manière, bien que l'on doive forcer la main aux tronçons Jϖ et ϖK, on obtient deux tronçons tri-dimensionnels plans et un palier horizontal. Mais, de toute évidence, tous les points C, B, H, A et D ne sont pas forcément impliqués *explicitement* dans l'objet résultant. Combien sont nécessaires et lesquels?

A ce stade, d'autres contraintes liées à l'essence même de la facétisation vont être utiles: la vertu d'une triangulation de Delaunay (étudiée en détail au chapitre suivant) est de construire dans le graphe les triangles les plus proches de triangles équilatéraux. Comme on dispose d'une certaine latitude dans la vectorisation des *r*-dilatés tri-dimensionnels, il est justifié d'essayer de préparer le terrain pour cette opération.

Les facettes sont d'autant plus représentatives qu'elles ne sont ni trop effilées ni trop minuscules: essayons d'imaginer ce qui se passerait si l'angle α devenait très petit ou, à l'inverse, très proche de $\pi/2$. On a vu dans les formules (V-3) que $AI = r \cdot \cos\alpha$. Si α est très petit, la longueur de AI est très proche de r; en revanche, si α est voisin de $\pi/2$, cette longueur devient pratiquement nulle: ceci est bien conforme à l'idée qu'on se fait de la distance entre A et B lorsque les deux tronçons sont presque exactement alignés.

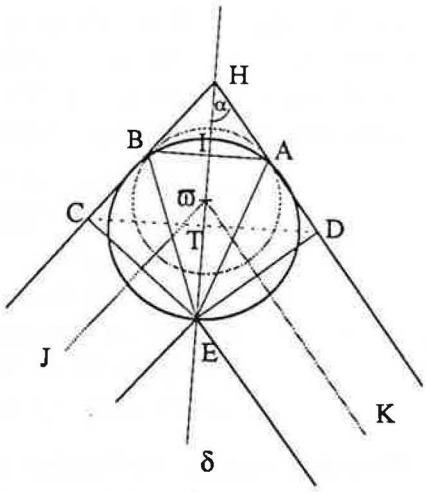


Figure V.43. Aménagement d'un palier dans le plan.

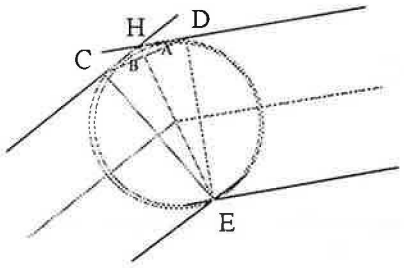
Que devient, quant à elle, la longueur du segment HE? De la formule $HE = 2r \cdot \sin^{-1}\alpha$ on déduit que HE est infiniment grand lorsque α est proche de zéro et voisin de $2r$ (la largeur totale de chaque tronçon) lorsque α est très près de $\pi/2$: ceci conforte encore notre intuition, car il est visible que plus le virage est 'serré' ($\alpha \approx 0$), plus le point H est déporté.

Il semblerait donc pertinent d'opérer deux choix, suivant la valeur de α: si cet angle passe en-dessous d'un certain seuil, on choisit d'approcher le palier circulaire par le pentagone CBADE et sinon par le quadrilatère CHDE: ce choix est conforté par le fait que, lorsque le virage est très serré, la longueur $EI = I\varpi + \varpi E = r(\sin\alpha + 1/\sin\alpha)$ devient très proche de $2 \cdot r$ mais reste bornée, alors que, si le virage est ouvert, la longueur HE reste aussi proche de la même borne: il est toujours possible d'imaginer le passage d'objets de taille proche de la demi-largeur sur chaque bande de tronçon (analogie avec la circulation sur une voie).

Quelle valeur seuil prendre pour α? L'aide va venir d'une propriété essentielle des triangles de Delaunay: dans une triangulation de ce type, tout cercle circonscrit à un triangle de Delaunay ne peut contenir, en son intérieur, un autre point de l'ensemble initial. Ceci se traduit, par exemple, par le fait que le cercle circonscrit au triangle AEB ne contient pas C et D. A quelle condition cette propriété est-elle vérifiée? On observe que lorsque $\alpha = \pi/4$, les deux tronçons de *f*-V.43 sont perpendiculaires et que $HA = AD = HB = CB = r$: En d'autres termes, les points A, B, C, D sont sur le même cercle de centre ϖ et de rayon r. Dès que α dépasse $\pi/4$, la face ABE n'est plus élément de la triangulation et elle le devient dès que cet angle est inférieur à $\pi/4$. Ceci renforce le choix car la symétrie de la figure suggère naturellement cette triangulation.

Enfin, la longueur $AB = 2 \cdot AI = 2 \cdot r \cdot \cos\alpha$ est toujours supérieure à $r\sqrt{2}$ quand α est inférieur à $\pi/4$. On a vu, d'autre part, que α est borné par la géométrie des segments: le virage ne peut être trop serré sans que ϖF devienne supérieur à la longueur du segment de tronçon ((V-4, V-5)): l'aire de la face ABE, égale à $^1/_2 EI \cdot AB = r^2\cos\alpha(\sin\alpha + 1/\sin\alpha)$ peut, *a priori*, atteindre des valeurs infinies pour $\alpha \approx 0$, mais ces valeurs ne sont

possibles que dans le cas de virages très serrés sur des tronçons assez longs: ces cas sont éventuellement améliorés par le processus de subdivision des tronçons. Pour une valeur de α proche de $\pi/4$ et inférieure, l'aire de la face ABE reste voisine de et inférieure à $^3/_2r^2$.



Que se passe-t-il pour la triangulation relative au choix du quadrilatère CHDE ($\alpha>\pi/4$)? Cette situation est reprise de *f*-V.43 et grossie en médaillon. Les quatre sommets du quadrilatère sont sur le même cercle de centre ϖ (identique à celui passant par A et B), car CE=DE=2*r*, CH=HD et $\angle HCE=\angle HDE=\pi/2$ (donc HE est un diamètre commun aux deux triangles HCE et HDE).

Pour des raisons explicitées au chapitre suivant, la triangulation de Delaunay ne peut décider entre les diagonales HE et CD car elles correspondent à des situations équivalentes. Comparons leurs longueurs:

$$\frac{CD}{HE} = \frac{4r\cdot\cos\alpha\cdot\sin\alpha}{2r} = \sin(2\alpha) \leq 1,\ \forall\ \alpha \in [\pi/4,\ \pi/2] \qquad \text{(V-6)}$$

On pourra décider de choisir (sauf si l'objet final était déformé par d'autres intersections, bien sûr) la diagonale HE qui est la plus longue du quadrilatère CHDE, afin d'éviter la petite facette CDH du médaillon, et obtenir un découpage 'dans le sens de la marche'.

Pour des valeurs de α proches de $\pi/4$, l'aire des faces HEC et HED de la triangulation de Delaunay, donnée par $r^2/tg\alpha$ est très proche de r^2; de même les longueurs HD=HC= $2r/tg\alpha$ restent proches de 2*r*. En revanche, lorsque α se rapproche de $\pi/2$, toutes ces quantités deviennent pratiquement nulles: il faut encore affiner le choix du seuil. On sait déjà que $CD=4r\cdot\cos\alpha$. Il est donc possible, dès que α franchi un second seuil, de vectoriser le palier tri-dimensionnel à partir du triangle CDE: la facétisation n'aura pas d'autre alternative que celle de choisir ce triangle comme facette de Delaunay. En testant à partir de quelle valeur de α CD devient inférieure à 2*r*, on obtient

$$CD = 4r\cos\alpha < 2r \Leftrightarrow \alpha < \pi/3$$

Ce seuil de $\pi/3$ permet de délimiter les cas où l'on peut choisir une représentation de palier triangulaire. On est maintenant amené à constater que l'aire du palier, égale à $^1/_2CD\cdot ET = 4r^2\cos^2\alpha$ est directement dépendante du cosinus de α: ceci entraîne à calculer un dernier seuil, cette fois-ci étroitement lié à la 'résolution' du graphe sous-jacent. si l'on décide qu'une facette ne peut avoir une aire inférieure à un seuil ε positif, on trouve une valeur minimale de α par la relation

$$4r^2\cos^2\alpha < \varepsilon \Leftrightarrow \alpha < \mathrm{Arcos}\left(\frac{\varepsilon^{0,5}}{2r}\right) \qquad \text{(V-7)}$$

en-dessous de laquelle la construction d'un virage de *r*-dilaté provoquerait l'apparition d'une facette triangulaire trop petite si l'on recourrait à la méthode précédente: dans ces cas, il est bien préférable de forcer les champs numériques de E et H, dont les différences de valeurs doivent être de toute façon minimes sous peine d'incohérence, de façon à supprimer la facette centrale et joindre en biseau les deux tronçons par la diagonale HE: il est pratiquement impossible que les deux tronçons

demeurent plans (les points E, C, H et B ne peuvent être 'coplanaires' sans que les deux tronçons le soient!), mais cette légère violation de la planéité permet de traiter ces cas assez correctement, bien que de manière approchée.

Les différentes techniques présentées ci-dessus permettent d'approcher l'objet continu défini par un *r*-dilaté du plan de sorte qu'il s'éloigne le moins possible du modèle décrit au début de la section. Le principal besoin de cette opération vient de l'étape de triangulation. Le modèle continu est intéressant en soi et nécessite le recours à des raccords dont les génératrices sont des morceaux d'hélices. D'autres travaux devraient suivre les concernant...

Où l'on reparle d'intersections...

Les *r*-dilatés tri-dimensionnels (r-3d en abrégé) posent de sérieux problèmes en ce qui concerne leur intersection avec deux types d'objet: les autres r-3d et les chaînes de segments plus prioritaires.

Lorsqu'un r-3d rencontre un segment plus prioritaire, le principal problème provient de la géométrie de l'intersection: le support du segment n'étant pas nécessairement dans le plan (de l'espace) du tronçon rencontré, on est obligé de recourir à une technique proche des précédentes si l'on désire résoudre le problème dans sa totalité.

Les cas particuliers sont redoutables: l'intersection peut avoir lieu au 'milieu' d'un tronçon et presque à angle droit (cas simple) ou non, ou encore dans un virage. L'angle formé entre le segment de contrainte et le segment de définition local du r-3d, projeté sur la plan, joue un rôle fondamental. Tout en précisant que l'étude de ces cas est encore à l'état de développement, donnons quelques éléments de réflexion.

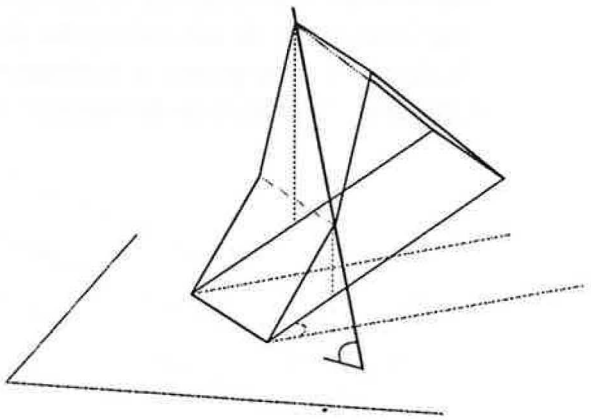


Figure V.44. Segment de contrainte et r-3d.

Pour comprendre le rôle d'un segment de contrainte, utilisons la visualisation tri-dimensionnelle de la figure V.44. On y trouve un *r*-dilaté du plan, sa version tri-dimensionnelle et un segment de contrainte: celui-ci force le champ numérique de r-3d à la verticale des deux intersections des projections de ces objets dans le plan. La situation est un peu caricaturale, mais on constate que le segment provoque l'apparition d'un pic et de trois sections planes successives.

Les segments de contrainte permettent de modéliser très fidèlement certains traits de relief.

La situation est idéale lorsque la projection du segment de contrainte coupe le r-dilaté avec un angle α pas trop faible. Une telle situation est représentée à la figure V.45.

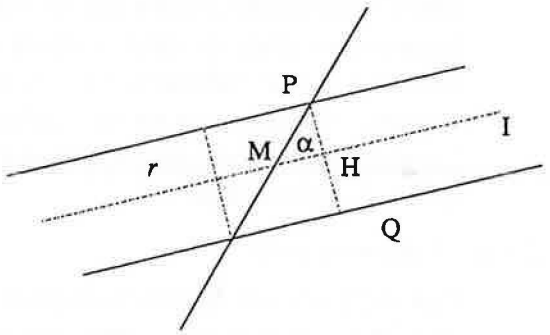


Figure V.45. Projection de l'intersection d'un segment de contrainte et d'un r-3d.

On retrouve un critère de validité semblable à ceux qui précèdent, grâce à la relation

(V-8) $$MH = \frac{r}{\text{tg}\alpha}$$

Il est facile de voir que cette construction n'est possible que si MH est inférieur à MI, I marquant la fin du segment de définition courant. Une relation équivalente existe pour la gauche. Les choses deviennent vite plus complexes lorsque la chaîne de contrainte coupe le r-dilaté de manière plus aléatoire: par exemple, pour des valeurs très petites de l'angle α, le rôle même d'un segment de contrainte devient moins clair: doit-il avoir une influence locale (au voisinage du point d'intersection avec le segment de définition) ou plus globale, mais que dire alors de l'influence d'un segment de contrainte qui serait parallèle au segment de définition? La situation devient paroxystique sur la figure V.46.

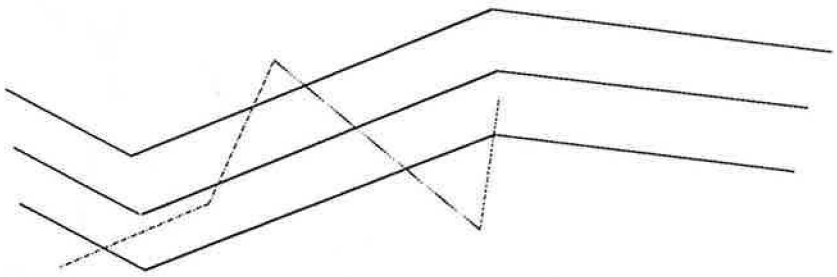


Figure V.46. Chaîne de contrainte un peu torturée...

L'heuristique adoptée pour résoudre ces problèmes est de considérer uniquement les intersections des segments de contrainte avec les segments de définition des r-dilatés et donc d'accréditer la thèse selon laquelle le point d'influence d'un segment de contrainte est maximal au contact du segment de définition. Par exemple, la situation précédente se résout en ne gardant que les deux intersections avec le segment de définition médian. Ceci donne quatre points sur les génératrices (figure V.47). Les segments de contrainte n'intersectant pas un segment de

définition sont simplement ignorés: l'avantage de cette simplification est de permettre un traitement de tous les cas rencontrés avec une relative fidélité dans la transcription du rôle des chaînes de contrainte.

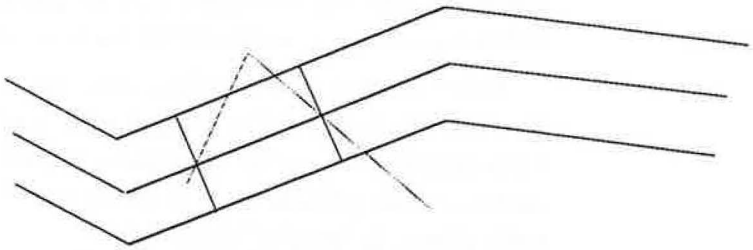


Figure V.47. Traitement simplifié du cas précédent.

Le deuxième type de difficulté que provoquent les r-3d est lié à l'intersection de ces objets avec d'autres objets de la même classe. On se rappelle que l'on a imposé une relation d'ordre sur les objets et celle-ci est valable pour les r-3d eux-mêmes: lorsqu'un tel objet en rencontre un plus prioritaire, il doit lui céder le chemin. Ceci se traduit par des résolutions de conflit de champs numériques qui ont déjà été traitées et surtout par des problèmes géométriques que nous allons examiner.

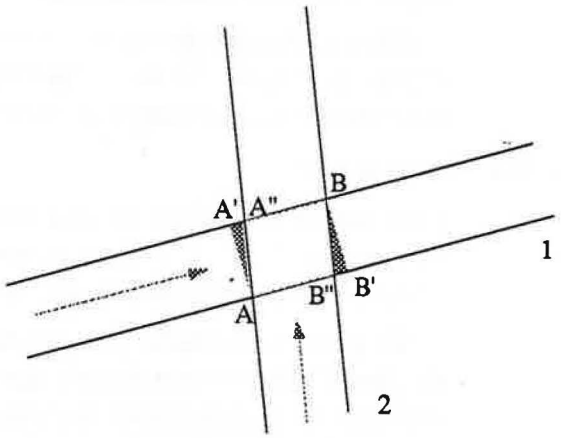


Figure V.48. Intersection de deux *r*-dilatés.

Dans le cas le plus simple, deux r-3d se coupent 'au milieu d'un tronçon' et avec un angle entre les supports orientés des segments de définition 'raisonnable'. Une situation coopérative de ce type est représentée à la figure V.48. L'objet 2 est plus prioritaire que l'objet 1. Les intersections A et B sur l'objet 2 se projettent orthogonalement sur l'autre génératrice en A' et B': suivant que α est supérieur ou non à $\pi/2$, l'intersection à projeter se trouve sur l'une ou l'autre des deux génératrices. Les triangles AA'A" et BB'B" sont des facettes de remise à niveau: le quadrilatère AB"BA" correspond à des sommets suivant la même loi de champ numérique (ils sont coplanaires si le champ numérique dénote l'altitude); il est nécessaire que les champs numériques de A' et B' soient copiés sur ceux de A et B. On retrouve encore des effets de bord dommageables lorsque α est trop proche de 0 ou de $\pi/2$. De même, certaines facettes peuvent être microscopiques lorsqu'un des deux objets est construit sur une valeur de r très faible. Ces événements peuvent être paramétrés comme précédemment.

Les choses sont légèrement plus complexes quand deux r-3d se croisent à l'endroit d'un virage. Mais comme les segments de définition imposent des contraintes locales fortes, on peut, en général, traiter toutes les intersections entre r-3d en profitant des techniques propres à celles caractérisant les intersections autres les objets.

Toute intersection multiple crée un nombre conséquent de petites facettes. On retrouve l'importance de la relation d'ordre entre les différents objets: elle permet de pouvoir traiter les intersections dans l'ordre décroissant de priorité, si l'on admet que l'on traite à part, et avant toute autre chose, le 'graphe' des r-3d.

L'heuristique adoptée pour la gestion des r-3d est la suivante: on commence par tester les intersections entre chaînes de définition de r-3d et les chaînes de contraintes. Cette opération, qui se fait grâce à la précaution que l'on a prise de permettre aux fonctions principales de travailler sur des ensembles de classe d'objets, fournit les chaînes de définition rectifiées selon la méthode de *f*-V.47. On construit ensuite les r-3d qui en découlent et on calcule les intersections de tous les objets présents, en ignorant toute intersection d'un segment de r-3d avec un segment de chaîne de contrainte.

Cette technique permet de traiter globalement les r-3d comme des objets 'normaux'. Mais la complexité de leur gestion nous entraîne maintenant vers les rivages de la sémantique des objets.

Dépendances sémantiques

Les calottes circulaires en bout des *r*-dilatés font penser que ces objets ne sont pas 'finis': si l'on est parvenu à les finaliser du point de vue des virages, rien n'a été fait concernant leurs extrémités.

En général, le graphe planaire est plongé dans un rectangle englobant du plan. Ceci provient du fait que le monde étant vaste, il faut bien le découper en parcelles pour le modéliser... Chaque parcelle porte le nom de domaine. Tout domaine possède, sauf s'il se situe au bout du monde (!), huit domaines voisins.

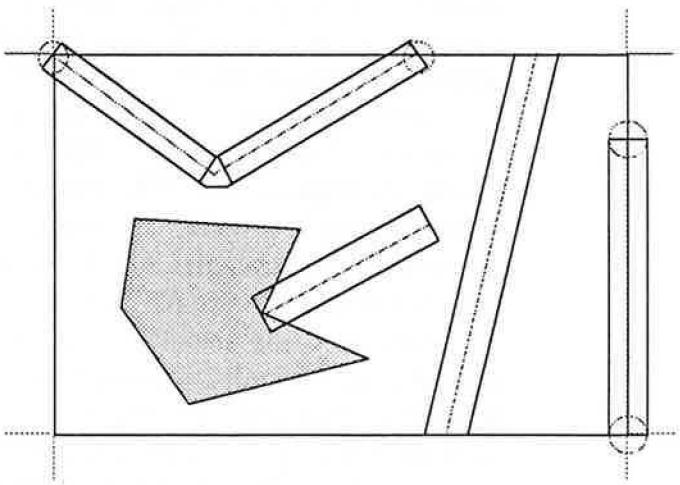


Figure V.49. Divers arrêts de *r*-dilatés suivant les circonstances.

Si un r-3d s'arrête sans rencontrer d'obstacle, on convient de l'arrêter à angle droit en projetant deux points perpendiculairement au dernier point de définition, sur les deux génératrices (figure V.49).

Si l'extrémité finale d'une chaîne de définition est située sur un des bords du domaine courant, il est clair que l'on doit faire face à deux problèmes: si le r-3d se prolonge dans le domaine attenant, il faut s'arranger pour prolonger ses génératrices jusqu'à concurrence du bord de domaine; sinon que faire? De même, comment raccorder les deux sections d'un r-3d à cheval sur deux domaines; est-ce même possible de le faire dans tous les cas?

Que se passe-t-il lorsqu'une génératrice d'un r-3d rencontre un bord du domaine ou que le dernier segment de définition aboutit à un coin du domaine? Celui-ci est l'un des problèmes qui accompagnent la gestion des *r*-dilatés polygonaux, c'est-à-dire approchés par des polygones. Citons-en quelques uns, afin de dégager une technique générale de résolution.

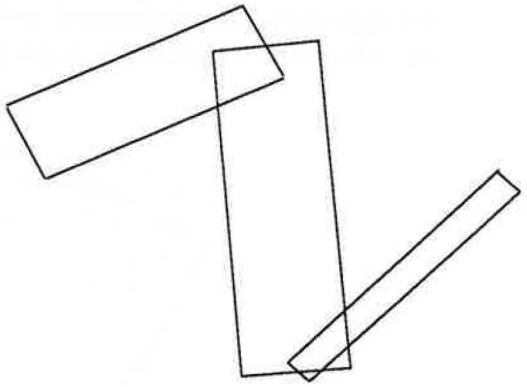


Figure V.50. La géométrie des intersections de *r*-dilatés est complexe.

La largeur des *r*-dilatés pouvant être très variable, la géométrie qui accompagne l'intersection de deux tels objets est assez complexe, comme en témoigne la figure V.50. L'intuition que l'on a de ce genre de problèmes est qu'il serait agréable de pouvoir utiliser l'information 'un segment de définition aboutit au milieu d'un autre' d'une manière plus fine, alors que tout ce qui est fait jusqu'ici est une opération d'intersection entre segments et une décomposition de faces: il serait utile de contrôler *sémantiquement* l'intersection de deux *r*-dilatés suivant la signification de certaines configurations de leurs segments de définition.

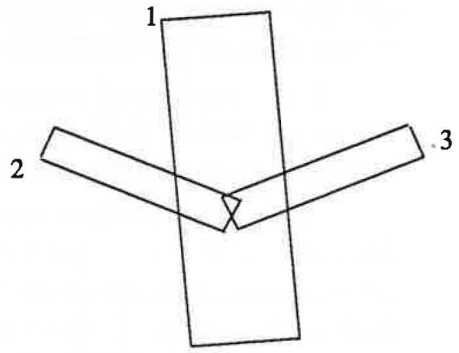


Figure V.51. Intersection de trois *r*-dilatés en un point.

Prenons maintenant le cas d'une intersection comme celle de la figure V.51. Les objets ont la même priorité que leur numéro d'identification. En fait, n'est-il pas souhaitable, dans ce genre de situation, de construire un 'raccord' entre les objets 2 et 3 puisque leurs segments de définition partagent un point et qu'ils ont individuellement priorité sur le troisième? Quand l'extrémité d'un r-3d est située sur la frontière d'un objet (moins prioritaire), ne serait-il pas souhaitable, dans certains cas, de permettre que le r-3d se termine en biseau le long de l'objet, et parfois de faire en sorte qu'il se termine d'une autre manière?

Il est parfois nécessaire de subdiviser les tronçons des r-3d: les facettes que l'on obtiendrait sinon, seraient beaucoup trop effilées. La subdivision consiste à découper chaque segment de définition en un certain nombre de sous-segments: ces nouveaux sommets se projettent en deux points sur les génératrices (comparer à des virages d'angle π) et l'on force ainsi les facettes intérieures aux r-3d à être moins longues. Il est souhaitable de se donner un moyen de prévoir le nombre de subdivisions pour chaque tronçon. Comme la frontière de chaque r-3d est susceptible d'intersecter d'autres objets, on constate que l'obtention de subdivisions équilibrées n'est pas une tâche triviale!

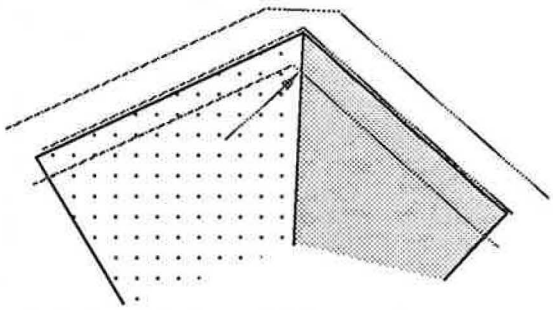


Figure V.52. Bordure mal définie.

Considérons comme dernier exemple la situation suivante, extrêmement dommageable et fréquente: un utilisateur appliqué s'est assuré qu'une route soit bordée par deux champs adjacents. Sachant que les r-3d sont définis par leurs segments de définition, il s'est évertué à représenter les champs par deux polygones ayant un segment en commun et partageant une des extrémités du segment de définition (figure V.52). Malheureusement, n'ayant aucun contrôle sur le développement du r-3d définitif, il ne peut empêcher que le bord de frontière commun des deux champs intersecte la frontière du r-3d en un autre endroit (flèche sur la figure) que le point d'intersection des génératrices. Ceci a la fâcheuse conséquence de créer une facette qui n'a aucune raison d'être.

Il serait fastidieux d'essayer de présenter une solution pour chacun de ces problèmes. Il est bien plus judicieux de présenter une méthode générique, dont on verra comment l'appliquer à chaque cas d'espèce.

La méthode générale s'appuie sur la notion de *graphe sémantique*. Supposons que l'on se donne une structure supplémentaire de graphe abstrait dont tout nœud coïncide avec un nœud unique du graphe planaire cité depuis le début du chapitre, et contient des informations sémantiques: par exemple, on peut y indiquer qu'un sommet est attaché

à un autre sommet ou à un segment par une loi explicite, un segment de r-3d porte l'indication qu'il aboutit sur le bord du domaine et est prolongé dans le domaine voisin par un segment approprié ou bien doit l'être. On peut aussi imaginer des collections d'objets dont la position est l'objet d'une loi géométrique (placer un objet tous les x mètres à la distance y et parallèlement à telle frontière d'objet) ...

Toutes ces indications doivent être contenues de manière explicite dans les données: celles-ci doivent être organisées comme les instructions d'un langage évolué de manière à rendre possibles des références à d'autres objets et l'utilisation de variables manipulant ces objets. Malgré tout, certaines contraintes doivent être imposées: un élément d'un objet ne peut dépendre que d'éléments plus prioritaires, on ne doit pas créer de circuits dans le graphe des relations, etc.

Une fois les données 'interprétées', la première tâche consiste à créer un réseau de contraintes sur le graphe abstrait: les techniques permettant de résoudre ces problèmes ne manquent pas (cf [WIN 84]).

Il est important de noter que les informations relatives à un domaine voisin ne sont peut-être pas disponibles au moment de la résolution des contraintes relatives à un domaine donné: il doit donc exister une relation d'ordre entre les domaines eux-mêmes afin que le *meta-résolveur* de domaines puisse organiser les résolutions de manière cohérente: ce super-module est indispensable pour permettre la cohésion des données et doit être la seule 'autorité' habilitée à pouvoir décider de la validation d'une action mettant la cohésion en danger.

La résolution du réseau de contraintes doit amener à déterminer en premier les relations de dépendances du sous-graphe des r-dilatés. Si un nœud de cette structure correspond à une extrémité de r-dilaté sur un bord de domaine, il est nécessaire de vérifier si le nœud possède une branche 'infinie' dans le domaine voisin: si c'est le cas, il faut vérifier si on est en présence d'un virage ou d'un simple raccord et construire en conséquence. S'il n'existe pas d'information en provenance du domaine voisin, on doit informer le super-module qu'un segment de r-3d de pente p et d'extrémité le nœud courant aboutit *sans virage* sur le bord b du domaine D. La construction autour de ce nœud peut se faire sans risque de problème ultérieur. Pour qu'un virage soit possible le super-module doit en être saisi à l'avance de manière à pouvoir résoudre des dépendances inter-domaniales.

Si un nœud du graphe des r-3d correspond à un 'coin' du domaine, il faut en référer au super-module avant construction (afin de vérifier si la chose est cohérente) et après, afin de fournir les références nécessaires à la validation, par les domaines avoisinants, de l'empiétement d'un objet étranger sur leur territoire: ces domaines ne pourront avoir aucune justification autre que les paramètres et les attributs bruts du morceau de r-3d qu'ils vont accepter.

Dans un même ordre d'idée, si un objet quelconque 'déborde' d'un domaine sur un autre, seul le super-module peut statuer et éventuellement entériner l'état de fait.

Les relations du type 'l'élément A dépend de l'élément B' se résolvent un peu de la même façon que les liens de dépendance numériques au début du chapitre: on crée des liens physiques entre éléments et on parcourt ce sous-réseau en réduisant chaque nœud quand il n'est plus sujet qu'à deux liens sur des nœuds réduits. Il est fondamental de n'autoriser, au maximum, que deux liens de dépendance pour un nœud donné.

Pour reprendre le dernier exemple mentionné, si un sommet de frontière est attaché à deux faces (les deux champs bordant la route) et un sommet de virage de *r*-dilaté, on rencontre deux cas, suivant que les polygones sont situés du côté de la corde du virage ou de son extérieur. Au moment de la construction, le nœud virage devra recenser explicitement tous les nœuds qui dépendent de lui. Au moment de la construction du virage, on déplace les liens de dépendance en provenance du côté corde vers le nœud matérialisant l'intersection des génératrices intérieures, et ceux côté extérieur vers un des nœuds de contour du palier, de manière heuristique.

Il n'y a pas à craindre d'indirections multiples car, au pire, le segment de frontière commune est flottant aux deux extrémités: à cause de la relation d'ordre instaurée et grâce à des techniques semblables à celles du début de chapitre, on parvient sans mal à déterminer les positions définitives de ces deux extrémités, à partir du moment où les données ont été filtrées de manière à ne pas comporter de boucles indéterminables.

Pour affiner l'approche géométrique des problèmes comme ceux de *f*-V.50, on peut imaginer que le nœud où se produit l'intersection des segments de définition contient suffisamment d'information pour permettre de dire que les deux segments se terminent en ce nœud: cela signifie qu'il n'y a ni prolongement ni virage; dans ces conditions, la construction des deux *r*-dilatés peut être reprise de sorte que le plus prioritaire avale le moins prioritaire, ou toute autre solution convenable.

Il n'est pas question ici de donner des solutions ponctuelles à des problèmes dont l'essence dépend avant tout de la signification des objets: si la modélisation porte sur une voie ferrée, une route ou un cours d'eau, les solutions risquent fort d'être dépendantes de chaque cas; il est des cas où la route passe à gué, d'autre ou la route passe sous le cours d'eau (pont canal), au-dessus, etc... Ce qui est important est de faire remarquer que le graphe planaire du début de chapitre peut être investi de beaucoup plus de pouvoirs qu'il n'en était question à l'origine, et que cette propriété ouvre le champ aux investigations les plus diverses.

Un autre exemple significatif concerne les 'carrefours' à plus de deux *r*-dilatés: le sous-graphe des *r*-dilatés plans est rendu planaire en premier et, après cette opération, il est possible de localiser les nœuds où se croisent ces objets: quand plus de deux d'entre eux se rencontrent en un nœud, il devient possible d'organiser un 'rond-point': il s'agit d'une classe d'objets plus prioritaire que toutes les autres, dont le but est d'aplanir toute la région voisine de l'intersection multiple et de

supprimer toutes les faces des *r*-dilatés qui sont recouvertes par son intérieur. Un tel exemple est présenté à la figure V.53.

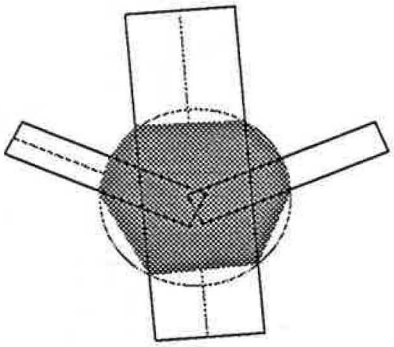


Figure V.53. Rond-point pour la figure V.51.

Si l'on considère, dans un premier temps, qu'un rond-point est modélisé par un disque, les deux points importants sont la recherche de son centre et de son rayon. Le centre est évidemment le point d'intersection des segments de définition. Pour trouver le rayon, on ne peut pas faire mieux que de calculer explicitement les distances des intersections des couples de génératrices 'alternées' (intérieure-extérieure) au centre: le rayon peut être pris comme supérieur à la plus grande de ces distances. On peut ensuite calculer les intersections de ce disque avec toutes les génératrices et relier ces points par des segments pour définir la frontière vectorisée de l'objet rond-point.

Il est à noter que des problèmes se posent concernant les cas où plusieurs *r*-dilatés se coupent de manière très voisine, mais pas en un point unique: on peut associer à un point des relations de dépendances afin de permettre éventuellement d'éviter ce cas en exigeant que tous ces segments se coupent en un point unique, de manière formelle. Malgré tout, le cas général d'un 'carrefour' est encore mal compris et nécessite des travaux plus poussés.

Revenons au problème de la subdivision de tronçons de r-3d: comme il est à craindre, les r-3d représentent souvent des objets très fins et très longs, dont on voit sans peine que les facettes vont être extrêmement effilées. On peut imaginer de subdiviser chaque tronçon (entre deux virages ou un virage et une extrémité finale) en autant de sous-tronçons destinés à provoquer l'apparition de deux points de projection sur les génératrices et ainsi assurer une augmentation des facettes et donc une meilleure harmonie dans leur apparence. Le seul véritable problème est que la frontière d'un *r*-dilaté peut très bien être elle-même coupée par d'autres objets, ce qui risque d'entrer en conflit avec la subdivision et provoquer l'apparition de micro-facettes indésirables.

La solution consiste à n'effectuer la subdivision qu'après l'opération de décomposition. D'une certaine manière, les arêtes canoniques indiquent une dépendance implicite entre les objets: par exemple, si un segment de frontière de *r*-dilaté est partagé avec la frontière d'un autre objet (champ qui borde la route), l'arête canonique correspondante permet de savoir que toute transformation apportée au segment de r-3d

doit se répercuter sur le *seul* autre segment polygonal valide représenté par cette même arête canonique après décomposition!

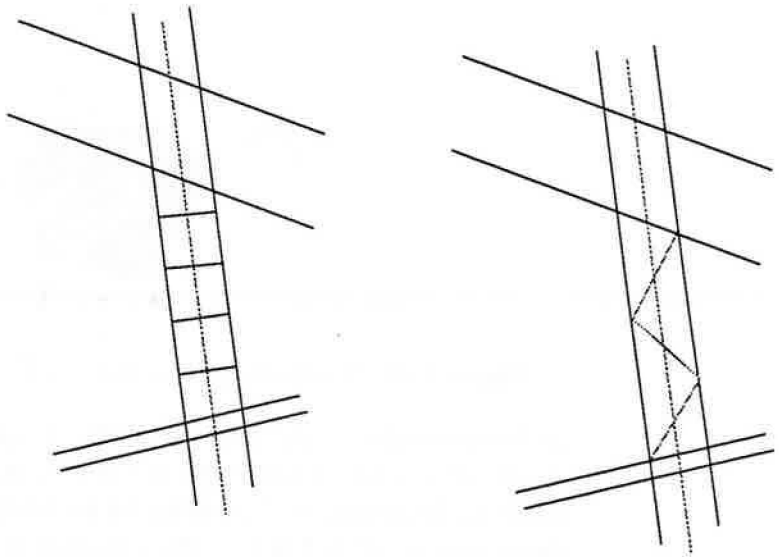


Figure V.54. Deux techniques de subdivision.

Mais il est nécessaire d'avoir gardé des liens de fraternité entre les deux génératrices et leurs segments après intersection/décomposition afin de pouvoir organiser une subdivision 'synchrone' des deux sections: on peut choisir de subdiviser par le biais des segments de définition originaux ou bien en alternance sur les deux génératrices, afin de préparer le terrain, encore une fois, pour la triangulation. La figure V.54 donne un exemple de chaque cas.

Le dernier problème que nous aborderons ici est celui des intersections indues: l'opération de décomposition par priorité a la fâcheuse conséquence d'entraîner la disparition de certaines faces. Prenons l'exemple le plus simple que l'on puisse imaginer (figure V.55), dans lequel un polygone P est inclus dans un polygone plus prioritaire Q: l'opération d'intersection introduit les sommets A et B sur la frontière de Q et celle de décomposition fait disparaître le polygone P tout entier.

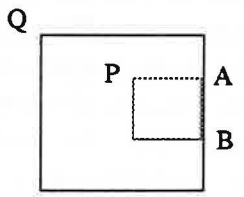


Figure V.55. Sommets zombies.

Les sommets A et B subsistent sur la frontière de Q bien qu'ils n'aient plus aucune raison d'exister. Il se peut, malgré tout, qu'il ait été volontaire de placer P de cette sorte *afin* de faire apparaître ces sommets: si l'on imagine des facétisations à différents niveaux de détail, certains points doivent subsister d'un niveau de facétisation à un autre, afin de permettre des raccords entre plusieurs plans de visualisation. De même les sommets rajoutés par subdivision des tronçons de r-3d ne doivent certainement pas disparaître.

Il est donc nécessaire de pouvoir faire la distinction entre sommets *forcés* et normaux. Ceci peut se faire par le biais d'une information booléenne supplémentaire dans le graphe de dépendance sémantique et une information de dépendance analytique entre deux segments subdivisés: dans une intersection entre deux segments, on sait que l'on obtient deux fois deux sous-segments de même support; il est possible d'adjoindre à chaque sous-segment un lien de co-linéarité avec son sous-segment associé.

Après la décomposition, on peut effectuer une passe supplémentaire sur la frontière de chaque composante connexe construite en ne s'intéressant plus qu'aux couples de segments successifs liés par une relation de co-linéarité: si le sommet commun à deux tels sous-segments n'a pas d'autres arêtes canoniques incidentes *valides* que celles qui leurs sont relatives, et s'il n'est pas de type forcé, le sommet peut être supprimé et les sous-segments 'fusionnés' en un sur-segment unique.

Radicaux

La création des r-dilatés s'accompagne de problèmes de calculs assez conséquents. D'une manière schématique, la recherche de l'équation des génératrices d'un tronçon revient à chercher les deux droites parallèles au support de segment de définition à une distance r de celui-ci.

Ainsi, si l'on cherche l'équation des génératrices situées à une distance r d'un support de vecteur directeur $(-v, u)$ passant par le point (x_0, y_0), on trouve

$$u(X-x_0) + v(Y-y_0) \pm r\sqrt{u^2+v^2} = 0 \qquad \text{(V-9)}$$

Ceci implique l'introduction de radicaux dans les coefficients des équations et la disponibilité d'une arithmétique rationnelle n'est pas suffisante. De même, les conditions de validité de construction données au début de cette section font intervenir des calculs non rationnels.

Il existe deux façons de considérer ce problème: comme les r-dilatés, par construction sont 'faits par l'homme' on peut considérer que les radicaux de l'équation précédente, qui représentent *fondamentalement*, des intercepts, peuvent être approchés, avant tout autre calcul, par des pseudo-réels bien choisis: on est sûr que les objets ne seront pas *exactement* r-dilatés mais dès qu'un radical est approché, il n'apparaît plus explicitement dans les calculs et est remplacé par un rationnel suffisamment proche, en-dessous d'un seuil de précision décidé: les sommets des r-dilatés se comportent comme de simples sommets ordinaires, situés qu'il sont sur des segments dont les supports ont maintenant des équations rationnelles.

Une autre solution consisterait à utiliser une approximation des nombres utilisés par des fractions continues (rationnelles). Ce genre de technique permet d'approcher aussi près que souhaité tout nombre solution d'une équation du second degré à coefficients entiers, ce qui est le cas des nombres présents dans les équations de génératrices des r-dilatés.

Il est possible de comparer de manière exacte une fraction continue et un rationnel pur: deux convergents successifs d'une même fraction continue encadrent ce nombre dans un intervalle de longueur inférieure à $|1/rs|$ si r et s sont les dénominateurs des deux convergents. Comme le rationnel admet une seule représentation irréductible (et finie), on parviendra toujours à pousser la convergence assez loin pour pouvoir explicitement localiser le rationnel par rapport à cet intervalle d'encadrement, sans erreur.

La comparaison de deux fractions continues relatives à deux solutions d'une équation du second degré est possible en utilisant deux propriétés fondamentales des approximations par fractions continues: si x est un nombre irrationnel quelconque, il existe une suite *unique* de naturels $a_0, a_1, \ldots, a_n, \ldots$, tels que

(V-10)
$$x = \lim_{n\to\infty} [a_0, a_1, \ldots, a_n, \ldots] = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \ \ldots}}$$

De plus, si x est racine d'une équation du second degré à coefficients entiers, la suite précédente est périodique. Un grand nombre de résultats concernant les fractions continues se trouvent dans [H/W 79] et [ROB 77]. Le résultat précédent permet de conclure qu'on peut comparer deux racines d'une équation du second degré à coefficients entiers en comparant leurs expansions périodiques, ce qui donne un algorithme fini, dans ce cas.

Malheureusement, les nombres engendrés par la présence de r-3d ne sont pas toujours comparables par un algorithme fini: toute combinaison linéaire de deux racines d'une équation du second degré à coefficients entiers ne donne pas nécessairement un nombre qui ait aussi cette propriété. Il semble donc difficile d'étendre les propriétés précédentes aux nombres rencontrés dans un contexte comme celui qui nous préoccupe, car il se peut que l'on ne puisse prouver qu'ils sont égaux qu'en comparant explicitement tous les entiers de leurs expansions en fractions continues.

La recherche de solutions dans ce cadre est un prolongement naturel et à venir des travaux de cette thèse.

VI. Facetisation Avec Contraintes

Ce dernier chapitre porte sur le problème de la facétisation des représentations par segments de graphes planaires (rsgp) obtenues par les méthodes développées depuis le début de cette thèse.

La méthode de facétisation est fondée sur une triangulation particulière dite 'de Delaunay', elle-même fortement liée aux diagrammes de Voronoi dans le plan.

Quelques définitions sur ces deux notions fondamentales relatives à un nuage de points du plan sont présentées, ainsi qu'un nouvel algorithme de construction des diagrammes de Voronoi dans le plan: celui-ci permet de donner un algorithme simple et linéaire de construction de tels diagrammes lorsque les points sont tous disposés sur la frontière d'un polygone monotone.

On étudie ensuite la triangulation de Delaunay de la représentation par segments d'un graphe planaire, en donnant une méthode incrémentale de complexité optimale, et enfin une spécialisation de ce genre de techniques lorsque les rsgp possèdent une structure hiérarchique, comme celles qui ont été étudiées au chapitre précédent.

. REFERENCES BIBLIOGRAPHIQUES POUR LE CHAPITRE VI .

A/G/S/S 89 A. Aggarval, L.J. Guibas, J. Saxe, P.W. Shor, *A linear-time Algorithm for Computing the Voronoi Diagram of a Convex Polygon*, Discrete and Computational Geometry, 4, pp. 591-604, 1989.

A/H/U 87 A. Aho, J. Hopcroft, J. Ullman, *Structures de données et Algorithmes*, Inter-Editions, Paris, traduction de J.M. Moreau, 1987. Version originale: *Data Structures and Algorithms*, Addison Wesley, Reading, Mass., 1983.

BRO 79 K. Brown, *Voronoi Diagrams from Convex Hulls*, Inf. Proc. Letters, Vol. 9, N° 5, 1979.

CHA 82 B. Chazelle, *A Theorem on Polygone Cutting with Applications*, Proc. of the 23rd IEEE Annual Symposium on Foundations of Computer Science, pp. 339-349, 1982.

CHE 87 L.P. Chew, *Constrained Delaunay Triangulations*, Proc. of the Third Annual Symposium on Computational Geometry, 1987. Version plus détaillée: *Constrained Delaunay Triangulations*, Algorithmica, 4: pp. 97-108, 1989.

DeF 87 L. De Floriani, *Surface Representations Based on Triangular Grids*, Visual Computer, 3: 27, pp. 27-50, 1987.

DF/F/P 85 L. De Floriani, B. Falcidieno, C. Pienovi, *Delaunay-based Representation of Surfaces Defined over Arbitrarily Shaped Domains*, Computer Vision, Graphics and Image Processing, 32, pp. 127-140, 1985.

EDE 87 H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, Springer Verlag, New York, 1987.

FOR 86 S. Fortune, *A Sweepline Algorithm for Voronoi diagrams*, Proceedings of the 2nd ACM Symposium on Computational Geometry, pp. 313-322, 1986.

G/K/L/N 83 I.G. Gowda, D.G. Kirkpatrick, D.T. Lee, A. Naamad, *Dynamic Voronoi Diagrams*, IEEE Trans. on Inf. Theory, Vol IT-29, N° 5, 1983.

G/S 85 L. Guibas, J. Stolfi, *Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams*, Proceedings of the 15th ACM Conference on the Theory of Computing, pp. 221-234, 1983. Version révisée, mêmes auteurs, même titre, ACM Transactions on Graphics, Vol 4. No. 2, April 1985, pp. 74-123.

KIR 79 D.G. Kirkpatrick, *Efficient Computation of continuous Skeletons*, Proc. of the 20th IEEE Symposium on the Foundations of Computer Science, pp. 18-27, 1979.

K/L 86 A.K. Lin, D.T. Lee, *Generalized Delaunay Triangulation for Planar Graphs*, Discrete and Computational Geometry, Vol. 1, N° 3, pp. 201-217, 1986.

KM3 84 K. Mehlhorn, *Data Structures and Algorithms: Multi-dimensional Searching and Computational Geometry*, Vol. 3, Springer Verlag, Berlin, 1984.

LAW 77 C.L. Lawson, *Software for C^1 Surface Interpolation*, Math. Software III, (J.R. Rice, Ed.), pp. 161-194, Academic Press, New York, 1977.

LEE 78 D.T. Lee, *Proximity and Reachability in the Plane*, Ph.D. Thesis, Rep. R-831, Coordinated Sci. Lab. Univ. Illinois, Urbana, Ill., 1978.

LEE 82 D.T. Lee, *On k-nearest neighbor Voronoi Diagrams in the Plane*, IEEE Trans. Comput. C-31, pp. 478-487, 1982.

PER 88 B. Péroche, *Modèles de partitionnement par diagramme de Voronoi: aspects algorithmiques*, E.N.S.M.S.E. 88-7, 1988.

P/S 85 F.P. Preparata, M.I. Shamos, *Computational Geometry - an Introduction*, Springer Verlag, New York, 1985.

S/W 87 L. Schubert, C.A. Wang, *An optimal Algorithm for Constructing the Delaunay Triangulation of a Set of Line Segments*, Proc. of the Third Annual Symposium on Computational Geometry, pp. 223-232, 1987.

SHA 78 M.I. Shamos, *Computational Geometry*, Ph.D. Dissertation,Dept. Comput. Sci. Yale University, New Haven, Conn., 1978.

VOR 08 G. Voronoi, *Nouvelles applications des paramètres continus à la théorie des formes quadratiques: recherches sur les parallèloèdres primitifs*, J. Reine Angew. Math. 134, pp. 198-287, 1908.

YAP 84 C.K. Yap, *An O(nlogn) Algorithm for the Voronoi Diagram of a set of simple curve segments*, Technical Report, Courant Institute, New York University, 1984. Version révisée: *An O(nlogn) Algorithm for the Voronoi Diagram of a set of simple curve segments*, Discrete and Computational Geometry, N°2, pp. 365-393, 1987.

VI.A. DIAGRAMMES DE PROXIMITE

On appellera ainsi dans ce chapitre une partition du plan par des régions convexes possédant des propriétés géométriques fortement liées à la notion de proximité de sommets de ce plan. Les deux exemples que l'on considérera sont les *diagrammes de Voronoi* et les *triangulations de Delaunay.*

La notion de diagramme de Voronoi a été traitée en détail dans plusieurs ouvrages ou articles fondamentaux auxquels le lecteur totalement étranger au sujet est expressément renvoyé. Parmi ceux-ci, citons [VOR 08], [SHA 78], [KM3 84], [P/S 85], [EDE 87], [PER 88] et [G/S/S 89].

On y trouvera l'ensemble des propriétés et particularités de ces objets (en deux dimensions et au-delà) et la présentation des techniques les plus représentatives liées à leur construction et leur manipulation.

Le but de cette section n'est pas de redire ce qui est par ailleurs parfaitement déjà dit, mais d'essayer de dégager des points-force qui permettront de développer de nouvelles approches sur le processus de triangulation de Delaunay d'une rsgp.

Néanmoins, afin de permettre une meilleure compréhension des algorithmes proposés, le chapitre débute par une section de récapitulation des propriétés utilisées par la suite et un aperçu de la méthode 'diviser-pour-résoudre'.

VI.A.1 Diagrammes de Voronoi

Définitions et propriétés générales

Soit S_n un ensemble de n points distincts du plan $\boldsymbol{P}$ qui porteront le nom de *sites*. On suppose que dans cet ensemble on ne trouve pas trois sites alignés ni quatre sites sur un même cercle: on dit qu'ils sont *en position générale.*

Ces précautions permettent seulement de simplifier les démonstrations, voire d'unifier les solutions: elles ne sont en aucun cas obligatoires et les solutions générales exigent seulement une attention plus soutenue pour le traitement des cas 'dégénérés'.

On suppose que ces sites ont été triés suivant l'ordre lexicographique de leurs coordonnées dans $\boldsymbol{P}$ et renumérotés $\{p_1, p_2, \ldots, p_n\}$, selon l'ordre dans lequel ils sont maintenant rangés. L'ensemble des i premiers sites (ordonnés) $\{p_1, p_2, \ldots, p_i\}$ sera toujours noté S_i, $i \in [1, n]$.

Si p_i et p_j sont deux sites quelconques de S_n, on notera p_{ij} ou éventuellement $[i, j]$ le segment $[p_i, p_j]$ et M_{ij} sa médiatrice. Sur les *graphiques* de ce chapitre, une telle droite sera parfois notée 'i,j'.

La médiatrice M_{ij} définit deux demi-plans fermés $\prod_{i,j}$ et $\prod_{j,i}$ tels que le premier contient tous les points de $\boldsymbol{P}$ qui sont plus proches de p_i que de p_j (ou à égale distance) et le second tous ceux plus proches de p_j que de p_i (ou à égale distance). Voir la figure VI.1.(a).

Comme les sites de S_n sont en position générale, si l'on en prend trois quelconques p_i, p_j et p_k, les médiatrices des segments p_{ij}, p_{ik} et p_{jk} se coupent nécessairement en un point unique ϖ_{ijk}, centre du cercle circonscrit aux trois points (Figure VI.1.(b)).

Un tel point sera appelé *sommet de Voronoi*, et l'on peut affirmer, par conséquent, que tous les sommets de Voronoi engendrés par S_n sont de degré trois. On voit d'ailleurs que cette propriété disparaîtrait si l'on autorisait que quatre sites ou plus puissent se trouver sur un même cercle.

Si p_i, p_j et p_k, $i \neq j$, $i \neq k$ et $j \neq k$, sont trois sites quelconques de S_n, l'intersection de $\Pi_{i,j}$ et $\Pi_{i,k}$, donne *nécessairement* une région infinie et convexe, V(i), qui a la propriété importante suivante:

(VI-1) $$\forall z \in V(i),\ d(z, p_i) \leq d(z, p_t)\ t \in \{j, k\}$$

En d'autres termes, tout point de la région V(i) est plus proche de p_i que de p_j ou de p_k et ceci s'applique, circulairement, à V(j) et V(k). La région V(i) s'appelle *le polygone de Voronoi* du site p_i et les sections de médiatrices qui en forment la frontière s'appellent des *arêtes de Voronoi.*

Le 'partitionnement' de $\boldsymbol{P}$ en $V(i) \cup V(j) \cup V(k)$ s'appelle le *diagramme de Voronoi* de l'ensemble $\{p_i, p_j, p_k\}$, noté $DV(\{p_i, p_j, p_k\})$. D'une manière générale, si $S_n = \{p_1, p_2, \ldots, p_n\}$, le diagramme de Voronoi de S_n, $DV(S_n)$, est le partitionnement du plan par l'union des polygones de Voronoi V(p_i), contenant tous les points de $\boldsymbol{P}$ situés à une distance de p_i inférieure ou égale à celle de tout site de S_n-$\{p_i\}$.

Chez certains auteurs, on trouve aussi les termes de 'tessellation de Dirichlet' ou 'tessellation de Thiessen' pour désigner les diagrammes de Voronoi.

On démontre assez facilement que, pour tout site p_i de S_n, le polygone de Voronoi V(i) est convexe, qu'il est ouvert si et seulement si p_i est un sommet de la frontière[1] de l'*enveloppe convexe* de S_n, $Fr(EC(S_n))$ et fermé-borné sinon, et enfin que ses arêtes sont des segments finis ou non de médiatrices de couples de sites de S_n: chaque polygone de Voronoi non borné admet deux *rayons* correspondants à deux demi-droites infinies, médiatrices de deux segments de l'enveloppe convexe.

Tous ces résultats, démontrés en détail dans les ouvrages ou articles cités en référence, sont considérés acquis ici et illustrés pour mémoire à la figure VI.2.

Si l'on considère un point q d'une arête quelconque d'un polygone de Voronoi, ce point est, par construction, à égale distance de deux éléments de S_n; cela signifie, à l'inverse, qu'il n'y a pas d'autre point de S_n que ceux-ci qui puisse être plus proche de q. Par suite, la liste des sommets du polygone de Voronoi V(i) permet de connaître les *plus proches voisins* de p_i dans S_n.

[1] On considère ici que l'enveloppe convexe d'un nuage de points est le plus petit polygone convexe contenant ces points et non pas la frontière de cet ensemble!

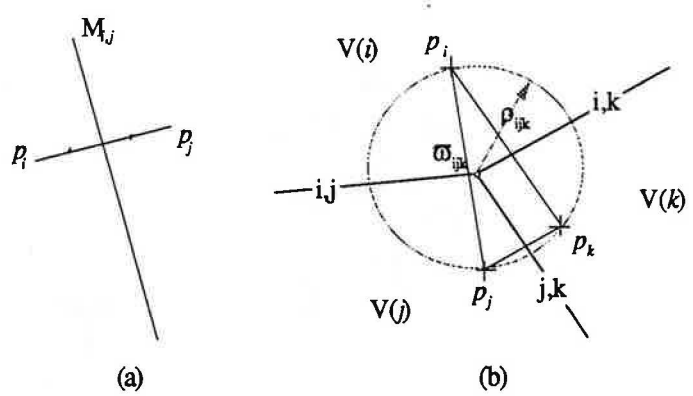


Figure VI.1. Diagramme de Voronoi sur deux (a) et trois (b) sites.

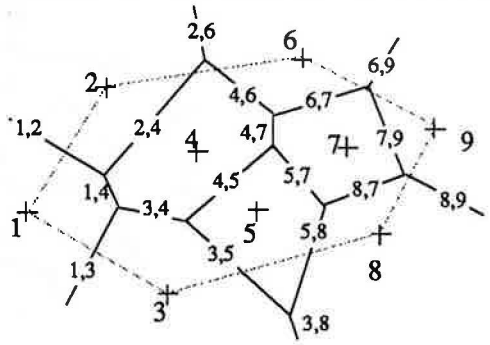


Figure VI.2. Diagramme de Voronoi sur 9 sites.

Construction par fusion

[SHA 78] expose une méthode très ingénieuse de construction d'un diagramme de Voronoi sur n sites en position générale. Cette méthode porte le nom de 'diviser-pour-résoudre' (*divide and conquer*) et elle portera ici souvent celui de *construction par fusion.*

Son principe, extrêmement simple, est l'équivalent des algorithmes de tri par fusion: on divise le vecteur à trier en deux parties égales, on appelle récursivement le tri par fusion sur ces deux parties qui sont retournées triées; il ne reste plus qu'à parcourir les deux listes triées en prenant l'élément minimum (ou maximum, suivant le sens du tri) à chaque étape. Cette opération se nomme la *fusion* et prend un temps proportionnel à la longueur totale de la liste triée.

Si la liste à trier contient n éléments, la complexité $T(n)$ de l'algorithme est donnée par l'équation de récurrence suivante:

$$T(n) = 2T(n/2) + cn,$$

où c est une constante positive. Une telle équation se résout par substitution [A/H/U/ 87, p. 303] et permet de trouver que le tri par fusion de n éléments a une complexité de $O(n\log n)$.

Dans le cas qui nous concerne, on subdivise l'ensemble S_n en deux sous-ensembles G et D (pour gauche et droit) de tailles sensiblement égales, séparables par une droite verticale, et on lance récursivement la procédure de construction sur chacun, ce qui donne deux sous-diagrammes de Voronoi DVG et DVD; on effectue enfin la 'fusion' de ces deux ensembles pour obtenir le diagramme de Voronoi de S_n.

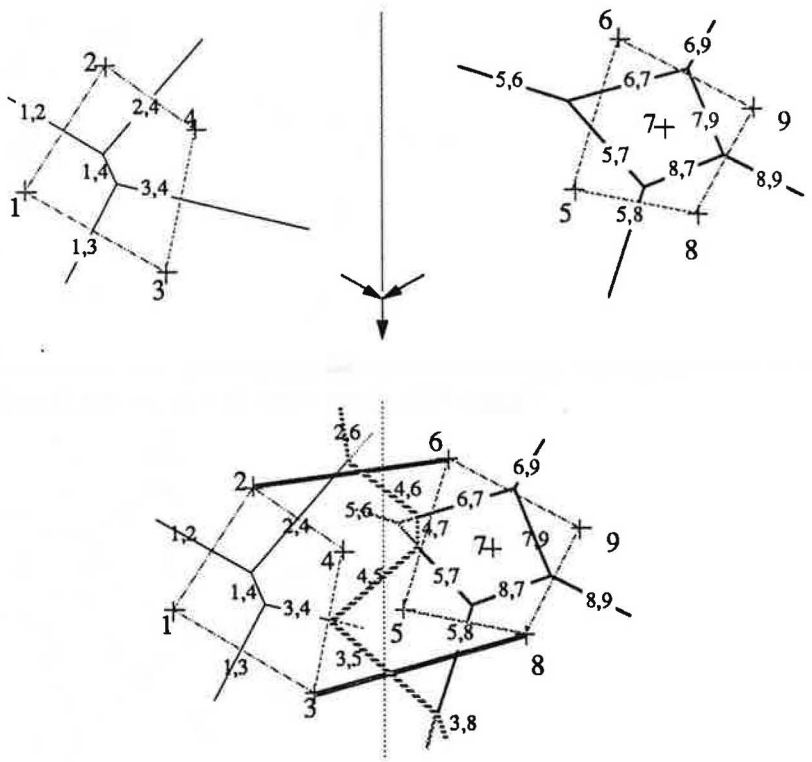


Figure VI.3. Fusion de deux sous-diagrammes de Voronoi.

La récursion s'arrête lorsque l'ensemble passé en paramètre a une taille inférieure ou égale à trois, car on peut construire les diagrammes de Voronoi de ce type en temps constant (cf. f-VI.1).

La phase de fusion est assez délicate et il est nécessaire d'en donner l'essence, afin de permettre la bonne compréhension des développements des prochaines sections. Supposons que l'on parte de l'ensemble de neufs sites de f-VI.2 et, qu'après avoir subdivisé cet ensemble en $\{p_1, \ldots, p_4\}$ et $\{p_5, \ldots, p_9\}$, on ait récursivement obtenu la construction des deux sous-diagrammes de Voronoi en haut de la figure VI.3.

Ces deux sous-diagrammes sont en quelque sorte 'dédoublés' mais ne sont pas dans leur véritable position relative: on voit dans le bas de la figure qu'il faut leur appliquer une translation parallèle à l'axe horizontal pour qu'ils retrouvent leurs véritables positions relatives dans le plan. Malgré tout, ceci permet de mieux visualiser l'ensemble de l'opération de fusion.

On commence par localiser les deux segments d'appui p_{26} et p_{38} permettant de fusionner les enveloppes convexes des deux sous-ensembles de sites en une seule (cf [P/S 85] ou [EDE 87] pour de plus amples détails sur la construction et la fusion d'enveloppes convexes).

On construit ensuite la 'chaîne de fusion' qui est la ligne de partage des eaux entre les deux sous-diagrammes: c'est l'unique chaîne polygonale qui reste continuellement à égale distance d'un site du sous-diagramme gauche et d'un site du sous-diagramme droit.

Cette chaîne commence et finit par les deux médiatrices des segments d'appui. M_{26} rencontre en premier[2] M_{24} de DVG, tout en restant à l'intérieur du polygone de Voronoi V(6) de DVD: cela signifie qu'à ce point de rencontre la chaîne est exactement à égale distance de p_2, p_4 et p_6. Comme elle s'apprête à franchir la frontière de V(4), la chaîne va quitter V(2) et devenir plus proche de p_4 que de p_2: on peut donc construire la médiatrice M_{46} et l'insérer comme nouveau chaînon de la chaîne de fusion. Parallèlement, la médiatrice M_{24} de V(2) est réduite jusqu'à concurrence du point d'intersection mentionné et le nouveau rayon infini de ce polygone devient M_{26}. Le résultat de l'opération de fusion donne, naturellement, le diagramme de *f*-VI.2.

On peut montrer [LEE 82] qu'il existe une 'loi d'addition' gouvernant les indices des régions fusionnées. Contentons-nous d'observer qu'elle se traduira ici par: fusion(M_{ij}, M_{ik}) → M_{jk}.

Remarquons aussi que la chaîne de fusion a un comportement monotone: tant qu'elle reste à l'intérieur d'un polygone de Voronoi d'un sous-diagramme alors qu'elle franchit la frontière séparant deux polygones de Voronoi du second sous-diagramme, la chaîne de fusion garde la même 'concavité': dans le cas précédent, elle est convexe si l'on suppose que le le plan est doté de l'orientation naturelle. Elle deviendra concave lorsqu'elle franchira la frontière séparant deux polygones de VDD en restant à l'intérieur du même polygone de VDG.

C'est exactement ce qui se passe à l'étape suivante: M_{46}, le chaînon de fusion courant, rencontre en premier M_{67} de DVD: la fusion donne donc M_{47} et le prédécesseur, dans le parcours direct des arêtes de V(4), devient le segment correspondant de M_{46}. On amorce donc une révolution dans le sens rétrograde autour de p_4 puisque la chaîne reste à l'intérieur de V(4) et franchit ensuite M_{57} de DVD. Le nouveau chaînon est donné par fusion (M_{47}, M_{57}) → M_{45}. Cette médiatrice rencontre ensuite M_{34} de DVG pour donner M_{35} qui rencontre à son tour M_{58} pour terminer en M_{38} dont on sait qu'elle constitue le rayon final bas.

Si l'on essaie de formaliser l'algorithme correspondant à cette méthode, il est utile de supposer l'existence d'une fonction permettant de construire la médiatrice d'un segment donné. Cette médiatrice est, *a priori*, une droite infinie, mais peut aussi être une demi-droite orientée 'd'origine' A, une demi-droite orientée 'd'extrémité' A ou un segment fini [A,B]. Pour distinguer ces cas, il est pratique, en plus de ceux nécessaires aux coefficients de l'équation du support, de disposer de deux champs qui sont des pointeurs sur des nœuds de l'arbre binaire des points du graphe sous-jacent.

Les quatre situations précédentes se traduisent par les quatre couples (NIL, NIL), (A, NIL), (NIL, A) et (A, B). L'opération Médiatrice(U,V,X,Y) construit la médiatrice de [U,V] et associe les pointeurs X et Y d'extrémités. L'opération '←' permet d'associer une extrémité finale ou originale à une droite ou à une demi-droite existante. Si l'un des arguments de cette opération est ?, il ne joue pas de rôle.

---

[2] Dans ce contexte, cette expression signifie 'avant de rencontrer une autre médiatrice d'un quelconque des deux sous-diagrammes'.

La synthèse de l'algorithme, inspirée de [KM3 84, p. 109], peut s'écrire ainsi:

```
base ← segment d'appui haut [G,D];
chaînon_fusion ← Médiatrice(G,D,NIL,NIL);
tant que ((Ω ← (chaînon_fusion ∩ (DVG ∪ DVD) ≠ {})
    chaînon_fusion ← (?,Ω);
    soit m la médiatrice intersectée par chaînon_fusion en premier;
    m ← (?,Ω);
    si (m est dans DVG)                    ¡m est la médiatrice d'une arête [G, G']¡
      G ← G';
    sinon                                  ¡m est la médiatrice d'une arête [D, D']¡
      D ← D';
    chaînon_fusion ← Médiatrice(G,D,Ω,NIL);
retourner(DVG ∪ DVD);
```

Algorithme VI-1: Fusion de deux sous-diagrammes de Voronoi [P/S 85]

On démontre ([SHA 78], [PER 88]) que la chaîne de fusion étant monotone, la recherche des intersections successives de la chaîne de fusion et des arêtes de Voronoi gauches et droites reste linéaire, et donc que le mécanisme de fusion a une complexité proportionnelle au nombre d'arêtes du diagramme fusionné et que celle de l'algorithme est $O(n\log n)$, ce qui constitue une complexité optimale.

Schématiquement, on doit trier les points pour pouvoir les diviser récursivement en deux parties de tailles égales et séparables par une droite; cette phase a donc une complexité dans le pire des cas de $O(n\log n)$. D'autre part l'étape de fusion a une complexité proportionnelle au nombre de sites du diagramme fusionné. Ceci permet d'obtenir même relation de complexité que pour le tri par fusion.

Remarque: Il faut rajouter à l'algorithme de fusion précédent l'infrastructure de chaînage et de 'dualisation' des arêtes découvertes par la ligne de partage des eaux: les polygones de Voronoi sont alternativement amputés de sections d'arêtes, reçoivent de nouveaux rayons ou voient leurs frontières rectifiées. Ceci doit être harmonieusement mis en œuvre et il est bien connu qu'il y a une grande distance entre la synthèse précédente et une implantation réelle.

Néanmoins, c'est la modélisation VI.i qui sera utile dans la suite de ce chapitre et il suffit de savoir que la mise à jour des structures représentant les polygones de Voronoi n'est pas de nature à changer la complexité théorique de l'algorithme.

C'est pourquoi elle est ignorée ici, pour ne pas cacher les principes qui seront les plus importants par la suite.

VI.A.2 Triangulation de Delaunay

La triangulation de Delaunay d'un nuage de points constitue la structure de base de l'étude de la triangulation avec contraintes que l'on veut mener dans la dernière section de cette thèse. Comme une triangulation de *Delaunay* est avant tout une *triangulation*, il est bon de jeter quelques bases sur les triangulations de type quelconque.

Définitions et propriétés générales

Soit[3] un graphe $G = (S, A)$. On appelle *cycle élémentaire* de ce graphe un cycle qui n'en contient aucun autre et *corde* d'un cycle élémentaire, une arête qui relie deux sommets non consécutifs de ce cycle.

On dit que le graphe planaire G est *triangulé* si tout cycle de longueur strictement supérieure à trois admet une corde et qu'il est *complètement triangulé* s'il contient un nombre d'arêtes maximal. Ceci équivaut à dire que l'ajout d'une arête supplémentaire détruit la planarité du graphe.

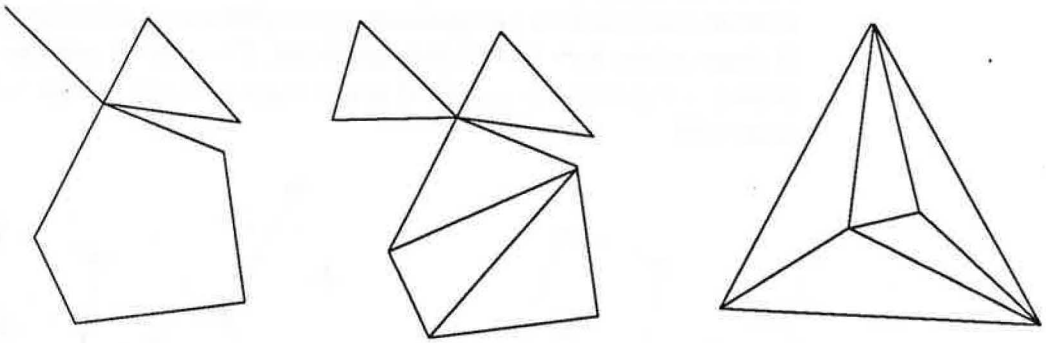


Figure VI.4. Graphes planaires et triangulations.

On montre que les faces bornées de toute représentation par segments d'un graphe triangulé sont des triangles: une telle représentation porte le nom de *triangulation.*

Cette triangulation T = (S, F, A) est constituée d'un ensemble de sommets S, d'un ensemble F de faces – dont la face externe F_{ext} (qui est la seule face non-triangulaire) – et d'un ensemble d'arêtes A. La triangulation est *complète* si tout segment n'appartenant pas à A joignant deux points quelconques de S coupe au moins une arête de A ailleurs qu'en ses extrémités. Une triangulation est donc complète si et seulement si le graphe sous-jacent est complètement triangulé.

La figure VI.4.(a) montre une rsgp qui n'est pas une triangulation, f-VI.4.(b) une triangulation et f-VI.4.(c) une triangulation complète: il est impossible de rajouter des arêtes (curvilignes ou rectilignes) joignant des sommets non déjà connectés de ce graphe sans détruire sa planarité.

Toute triangulation de cette sous-section portera sur un ensemble S égal à l'ensemble de sites S_n défini précédemment et sujet aux restrictions que l'on sait. L'ensemble A des arêtes est donc un sous-ensemble de $S_n{*}S_n$ et les faces sont des polygones (triangles) dont la frontière est définie par des arêtes de A.

[3] Je voudrais tout spécialement remercier R. Jégou du Département d'Informatique Appliquée de l'E.N.S.M.S.E. pour sa précieuse aide et les longues discussions que nous avons eues sur la manière dont il faudrait généraliser certaines définitions parfois trop restrictives.

Triangulation d'une rsgp

Il est naturel de penser à transposer la définition précédente à une représentation de graphe planaire par segments: si H est une rsgp, de face externe FH_{ext}, on dit que T_H est une *triangulation* (complète ou non) de ce graphe *selon sa face externe* si le complémentaire de FH_{ext} est une triangulation (complète ou non).

Il est évident, dans la définition initiale, que la face externe d'un nuage de points est simplement le complémentaire de son (polygone) enveloppe convexe et donc que les deux définitions coïncident, dans ce cas. D'autre part, si la face externe d'une rsgp n'est pas explicitement connue, on définit sa triangulation *par défaut* comme la triangulation de la rsgp selon son enveloppe convexe. Dans tous les cas, on peut considérer que la triangulation d'une rsgp quelconque est définie sans ambiguïté.

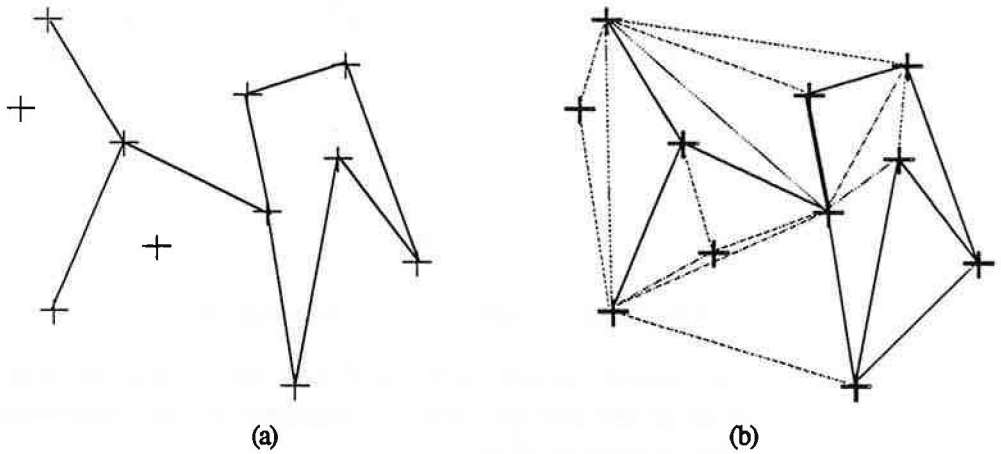


Figure VI.5. Rsgp (a) et triangulation 'par défaut' (b).

Il est important de constater que cette extension de la notion de triangulation à une rsgp implique, évidemment, que les arêtes de la rsgp soient toutes des arêtes de triangles et qu'aucune arête d'un triangle ne coupe une arête initiale de la rsgp ailleurs qu'en une extrémité. Un exemple de triangulation par défaut de rsgp est donné à la figure VI.5. On constate que cette triangulation ne correspond pas à un graphe complètement triangulé, car il est possible de rajouter des arêtes non rectilignes en différents sommets de la frontière de la face externe (enveloppe convexe) sans détruire la planarité du graphe.

Dénombrement

La face externe joue un rôle primordial dans le dénombrement des arêtes et des faces d'une triangulation. Si l'on associe, à chaque arête d'une triangulation, une flèche pointant vers la face située dans un demi-plan défini par son support et une autre pointant vers la face contenue dans le second, chaque triangle reçoit 3 flèches, chaque arête porte deux flèches et la face externe reçoit autant de flèches qu'elle compte d'arêtes. Si l'on note n le nombre de sommets du graphe, n_A le nombre d'arêtes du graphe, n_{ext} le nombre d'arêtes de la face externe,

n_T le nombre de triangles et $n_F = n_T+1$ le nombre total de faces, on trouve donc la relation

(VI-2) $$2n_A = n_{ext} + 3n_T = n_{ext} + 3(n_F-1)$$

Alliée à la relation d'Euler

(VI-3) $$n + n_F - n_A = 2,$$

la première relation permet d'écrire:

(VI-4) $$\begin{cases} n_T = 2(n-1)-n_{ext} \\ n_A = 3(n-1)-n_{ext} \end{cases}$$

Lorsque la triangulation est complète, la face externe est bordée par un triangle et tous les autres points sont à l'intérieur de celui-ci, donc

(VI-5) $$n_{ext} = 3 \Rightarrow \begin{cases} n_T = 2n-5 \\ n_A = 3n-6. \end{cases}$$

Lorsque tous les sommets du graphe se trouvent sur le bord de la face externe on trouve

(VI-6) $$n_{ext} = n \Rightarrow \begin{cases} n_T = n-2 \\ n_A = 2n-3 \end{cases}$$

C'est le cas, entre autres, des polygones simples sans trous, que l'on retrouvera plus loin. En tout état de cause, on peut affirmer que le nombre d'arêtes et le nombre de triangles d'une triangulation $T(S_n, A, F)$ est toujours $O(n)$ puisque l'on peut écrire

(VI-7) $$\begin{cases} n-2 \le n_T \le 2n-5 \\ 2n-3 \le n_A \le 3n-6 \end{cases}$$

Les minorants correspondent au cas où les sommets du graphe sont tous disposés sur la frontière de la face externe (polygone sans trou) et les majorants au cas du graphe complètement triangulé, c'est-à-dire dont tous les points sont inclus dans le complémentaire de la face externe, qui est un triangle (f-VI.4.(c)). Dans ce dernier cas, l'enveloppe convexe des sommets du graphe est bien entendu ce même triangle.

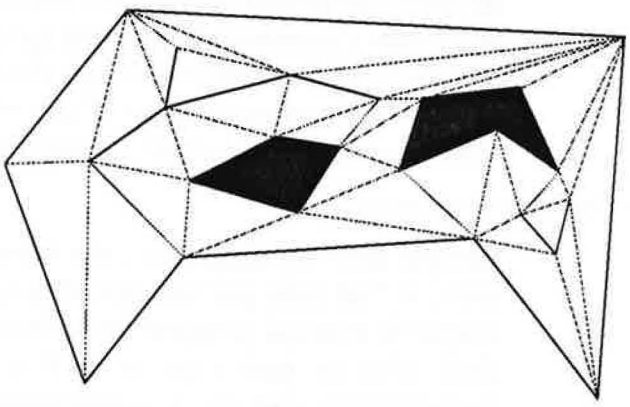


Figure VI.6. Triangulation d'un polygone 'plein'.

Le dernier résultat combinatoire qu'il sera utile de posséder par la suite concerne les situations comme celle de la figure VI.6, que l'on rencontre après avoir appliqué les techniques du chapitre précédent. La rsgp est divisée hiérarchiquement en faces polygonales contenant éventuellement d'autres polygones strictement inclus dans leur intérieur.

Chacune de ces faces considère que ses polygones 'fils' sont en fait des trous (en noir sur la figure), car leur triangulation produit des facettes qui ne les concernent pas. De plus chaque face est susceptible de contenir un ensemble d'arêtes et de sommets isolés strictement inclus dans l'intérieur de la face, constituant un sous-graphe sans faces propres. Le sous-graphe compte z sommets et a arêtes et les trous forment une suite de polygones simples, sans trous[4] et disjoints $\{t_1, t_2, \ldots, t_k\}$, dont chaque élément t_u représente un 'potentiel' de sommets de frontière externe égal à n_u, son propre nombre de sommets.

Comme t_u est un polygone simple sans trou, on peut lui appliquer la formule (VI-6) et l'on constate qu'il apporte $(n_u\text{-}2)$ triangles qui doivent être supprimés de la triangulation de la face qui le contient.

On dira d'un polygone comme celui de f-VI.6 qu'il est *plein*. Pour connaître, par exemple, le nombre de triangles de la triangulation d'un polygone plein, on remarque qu'il compte

(VI-8) $$n_{ext} + z + \sum_{u=1}^{k} n_u$$

sommets en tout et on obtient, en appliquant (VI-6) et (VI-8)

$$n_T = \{2[(n_{ext}+z+\sum_{u=1}^{k} n_u - 1)] - n_{ext}\} - \sum_{u=1}^{k}(n_u - 2),$$

(VI-9) soit $$n_T = n_{ext} + \sum_{u=1}^{k} n_u + 2(z+k-1)$$

Ainsi, sur la figure VI.6., on trouve $n_{ext} = 7$, $z = 8$, $\sum_{u=1}^{k} n_u = 9$ et $k = 2$, ce qui donne un nombre de triangles égal à 7+9+2(8+2-1)=34.

Ce qui est remarquable dans ces propriétés est que tous les résultats obtenus sont vrais *quelle que soit* la triangulation construite. En prenant l'exemple d'un polygone convexe simple de n sommets, dont on a vu qu'une triangulation contient n-2 triangles, le lecteur n'ayant jamais fait ce constat s'apercevra facilement qu'il existe autant de triangulations possibles que de sommets! Ce que disent les relations précédentes c'est que *toutes* les triangulations d'une même rsgp ont le *même* nombre d'arêtes et de triangles. Ceci est une propriété fondamentale.

Dualité Voronoi-Delaunay

Chaque arête du diagramme de Voronoi $DV(S_n)$ correspond à deux sites; si l'on relie par une nouvelle arête, dite de *Delaunay*, chaque couple de sites qui partagent une arête de Voronoi, on obtient un graphe *dual* dont on peut montrer qu'il s'agit d'une triangulation; cette triangulation, qui est à comprendre bien entendu par rapport à l'enveloppe convexe du nuage de points, se nomme la *triangulation de Delaunay de* S_n, que l'on note $TD(S_n)$. La triangulation de Delaunay de f-VI.2 est illustrée à la figure VI.7.

[4] Même si, dans la hiérarchie complète, ils ont une descendance: pour le polygone courant, la descendance des fils ne se comptabilise qu'au niveau suivant de hiérarchie.

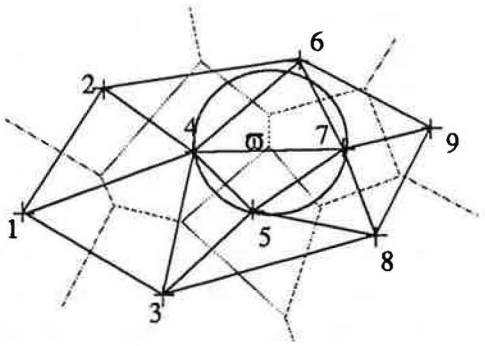


Figure VI.7. Triangulation de Delaunay à partir du diagramme de *f*-VI.2.

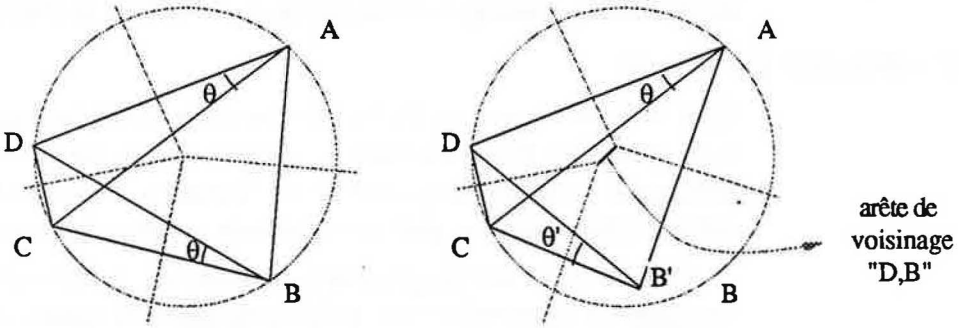


Figure VI.8. Equivalence des critères du cercle et de l'angle minimum.

La triangulation de Delaunay d'un ensemble de points est unique et possède une propriété fondamentale, qui la distingue et permet de la caractériser entièrement: parmi toutes les triangulations sur S_n, la triangulation de Delaunay est telle que le disque circonscrit à chacun de ses triangles ne contient aucun site de S_n en son intérieur. On vérifie sur *f*-VI.7 que le cercle circonscrit au triangle $p_4p_5p_7$ a bien cette propriété.

[LAW 77] montre qu'il y a équivalence entre le critère précédent, dit *critère du cercle* et du suivant, dit *critère de l'angle minimum maximal*: Sur le quadrilatère ABCD de la figure VI.8.(a), dont les quatre sommets sont sur un même cercle, l'angle $\angle CBD=\angle CAD=\theta$ est le plus petit des douze angles que définissent les deux triangulations possibles {DBC, DBA} et {CAD, CAB}. Ces deux triangulations sont en quelque sorte déterminées par la diagonale de ABCD qu'elles contiennent (DB pour la première et CA pour la seconde).

Comme les quatre sites sont co-cycliques, les quatre médiatrices des arêtes du quadrilatère se rencontrent en un seul sommet de Voronoi, de degré quatre, centre du cercle circonscrit à ABCD. Si l'on déplace de manière infinitésimale (*f*-VI.8.(b)) le site B en B', à l'intérieur de ce cercle, les sommets A, B', C et D ne sont plus co-cycliques et l'on constate l'apparition d'une section non dégénérée de la médiatrice de [D,B']: les sites D et B' deviennent voisins et la triangulation fondée sur la diagonale AC n'est plus de type Delaunay car le disque circonscrit à CAD contient B' en son intérieur.

D'autre part, comme le déplacement de B est infinitésimal, θ' est encore le plus petit des six angles de la triangulation {DB'C, DB'A} et

θ celui de la triangulation {CAD, CAB'}: la triangulation de Delaunay maximise le plus petit des angles des triangles qu'elle contient. Un raisonnement analogue peut être mené dans le cas d'un déplacement infinitésimal de B vers l'extérieur du cercle circonscrit à ABCD.

Le critère du cercle et celui de l'angle minimum maximal sont équivalents car ils ont le même cas limite (co-cyclicité) et provoquent toujours les mêmes décisions. On constate notamment que, lorsque les quatre sites A, B, C et D sont sur un même cercle, les deux triangulations possibles sont équivalentes, au sens des deux critères. Il n'y a aucun moyen de choisir l'une plutôt que l'autre, si ce n'est par le biais de méthodes comme celle suggérée en (V-6) dans le chapitre précédent.

Construction par fusion

[L/S 80] donne une méthode 'diviser-pour-résoudre' qui est l'équivalent, pour les triangulations de Delaunay, de la méthode de fusion présentée pour les diagrammes de Voronoi. Cette technique va être utilisée abondamment plus loin et mérite un peu d'attention.

Comme pour les diagrammes de Voronoi, on divise l'ensemble S_n en deux sous-ensembles de tailles sensiblement égales et l'on construit récursivement les sous-triangulations TDG et TDD, puis on fusionne ces deux dernières pour obtenir la triangulation définitive. Ici aussi, la récursion s'arrête quand l'ensemble passé en paramètre a une taille inférieure ou égale à trois, pour des raisons évidentes.

La fusion est similaire à celle des sous-diagrammes de Voronoi: schématiquement, la chaîne de fusion (que l'on imaginera partir du segment d'appui bas) remonte entre les deux enveloppes convexes de TDG et TDD et cherche à éliminer toute arête de la sous-triangulation gauche qui interfère avec la sous-triangulation droite et *vice versa.*

Supposons que le segment d'appui bas ait les extrémités ExtG et ExtD (une dans chaque camp). Ce segment portera plus loin le nom générique de 'base'. ExtG est un site sur la frontière de l'enveloppe convexe de TDG vers lequel converge un certain nombre d'arêtes de TDG. Soient $\{G_1, G_2, \ldots, G_k\}$ les secondes extrémités de ces arêtes incidentes en ExtG, telles que G_1 soit le premier site au-dessus du segment d'appui dans le sens direct. On construit le cercle circonscrit au triangle $ExtG,G_1,G_2$ et l'on teste s'il contient ou non ExtD en son intérieur: si oui, il faut éliminer l'arête $ExtGG_1$ et passer à l'arête $ExtGG_2$ où G_2 est le site suivant G_1 dans le sens direct autour de ExtG; dans le cas contraire on peut garder $ExtGG_1$.

En cas de test négatif on avance dans la liste des arêtes incidentes en ExtG jusqu'à rencontrer un site 'en-dessous' du segment d'appui courant ou bien tel que le cercle circonscrit qui l'accompagne ne contienne plus ExtD. On exécute ensuite la même opération pour TDD, en tournant autour de ExtD dans le sens rétrograde, cette fois.

Quand ces deux opérations sont terminées, les deux sous-triangulations ont peut-être perdu quelques arêtes mais ces arêtes ne pourront plus jamais faire partie de la triangulation de Delaunay car elles ont été éliminées pour cause de violation du critère du cercle.

Après ces opérations d'élimination, on se trouve avec un site gauche G et un site droit D tels que ExtG et ExtD sont extérieurs aux deux cercles circonscrits liés à D et G, et il suffit d'appliquer le critère du cercle au quadrilatère ExtG,ExtD,D,G pour savoir quelle diagonale retenir dans la triangulation. La diagonale retenue est insérée dans la triangulation et sert de base pour recommencer le processus qui se termine lorsque la dernière base coïncide avec le segment d'appui haut. Pour que le test sur ExtG,ExtD,D,G ait un sens, il faut que le quadrilatère obtenu soit convexe, ce qui explique l'arrêt du parcours des listes d'incidence au moment où l'angle géométrique avec la base devient supérieur à $\pi/2$ ('passage sous la base').

La figure VI.9 illustre la méthodologie précédente sur un cas particulier. Sur chacune des deux lignes de la figure, on trouve trois colonnes: dans la première on teste les couples d'arêtes successives dans le sens direct incidentes en ExtG de la sous-triangulation gauche dont le cercle circonscrit contiendrait ExtD en son intérieur; dans la deuxième colonne on teste les couples d'arêtes successives dans le sens rétrograde incidentes en ExtD de la sous-triangulation droite dont le cercle circonscrit contiendrait ExtG en son intérieur; dans la dernière colonne on décide quelle diagonale du quadrilatère ExtG,ExtD,D,G est retenue; les cercles des critères de cette dernière colonne ne sont pas représentés à cause de leurs très grands rayons.

Bien que dans les deux lignes de cet exemple aucune arête des sous-triangulations passées ne soit invalidée, on remarque que le point qui deviendra ExtD à l'étape 3 provoquera quelques remaniements. Intuitivement, le processus de fusion grignote éventuellement les sections d'exposition mutuelle des deux enveloppes convexes, mais ne peut pas rentrer très profondément dans leur intérieur.

On remarque, d'une certaine façon, que le segment de base évolue 'en canard' entre les deux enveloppes convexes: suivant la diagonale retenue, on fait 'remonter' ExtG ou ExtD, jamais les deux en même temps. On retrouve bien là l'évolution de la chaîne de fusion des médiatrices de ces bases successives, dans la fusion de sous-diagrammes de Voronoi.

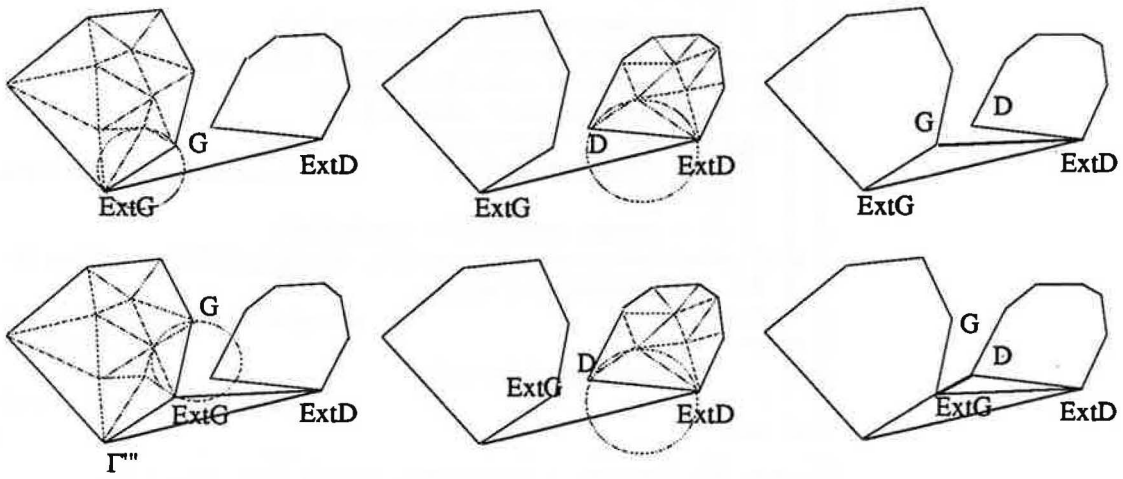


Figure VI.9. Principe de base de la fusion de deux sous-triangulations.

La figure VI.10 illustre la phase de fusion relative à la triangulation des neufs sites de f-VI.3. Comme précédemment, la colonne de gauche contient les tests d'élimination d'arêtes de TDG, celle du milieu les tests d'élimination d'arêtes de TDD et la colonne de droite les critères du cercle appliqués aux quadrilatères finaux; chaque ligne de la figure correspond à une itération de la procédure de fusion.

Par exemple, à la première ligne la base est [3,8], les arêtes incidentes en 3 sont représentées par {4,1} dans cet ordre, celles incidentes en 8 sont représentées par {5, 7, 9} dans cet ordre; le cercle circonscrit à 3,4,1 ne contient pas 8 en son intérieur; le cercle circonscrit à 8,5,7 ne contient pas 3 en son intérieur; le quadrilatère final est donc 3,8,5,4 et le critère du cercle élimine la diagonale 4,8. On insère donc [3,5] dans la triangulation et cette arête devient la nouvelle base. Lors de l'itération suivante, l'arête [5,6] de TDD n'est pas éliminée car le cercle circonscrit au triangle 5,6,7 ne contient pas en son intérieur l'extrémité gauche de la base, 3. En revanche, c'est la diagonale [4,5] qui est choisie, car le cercle circonscrit au triangle 3,4,5 ne contient pas en son intérieur le quatrième sommet, 6. L'arête [5,6] disparaît bien à l'itération suivante car, cette fois-ci, l'extrémité gauche de la base est 4 et l'inclusion de ce site dans le cercle circonscrit au triangle 5,6,7 est détectable.

L'exécution de la phase de fusion se poursuit jusqu'au stade où la base serait égale au segment d'appui haut: il ne reste plus qu'à insérer cette arête dans la triangulation.

On peut donner la synthèse suivante de l'opération de fusion, inspirée de [L/S 80] et du survol [DeF 87], extrêmement bien documenté, sur les méthodes recensées de triangulation.

```
T ← TDG ∪ TDD;                          ¡union de tous les triangles avant fusion¡
[ExtG,ExtD] ← segment d'appui bas [G_B,D_B];
tant que ([ExtG,ExtD] ≠ [G_H,D_H])
    G' ← extrémité de la 1ère arête incidente en ExtG 'suivant' la base;
    si (G' est au-dessus de la base [ExtG,ExtD])
        G" ← suivant direct de G' autour de ExtG;
        tant que (ExtD ⊂ Γ(ExtG,G',G"))
            T ← T-triangle(ExtG,G,G');          ¡suppression d'une facette¡
            G' ← G";
            G" ← suivant direct de G' autour de ExtG;
    D' ← extrémité de la 1ère arête incidente en ExtD 'suivant' la base;
    si (D' est au-dessus de la base [ExtG,ExtD])
        D" ← suivant direct de D' autour de ExtD;
        tant que (ExtG ⊂ Γ(ExtD,D',D"))
            T ← T-triangle(ExtD,D,D');          ¡suppression d'une facette¡
            D' ← D";
            D" ← suivant direct de D' autour de ExtD;
    si (G' est en-dessous de la base ou D' ⊂ Γ(ExtG,ExtD,G'))    ¡G' est OK¡
        T ← T ∪ triangle(ExtG,ExtD,G');
        ExtG ← G';                          ¡on retient la diagonale [G', ExtD]¡
    sinon                                                       ¡D' est OK¡
        T ← T ∪ triangle(ExtG,ExtD,D');
        ExtD ← D';                          ¡on retient la diagonale [ExtG,D']¡
retourner(T);
```

Algorithme VI-1: Fusion de deux sous-triangulation de Delaunay [L/S 80]

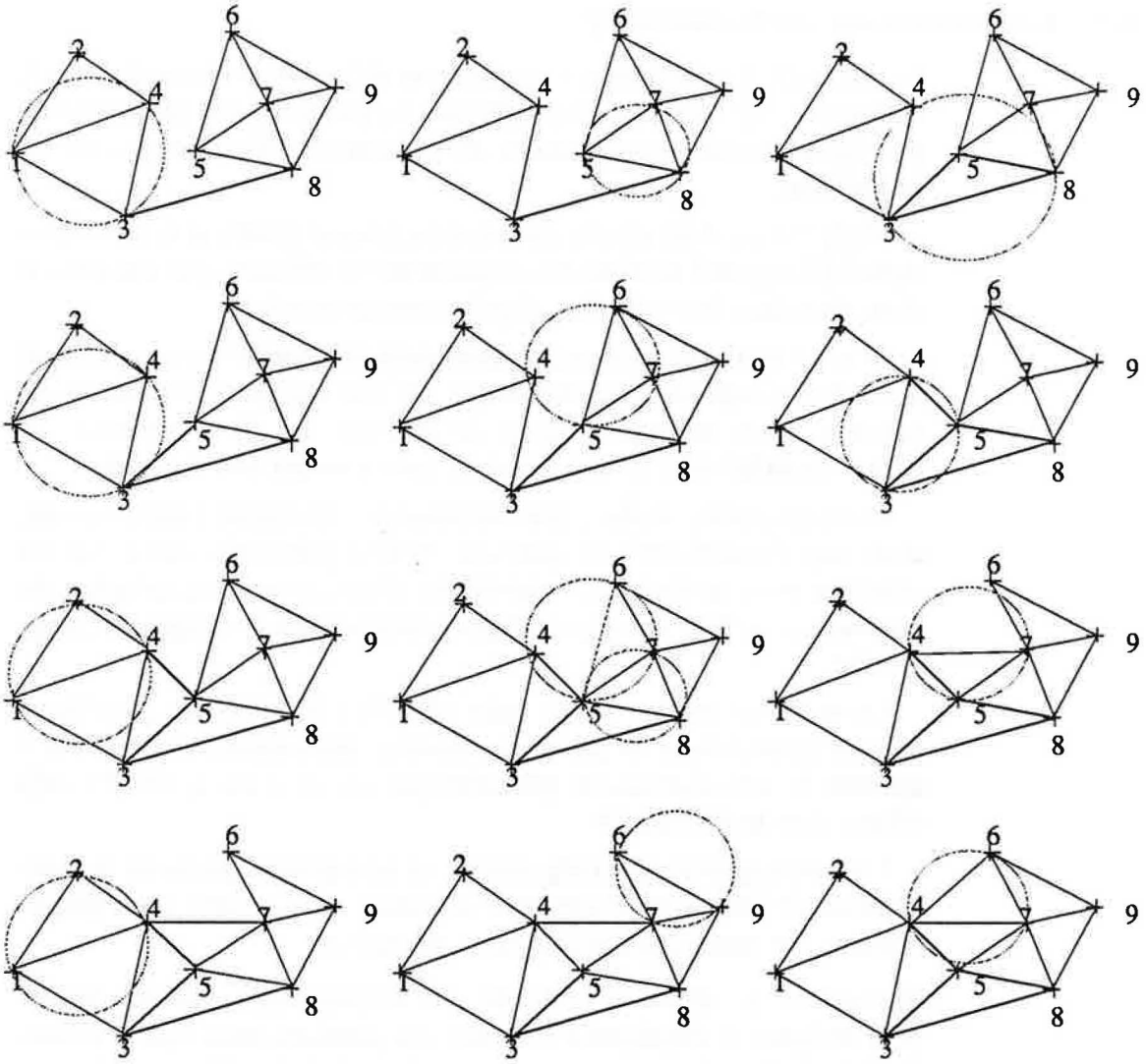


Figure VI.10. Exemple complet de fusion de deux sous-triangulations de Delaunay.

Dans cette synthèse, le test ($U \subset \Gamma(A,B,C)$) signifie que le point U est contenu dans l'intérieur du disque circonscrit au triangle ABC. Il est rappelé que l'on tourne toujours dans le sens direct autour de ExtG et rétrograde autour de ExtD.

Par rapport à la présentation précédente, l'algorithme travaille non plus sur les arêtes insérées ou supprimées, mais sur les triangles. Cela signifie que que la triangulation est organisée de façon à permettre l'insertion et la suppression de triangles en temps constant, dès lors qu'ils sont sélectionnés dans la fusion, par le biais de deux arêtes.

Bien entendu, le nombre d'opérations de la phase de fusion de deux sous-triangulations est proportionnelle au nombre d'arêtes de la triangulation fusionnée et la construction par fusion de la triangulation de Delaunay de S_n a aussi une complexité globale de $O(n\log n)$.

Le traitement du cas particulier où plusieurs sites pourraient se trouver sur un même cercle se traite de manière directe en sélectionnant, parmi les triangulations possibles, une triangulation dont le choix puisse être reproductible.

VI.A.3 Constructions incrémentales

La méthode de construction par division et fusion des tessellations de Voronoi et de Delaunay est optimale et extrêmement puissante: le paradigme 'diviser-pour-résoudre' est certainement l'un des plus efficaces qui soit.

Malgré tout, il est connu que cette technique implique le traitement séparé d'un grand nombre de cas particuliers délicats (par exemple la séparation dans le cas de sites alignés verticalement).

D'autre part, on sait bien que toute méthode non récursive possédant la même complexité asymptotique qu'une méthode récursive est souvent plus efficace, dans la mesure où la constante de proportionnalité qui l'accompagne est aussi souvent moins élevée.

Dans cette sous-section, une méthode de construction incrémentale, mais non dynamique[5], et optimale va être présentée, qui a comme première vertu de donner un algorithme simple pour la construction du diagramme de Voronoi d'un polygone monotone à n sommet en temps linéaire.

Le meilleur résultat sur le sujet [A/G/S/S 89] était un algorithme linéaire dans le cas d'un polygone convexe, mais assez complexe car il utilisait la transformation géométrique $(x, y) \rightarrow (x, y, x^2+y^2)$ déjà définie dans le chapitre IV.

Une autre qualité de cet algorithme est de s'étendre au cas de la triangulation de Delaunay d'une représentation par segment d'un graphe planaire, qui sera l'objet de la prochaine section.

Remarque: Le terme incrémental sera employé dans tout le chapitre pour désigner la technique d'insertion des sommets dans l'ordre lexicographique. Ce terme a souvent un sens plus général qu'il vaut mieux ne pas retenir ici.

Construction incrémentale de l'enveloppe convexe d'un nuage de points

Pour pouvoir analyser plus facilement l'algorithme central de cette section, on va s'intéresser à un algorithme fort ingénieux [EDE 87, pp. 143-145] de construction incrémentale de l'enveloppe convexe de n points distincts du plan. On suppose ici encore, pour éviter les cas dégénérés dans la présentation, que trois sommets quelconques de S_n ne sont pas *alignés*. Comme toujours, ces cas peuvent être traités moyennant quelques modifications, modestes dans le cas présent.

On commence par ranger les n sommets selon l'ordre lexicographique de leurs coordonnées dans le plan, on les renumérote $\{p_1, p_2, \ldots, p_n\}$ et l'on considère l'enveloppe convexe des trois premiers points. Comme ceux-ci ne peuvent être alignés, le triangle qu'ils définissent est non dégénéré. On considère ensuite les n-3 points qui restent, dans l'ordre lexicographique.

---

[5] i.e. les points sont connus à l'avance et n'arrivent pas un à un, exigeant une mise à jour plus complexe et 'dynamique' du diagramme. On sait qu'une telle technique s'accompagne d'une borne inférieure de $\Omega(n^2)$ [G/K/L/N 83].

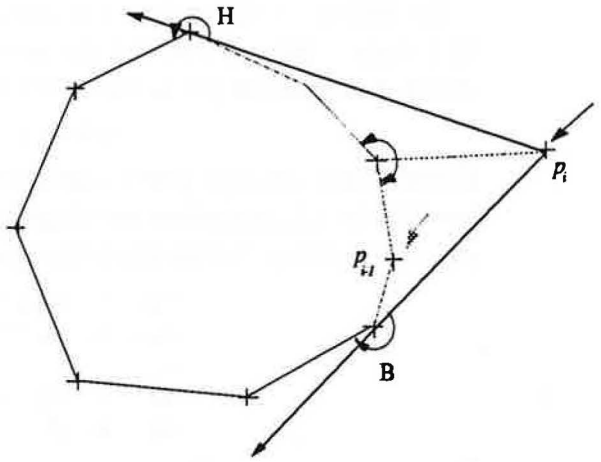


Figure VI.11. Recherche des deux segments d'appui depuis p_i.

Puisque le nouveau point p_i inséré se trouve à droite de tous les sommets déjà considérés, il constitue nécessairement un point extrémal de S_i et se trouve donc sur la frontière de son enveloppe convexe, Fr(EC(S_i)): l'insertion de p_i entraîne l'éventuelle disparition d'ancien sommets de la frontière de la dernière enveloppe convexe construite (initialement, cette enveloppe est le triangle $p_1p_2p_3$).

On part donc du point p_{i-1} rentré précédemment (initialement, p_{i-1} est égal à p_3) et l'on parcourt la frontière de l'enveloppe convexe d'abord dans le sens direct jusqu'à trouver le sommet H tel que [H,p_i] est un segment d'appui ('haut'); puis on repart de p_{i-1} et l'on parcourt la frontière de l'enveloppe convexe dans le sens rétrograde jusqu'à trouver le sommet B tel que [p_i, B] est le deuxième segment d'appui ('bas').

En se reportant à la figure VI.11, on constate que pour trouver H (resp. B) qu'il suffit de rechercher, dans le parcours direct (resp. rétrograde), un point Q tel que, si R est son successeur dans l'ordre direct (resp. rétrograde), l'angle géométrique $\angle p_iQR$ soit supérieur à π.

L'analyse de cet algorithme très simple, nécessite, quant à elle, de raisonner sur le nombre total d'opérations effectuées, sur l'ensemble des boucles de parcours: en effet, il est très possible que l'insertion d'un point quelconque provoque la suppression de $O(n)$ sommets de l'enveloppe convexe précédente; puisque cette situation peut théoriquement se présenter pour chacun des n-3 points traités, cela signifie que la complexité de l'algorithme est potentiellement quadratique!

Heureusement, on peut réfuter cet argument en s'appuyant sur un raisonnement qui sera précieux pour l'algorithme de construction d'un diagramme de Voronoi: à l'étape i (correspondant à l'insertion du sommet p_i), désignons par s_i le nombre d'arêtes supprimées de l'enveloppe convexe précédente. Comme toute arête détruite ne peut l'être qu'une fois, on est sûr que les sommets auxquels correspondent ces arêtes seront définitivement éliminés de la liste des sommets de l'enveloppe convexe courante, après l'insertion de p_i. Si l'on retire s_i arêtes de l'enveloppe, on n'en rajoute pas moins deux nouvelles arêtes, nommément les deux segments d'appui [p_i, H] et [p_i, B]: ceci est aussi vrai si aucun segment n'est supprimé de l'enveloppe convexe précédente.

En définitive, le nombre κ_i d'arêtes de l'enveloppe convexe à la fin de l'étape i peut se déduire du nombre κ_{i-1} d'arêtes de l'enveloppe convexe précédente par la formule très simple suivante

(VI-10) $$\kappa_i = \kappa_{i-1} - s_i + 2$$

Comme cette relation peut s'appliquer pour tout $i \in [1, n]$, et comme κ_3 correspond à la première enveloppe qui contient nécessairement trois arêtes, on peut écrire les relations suivantes:

$$\begin{aligned} \kappa_n &= \kappa_{n-1} - s_n + 2 \\ \kappa_{n-1} &= \kappa_{n-2} - s_{n-1} + 2 \\ &\dots \\ \kappa_4 &= \kappa_3 - s_4 + 2 \\ \kappa_3 &= 3 \end{aligned}$$

Puisque $\kappa_1+\kappa_2+\kappa_3=3$, et puisque $s_1+s_2+s_3 = 0$ par construction, on obtient, en additionnant membre à membre ces égalités:

(VI-11) $$\kappa_n = 3 + 2(n-3) - \sum_{i=4}^{n} s_i = 2n-3 - \sum_{i=1}^{n} s_i$$

Grâce aux remarques faites plus haut, on sait que la quantité la plus à droite représente le nombre total d'opérations de l'algorithme, puisque celui-ci parcours autant de sommets qu'il détruit d'arêtes et qu'il insère toujours un nombre constant d'arêtes.

En dernier lieu, κ_n n'est rien d'autre que le nombre total d'arêtes de la frontière de l'enveloppe convexe de S_n: il est évident que ce nombre ne peut jamais excéder n ni être inférieur à 3, par construction. On peut donc affirmer que le nombre $Conv_n$ d'opérations entreprises par l'algorithme est borné par:

(VI-12) $$Conv_n = \sum_{i=1}^{n} s_i = 2n - 3 - \kappa_n \Leftrightarrow n+3 \leq Conv_n \leq 2n-3$$

On en déduit que la complexité dans le pire des cas de cette partie de l'algorithme est $O(n)$. Bien entendu, la complexité globale de l'algorithme est $O(n\log n)$ puisqu'il inclut le tri des sommets au départ.

Construction incrémentale de $DV(S_n)$

On désire maintenant généraliser généraliser la technique précédente à la construction du diagramme de Voronoi. Bien que la situation soit bien plus complexe, les résultats seront aussi satisfaisants[6].

On part, comme dans l'algorithme précédent, d'un ensemble S_n dont on se rappelle que les éléments, appelés sites, sont en position générale. On les range par ordre croissant lexicographique et l'on considère le diagramme de Voronoi des trois premiers points. Celui-ci se construit sans difficulté, puisque les trois sites ne peuvent être alignés. Il a été représenté en *f*-VI.1.(b).

[6] Le lecteur intéressé pourra comparer l'algorithme analysé ici à celui de construction par balayage du diagramme de Voronoi d'un nuage de points du plan [FOR 86]. Il est à noter que le résultat attendu dans ce chapitre est une unification des algorithmes de construction des diagrammes de proximité autour du paradigme précédent, alors que l'algorithme cité fait usage d'une transformation du plan afin de permettre le balayage et est d'une essence très différente.

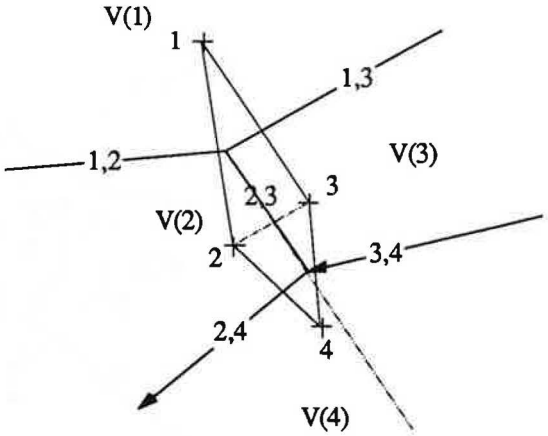


Figure VI.12. Insertion de p_4 dans le diagramme f-VI.1.(b).

Essayons maintenant de construire la suite du diagramme de Voronoi pour les n-3 sites restants de S_n. La figure VI.12 montre 'l'arrivée' du nouveau point p_4. Il y a deux phénomènes à observer: l'insertion du quatrième point a une influence sur l'enveloppe convexe antérieure ($p_1p_2p_3$) et sur le diagramme de Voronoi antérieur ($DV(S_3)$).

En quoi l'évolution de l'enveloppe convexe est-elle importante? Puisque l'on sait que p_4 se trouve *nécessairement* sur $Fr(EC(S_4))$, on peut reconstruire la nouvelle enveloppe convexe à partir des deux segments d'appui $[p_4, p_3]$ et $[p_4, p_2]$. Mais, puisque le nouveau point est sur la frontière de la nouvelle enveloppe convexe, le polygone de Voronoi V(4) possède obligatoirement deux rayons infinis: le premier correspond à la médiatrice de $[p_4, p_3]$ et le second à celle de $[p_4, p_2]$.

D'une manière générale, ces deux rayons sont *toujours* déterminés par les deux segments d'appui trouvés dans la mise à jour de l'enveloppe convexe: ce résultat est une conséquence d'une propriété plus générale donnée précédemment, concernant les rayons infinis des polygones de Voronoi. Consulter [SHA 78, pp. 178-191], [LEE 78] et [PER 88, pp. 4-12] pour des démonstrations détaillées de ce point.

Les éléments de réflexion qui suivent sont liés à une suite de points choisie pour 'provoquer' quelque peu le destin. Leur but est de donner les moyens de comparer le comportement de la ligne de fusion de la méthode 'diviser-pour-résoudre' et de la chaîne de mise à jour du présent algorithme: toute ressemblance est la bienvenue.

Sur la figure VI.12, on peut interpréter les événements de la manière suivante: on part de $DV(S_3)$; le rayon M_{34} se trouve dans le polygone de Voronoi V(3) et intersecte le rayon infini M_{23} de $V(2) \cap V(3)$: en ce point, on est à égale distance des points p_2, p_3 et p_4. Ce point coïncide aussi avec la sortie de V(3) et l'entrée dans V(2): il est donc nécessaire de construire la médiatrice M_{24} afin de rester encore une fois à égale distance de p_4 et d'un point de S_3. Cette dernière n'intersecte plus aucune autre médiatrice de V(2).

En ce qui concerne V(2), la construction précédente a ajouté le rayon infini M_{24} et réduit le rayon infini M_{23} en segment fini. V(3) a subi une transformation comparable.

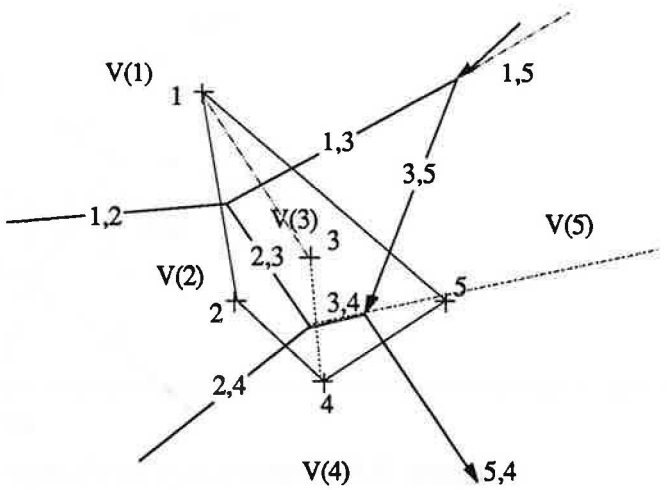


Figure VI.13. Insertion du cinquième point.

L'arrivée de p_5 (figure VI.13) a plus de conséquences que précédemment. En particulier, le polygone de Voronoi V(3) devient fermé-borné: en effet, p_5 fait disparaître p_3 de la frontière de l'enveloppe convexe de S_5 et la médiatrice M_{35} vient fermer V(3) par un segment compris entre les intersections de cette première avec les deux rayons infinis antérieurs de ce polygone, M_{13} et M_{34}.

Comme on le voit sur ce dernier cas, le cheminement de la chaîne de mise à jour du diagramme de Voronoi est fortement influencé par la géométrie de la nouvelle enveloppe convexe, mais aussi par celle des polygones de Voronoi rencontrés. Ceci va devenir encore plus probant sur l'exemple suivant, où l'on insère le sixième point de la liste (Figure VI.14).

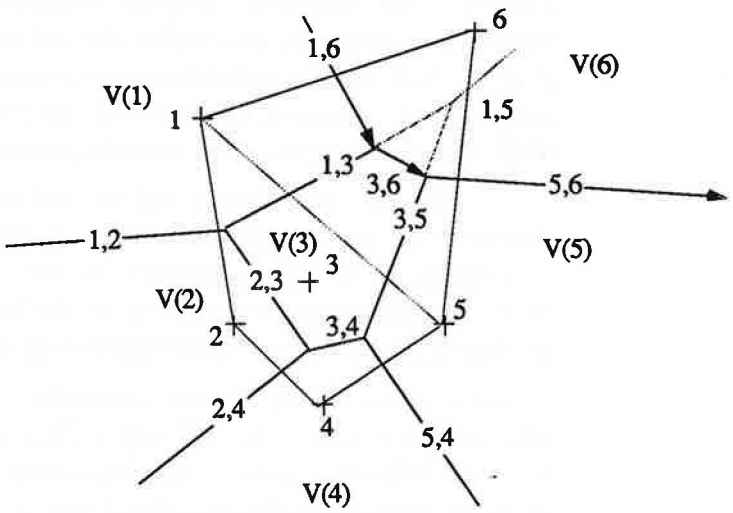


Figure VI.14. Insertion du sixième point.

Cette fois-ci, aucun point de la frontière de l'ancienne enveloppe convexe n'est remis en cause, mais, en revanche, c'est un rayon infini de l'ancien diagramme de Voronoi qui est détruit! Le segment d'appui 'haut' est $[p_1, p_6]$. La chaîne de mise à jour commence donc par le rayon infini M_{16}.

Chemin faisant, elle intersecte M_{13}, se transforme en M_{36}, intersecte M_{35}, et se transforme finalement en M_{56} qui est le rayon correspondant au segment d'appui 'bas' final.

On constate que la rectification des segments de médiatrice M_{13} et M_{35} de V(3) entraîne la disparition définitive du rayon infini M_{15}. Comme dans le cas de l'algorithme précédent, bien que toute arête supprimée dans le diagramme de Voronoi de l'étape i entraîne sa disparition de tous les diagrammes, y compris du diagramme final, il n'en reste pas moins que l'insertion d'un seul point peut éventuellement entraîner $O(n)$ opérations à elle seule.

Ce qui paraît inquiétant, dans l'algorithme qui se dessine est qu'il semble possible de 'retoucher' plusieurs fois les mêmes arêtes: par exemple, l'arête de médiatrice M_{13} a été retouchée une première fois, lorsque p_5 a été inséré, puis une seconde fois lorsque p_6 a été inséré à son tour. Pour se rendre compte de la portée des remaniements que peut engendrer l'insertion d'un seul point, insérons un dernier sommet:

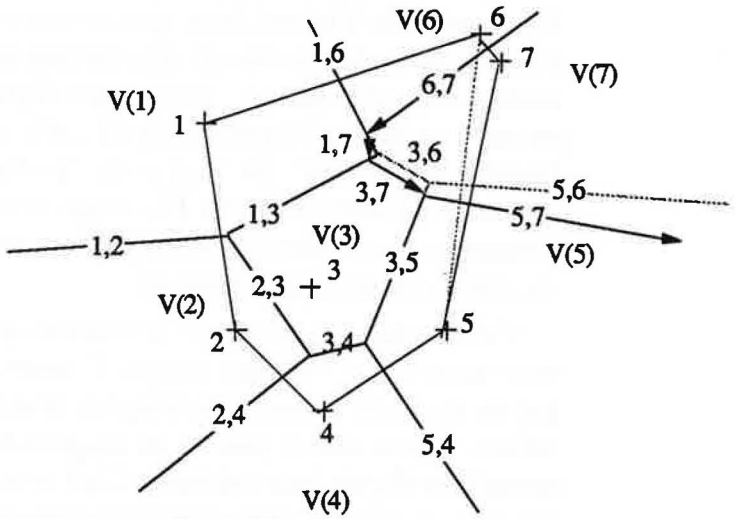


Figure VI.15. Insertion ultime.

Cette fois-ci, on constate sur f-VI.15 que M_{13} est retouchée pour la troisième fois et que deux arêtes disparaissent, dont un rayon infini. Pour prouver que cet algorithme ne passe pas trop de temps à créer puis détruire des médiatrices inutilement et surtout à les remanier indéfiniment, il va falloir montrer que le nombre maximum d'arêtes de médiatrices créées reste 'raisonnable'.

La chaîne de mise à jour que l'on construit est un cas extrêmement simplifié de la *chaîne de fusion* de la méthode 'diviser-pour-résoudre'. En fait, à l'étape i, on est en présence d'un diagramme de Voronoi de i-1 points à gauche et d'un diagramme de Voronoi de 1 point à droite, séparés par une droite verticale: ceci ne serait pas suffisant pour garantir une complexité optimale dans la méthode 'diviser-pour-résoudre', car il est indispensable que les deux diagrammes de Voronoi aient des tailles sensiblement égales; or la taille du diagramme de gauche ne cesse de croître, alors que celle du diagramme de droite reste constamment égale à un. Malgré tout, ne perdons pas courage!

Une propriété de la chaîne de fusion qui se transmet merveilleusement bien à la chaîne de mise à jour est sa monotonie. Cette propriété est beaucoup plus forte ici, du fait que le diagramme de droite est *toujours* réduit à un point.

Propriété 1: La chaîne de mise à jour est convexe.

> Démonstration: Comme toute chaîne de subdivision, dans la méthode 'diviser-pour-résoudre', est monotone, cela signifie qu'une telle chaîne est convexe et concave de manière alternée. En fait, elle est convexe tant qu'elle reste à l'intérieur d'un polygone de Voronoi du diagramme de Voronoi de droite et concave tant qu'elle reste à l'intérieur d'un polygone de Voronoi du diagramme de gauche. Ainsi le changement de 'concavité' se fait lorsque la chaîne franchit deux fois de suite la frontière d'un polygone de Voronoi d'un des deux diagrammes sans franchir celle de l'autre.
> Il est clair qu'une chaîne de mise à jour ne peut jamais franchir deux fois de suite la frontière du seul polygone de Voronoi du diagramme de droite, puisque celui-ci, avant sa construction, n'a pas d'arêtes! Toute chaîne de mise à jour est donc nécessairement convexe.

Cette propriété, qui correspond, évidemment, au fait que l'on construit le polygone de Voronoi de p_i, qui est avant tout un convexe, va être un atout de taille. Intuitivement, elle garantit qu'une chaîne de mise à jour tourne sagement dans le sens direct depuis la position de pente du premier segment d'appui jusqu'à celle du deuxième, sans jamais 'revenir en arrière'. Si l'on a de la chance, les modifications et suppressions engendrées par l'insertion d'un nouveau point ne vont pas remettre en cause gravement (en profondeur et trop souvent) la stabilité acquise du diagramme précédent.

S'il est vrai que certaines médiatrices sont soumises à beaucoup de mouvements ces derniers temps, d'autres, et elles sont légion, n'ont pas eu trop à se plaindre de mauvais traitements: beaucoup parmi les médiatrices situées à gauche du diagramme de *f*-VI.15 n'ont pas été remaniées depuis leur création. C'est exactement sur ce phénomène, que l'on va prouver, que l'algorithme tire son efficacité: plus on insère de points par la droite, et moins il devient possible de remanier la partie gauche du diagramme; après le point p_7 le lecteur peut constater qu'il devient assez délicat de provoquer une rectification de la médiatrice M_{13}, même si cela reste toujours théoriquement possible!

Afin de pouvoir en donner une analyse précise, écrivons formellement l'algorithme suggéré par l'exemple précédent. Ce que l'on a vu plus haut permet de supposer l'existence d'un module, MajEC(*p*, *EC*), qui met à jour l'enveloppe convexe courante *EC* après l'insertion du point *p* et qui retourne les deux sommets H et B de l'ancienne enveloppe convexe qui permettent de construire les segments d'appui $[p_i, H]$ et $[p_i, B]$, à l'étape *i*.

Encore une fois, l'opération d'union dans la construction du polygone de Voronoi de p_i nécessite aussi la prise en compte des liens de jumelage avec le polygone de Voronoi de DV adjacent et le 'recollement' des arêtes successives créées dans ces deux objets. Tous ces détails sont laissés de côté, car ils ne jouent pas un rôle déterminant dans la complexité asymptotique de l'algorithme.

```
trier S_n de manière à obtenir l'ensemble {p_1, p_2, ..., p_n};
EC ← triangle p_1p_2p_3;
initialiser DV au diagramme de Voronoi de {p_1, p_2, p_3}
pour (i=4; i≤n; i++)
    V(p_i) ← NIL;
    (EC, H, B) ← MajEC(p_i, EC);
    chaînon_maj ← Médiatrice(DV,p_i,NIL,NIL);
    V(p_i) ← DV(p_i) ∪ chaînon_maj;
    tant que ((Ω ← (chaînon_maj ∩ (DV ) ≠ {}) ¡partir du rayon(?,NIL) de V(H)¡
        chaînon_maj ← (NIL,Ω);
        soit m la médiatrice de DV intersectée par chaînon_maj;
        m ← (?,Ω);
        soit [G,G'] le segment dont m est la médiatrice;
        G ← G';
        chaînon_maj ← Médiatrice(G,D,Ω,NIL);
    DV ← DV ∪ V(p_i));
retourner (DV);
```

Algorithme VI-iii: Construction incrémentale du diagramme de Voronoi de S_n.

Naturellement, l'algorithme VI.iii est une version extrêmement simplifiée de l'algorithme VI.i, ce qui est bien ce que l'on cherche à obtenir! La partie de l'algorithme concernant le tri a une complexité de $O(n\log n)$ et la construction implicite de l'enveloppe convexe a un complexité linéaire, comme on l'a vu précédemment.

Reste la complexité de la deuxième partie de l'algorithme, dont on souhaite montrer qu'elle est aussi linéaire dans le nombre de sites.

Il est primordial de noter que dans la boucle '**tant que**', toute arête du polygone de Voronoi dans lequel on est entré et qui n'intersecte pas la chaîne de mise à jour mais se trouve *à droite* de celle-ci (i.e. à l'intérieur du futur polygone de Voronoi de p_i) est définitivement détruite et ne pourra plus jamais être rencontrée. On commence le parcours par le rayon initial du polygone de Voronoi du point d'appui haut, dans le parcours rétrograde de cette chaîne; tous les polygones de Voronoi 'de gauche' sont parcourus dans le sens rétrograde, ce qui signifie, puisque la chaîne de mise à jour reste convexe, que l'on n'effectue pas de retour en arrière et donc que la seule possibilité pour sortir d'un tel polygone est que la chaîne de mise à jour y laisse comme trace un unique segment, appelé arête de traversée. Toute autre possibilité contredirait la convexité des polygones de gauche ou de $V(p_i)$.

En définitive, on s'aperçoit, à l'instar de l'algorithme correspondant à l'enveloppe convexe, que la complexité de cet algorithme est donnée par le nombre total d'arêtes qu'il crée: même si une arête peut être rectifiée plusieurs fois dans l'exécution de l'algorithme VI.iii (comme l'a montré *f*-VI.15), chaque modification est consécutive à la création d'une arête de traversée. Naturellement, on ne peut pas supprimer plus d'arêtes que l'algorithme en a créées et toute suppression est aussi consécutive à la création d'une arête de traversée.

Ainsi, si l'on arrive à cerner précisément le nombre d'arêtes créées en tout par l'algorithme, on connaîtra une borne supérieure sur le nombre total d'opérations qu'il entraîne.

Encore un peu de dénombrement...

Les résultats de dénombrement obtenus plus haut vont se révéler d'une grande utilité dans le calcul du nombre d'arêtes créées par l'algorithme VI.iii.

Il est facile de voir que le nombre de faces d'un diagramme de Voronoi est exactement égal au nombre de sites. D'autre part, comme tout diagramme de Voronoi est le dual de la triangulation de Delaunay sur le même ensemble de sites, cela signifie qu'à une arête de l'un correspond une et une seule arête de l'autre. Les résultats concernant le nombre d'arêtes des triangulations quelconques vont pouvoir s'appliquer ici, directement.

On se rappelle que l'on a pu établir que le nombre d'arêtes n_A d'un diagramme de Voronoi sur n sites était donné par la relation suivante, où la face externe est bordée par l'enveloppe convexe des n sites:

(VI-13) $$n_A = 3(n\text{-}1)\text{-}n_{ext} = 3(n\text{-}1)\text{-}n_{EC}$$

Ainsi, l'enveloppe convexe joue effectivement un rôle primordial dans le dénombrement que l'on tente de faire, et ceci est heureux car l'algorithme VI.iii fait explicitement intervenir la construction de celle-ci[7].

On se place à l'étape i de la construction, c'est-à-dire lors de l'insertion de p_i. D'après (VI-13), si, avant l'arrivée de ce site, la frontière d'enveloppe convexe compte $EC_{i\text{-}1}$ points, le diagramme de Voronoi construit sur $S_{i\text{-}1}$ contient

$$\kappa_{i\text{-}1} = 3((i\text{-}1)\text{-}1) - EC_{i\text{-}1} = 3(i\text{-}2) - EC_{i\text{-}1}$$

arêtes. A l'arrivée de p_i, on peut supposer que s_i points vont être supprimés de la frontière précédente d'enveloppe convexe, tandis qu'un point (le nouveau site p_i) lui est ajouté: le nombre de sommets de la frontière d'enveloppe convexe de S_i, EC_i, est donc égal à $EC_{i\text{-}1}\text{-}s_i\text{+}1$. Ainsi, le nombre d'arêtes du diagramme de Voronoi sur S_i est

$$\kappa_i = 3(i\text{-}1) - EC_i = 3(i\text{-}1) - (EC_{i\text{-}1} - s_i + 1)$$

Si l'on note δ_i la différence entre le nombre d'arêtes du diagramme à l'étape $(i\text{-}1)$ et celui du diagramme à l'étape i, on peut écrire

(VI-14) $$\delta_i = \kappa_i - \kappa_{i\text{-}1} = s_i + 2$$

Si l'on évalue ces différences de $i=n$ à $i=3$, pour lequel il suffit de comptabiliser les trois premières arêtes construites, on trouve:

$$\begin{aligned} \delta_n &= s_n + 2 \\ \delta_{n\text{-}1} &= s_{n\text{-}1} + 2 \\ &\cdots \\ \delta_4 &= s_4 + 2 \\ \delta_3 &= 3 \end{aligned}$$

Comme $\delta_1+\delta_2+\delta_3=3$, on obtient, en additionnant membre à membre ces égalités

(VI-15) $$DV_n = \sum_{i=1}^{n}\delta_i = 3+ 2(n\text{-}3) + \sum_{i=1}^{n}s_i = 2n\text{-}3 + Conv_n$$

[7] Le lecteur intéressé trouvera dans [BRO 79] la présentation d'un tout autre type de relation entre les enveloppes convexes et les diagrammes de Voronoi.

La quantité DV_n donne le nombre total d'arêtes créées dans une exécution quelconque de l'algorithme VI.iii, nombre qui dépend directement de $Conv_n$, le nombre d'opérations de l'algorithme de construction de l'enveloppe convexe de l'ensemble S_n. On sait, par (VI-12), que $Conv_n$ est compris entre n+3 et $2n$-3, ce qui veut dire que le nombre total d'arêtes créées dans l'algorithme VI.iii est compris entre $3n$ et $4n$-6: le nombre d'opérations de la partie itérative de l'algorithme VI.iii reste proportionnel au nombre de sites.

Polygones monotones et convexes

Un polygone Q du plan $\boldsymbol{P}$ est *monotone* s'il existe une direction Δ par rapport à laquelle tous les sommets de la frontière de Q sont naturellement ordonnés par abscisses croissantes. Un polygone convexe est un cas particulier de polygone monotone. Comme les sommets d'un polygone monotone sont triés dans la direction de monotonie, la phase de tri initiale de l'algorithme VI.iii est inutile et l'on en déduit:

(VI-16) *Il est possible de construire incrémentalement le diagramme de Voronoi d'un ensemble de n sites situés sur la frontière d'un polygone monotone (ou convexe) en O(n).*

Construction incrémentale de $TD(S_n)$

Il est important pour la suite des événements, de donner un aperçu de la version adaptée à la construction d'une triangulation de Delaunay d'un nuage de points de l'algorithme incrémental précédent. Sachant que la triangulation de Delaunay est le dual du diagramme de Voronoi pour un même ensemble de sites, il est facile de passer de l'un à l'autre en temps linéaire. Mais cette solution n'est pas toujours désirable, car le diagramme de Voronoi peut n'être d'aucune utilité en tant que tel. Dans ces cas, le recours d'un algorithme direct de construction est le bienvenu.

Le principe de l'algorithme incrémental est identique à celui de la méthode par fusion, à ceci près que la sous-triangulation droite est réduite à un site, p_i, à l'étape i, ce qui donne l'algorithme recherché:

```
trier Sn pour obtenir l'ensemble {p1, p2, ..., pn};
T ← triangle(p1,p2,p3);
EC ← triangle p1p2p3;
pour (i=4; i ≤ n; i++)
    (EC, H, B) ← MajEC(pi, EC);
    [ExtG,ExtD] ← segment d'appui bas [B,pi];
    tant que ([ExtG,ExtD] ≠ [H,pi])
        G' ← extrémité de la 1ère arête incidente en ExtG 'suivant' la base;
        si (G' est au-dessus de la base)
            G" ← suivant direct de G' autour de ExtG;
            tant que (ExtD ⊂ Γ(ExtG,G',G"))
                T ← T-triangle(ExtG,G,G');        ¡suppression d'une facette¡
                G' ← G";
                G" ← suivant direct de G' autour de ExtG;
    T ← T ∪ triangle(ExtG,ExtD,G');
    ExtG ← G';
retourner(T);
```

Algorithme VI-iv: Construction incrémentale d'une triangulation de Delaunay.

La complexité de cet algorithme est la même que celle de l'algorithme incrémental de construction d'un diagramme de Voronoi, pour des raisons évidentes de dénombrement.

VI.B. TRIANGULATION DE DELAUNAY D'UNE RSGP

La dernière section de cette thèse concerne la triangulation de Delaunay de la représentation par segments d'un graphe planaire. Ce type d'opération porte souvent le nom générique de *triangulation de Delaunay contrainte*, pour signifier que les arêtes du graphe jouent le rôle de *contraintes* obligeant le choix de triangles qui ne les intersectent pas. Comme il n'y a pas de risque d'ambiguïté, le terme de *triangulation contrainte* sera employé pour désigner une triangulation de Delaunay d'une représentation par segments d'un graphe planaire.

On peut distinguer deux cas importants de triangulation contrainte, qui feront l'objet de deux parties distinctes: le cas où le graphe est hiérarchisé et celui où il ne l'est pas. Dans le second cas, il peut être assimilé à un ensemble de segments et de sommets isolés, sans faces: toutes les faces d'une telle rsgp sont triangulées 'au même niveau'.

Dans le premier cas, chaque face est un polygone plein contenant lui-même un graphe non hiérarchisé et un certain nombre de faces polygonales strictement incluses et plus prioritaires. La triangulation de chaque face est entreprise indépendamment des autres. La principale différence est que les polygones inclus dans une face (les faces plus prioritaires) sont considérés comme des trous du polygone courant. Ceci permet d'entériner des propriétés de la hiérarchie construite sur le graphe, comme, par exemple, l'attribution de couleurs, de textures ou tout autre attribut pertinent.

Quelques définitions

G_n sera maintenant la rsgp de n sommets que l'on cherche à trianguler. Soient A et B deux sommets de G_n et [U,V] une arête quelconque de ce graphe. On dit que B est *visible* de A (et réciproquement) si le segment de droite [A,B] ne coupe aucun segment [U,V].

Soit *TDC* une triangulation du nuage de sommets de la rsgp (incluant bien sûr les extrémités des segments). Si un triangle de *TDC* ne contient aucune arête du graphe, on le dira *libre* et sinon on dira qu'il est *contraint* (par le graphe). *TDC* est la triangulation de Delaunay contrainte de G_n si et seulement si:

- Toute arête de G_n est une arête de *TDC*; ces arêtes portent le nom d'*arêtes de graphe* ou de *contrainte*; les autres arêtes de *TDC* portent le nom d'arêtes de Delaunay, comme d'habitude.
- Si UVW est un triangle libre quelconque de *TDC*, le cercle circonscrit à UVW ne contient aucun autre sommet de G_n en son intérieur. On dira que ce triangle est un triangle de Delaunay *libre*.
- Si UVW est un triangle contraint quelconque de *TDC*, il possède au moins une arête de graphe [U,V]; alors le cercle circonscrit à UVW ne contient, éventuellement, en son intérieur qu'un point non visible depuis W.

Un exemple de rsgp, de triangulation de Delaunay des *sommets* de cette rsgp et de triangulation de Delaunay (par défaut) du graphe est illustré à la figure VI.16.

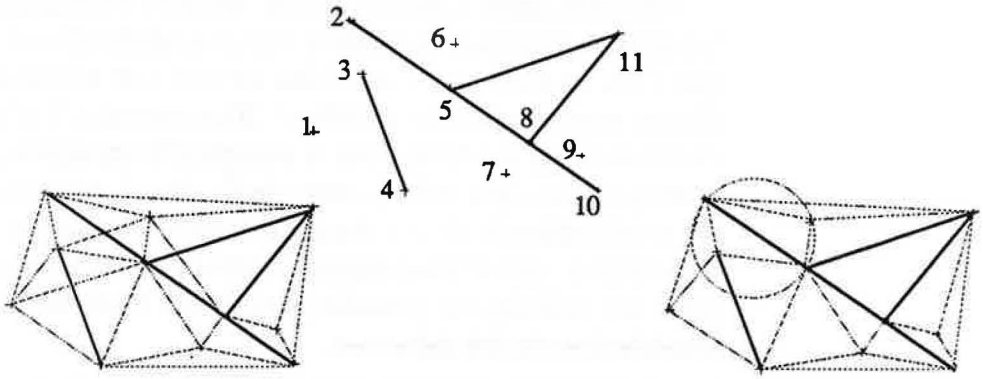


Figure VI.16. Rsgp (en haut), triangulation de Delaunay des sommets de la rsgp (en bas à gauche) et triangulation contrainte (en bas à droite).

On remarque, sur cette figure, que certaines arêtes de Delaunay de la triangulation non contrainte intersectent des arêtes du graphe et ne sont pas valides dans la triangulation contrainte; le cercle circonscrit au triangle 2,5,6 dans cette dernière, contient effectivement le sommet 3: l'arête [2,5] appartient au graphe de contrainte et 3 n'est pas visible depuis 6. Cette définition se visualise encore mieux dans le cas d'un polygone simple où le cercle (de Delaunay) circonscrit à une concavité locale de la frontière peut très bien abriter en son intérieur un autre point de la frontière pourtant caché par non-convexité.

Il est aussi patent sur la figure que certaines arêtes de contraintes sont de 'bonnes' arêtes: par exemple, [8,10] est une arête de Delaunay dans les deux triangulations. Enfin, on constate aisément que la définition donnée ici englobe celle de triangulation sans contrainte, lorsque la rsgp n'a aucune arête.

Remarques: Dans la première sous-section, toutes les triangulations considérées sont à comprendre relativement à l'enveloppe convexe du nuage de sommets de la rsgp. Les triangulations de Delaunay relativement à la face externe sont abordées dans la seconde sous-section. De plus, dans toute la suite, n désigne la taille de la rsgp, i.e. le nombre de ses sommets (isolés ou non).

VI.B.1 Triangulation de Delaunay d'une rsgp non-hiérarchisée

Les deux principaux résultats dans ce domaine sont dus à L.P. Chew [CHE 87] et C.A. Wang & L. Schubert [S/W 87], qui présentèrent leurs méthodes au même Congrès de Géométrie Algorithmique de l'A.C.M. sans qu'on puisse dire qu'il s'agisse exactement du même sujet: Wang et Schubert s'intéressent à la triangulation de Delaunay d'un ensemble de segments du plan par le biais d'une structure généralisant la notion de diagramme de Voronoi sur des segments, appelée *diagramme de Voronoi borné*. Les deux auteurs proposent une méthode optimale pour ce problème, de complexité $O(n\log n)$. Chew s'intéresse à la triangulation directe d'un graphe planaire non hiérarchisé et propose une méthode directement liée à la méthode de construction par fusion de Lee & Schachter. La méthode qu'il obtient est aussi optimale, puisque de complexité $O(n\log n)$.

Pourquoi, alors, chercher à faire 'mieux'? Principalement parce que les deux algorithmes présentent certaines difficultés d'implantation et que l'on pressent qu'il doit bien exister une méthode 'médiane' et simple pour résoudre le problème. Bien entendu, il n'est pas question de trouver une méthode dont la complexité asymptotique serait meilleure que la complexité optimale de $O(n\log n)$, mais seulement de tirer les enseignements de ces deux méthodes, et bien sûr de ce qui a été patiemment mis sur pied depuis le début de ce chapitre, pour mettre au point une technique si possible plus simple résolvant le problème avec la même complexité théorique.

L'algorithme proposé dans [S/W 87] est, d'une certaine façon, moins général que celui de [CHE 87] et certainement plus complexe à mettre en œuvre. Malgré tout, l'article mentionné contient un résultat important sur la relation entre le diagramme de Voronoi et la triangulation de Delaunay d'une rsgp, qu'il sera utile de signaler.

Le lecteur n'a pas dû manquer de remarquer que l'idée principale derrière les algorithmes proposés dans ce chapitre, est que la méthode 'diviser-pour-résoudre' se transpose d'une manière très simple en méthode incrémentale pour la résolution des mêmes problèmes. Cette propriété est encore vraie ici, dans une certaine mesure, et c'est donc la méthode par fusion de L.P. Chew qui va servir de fil conducteur. Signalons que la technique de division par bandes de Chew s'inspire d'une méthode imaginée par C.K. Yap [YAP 84] pour la construction du diagramme de Voronoi d'un ensemble d'arcs de courbes simples.

Construction d'une triangulation contrainte par fusion

On va donner ici les grandes lignes de l'algorithme afin d'y puiser l'idée centrale, selon laquelle il n'est pas nécessaire de garder toute l'information relative aux arêtes de contrainte, à tout moment.

La première étape de la construction consiste à préparer le terrain pour la récursion. Il est nécessaire de diviser le graphe en sous-graphes de tailles sensiblement égales et pour ce faire, Chew suggère de répartir, après un tri lexicographique des sommets de la rsgp, chaque sommet du graphe initial dans une bande verticale de largeur non nulle: ceci suppose qu'il n'y ait pas deux points sur une même verticale.

Il y a deux façons de voir les choses. Théoriquement, ceci ne pose pas de problème; l'auteur donne même une solution à ce problème en suggérant de faire subir au graphe une rotation telle que l'on ne rencontre plus deux sommets à la même abscisse.

Du point de vue pratique, il est *hors de question* de se résoudre à de telles extrémités puisque la moindre rotation fait intervenir des fonctions transcendantes et introduit dans les données des niveaux d'imprécision dont on peut penser qu'elles sont exemptes à l'origine! D'un autre côté, il paraît naturel de penser que l'introduction de cette restriction n'a qu'un but: faciliter l'analyse de complexité dans le pire des cas. Il est vrai que *même* la construction par fusion de la triangulation non contrainte de S_n (ou du diagramme de Voronoi) nécessite une assez grande attention en ce qui concerne la séparation des ensembles.

En fait, on sait depuis [KIR 79] que la fusion de deux diagrammes de Voronoi de tailles n et m non nécessairement séparables par une droite peut se faire en $O(m+n)$. En quelque sorte, la restriction imposée ici est purement formelle. Il se peut, malgré tout, qu'une implantation trop naïve de l'algorithme, ne tenant pas compte, notamment, du cas particulier de points de même abscisse, ait une complexité quadratique, dans le pire des cas.

Ceci dit, acceptons de bonne grâce les conditions imposées. Une fois cette répartition effectuée, il est nécessaire, à l'aide d'un algorithme de balayage du plan et donc de complexité $O(n\log n)$ de déterminer quel est le voisin directement inférieur et directement supérieur de tout sommet isolé ou extrémité d'arête du graphe: ceci permet de ne garder, dans une bande initiale donnée, que les deux portions de segments qui 'encadrent' le sommet central. Le quadrilatère initial qui encadre la bande initiale $B(p_i)$ s'appelle la *zone utile* de la région.

Lorsque l'on va désirer fusionner deux bandes adjacentes, on devra effectuer deux opérations totalement disjointes: d'une part la fusion traditionnelle des sous-triangulations (bien que contraintes) et d'autre part le 'recollement' de deux bandes adjacentes en une seule.

La seconde opération est la plus simple. Prenons les choses au début: lorsque l'on recolle deux bandes adjacentes initiales, celles-ci contiennent un sommet du graphe chacune, éventuellement origine ou fin d'une arête 'tronquée' par le bord droit ou gauche de sa bande et deux segments *plancher* et *plafond*, supposés indiquer les arêtes de contrainte les plus proches qui *traversent*, en-dessous et au-dessus de l'élément central, la bande qui l'accompagne: en fait, il se peut très bien que l'un des deux segments plancher ou plafond soit inexistant, simplement parce que le sommet n'a pas de voisins directs, au-dessus et en-dessous! Pour parer à ceci, on décide que le graphe est inclus dans un rectangle de très grande taille et les segments plancher et plafond sont éventuellement 'empruntés' aux bords de ce rectangle.

Sur tous les graphiques de cette sous-section, les extrémités cerclées indiquent des points virtuels, comme les extrémités des deux segments précédents. La figure VI.17 décrit plusieurs cas de figure pour des bandes initiales.

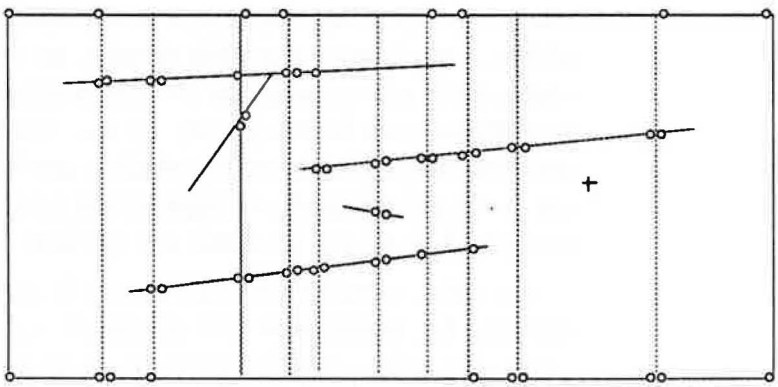


Figure VI.17. Bandes initiales et segments d'encadrement.

Comme les régions utiles de deux bandes initiales adjacentes ne sont pas nécessairement 'alignées' on est obligé de construire une sorte d'union: on garde le (et l'on prolonge éventuellement dans la bande adjacente le support du) segment plafond le plus bas des deux et identiquement pour le segment plancher le plus haut des deux. Ceci donne une région bordée de nouveau par deux segments plancher et plafond, contenant cette fois-ci deux sommets du graphe. Il va de soi que si deux segments tronqués se recollent ils fusionnent un seul segment. Ce serait le cas, par exemple, dans le recollement des bandes 6 et 7 (à partir de la gauche) dans *f*-VI.17. Ceci ne nécessite pas d'avoir intersecté réellement les arêtes du graphe avec les bordures de bandes, mais d'avoir 'entériné' la séparation des arêtes originelles en sous-arêtes collatérales.

Enfin, il se peut que, dans le recollement de deux bandes adjacentes, une arête traversante soit plancher ou plafond d'une bande mais pas de l'autre; ceci serait le cas dans le recollement de bandes initiales 8 et 9 de *f*-VI.17. On constate que, dans ce cas, il n'existe pas d'adjacence dans la suite de sommets virtuels. Malgré tout, le 'chaînage' des sous-segments peut être fait 'en référence' à une même arête canonique, ce qui permet de recoller les deux bouts de segments sans aucun problème. En revanche, il est important de rajouter autant de sommets virtuels que nécessaire après le recollement.

Il est maintenant évident que les recollements font grossir la 'taille' des bandes et qu'à la fin de l'algorithme, la bande finale est le rectangle entier. On a vu, implicitement, qu'il existe deux sortes de sommets virtuels: ceux qui sont liés à la segmentation des arêtes originelles par les bordures verticales et ceux qui sont extrémités de segments plancher ou plafond, eux-mêmes virtuels: les sommets du premier type sont véritablement muets, dans la mesure où ils ne jouent qu'un rôle de témoin passif: ils ne sont là que pour signaler le recollement entre deux sous-segments.

En revanche, les extrémités virtuelles de segments plancher et plafond jouent le rôle primordial de points à l'infini: il sont destinés à compléter une enveloppe convexe localement imprécise (par manque d'information sur les extrémités des segments de contrainte traversant la bande courante) et à servir de témoin de présence *locale* d'arêtes de contraintes dont les extrémités sont trop éloignées pour être vraiment utiles: pour donner une idée simple, un triangle candidat pour faire partie de la triangulation de Delaunay peut être invalidé par une arête traversant tout le rectangle; ce qui compte le plus n'est pas la localisation propre de ses extrémités que la proximité de son passage *vis-à-vis* des arêtes du triangle. C'est cette idée très importante qui a permis à L.P. Chew d'obtenir son résultat.

En effet, considérons maintenant la fusion de deux sous-triangulations. La répartition des sommets du graphe dans des bandes verticales adjacentes de taille un, et le processus de recollement de bandes adjacentes permettent d'affirmer que la fusion s'opère effectivement sur des sous-triangulations séparables par une droite verticale.

Avant de présenter l'algorithme de fusion, imaginons un sommet p dans une bande. Si celle-ci contient une arête de graphe, cette arête peut être entièrement incluse, à demi-incluse ou traversante. Les deux premiers cas ne posent pas de problème, comme on le verra. Considérons donc une arête de graphe traversant de bout en bout la bande courante. Le sommet p peut être situé au-dessus de l'arête (on rappelle qu'il ne peut y avoir d'arêtes verticale, puisqu'il n'y a pas deux abscisses identiques dans le graphe) ou en-dessous de celle-ci.

Dans le premier cas, p aura tendance à considérer que l'arête est, localement, un segment plancher infranchissable, et un segment plafond infranchissable dans le second: en effet, si cette arête de graphe se trouve entre lui et un autre site, rendant ce dernier invisible, il est clair que le segment qu'ils déterminent ne peut être inclus dans la triangulation contrainte. Quelles que soient les positions effectives des extrémités de l'arête de graphe traversante, elle joue le rôle d'écran pour p si tant est que ce dernier ait pu être choisi dans le processus de fusion et que l'arête lui masque la visibilité.

Il se trouve que l'algorithme de balayage initial a permis de déterminer les voisins directs supérieur et inférieur de tout sommet du graphe, mais aussi de toute arête du graphe, à la verticale de son extrémité initiale: c'est ce qui a été présenté dans le chapitre précédent. Ainsi, sans qu'il soit besoin de calculer effectivement les extrémités virtuelles des arêtes traversantes d'une bande, celles-ci ont été insérées réellement dans les listes d'incidence de leurs extrémités véritables. Par conséquent, au cours de la fusion on peut trouver en temps linéaire l'ordre vertical de succession des différents éléments de la bande.

Etudions l'exemple de fusion de la figure VI.18, afin d'apporter quelques précisions sur cette phase.

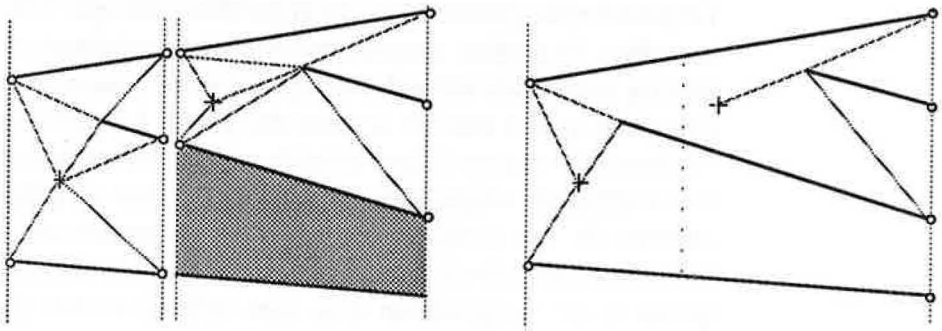


Figure VI.18. Fusion contrainte.

Le recollement des deux bandes provoque le recollement d'arêtes de contrainte: les sommets virtuels centraux disparaissent ainsi que toute arête de Delaunay qui était incidente en de tels sommets. Il suffit maintenant d'exécuter une fusion 'normale' à ceci près que la rencontre, dans la sous-triangulation gauche ou droite, d'une arête de contrainte incidente en ExtG ou ExtD provoque l'arrêt de la rotation autour de l'extrémité de base concernée, car il s'agit d'une contrainte que l'on ne doit pas franchir. Les tests sur les cercles portent sur les véritables coordonnées des extrémités effectives des segments de graphe tronqués.

Ceci permet de donner la synthèse de l'algorithme de fusion recherché. Les segments d'appui haut et bas sont les segments plafond et plancher dont les extrémités sont considérées comme des points à l'infini. La seule différence flagrante avec l'algorithme par fusion est la présence de deux clauses de test supplémentaires concernant la rencontre d'arêtes de G_n dans les listes d'incidence de ExtG et ExtD:

```
supprimer les triangles caducs de TDG et TDD;
T ← TDG ∪ TDD;                          ¡union de tous les triangles avant fusion¡
[ExtG,ExtD] ← segment d'appui bas [G_B,D_B];
tant que ([ExtG,ExtD] ≠ [G_H,D_H])
    G' ← extrémité de la 1ère arête incidente en ExtG 'suivant' la base;
    si (G' est au-dessus de la base [ExtG,ExtD])
        G" ← suivant direct de G' autour de ExtG;
        tant que (([G,G'] ∉ G_n) ∧ (ExtD ⊂ Γ(ExtG,G',G")))
            T ← T-triangle(ExtG,G,G');          ¡suppression d'une facette¡
            G' ← G";
            G" ← suivant direct de G' autour de ExtG;
    D' ← extrémité de la 1ère arête incidente en ExtD 'suivant' la base;
    si (D' est au-dessus de la base [ExtG,ExtD])
        D" ← suivant direct de D' autour de ExtD;
        tant que (([D,D'] ∉ G_n) ∧ (ExtG ⊂ Γ(ExtD,D',D")))
            T ← T-triangle(ExtD,D,D');          ¡suppression d'une facette¡
            D' ← D";
            D" ← suivant direct de D' autour de ExtD;
    si (G' est en-dessous de la base ou D' ⊂ Γ(ExtG,ExtD,G'))    ¡G' est OK¡
        T ← T ∪ triangle(ExtG,ExtD,G');
        ExtG ← G';                          ¡on retient la diagonale [G', ExtD]¡
    sinon                                                       ¡D' est OK¡
        T ← T ∪ triangle(ExtG,ExtD,D');
        ExtD ← D';                          ¡on retient la diagonale [ExtG,D']¡
retourner(T);
```

Algorithme VI-v: Fusion de deux sous-triangulations contraintes de Delaunay [CHE 87].

Le grand enseignement de cet algorithme est que l'on peut considérer les arêtes de graphe comme des objets temporairement virtuels. Le fait que les extrémités virtuelles de ces arêtes ne soient jamais explicitement calculées, soient traitées comme des points à l'infini, et soient à la fois totalement connues (leurs véritables coordonnées restent toujours celles des extrémités originelles du segment entier de graphe sous-jacent), permet de les intégrer comme points extrémaux des enveloppes convexes reconstruites dans la fusion. Dans la représentation visuelle qu'on a de l'algorithme elle sont effectivement présentes sur les bordures verticales; elles sont aussi présentes, pour chaque extrémité gauche ou droite de la base [ExtG, ExtD], et comme telles sont considérées comme des points à $+\infty$ ou $-\infty$ en ordonnée, suivant leur position par rapport à ces extrémités; elles sont aussi parfaitement matérielles dans l'ordre local à la verticale d'une abscisse donnée, ce qui permet de pouvoir les situer dans les ordres polaires d'incidence; et pourtant, elles n'existent que dans les bandes extrêmes contenant leurs propres extrémités: en fait, on peut même dire que toute arête de graphe est divisée en deux sous-arêtes, dont chacune est matériellement active dans une et une seule bande initiale.

L.P. Chew montre que la complexité asymptotique de l'algorithme est $O(n\log n)$ si n est la taille de la rsgp car la procédure de fusion a une complexité propre $O(s + d)$ ou s est le nombre de sommets dans la nouvelle bande recollée et d le nombre de demi-arêtes.

Construction incrémentale d'une triangulation contrainte

Il va maintenant être possible d'utiliser les enseignements de l'algorithme précédent pour l'adapter à la construction incrémentale d'une triangulation de Delaunay avec contraintes, moyennant quelques modifications structurelles et une adaptation des concepts précédents à la situation présente.

Tout d'abord, une chose ne va pas changer par rapport à l'algorithme incrémental de construction d'une triangulation sans contraintes: la construction de l'enveloppe convexe n'a que faire de la présence de segments reliant des couples de sommets du graphe. Elle sera construite exactement comme précédemment, dans l'ignorance heureuse de ces éléments.

D'autre part, il ne sera pas besoin de découper le graphe en bandes verticales ni de considérer qu'il sera inclus dans un rectangle. Ceci est une bonne surprise. En revanche, on va supposer, pour faciliter les explications, que l'on ne trouve pas deux sommets de même abscisse. Encore une fois, cette restriction peut être levée sans impliquer de changements profonds.

On supposera qu'un algorithme de balayage, de complexité $O(n\log n)$ a été utilisé (par exemple pour rendre le pré-graphe planaire) qui a permis d'affecter, à chaque sommet de G_n, une arête de contrainte (donc appartenant à G_n) voisine immédiate supérieure et une arête de contrainte voisine immédiate inférieure. On a vu au chapitre précédent comment ce genre d'opération pouvait être effectuée. Cette information sera porteuse de renseignements très précieux concernant l'ordre polaire à la verticale des sommets. Si l'on trouvait plusieurs segments de contrainte qui s'intersectaient juste au-dessus ou au-dessous d'un sommet donné, il suffirait de choisir le segment le plus bas dans un intervalle infinitésimal à droite de l'abscisse du sommet.

Il est clair que lorsque la rsgp est *donnée* comme polygone monotone (ou convexe), ce 'pré-traitement' redevient linéaire car la monotonie implique que toute perpendiculaire à la direction de monotonie ne peut intersecter la frontière du polygone que deux fois, et ceci est vrai dans toute direction pour un polygone convexe.

Supposons qu'on ait choisi trois sommets A, B et C de G_n rangés dans l'ordre lexicographique (en particulier, leurs abscisses vérifient $x_A < x_B < x_C$). Supposons enfin que les arêtes canoniques de AB et AC se trouvent dans cet ordre dans le sens direct autour de A (figure VI.19) et que l'on se pose la question: "Le triangle ABC est-il un triangle de Delaunay?". Démontrons le:

Lemme 1: Dans les conditions précédentes, s'il existe une arête quelconque [A,X] de G_n entre [A,B] et [A,C], le triangle ABC ne peut faire partie de la triangulation de Delaunay contrainte de G_n.

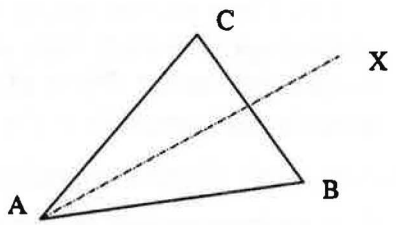


Figure VI. 19. Illustration du lemme 1.

Démonstration: Si X est à l'intérieur du cercle circonscrit à ABC, le lemme est démontré aisément. Sinon, il l'est aussi car [A,X] cache nécessairement C de B et réciproquement: ceci revient à dire que l'arête [C,D] coupe obligatoirement l'arête de contrainte [A,X], ce qui est interdit dans une triangulation contrainte.

Ce que permet de conclure ce lemme est que, si [ExtG, p_i] constitue le segment de base courant et si l'on apprend qu'entre cette ligne de base et l'arête canonique de [ExtG,G'] il y a *au moins* un segment de contrainte incident en ExtG, comme ce segment ne peut avoir une extrémité inférieure à celle de p_i sans être déjà englobé dans la triangulation, puisque p_i est, couramment, le point le plus à droite, cette arête de contrainte rend le triangle ExtG,G,G' inqualifiable pour le grade de triangle de Delaunay. Si l'abscisse de l'extrémité finale de cette arête de contrainte est inférieure à celle de p_i, elle aura en fait provoqué un mécanisme de blocage qui va être décrit plus loin.

Corollaire: Lorsque l'on insère le sommet p_i, la seule arête de contrainte d'origine G et différente de [ExtG,G] qui permette de fermer un triangle sur la base [ExtG,p_i] est une arête d'extrémité p_i lui-même.

En effet, toute autre arête de contrainte d'origine G serait un obstacle, par suite du lemme 1.

On remarque que la condition du lemme n'est *pas* nécessaire pour qu'un triangle soit rejeté de la triangulation: il existe d'autres motifs de rejet que l'on sait déjà bien traiter. En revanche, si cette condition est vérifiée, il est impossible que le triangle fasse partie de la triangulation contrainte.

On verra plus loin que l'information engrangée dans le balayage va permettre de pouvoir décider, à tout moment, si une arête de contrainte interdit la reconstruction de triangles de l'algorithme incrémental (situation dite de *blocage contraint*) ou si la voie est libre.

La question qui se pose maintenant est la suivante: le fait de ne pas reconstruire intégralement la triangulation à cause du lemme 1 va introduire une différence non négligeable entre la frontière de la triangulation 'impossible à compléter pour l'instant' et celle de l'enveloppe convexe, qui est, elle, toujours parfaitement cohérente. Ceci ne va-t-il pas provoquer un coût bien plus important: combien de fois peut-on affirmer que l'on va buter sur la même impossibilité liée au lemme 1, et donc ne plus pouvoir bénéficier de la complexité idéale précédente?

La réponse vient justement du fait que l'on s'attache à construire fidèlement l'enveloppe convexe: de même que les sommets virtuels jouent un rôle fondamental dans l'algorithme de Chew, de même ici le fait que d'une enveloppe convexe à la suivante, on ne cherchera à appliquer la procédure de mise à jour de la triangulation que le long des

segments visibles depuis p_i vont permettre d'attendre que les segments qui jaillissent de l'enveloppe convexe aboutissent et débloquent les situations bloquées sans qu'il ait jamais été question de retoucher des triangles dont on peut imaginer qu'ils avaient, jusqu'au moment d'insertion de leur troisième sommet, un sommet virtuel.

Notons que la condition d'invalidation consécutive au lemme 1 clôt la recherche autour du sommet ExtG courant, de même que la rencontre d'une arête [ExtG,G] de contrainte, comme dans la procédure de fusion de L.P. Chew.

Il faut maintenant étudier la situation suivante: p_i est le point à insérer dans la méthode incrémentale, ExtG est le sommet courant considéré dans la triangulation, G et G' les deux successeurs de ExtG dans le sens direct autour de ExtG et une arête de contrainte [U,V] coupe l'un ou l'autre des deux segments [ExtG,p_i], [G,p_i] (figure VI.20).

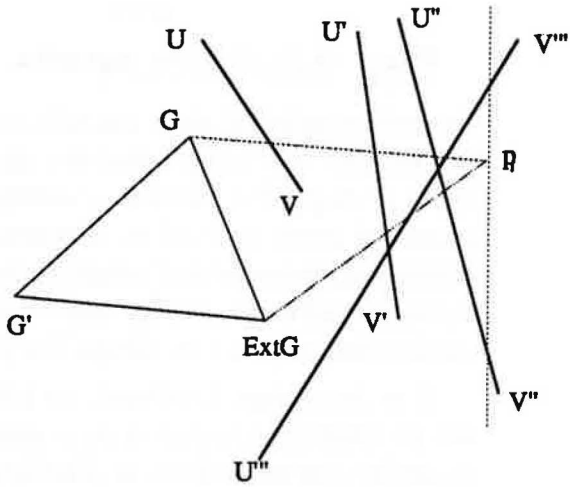


Figure VI.20. Arêtes de contrainte bloquantes.

En fait, le nombre de situations possibles est limité: les arêtes de type [U,V] et [U',V'] ne sont pas une menace car, de toute évidence V comme V' ont des abscisses inférieures à celle de p_i et le mécanisme de construction incrémentale a déjà formé un blocage contraint qui empêche l'examen de [ExtG,G] pour p_i. Le segment de contrainte de type [U'",V'"] n'est pas non plus une menace: si l'abscisse de V'" était inférieure à celle de p_i, V'" aurait été déjà inséré et serait un ExtG *avant* celui qui est représenté sur le graphique, empêchant tout problème; dans le cas où V'" est à droite lexicographiquement de p_i, l'arête de contrainte [U'",V'"] (ou une plus proche) est détectée au-dessus de p_i, donc en situation de blocage et l'arête [ExtG,p_i] ne peut être considérée pour passer les tests d'élimination. Enfin, l'arête de type [U",V"] permet la même analyse, car elle est détectée (ou une autre plus proche) cette fois-ci en-dessous de p_i.

Pour revenir à [U,V], cette arête ne pose pas de problème car V est *forcément* à l'intérieur du disque circonscrit à ExtG,p_i, et est donc remis en cause directement. Ce qui est exploité ici est la combinaison de la notion de plus proche sommet et de l'ordre des arêtes de contrainte à la verticale de p_i.

Afin de permettre déterminer la nature des tests à utiliser, examinons une situation qui serait *potentiellement* problématique. Sur la figure VI.21, les sommets p_i et ExtG n'ont pas de voisin vertical commun. De plus, le segment [ExtG,p_i] ne coupe ni le voisin inférieur de p_i, ni le voisin supérieur de ExtG, mais une arête de contrainte s 'intercalée entre les deux.

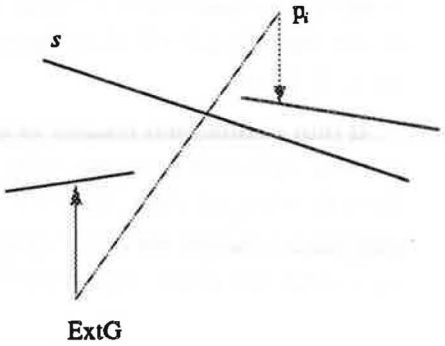


Figure VI.21. Situation impossible.

La seule possibilité, dans une telle situation est que la fin de v.s.(ExtG) et l'origine de v.i.(p_i) aient des abscisses comprises entre celles de ExtG et de p_i. Par chance, l'insertion incrémentale des sommets p_i permet d'avoir localisé et 'circonscrit' le problème créé par s , par exemple au moment de l'insertion de l'origine du voisin inférieur de p_i. Ainsi, il suffit de vérifier que [ExtG,p_i] ne coupe jamais ces deux voisins directs pour être sûr qu'il n'y a pas de situation de blocage.

Il ne reste plus, à présent, qu'à formaliser le test d'acceptation du couple (ExtG,p_i), qui peut se décomposer en une série de conditions, mutuellement exclusives: si [ExtG,p_i] est une arête de G_n, aucune autre ne peut la rencontrer car le graphe est planaire et l'on peut procéder au test du cercle relatif à [ExtG,G]; sinon, considérons d'abord le cas où p_i est au-dessus de ExtG: si ExtG n'a pas de voisin supérieur ou si p_i n'a pas de voisin inférieur, ou si le voisin supérieur de ExtG est l'arête de contrainte [G,p_i], l'arête [ExtG,G] peut être soumise aux tests de Delaunay; sinon, si le voisin supérieur de ExtG est égal au voisin inférieur de p_i, forcément d'extrémité différente de p_i, on doit ignorer l'arête [ExtG,p_i] et passer de ExtG à G (*blocage*); sinon, si le voisin supérieur de ExtG *est* [G',G], l'arête [ExtG,G] peut être soumise au test du cercle car ExtG est 'satisfait' (le triangle *contraint* ExtG,G,G" est immuable); sinon, enfin, si le segment [ExtG,p_i] intersecte le voisin supérieur de ExtG ou inférieur de p_i, l'arête [ExtG,p_i] doit être ignorée et elle peut passer les tests d'élimination sinon. Un raisonnement analogue s'applique au cas où p_i est en-dessous de ExtG.

Ce test est plus lourd que les précédents, mais ne prend qu'un temps constant. Dans la synthèse présentée plus loin, ce test porte le nom de *Test_Candidat* et retourne la valeur vraie si le couple (ExtG,p_i) permet à l'arête ExtG,G de passer les tests d'élimination habituels. Examinons le déroulement de la méthode proposée sur un exemple, afin de pouvoir en commenter les principales étapes. La figure VI.22 montre la rsgp que l'on veut trianguler et les relations de 'voisinage'.

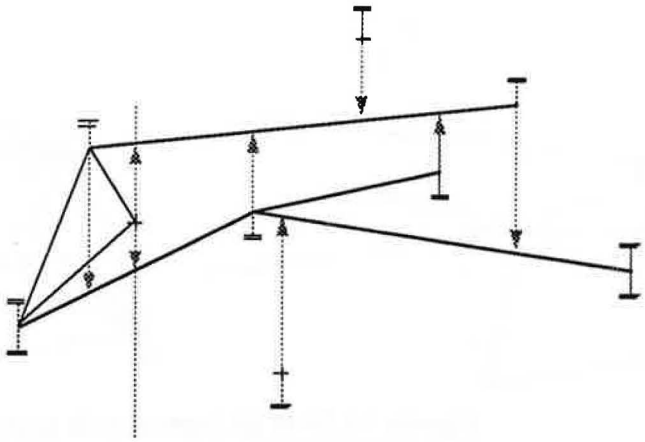


Figure VI.22. Triangulation d'une rsgp. Les symboles (⊤) représentent l'absence de voisin et les flèches pointent sur le voisin directement supérieur ou inférieur de chaque sommet.

La phase de construction de triangulation initiale peut être elle-même inhibée par la présence d'un segment de contrainte incident en p_1 et de pente comprise entre celles de $[p_1.p_3]$ et de $[p_1, p_2]$:il se peut donc que la première triangulation soit vide. A l'insertion du quatrième point, on ne rencontre aucune difficulté: tout se passe comme si l'on triangulait le nuage de sommets uniquement. L'insertion du cinquième sommet, figure VI.23, introduit le premier cas particulier.

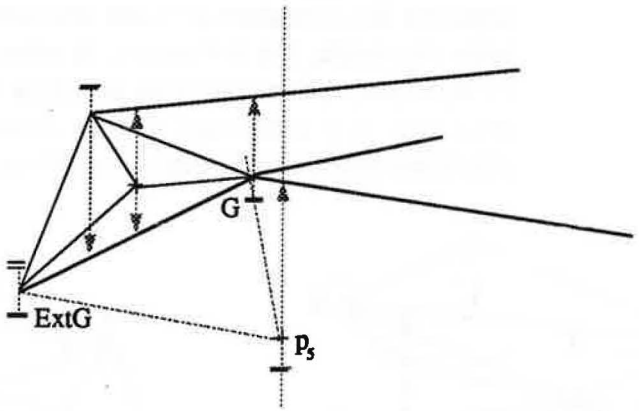
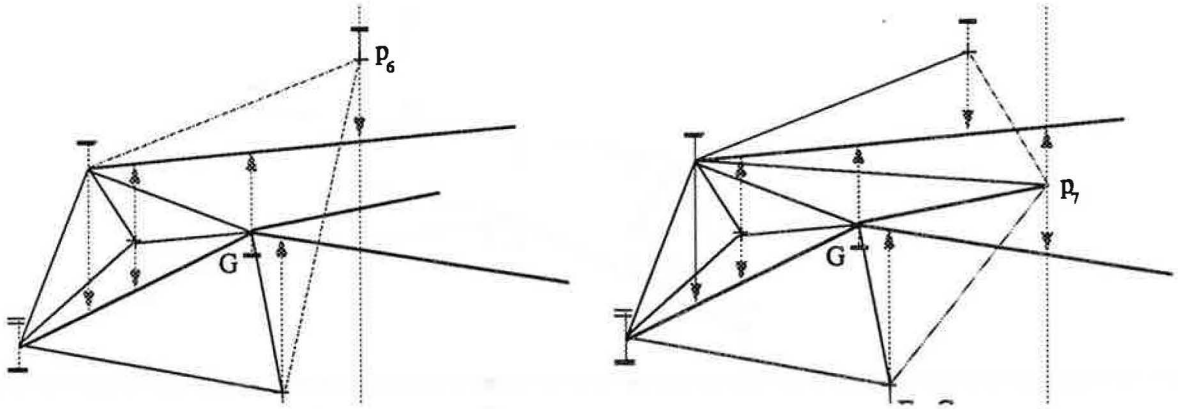


Figure VI.23. Insertion de p_5.

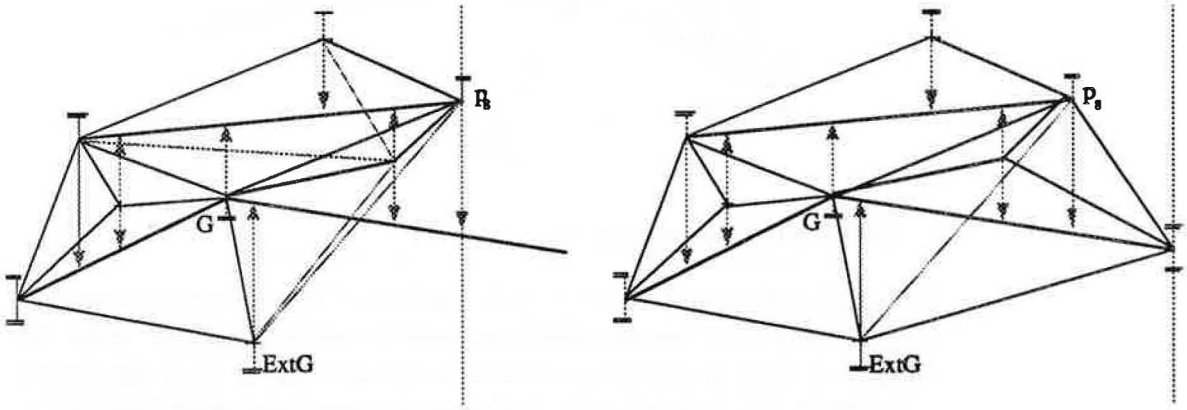
Le segment d'appui bas est $[p_1, p_5]$, ExtG=p_1, G=p_4, G'=p_3; le sommet inséré n'a pas de voisin inférieur, mais [ExtG,G] est une arête de contrainte (ceci se reconnaît en temps constant par le biais des arêtes canoniques, par exemple). On doit donc accepter le triangle ExtG,p_5,G. Le point d'appui haut étant G, on s'arrête après. C'est avec l'insertion du sixième sommet, figure VI.24, que l'on entre dans le vif du sujet.



Figures VI.24 et 25. Insertions de p_6 et p_7.

ExtG=p_5, G=p_4, p_6 admet un voisin direct inférieur différent du voisin direct supérieur de ExtG, et [ExtG,p_6] intersecte les deux. On remonte le long de l'enveloppe convexe et l'on remplace G par ExtG. Cette fois-ci, les deux sommets ont le même voisin, mais, sa présence signifie obstacle, tant que son extrémité finale est 'virtuelle' (blocage). On passe et la remontée se termine sur le point d'appui haut. L'insertion du septième sommet (figure VI.25) va changer l'état des choses quelque peu.

ExtG connaît le même blocage que précédemment (mêmes voisins, mais fin virtuelle). On passe directement au moment ou ExtG devient G, G devient p_2 et G' devient p_3 sur le graphique: [ExtG,p_7] est un segment de contrainte; ceci est une condition permettant de lancer les tests de cercle. En définitive, le triangle ExtG,G,G' est ajouté à la triangulation et l'on passe au prochain sommet sur l'enveloppe, p_6. De nouveau, il y a blocage car on rencontre une arête de contrainte intercalée (voisins inférieur et supérieur identiques, fin virtuelle).



Figures VI.26 et 27. Insertion des deux derniers sommets.

A l'insertion de p_8, figure VI.26, [ExtG,G] est ignorée pour des raisons connues; pour le prochain point sur l'enveloppe convexe, le voisin supérieur de ExtG est le segment de contrainte [G,p_8], donc on passe aux tests d'élimination qui remettent en cause l'arête [p_1, p_7] et imposent son remplacement par [p_4, p_8]. Enfin, le dernier sommet vient combler toutes les faces manquantes (figure VI.27).

Les opérations que provoquerait l'algorithme incrémental VI.iv ne sont pas exécutées du tout lorsqu'un segment de contrainte crée un blocage. Ainsi, lorsque les triangles attendant une arête bloquante de ce type sont réellement fermés, on peut affirmer sans erreur que le coût de l'opération est prélevable par le crédit précédent non utilisé.

Fondamentalement, l'algorithme de construction incrémentale d'une triangulation de Delaunay avec contrainte diffère peu de l'algorithme VI.iv: on rajoute le même test d'arrêt de recherche autour de ExtG que celui de l'algorithme VI.v et l'on inclut le *Test_Candidat* présenté plus haut. Enfin, l'algorithme résumé ci-dessous procède à un balayage afin d'établir les relations de voisinage direct supérieur et inférieur de chaque sommet vis à vis des segments de contrainte et construit la première triangulation: celle-ci est vide s'il existe un segment de contrainte entre les deux arêtes initiales et est égale au triangle p_1,p_2,p_3 sinon. La notation v.i.(pt) désigne le segment de contrainte voisin inférieur immédiat du sommet pt, détecté lors du balayage.

```
trier S_n pour obtenir l'ensemble {p_1, p_2, ..., p_n};
balayer G_n pour établir les relations de v.s. et v.i;
si ((p_2 est au-dessus de p_3))
   | si (v.i.(p_2) ≠ [p_1, p_3])
   | T ← NIL;
   | sinon
   | T ← triangle(p_1,p_2,p_3);
sinon
   | si (v.i.(p_3) ≠ [p_1, p_2])
   |   T ← NIL;
   | sinon
   | T ← triangle(p_1,p_2,p_3);
EC ← triangle p_1p_2p_3;
pour (i=4; i ≤ n; i++)
   | (EC, H, B) ← MajEC(p_i, EC);
   | [ExtG,ExtD] ← segment d'appui bas [B,p_i];
   | si (Test_Candidat(p_i,ExtG)
   |    | tant que (([G,G'] ∉ G_n) ∧ ([ExtG,ExtD] ≠ [H,p_i]))
   |    |    | G' ← extrémité de la 1ère arête incidente en ExtG 'suivant' la base;
   |    |    | si (G' est au-dessus de la base)
   |    |    |    | G" ← suivant direct de G' autour de ExtG;
   |    |    |    | tant que (ExtD ⊂ Γ(ExtG,G',G"))
   |    |    |    |    | T ← T-triangle(ExtG,G,G');      ¡suppression d'une facette¡
   |    |    |    |    | G' ← G";
   |    |    |    |    | G" ← suivant direct de G' autour de ExtG;
   |    | T ← T ∪ triangle(ExtG,ExtD,G');
   |    | ExtG ← G';
retourner(T);
```

Algorithme VI-vi: Construction incrémentale d'une triangulation de Delaunay d'une rsgp, ou triangulation contrainte.

On constate aisément dans l'algorithme VI.vi que l'on ne pénètre dans l'intérieur de la triangulation que lorsque aucune arête de contrainte ne s'intercale à l'intérieur de l'éventuelle facette créée sur [ExtG,p_i]. Ceci confirme bien le fait que, dans ces cas, on remonte de position en position sur *l'enveloppe convexe* sans passer en revue les éventuelles facettes 'creuses' que l'on ne peut compléter sans trouver l'extrémité finale du segment de contrainte qui empêche leur mise à jour.

Les facettes qui ne sont pas créées, alors que l'algorithme sans contrainte les créerait, sont définitivement créées en une seule fois, puisqu'elles contiennent une arête de contrainte non-interchangeable par la suite dans un quadrilatère quelconque. Le coût de cette opération peut être 'prélevé' sur le crédit occasionné auparavant.

Comme le nombre d'arêtes de toutes les triangulations d'un même ensemble de sommets (graphe ou non) est le même, on en déduit que la complexité générale de l'algorithme est $O(n\log n)$, puisque le nombre d'opérations effectuées, sur la partie incrémentale pure, est toujours linéaire, comme $Conv_n$.

On remarque certainement encore la puissance des algorithmes de balayage qui permettent de jeter un pont entre l'ordre lexicographique (principe d'insertion incrémentale) et l'ordre polaire (ordre local à la verticale d'un sommet). Il est bien entendu possible de combiner le balayage et la procédure incrémentale pour obtenir, globalement, une méthode de construction par balayage des triangulations de Delaunay contraintes.

Diagramme de Voronoi et triangulation de Delaunay avec contraintes

[S/W 87] présente la notion de diagramme de Voronoi *borné*. Il s'agit, en fait du diagramme de Voronoi que l'on obtient en 'arrêtant' les médiatrices qui intersectent des arêtes de contrainte à ce point d'intersection. Cette notion découle de celle de distance bornée, dans laquelle la distance de deux sommets du graphe n'est définie que si ces deux segments se voient mutuellement. La figure VI.28 donne l'exemple du diagramme de Voronoi borné correspondant à la rsgp précédente.

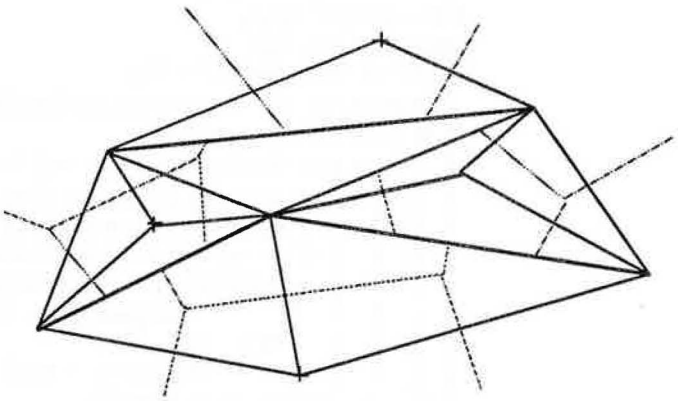


Figure VI.28. Diagramme de Voronoi borné de la rsgp précédente.

Le polygone de Voronoi de chaque site n'est plus toujours convexe car ses arêtes peuvent suivre des sections d'arêtes de contrainte.

De même, la dualité n'est plus aussi triviale que précédemment, mais les auteurs montrent qu'on peut effectuer le passage du diagramme de Voronoi borné à la triangulation contrainte de manière linéaire en reliant les sommets du graphe naturellement voisins et ceux du diagramme de Voronoi 'standard' dont les médiatrices sont coupées par une arête de contrainte. Inversement, on passe de la triangulation en parcourant chaque triangle, en construisant les médiatrices des arêtes et en les arrêtant lorsqu'elles butent contre une arête de contrainte.

VI.B.2 Triangulation de Delaunay d'une rsgp hiérarchique

Dans cette dernière et très courte sous-section, on considère une rsgp hiérarchique comme celles qui ont été étudiées au chapitre précédent. On cherche à obtenir une triangulation de Delaunay de cette structure, sachant que la triangulation doit explicitement faire intervenir la frontière de la face externe de la rsgp. Trois méthodes seront décrites très brièvement, ainsi que des directions de recherche pour des travaux à venir.

On constate une difficulté immédiate dans la résolution du problème posé: il ne semble plus possible d'appliquer une méthode incrémentale fondée sur la construction de l'enveloppe convexe, puisque celle-ci n'a plus tellement de rapport avec la triangulation recherchée.

La première solution qui vient à l'esprit consiste à inclure la rsgp dans un rectangle englobant et à trianguler l'ensemble sans se soucier de la structure hiérarchique, mais en respectant la frontière de chaque face et chaque segment de chaîne qu'elle contient comme autant de contraintes.

Chaque fois qu'un triangle est inséré dans la triangulation construite incrémentalement, l'information relative aux voisins inférieurs de ses sommets permet de retrouver, sans trop de difficultés, la face qui contient ce triangle dans l'arbre hiérarchique construit au chapitre précédent.

Il est simplement nécessaire d'effectuer une dernière passe, de complexité linéaire, pour éliminer les triangles qui couvrent le complémentaire de la rsgp dans le rectangle englobant: une fois ces triangles supprimés, l'ensemble obtenu est bien la triangulation de Delaunay contrainte de la rsgp, suivant sa face externe.

Une autre méthode a été traitée en détail par Leila de Floriani et son équipe de recherche [DF/F/P 85]: leur sujet d'étude porte sur la triangulation d'un polygone simple et sans trous, contenant des sommets isolés, correspondant à des données altimétriques à l'intérieur d'une frontière de domaine, rentrées de manière aléatoire et dynamique.

La solution proposée consiste à travailler en deux temps: on commence par trianguler la frontière du polygone puis l'on rajoute un à un les sommets contenus dans l'intérieur du polygone.

On triangule le polygone en créant des arêtes de Delaunay entre des sommets mutuellement visibles et voisins de Voronoi. Les sommets isolés contenus dans l'intérieur du polygone sont ensuite insérés un à un et la triangulation de Delaunay est mise à jour en localisant le triangle contenant chaque nouveau point et en appliquant localement la technique d'échange de diagonale jusqu'à ce que l'on retrouve une triangulation de Delaunay comprenant le nouveau point.

Les techniques d'insertion de sommets dans un ordre arbitraire utilisent presque toutes une pile et ont une complexité quadratique ([DeF 87]).

D.T. Lee et A.K. Kin [K/L 86] proposent une définition générale des triangulations de rsgp, bien que celle-ci soit toujours définie par rapport à l'enveloppe convexe, étudient leurs propriétés et proposent un algorithme de complexité $O(n\log n)$ optimale pour construire la triangulation de Delaunay d'un polygone simple sans trous, cette fois-ci relativement à sa frontière.

L'algorithme proposé par ces auteurs dépend fortement d'un résultat important ([CHA 82]) qui permet de trouver de manière linéaire sur la frontière d'un polygone de n arêtes, deux sommets tels que le segment qu'ils définissent soit entièrement contenu dans le polygone et que la frontière des deux sous-polygones ainsi créés possède chacune au moins $n/3$ arêtes. L'article dont est tiré ce résultat suggérait une méthode de triangulation polygonale en $O(n\log n)$ par décomposition récursive du polygone originel en sous-polygones. Kin et Lee eurent l'idée de spécialiser l'algorithme de triangulation en lui greffant une technique similaire à la construction par fusion d'une triangulation de Delaunay d'un nuage de points, dont Lee était l'auteur.

Cette fois-ci, la technique utilisée est plus délicate à décrire, puisqu'il est nécessaire de faire attention aux possibles conflits de visibilité et d'ingérence de la frontière du polygone dans les cercles de Delaunay construits à partir de la base, au moment de la fusion.

Il paraîtrait logique de vouloir étendre cette technique à la résolution du problème de triangulation d'un polygone plein afin de permettre une solution récursive pour la triangulation face par face d'une rsgp hiérarchique. Ce travail, qui nécessite une adaptation, à des polygones pleins, du résultat sur le découpage polygonal, et de la méthode de construction par fusion interne, est un axe de recherche prometteur.

On peut aussi se demander s'il est possible de trouver un algorithme de balayage exploitant les propriétés de visibilité et permettant de construire la triangulation d'un polygone plein recherchée, en empruntant un peu au principe d'insertion incrémentale développé dans ce chapitre.

Conclusion

Dans cette thèse, on a vu qu'il était possible de recourir à une méthode efficace de calcul mixte, permettant de garder une cohérence totale sur la topologie des données à traiter. Une modélisation a été proposée pour représenter des entiers de taille arbitraire, qui s'accompagne d'algorithmes efficaces pour les opérations arithmétiques élémentaires.

On a aussi étudié quelques méthodes permettant la structuration des données soumises à des contraintes et à des règles sémantiques de priorité ou de dépendance. La dernière préoccupation a été la description d'algorithmes plus simples que les algorithmes déjà existants pour la triangulation d'une représentation par segments d'un graphe planaire.

Toutes ces méthodes sont nées d'algorithmes validés par une application en grandeur réelle, réalisée dans un contrat pour un industriel, et il est permis de penser que les techniques de la géométrie algorithmique seront de plus en plus sollicitées dans la résolution de problèmes de ce type à l'avenir.

Certaines questions posées dans cette thèse restent ouvertes pour lesquelles il semble que des solutions devraient venir assez rapidement, à partir des bases jetées dans ce travail.

Bibliographie Générale

Si un titre est précédé d'une astérisque, il provient d'une autre référence citée par ailleurs.

1. **A/G/S/S 89** A. Aggarval, L.J. Guibas, J. Saxe, P.W. Shor, *A linear-time Algorithm for Computing the Voronoi Diagram of a Convex Polygon*, Discrete and Computational Geometry, 4, pp. 591-604, 1989.

2. **A/H/U 74** A. Aho, J. Hopcroft, J. Ullman, *The Design and Analysis of Computer Algorithms*, Addison Wesley Publishing Company, 1974.

3. **A/H/U 87** A. Aho, J. Hopcroft, J. Ullman, *Structures de données et Algorithmes*, InterEditions, Paris, traduction de J.M. Moreau, 1987. Version originale: *Data Structures and Algorithms*, Addison Wesley, Reading, Mass., 1983.

4. **A/W 77** P. Atherton, K. Weiler, *Hidden Surface Removal Using Polygon Area Sorting*, Computer Graphics, Vol. 11, pp. 214-222, (Proc. SIGGRAPH 77), 1977.

5. **BER 58** C. Berge, *La théorie des graphes et ses applications*, Dunod, Paris, 1958.

6. **B/OT 79** J.L. Bentley, T. Ottman, *Algorithms for Reporting and Counting Geomeric Intersections*, IEEE Trans. Comp. C-28, 9, pp. 643-647, Sept. 1979.

7. **BRO 77** W.S. Brown, *A realistic Model of floating-point Computation*, in Mathematical Software III, Ed. John R. Rice, Academic Press, pp. 343-360, 1977.

8. **BRO 79** K. Brown, *Voronoi Diagrams from Convex Hulls*, Inf. Proc. Letters, Vol. 9, N° 5, 1979.

9. **CHA 82** B. Chazelle, *A Theorem on Polygon Cutting with Applications*, Proc. of the 23rd IEEE Annual Symposium on Foundations of Computer Science, pp. 339-349, 1982.

10. **CHA 90** B. Chazelle, *A Linear-time Algorithm for the Triangulation of a Simple Polygon*, Conférence aux Journées de Géométrie Algorithmique, Valbonne, 1990.

11. **C/E 88** B. Chazelle, H. Edelsbrunner, *An Optimal Algorithm for Intersecting Line segments in the Plane*, CS-TR 148-88, Department of Computer Science, Princeton University, 1988.

12. **CHE 87** L.P. Chew, *Constrained Delaunay Triangulations*, Proc. of the Third Annual Symposium on Computational Geometry, 1987. Version plus détaillée: *Constrained Delaunay Triangulations*, Algorithmica, 4: pp. 97-108, 1989.

13. **DeF 87** L. De Floriani, *Surface Representations Based on Triangular Grids*, Visual Computer, 3: 27, pp. 27-50, 1987.

14. **DF/F/P 85** L. De Floriani, B. Falcidieno, C. Pienovi, *Delaunay-based Representation of Surfaces Defined over Arbitrarily Shaped Domains*, Computer Vision, Graphics and Image Processing, 32, pp. 127-140, 1985.

15. **EDE 87** H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, Springer Verlag, New York, 1987.

16. **E/M 88** H. Edelsbrunner, E.P. Mücke, *Simulation of Simplicity: A Technique to Cope with Degenerate Cases in Geometric Algorithms*, Proceedings of the 4th ACM Symposium on Computational Geometry, pp. 118-133, 1988.

17. **FOR 86** S. Fortune, *A Sweepline Algorithm for Voronoi diagrams*, Proceedings of the 2nd ACM Symposium on Computational Geometry, pp. 313-322, 1986.

18. **G/K/L/N 83** I.G. Gowda, D.G. Kirkpatrick, D.T. Lee, A. Naamad, *Dynamic Voronoi Diagrams*, IEEE Trans. on Inf. Theory, Vol IT-29, N° 5, 1983.

19. **G/S 85** L. Guibas, J. Stolfi, *Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams*, Proceedings of the 15th ACM Conference on the Theory of Computing, pp. 221-234, 1983. Version révisée, mêmes auteurs, même titre, ACM Transactions on Graphics, Vol 4. No. 2, April 1985, pp. 74-123.

20. **G/S/S 89** L. Guibas, D. Salesin, J. Stolfi, *Epsilon Geometry: Building Robust Algorithms from Imprecise Computations*, Proceedings of the 5th ACM Symposium on Computational Geometry, pp. 208-217, 1989.

21. **G/Y 86** D.H. Greene, F. F. Yao, *Finite-resolution Computational Geometry*, 27th Annual Symposium of the Foundations of Computer Science, pp. 143-152, 1986.

22. **H/H/K 88** C. Hoffmann, J. Hopcroft, M. Karasick, *Towards Implementing Robust Geometric Computations*, Proceedings of th 4th ACM Symposium on Computational Geometry, pp. 106-115, 1988.

23. **H/W 79** G.H. Hardy, E.M. Wright, *An Introduction to the Theory of Numbers*, Oxford University Press, 1979.

24. **KAR 88** M. Karasick, *On the Representation and Manipulation of Rigid Solids*, Ph.D. Disseration, McGill University, 1988.

25. **KIR 79** D.G. Kirkpatrick, *Efficient Computation of continuous Skeletons*, Proc. of the 20th IEEE Symposium on the Foundations of Computer Science, pp. 18-27, 1979.

26. **K/L 86** A.K. Lin, D.T. Lee, *Generalized Delaunay Triangulation for Planar Graphs*, Discrete and Computational Geometry, Vol. 1, N° 3, pp. 201-217, 1986.

27. **K/L/N 89** M. Karasick, D. Lieber, L. Nackman, *Efficient Delaunay Triangulation using Rational Arithmetic*, IBM Research Report RC 14455, 1989.

28. **KM3 84** K. Mehlhorn, *Data Structures and Algorithms: Multi-dimensional Searching and Computational Geometry*, Vol. 3, Springer Verlag, Berlin, 1984.

29. **KII** D.E. Knuth, *Seminumerical Algorithms*, Addison Wesley, Reading, Mass., 1969.

30. **LAW 77** C.L. Lawson, *Software for C^1 Surface Interpolation*, Math. Software III, (J.R. Rice, Ed.), pp. 161-194, Academic Press, New York, 1977.

31. **LEE 78** D.T. Lee, *Proximity and Reachability in the Plane*, Ph.D. Thesis, Rep. R-831, Coordinated Sci. Lab. Univ. Illinois, Urbana, Ill., 1978.

32. **LEE 82** D.T. Lee, *On k-nearest neighbor Voronoi Diagrams in the Plane*, IEEE Trans. Comput. C-31, pp. 478-487, 1982.

33. **L/S 80** D.T. Lee, B.J. Schacter, *Two Algorithms for Constructing a Delaunay Triangulation*, Int. J. Comput. Inf. Sci., 9(3), pp. 219-242, 1980.

34. **MIC 87** D. Michelucci, *Les représentations par les frontières: quelques constructions; difficultés rencontrées*, Thèse de 3ème cycle, Ecole des Mines de Saint-Etienne, 1987.

35. **MIL 88** V.J. Milenkovic, *Verifiable Implementations of Geometric Algorithms Using Finite Precision Arithmetic*, Ph.D. Dissertation, CMU-CS-168, Carnegie-Mellon, 1988. Ou encore V.J. Milenkovic, *Verifiable Implementations of Geometric Algorithms Using Finite Precision Arithmetic*, Artificial Intelligence, 37 377-401, 1988.

36. **MIL 89a** V.J. Milenkovic, *Double Precision Geometry A General Technique for Calculating Line and Segment Intersections Using Rounded Arithmetic*, 30th Annual Symposium on the Foundations of Computer Science, 1989.

37. **MIL 89b** V.J. Milenkovic, *Robust Geometric Computations for Vision and Robotics*, DARPA, 1989.

38. **M/L 89** V.J. Milenkovic, Z. Li, *Constructing Strongly Convex Hulls Using Exact or Rounded Arithmetic*, ACM-SIAM Symposium on Discrete Algorithms, 1989.

39. **O/T/U 87** T. Ottman, G. Thiemt, C. Ullrich, *Numerical Stability of Geometric Algorithms*, Proceedings of the Third ACM Symposium on Computational Geometry, pp. 119-125, 1987.

40. **PER 88** B. Péroche, *Modèles de partitionnement par diagramme de Voronoi: aspects algorithmiques*, E.N.S.M.S.E. 88-7, 1988.

41. **P/S 85** F.P. Preparata, M.I. Shamos, *Computational Geometry - an Introduction*, Springer Verlag, New York, 1985.

42. **ROB 77** J. Roberts, *Elementary Number Theory: A Problem-oriented Approach*, MIT Press, 1977.

43. **ROG 85** D.F. Rogers, *Procedural Elements for Computer Graphics*, McGraw-Hill Book Company, New York, 1985.

44. **SED 90** R. Sedgewick, *Algorithms in C*, Addison Wesley Publishing Company Inc., 1990. Version française: *Algorithmes en C*, InterÉditions, traduction de J.M. Moreau, en cours (1990).

45. **SHA 78** M.I. Shamos, *Computational Geometry*, Ph.D. Dissertation, Dept. Comput. Sci. Yale University, New Haven, Conn., 1978.

46. **S/I 89a** K. Sugihara, M. Iri, *Two Design Principles of Geometric Algorithms in Finite-precision Arithmetic*, Appl. Math. Letter., Vol 2, No. 2, pp. 203-206, 1989.

47. **S/I 89b** K. Sugihara, M. Iri, *Construction of the Voronoi Diagram for One Million Generators in Single-precision Arithmetic*, Proceedings of the First Canadian Conference on Computational Geometry, 1989.

48. **S/I 89c** K. Sugihara, M. Iri, *A Solid Modelling System Free from Topological Inconsistency*, Research Memorandum RM 89-03, University of Tokyo, 1989.

49. **S/S 85** M. Segal, C. H. Séquin, *Consistent Calculations for solids modeling*, Proceedings of the First ACM Symposium on Computational Geometry, pp. 29-38, 1985.

50. **S/W 87** L. Schubert, C.A. Wang, *An optimal Algorithm for Constructing the Delaunay Triangulation of a Set of Line Segments*, Proc. of the Third Annual Symposium on Computational Geometry, pp. 223-232, 1987.

51. **T/VW 88** R.E. Tarjan, C.J. Van Wyck, *An O(nloglogn)-time algorithm for triangulating a simple polygon*, SIAM J. Comput., Vol 17, No.1, 1988.

52. **VOR 08** G. Voronoi, *Nouvelles applications des paramètres continus à la théorie des formes quadratiques: recherches sur les parallèloèdres primitifs*, J. Reine Angew. Math. 134, pp. 198-287, 1908.

53. **WIN 84** P.H. Winston, *Artificial Intelligence*, Addison Wesley, 1984. Version française, *Intelligence Artificielle*, InterEditions, traduction de J.M. Moreau, Paris, 1988.

54. **YAP 84** C.K. Yap, *An O(nlogn) Algorithm for the Voronoi Diagram of a set of simple curve segments*, Technical Report, Courant Institute, New York University, 1984. Version révisée: *An O(nlogn) Algorithm for the Voronoi Diagram of a set of simple curve segments*, Discrete and Computational Geometry, N°2, pp. 365-393, 1987.

55. ***YAP 87** C-K. Yap, *Symbolic Treatment of Geometric Degeneracies*, in Proceedings of the 13th IFIP Conference on System Modelling and Optimization, Chuo University, Tokyo, 1987. (← [YAP 88]).

56. **YAP 88** C-K. Yap, *A Geometric Consistency Theorem for a Symbolic Perturbation Scheme*, Proceedings of the 4th ACM Symposium on Computational Geometry, pp. 134-142, 1988. (→ [YAP 87]).

PUBLICATIONS PERSONNELLES

J.M. Moreau, M. Rubio *The Computation of Some Morphological Parameters*, Acta Stereologica, Proc. ICS VII Cæn, 6/III, pp. 1029-1034, 1987.

J.M. Moreau *A Simple Linear Algorithm for Polygon Visibility*, Proc. PIXIM 88, Paris, Ed. Hermès, pp. 151-161, 1988.

INDEX

A

Addition
- entiers 81
- rationnels 94

Algorithme
- addition de deux naturels 81
- Bentley-Ottman amélioré 127
- construction incrémentale
  - diagramme de Voronoi 213
  - triangulation de Delaunay 215
  - triangulation contrainte 229
- décomposition par priorité 160
- division de deux naturels 86, 92
- fusion de deux
  - diagrammes de Voronoi 196
  - triangulations de Delaunay 204
  - triangulations contraintes 222
- epsilon-vérification 102
- gestion des listes de test 135
- multiplication de deux naturels 84
- paramétrique 32
- puissances 36
- résolution de dépendances 142
- soustraction de deux naturels 82

Alignement 66, 99
Altimétrie 3
Anticipation 157
Angle 100-101, 170
Arbre
- binaire 110
- de décision 25, 33, 37
- de dépendances 138
- équilibré 126, 130

Arête 99, 105, 119-168, 197
- canonique 148-150, 154-160, 167
- duale 150, 154-160
- polygonale 149
- rentrante 152

Arc 121
Arithmétique
- algébrique 72
- entiers 80-93
- mixte 97-117
- rationnelle 93-96

B

Balayage 57-58, 109, 113, 123-136, 165
Base 76, 95
Bentley-Ottman 56, 123-136, 172

C

Canonique
- arête 148-150, 154-160, 167
- nœud 112
- représentant 112

Cascade 138-143
Chaînage 78, 85
Chaîne 121, 168
Chemin 121
Chiffre 76
Circuit 121
Circulation 175
Classe d'équivalence 111
Clipping: voir Décomposition
Cocyclique 101
Cohérence 58, 64
Comparaison
- entiers 80
- rationnels 93

Connexe 121
- composante 151-167

Construction incrémentale
- diagramme de Voronoi 208-215
- enveloppe convexe 206-208
- triangulation de Delaunay
  - contrainte 223-232
  - non contrainte 215

Contrainte 4, 13,168, 216 119-188, 177
- géométrique 168-188

Copie 116
Coplanaire 23, 101
Courbe cachée 63
Coût moyen, normal 25, 27

Critère
  angle minimum maximal 201
  cercle 201
Crochets 62
Cycle 121
  élémentaire 197

D

Débordement 79
Décomposition 11, 144-168
Dégénérescence 65, 70, 73, 106
Degré 121
Delaunay 13, 69, 93, 101, 175, 189-232
Dépendance 11, 52, 137-143
  sémantique 168-188
Déterminant 23, 100-101
Diagramme de Voronoi 68, 189-232
  borné 230
Dilaté 60-61
Disparition 159
Distance 34, 100
Diviser pour résoudre 84, 193, 202
Division
  entiers 85-93
  euclidienne 85
  rationnels 94
Données 9
Dynamique 102

E

En-tête 149
Entier 71, 76-96
Enveloppe convexe 192
Epsilon 59, 61, 102-117
Evénement 56
  imprévisible, prévisible 123-136
Expansion binaire 47, 51
Extrémité 121

F

Face 99, 105, 144-168
  externe 198
Facétisation 13, 189-232
Facettes 13, 183
Fiabilité 104-108, 117
Fraction continue 187
Frontière 144, 192
Fusion 193, 202, 218-223

G

Génération 56, 106, 129
Génératrice 170
Géométrie
  à epsilon près 61
  algorithmique 1-239, 50-74, 100
  robuste 62-64
Graphe 121
  sémantique 182
  triangulé 197
Grille 3, 55, 71

H

Hiérarchie 12, 160-167, 231

I

Identificateur 149
Imprécision 43-74
Incidente (arête) 99, 121, 164
Inclusion 161-167
Incohérence 58
Indice 111
Intérieur 144
Interpolation 11
Intersection 9, 11, 22-41, 56, 122-136, 144
  de *r*-3d 177-179
  normale, remarquable 26
Inversion 125
Irréductible 93

K

Knuth 75

L

Lexicographique 28, 56, 99, 166
Liste d'adjacence: voir Incidente (arête).

M

Majorant 105
Masque 95
Médiatrice 100, 191
Mémoire 80, 83, 91, 95, 103
Méta-résolveur 183
Minorant 105
Multiplication
  entiers 83
  nombre de 24-41, 101
  rationnels 94
Multiplicité 146-150

N

Normalisation 87

O

Opération
  élémentaire 99
Ordre 147
  lexicographique 28, 56, 99, 166
  naturel 149
  x-polaire 166

P

Palier 173-177
Paraboloïde 101
Paramètre 23, 51, 100
Performances 114
Perturbation 65-66
pgcd 93
Pente 31, 100, 125
Polygone 144, 169
  à trous 151, 199
  de Voronoi 192
  s-, *f*-, *h*- 6, 18
  plein 199
Précision 47, 49, 73-117
Primitive 99
Priorité 8, 144-162
Probabilité 115
Pseudo-réel 47
Puissance 34, 100

R

Radicaux 187
Rationnels 76-96, 109, 188
Rayon (infini) 192
r-dilaté 168-188
Recalage 71
Réduction (d'arbre) 139-142
Référence 130-134
Réflexe 109
Répétition 130-136
Représentable 79
Représentation
  entiers, rationnels 77
  planaire 121
  topologique 121
Réseau
  de contraintes 183
  de dépendances 137
Réticence 102
Rond-point 185
rsgp 98, 122, 119-188, 216-232
r-3D 173

S

Saule (pleureur) 111
Sémantique 8, 106
  graphe 182
Seuil de fiabilité 103-108, 117
Simplicité
  d'un polygone 171
  simplexe 65
Simulation 2, 167
Sites 191
Sommet 105, 121
  de Voronoi 192
Soustraction
  entiers 82
  rationnels 94
Stabilité numérique 65-67
Subdivision (de tronçons) 186
Super-module 183
Support 28, 100
Système d'équations 23

T

Temps de calcul 17, 40
Topologie (cohérence) 62, 64100
Triangle 197
  de Delaunay 175
Triangulation 175, 196-200
  de Delaunay 13, 69, 93, 101, 175
    contrainte 216-232
    non contrainte 202-205
  par défaut 198
3-D 116
Tronçon 170

U

ulp 54
  voir aussi 105
Underflow 79

V

Valence 64
Valeur par défaut 109
Validité 159-160, 171, 187
Virage (à droite, à gauche) 100, 171, 183
Voisin 192
Voronoi 68, 189-232

ECOLE NATIONALE SUPERIEURE
DES MINES DE SAINT-ETIENNE

N° d'ordre : 46 ID
Année 1990

HIERARCHISATION ET FACETISATION DE LA REPRESENTATION PAR SEGMENTS D'UN GRAPHE PLANAIRE

J.M. Moreau

Spécialité: Informatique.

Mots Clés: Arithmétique mixte, précision arbitraire, diagramme de Voronoi, triangulation de Delaunay avec contrainte, représentation par segments d'un graphe planaire, priorités et hiérarchies, construction incrémentale, 'diviser-pour-résoudre'.

Résumé: L'organisation structurée (graphe avec hiérarchies et priorités 'sémantiques') d'objets du plan implique plusieurs opérations complexes qui doivent être effectuées en toute sécurité de cohérence topologique. La précision inhérente d'une machine étant nécessairement limitée, il faut souvent recourir à une arithmétique exacte coûteuse.
Cette thèse présente, à partir de travaux liés à la réalisation du module de facétisation d'un simulateur de vol industriel, une solution permettant l'utilisation d'une arithmétique mixte, de précision arbitraire et de coût très inférieur, statistiquement, à la solution exacte.
On y trouve aussi l'unification des méthodes de construction d'un diagramme de Voronoi, d'une triangulation de Delaunay pour un nuage de points dans le plan et de la triangulation *contrainte* de Delaunay de la représentation par segments d'un graphe planaire, autour d'une technique incrémentale optimale, fondamentalement plus simple que la méthode 'diviser-pour-résoudre' classique.
La technique incrémentale permet, par ailleurs, de donner un algorithme linéaire et très simple de construction du diagramme de Voronoi et de la triangulation de Delaunay d'un nuage de points situés sur la frontière d'un polygone monotone ou convexe.