

**Ecole Nationale des Sciences Géographiques
D.E.A. des Sciences de l'Information Géographique**



Algorithmes & traitements géométriques

Serge Motet

Conditions d'utilisation de ce document

L'auteur de ce cours "algorithmes et traitements géométriques" est Serge Motet. L'auteur l'a écrit dans le cadre de son activité à l'Institut Géographique National, affecté à l'Ecole Nationale des Sciences Géographiques (ENSG).

Le cours "algorithmes et traitements géométriques" est la propriété de l'IGN. Il est mis à la disposition du public gratuitement sur le site internet de l'ENSG dans le but :

- de faciliter l'accès des scientifiques et des techniciens à des connaissances développées à l'IGN,
- de mettre en valeur les compétences de l'IGN et les formations de l'ENSG.

En conséquence, ce cours est publié aux conditions suivantes :

-Il est interdit de recopier ce document pour le placer sur d'autres sites. Il est interdit de le publier à une autre adresse internet ou une autre *URL*.

-Le cours peut être imprimé ou photocopié, à condition que les entêtes et les pieds de page de chaque page soient reproduits et soit clairement lisibles. S'il est utilisé dans une formation, l'enseignant doit informer explicitement tous les étudiants qu'il utilise un cours de l'ENSG.

-Le cours est mis à la disposition du public gratuitement. S'il est communiqué dans le cadre d'une activité commerciale, le client doit être informé que ce cours est gratuit et qu'il le trouvera sur le site internet de l'ENSG.

Algorithmes et traitements géométriques

Commanditaire : Ecole Nationale des Sciences Géographiques
2, avenue Pasteur
BP 68
94160 SAINT MANDE

Réalisé à l'Ecole Nationale des Sciences Géographiques
2, avenue Pasteur
BP 68
94160 SAINT MANDE

Auteur : Serge MOTET, Ingénieur en Chef Géographe

Titre : Traitements géométriques

Date du cours : décembre 1992
(quelques compléments datent de novembre 1997)

Résumé : Ce cours traite de la conception et la réalisation d'algorithmes sur l'information localisée ou géométrique.

Mots clés ; Thème principal : algorithmique géométrique

Autres mots clés : SIG, géométrie, algorithme, chaîne, arbre, B-tree, pile, file d'attente, dictionnaire, sac, ensemble, mesure d'algorithmes, transformation de coordonnées, mode matriciel, code de Freeman, codage par plage, dallage, quad-tree, clé de Péano, 2DRE, mode vecteur, primitives, filtres de lignes, passage matriciel-vecteur, index géométrique, topologie, cartes combinatoires, théorie des graphes, graphe dual, plus court chemin, allocation, superposition de polygones, triangulation, triangulation de Delaunay, calcul d'un MNT maillé, propriétés différentielles de la surface du terrain, contribution des MNT à la géomorphologie, lignes caractéristiques

Couverture : Statue de Khârezami (Muhammad ibn Mûsâ al-) à Ourgouentch en Ouzbékistan (région dont il est originaire). Les différents traités de ce mathématicien (fin VIIIe-début IXe) lui ont valu d'être à l'origine du mot "algorithme". (photo S.Motet)

Table des matières

Conditions d'utilisation de ce document	2
Structures élémentaires	8
Modèle de types	8
Chaîne.....	8
Arbre.....	9
B tree	10
Constructeurs logiques	10
Pile, file d'attente	10
Dictionnaire, sac, ensemble.....	11
Mesure d'algorithmes	12
Modèle de calcul	12
Notation $O()$	12
Comment trouver l'ordre d'un algorithme ?.....	13
Transformation de coordonnées.....	14
Méthode itérative.....	14
Interpolation sur un maillage.....	14
Stockage en mode matriciel & géométrie discrète	16
Les données matricielles	16
Introduction aux structures de stockage	16
Code de Freeman.....	17
Codage par plage.....	17
Dallage	18
Quad-tree	19
Insertion d'un rectangle	19
Clé de Péano.....	21
2DRE	21
Stockage en mode vecteur et géométrie élémentaire	22
Primitives	22
Calcul d'intersection de deux segments.....	22
Equation analytique.....	23
Méthode vectorielle.....	23
Equation paramétrique	23
Min-Max.....	24
Robustesse & précision des algorithmes.....	24
Intersection d'un ensemble de segments.....	25
Constitution de E	25
Recherche des intersections	26

Point intérieur d'un polygone	27
Filtrages	28
Echantillonnage selon la longueur	28
Echantillonnage selon l'angle	29
Méthode de la corde	29
Douglas-Peuckert.....	29
Passage matriciel -> vecteur.....	30
Transformation en vecteur d'un codage par plage.....	31
Index géométriques	32
Critères d'évaluation.....	33
Index raster.....	33
k-d-tree	33
Quad-tree-point	34
Dallage fixe	34
R-tree.....	35
Topologie	37
Définitions	37
Couches topologiques	38
Réseau	38
Partition	39
Modèles	39
Cartes combinatoires	39
Définitions.....	39
Représentation graphique.....	40
Structure de donnée.....	41
Théorie des graphes.....	41
Définitions.....	42
Graphe dual	42
Structure de données de graphe.....	43
Structure de données de graphe dual.....	43
Lien entre les cartes combinatoires et les graphes	44
Applications topologiques	45
Plus court chemin	45
Structure de données	47
Algorithme.....	47
Optimisation	48
Allocation	49
Superposition de polygones	49
Superposition de deux polygones.....	49
Superposition de partitions.....	49
Traitements altimétriques.....	51
Triangulation	51
Problème de proximité	51
Construction d'une triangulation de Delaunay	52
Propriétés géométriques utilisées	53
Propriétés de L	54
Identification des modifications Construction de L	55
Réalisation des modifications.....	56
Application à la triangulation.....	57

Calcul d'un MNT maillé	59
Exploitation des triangulations.....	59
Exploitation des MNTs	59
Propriétés différentielles de la surface du terrain.....	59
Contribution des MNT à la géomorphologie	60
Propriétés locales.....	60
Lignes caractéristiques	61
Conclusion	63
Annexe : Fenêtrage	64
k-d-tree	64
Structure	64
Fenêtrage	64
Quad-tree-point	65
Structure	65
Fenêtrage	65
Dallage régulier	66
Structure	66
Fenêtrage	66
R-tree	67
Structure	67
Fenêtrage	67

Ce cours porte sur la conception et la réalisation d'algorithmes traitant de l'information localisée ou géométrique. Il est motivé par la constatation que beaucoup de projets de recherche et développement en SIG requièrent la programmation d'algorithmes géométriques. Cela est dû à la relative pauvreté des SIG dans ce domaine et à la nécessité d'adapter des algorithmes à un besoin particulier ou nouveau. Enfin, l'algorithmique géométrique est un domaine de recherche en soi.

L'algorithmique géométrique se distingue de l'algorithmique générale par des structures de données et des méthodes particulières.

Le but du cours est d'acquérir ces méthodes. On cherchera donc à présenter :

- les structures de données utilisées,
- quelques algorithmes simples pour se familiariser avec les structures,
- des algorithmes plus complexes représentatifs de certaines méthodes de programmation.

C'est que certains auteurs appellent des paradigmes.

Enfin, on remarquera que quelques algorithmes sont présentés en détail, jusqu'à la programmation. Il ne faut pas cependant voir ce cours comme un catalogue d'algorithmes prêts à emploi. Les algorithmes présentés ne sont pas les plus utiles mais ils sont parmi les plus démonstratifs. Les morceaux de programmes sont là pour que l'ensemble du raisonnement soit compris, depuis la conception jusqu'à la réalisation pratique.

N.B. Les programmes sont écrits en langage C avec parfois une syntaxe approximative. En particulier, on utilise un type boolean que peut être vu comme un entier prenant les valeurs vrai (1) ou faux (0). D'autre part, le passage des paramètres n'a pas été nécessairement optimisé.

Structures élémentaires

Ce paragraphe est un rappel des structures élémentaires utilisées en algorithmique non géométrique. Les manuels d'informatique de niveau licence ou maîtrise présentent ces structures avec plus de détails.

On cherche à gérer une collection d'entités. Chaque entité est représentée par une "struct" du langage C.

Il existe deux niveaux: Au niveau déclaratif, on définit des modèles de types. Au niveau de l'analyse, on définit plutôt des constructeurs logiques qui sont associées à des fonctions de manipulation et de construction.

Modèle de types

Le premier modèle de stockage d'une collection est le tableau. Il a l'avantage de la simplicité mais le défaut d'avoir un nombre fixe d'entités.

On cherche donc des structures plus dynamiques dont le nombre d'entités peut varier. Les entités sont alors connues par leur pointeur (ou référence). Un pointeur est une variable dont la valeur est un code (adresse) qui identifie une entité et qui permet à l'ordinateur d'atteindre immédiatement aux valeurs stockées dans l'entité.

Chaîne

Une chaîne est une structure linéaire ou horizontale. Chaque entité a une entité suivante et une entité précédente. Donc, connaissant une entité par son pointeur, on ne peut accéder qu'aux pointeurs des entités suivante ou précédente. Il existe des chaînes simples qui ne donnent accès qu'au suivant et des chaînes doubles. Un type d'entité d'une chaîne double se déclare comme suit:

```
struct Entité {
    ... /* Valeurs de l'entité */
    struct Entité *precedent, *suivant;
}
```


Arbre

Un arbre est une structure hiérarchique. Une entité est appelée un nœud. Chaque nœud a un nœud-père (sauf la racine) et un ou plusieurs nœuds-fils. Chaque nœud ne peut être père d'un de ces ancêtres. Enfin, tous les nœuds ont un ancêtre commun. Cet ancêtre est appelé la racine. Les nœuds sans fils sont des feuilles.

Le degré de parenté d'un nœud par rapport à la racine est le niveau ou la profondeur. C'est aussi le nombre de nœuds intermédiaires qu'il faut traverser pour aller de la racine au nœud. Si les feuilles sont au même niveau, l'arbre est dit équilibré ou balancé.

Il faut considérer un nœud comme la racine d'un sous-arbre. Les algorithmes traitant des arbres sont généralement récurifs. Ils s'appliquent à l'arbre total comme au sous-arbre d'un nœud. Ils se propagent en s'appliquant de nouveau aux sous-arbres des nœuds-fils.

Si chaque nœud a un nombre maximal N de fils, son type peut se déclarer comme suit:

```
struct Noeud {
    ... /* Valeurs de l'entité */
    int nombre_de_fils;
    struct Noeud *fils[ N];
};
```

Un arbre binaire est un arbre particulier tel que $N = 2$. Parmi les champs de l'entité, on en distingue un que l'on appelle la clé. Un arbre binaire vérifie la propriété qu'en chaque nœud, les nœuds du sous-arbre du fils à droite -fils[0]- (resp. à gauche -fils[1]-) ont tous une valeur de clé inférieure (resp. supérieure) à celle du nœud. (Cf. figure. 21)

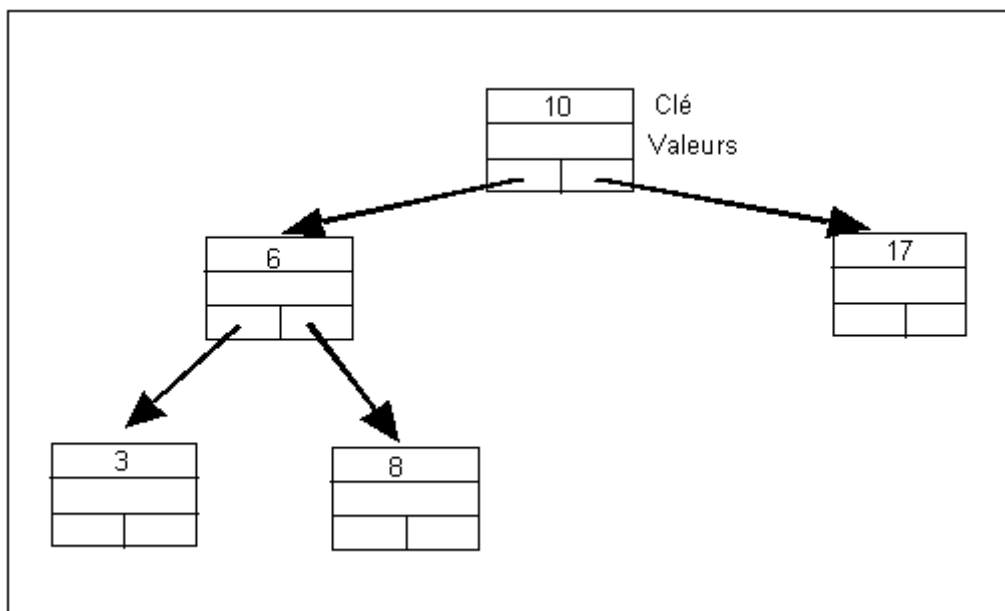


figure 21

B tree

Un B tree ou B Arbre est un arbre avec clé dont un nœud contient une liste de N clés (notée C) et une liste de N+1 pointeurs (notée Fils). Le $i^{\text{ème}}$ pointeur Fils[i] pointe sur un sous-arbre contenant toutes les clés Cl qui vérifient :

$C[i-1] \leq Cl < C[i]$ (Pour la première (resp. la dernière) clé, on a seulement l'inégalité de droite (resp. de gauche)).

Le B Arbre est utilisé dans les index sur disque. Un disque est organisé en pages de taille fixe. On peut choisir N pour qu'un nœud s'insère dans une page. Il existe des algorithmes de recherche qui nécessitent au plus l'accès à $1 + \log_N(M+1)/2$ pages (où M est le nombre de clés).

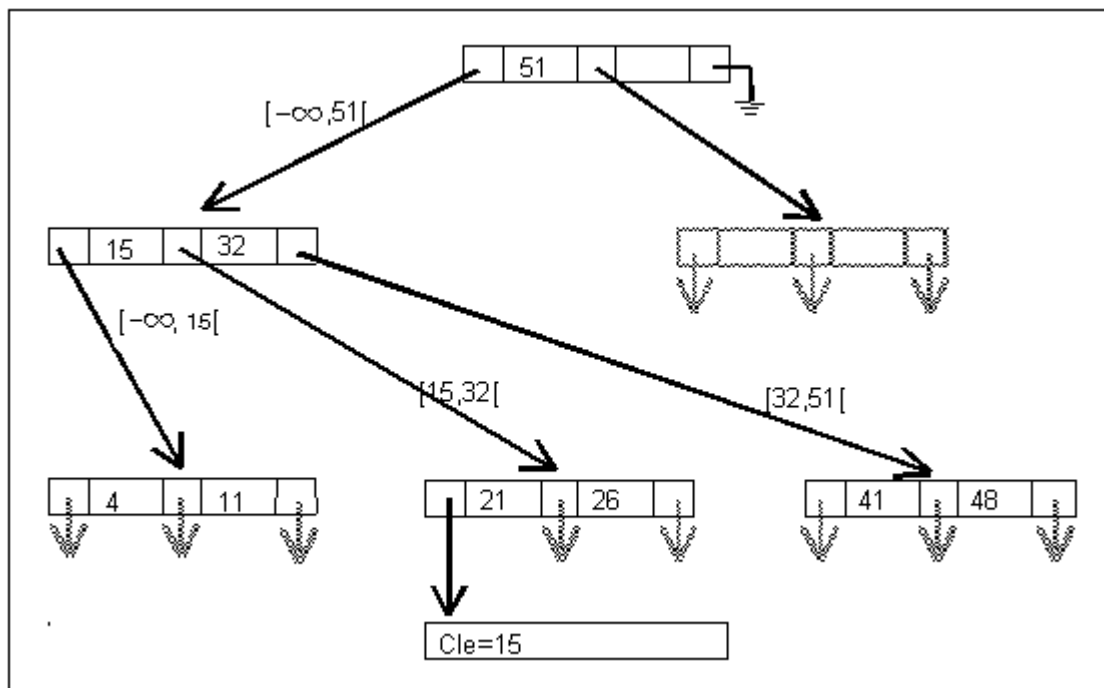


figure 22

Constructeurs logiques

Pile, file d'attente

Une pile est une liste d'entités associée aux fonctions suivantes:

- ajout d'une entité en début de liste,
- recherche de l'adresse du début de liste,
- suppression de l'entité en début de liste.

Une file d'attente est une liste d'entités associée aux fonctions suivantes:

- ajout d'une entité en fin de liste,
- recherche de l'adresse du début de liste,
- suppression de l'entité en début de liste.

Les piles et listes se programment au dessus d'une chaîne simple.

Dictionnaire, sac, ensemble

Ces structures utilisent une clé. Une clé est un champ de l'entité (généralement un entier). Les structures ont en commun trois fonctions :

- ajout d'une entité,
- effacement d'une entité,
- recherche d'une entité dont on connaît la valeur de la clé.

Un sac est une structure qui peut contenir plusieurs entités ayant même valeur de clé. Un ensemble est un sac qui ne permet pas l'ajout d'une entité ayant une valeur de clé déjà existante dans l'ensemble. Enfin, un dictionnaire est un ensemble qui permet de plus de rechercher les entités dans l'ordre des valeurs de clés.

Ses structures peuvent se programmer au dessus d'un arbre binaire ou un B tree.

Mesure d'algorithmes

Ce paragraphe présente les moyens d'évaluation d'un algorithme. Le but de ces mesures n'est pas de calculer précisément la durée d'un algorithme mais d'estimer l'ordre de grandeur (quelques secondes ou plusieurs jours).

Modèle de calcul

Deux mesures de performance sont généralement utilisées: la consommation en temps et l'occupation en mémoire. Dans certains cas, on mesure également le nombre d'appels à une fonction d'entrée/sortie sur disque.

Le but étant d'avoir un ordre de grandeur, on ne distingue pas les durées des opérations élémentaires: variables entières ou réelles, opérations arithmétiques ou fonctions mathématiques (trigonométrique, logarithme...).

La mesure de performance est fonction de paramètres appelés "taille du problème". Par exemple, un paramètre du problème de tri d'un sac est le nombre d'entités.

Généralement, il est très difficile de connaître tous les paramètres. Certains algorithmes de tri sont plus rapides si la liste est déjà partiellement triée. Un paramètre est donc l'ordre initiale de la liste.

La taille du problème comprend les valeurs faciles à connaître, mais qui ne déterminent pas entièrement la performance. Il y a donc plusieurs performances: la meilleure, la moyenne ou la pire.

Notation $O()$

On ne considère que la performance en temps de calcul. L'utilisation de la mémoire se mesure de la même façon.

Définition: Si n est une mesure de la taille d'un problème, un algorithme est dit en $O(f(n))$ si il existe k_1, k_2 tels que. le temps de calcul $T(n)$ vérifie:

$$\forall n \ T(n) < k_1 f(n) + k_2$$

k_1 et k_2 peuvent être très grands.

Comment trouver l'ordre d'un algorithme ?

Il existe plusieurs recettes:

1) S'intéresser surtout à la structure de commande ou squelette: c'est à dire, en première approximation, aux nombres d'itérations dans les boucles.

2) Décomposer en problèmes élémentaires. En particulier certains algorithmes transforment les données avant de les traiter. On peut alors utiliser les règles suivantes:

Si A et B sont des problèmes tels que l'on peut résoudre A en transformant les données de A pour les adapter au format d'entrée de B; Si la transformation se fait en $O(tc(N))$ alors:

- Si on sait que A demande un temps $T(N)$ alors B demandera au moins $T(N) - O(tc(N))$.
- Si B se résout en $T(N)$ alors A demandera au plus $T(N) + O(tc(N))$.

On dit que A est $O(tc(N))$ -transformable en B.

3) Prendre en compte l'appel des fonctions. On peut calculer le temps total à partir du temps de chaque fonction par la règle suivante: Si une fonction s'exécute en $O(t(N))$ et si elle est appelée $f(N)$ fois, alors la contribution de l'appel des fonctions est en $O(f(N).t(N))$.

Les algorithmes sont dits linéaires pour $O(n)$, polynomiaux pour $O(n^c)$ et exponentiels pour $O(c^n)$.

En pratique, les algorithmes exponentiels et polynomiaux avec $c > 2$ ne sont pas utilisables dès que la taille du problème atteint 1000 (ce qui est usuelle pour un algorithme géométrique). En effet, la valeur de k_1 dans la définition de $O()$ est généralement supérieure un millièm de seconde. Donc un algorithme en $O(n^3)$ dure plus de $1000^3 \text{ms} = 11 \text{ jours}$.

Enfin, dans les algorithmes de ce cours, on utilise souvent des ensembles et des dictionnaires. Il faut savoir qu'il existe des algorithmes qui permettent l'exécution des fonctions d'ajout, suppression et recherche en $O(\log(n))$ où n est le nombre d'entités.

Transformation de coordonnées

Une transformation est une application de R^3 dans R^3 . Cette application peut être connue explicitement par des formules de la forme :

$$(u, v, w) \rightarrow (x, y, z): x = f(u, v, w), y = g(u, v, w), z = h(u, v, w)$$

Ce qui ne pose pas de problème. Mais la transformation peut aussi être connue soit par des formules où (u, v, w) et (x, y, z) ne peuvent être séparées soit seulement sur un maillage.

Méthode itérative

Soit une transformation connue par une formule du type :

$$x = f(u, v, w, x, y, z); y = g(u, \dots, x, \dots); \dots$$

Pour (u_0, v_0, w_0) donné, on peut calculer (x, y, z) par itération si on connaît une valeur approchée. Soit (x_1, y_1, z_1) cette valeur approchée.

-On calcule : (x_2, y_2, z_2) par $x_2 = f(u_0, v_0, w_0, x_1, y_1, z_1)$ $y_2 = \dots$

-On répète en calculant $x_{n+1} = f(u_0, v_0, w_0, x_n, y_n, z_n) \dots$

-On arrête lorsque $|x_{n+1} - x_n| < \epsilon, |y_{n+1} - y_n| < \epsilon \dots$ (ϵ = précision souhaitée)

La convergence vers la valeur exacte est, en règle générale, difficile à montrer. Cependant, elle est probable a priori si :

$$\left| \frac{\partial f}{\partial x} \right| < 1, \left| \frac{\partial f}{\partial y} \right| < 1 \dots \left| \frac{\partial g}{\partial x} \right| < 1 \dots$$

et a posteriori pour une valeur (x_1, y_1, z_1) lorsque :

$$|x_{n+1} - x_n| < |x_n - x_{n-1}|, |y_{n+1} - y_n| < |y_n - y_{n-1}| \dots \text{pour tout } n.$$

Interpolation sur un maillage

Les calculs selon les méthodes précédentes (ou d'autres méthodes) peuvent être longs. On peut parfois supposer que la transformation est "lisse", il est alors possible de calculer la transformation sur un maillage et d'interpoler entre les mailles (Cf.figure 31).

Soit, en mode vectoriel, la transformation :

$$X = F(U) \text{ avec } X = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \quad U = \begin{pmatrix} u \\ v \\ w \end{pmatrix} \quad F = \begin{pmatrix} f \\ g \\ h \end{pmatrix}$$

Si F est connue aux nœuds d'un maillage régulier de maille u , v , w . C'est à dire que l'on a calculé les valeurs de :

$$F^*(i, j, k) = F \begin{pmatrix} u_0 + i u \\ v_0 + j v \\ w_0 + k w \end{pmatrix}$$

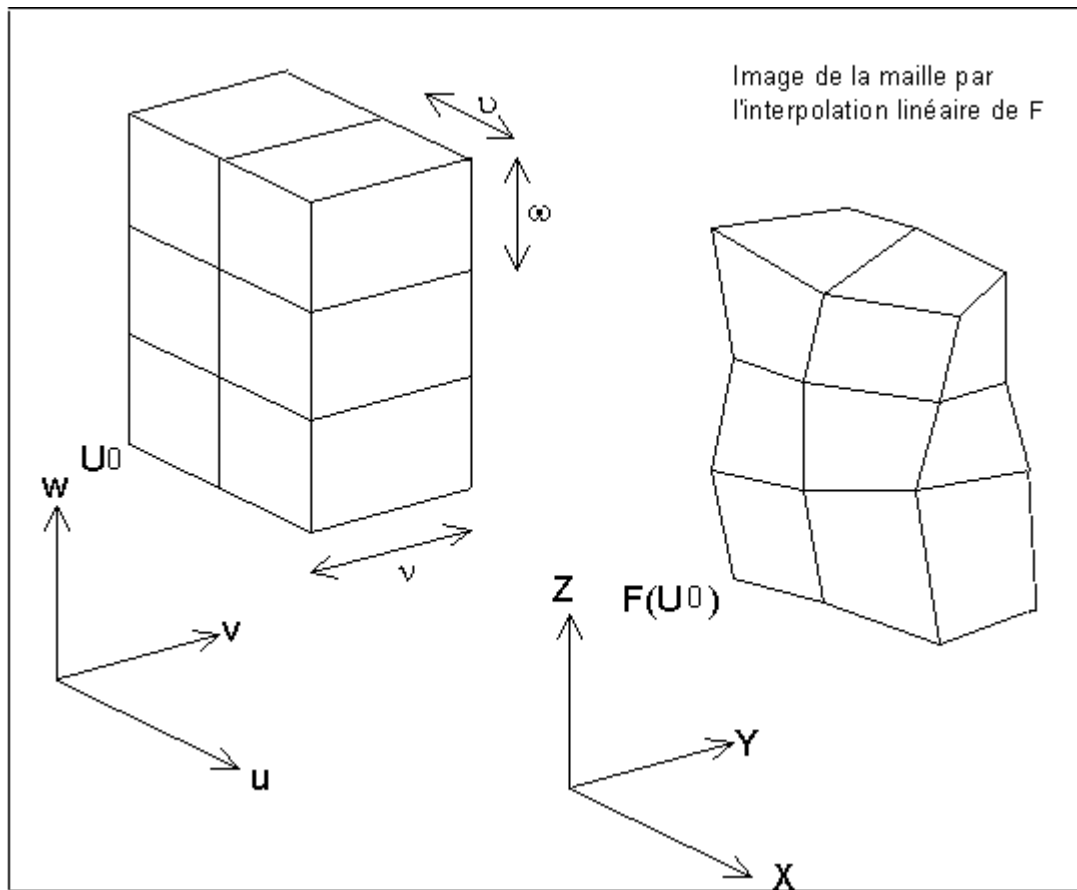


figure 31

On peut calculer par interpolation, la valeur de F en tout point. L'interpolation la plus simple est l'interpolation linéaire. Soit un point U , il existe une maille (i, j, k) telle que :

$$U = \begin{pmatrix} u_0 + (i + u_r) u \\ v_0 + (j + v_r) v \\ w_0 + (k + w_r) w \end{pmatrix} \text{ avec } 0 \leq u_r, v_r, w_r < 1$$

L'interpolation peut s'écrire:

$$X = F(U) = \sum_{a=0, b=0, c=0}^{a=1, b=1, c=1} F^*(i+a, j+b, k+c) q(a, u_r) q(b, v_r) q(c, w_r)$$

Avec la fonction q définie de $\{0, 1\} \times [0, 1] \rightarrow [0, 1]$ par:

$$q(0, r) = 1 - r$$

$$q(1, r) = r$$

Cette interpolation conserve la continuité de F mais pas la continuité des dérivées.

Stockage en mode matriciel & géométrie discrète

Les données matricielles

Les propriétés spatiales des données matricielles concernent en premier lieu la relation entre les valeurs d'un pixel et de ses voisins. Il y a 3 cas :

1) L'égalité de valeurs peut être fortuite. C'est le cas des photos numérisées ou des images issues de satellites.

2) Les pixels voisins ont souvent la même valeur. Par exemple des données issues d'une classification de l'occupation du sol ou une carte statistique numérisée au scanner. On parle de carte de zones ou de domaines.

3) certains pixels peuvent s'organiser en chaînes de pixels voisins de même valeur qui forment des traits (Par exemple les limites d'une carte de zone). Ces données sont appelées carte ou image de contours.

D'autre part le nombre de valeurs possibles varient. Dans le cas de deux valeurs, on parle d'image binaire. Par convention les valeurs sont 0 et 1. Les pixels de valeur 0 forment le fond de l'image.

Introduction aux structures de stockage

La structure de stockage la plus simple est une matrice à deux dimensions. Cependant les matrices utilisées ayant plusieurs milliers de lignes ou colonnes, on a recours à d'autres structures qui améliorent deux critères:

- la place mémoire nécessaire (compression des données),
- la proximité en mémoire des pixels voisins.

Les techniques de compression se fondent sur la probabilité que deux pixels voisins aient la même valeur. Elles sont surtout efficaces pour les cartes de zones ou de contours.

Code de Freeman

Ce codage ne s'applique qu'à une carte de contours. Par rapport à un pixel courant, on se déplace de n pixels dans 4 ou 8 directions. On stocke des couples (nombre de pixels, direction) (figure 41).

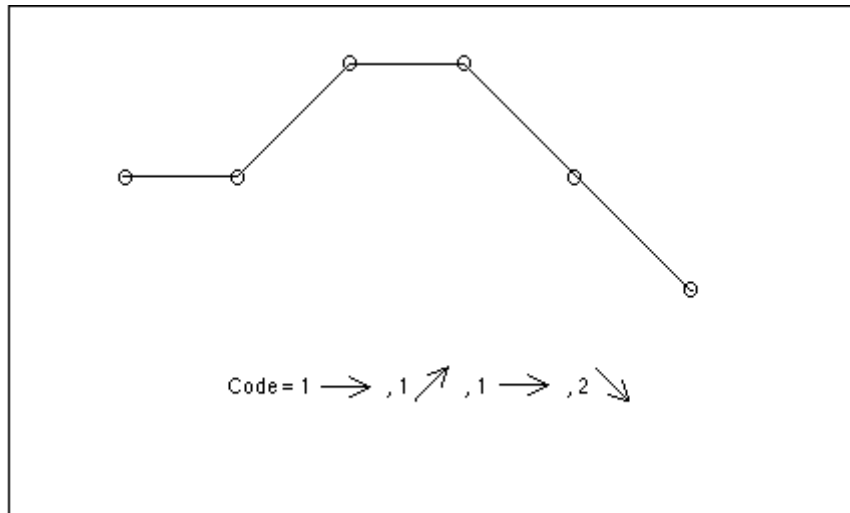


figure 41

Codage par plage

Une plage est une série de pixels consécutifs sur la même ligne horizontale et ayant la même valeur. Chaque ligne est décrite par des couples (nombre de pixels, valeur) (figure 42).

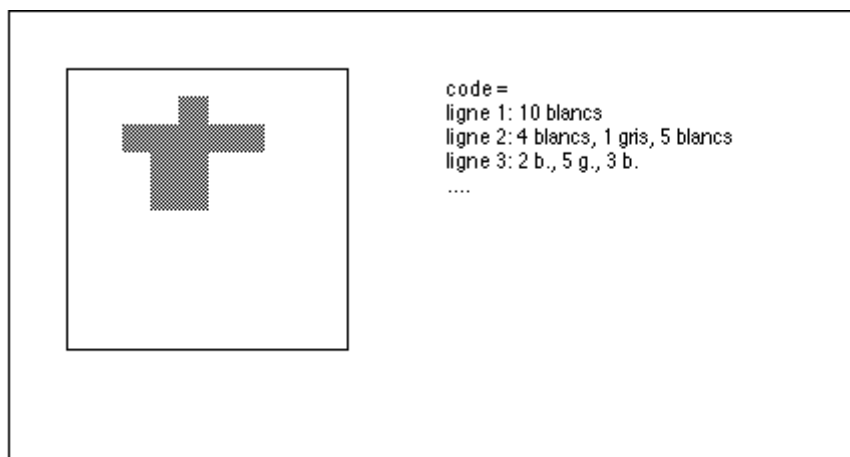


figure 42

Les plages d'une ligne sont groupées en un enregistrement. Par ailleurs on conserve un index sur l'adresse de stockage et l'encombrement de chaque ligne.

Cette structure est appropriée pour un scanner. Elle privilégie le parcours ligne par ligne. Si l'enregistrement d'une ligne est trop grand, il faut combiner le codage par plage avec le dallage.

Dallage

Un dallage est une partition du plan en zones rectangulaires appelées dalles ou tuile. Le découpage peut être régulier ou récursif. A l'intérieur de chaque dalle, la carte peut être codée avec une autre méthode (par plage...).

Le dallage permet de ne mobiliser que les données proches de la zone de travail. Par exemple, soit une image 6000 sur 6000 pixels divisée en 144 dalles de 500 sur 500 pixels. Pour afficher sur une partie de 400 sur 400 pixels, on n'utilisera que 4 dalles.

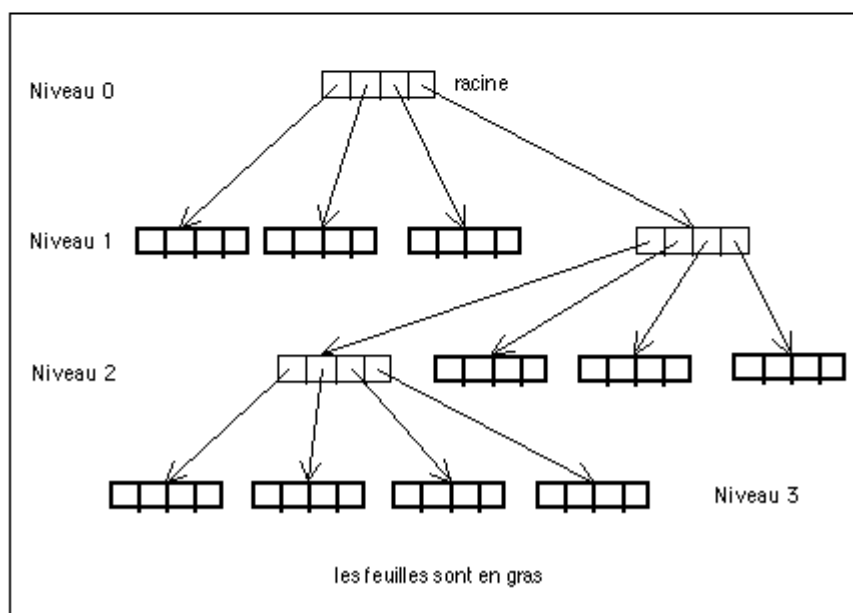
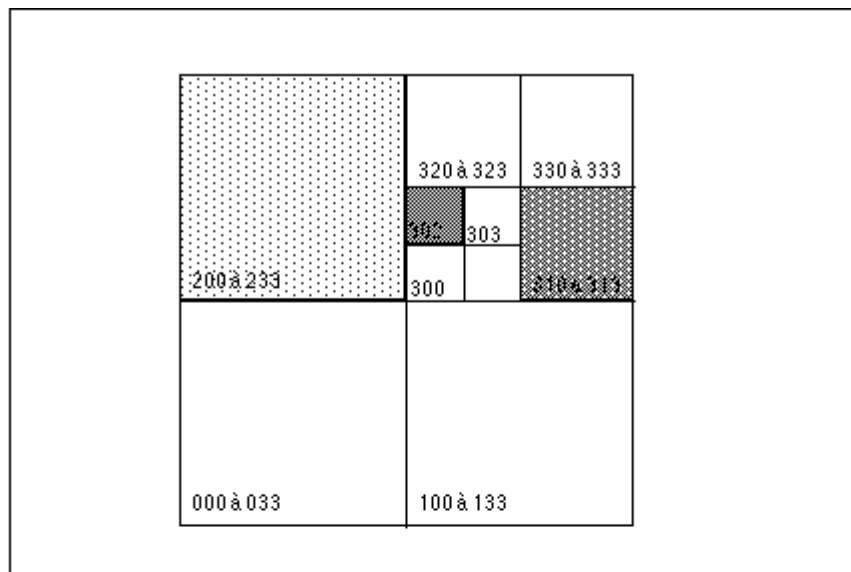


figure 43

Quad-tree

Le quad-tree est un dallage récursif en quatre. La décomposition se poursuit jusqu'à ce qu'une certaine propriété soit vérifiée par la dalle. Cette propriété s'appelle le critère d'arrêt. On dit qu'on a un quad-tree parfait si le critère d'arrêt est que tous les pixels d'une dalle ont la même valeur.

Un quad-tree est donc un arbre dont chaque nœud a au plus 4 fils. La figure 43 représente une image en haut et le quad-tree correspondant en bas.

On peut représenter un nœud d'un quad-tree parfait par la structure :

```
struct Noeud { boolean feuille; /* indique si le nœud est une feuille */
               X... valeur;    /* valeur du pixel */
               struct Noeud *fils[4]; }
```

Insertion d'un rectangle

Comme exemple d'utilisation d'un quad-tree, on propose l'algorithme d'insertion d'un rectangle dans un quad-tree parfait. On part d'un quad-tree et d'un rectangle dont on connaît les coins. On veut réorganiser le quad-tree pour que chaque pixel inclus dans le rectangle prenne une valeur donnée (appelée valeur du rectangle). La figure 44 représente un rectangle que l'on a inséré dans l'image de la figure 43 et le nouveau découpage en dalle.

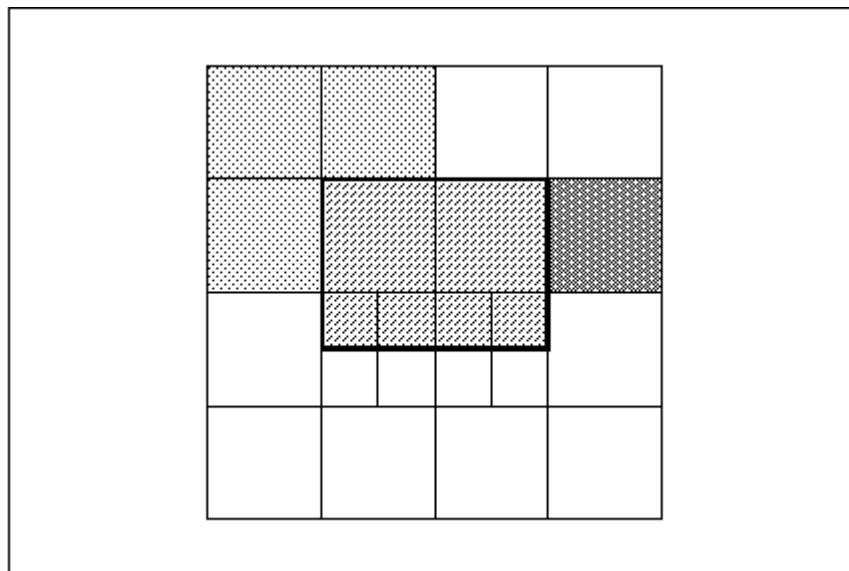


figure 44

On suppose que l'on dispose de la fonction de division d'une feuille en quatre et de la fonction inverse de fusion d'un nœud en une feuille. Lors d'une fusion, on supprime tout le sous-arbre dont la racine est le nœud.

Un algorithme sur un arbre est généralement fondé sur une fonction qui s'applique à un nœud. Cette fonction a deux actions:

- Elle réalise les modifications nécessaires sur le nœud. Ces modifications peuvent être différentes selon que le nœud est ou non une feuille. Les modifications peuvent concerner le nœud lui-même ou le sous-arbre dont le nœud est la racine.
- Elle propage l'algorithme aux fils du nœud en lançant l'appel récursif de la fonction sur ses fils.

L'algorithme général est déclenché par l'appel de la fonction sur la racine.

Pour l'insertion d'un rectangle dans un nœud, trois positions relatives du rectangle et de la dalle du nœud sont possibles :

1- La dalle est incluse dans le rectangle. Alors tous les pixels du nœud prennent la valeur du rectangle, le nœud devient une feuille. Il y a donc fusion du nœud et affectation de la valeur du rectangle au nœud. L'algorithme ne se propage pas car on est sur une feuille.

2- Le rectangle recouvre partiellement la dalle. Le problème est trop complexe pour être traité à ce niveau.

- Si le nœud est une feuille, le critère d'arrêt du quad-tree n'est pas vérifié, Il faut diviser la feuille et propager l'algorithme aux nouvelles feuilles.
- Sinon, il faut propager aux fils.

Par ses divisions, on assure que toute feuille contient des pixels de même valeur. Cependant, il est possible qu'un sous-arbre soit formé de feuilles ayant la même valeur. Il faut donc le vérifier après propagation; Et le fusionner si cela est le cas.

3- La dalle et le rectangle sont disjoints. Alors le nœud et le sous arbre dont il est la racine ne sont pas modifiés par l'insertion. Il n'y a pas de modification et la propagation de l'algorithme aux fils est inutile.

On obtient la fonction suivante :

```
insere_rect( struct Rectangle *r, struct Noeud *n )
{
    int i;
    if (inclusion( n, r ))
    { /* Si n est inclus dans r alors fusion */
        fusion( n );
        n->valeur= r->valeur;
    }
    else if (intersection( n, r ))
    { /* Si n et r ont une intersection non vide,
      ** alors on divise si nécessaire*/
        if (n->feuille) division(n);
        /* Puis on propage */
        for (i= 0; i<4; i++) insere_rect( r, n->fils[i] );
        /* Enfin on vérifie si on peut fusionner */
        for (i= 1; i<4; i++)
            if (!(n->fils[i]->feuille) || (n->fils[i]->valeur !=
                n->fils[0]->valeur)) break;
        if (i == 4) fusion(n);
    }
    /* Si n et r sont disjoints, alors rien... */
}
```

Le programme ci-dessus utilise deux fonctions géométriques : "inclusion" et "intersection". Pour programmer ces fonctions, il faut disposer de la géométrie de la dalle. Cela peut se faire soit en ajoutant à chaque nœud des champs décrivant la position et le coté de sa dalle, soit en recalculant ces dimensions avant chaque appel de "insere_rect".

Clé de Péano

De façon générale, une clé de Péano est une transformation récursive de N^2 dans N .

Soit une image de 2^N sur 2^N pixels. On obtient une dalle d'un seul pixel en découpant N fois en quatre. Pour chaque découpage, on numérote les dalles de 0 à 3. Pour accéder au pixel de coordonnées (x,y) , on découpe l'image en quatre et on se place dans la dalle de numéro d_1 ($0 \leq d_1 < 4$) contenant le point, puis on découpe cette dalle et on se place dans la dalle numéro d_2 , et ainsi de suite jusqu'au niveau N . Le nombre en base 4:

$d_1 d_2 \dots d_N$ est une clé de Péano du point (x,y) .

L'ordre de la numérotation peut varier. L'ordre de la figure 43 (bas-gauche, puis bas-droit, puis haut-gauche, puis haut-droit) donne la clé de Péano dite en Z.

On peut classer les pixels dans l'ordre de la clé de Péano en Z. Alors, les pixels qui appartiennent à une dalle d'un quad-tree sont consécutifs (Cf. figure 43). Cela est vrai pour tous les types de clé de Péano.

L'ordre de Péano est aussi appelé ordre de Morton.

2DRE

Le format 2DRE est une synthèse entre le codage par plage et les clés de Péano. On trie les pixels dans l'ordre de Péano. Une plage est formée des pixels consécutifs dans cet ordre qui ont la même valeur. On code chaque plage sous la forme : clé de Péano (facultatif), nombre de pixels, valeur.

Par exemple, pour la figure 43, on a la séquence:

(000, 32, blanc), (200, 16, gris-clair), (300, 2, blanc), (302, 1, gris-foncé), (303, 1, blanc), (310, 4, gris-foncé), (320, 8, blanc).

N.B. 2DRE signifie *two dimensions run-length encoding* (*run-length encoding* signifie codage par plage).

Stockage en mode vecteur et géométrie élémentaire

Primitives

Un point est défini par une structure de type:

```
struct Point { int x, y };
```

Certains systèmes utilisent des coordonnées en flottant, ce qui rend les programmes plus lents mais moins dépendant des unités utilisées. Le nombre de chiffres significatifs n'est pas nécessairement meilleur en flottant.

Un segment est une paire de points. Une polyligne est une ligne brisée, une polymarque est une liste de points, un polygone est une surface limitée par une ligne brisée (à priori sans clairière). Un polyligne, polymarque ou polygone est conservé dans une structure telle que:

```
struct Polyligne {
    int nb_points;
    struct Point liste[MAX_POINTS];
};
```

Lorsque les polylignes ne se coupent pas et si plusieurs lignes partagent les mêmes extrémités, il est parfois utile de donner ces extrémités une structure propre et de conserver leurs relations avec les lignes. Ces relations peuvent être représentées par un graphe. Les polylignes sont alors des arcs et les extrémités, des nœuds. Ces structures seront développées plus loin dans la partie Topologie.

Calcul d'intersection de deux segments

Problème: Soient les points A, B, C, D; Déterminer si les segments AB et CD se coupent.

Problème équivalent: Les segments AB et CD se coupent si et seulement si A et B sont de part et d'autre de la droite CD et C et D sont de part et d'autre de la droite AB.

Equation analytique

La droite support de AB a pour équation :

$$u_1 x + v_1 y + w_1 = 0$$

$$\text{avec } u_1 = B.y - A.y, v_1 = A.x - B.x, w_1 = A.y B.x - A.x B.y$$

La droite support de CD a pour équation

$$u_2 x + v_2 y + w_2 = 0 \dots$$

A et B sont de part et d'autre de (CD) s'exprime par

$$(u_2 A.x + v_2 A.y + w_2) (u_2 B.x + v_2 B.y + w_2) < 0$$

De même pour C et D

$$(u_1 C.x + v_1 C.y + w_1) (u_1 D.x + v_1 D.y + w_1) < 0$$

Méthode vectorielle

A et B sont de part et d'autre de (CD) si et seulement si les angles CAD et CBD sont de signes opposés. Ce qui équivaut à $\sin(\angle CAD)$ et $\sin(\angle CBD)$ opposés. Or $AC \wedge AD = |AC| |AD| \sin(\angle CAD)$ Or (Oz est le vecteur normal au plan).

D'où:

$$A \text{ et } B \text{ sont de part et d'autre de (CD)} \Leftrightarrow (AC \wedge AD) \cdot (BC \wedge BD) < 0$$

De même pour C et D: $(CA \wedge CB) \cdot (DA \wedge DB) < 0$

$$AC \wedge AD = (C.x - A.x) (D.y - A.y) - (D.x - A.x) (C.y - A.y) \text{ } Oz$$

$$(AC \wedge AD) \cdot (BC \wedge BD) =$$

$$((C.x - A.x) (D.y - A.y) - (D.x - A.x) (C.y - A.y))$$

$$((C.x - B.x) (D.y - B.y) - (D.x - B.x) (C.y - B.y)) < 0$$

Equation paramétrique

Le segment [AB] est l'ensemble des points M tels que:

$$M.x = A.x + (B.x - A.x) t_1 \text{ avec } 0 < t_1 < 1$$

$$M.y = A.y + (B.y - A.y) t_1$$

Il existe une équation identique pour [CD] dont le paramètre est t_2 . Un point commun à [AB] et [CD] vérifie:

$$A.x + (B.x - A.x) t_1 = C.x + (D.x - C.x) t_2$$

$$A.y + (B.y - A.y) t_1 = C.y + (D.y - C.y) t_2$$

On en déduit les conditions sur t_1 et t_2 :

$$0 < \frac{(C.x - A.x) (C.y - D.y) - (C.x - D.x) (C.y - A.y)}{(B.x - A.x) (C.y - D.y) - (C.x - D.x) (B.y - A.y)} < 1$$

$$0 < \frac{(B.x - A.x)(C.y - A.y) - (C.x - A.x)(B.y - A.y)}{(B.x - A.x)(C.y - D.y) - (C.x - D.x)(B.y - A.y)} < 1$$

Min-Max

Cet algorithme calcule d'abord le point d'intersection entre les droites (AB) et (CD), puis vérifie que ce point est intérieur aux rectangles englobant des segments [AB] et [CD].

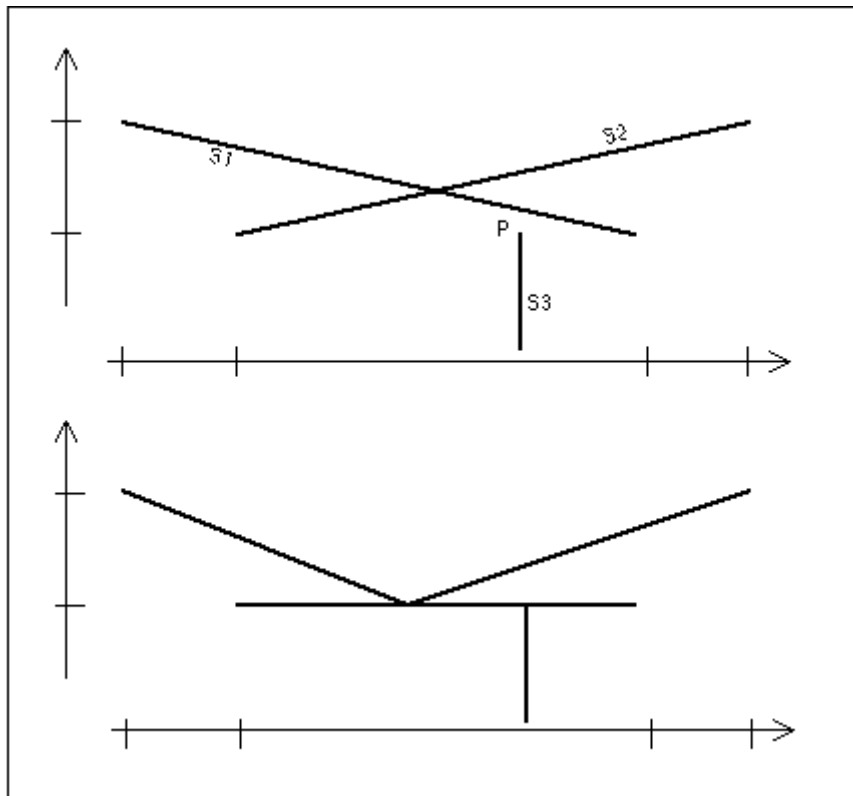


figure 51

Robustesse & précision des algorithmes

Les algorithmes précédents donnent des résultats faux si un point est sur l'autre segment ou si ABCD sont co-linéaires (Cas particulier t_1 ou $t_2 = 1$ ou 0).

Lorsque on calcule l'intersection de 2 segments, le nouveau point est placé au point de coordonnées entières le plus proche. Cela crée 4 segments dont la position peut être différente de celle des segments initiaux. Par exemple, la figure 51 montre que les segments S1 et S2 ne touchent pas le segment S3 avant le calcul de l'intersection alors que ils le coupent au point P après le calcul du point d'intersection.

Intersection d'un ensemble de segments

Soient N segments de droites dans un plan; On cherche à calculer toutes les intersections entre les segments. Pour simplifier la présentation de l'algorithme, on suppose :

- qu'il n'y a pas de segments verticaux,
- que les extrémités de segments ont des X distincts,
- qu'il n'existe pas trois segments se coupant au même point.

Pour une abscisse X_D donnée, les segments qui coupent la verticale $\Delta : x=X_D$ peuvent être ordonnés par Y de l'intersection croissante. Soit L la liste ordonnée des segments.

On utilise une liste E triée en x qui contient les événements suivants:

- 1 Le segment n ($n=1\dots N$) débute à x ,
- 2 Le segment n se termine en x ,
- 3 Il existe une intersection en x entre s_1 et s_2 .

Les événements 1 et 2 sont déterminés et stockés en début d'algorithme (§ 1). L'événement 3 apparaîtra au cours de l'algorithme.

On utilise un algorithme de balayage d'une ligne verticale Δ allant de l'extrémité la plus à gauche à l'extrémité la plus à droite.

On supposera données les fonctions de gestion pour L et E : insertion, effacement, recherche du premier, précédent, suivant. Ces fonctions sont en $O(\log(\text{nombre d'éléments}))$.

La structure E est une file d'attente avec priorité. La priorité est la coordonnée X . L'algorithme traite les événements dans l'ordre des X croissant. Lorsqu'une intersection sera détectée, elle est insérée dans E (événement 3) dans l'ordre des X .

La structure L est un dictionnaire trié selon Y .

Constitution de E

L'algorithme s'effectue en 2 étapes. La première étape est la constitution de E . Elle prend chaque segment, détermine l'extrémité avec le X le plus petit, l'insère comme événement "le segment débute en X " et insère l'autre extrémité avec l'événement "le segment se termine en X ".

Pour chaque segment, on a un test et deux insertions. L'algorithme est en $O(N \log N)$ où N est le nombre de segments.

Recherche des intersections

La seconde étape est la recherche d'intersections. Cette étape est performante si elle réduit le nombre de recherches d'intersections entre segments. En fonction des 3 types d'événement définis pour E, on constate:

- Si un segment débute alors il ne peut avoir d'intersections qu'avec celui du dessus ou celui du dessous.
- Si un segment se termine alors il ne peut avoir d'intersection qu'entre le segment du dessus et du dessous.
- Si il y a intersection, les segments qui se coupent sont permutés, puis se comportent comme des segments qui débutent.

La liste L permet d'obtenir les segments au-dessus (suivant dans la liste) et au-dessous (précédent).

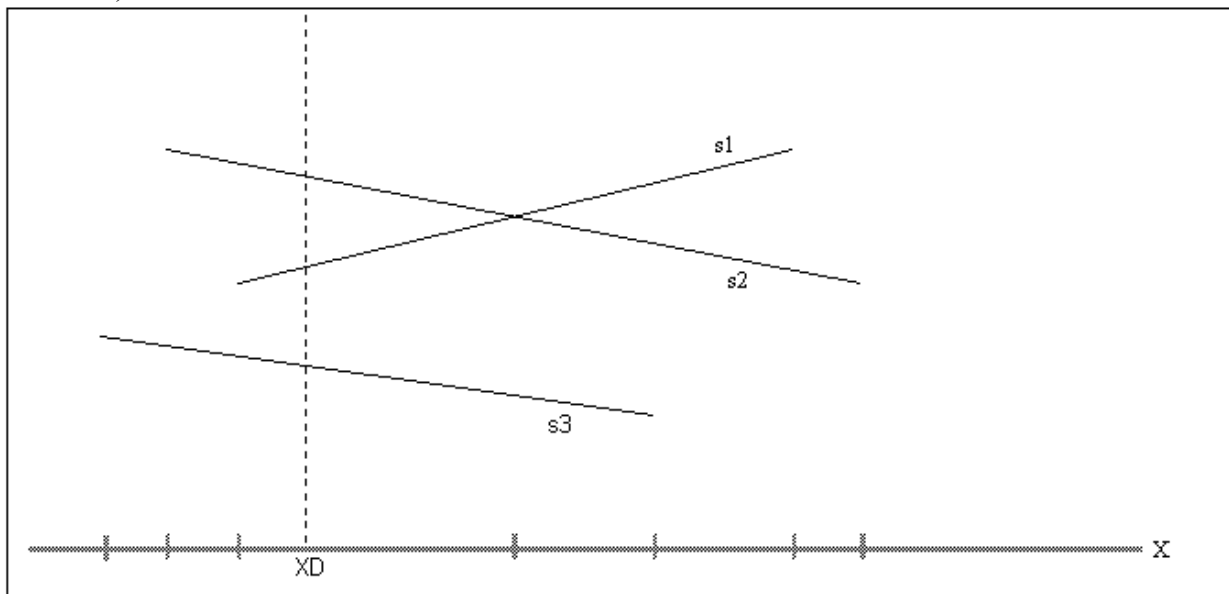


figure 52

Ordre de E = début s3 < début s2 < début s1 < intersection s1 et s2

Ordre de L (pour x = XD) = s3 < s1 < s2

L'algorithme suivant applique les règles ci-dessus.

Pour chaque événement de E pris dans l'ordre:

Si l'événement est "début d'un segment S"

insérer S dans L

Chercher l'intersection entre S et le segment du dessous

Si l'intersection existe :

la calculer et insérer l'événement "intersection dans E"

Chercher l'intersection entre S et le segment du dessus

Si l'intersection existe :

la calculer et insérer l'événement "intersection dans E"

Si l'événement est "fin d'un segment S"

soient S1 le segment au-dessus de S et S2 le segment en dessous

Chercher l'intersection entre S1 et S2

Si l'intersection existe :

la calculer et insérer l'événement "intersection dans E"
enlever S de L

Si l'événement est "intersection de S1 et S2"

(supposons que S1 est en dessous de S2 avant l'intersection)

échanger S1 et S2

soit S3 le segment au-dessus de S1

Si S1 et S3 se coupent

calculer l'intersection et insérer l'événement "intersection dans E"

soit S4 le segment au-dessous de S2

Si S2 et S4 se coupent

calculer l'intersection et insérer l'événement "intersection dans E"

Point intérieur d'un polygone

Problème: Déterminer si un point P est intérieur à un polygone.

Un point P est intérieur à un polygone si le nombre d'intersections d'une demi-droite issue de P est impair. L'algorithme le plus simple est de considérer une demi-droite horizontale issue de P (par exemple x croissant) et de déterminer pour chaque segment de la limite du polygone si la ligne coupe ce segment (c'est à dire si l'ordonnée des extrémités encadre l'ordonnée de P).

Il existe cependant deux cas particuliers: le segment est horizontal d'ordonnée P.y et une extrémité est d'ordonnée P.y.. On remarque qu'un segment horizontal ne peut permettre à la ligne d'entrer ou de sortir du polygone: on peut donc ignorer ce segment. Un point du polygone d'ordonnée P.y ne compte pas si il est un extremum local. Si, pour chaque segment [AB] du polygone, on compare P.y à l'intervalle $[\min(A.y, B.y), \sup(A.y, B.y)[$ on comptera 2 pour les minima et 0 pour les maxima.

D'où l'algorithme:

```
boolean intérieur( struct Point p, struct Polyligne pl )
{
    int c, i, ymin, ymax;
    float xinter;
    extern double floor( double d );

    /* Par convention, pl contient la limite fermée:
    ** liste[0]=liste[nb_point-1] */
    c= 0;
    for (i= 0; i < pl.nb_point - 1; i++)
    {
        if (pl.liste[i].y != pl.liste[i+1].y)
            /* Non horizontal */
            {
                /* Calcul des ordonnées basse et haute du segment */
                if (pl.liste[i].y < pl.liste[i+1].y)
                {
                    ymin= pl.liste[i].y;
                    ymax= pl.liste[i+1].y;
                }
                else
```

```

        {
            ymax= pl.liste[i].y;
            ymin= pl.liste[i+1].y;
        }
/* Comparaison de la droite y = p.y à
** l'intervalle [ymin, ymax[ */
if (ymin <= p.y && p.y < ymax)
{
    xinter=pl.liste[i].x+
        (pl.liste[i+1].x-pl.liste[i].x)
        *(p.y - pl.liste[i].y)
        /(pl.liste[i+1].y-pl.liste[i].y);
/* Appartenance de l'intersection à la demi-droite */
/* floor est la fonction de troncature d'un réel
** en un entier */
if (p.x < floor( xinter ))
    /* Comptage */
    c= c + 1;
}
    }
}
return ((c % 2) == 1);
}

```

La performance de l'algorithme est en $O(\text{nb_points})$.

Filtrages

Problème: Réduire le nombre de points d'un polyligne tout en conservant la même "forme".

Soient P le polyligne à réduire et Q le polyligne résultant du filtrage. Les points formant Q sont des points répartis sur P.

Echantillonnage selon la longueur

Soit S un seuil; On veut que chaque point de Q soit à une distance supérieure à S du précédent et du suivant. On utilise l'algorithme ci-dessous (la fonction distance fournit la distance euclidienne entre deux points):

```

int i, j;
Q.liste[0]= P.liste[0];
j= 0;
for (i= 1; i < P.nb_point-1; i++)
    if (distance( Q.liste[j], P.liste[i] ) > S)
        {
            Q.liste[j+1]= P.liste[i];
            j= j+1;
        }
Q.liste[j]= P.liste[ P.nb_point-1 ];
Q.nb_point= j;

```

Le défaut de cette méthode est qu'elle supprime les courbes représentées par une suite de points proches.

Echantillonnage selon l'angle

La contrainte de distance est remplacée par l'angle ($Q.\text{liste}[j]$, $P.\text{liste}[i]$, $p.\text{liste}[i+1]$) égale à $\pi \pm S$.

Méthode de la corde

Cette méthode consiste à choisir les points de Q de sorte que les points compris entre $Q.\text{liste}[i]$ (noté par la suite Q_i) et $Q.\text{liste}[i+1]$ (noté Q_{i+1}) soient à une distance inférieure à S du segment $[Q_i, Q_{i+1}]$.

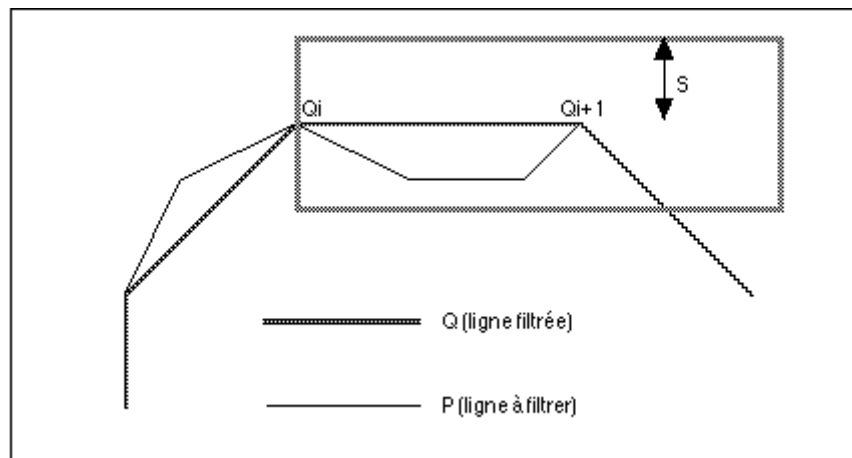


figure 53

La méthode de la corde est itérative. Initialement, Q_1 est $P.\text{liste}[1]$. Supposons Q_i connu, on étudie chaque point de P dans l'ordre à partir de Q_i . Pour chaque point P_j , on calcule les distances entre le segment $[Q_i, P_j]$ et les points de P compris entre Q_i et P_j . Si une des distances dépasse S, P_{j-1} devient Q_{i+1} . (Cf. figure 53)

Douglas-Peucker

Cette méthode utilise le même critère que celle de la corde: la distance entre un segment et les points intermédiaires. La différence est qu'elle est récursive.

Initialement (1er niveau de la figure 54), on considère le segment formé des points extrémités de P (noté P_a et P_b) (P_1 et P_9 dans la figure 54). On calcule les distances entre le segment $[P_a, P_b]$ et les points de P compris entre P_a et P_b . On trouve le point le plus éloigné (noté P_i) (P_6 dans la figure 54).

Si sa distance est inférieure à un seuil S, les points de P entre P_a et P_b ne sont pas dans la ligne filtrée (P_a et P_b sont dans la ligne filtrée).

Sinon on applique récursivement 2 fois la méthode :

- au segment $[Pa, Pi]$ (Pb est remplacé par Pi)
 - au segment $[Pi, Pb]$ (Pa est remplacé par Pi)
- (2ème niveau de la figure 54).

La méthode n'est pas appliquée de nouveau si les points Pa et Pb sont successifs (s'il n'y a pas de point intermédiaire entre Pa et Pb).

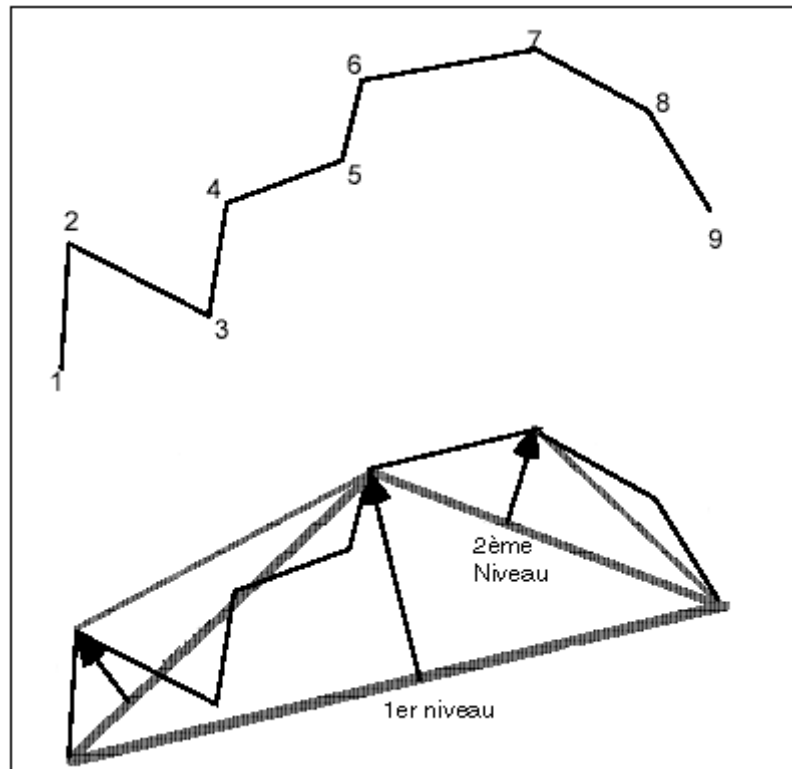


figure 53

Passage matriciel -> vecteur

Ce chapitre présente des algorithmes de transformation de données structurées en mode matriciel dans une structure en mode vecteur.

Il existe des méthodes de traitement d'image qui produisent des cartes de contours. Lorsque les contours ont plus d'un pixel d'épaisseur, il faut rechercher le squelette. Il existe plusieurs méthodes qui donnent des résultats différents.

L'algorithme suivant part d'une carte de zone et en cherche les contours.

Transformation en vecteur d'un codage par plage

Rappel : une ligne (resp. une colonne) est l'ensemble des pixels ayant la même ordonnée (resp. abscisse). Une plage est une partie de ligne formé de pixels consécutifs de même valeur.

L'algorithme utilise la cohérence entre lignes. On part de la ligne du haut. On descend en comparant chaque ligne et la ligne au dessus. On met à jour les contours déjà obtenus sur toutes les lignes au dessus. Puis on passe à la ligne du dessous.

La transformation de zones produit des arcs et des nœuds (c'est à dire le point d'intersection d'au moins 3 arcs).

La structure en sortie contient:

- des zones qui sont identifiées par un numéro arbitraire et unique,
- des nœuds,
- des arcs-limite qui peuvent être codées en utilisant la méthode de Freeman. Chaque limite a deux attributs: le numéro de zone à droite et le numéro de zone à gauche. La direction de référence est donnée par l'ordre des points dans le polygone.

Au cours du traitement, on a donc une structure de sortie partiellement remplie et un état de la frontière de la dernière ligne. Cet état est une liste alternée de numéros de faces en attente et de descriptions d'extrémités.

Une extrémité est décrite par:

- la référence à son arc
- sa coordonnée X.
- un indicateur précisant si il s'agit de l'extrémité de début ou de fin. Pour simplifier d'éventuelles fusion d'arc, il est astucieux d'orienter les arcs de façon à avoir un numéro de face à droite supérieur au numéro à gauche. On peut donc prolonger l'arc au début ou à la fin.

La ligne suivante est décrite par une liste de changement de plages. Chaque changement est décrit par sa coordonnée X.

On part de la ligne du haut.

La structure de sortie va être mise à jour pour incorporer la ligne suivante. La mise à jour peut être déclenchée par 3 types d'événement:

- 1: la plage précédente est interrompue,
- 2: la plage suivante est interrompue,
- 3: les plages précédente et suivante sont interrompues.

L'algorithme parcourt, par X croissant, simultanément les deux listes (état de la frontière et changement de plages) et s'arrête sur chaque événement défini ci-dessus.

L'état précédent un événement dépend de deux indicateurs: les plages de droite ou les pages de gauche sont identiques ou différentes.

On a donc 12 configurations. L'algorithme doit:

- reconnaître chaque événement,
- déterminer la configuration,

- ajouter à la structure de sortie les arcs, nœuds ou faces nécessaires (cette mise à jour n'est pas détaillée ici, car elle suppose une description détaillée des structures de données).

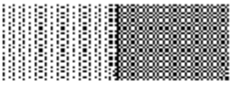


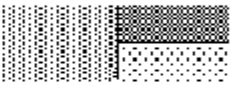


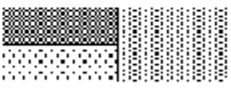
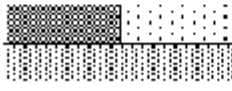
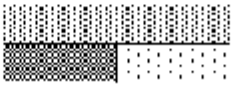
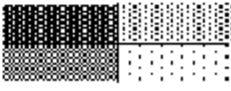
		Interruption en haut	Interruption en bas	Interruption en haut et en bas
Evenement		1	2	3
Trame de gauche	Trame de droite			
	Trame de gauche			
	Trame de droite			
	Trame de gauche			

figure 55

En fonction de l'état précédent, de l'état suivant et de l'événement, on peut construire un tableau des mises à jour de la structure de sortie: des créations de ligne ou de nœud et des fusions de lignes (figure 55).

La méthode qui consiste à imaginer une ligne horizontale qui se déplace verticalement est utilisée dans de nombreux algorithmes. Pour chaque déplacement de la ligne, on étudie ce qui a changé. Ces algorithmes portent le nom d'algorithme de balayage de ligne (line scan) ou de cohérence de ligne.

Index géométriques

Un index géométrique est un ensemble d'informations organisé de façon à accélérer les opérations d'accès à des primitives sélectionnées sur des critères de position. Les informations contenues dans les index sont donc dérivées de la géométrie des primitives.

Certains index déterminent la zone de stockage de chaque primitive : on dit qu'ils réalisent le placement des données. Le principe général du placement géométrique est de regrouper sur une page de disque, les primitives qui représentent des détails proches sur le terrain.

Certains index modifient les primitives; En particulier, ils peuvent les couper en plusieurs morceaux.

Un index peut être construit sur un lot de primitives existant et immuable. On parle d'une construction statique de l'index. A l'opposé, il peut exister des fonctions d'ajout ou de suppression d'une primitive, entraînant une modification de l'index. On parle de construction dynamique.

Pour l'utiliser un index, il faut lui associer des opérateurs. Les plus courants sont :

- Réaliser les modifications de l'index résultant de l'ajout et l'effacement d'une primitive.

- Etant donné un point, trouver les primitives qui contiennent ce point. Cette opération s'appelle le pointé.

- Etant donné un rectangle, trouver les primitives qui ont une intersection avec ce rectangle. C'est le fenêtrage. Il existe des variantes : trouver seulement les primitives incluses ou remplacer un rectangle par un polygone voire une ligne.

Critères d'évaluation

Les critères d'évaluation d'un index sont très divers. Le premier est le temps pris par les opérations présentées ci-dessus. Le second est la place requise par cet index. Les performances peuvent varier très sensiblement entre la meilleure configuration et la pire. En particulier, la construction statique d'un index peut donner une configuration optimale qui se dégradera lors d'ajout ou effacement de primitives.

Enfin, chaque index a des limitations de nature géométrique : impossibilité de conserver des primitives superposées, nombre maximal de primitives se coupant au même point...

Index raster

Ces index sont cités pour mémoire car il s'agit de structures de données vues dans d'autres chapitres. Par exemple le dallage et le quad-tree sont en fait des index.

k-d-tree

Le k-d-tree est un index sur des points. C'est un arbre binaire. On pourrait considérer qu'il tire son origine d'un arbre binaire où les points seraient insérés dans l'ordre de la coordonnée x. Cet arbre privilégie un axe de coordonnées. Pour palier à cela, on alterne les axes de coordonnées.

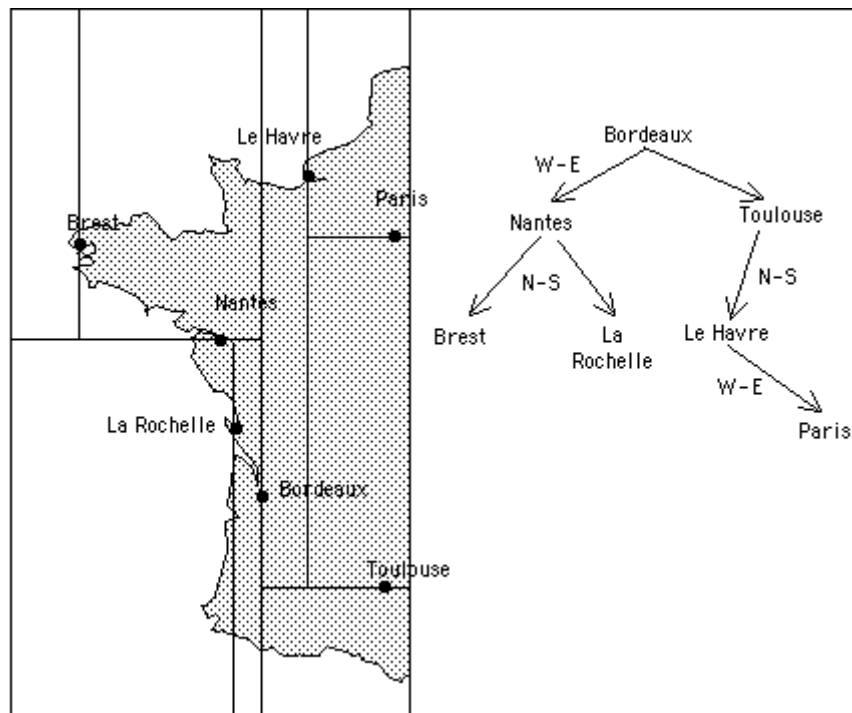


figure 56

Soit un point P en K dimensions. Soit N le niveau du nœud qui représente ce point P . Les nœuds du sous arbre $\text{fils}[0]$ (resp. $\text{fils}[1]$) représentent des points dont la $(N \bmod K)^{\text{ième}}$ coordonnée est inférieure (resp. supérieure) à celle de P .

La figure 56 présente un exemple $K=2$. Le découpage géométrique est à gauche et l'arbre correspondant est à droite.

Les performances dépendent des déséquilibres de l'arbre. Si les feuilles sont bien réparties, on obtient un accès en temps logarithmique, mais dans le pire des cas, il peut être linéaire.

En principe, on ne peut pas conserver des points superposés.

Quad-tree-point

Le quad-tree-point un index sur des points. C'est une variante du quad-tree où le découpage s'arrête lorsque la dalle ne contient que zéro ou un point.

Dallage fixe

Comme pour le mode maillé, le dallage fixe est un découpage du plan en rectangles de dimensions fixes. Dans une page, on place les primitives découpées aux dimensions de la dalle. Si la place occupée par les primitives est supérieure à la page, on prévoit des pages de débordement.

Cet index est efficace pour les primitives uniformément réparties. Le choix des dimensions de la dalle est déterminant pour les performances.

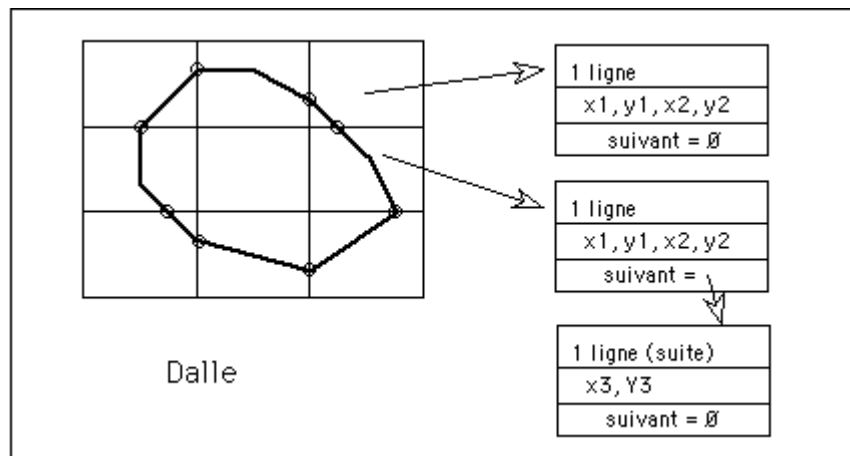


figure 57

R-tree

Le R-tree est un index sur des rectangles. Il ne gère pas directement les primitives mais leur rectangle englobant (plus petit rectangle contenant la primitive). L'usage du rectangle englobant simplifie les algorithmes mais peut dégrader la localisation : par exemple, une ligne diagonale très longue aura un rectangle de grande superficie.

Le R-tree est récursif. Initialement, les primitives sont conservées dans une page. Lorsque cette page est pleine, on divise en deux en répartissant les rectangles les plus à droite ou les plus bas (resp. les plus à gauche ou les plus haut) dans la première dalle (resp. la deuxième). On calcule, pour chaque page, les rectangles englobant tous les rectangles contenu dans cette page. On conserve ces nouveaux rectangles dans une nouvelle page.

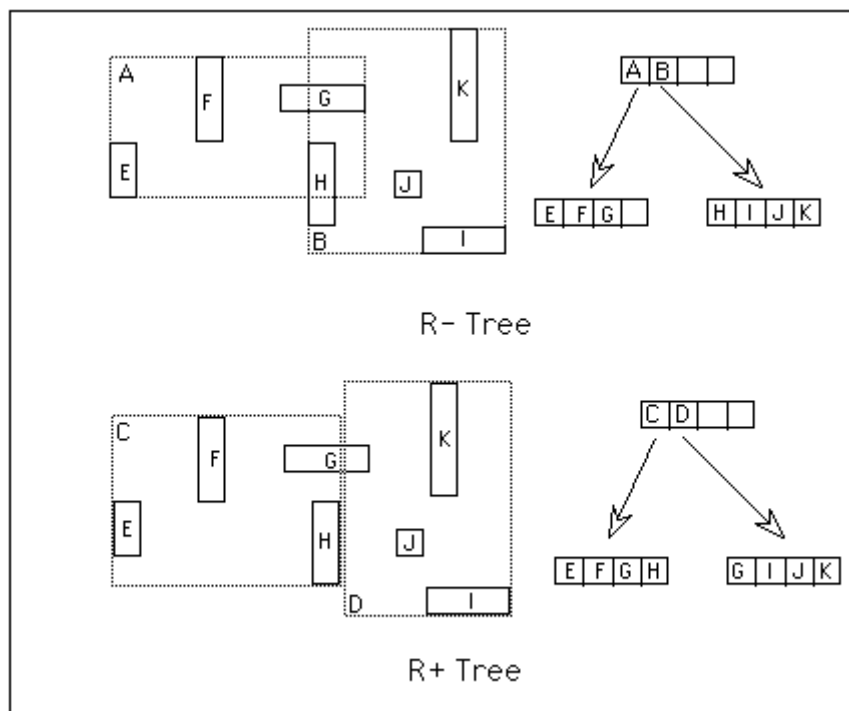


figure 58

Le R-tree est un index universel. Il a des performances moyennes qui peuvent se dégrader au fur et à mesure des mises à jours.

On voit sur la figure 58 que les rectangles de la racine et des nœuds intermédiaires se recouvrent (rectangles A et B de la figure). On peut aussi utiliser des rectangles disjoints mais un rectangle de feuille peut se trouver dans plusieurs nœuds. C'est un R+Tree.

La qualité d'un R Tree ou R+Tree dépend de la façon de répartir les rectangles entre 2 pages lorsqu'une page est pleine. Pour un R+tree, on peut procéder comme suit :

Soit un rectangle père P intersectant des rectangles fils F_1, F_2, \dots, F_N . N étant supérieur à la capacité d'une page, on veut diviser P.

Dans un premier temps, on travaille selon l'axe Ox. On trie sur Ox, et on forme la liste des abscisses extrémales des rectangles : $F_i.x_{\min}, F_i.x_{\max}$. Pour chaque abscisse x_j de la liste, on détermine N_g (resp. N_d) le nombre de rectangles F_i tels que $F_i.x_{\min} \leq x_j$ (resp. $F_i.x_{\max} > x_j$).

On veut que :

- $N_g + N_d$ minimal (on notera que $N_g + N_d \geq N$),
- N_g et N_d inférieurs à la capacité d'une page,
- éventuellement N_g et N_d supérieurs à un taux de remplissage minimal.

On trouve la valeur de x_j optimale pour la première condition et vérifiant les 2 autres.

On travaille ensuite selon l'axe Oy. on trouve y_k . On choisi parmi x_j ou y_k celui qui à la plus petite valeur de $N_g + N_d$. Le rectangle P est découpé en deux rectangles adjacents limités par une verticale (resp. une horizontale) d'abscisse x_j (resp. d'ordonnée y_k) si on choisi x_j (resp. y_k).

Topologie

Définitions

Une information topologique est une information sur la superposition ou la mitoyenneté de deux détails topographiques. Entre surfaces, les informations topologiques sont l'intersection ou l'inclusion. Entre ligne et surface, ce sont l'inclusion et le fait d'être adjacents (figure 70).

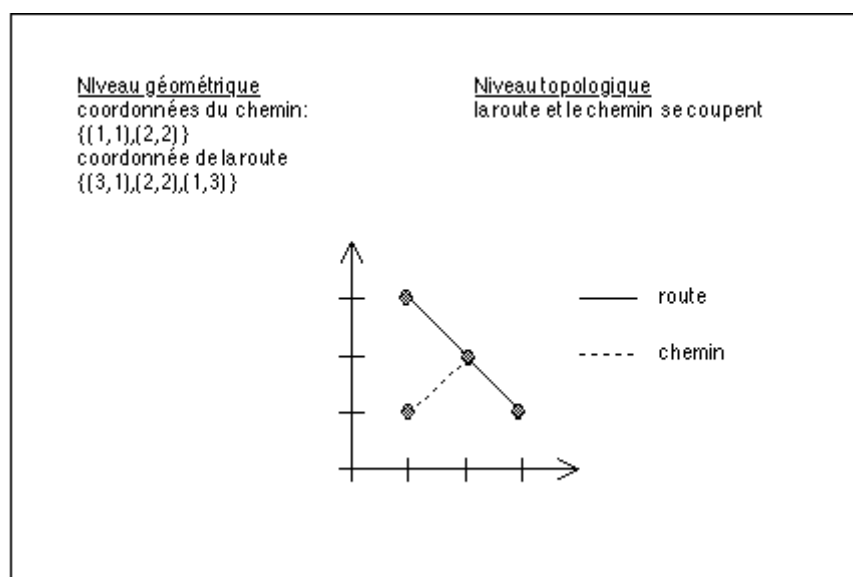


figure 70

En mode vecteur, la topologie porte sur les primitives mises en commun (points extrémités de plusieurs lignes, lignes partagées, etc...).

La topologie peut toujours être calculée à partir de la géométrie. Cependant il existe des structures de données qui permettent de la conserver. Il est intéressant dans certains cas de calculer toutes les propriétés à la saisie pour qu'elles soient disponibles pour les interrogations.

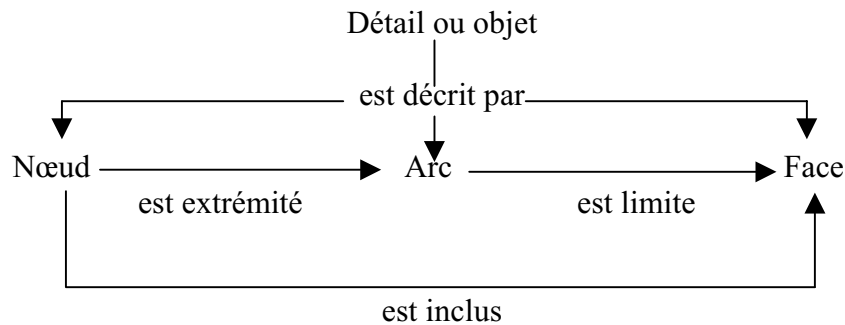
Les primitives dont les propriétés topologiques sont constantes sont:

- les points isolés et les intersections entre lignes,
- les lignes isolées et les lignes entre deux intersections et qui appartiennent aux mêmes détails topographiques,

-les surfaces ne contenant que des points ou des lignes isolées et qui appartiennent aux mêmes détails topographiques.

Les primitives définies ci-dessus sont appelées: nœud pour un point, arc pour une ligne et face pour une surface.

Les relations topologiques sont décrites par le diagramme ci-après:



Couches topologiques

La création de toutes les informations topologiques induit une structure de données de taille importante et impose en particulier le calcul de toutes les intersections entre lignes. D'autre part, certaines informations sont peu demandées et peuvent être calculées au moment de la requête.

C'est pourquoi beaucoup de logiciels se contentent de maintenir la topologie à l'intérieur d'un thème, la base de données contenant plusieurs thèmes.

On appelle couche ou niveau (ne pas confondre avec les niveaux géométrique, topologique, sémantique) un ensemble d'entités où la topologie est maintenue.

D'autre part, la topologie est utilisée pour maintenir des contraintes sur les détails topographiques. Par exemple, il est courant de créer un thème des routes de façon à rendre identique le graphe topologique et le réseau routier. Ces contraintes imposent que les liens "est extrémité", "est limite" ou "est inclus" soient cohérents avec la géométrie.

Le plus souvent, les liens sont maintenus partiellement. Deux structures topologiques partielles sont usitées : le réseau et la partition.

Réseau

Si on ne considère que les nœuds et les arcs, les instances de la relation "est extrémité" forment un réseau. Si on se restreint à cette relation, la théorie de graphe est suffisante. Le graphe est à priori non orienté: cependant le sens des arcs est utile dans certaines applications (écoulement, sens unique...).

Partition

De même, certains thèmes surfaciques forment une partition: en un point donné, on ne trouve qu'une entité. Le découpage communal ou l'occupation du sol sont des exemples de partition. La contrainte de partition est équivalente à une contrainte de type 1:n sur le lien "est décrit" de détail vers face (c'est à dire une face "décrit" au plus un détail).

Modèles

Dans les chapitres suivants, on présentera deux modèles pour représenter la topologie: le premier est fondé sur la théorie des cartes combinatoires et le second sur la théorie des graphes. Pour chaque modèle, on proposera une structure de données.

Ensuite, les principales applications faisant intervenir la topologie seront présentées avec les algorithmes correspondants.

Cartes combinatoires

Les cartes combinatoires ou topologiques sont des objets mathématiques. Leur application à l'information géographique est récente et encore restreinte.

Définitions

On appelle orbite ou cycle d'une bijection p , un sous ensemble minimal de B stable par p (c'est à dire si b appartient à Orb alors $p(b)$ appartient à Orb).

Une carte combinatoire ou topologique comprend:

- un ensemble fini B de cardinal pair dont les éléments sont appelés des brins,
- une involution sans point fixe sur B nommée α : $\alpha(\alpha(b)) = b$ et $\alpha(b) \neq b$,
- une permutation (ou bijection) sur B appelée σ .

Les orbites de α sont appelés arc, les orbites de σ sont les nœuds.

α étant une involution, les cycles ont 2 éléments $\{b, \alpha(b)\}$. Par convention, on note $B = \{-m, -m+1, \dots, -1, 1, \dots, m-1, m\}$ et $\alpha(b) = -b$.

Les faces sont définies comme les orbites de $\Phi = \alpha \circ \sigma^{-1}$.

Le dual d'une carte (B, σ, α) est la carte (B, Φ^{-1}, α) . On montre aisément que le dual du dual est la carte initiale. On a les équivalences suivantes:

Carte	Carte duale
arc	arc
nœud	face
face	nœud

Le genre d'une carte est égal à $1/2(m + 2 - z(\sigma) - z(\Phi))$ où z est le nombre de cycles. Une carte est dite plane si son genre est nul.

Représentation graphique

La représentation graphique conventionnelle est celle d'un graphe.

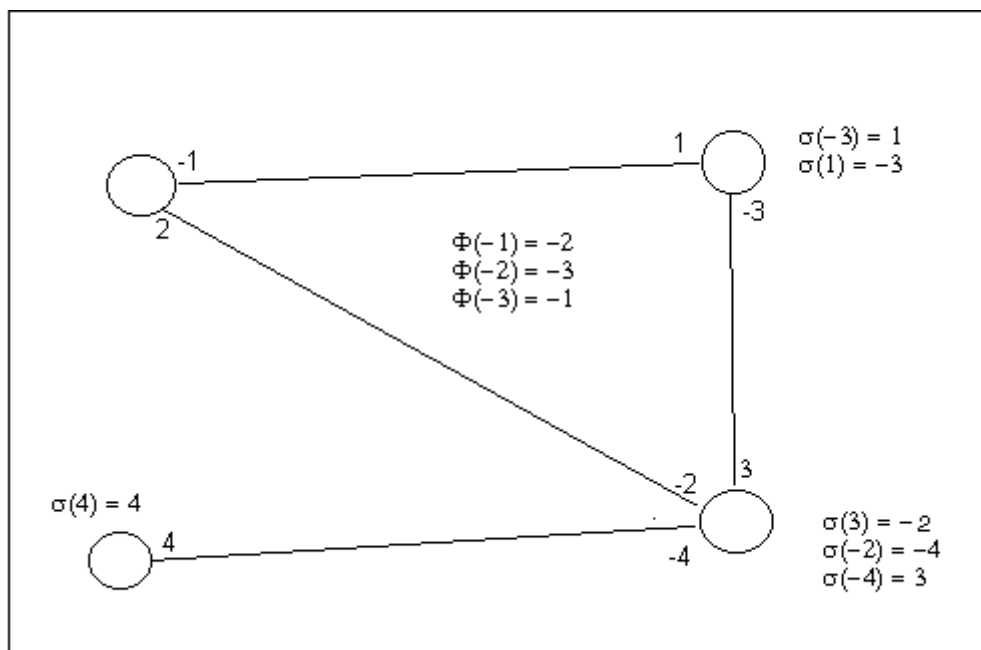


figure 71

Un brin est une extrémité d'arc, c'est à dire un couple (arc, nœud).

Un arc est le couple d'extrémités (b, -b). Il représente donc les liens issus de α .

Un nœud est un cycle de σ . Soit une extrémité b , $\sigma(b)$ est l'extrémité sur le même nœud que l'on rencontre en premier quand on tourne dans le sens trigonométrique.

$\Phi(b)$ s'obtient en tournant dans le sens des aiguilles: sur le premier arc rencontré, on va sur l'autre extrémité.

Structure de donnée

Une structure possible est le stockage de α et σ par une double liste.

```
struct Brin {
    struct Brin *alphaSuiv;    /*  $\alpha(b)=\alpha^{-1}(b)$  */
    struct Brin *sigmaSuiv;    /*  $\sigma(b)$  */
    struct Brin *sigmaPrec;    /*  $\sigma^{-1}(b)$  (optionnel) */
};
```

On peut regrouper les deux brins opposés. Pour plus de clarté, -b peut être qualifié de début et b de fin de l'arc.

```
struct Brin {
    struct arc *Arc;
    enum { début, fin } signe; /* Signe du brin */
}
```

```
struct Arc {
    struct Brin suivFin;      /*  $\sigma(b)$  */
    struct Brin precFin;      /*  $\sigma^{-1}(b)$  */
    struct Brin suivDebut;    /*  $\sigma(-b)$  */
    struct Brin precDebut;    /*  $\sigma^{-1}(-b)$  */
};
```

Il est facile d'affecter des attributs aux arcs (c'est à dire aux doubles brins). Pour les faces et les nœuds, il faut choisir un brin particulier comme point de départ du cycle. De plus les faces et les nœuds doivent être référencés dans les arcs.

```
struct Face /* ou Noeud */ {
    ... valeur;              /* attributs éventuels */
    struct Brin cycle;
};

struct Arc {
    struct Noeud *fin;        /* Cycle  $\alpha(b)$  */
    struct Noeud *début;      /* Cycle  $\alpha^{-1}(b)=-b$  */
    struct Face *gauche;      /* Cycle  $\sigma(b)$  */
    struct Face *droit;       /* Cycle  $\sigma(-b)$  */
    ...                      /* définition précédente */
}
```

Les structures de carte sont en pratique plus complexes à cause de la non-connexité du graphe (présence de clairières) et par les liens entre la topologie et la géométrie.

Théorie des graphes

On s'intéresse aux arcs et aux nœuds d'un thème possédant des propriétés de réseaux (route, voie ferrée, hydrographie). La théorie des graphes permet alors des analyses dont quelques exemples sont donnés dans le chapitre suivant.

Définitions

Un graphe est un couple $G=(X, U)$ constitué:

- par un ensemble X de nœuds,
- par une famille U d'éléments du produit cartésien $X \times X$. Un élément de U est un arc.

L'ordre d'un graphe est le nombre de nœuds.

y est successeur de x si (x,y) appartient à U . L'ensemble des successeurs de x est noté $\Gamma(x)$ ou $\Gamma^+(x)$. L'ensemble des prédécesseurs est $\Gamma^{-1}(x)$ ou $\Gamma^-(x)$.

Le demi-degré extérieur d'un nœud est le nombre d'arcs partant du nœud (noté $d^+(x)$).

Le demi-degré intérieur d'un nœud est le nombre d'arcs arrivant au nœud (noté $d^-(x)$).

Le degré $d(x) = d^+(x) + d^-(x)$.

Si pour tout (x,y) de U , (y,x) appartient à U , le graphe est non orienté. l'ensemble $\{(x,y), (y,x)\}$ est une arête.

Un chemin est une séquence d'arcs dont le nœud final est le nœud initial du suivant. $((x_1, x_2), (x_2, x_3), \dots, (x_{k-1}, x_k))$ est noté $[x_1, x_2, \dots, x_k]$.

Un graphe est connexe si deux nœuds quelconques sont reliés par un chemin.

Soit A une partie de X , un arc (x,y) est incident vers l'extérieur si et seulement si $x \in A$ et $y \notin A$.

$\omega^+(A)$ est l'ensemble des arcs incidents vers l'extérieur.

Un p -graphe est un graphe où il n'y pas plus de p arcs reliant deux nœuds. En particulier, dans un 1-graphe, tous les éléments de U sont distincts. On a $d^+(x) = \text{card}(\{(x,y) \in U\})$, $d^-(x) = \text{card}(\{(y,x) \in U\})$.

Graphe dual

Un graphe admet une représentation graphique dans le plan par des courbes figurant les arcs et dont les extrémités figurent les nœuds.

Un graphe non orienté est dit planaire si et seulement si il admet une représentation où les arcs ne se coupent pas. Les parties connexes du plan délimitées par le graphe sont des faces.

Formule d'Euler: Si un graphe connexe planaire a N nœuds, M arêtes et f faces (y compris la face extérieure), on a: $N - M + f = 2$.

Pour un graphe planaire et connexe G , on définit un graphe dual G^* par:

- les nœuds de G^* sont les faces de G ,
- il existe un arc de G^* entre deux faces de G si et seulement si il existe un arc de G séparant ces deux faces (figure 81).

Il peut exister plusieurs graphes duales pour un graphe donné car plusieurs schémas sont possibles. Cependant, pour les graphes géographiques déduit de la géométrie, il n'existe qu'un graphe dual.

Structure de données de graphe

Un graphe peut être représenté par deux matrices.

La matrice d'incidence associée un graphe de N nœuds et M arcs est une matrice N lignes sur M colonnes de valeurs -1, 0 ou 1 définie par:

- $A_{ij} = 1$ / l'arc numéro i part du nœud numéro j,
- $A_{ij} = -1$ / l'arc numéro i arrive au nœud numéro j,
- $A_{ij} = 0$ sinon.

Pour un 1-graphe (c'est à dire qu'il n'existe pas plus d'un arc allant d'un nœud à un autre), la matrice d'adjacence est une matrice N sur N de booléens définie par: A_{ij} si et seulement si un arc va du nœud i au nœud j.

Ces matrices sont utilisables dans la plupart des algorithmes mais elles croissent en $O(n^2)$ avec le nombre de nœuds. Les graphes sont donc plutôt stockés dans des structures d'encombrement $O(n)$ qui ressemblent à celle des cartes:

```
struct Arc {
    ... valeurs                /* valeurs éventuelles */
    struct Noeud *début; /* référence du nœud de départ */
    struct Noeud *fin;  /* référence du nœud d'arrivée */
};

struct Noeud {
    ... valeurs /* valeurs éventuelles */
    int degré; /* nombre d'arcs arrivant et partant */
    /* tableau des références d'arcs partant et arrivant au nœud */
    struct Arc *omega[MAXARCS];
};
```

Structure de données de graphe dual

Le graphe dual peut être ajouté à une structure Arc/Nœud (par convention, l'arc du dual part de la face à gauche de l'arc du graphe et arrive à la face de droite.

```
Struct Arc {
    valeurs, début, fin... /* structure de graphe */
    struct Face *gauche;
    struct Face *droite;
};

struct Face {
    ... valeurs                /* valeurs éventuelles */
    int degréDual; /* nombre d'arcs entourant une face */
    /* tableau des références d'arcs entourant une face */
    struct Arc *omegaDual[MAXARCS];
};
```

Lien entre les cartes combinatoires et les graphes

A chaque carte, on associe un graphe sur l'ensemble des orbites de σ , dont les arcs sont les orbites de α .

Les définitions de cartes duales ou planaires correspondent à celles des graphes. Une carte est planaire si et seulement si son genre est nul alors que si la formule d'Euler est vérifiée pour un graphe, cela n'entraîne pas qu'il est planaire.

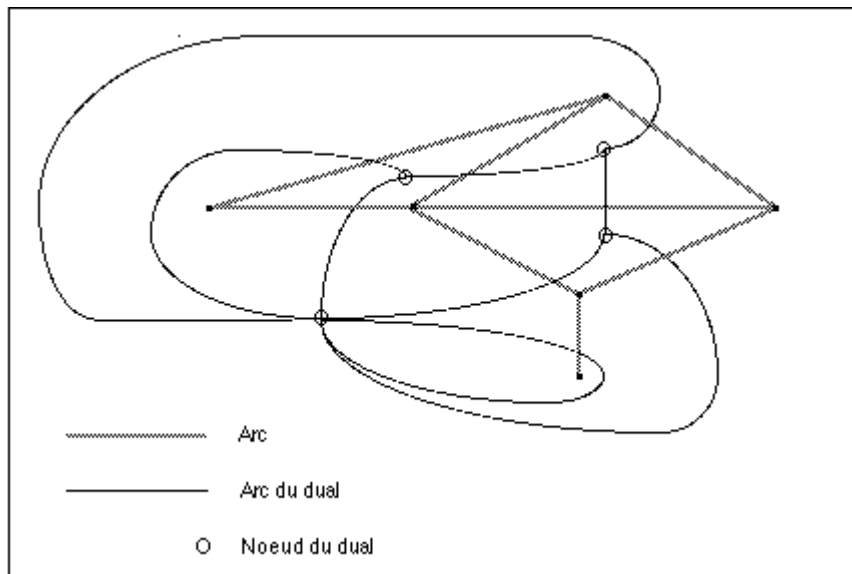


figure 81

Applications topologiques

Plus court chemin

Problème: soit un graphe connexe dont chaque arc porte une valeur positive notée L et appelée longueur; Soit un nœud particulier S_0 ; trouver, pour chaque nœud s , un chemin allant de S_0 à s et dont la somme des longueurs est minimale.

Ce problème s'applique à la recherche d'itinéraire: le graphe est le réseau routier et on veut aller d'une ville particulière à toutes les autres. La longueur est la distance ou le temps de parcours. Le problème usuel est la recherche d'un itinéraire pour une destination particulière, mais l'algorithme est aussi coûteux que pour toutes les villes.

Propriété fondamentale: Soit s' un nœud sur un plus court chemin entre S_0 et s , alors la restriction du chemin de S_0 à s' est un plus court chemin pour s' .

L'algorithme proposé est celui de Moore-Dijkstra. Soit $E(s)$ la fonction qui associe à un sommet s la longueur minimale des chemins reliant S_0 à s (que l'on appellera l'éloignement de s dans la suite).

La recherche du plus court chemin se fait en deux étapes. La première consiste à affecter à chaque nœud son éloignement $E(s)$. La deuxième est la recherche d'un plus court chemin entre S_0 et un point particulier s : Pour cela on part de s et on cherche un arc (s_{-1}, s) tel que $L(s_{-1}, s) = E(s) - E(s_{-1})$. Il en existe toujours un et il est sur un plus court chemin (pour la démonstration, voir le théorème ci-après). On répète en remplaçant s par s_{-1} jusqu'à $s_{-1} = S_0$.

Dans la suite, on traitera la première étape.

Soit D la liste partielle des sommets triée par ordre croissant d'éloignement; Au sens des structures élémentaires du début du cours, D est un dictionnaire dont la clé est $E(s)$.

Soit I_D l'ensemble des sommets des arcs incidents vers l'extérieur de D qui n'appartiennent pas à D et H_D l'ensemble des sommets des arcs incidents vers l'extérieur de D qui appartiennent à D (H_D est inclus dans D).

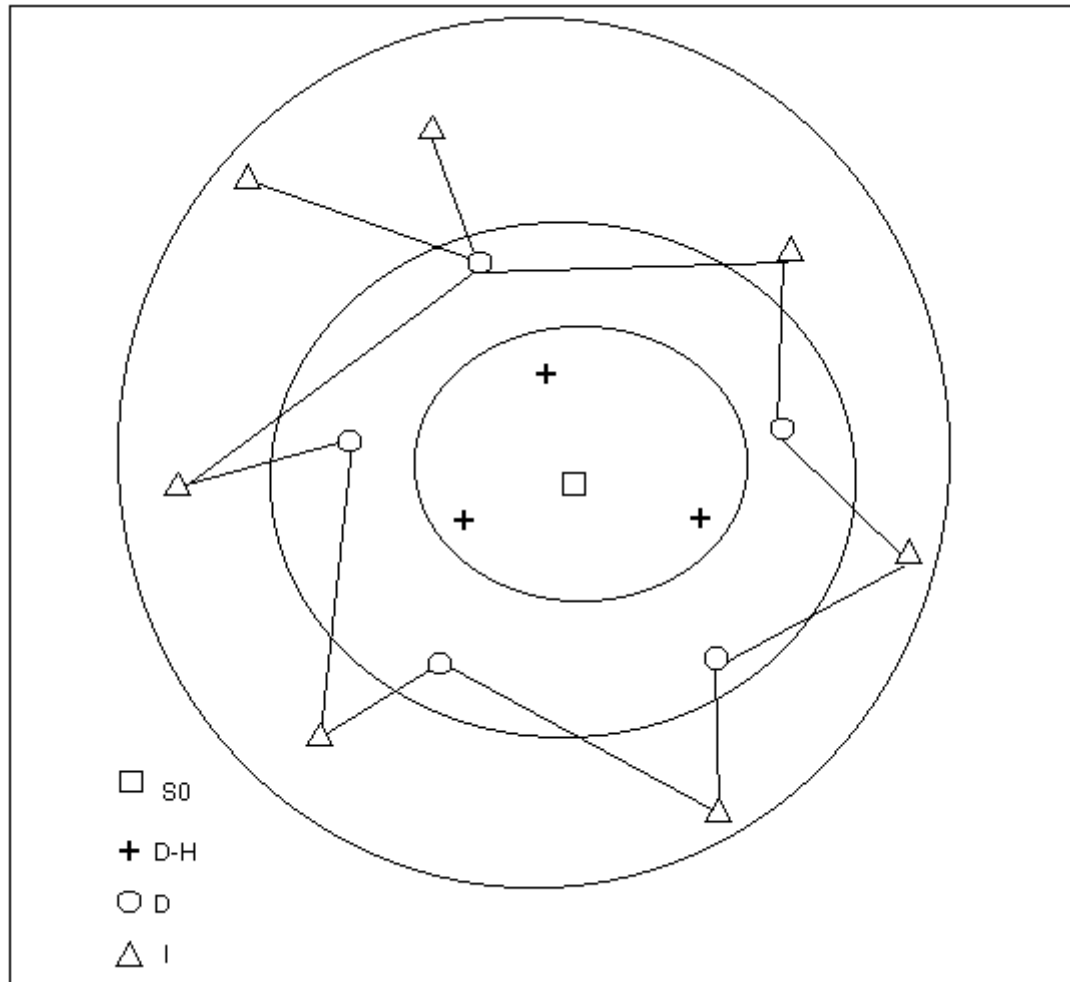


figure 91

Le but est de construire D.

Théorème: Soit un nœud s' de I_D dont la valeur $V = E(s) + L((s, s'))$, s appartenant à D et (s, s') arc du graphe, est minimale. Alors

- 1) la longueur minimale des chemins $[S_0, \dots, s']$ est V ,
- 2) V est la plus petite longueur des chemins allant de S_0 à $s \in D$.

Démonstration:

1) Soit $[S_0, \dots, s']$ un chemin minimal de S_0 à s' . Soient s_1 le premier nœud extérieur à D et s_2 le nœud précédent. s_1 appartient à I_D et s_2 appartient à H_D . Comme V est la longueur d'un chemin:

$$E(s_2) + L((s_2, s_1)) + L((s_1, \dots)) + \dots + L((\dots, s')) \leq V$$

$$\text{et } E(s_2) + L((s_2, s_1)) \geq V \text{ car } V \text{ est le minimum de } E(s) + L((s, s'))$$

$$\text{d'où } L((s_1, \dots)) + \dots + L((\dots, s')) = 0 \text{ ou } s_1 = s'.$$

2) Soit $[S_0, \dots, s]$ un chemin de S_0 à $s \in D$. Soient s_1 et s_2 définis comme précédemment.

Longueur de $[S_0, \dots, s]$

$$\geq E(s_2) + L((s_2, s_1)) + L((s_1, \dots)) + \dots + L((\dots, s))$$

$$\geq V + L((s_1, \dots)) + \dots + L((\dots, s)) \geq V$$

s' est donc le prochain sommet à entrer dans la liste D.

Ce théorème montre que la recherche du sommet le moins éloigné de S_0 qui n'est pas dans D est simplifiée par :

- 1) la recherche se réduit aux éléments de I_D ,
- 2) il n'est pas utile d'explorer $D-H_D$. Il suffit de faire comme si S_0 était relié aux sommets s de H_D par un arc de longueur $E(s)$.

Structure de données

On numérote les noeuds de 0 à n. La structure de donnée utilisée en pratique comprend:

- une structure de graphe (matrice ou structure arc/noeud)
- un tableau TC à une dimension qui contient:
 - pour les noeuds de D: $TC[j] = E(s_j)$
 - pour les noeuds de I: $TC[j] = \min(E(s) + L(s, s_j))$ (s noeud de D)
 - pour le reste: $TC[j]$ est une valeur très grande.
- une file d'attente FA de numéros de noeud de I triée par ordre croissant de TC. Le prochaine à entrer dans D sera le premier de la liste.

La structure de graphe existe au départ et ne change pas. La liste I et le tableau TC sont remplis et construits progressivement. A la fin le tableau TC contiendra les éloignements.

Algorithme

Initialisation :

$D = \emptyset$

$I = \{ S_0 \}$ c'est à dire:

$TC[0] \leftarrow 0$ /* $E(S_0) = 0$ */

$TC[j] \leftarrow +\infty$ pour $j \neq 0$

0 est placé dans la file d'attente FA.

Itération:

Tant que $FA \neq \emptyset$,

$sp \leftarrow$ premier noeud de FA

/* sa valeur est le minimum */

on enlève p de FA,

pour $s_j \in \Gamma^+(s_p)$

si $TC[j] = +\infty$

$TC[j] \leftarrow TC[p] + L(sp, s_j)$

j est inséré dans FA

sinon

si $TC[j] > TC[p] + L(sp, s_j)$

$TC[j] \leftarrow TC[p] + L(sp, s_j)$

j est remplacé dans FA

Un exemple de construction de D est présenté par la figure 91.

Temps de calcul: Soit N est le nombre de nœuds et M le nombre d'arcs. Les opérations (ajout, effacement, accès) sur une liste triée de k éléments se font en $O(\log(k))$. Il y a N passages dans la boucle "tant que.." et environ $\text{card}(\Gamma^+(s))$ passages dans "pour s_j ...".

Or $\sum_{s=1}^N \Gamma(s) = M$. D'où $N \text{ card}(\Gamma^+(s))$ est de l'ordre de M . L'algorithme est donc en $O(N \log(k) + M \log(k))$. M étant généralement supérieur à N , on peut considérer que l'algorithme est en $O(M \log(N))$.

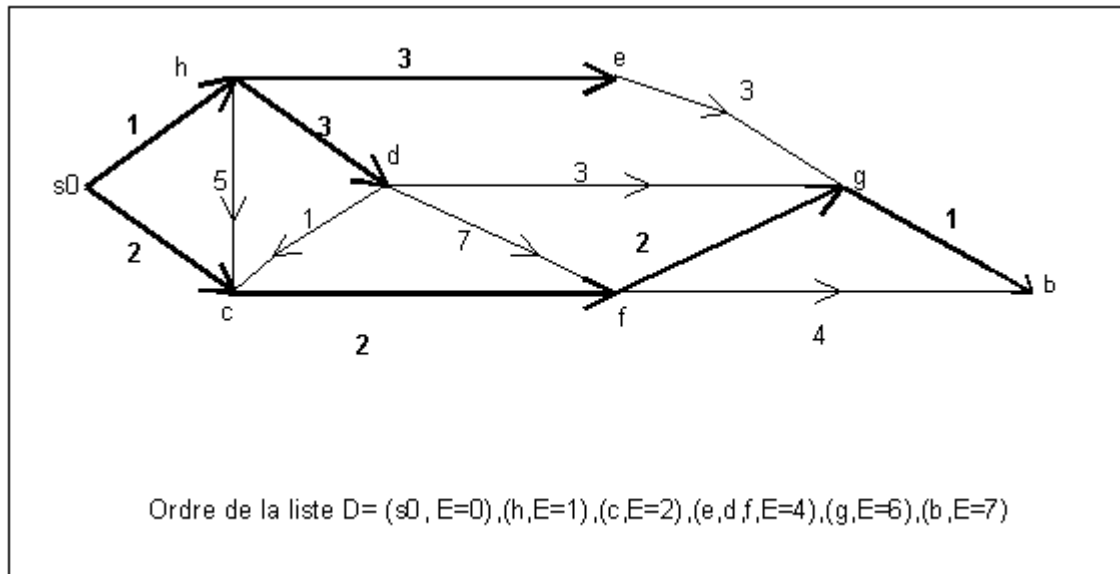


figure 92

Dans la construction de D , on part d'un élément de D et on ne revient jamais dans D . On construit donc un arbre qui passe par tous les nœuds et tel que chaque chemin partant de s_0 et contenu dans l'arbre est un plus court chemin.

Optimisation

On peut tenir compte d'informations extérieures au graphe pour supprimer des branches de l'arbre. En particulier, si la position d'un nœud devient trop éloignée du segment liant les points de départ et d'arrivée, on peut l'extraire de I_D .

En ce qui concerne un réseau routier, on peut imposer que, pour chaque nœud de l'arbre, les arcs sortants soient de meilleure viabilité que l'arc entrant. On doit alors partir des deux extrémités pour que cette règle s'applique au départ et à l'arrivée s_d et s_a . On a deux longueurs minimales E_d et E_a . Pour chaque itération de l'algorithme, on ajoute le nœud de I_d ou I_a dont la valeur $E(s) + L((s, s'))$ est la plus petite. On s'arrête lorsqu'un nœud est commun. La solution trouvée est approchée (Ce n'est pas le meilleur chemin).

Allocation

Etant donné un ensemble de nœuds $\{ X_i \}$, on cherche pour chaque sommet du graphe, le nœud X_i le plus proche.

On peut utiliser un algorithme proche du précédent. On considère la liste D_i associée à chaque X_i et $E_i(s)$ la longueur minimale des chemins entre X_i et s .

L'itération se fait sur tous les nœuds. On cherche pour chaque D_i , le suivant dans la liste et on insère celui dont $E_i(s) + L(s,s')$ est minimum pour tout i

Superposition de polygones

Superposition de deux polygones

Un polygone est l'ensemble des points qui sont à l'intérieur. On peut utiliser les opérateurs ensemblistes \cap , \cup , complémentaire. Par exemple la différence $G - H = G \cap \overline{H}$.

Certains auteurs confondent les opérations sur les ensembles et les opérateurs logiques sur les points. Par exemple: $G - H$ sera écrit:

G et non (G et H)

car $G-H = \{ P \mid P \in G \text{ et non } (P \in G \text{ et } P \in H) \}$

On a donc la correspondance:

\cup = ou, \cap = et, $\overline{}$ = non.

Les définitions précédentes soulèvent le problème théorique de l'appartenance de la frontière soit à un polygone soit à son complémentaire. En pratique, si on ne travaille que sur des polygones, cela n'a pas d'importance.

Par contre, le fait que l'intersection de deux polygones connexes n'est pas connexe pose des problèmes pratiques lorsque on veut construire une bibliothèque générale d'opérations élémentaires: \cap , \cup ... On doit prévoir que ces fonctions utilisent des listes de polygones.

Superposition de partitions

La superposition de cartes ou de calques est une manipulation usuelle lors d'analyse géographiques ou lors des opérations photographiques en cartographie (masque...). On cherche donc à définir des opérateurs mathématiques qui correspondent à des superpositions manuelles (et qui offrent si possible encore plus de possibilités).

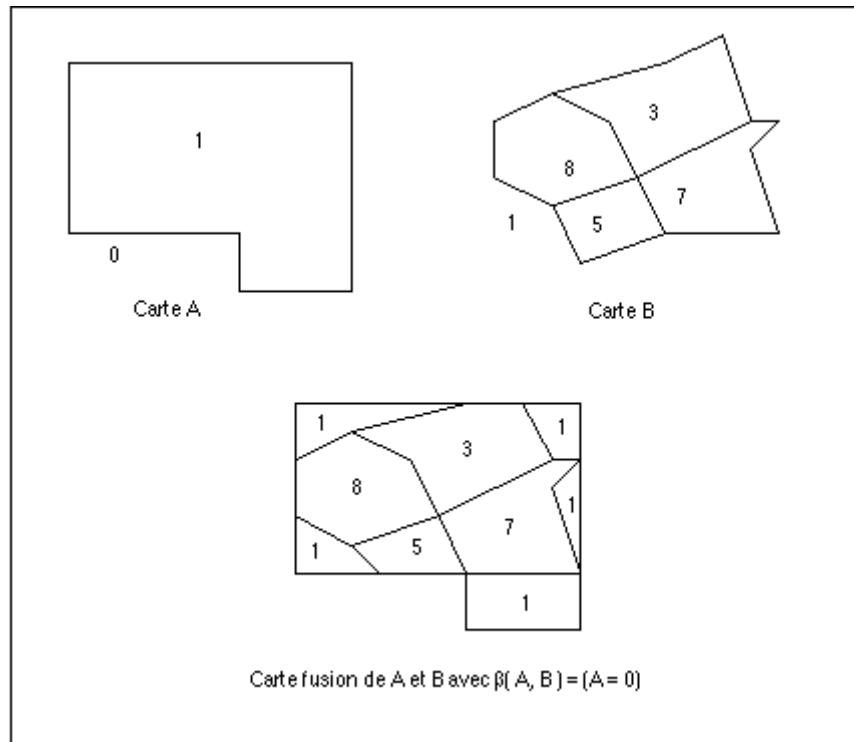


figure 93

Ces opérations ressemblent à des jointures en algèbre relationnelle.

Soit une partition formée de polygones qui portent des valeurs d'attributs $A_1...A_n$. Par exemple le découpage administratif en communes peut faire porter à chaque polygone le numéro INSEE, le nom...

Soit une autre partition portant les attributs $B_1...B_p$. La superposition simple est la partition:

- portant les attributs $A_1...A_n, B_1...B_p$
- formée des polygones $G \cap H$ tels que $G \in 1^{\text{ère}}$ partition et $H \in 2^{\text{ème}}$ partition
- et dont les valeurs d'attributs de $G \cap H$ sont celles de $A_1...A_n$ pour G et $B_1...B_p$ pour H .

Exemple: superposition d'une carte de végétation et d'une carte de précipitation.

On peut construire des superpositions plus complexes: Supposons que $n = p$, que A_i et B_i soient de même type, et qu'il existe un prédicat $\beta(A_1...B_n)$. On peut former la partition des polygones $G \cap H$ tel dont les valeurs soient celles de $A_1...A_n$ si $\beta(A_1...B_n)$ et $B_1...B_n$ sinon.

Ce type de fusion est utilisée pour faire des fenêtrages (Cf.figure 93)

Traitements altimétriques

Les traitements altimétriques ont d'abord pour objectif d'estimer la surface terrestre à partir des informations disponibles. On peut partir d'un semis de points et construire une surface à facettes dont les nœuds sont les points. C'est notamment le but de la triangulation. On peut également estimer l'altitude des mailles d'un MNT.

D'autres traitements fournissent des produits dérivés d'un MNT.

Triangulation

Etant donné un ensemble fini de points, une triangulation est un ensemble de segments qui joignent les points, qui ne se coupent pas et dont chaque face intérieure est un triangle. Il y a généralement plusieurs triangulations possibles pour un semi de points. La plus usitée est la triangulation de Delaunay. Elle est fondée sur les zones de proximité autour des points.

Problème de proximité

Soit $E = \{ S_i \} \ i=1..N$, un ensemble de points (appelé sommets). La zone où on se trouve plus près d'un sommet S_i que de tout autre est appelée polygone de Voronoï:

$$V_i = \{ P, \text{ tels que } PS_i < PS_j \text{ pour tout } j \text{ différent de } i \}$$

Le polygone de Voronoï est la région touchée par S_i dans un processus d'accroissement (Par exemple si chaque S_i est la source d'un fluide se répandant à vitesse uniforme).

Les polygones de Voronoï forment une partition. Les limites des polygones constituent un graphe appelé diagramme de Voronoï.

- Caractérisation des faces. Un point est intérieur à une face si il est plus près d'un sommet que de tous les autres.

- Caractérisation des arcs : Les points des arcs sont à égale distance de 2 sommets. ce sont donc des morceaux de médiatrices. Inversement, deux sommets sont dits voisins, si leurs polygones ont un arc en commun.

$$S_i \text{ et } S_j \text{ voisins} \Leftrightarrow \exists M \text{ tels que } S_i M = S_j M < S_k M \ \forall k \neq i, j$$

Corollaire: S_i et S_j sont voisins si et seulement si il existe un cercle passant par S_i et S_j et ne contenant aucun autre sommet.

Démonstration: Si S_i et S_j voisins, le cercle de centre M de rayon S_iM ne contient aucun autre sommet. Si le cercle existe, son centre est un point du diagramme.

-Caractérisation des nœuds. Un nœud est l'intersection de plusieurs arcs. Donc

$$N \text{ nœud} \Leftrightarrow \exists S_{i_1} \dots S_{i_n} \mid S_{i_1}N = S_{i_2}N = \dots = S_{i_n}N < S_kN \quad \forall k \neq i_1 \dots i_n$$

$$\Leftrightarrow S_{i_1} \dots S_{i_n} \text{ sont sur le même cercle (de centre } N)$$

Théorème: Si E ne contient pas quatre sommets co-circulaires et si les sommets ne sont pas colinéaires, alors on construit une triangulation en joignant les sommets voisins. Cette triangulation est appelée triangulation de Delaunay. (figure 101)

Démonstration: le graphe dual du diagramme de Voronoï vérifie les propriétés suivantes:

Un nœud du dual est une face du diagramme (donc un polygone). On peut donc identifier ce nœud au sommet du polygone.

Un arc du dual existe si et seulement si un arc du diagramme existe c'est à dire si les deux sommets sont voisins.

D'après la caractérisation des nœuds N du diagramme, si les sommets ne sont pas co-circulaires alors $\text{degré}(N) \leq 3$. Or les nœuds sont des intersections d'au moins 3 arcs; Donc $\text{degré}(N) = 3$. Donc les faces du dual sont limitées par 3 arcs. C'est une triangulation.

La triangulation de Delaunay est donc le graphe dual du diagramme de Voronoï. La propriété de joindre les sommets voisins (et en particulier les sommets les plus proches) fait que cette triangulation est "bien proportionnée".

On démontre les propriétés suivantes de la triangulation:

-Les arêtes sur le bord de l'enveloppe convexe est une arête de la triangulation.

-La triangulation de Delaunay est également définie comme la seule triangulation telle que le cercle circonscrit aux sommets de chaque triangle ne contient aucun autre point. Cette propriété est utilisée pour la construction.

Construction d'une triangulation de Delaunay

La construction d'une triangulation de Delaunay est un exemple de la stratégie "diviser pour régner". Dans ce type d'algorithme, on simplifie les informations géométriques en les divisant. La division est généralement récursive en deux parties. On résout le problème simple et on recompose les résultats partiels.

Cette stratégie est utilisable lorsque:

- le problème se simplifie en se divisant,
- on sait résoudre les cas simples,

-on sait recomposer deux problèmes.

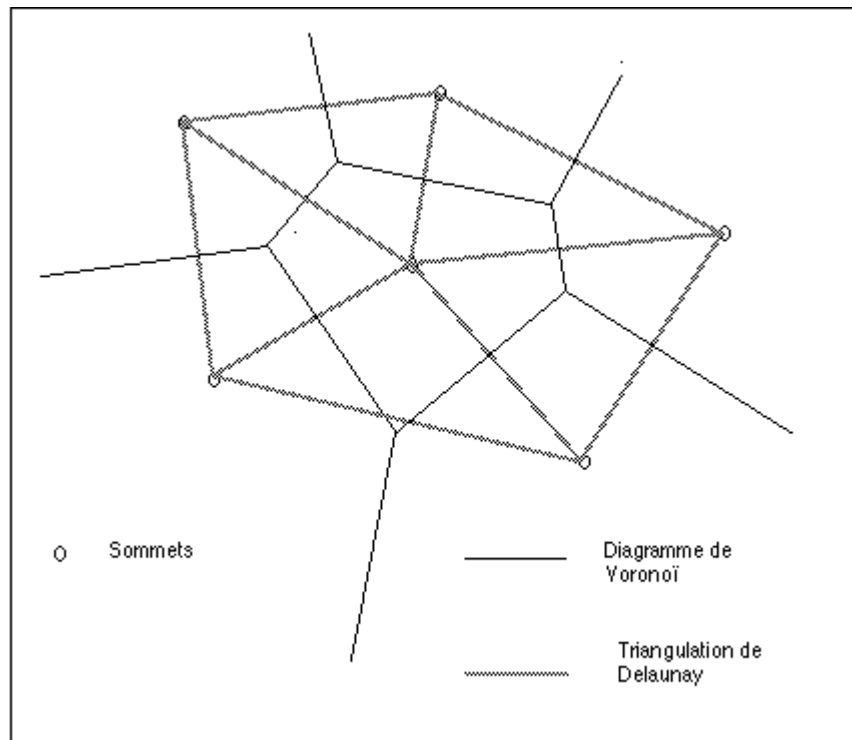


figure 101

Pour la triangulation de Delaunay, ces conditions sont remplies:

- On peut construire un arbre géométrique binaire ou un quad-tree qui diminue le nombre de points dans chaque dalle.
- Les cas simples sont les dalles avec 0 ou 1 sommet (rien), 2 sommets (un segment), 3 sommets (un triangle).
- Il reste la recombinaison de deux dalles. C'est à ce niveau que réside la difficulté que l'on va résoudre par l'algorithme présenté ci-dessous

Le problème s'énonce ainsi: Etant données deux triangulations sur deux ensembles de sommets séparés par une droite horizontale ou verticale, trouver la triangulation de l'union. Supposons que la droite de séparation soit verticale.

La présentation de l'algorithme nécessite un préambule sur les propriétés géométriques et topologiques de la triangulation et de son dual. Dans la suite, on parlera de sommet pour désigner les S_i , de "polygone" pour "polygone de Voronoï", de "diagramme" pour "diagramme de Voronoï" et de "triangulation" pour "triangulation de Delaunay".

Propriétés géométriques utilisées

Soit E_g (à gauche) et E_d (à droite) les ensembles de sommets de part et d'autre de la ligne verticale.

Revenons aux diagrammes. Soit G_g (resp. G_d) la surface qui contient les points plus près d'un sommet de E_g (resp. E_d) que de tous les sommets de E_d (resp. E_g).

$$G_d = \{ M \mid \exists S_d \in E_d ; S_d M < S_g M \forall S_g \in E_g \}$$

Cette définition peut être interprétée de 2 façons.

1) Pour chaque sommet S_i de E_d , soit V_i son polygone est calculé en tenant compte de tous les sommets de E et V_{di} son polygone obtenu en se restreignant aux sommets de droite, alors :

$$V_i = V_{di} \cap G_d$$

2) On montre aisément que G_d est l'union des polygones des sommets de droite. $G_d = \cup V_i \mid S_i \in E_d$. Donc G_g et G_d forment une partition et il existe une ligne L qui sépare G_g et G_d .

L est utile pour deux raisons :

D'une part elle permet à l'algorithme d'identifier les entités qui vont être modifiées (ce sont celles qui sont liées aux sommets de G_d (resp. G_g) se trouvant à gauche (resp. à droite) de L . Le calcul préalable de L accélère donc le traitement car on examinera ensuite que ce qui est nécessaire.

D'autre part L va permet de déterminer quels sont les types de modifications nécessaires.

Propriétés de L

N.B. Les propriétés énoncées ci-dessous ne sont pas démontrées.

L est une limite du diagramme complet de E . En tout point, on peut trouver 2 sommets de E_d et E_g dont c'est la médiatrice. Donc elle est monotone verticale (l'ordonnée est toujours croissant ou décroissant).

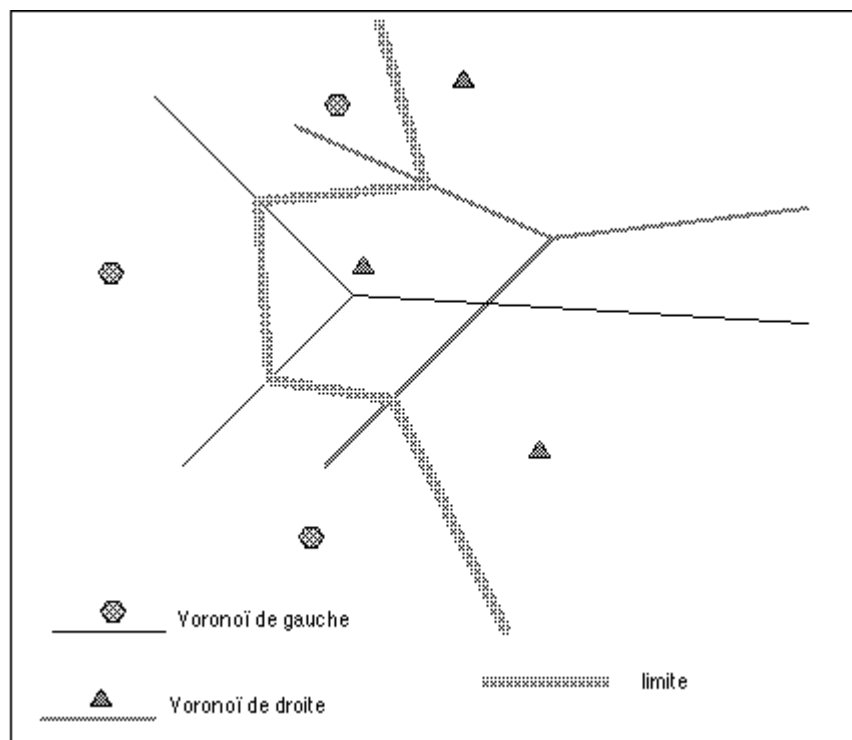


figure 102

L est formée d'une seule ligne brisée.

Les extrémités de L sont formées de deux demi-droites. En effet, pour former l'enveloppe convexe de 2 enveloppes, il ne faut ajouter que deux segments.

Identification des modifications Construction de L

La construction de L consiste à trouver les couples de sommets (1 à droite noté D, 1 à gauche noté G) dont la médiatrice supporte une partie de L. Cela se fait par l'algorithme suivant:

Initialement, on calcule les deux segments de l'enveloppe puis les demi-droites médiatrices. On part de la demi-droite du bas et on s'arrête lorsque l'on atteint la demi droite du haut. Les extrémités du segment du bas de l'enveloppe donnent les valeurs initiales à D et G

Itération: Supposons calculés les couples jusqu'à deux sommets D et G. La limite L est portée par la médiatrice de D et G. Le couple suivant contient D ou G et un sommet au dessus de G ou D noté supG ou supD. La figure 102 montre que la limite L change de direction chaque fois que L coupe un arc du diagramme. On cherche donc le premier arc que va rencontrer L. On sait que cet arc est limite du polygone de G ou de D et donc supG (resp. supD) est voisin de G (resp. D).

La recherche du premier arc se fait en 2 temps :

1) deux recherches du premier arc parmi les arcs limitant le polygone de D puis de G et du sommet voisin associé à l'arc (supD puis supG).

2) recherche parmi supD et supG, du sommet dont l'arc est le plus près de la limite.

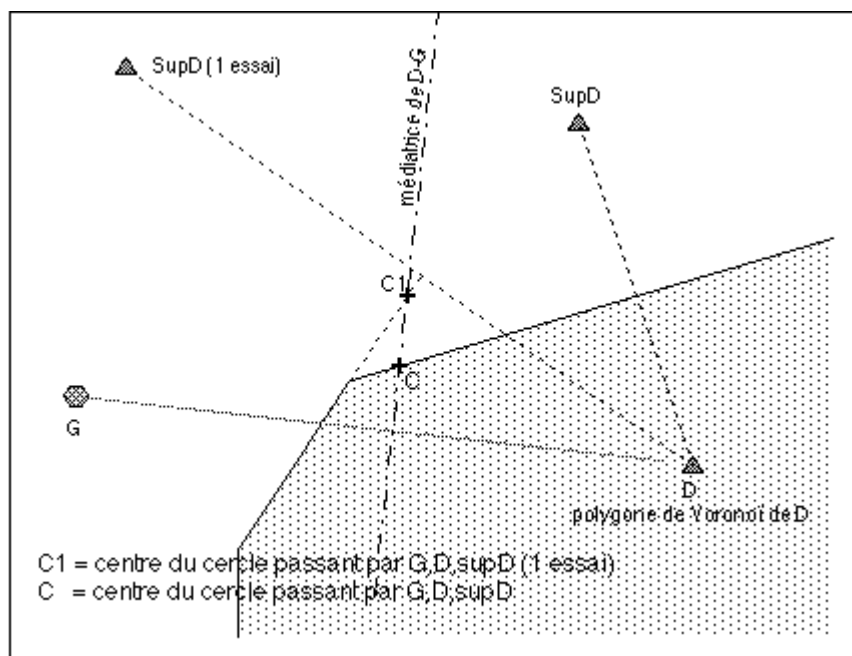


figure 103

Pour la recherche 1), on parcourt les sommets de E_d (resp. E_g) voisins de D (puis G) en partant de la direction $D \rightarrow G$ (resp. $G \rightarrow D$) et en tournant dans le sens inverse (resp. trigonométrique).

Soit supD (resp. supG) le sommet courant. Soit C l'intersection entre la médiatrice de $[D G]$ et celle de $[D \text{supD}]$ (Cf. figure 103). C est centre du cercle circonscrit à D , G et supD . On veut que l'ordonnée de C soit minimum. On montre

a) que cela est vérifié si le cercle circonscrit à D , G et supD ne contient aucun autre sommet de E_d ,

b) qu'il suffit de vérifier que ce cercle ne contient pas le point suivant dans le parcours autour du polygone.

N.B. a) est une conséquence de la propriété qu'un cercle circonscrit à un triangle de Delaunay ne contient aucun autre point. Pour b) c'est une conséquence de la convexité des polygones.

Cependant, il existe le cas particulier où D (resp. G) est sur l'enveloppe convexe du diagramme de E . il risque d'y avoir un saut dans le parcours des sommets voisins dans le sens inverse (resp. trigonométrique) car le polygone est ouvert. Il faut alors arrêter le déplacement de D (resp. G).

Pour la recherche 2) On a obtenu deux intersections à droite et à gauche qui sont des centres de cercles passant par D et G . L'intersection la plus proche de D et G est celle dont le cercle ne contient pas l'autre point. (Cf. figure 104).

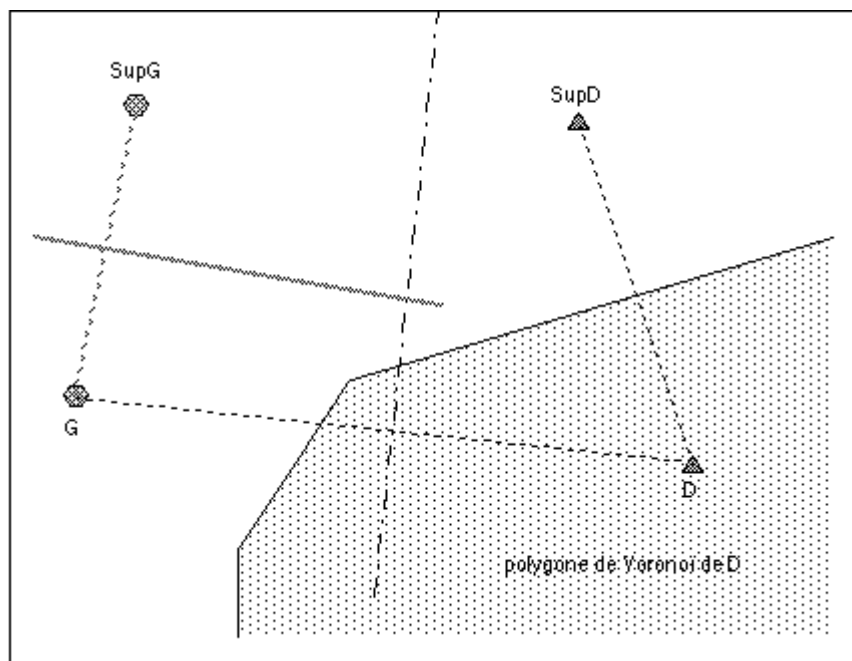


figure 104

Réalisation des modifications

Les modifications à apporter au cours de la construction de L sont les suivantes.

Pour chaque couple de sommets DG , on a montré qu'ils sont voisins. Il existe un arc du diagramme complet qui sépare les polygones de D et de G , et il existe un arc de la triangulation entre reliant D et G .

Si L passe au delà d'un arc du diagramme (comme dans les figures 102 et 103), on a montré que le prolongement de cet arc coupe la médiatrice de DG au dessus du suivant. Alors l'arc du diagramme disparaît dans le diagramme complet. Donc l'arc de la triangulation est effacé.

Application à la triangulation

L'algorithme présenté travaille sur les sommets et les arcs du diagramme. Une opération sur un arc du diagramme se transpose aux arcs du dual. Donc, on peut reprendre le même algorithme pour la triangulation.

La structure de donnée est celle d'une carte combinatoire. On a des sommets, des arcs. On peut se déplacer autour d'un nœud ou autour d'un triangle.

Plus précisément, on utilise les fonctions suivantes (les paramètres des fonctions sont des arcs ou des sommets):

Trouver l'arc de sommets s_1 s_2 :

```
struct arc segment( struct sommet s1, s2 );
```

Précédent de l'arc a en s:

```
struct sommet prec( struct arc a, struct sommet s );
```

Suivant de l'arc a en s:

```
struct sommet suiv( struct arc a, struct sommet s );
```

Insérer l'arc entre s_1 et s_2 :

```
void inser( struct sommet s1, s2 );
```

Effacer l'arc a:

```
void efface( struct arc a );
```

Vérifier que le sommet x est intérieur au cercle passant par s_1, s_2, s_3 . C'est à dire si la somme des angles en x et s_2 du quadrilatère s_1, s_2, s_3, x est supérieure à π :

```
boolean inclus( struct sommet s1, s2, s3, x );
```

Valeur de l'angle $s_1 s_2 s_3$:

```
float angle( struct sommet s1, s2, s3 );
```

Initialement, l'enveloppe convexe est formée en ajoutant deux arcs: A_bas et A_haut. Les deux triangulations vont être parcourues de bas en haut. On ajoute ou on supprime des arêtes, en partant de A_bas jusqu'à A_haut.

Itération: Soit l'arc courant dont les extrémités sont D et G.

La recherche 1) se traduit dans le dual par le parcours de sigma autour du nœud D (resp. G) dans le sens inverse (resp. trigonométrique) jusqu'à ce que le cercle circonscrit à D,G et supD (resp. supG) ne contienne pas le suivant (noté sup_sup dans l'algorithme ci-après).

Dans le cas contraire, on a vu que l'arc de Voronoï disparaît, donc l'arc de Delaunay est effacé.

La recherche 2) consiste à tester si supG est dans le cercle circonscrit à G, D, et supD. Si c'est le cas l'arc [supG D] est à insérer, sinon c'est l'arc [G supD].

Le programme ci-dessous reprend cette démarche. De plus il vérifie que les angles G-D-supD et D-G-supG ne sont jamais convexe qui est un problème qui peut survenir si l'arête D-supD ou G-supG est à la périphérie.

```

struct arc a;
struct sommet D, G;
struct sommet supD, supG;
boolean depasse_droit, depasse_gauche;
droit= A_bas_droit;
gauche= A_bas_gauche;
while (droit != A_haut_droit && gauche != A_haut_gauche)
{
    arc= inser( G, D );
    supD= prec( arc, D );
    supG= suiv( arc, G );
    /* Recherche 1 */
    depasse_droit= angle( G, D, supD ) < pi;
    depasse_gauche= angle( D, G, supG ) > pi;
    if (!depasse_droit)
    {
        sup_sup= prec( segment( D, supD ), droit );
        while (inclus( G, D, supD, sup_sup ))
        {
            efface( segment( D, supD ) );
            supD= sup_sup;
            sup_sup= prec( segment( D, supD ), D );
        }
    }
    if (!depasse_gauche)
    {
        sup_sup= suiv( segment( G, supG ), G );
        while (inclus( D, G, supG, sup_sup ))
        {
            efface( segment( G, supG ) );
            supG= sup_sup;
            sup_sup= suiv( segment( G, supG ), G );
        }
    }
    if (depasse_droit)
        G= supG;
    else if (depasse_gauche)
        D= supD;
    /* Recherche 2 */
    else if (inclus( G, D, supD, supG ))
        G= supG;
    else
        D= supD;
}
}

```

Performance: Soit N' le cardinal de E_d et de E_g . La recherche des arêtes A-inf et A-sup peut se faire en temps $O(N')$. Le nombre d'itérations de l'algorithme ci-dessus est proportionnel au nombre de sommets sur un coté c'est à dire en $O(N')$.

Soit $N = \text{card}(E)$; On effectue environ $\log(N)$ divisions de l'ensemble pour obtenir les cas simples (0, 1 2 ou 3 points). Au niveau p, il y a $N/2^p$ ensembles de 2^p éléments. Chaque niveau est traité en $N/2^p O(2^p) = O(N)$. D'où, le temps total est $\log(N) O(N) = O(N \log(N))$.

Calcul d'un MNT maillé

Un MNT maillé peut être dérivé d'une triangulation en calculant pour chaque nœud du maillage, le triangle concerné et l'altitude du point par interpolation.

Cependant il peut être calculé à partir de courbes de niveau. On calcule des profils selon une des droites du maillage. Un profil est une approximation de coupes verticales du terrain. Pour une coupe selon l'axe Y (plan $X=cste.$), on calcule les coordonnées y et z des intersections avec les courbes de niveau. Ces points sont triés selon y. On relie les points par des segments de droite ou de cubique. Enfin on calcule les altitudes aux nœuds du maillage.

Les coupes selon X et Y donnent deux altitudes par nœud. La valeur prise est une pondération en fonction de la proximité des courbes.

On peut améliorer l'algorithme en ajoutant des coupes diagonales ($X+Y=cste.$ et $X-Y=cste.$).

Exploitation des triangulations

On peut transformer un MNT en une triangulation en coupant chaque maille en deux triangles.

Sur une triangulation, il est simple de calculer un estompage en affectant à chaque triangle un niveau de gris proportionnel à l'angle entre la normale et la direction du soleil.

L'intervisibilité et l'ensoleillement sont des problèmes équivalents à l'élimination des faces cachées: La perspective est conique pour l'intervisibilité et parallèle pour l'ensoleillement. Il faut noter que les algorithmes simples comme le Z-buffer détruisent la structure des triangles et ne produisent qu'une image. Les algorithmes conservant cette structure sont assez complexes.

Exploitation des MNTs

Propriétés différentielles de la surface du terrain

La régularité d'un MNT facilite la représentation du relief sous forme d'une fonction:

$Z = H(x, y)$ H étant continue et suffisamment dérivable.

On peut estimer les dérivées en un nœud du maillage en fonction des altitudes du voisinage.

Par exemple, on fait passer au plus près des points du voisinage une fonction polynôme de type $H(x,y) = h_0 + ax + by + cx^2 + dxy + ey^2 + \dots$. On calcule les coefficients $h_0, a, b \dots$ de façon à minimiser la distance entre l'altitude du MNT aux nœuds du voisinage et la valeur du polynôme aux nœuds. Ensuite on déduit les dérivées (par exemple $dH/dx = a$, $d^2H/dx^2 = c/2$).

Contribution des MNT à la géomorphologie

La géomorphologie décrit la forme du terrain. Les notions élémentaires sont les points et les lignes caractéristiques.

Les points caractéristiques sont les sommets, les cols et les cuvettes. Les lignes caractéristiques sont les crêtes et les talwegs.

On cherche à retrouver les lignes caractéristiques constatées sur le terrain à travers des courbes définies par la géométrie différentielle. Les traitements présentés ci-dessous sont très simples. La coïncidence n'est donc pas totale.

Cette structuration automatique du relief a des applications hydrologiques (écoulement) ou cartographique (généralisation conservant la position des crêtes et des vallées).

Propriétés locales

La pente p en un point est l'angle entre le plan tangent à la surface et le plan horizontal. L'orientation de la pente est la perpendiculaire de l'intersection entre le plan tangent et le plan horizontal. Par convention, le sens de la pente est vers les altitudes décroissantes (c'est la direction d'écoulement de l'eau).

La pente (Cf. figure 105) vérifie:

$$p = \arctg(|\text{grad}(H)|) = \arctg(\sqrt{(dH/dx)^2 + (dH/dy)^2}).$$

L'orientation de la pente est définie dans le plan horizontal par la droite de vecteur directeur $\text{grad}(H)$.

La pente n'est pas définie si $dH/dx = 0$ et $dH/dy = 0$. Dans ce cas, les points correspondent à peu près aux points caractéristiques:

Les sommets correspondent généralement aux points tels que:

$$d^2H/dx^2 < 0, d^2H/dy^2 < 0$$

Les cuvettes aux points tels que: $d^2H/dx^2 > 0, d^2H/dy^2 > 0$

Les cols sont les intersections entre deux fonds de vallées et deux crêtes. Une bonne caractérisation des cols est: $d^2H/dx^2 \cdot d^2H/dy^2 < 0$

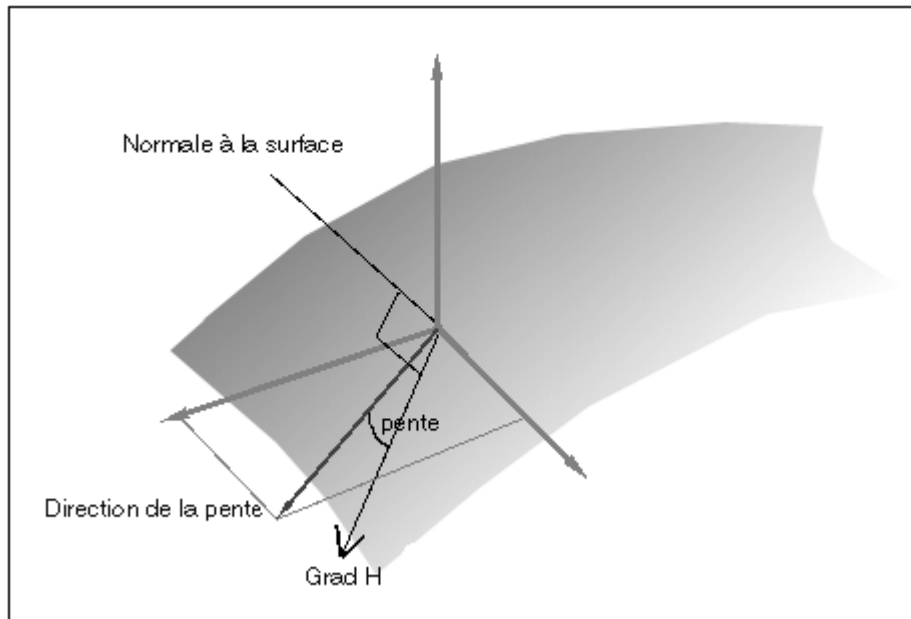


figure 105

Lignes caractéristiques

Les lignes de niveau sont les lignes $H=cste$. Par convention, une ligne de niveau est orientée pour que les points hauts soient à droite.

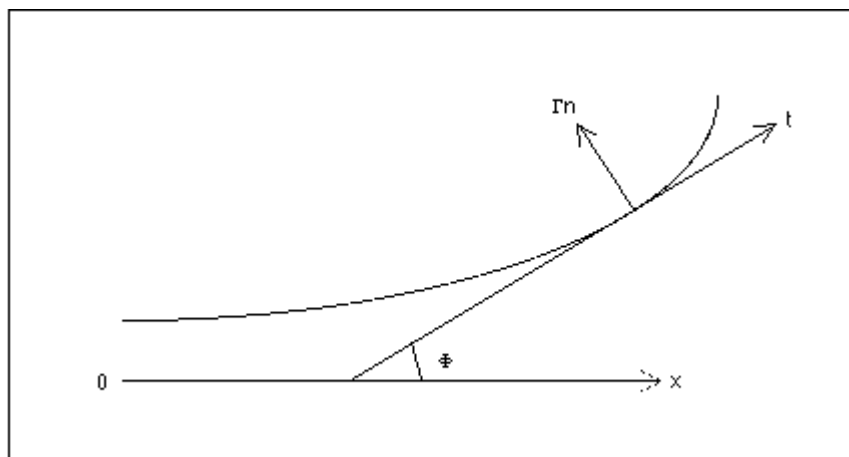


figure 106

La courbure des lignes de niveau Γ_n est définie par $d\Phi/ds$ où Φ est l'orientation de la tangente.

Un terrain est dit convexe si $\Gamma_n < 0$. Sinon il est concave. Un écoulement qui arrive dans une zone concave qui n'est pas limitée vers le bas ne peut plus en sortir. Donc les zones concaves (resp. convexes) sont des zones de talwegs (resp. de crêtes).

Plus précisément on peut définir une crête (resp. un talweg) par une ligne où Γ_n est minimum (resp. maximum).

Les talwegs (resp. crêtes) principaux sont alors les courbes de Γ_n extrémales qui partent des cols (resp. des sommets).

On peut aussi définir une crête comme la ligne tracée depuis un col jusqu'à un sommet en suivant la plus grande pente ascendante. Un talweg est une ligne tracée depuis un col jusqu'à une cuvette en suivant la plus grande pente descendante.

Conclusion

A partir des algorithmes présentés ci-dessus, on peut tirer quelques règles générales.

1) On peut — et il faut — estimer la performance des algorithmes. Cette estimation en $O(\dots)$ ne donne qu'un ordre de grandeur, mais elle est indispensable, car pour un même résultat, il existe des algorithmes qui demandent quelques minutes et d'autres plusieurs heures.

2) Certaines structures de données sont d'un usage très répandu : codage par plage, quad-tree, polyligne, structures de graphe... Sur ces structures et les algorithmes qui les utilisent, il existe une bibliographie importante à laquelle il faut se reporter avant toute analyse d'un problème.

3) Il y a deux façons de rendre un algorithme performant :

a) Trouver les méthodes qui permettent d'accéder le plus rapidement possible aux entités que l'on doit traiter;

b) Trouver les propriétés (généralement géométriques) qui traitent les entités avec un minimum de calcul.

Annexe : Fenêtrage

Comme complément des index géométriques, on présente pour chaque type d'index,
 -une mise en œuvre de l'index par des structure de données,
 -un algorithme de recherche des objets contenus dans une fenêtre.

k-d-tree

Structure

La structure d'arbre est celle d'un arbre binaire. L'autre champ contient le point.

```
Struct kD
{
  struct Point p;
  struct kD fils[2];
}
```

Fenêtrage

On parcourt l'arbre en examinant si toutes les branches doivent être suivies. En fait, on poursuit la recherche dans une branche que :

-si la coordonnée minimum du rectangle est inférieure à la coordonnée du noeud dans le cas d'une branche "fils[0]".

-si la coordonnée maximum du rectangle est supérieure à la coordonnée du noeud dans le cas d'une branche "fils[1]".

D'où l'algorithme suivant où "axe_x" est un booléen qui est vrai si on travaille sur l'axe X et faux si on travaille sur l'axe y.

```
Fenêtre(struct kD *d; struct Rectangle *r; booléen axe_x )
{
  booléen sous, sur;
  if (d != 0)
  {
    if (axe_x)
    {
```



```

        sous= (r->xmin < d->p.x);
        sur= (r->xmax > d->p.x);
    }
else
    {
        sous= (r->ymin < d->p.y);
        sur= (r->ymax > d->p.y);
    }
if (sous)
    Fenêtre( d->fils[0], r, ! axe_x );
if (inclusion( d->p, r)
    .../* point dans le rectangle */
if (sur)
    Fenêtre( d->fils[1], r, ! axe_x );
}
}

```

Quad-tree-point

Structure

La structure est semblable à celle du mode raster. La valeur de la dalle est le point éventuel. Le champ "type" indique s'il s'agit d'un noeud intermédiaire ou d'une feuille. S'il s'agit d'une feuille, le type indique si la dalle contient un point ou bien est vide.

Struct Noeud

```

{
    enum { vide, point, intermédiaire } type;
    struct Point p;
    struct Noeud fils[R];
}

```

Fenêtrage

On parcourt l'arbre en éliminant les dalles qui n'intersectent pas la fenêtre. On utilise deux fonctions géométriques : intersection et inclusion. Pour que ces deux fonctions, il est nécessaire de connaître les dimensions de la dalle. L'algorithme effectif est donc plus développé celui ci-dessous.

fenêtre(struct Noeud *q, struct rectangle *r)

```

{
    if (intersection( q, r );
        if (q->type == point)
        {
            if (inclusion( q->p, r )
                .../* point dans le rectangle */
            }
        }
        else if (q->type == intermédiaire)
            for (i= 0; i < 4; i++)
                fenêtre( q->fils[i], r );
    }
}

```

Dallage régulier

Structure

Si on se restreint à ne gérer que des points, un dallage simple peut être conservé dans une grille :

```
struct dalle *grille[NBR_DALLES_EN_X,NBR_DALLES_EN_Y];
```

qui référence des dalles de type:

```
typedef struct {
    int nbr_primitives;           /* Nombre de primitives
                                   ** effectivement dans la page */
    dalle *débordement           /* dalle suivante en cas de débordement */
    struct Point contenu[MAX_POINT]; /* Point dans la dalle */
} dalle;
```

Fenêtrage

Si les points sont organisés dans un dallage régulier, le test d'inclusion est limitée aux dalles qui ont une intersection avec le rectangle. On a donc l'algorithme (le débordement n'est pas géré pour simplifier) :

```
Fenêtre( struct Rectangle *r )
{
    struct dalle d;
    int i, j, k;
    for (i= r->xmin/TAILLE_DALLE_EN_X;
         i <= r->xmax/TAILLE_DALLE_EN_X; i++)
        for (j= r->ymin/TAILLE_DALLE_EN_Y;
             j <= r->ymax/TAILLE_DALLE_EN_Y; j++)
            /* dalles intersectant le rectangle */
            d= grille[i,j]
            for (k= 0; k<d->nbr_primitives; k++)
                if (inclusion( d->contenu[k], r))
                    ... /* point dans le rectangle */
}
```

R-tree

Structure

Le paramètre des R-Tree ou R+Tree est le nombre de rectangles inclus dans un rectangle de niveau supérieur (NR=4 dans la figure 58).

Les nœuds intermédiaires ont une structure différente de celle des feuilles. Un nœud intermédiaire comporte deux tableaux contenant respectivement les rectangles et les pointeurs vers le nœud qu'englobe ce rectangle. Une feuille est un tableau de points (dans le cas simplifié où les primitives sont des points). La constante PdF est calculée pour qu'une feuille soit de taille inférieure ou égale à un nœud intermédiaire.

```
#define NR          /* nombre maximum de rectangle */

int Prof           /* profondeur courant du R+Tree */

struct Noeud {      /* Noeud intermédiaire */
    int nb_fils;      /* nombre de fils */
    struct Noeud *fils[NR];
    struct rectangle r[NR]; }

#define PdF (sizeof(Noeud)-sizeof(int))/sizeof(Point)

struct Feuille {
    int nb_points;      /* nombre de points */
    struct Point p[PdF]; }
```

Fenêtrage

```
fenêtre( struct Noeud *q, struct rectangle *r, int niv )
{
    if (niv == Prof)
        for (i= 0; i<(struct Feuille)q->nb_points; i++)
            if (inclusion( (struct Feuille)q->p[i], r))
                ...
    else
        for (i=0; i<q->nb_fils; i++)
            if (intersection( q->r[i], r ))
                fenêtre( q->fils[i], r, niv+1 )
}
```