synthèse d'image : algorithmes élémentaires

DUNOD
informatique

Monographies de l'AFCET-Informatique
Un regard de professionnels

- D. Le Verrand, **Le langage ADA, manuel d'évaluation**, Dunod, 1982.
  Evaluating ADA, North Oxford Academic Publishing Company, 1985.
- F. André, D. Herman et J.P. Verjus, **Synchronisation des programmes parallèles**, Dunod, 1983.
  Synchronization of parallel programs, North Oxford Academic Publishing Company, 1985.
- M. Raynal, **Algorithmique du parallélisme : le problème de l'exclusion mutuelle**, Dunod, 1984.
  Algorithms for mutual exclusion, North Oxford Academic Company, 1985.
- C. Chrisment, J. B. Crampes et G. Zurfluh, **Bases d'informations généralisées**, Dunod, 1985.
- G. Hégron, **Synthèse d'image : algorithmes élémentaires**, Dunod, 1985.

Ouvrage publié avec le concours des ministères de l'éducation nationale (direction de la recherche) et de la recherche et de la technologie. (MIDIST)

Programme mobilisateur : « promotion du français langue scientifique et diffusion de la culture scientifique et technique. »

ISBN 2-04-016427-8

PREFACE

A l'heure où tout un chacun peut admirer les images produites par ordinateur, en particulier pour leur degré de réalisme et de finition, on peut se poser la question de savoir s'il est réellement utile de consacrer un ouvrage complet à l'étude des techniques élémentaires pour la synthèse d'image. Et pourtant, l'explosion de la micro – informatique conduit des milliers de personnes à recréer sur leur ordinateur personnel des algorithmes de base. Et pourtant, les ingénieurs chargés de concevoir et réaliser les générateurs d'images synthétiques souhaitent connaître les meilleurs algorithmes pour pouvoir les intégrer dans leur machine.

Et c'est là tout le paradoxe : des images extraordinaires sont produites quotidiennement, à partir d'algorithmes qui ne sont pas forcément maîtrisés, ou tellement nombreux que l'on ne sait plus lequel choisir et pourquoi.

Le mérite de cet ouvrage est de passer en revue de manière systématique un certain nombre d'algorithmes plus ou moins célèbres, en mettant en valeur pour chacun ses avantages et ses inconvénients. Ce travail méticuleux et très austère a comme premier résultat de rassembler en un même endroit des techniques habituellement éparpillées dans des publications qui ne sont pas toujours facilement accessibles.

Le deuxième mérite est de fournir une bibliothèque d'outils raisonnés, dans trois domaines à la base de la synthèse d'image (même au delà des algorithmes élémentaires), à savoir la génération de courbes, le remplissage de taches et les traitements de nature géométrique. Ainsi, le lecteur trouvera rassemblés les fondements de tout système de synthèse d'image.

Enfin, par delà la diversité des algorithmes, l'auteur a su faire converger les différentes idées, en traitant plus particulièrement de deux techniques, qui relient entre eux quelques algorithmes, et peuvent donc servir de base systématique à un logiciel : les générations de courbes avec le mécanisme de Bresenham, et les traitements variés à partir du suivi de contour. Cet aspects synthétique à partir de méthodes a priori très diverses constitue un des apports les plus importants de cet ouvrage.

Je crois fermement que ce livre rendra beaucoup de services et que, par suite, il aura beaucoup d'influence sur la conception des logiciels graphiques. Je lui souhaite longue vie.

Michel Lucas
Professeur à l'Université de Nantes

SOMMAIRE

INTRODUCTION

L'apparition des terminaux de visualisation du type balayage télévision a ouvert une nouvelle voie de recherche très importante en infographie interactive : la synthèse d'image. Nous employons ici le terme image par opposition à dessin au trait (présentation graphique utilisant seulement des lignes comme élément de base). Les images permettent d'utiliser toutes les ressources de l'expression graphique en combinant aussi bien les représentations à base de traits que les représentations à base de surfaces coloriées (taches).

Devant la diversité des familles de procédés employés pour créer une image et la disparité des solutions dispersées à travers une littérature abondante, cet ouvrage fait le point sur les problèmes de production d'images en s'attachant au niveau élémentaire, c'est-à-dire celui des outils de base nécessaires à la gestion d'un écran.

Les trois grandes classes de problèmes abordés sont :

- la génération des courbes, (Chapitre 2)
- le remplissage de taches, (Chapitre 3)
- les algorithmes de découpage et les traitements élémentaires de nature géométrique (Chapitre 4).

Pour chacune d'elles, le choix des algorithmes présentés et développés résulte de notre "étude comparative d'algorithmes élémentaires pour la synthèse d'image" (voir <HEGRON 83c>) qui nous a permis de répondre à plusieurs objectifs

- rassembler et classifier les publications nombreuses et dispersées dans ce domaine (le volume de la bibliographie s'avère éloquente),
- sélectionner et présenter une bibliothèque d'algorithmes corrects,
- dégager un ensemble de procédés méthodologiques qui nous permettent de résoudre le plus grand nombre de problèmes, du cas particulier au cas le plus général possible.

Cet ouvrage veut tout d'abord mettre à la disposition des réalisateurs de systèmes graphiques des éléments de réalisation pratique (algorithmes élémentaires de visualisation). A cet égard, on remarquera que si les algorithmes développés sont de façon générale destinés à la synthèse

d'image, la plupart d'entre eux sont utilisables pour la production de dessins (génération de courbes, hachurage, problèmes de découpage appliqués aux contours et traitements de nature géométrique). Les algorithmes sont volontairement écrits dans un langage de description afin que l'utilisateur les programme dans le langage de son choix.

Le second objectif de ce manuscrit se veut plus didactique en offrant une classification des diverses familles de procédés utilisés, en présentant les problèmes et les techniques inhérentes à chacune d'elles, et en proposant des solutions originales, fruit d'un cheminement méthodologique et unitaire.

A cet effet, nous avons exhibé deux méthodes générales. La première est la technique de BRESENHAM pour la génération incrémentale des segments de droite et des coniques. La seconde permet de traiter la plupart des traitements élémentaires relatifs à une tache : il s'agit de la technique générale du suivi de contour.

Remerciements

Le contenu de cet ouvrage résulte de nos travaux effectués dans le cadre de notre Thèse de troisième cycle en informatique (voir <HEGRON 83c> sous la direction de Monsieur M. LUCAS, Professeur à l'Université de Nantes et Responsable de l'équipe "Traitements graphiques et CAO". Sa rédaction a pu être réalisée en partie grâce à un prolongement de deux mois de notre allocation de recherche DGRST délivrée par l'I.R.I.S.A. (Institut de Recherche en Informatique et Systèmes Aléatoires, Université de Rennes I).

Je tiens à exprimer ici toute ma reconnaissance à Monsieur LUCAS pour m'avoir donné l'opportunité de rédiger ce livre et pour en avoir amicalement composé la préface. Je remercie également Marie-Anne ROY et Jacques ANDRE qui nous ont permis, à la lecture de la version initiale, d'apporter les corrections et les améliorations nécessaires. Je remercie tout particulièrement Maryse FOUCHE pour avoir accompli avec compétence et beaucoup de patience la tâche dactylographique.

Chapitre 1

Généralités sur les traitements élémentaires en synthèse d'image

Pour mieux cerner ce que recouvre l'appellation "algorithmes élémentaires pour la synthèse d'images", nous allons préciser dans ce qui suit la notion de traitements dits "élémentaires" et leur champ d'application qu'est l'image.

1.1. Algorithmes et traitements élémentaires

Le schéma le plus général de la chaîne de programmes nécessaires à l'élaboration d'un logiciel graphique interactif comporte les éléments suivants (voir figure 1.1.) :

- un logiciel de description qui permet à partir des informations fournies par le programme d'application de coder la scène ;
- un logiciel de préparation à la visualisation qui compose à partir de la scène codée, une scène bidimensionnelle constituant le fichier graphique ;
- un logiciel élémentaire qui permet d'obtenir le dessin ou l'image finale à partir des indications fournies par le logiciel de préparation à la visualisation. Le logiciel élémentaire a pour rôle d'établir la liste de visualisation et d'assurer la gestion de l'écran.

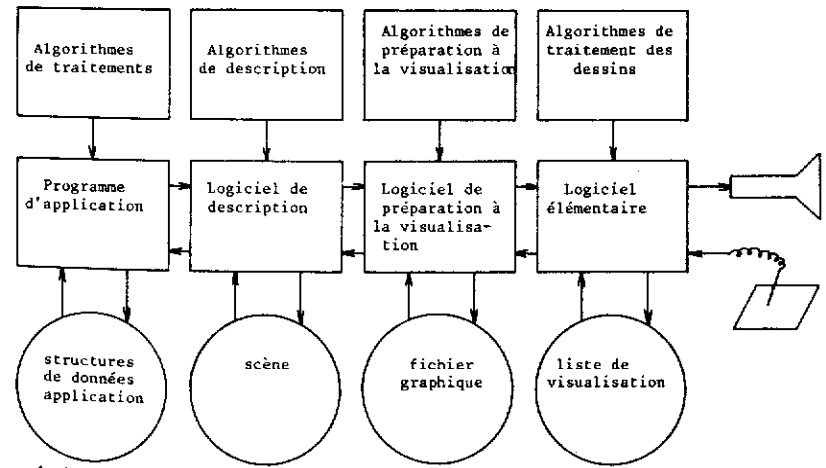


Figure 1.1.: Schéma général d'un logiciel graphique interactif (tiré de <LUCAS 77>).

L'ensemble des algorithmes et traitements élémentaires appartient au logiciel élémentaire. La structure à représenter et à afficher est donc une scène bidimensionnelle.

1.2. L'information image

L'image est un ensemble structuré d'informations qui après affichage sur l'écran ont une signification pour l'oeil humain.

De cette définition, se dégagent deux espaces distincts auxquels appartient l'image. Le premier correspond à l'espace utilisateur dans lequel nous trouvons les structures de l'objet (ou scène) à représenter. Le second demeure l'espace écran où se situent la structure graphique numérisée de la scène et l'affichage proprement dit de l'image sur l'écran (figure 1.2.).

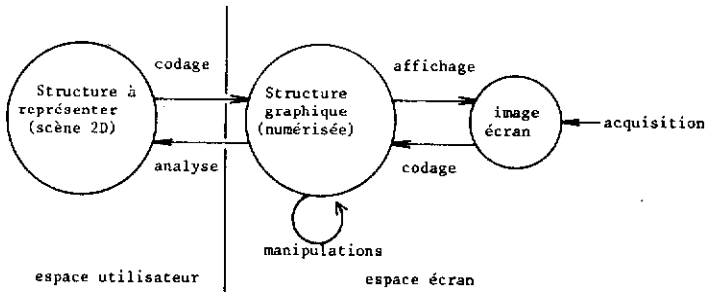


Figure 1.2.: Espaces "image" et traitements.

La surface de visualisation est un écran à pointillage constitué d'un quadrillage régulier définissant un système de coordonnées où tout couple (X,Y) désigne un carreau élémentaire ("pixel") (voir figure 1.3.).

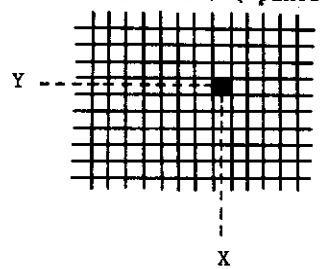


Figure 1.3.: Surface pour le dessin point à point.

Chaque point de cette surface de visualisation possède une couleur et une intensité. Les images sont alors constituées essentiellement de points ou de taches formées d'ensembles de points. Par opposition aux dessins (constitués de traits et de points), nous parlerons d'images (c'est également le terme technique désignant les dessins obtenus sur un écran de télévision, technologie dominante pour les surfaces à pointillage).

L'image permet de combiner aussi bien les techniques de représentation à base de courbes que les méthodes de représentations à base de surfaces

coloriées (taches). Le paramètre le plus important qui caractérise une tache est son contour, c'est-à-dire sa silhouette. En effet, l'affichage de l'ensemble des points qui constituent la tache est réalisé à partir de la description de son contour.

Dans ce qui suit lorsque nous décrivons le contour d'une tache par un ensemble de primitives d'affichage (segments de droite, arcs de cercle...) nous le modélisons de la manière suivante :

Un contour sera défini par un ensemble de parois appelées par extension "contours"constitués d'une suite continue et fermée de primitives (structure d'anneau). Dans le cas le plus général, nous supposons que les primitives d'un même contour peuvent se couper ou se chevaucher ainsi que les primitives des différents contours entre elles. On dira qu'un contour est polygonal s'il est formé d'un ensemble de contours qui sont des polygones. Chaque polygone sera décrit par la suite ordonnée de ses sommets.

Le mode de génération d'une tache, ou de façon plus générale d'une image résultant de la combinaison d'un ensemble de taches, dépend de la codification initiale des contours ou de la surface qu'ils délimitent.

1.3. Codification de l'image

Plusieurs critères conditionnent le choix du code qui décrira l'image:

- espace mémoire disponible,
- temps de rafraichissement de l'écran,
- vitesse de transmission de l'image,
- vitesse de génération de l'image.

En pratique, l'univers des codes pour les images est infini. Il faut cependant distinguer les codes décrivant l'image au niveau le plus bas lorsqu'elle est déjà discrétisée (espace écran), de ceux décrivant l'image indépendamment du support de visualisation (espace utilisateur) (voir figure 1.2.).

Codage dans l'espace écran :

. Codage de la surface :
- matrice binaire : la tache est codée point par point ('0' éteint, '1' allumé) dans une matrice binaire,
- codage par plages : pour chaque ligne de balayage, on mémorise les points d'intersection du contour avec la ligne ; les segments ainsi déterminés sont des plages.

. Codage du contour :
- code par plages différentiel : on note les variations des extrémités des plages d'une ligne à l'autre ;
- code incrémental : le contour de la tache est défini par un point de départ et une suite de mouvements élémentaires qui permettent de passer d'un point à un autre (voir code de Freeman, figure 1.4.).

Codage dans l'espace utilisateur :

. Les codes syntaxiques : les dessins sont considérés comme des graphes de R^2 ;

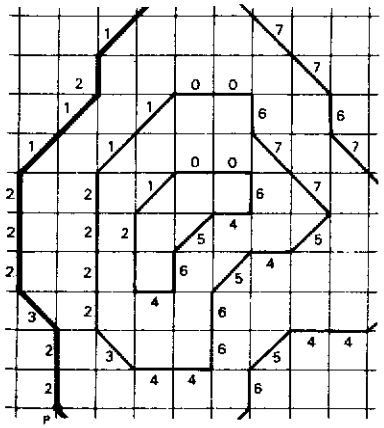
. Codage du contour : le contour est décrit par un ensemble de primitives, telles que les segments de droite, les arcs de conique ou les splines, mémorisées sous forme analytique ;

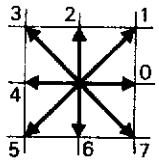
. Codage de la structure :
- primitives : la tache est décrite comme la juxtaposition de primitives globales (barres, courbes...) sur lesquelles on peut opérer certaines opérations (inclinaisons, symétries,...).
- squelettes : la tache est codée par son squelette et par une fonction qui en définit l'épaisseur en tout point.

L'un des critères important pour le choix d'un code est son pouvoir ou sa souplesse d'adaptation au mode de génération de l'image employé. Par exemple, le codage incrémental du contour peut s'adapter au prix de quelques aménagements aux terminaux à balayage télévision (voir <MERIAUX 79>). FRANKLIN, <FRANKLIN 79> a comparé onze algorithmes qui permettent d'afficher un dessin ligne par ligne, pour conclure que la méthode optimale est de diviser l'écran en bandes horizontales où chacune d'elles utilise toute la mémoire disponible, de découper toutes les droites et les courbes qui traversent une frontière, de trier les morceaux par bandes, et pour chaque bande d'engendrer chaque courbe point par point puis de l'afficher.

Nous nous sommes intéressés à deux modes de génération d'un dessin ou d'une image sur des surfaces à pointillage :

1. Le premier mode est apparenté aux techniques utilisées sur les surfaces pour le dessin au trait (les traceurs de courbes par exemple) : il s'agit de l'affichage point à point de manière incrémentale (Code de Freeman : voir figure 1.4.). Cette méthode est bien adaptée à la génération d'objets élémentaires comme les courbes (segments de droite, coniques,...).





principe du codage

P point de départ
Cet élément de dessin serait codé : 2 2 3 2 2 2 1 1 2 1.

codage d'un élément de dessin

Figure 1.4.: Code de Freeman : application.

2. Le second mode s'est développé avec l'apparition des terminaux du type balayage télévision (ou balayage récurrent) : c'est l'affichage ligne par ligne (voir figure 1.5.).

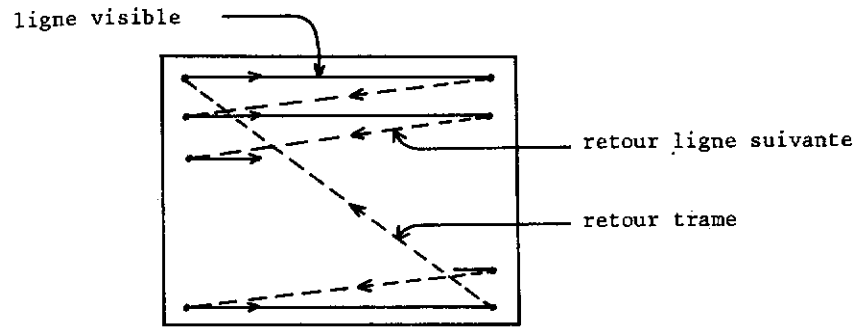


Figure 1.5.: Balayage récurrent.

Les avantages généraux du balayage ligne par ligne sont les suivants :

- il est bien adapté à la génération et à la production du dessin au trait ou de l'image ;
- il est également bien adapté à l'affichage et à la transmission de l'image pour la télévision parce qu'il n'y a pas de transformation de codes à effectuer contrairement aux techniques de codage incrémental par exemple.

Nous verrons dans le chapitre suivant que les diverses techniques de création d'une image combinent souvent ces deux modes de génération.

1.4. Les traitements en synthèse d'image

Pour réaliser les différents traitements, plusieurs familles de procédés existent, liées à la structure codée de l'image à partir de laquelle nous travaillons.

Nous envisageons plusieurs approches en définissant la tache, soit dans l'espace utilisateur, soit dans l'espace écran (voir figure 1.6.):

- espace utilisateur : la tache est définie par la description mathématique de son contour ;
- espace écran : nous définirons la tache par l'ensemble des points qui la constituent (nous manipulons alors des matrices de points), par l'ensemble des intervalles horizontaux qui sont les parties des lignes de balayage intérieures à la tache (nous manipulons dans ce cas une structure d'intervalles appelés aussi plages), par l'inscription point à point du contour dans la matrice image, ou bien par l'ensemble de ses hachures si la tache est hachurée (nous manipulons alors un ensemble de segments de droite).

Pour engendrer une tache sur l'écran, nous aborderons successivement les opérations suivantes (voir figure 1.6.):

- inscription du contour d'une tache point à point : c'est le problème général de la génération des courbes,
- génération d'une tache monochrome (remplissage proprement dit et coloriage) ou hachurée (hachurage).

Comme nous le remarquons sur la figure 1.6., nous pouvons réaliser certaines opérations en passant d'un espace à l'autre (flèches en pointillés).

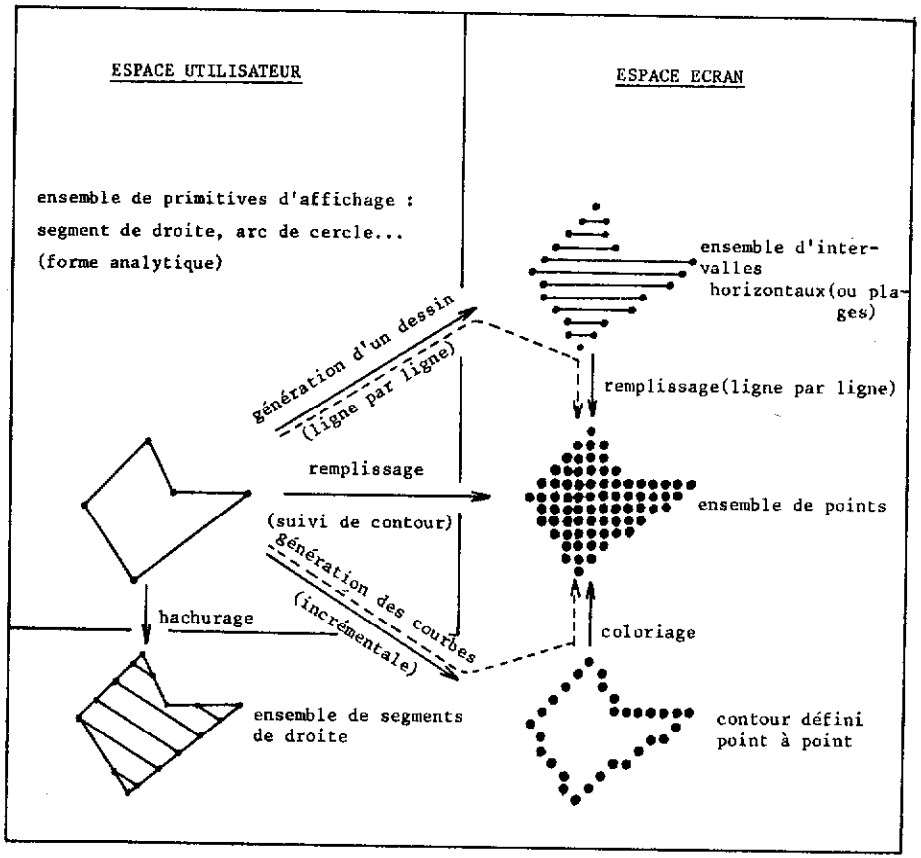


Figure 1.6.: Codifications d'une tache et traitements.

Mais la création d'une image résulte également de la combinaison d'un ensemble de taches. Nous nous intéresserons alors au découpage des taches entre elles et de leur contour. Nous verrons que les problèmes élémentaires de nature géométrique qui en résultent, sont très différents suivant la structure de la tache sur laquelle nous raisonnons (contour, ensemble de points ou d'intervalles horizontaux, etc...).

Parmi l'ensemble des procédés employés pour créer une image, l'un d'entre eux a retenu notre attention. Il permet de résoudre la plupart des traitements abordés ici à partir d'une description structurée du contour

d'une tache dans l'espace utilisateur et sans avoir besoin de nous ramener à une description de la tache dans l'espace écran : il s'agit du principe général de la technique du suivi de contour.

Nous n'explorerons pas les problèmes d'acquisition, ni les autres techniques (analyse, reconnaissance) qui sont plutôt du ressort de ce que l'on nomme le "traitement d'image" par opposition à "synthèse d'image". Les notions de traitement et de synthèse d'image, quoique distinctes dans leur définition, demeurent fortement complémentaires dans l'élaboration et la création d'images par ordinateur.

Chapitre 2

Génération de courbes sur une surface à pointillage

2.1. Introduction

En synthèse d'image, nous faisons très souvent appel aux algorithmes de génération de courbes sur une surface à pointillage et principalement dans les situations suivantes :

- génération d'un dessin (ensemble de courbes) sur une surface point par point ;

- génération du contour des taches définies par un ensemble de primitives d'affichage (segments de droite, arcs de conique,...);

- remplissage de taches : génération ligne par ligne du contour pour la méthode du suivi de contour, ou codage du contour pour la méthode du balayage par ligne (chapitre 3.).

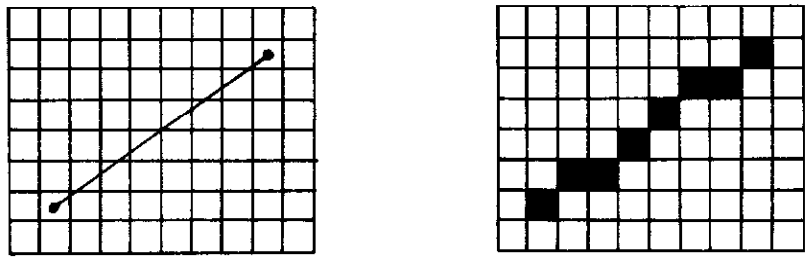
Le problème :

Le problème de l'inscription d'une courbe sur une surface à pointillage revient à trouver une approximation telle que l'oeil puisse reconstruire la courbe représentative. Nous passons d'une représentation continue à une représentation discrète de la courbe (quantification).

Par exemple, un segment de droite sera défini par la donnée des coordonnées de ses extrémités (figure 2.1.a). Le problème consiste à engendrer *la suite de points situés sur la grille de l'écran approchant le mieux* le segment initial (figure 2.1.b).

L'étude des techniques résolvant ce problème nous a permis de dégager deux grandes classes d'algorithmes :

- les méthodes "numériques",
- les méthodes "incrémentales".



a- La grille et les extrémités du segment b- Segment engendré

Figure 2.1.: Génération d'un segment de droite.

2.2. Classification générale

Nous développons ici les caractéristiques générales des deux grandes classes d'algorithmes qui engendrent les courbes ; c'est-à-dire des méthodes numériques et incrémentales.

2.2.1 - LES METHODES NUMERIQUES

Pour les méthodes numériques, les calculs sont basés sur l'analyse numérique de la courbe en utilisant de façon explicite son équation et ses dérivées partielles.

On recherche de façon générale les intersections de la courbe avec les lignes horizontales (y=n). On utilise généralement une structure répétitive qui calcule y(t) pour $t=m.\Delta t$ où $m = m_0, m_0+1, \ldots$ et pour chaque m tel que $y(m.\Delta t) \leq n < y(m.\Delta t + \Delta t)$ on enregistre $x(m.\Delta t)$. Si l'incrément Δt est fixé, le calcul de $y(m.\Delta t)$ et $x(m.\Delta t)$ se fera de manière récurrente.

Cette méthode présente divers handicaps. Premièrement, il est souvent impossible de fixer Δt pour ne pas franchir plus d'une seule ligne horizontale à chaque pas. Deuxièmement, la précision est affectée par les calculs récursifs qui engendrent une dégradation de la courbe. Pour pallier ce dernier phénomène MAXWELL et BAKER, <MAXWELL-BAKER 79>, proposent d'examiner et de tenir compte de toutes les sources potentielles d'erreurs.

Deux mises en oeuvre sont principalement utilisées :

- la technique de programmation qui repose sur la forme paramétrique de la courbe définie par les équations :

$x = x(t)$
$y = y(t)$

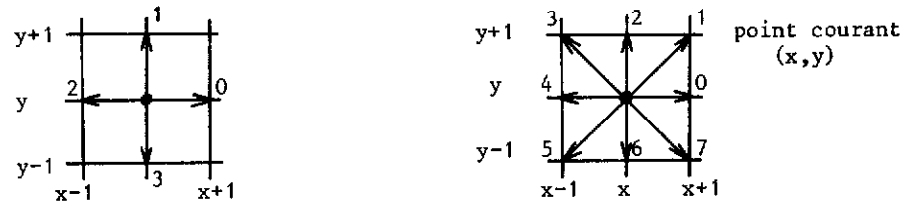
- la technique de l'Analyseur Différentiel Numérique (ADN) qui est un dispositif de calcul hybride dans lequel les techniques numériques et analogiques interviennent simultanément.

Les méthodes numériques utilisent une arithmétique réelle sans pouvoir éviter dans la plupart des cas l'emploi des opérations de multiplication et de division.

2.2.2 - LES METHODES INCREMENTALES

Le principe général des méthodes incrémentales est le suivant : on engendre la courbe d'un point origine jusqu'à un point final en calculant pas à pas à partir d'un point courant le mouvement élémentaire qui déterminera le point suivant.

En chaque point, on peut définir 4 mouvements (figure 2.2.a) ou 8 mouvements (figure 2.2.b) élémentaires possibles. C'est le principe du code de FREEMAN :



a- 4 mouvements b- 8 mouvements

Figure 2.2.: Code de Freeman.

A chaque pas, le calcul du mouvement élémentaire peut se faire à l'aide de méthodes implicites (on utilise la représentation non-paramétrique de la courbe définie par l'équation $f(x,y)=0$ où $x,y \in R$, arithmétiques (on emploie à la fois la représentation implicite de la courbe et les propriétés géométriques de la courbe et de la surface de visualisation) et à l'aide de propriétés structurelles.

Les méthodes incrémentales utilisent avant tout une arithmétique entière et évitent le plus souvent l'emploi des opérations de multiplication et de division.

2.2.3 - CONCLUSION

Dans cet ouvrage, nous nous intéressons essentiellement aux méthodes incrémentales car par rapport aux méthodes numériques, elles possèdent les avantages suivants :

- elles utilisent avant tout une arithmétique entière,

- elles évitent le plus souvent l'emploi des opérations de multiplication et de division,

- leur mise en oeuvre programmée assure leur indépendance par rapport au matériel (ce qui n'exclut pas le choix d'une implémentation câblée).

- d'autre part, elles s'apparentent aux techniques de "suivi de contour" car lors de la génération de la courbe nous "suivons" le tracé idéal d'une extrémité à l'autre. A cet effet, nous verrons de quelle manière elles s'insèrent dans les algorithmes de remplissage de taches.

2.3. Les méthodes incrémentales

2.3.1 -INTRODUCTION

A travers les trois classes d'algorithmes (implicites, arithmétiques et structurelles) qui constituent l'ensemble des méthodes incrémentales, deux familles d'algorithmes se dégagent :

- dans la première famille, nous trouvons trois méthodes incrémentales générales développées par DANIELSSON, par JORDAN et al. et par COHEN, qui raisonnent à partir de l'équation implicite de la courbe,

- dans la seconde famille, nous rencontrons une multitude d'algorithmes plus ou moins spécialisés et spécifiques à la génération de tel ou tel type de courbe (*segments de droite et coniques*).

2.3.2- LES METHODES INCREMENTALES GENERALES

2.3.2.1 - Généralités

Les méthodes incrémentales générales s'appliquent sur des fonctions C^∞ définies par leurs équations implicites $\{f(x,y)=0$ où $x, y \in R\}$.

Le calcul des courbes repose sur l'équation de la formule de Taylor

$$f(x+h,y+k)=f(x,y)+\sum_{n=1}^{\infty}\frac{1}{n!}\left(h\frac{d}{dx}+k\frac{d}{dy}\right)^n f(x,y)$$

Les variables h et k sont égales ici à ± 1 ou 0 et représentent les pas ou mouvements élémentaires effectués en x et en y. A ce sujet, DANIELSSON et COHEN restreignent les possibilités de déplacement à partir d'un point donné aux quatre mouvements axiaux alors que JORDAN inclut les quatre mouvements diagonaux.

Sachant que le point de départ appartient à la courbe et suivant les hypothèses initiales (mouvements élémentaires, sens de parcours de la courbe), le point suivant est calculé à chaque pas à partir de la valeur et du signe de la fonction et de ses dérivées premières au point courant. Le calcul et la mise à jour de ces équations sont réalisés grâce au développement de Taylor.

Critique :

L'étude de ces algorithmes (voir <HEGRON-ROY 82>) nous a permis de faire les constatations qui suivent.

. La qualité première des algorithmes de DANIELSSON, de JORDAN et de COHEN demeure le cadre théorique général dans lequel ils raisonnent ; ils peuvent donc s'appliquer à une grande famille de courbes.

. D'un point de vue pratique, ces méthodes générales nous permettent souvent d'employer uniquement l'addition et la soustraction et une arithmétique entière (en particulier pour les polynomes de degré deux et moins). Mais RAMOT et BELSER ont mis l'accent à l'aide de contre-exemples sur la faillibilité de ces algorithmes, même dans des cas très simples : ainsi l'adaptation de ces algorithmes à des fonctions particulières devra être toujours soigneusement testée avant leur mise en application. On examinera en particulier si la tangente locale ne variera pas trop à l'intérieur de la distance unité de la grille.

. D'autre part, l'algorithme de JORDAN produit des courbes plus "lisses" que les algorithmes de DANIELSSON et de COHEN car l'utilisation des 8 mouvements élémentaires atténue l'effet d'escalier. L'utilisation des 8 mouvements a aussi pour effet d'engendrer une courbe plus précise car les points calculés sont plus proches de la courbe idéale, et elle diminue aussi le nombre de pas car le nombre de points déterminés est inférieur à celui engendré par les techniques à 4 mouvements. A cet effet, nous développons ci-dessous la méthode de JORDAN.

2.3.2.2 - La méthode de JORDAN et al. {<JORDAN et al 73>}

Soit la courbe plane définie par son équation implicite :

$$f(x,y)=0 \qquad (1)$$

et ses dérivées supposées continues :

$$f_x=\frac{\partial f}{\partial x},\ f_y=\frac{\partial f}{\partial y}$$

$$f_{xx}=\frac{\partial^2 f}{\partial x^2},\ f_{xy}=\frac{\partial^2 f}{\partial y \partial x},\ f_{yy}=\frac{\partial^2 f}{\partial y^2} \qquad (2)$$

Nous supposons que le point de départ appartient à la courbe et que le pas est égal à 1.

La méthode proposée est de "suivre" la courbe aussi près que possible.

Supposons que nous soyons au point P=(x,y) (voir figure 2.3), nous avons alors 8 points voisins possibles : $(x\pm1,y),(x,y\pm1),(x\pm1,y\pm1)$

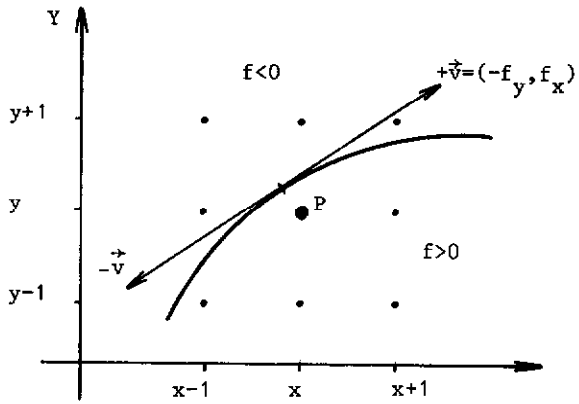


Figure 2.3.: La courbe, le point P et les 8 pas possibles.

Il existe deux sens pour parcourir la courbe : + $\vec{v}=(-f_y,f_x)$ ou $-\vec{v}$.

ΔX sera du signe opposé à f_y

ΔY sera du signe de f_x.

En choisissant la direction $+\vec{v}$ ou $-\vec{v}$ on réduit les 8 pas possibles à trois, ainsi :

$$\text{Soit } D = \begin{cases} 1 \text{ si on choisit } +\vec{v} \\ 0 \text{ si on choisit } -\vec{v} \end{cases} \qquad (3)$$

alors :

si ($f_y \geq 0$ et D=0) ou ($f_y < 0$ et D=1) alors $\Delta X = +1$;
si ($f_y < 0$ et D=0) ou ($f_y \geq 0$ et D=1) alors $\Delta X = -1$; (4)

si ($f_x < 0$ et D=0) ou ($f_x \geq 0$ et D=1) alors $\Delta Y = +1$;
si ($f_x \geq 0$ et D=0) ou ($f_x < 0$ et D=1) alors $\Delta Y = -1$; (5)

Puis nous choisissons le prochain point pour lequel $|f|$ est minimum :

soient $f^x = f(x+\Delta X, y)$ (6)

$f^y = f(x, y+\Delta y)$ (7)

$f^{xy} = f(x+\Delta X, y+\Delta y)$ (8)

ainsi :

si ($|f^x| < |f^y|$) et ($|f^x| < |f^{xy}|$) alors $\Delta Y := 0$; co pas en X fco
si ($|f^y| \leq |f^x|$) et ($|f^y| < |f^{xy}|$) alors $\Delta X := 0$; co pas en Y fco (9)
si ($|f^{xy}| \leq |f^x|$) et ($|f^{xy}| \leq |f^y|$) alors co pas en X et en Y fco

Les dérivées partielles sont mises à jour par les équations :

$$f_x = f_x + f_{xx}\Delta x + f_{xy}\Delta y + \frac{1}{2}(f_{x^3}(\Delta x)^2 + 2f_{x^2y}(\Delta x \Delta y) + f_{xy^2}(\Delta y)^2) + \ldots$$
$$f_y = f_y + f_{yy}\Delta y + f_{yx}\Delta x + \frac{1}{2}(f_{yx^2}(\Delta x)^2 + 2f_{y^2x}(\Delta x \Delta y) + f_{y^3}(\Delta y)^2) + \ldots \qquad (10)$$

Les variables Δx et Δx valant +1, -1 ou 0, le calcul des équations (10) ne peut se faire qu'avec des additions et des soustractions. De plus, ce calcul se simplifie pour des polynomes du second degré en X et en Y.

La mise à jour des variables f^x, f^y et f^{xy} dépend du pas que nous avons pris. Soit α=x pour un pas en X, α=y pour un pas en Y et α=xy pour un pas en X et Y.

On obtient alors :

$$f^x = f^\alpha + f_x \Delta x + \frac{1}{2} f_{xx} \Delta x^2 + \ldots$$
$$f^y = f^\alpha + f_y \Delta y + \frac{1}{2} f_{yy} \Delta y^2 + \ldots \qquad (11)$$
$$f^{xy} = f^\alpha + f_x \Delta x + f_y \Delta y + \frac{1}{2}(f_{xx} \Delta x^2 + 2f_{xy}\Delta x \Delta y + f_{yy} \Delta y^2) + \ldots$$

L'algorithme général a donc la forme suivante :

ALGORITHME DE JORDAN
Début
  obtenir premier point;
  afficher premier point;
  Initialisation - de f^α=0, des dérivées partielles;
                 - de la direction D(=0 ou 1);
  Tantque point courant non final faire
  Début
    Calcul de Δx et Δy (équations(4) et (5));
    Calcul de f^x, f^y, f^{xy} (équations (11));
    Détermination du pas et mise à jour de f^α(équations (9))
    Point suivant := point courant + pas;
    Afficher point suivant;
    Mise à jour des dérivées partielles (équations (10));
  fin
Fin

Pour les courbes où le dernier point à atteindre n'est pas nécessairement calculé par l'algorithme, le critère d'arrêt que nous avons retenu est le suivant:

Soient (x_f, y_f) les coordonnées du dernier point de la courbe.

Tout point de la courbe, engendrée par l'algorithme de JORDAN, se situe à ± 1 unité de la courbe théorique en x et en y et donc du point final (x_f, y_f) qui par hypothèse appartient à la courbe originale. Un critère adéquat serait d'arrêter lorsque le point courant est à ± 1 unité du point final, c'est-à-dire que ($|x-x_f| \leq 1$ et $|y-y_f| \leq 1$). (1)

Mais neuf points répondent à ce critère. Ceci peut être amélioré en examinant la pente de la tangente à la courbe au point final. Si la tangente se situe dans les 1er, 4ème, 5ème et 8ème octants, il y a plus de pas en x qu'en y ; ainsi une égalité en x et une inégalité en y suffiront. Le critère d'arrêt est alors :

$$(x=x_f) \text{ et } (|y-y_f| \leq 1) \qquad (2)$$

Pour les 2,3,6 et 7ème octants, le critère d'arrêt sera basé sur une égalité en y comme suit :

$$(|x-x_f| \leq 1) \text{ et } (y=y_f) \qquad (3)$$

Il y a seulement 3 points qui satisfont à ce critère. De plus, ce test est plus simple que le test (1) mais une pré-détermination de l'octant dans lequel le point final se trouve sera nécessaire pour décider si on emploie les tests (2) ou (3).

2.3.2.3 - *Conclusion*

Nous avons montré dans <HEGRON 83c> que l'adaptation des algorithmes généraux de DANIELSSON, JORDAN et COHEN n'est pas compétitive pour la génération des segments de droite et des coniques par rapport aux algorithmes spécifiques. Ceci est dû à la redondance des calculs effectués pour la mise à jour de f(x,y) et de ses dérivées et pour la comparaison des valeurs obtenues, et en particulier à un plus grand nombre de points engendrés en ce qui concerne les méthodes de DANIELSSON et de COHEN. De façon générale, ces techniques générales n'exploitent pas les propriétés inhérentes à chaque courbe.

2.3.3 - LES METHODES INCREMENTALES SPECIFIQUES

2.3.3.1 - *Généralités*

La Table 1 nous présente la plupart des algorithmes incrémentaux spécifiques à la génération des segments de droite et des coniques. L'étude comparative de ces algorithmes (voir <HEGRON 83c>) nous a montré que le principe de BRESENHAM réunissait les qualités suivantes :

- généralisation possible pour un grand nombre de courbes ;
- les courbes engendrées possèdent les meilleures qualités graphiques: précision, symétrie ;
- la séquence des points est obtenue grâce à une arithmétique entière et des opérations d'addition et de soustraction ;
- les algorithmes obtenus possèdent également la meilleure complexité pratique (évaluation du nombre d'opérations élémentaires).

2.3.3.2 - *Généralisation du principe de BRESENHAM*

Supposons que nous sommes arrivés au point de coordonnées entières (r,q) le plus proche de la courbe idéale (voir figure 2.4.). Pour calculer le point suivant, nous limiterons notre choix parmi deux mouvements élémentaires ou trois en découpant respectivement l'espace autour du point courant en huit octants ou en quatre quadrants. Ces derniers sont déterminés grâce aux valeurs (pente et direction) des tangentes en ce point.

METHODES \ COURBES	Segments de droite	Cercles	Ellipses	Paraboles	Hyperboles
Implicite	LUCAS				
	HORN →	HORN → DOROS			
	BRESENHAM → STOCKTON	BRESENHAM →	PITTEWAY → ROY	ROY →	HEGRON →
Arithmétique	EARNSHAW				
	SUENAGA et al. →	SUENAGA et al. →			
Structurelle	CEDERBERG				

Notations : . -> filiation des algorithmes
. Les algorithmes dont le nom est souligné (exemple: BRESENHAM) renferment les principes de base utilisés par les algorithmes qui leur sont affiliés.

TABLE I : Tableau récapitulatif des algorithmes incrémentaux spécifiques à la génération de segments de droite et des coniques.

Nous avons constaté que le découpage en octants (deux mouvements) était plus avantageux qu'une partition du plan en quadrants (trois mouvements). Dans ce qui suit nous ferons donc notre analyse au niveau de l'octant, et en particulier dans le premier, le raisonnement demeurant identique pour les sept autres.

Supposons que nous nous déplaçons dans le premier octant (voir figure 2.4.).

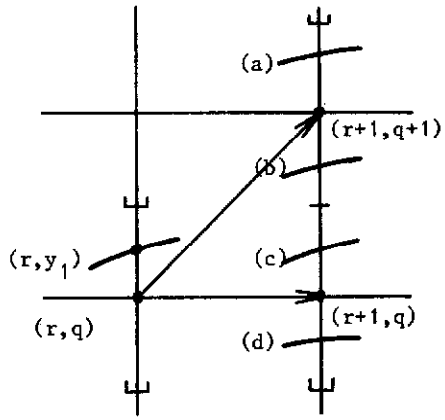


Figure 2.4.: Positions relatives des points candidats par rapport à la courbe idéale (premier octant).

Soit $P_1(r,y_1)$ le point de la courbe idéale d'abscisse r, on a $y_1 \in [q-\frac{1}{2},\ q+\frac{1}{2}[$.

Soit D(r,q) une expression arithmétique qui nous permette d'évaluer l'écart y_1-q.

Soit $P_2(r+1,y_2)$ le point suivant de la courbe à afficher. La pente de la tangente à la courbe appartenant à $[0,1[$ on aura :

$$q + \frac{3}{2} > y_2 \geq q - \frac{1}{2}$$

Les deux candidats possibles pour l'étape suivante seront donc : (voir figure 2.4.)

- soit le point (r+1,q+1) correspondant au mouvement diagonal ;
- soit le point (r+1,q) correspondant au mouvement axial.

Le point P_2 peut vérifier l'un des 4 cas de figures suivants :

a) $q + \frac{3}{2} > y_2 \geq q+1$ — mouvement diagonal

b) $q + 1 > y_2 \geq q+\frac{1}{2}$

c) $q + \frac{1}{2} > y_2 \geq q$ — mouvement axial

d) $q > y_2 \geq q-\frac{1}{2}$

Le mouvement suivant sera déterminé grâce aux expressions D(r+1,q) et D(r+1,q+1) qui évaluent respectivement les différences (y_2-q) et $(y_2-(q+1))$ identifiant ainsi le point de l'écran le plus proche de la courbe idéale.

Pour chacune des courbes étudiées, le problème est de trouver à partir de son équation implicite l'expression D(r,q) et à partir de celle-ci l'expression E qui permettra de calculer le mouvement élémentaire suivant, en utilisant une arithmétique entière et les opérations d'addition et de soustraction uniquement.

De façon générale, l'expression E est donnée par l'équation E=D(r+1,q)+D(r+1,q+1), et le point suivant par la séquence : si E≤0 alors mouvement axial sinon mouvement diagonal.

2.3.3.3 - *Conclusion*

Dans ce qui suit, nous redonnerons les algorithmes préconisés par BRESENHAM pour la génération des segments de droite, puis nous appliquerons la généralisation de sa méthode au calcul des ellipses, des paraboles et des hyperboles. Pour ces trois coniques, nous distinguons le cas "simple" où les axes de symétrie de la courbe sont parallèles aux axes du repère, du cas plus général où les axes de symétrie ont subi une rotation.

2.4. Génération des segments de droite

Le segment de droite est la primitive graphique la plus couramment utilisée pour la génération de dessins ou d'images. Un segment est caractérisé par les coordonnées de ses extrémités que nous noterons (x_d,y_d) pour le début et (x_f,y_f) pour la fin.

Deux algorithmes s'avèrent particulièrement intéressants, celui de LUCAS et celui de BRESENHAM.

2.4.1 - *L'ALGORITHME DE LUCAS*

Pour engendrer un segment de droite, LUCAS(<LUCAS 77>) se ramène au premier octant. Comme nous nous déplaçons systématiquement à chaque pas suivant l'axe des x, nous évaluons l'erreur commise sur les ordonnées si on ne modifie pas y. A chaque étape, l'erreur ainsi commise se cumule. Pour la minimiser, il suffit de modifier y (en ajoutant 1) lorsque cette erreur atteindra ou dépassera l'unité.

Sous sa forme générale, l'algorithme s'écrit :

```
Génération SEGMENT-1 (xd,yd,xf,yf : entier)
Début co méthode de LUCAS fco
     entier : ΔX,ΔY, XINCR,YINCR,CUMUL,X,Y;
     x := xd, y := yd;
     afficher point (x,y);
     XINCR := si xd < xf alors +1 sinon -1;
     XINCR := si yd < yf alors +1 sinon -1;
     ΔX    := abs(xd-xf), ΔY := abs(yd-yf);
      si ΔX > ΔY co on dessine le maximum de points fco
         alors CUMUL := X/2;
               Pour I := 1 jusqu'à ΔX faire
                    X := X + XINCR;
                    CUMUL := CUMUL + ΔY;
                    si CUMUL ≥ ΔX
                       alors CUMUL := CUMUL - ΔX;
                             Y := Y + YINCR;
                    fsi
                    afficher point (x,y);
               finpour
         sinon CUMUL := ΔY/2;
               Pour I := 1 jusqu'à ΔY faire
                    Y := Y + YINCR;
                    CUMUL := CUMUL + ΔX;
                    Si CUMUL ≥ ΔY
                       alors CUMUL := CUMUL - ΔY;
                             X := X + XINCR;
                    fsi
                    afficher point (x,y);
               finpour
     fsi
Fin
```

2.4.2 - L'ALGORITHME DE BRESENHAM

Pour la génération des segments de droite, BRESENHAM (<BRESENHAM 65>), détermine à chaque pas le point le plus proche du segment idéal. Pour ce faire, il évalue de façon récurrente la différence (S_1-S_2) (voir figure 2.5) en se situant dans le premier octant. Si $(S_1-S_2)<0$, on effectue un mouvement axial, sinon on effectue un mouvement diagonal.

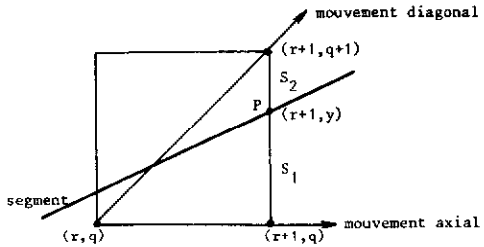


Figure 2.5.: Prochain mouvement (algorithme de BRESENHAM).

Sous sa forme générale, l'algorithme s'écrit :

```
Génération SEGMENT-2 (xd,yd,xf,yf : entier)
Début co méthode de BRESENHAM fco
     entier : DX,DY,XINC,YINC,S,DX2,DY2,DXY,X,Y ;
     DX2 := DX + DX ; DY2 := DY + DY;
     XINC := si xd < xf alors +1 sinon -1 ; DX := abs(xd-xf);
     YINC := si yd < yf alors +1 sinon -1 ; DY := abs(yd-yf);
     X := xd ; Y := yd ; afficher point (x,y);
     si DX > DY
        alors co x change plus vite que y fco
              S := DY2 - DX ; DXY := DY2 - DX2;
              Pour I := 1 jusqu'à DX faire
                   si S≥0
                      alors Y := Y+YINC;
                            S := S+(DXY);
                      sinon S := S+(DY2);
                   fsi
                   X := X+XINC
                   afficher point (x,y);
              finpour
        sinon co y change plus vite que x fco
              S := DX2 - DY ; DXY := DX2 - DY2;
              Pour I := 1 jusqu'à DY faire
                   si S≥0
                      alors X := X + XINC;
                            S := S + (DXY);
                      sinon S := S + (DX2);
                   fsi
                   Y := Y + YINC;
                   afficher point (x,y);
              finpour
     fsi
Fin
```

2.4.3 - GENERATION D'UN ENSEMBLE DE SEGMENTS DE DROITE

Dans le cadre de la génération d'un ensemble de segments droite, pour opérer un gain de temps ou de mémoire diverses idées peuvent être exploitées :

- l'algorithme de LUCAS est plus compétitif que celui de BRESENHAM sur un ensemble de segments de droite où la majorité d'entre eux sont verticaux, horizontaux,

- nous pouvons également utiliser les techniques de compression et de répétition.

2.4.3.1 - Compression

Les algorithmes de génération de segments de droites étudiés précédemment ne tiennent pas compte du fait que la séquence des mouvements élémentaires calculée puisse contenir des monotonies, c'est-à-dire des sous-suites composées d'un mouvement unique : c'est le cas pour des segments horizontaux, verticaux ou diagonaux.

La compression de ces monotonies est intéressante lorsque le problème de place mémoire intervient : le gain est appréciable mais varie en fonction de la nature de la scène.

Par ailleurs, la longueur à partir de laquelle il est préférable de compresser une monotonie dépend évidemment des caractéristiques de représentation des nombres propres à chaque matériel.

Enfin, il ne faut pas que le temps nécessaire à la compression au moment de la génération de la séquence et à la décomposition au moment de l'affichage soit trop important ; sinon les performances globales du tracé en seraient affectées.

2.4.3.2 - Répétition

Dans la génération des segments de droite, la séquence des mouvements élémentaires calculés peut comporter des sous-séquences qui se répètent un nombre fixe de fois.

En effet, plaçons-nous dans le premier quadrant, la généralisation aux autres quadrants est immédiate.

Soient ΔX et ΔY les accroissements respectifs en X et en Y. Les cas qui peuvent se présenter sont :

1er cas :

segment vertical ou horizontal,
i.e. $\Delta X=0$ ou $\Delta Y=0$
On a alors une répétition du mouvement axial.

2ème cas :

segment diagonal,
i.e. $\Delta X=\Delta Y$
On a alors une répétition du mouvement diagonal.

3ème cas :

$\Delta X=1$ ou $\Delta Y=1$
Prenons par exemple $\Delta Y=1$
Le mouvement diagonal aura lieu au milieu du tracé. Soit N1=partie entière $((\Delta X-1)/2)$;

on aura alors :
si $(\Delta X-1)/2$ est entier
  alors on a N1 mouvements axiaux
            1 mouvement diagonal
           N1 mouvements axiaux
  sinon on a N1 mouvements axiaux
            1 mouvement diagonal
         N1+1 mouvements axiaux
       ou
         N1+1 mouvements axiaux
            1 mouvement diagonal
           N1 mouvements axiaux.

4ème cas : $\Delta Y>1$ et $\Delta X/\Delta Y$ est entier ($\neq 1$)
On est ramené au cas précédent, mais la séquence de base est répétée ΔY fois.

5ème cas : $\Delta Y>1$ et $\Delta X/\Delta Y$ est non entier
On calcule le plus grand commun diviseur de ΔX et ΔY.
Ce PGCD fournira le facteur de répétition. La séquence de base sera calculée par un des algorithmes de génération de segment.

Pour chaque cas plus les segments sont longs et plus les facteurs de répétitions sont importants, plus le gain d'opérations, donc de temps est important.

L'utilisation des quatre premiers cas devient très vite intéressante car les pourcentages minimaux exigés des segments appartenant à chacun des quatre cas peuvent être facilement atteints. Par contre, celle du cinquième cas demeure beaucoup plus aléatoire. D'autre part, l'utilisation de la division dans les quatrième et cinquième cas demeure un inconvénient.

2.5. Génération des cercles

Le cercle est caractérisé par les coordonnées de son centre et par son rayon R.

2.5.1 - L'ALGORITHME DE BRESENHAM

BRESENHAM (<BRESENHAM 77>) cherche comme pour son algorithme de génération des segments de droite, à engendrer les points les plus proches possibles du cercle initial en n'utilisant que des valeurs entières, l'addition et la soustraction.

L'analyse se fait dans le 1er quadrant à partir du point (0,R) jusqu'au point (R,0) ; le centre du cercle étant ramené à l'origine.

A partir d'un point donné, trois mouvements élémentaires peuvent être envisagés m_1, m_2, m_3 : le mouvement est déterminé en évaluant de manière récurrente la ^{2}distance minimale qui existe entre le cercle idéal et celui passant par les trois points candidats (voir figure 2.6.).

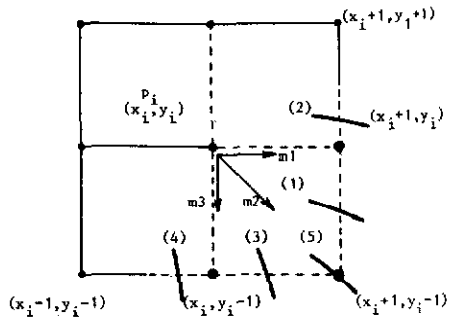
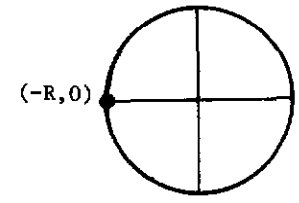


Figure 2.6.: Positions relatives des candidats possibles par rapport au cercle initial (méthode de BRESENHAM).

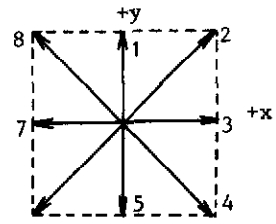
L'algorithme de BRESENHAM qui suit, calcule le cercle de centre (x_c,y_c) et de rayon R.

```
Génération CERCLE (xc,yc,R : entier)
Début co méthode de BRESENHAM fco
    M3 := 1;
    X1 := -R+xc, Y1 := yc co point courant fco
    Répéter
        X := 1, Y := R + R - 1;
        Δ := Y-1;
        M1 := M3, M2 := M1+1, M3 := M2+1;
        si M3 = 9 alors M3 := 1 fsi
        tantque Y≥0 faire
        Début
            ∂:= Δ + Δ;
            si ∂<0
                alors ∂ := ∂ +X ;
                      si ∂<0 alors Y := Y-2, Δ:=Δ + Y;
                                   MOVE <- M3;
                              sinon X := X+2, Y:=Y-2;
                                   Δ := Δ+Y-X;
                                   MOVE <- M2;
                      fsi
                sinon ∂ := ∂-Y;
                      si ∂≥0 alors X := X+2, Δ:=Δ-X;
                                   MOVE <- M1;
                              sinon X := X+2, Y:=Y-2;
                                   Δ := Δ+Y-X;
                                   MOVE <- M2;
                      fsi
            fsi
        fin
    jusqu'à M3 := 1
FIN
```

Dans cet algorithme, nous partons du point (-R,0) et progressons dans le sens des aiguilles d'une montre (figure 2.7.a). Nous utilisons la numérotation des mouvements de la figure 2.7.b :

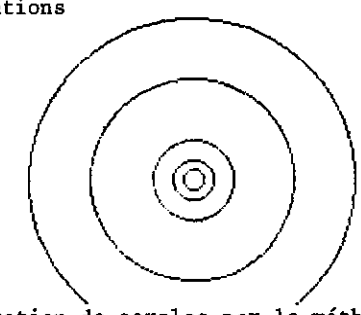


a) Sens de parcours du cercle



b) Numérotation des mouvements élémentaires.

Figure 2.7.: Conventions



Exemple : génération de cercles par la méthode de BRESENHAM.

2.5.2 - GENERATION DES ARCS DE CERCLES

2.5.2.1 - Le problème

. Le problème est de dessiner l'arc d'un cercle de centre (x_c,y_c) et de rayon R où l'arc est déterminé par ses coordonnées début et fin.

Considérons la figure 2.8. où (x_o,y_o) est le point de départ de l'arc. Nous avons deux façons d'atteindre le point final (x_1,y_1) : soit en parcourant le cercle dans le sens des aiguilles d'une montre (D=0), soit dans le sens inverse (D=1).

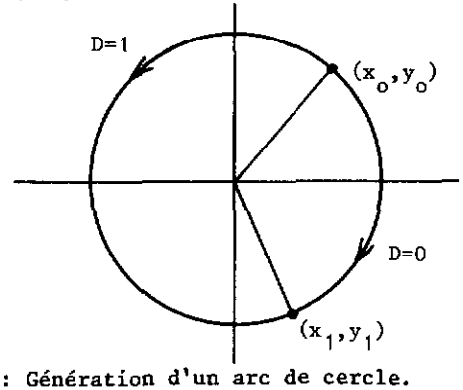


Figure 2.8.: Génération d'un arc de cercle.

. Dans la pratique, l'arc de cercle peut être défini de deux façons différentes : par trois points ou par deux points et les tangentes en ces points. Il faut alors calculer le rayon R et les coordonnées du centre (x_c,y_c) :

- cercle passant par trois points $(x_i,y_i)_{i=1,2,3}$:

A partir de l'équation du cercle $(x-x_c)^2+(y-y_c)^2-R^2=0$ et en considérant le nouveau repère d'origine (x_1,y_1), on obtient un système de deux équations à deux inconnues

$$(X_i^2+Y_i^2)-2\,X_i\,X_c - 2\,Y_i\,Y_c = 0,\ i=2,3$$

où $X_i = x_i - x_1$, $Y_i = y_i-y_1$, $i=2,3$

$X_c = x_c - x_1$, $Y_c = y_c-y_1$

avec $R^2 = X_c^2 + Y_c^2$.

- cercle défini par deux points A et B et les tangentes en ces points de pentes respectives Ta et Tb.

Le centre est obtenu en calculant le point d'intersection des deux normales aux tangentes données par les équations :

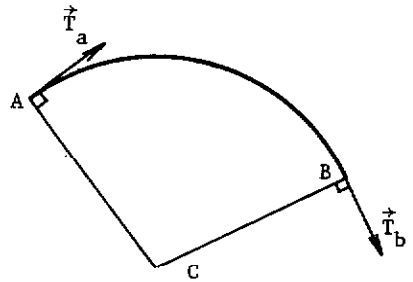
normale en A $Y=-(1/Ta).X+Y_a$ où $Y_a=YA+(1/Ta)*XA$

normale en B $Y=-(1/Tb).X+Yb$ où $Y_b=YB+(1/Tb)*YB$

Les coordonnées du centre sont données par les égalités qui suivent.

$$X_c = \frac{(Yb-Ya)(Ta*Tb)}{Ta-Tb}$$

$$Y_c = \frac{Ya.Ta - Yb.Tb}{Ta-Tb}$$



A=(XA,YA)
B=(XB,YB)

2.5.2.2 - *L'algorithme de BRESENHAM*

BRESENHAM (<BRESENHAM 77>) adapte son algorithme traçant des cercles, à la génération d'arcs de cercle ayant un rayon et les coordonnées du centre non entiers.

Le cercle général sera $(x-a)^2+(y-b)^2=R^2$ avec le point de départ (x_s,y_s) et le point final (x_t,y_t) appartenant à la circonférence. La direction de rotation sera le sens des aiguilles d'une montre (D=1) ou l'inverse (D=-1).

Il utilise les notations suivantes :

x_i : c'est la valeur de l'abscisse translatée au $i^{ème}$ pas dans le premier quadrant suivant le sens D=1

y_i : même chose pour l'ordonnée

Q_j : nombre de quadrants restant à traverser

M_1,M_2,M_3 : mouvements suivant les angles 0°, 315° et 270° dans le 1er quadrant.

Δ_i : valeur de $\{[(x_i+1)^2+(y_i-1)]-R^2\}$; le signe de Δ_i indique si le point (x_i+1,y_i-1) est à l'intérieur ou à l'extérieur du cercle initial.

ϑ : valeur de $\{[(x_i+1)^2+y_i^2]-R^2\}+\Delta_i$; le signe de ϑ indique lequel des deux points (x_i+1,y) ou (x_i+1,y_i-1) est le plus proche du cercle.

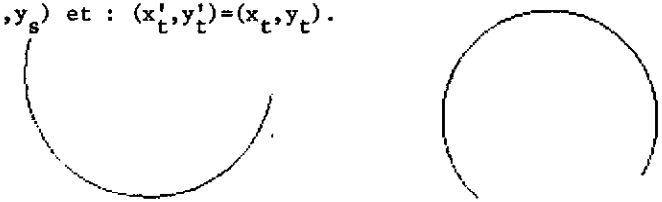
ϑ' : valeur de $\{[(x_i^2+(y_i-1)^2]-R^2\}+\Delta_i$; le signe de ϑ' indique lequel des deux points (x_i,y_i-1) ou (x_i+1,y_i-1) est le plus près du cercle.

Quadrant	Rotation	$\hat{X}$	$\hat{Y}$	Index	X	Y	q	M1	M2	M3
III	CCW	<0	<0	0	$\hat{Y}$	$\hat{X}$	3	0,-1	1,-1	1,0
II	CCW (D=-1)	<0	≥0	1	$\hat{X}$	$\hat{Y}$	2	-1,0	-1,-1	0,-1
IV	CCW (D=-1)	≥0	<0	2	$\hat{X}$	$\hat{Y}$	0	1,0	1,1	0,1
I	CCW	≥0	≥0	3	$\hat{Y}$	$\hat{X}$	1	0,1	-1,1	-1,0
III	CW	<0	<0	4	$\hat{X}$	$\hat{Y}$	2	-1,0	-1,1	0,1
II	CW (D=1)	<0	≥0	5	$\hat{Y}$	$\hat{X}$	3	0,1	1,1	1,0
IV	CW (D=1)	≥0	<0	6	$\hat{Y}$	$\hat{X}$	1	0,-1	-1,-1	-1,0
I	CW	≥0	≥0	7	$\hat{X}$	$\hat{Y}$	0	1,0	1,-1	0,-1

Table 2 : Table de transformation des paramètres.

Si les coordonnées (x_s,y_s) et (x_t,y_t), et le rayon R sont des valeurs entières, l'initialisation de (x'_s,y'_s) et (x'_t,y'_t) se réduit à :

$(x'_s,y'_s)=(x_s,y_s)$ et : $(x'_t,y'_t)=(x_t,y_t)$.



Exemples : Génération d'arcs de cercle par la méthode de BRESENHAM.

Génération ARC-CERCLE ($x_\delta, y_\delta, x_t, y_t$=réel, D:entier)

Début co méthode de BRESENHAM ∫co
co Initialisation ∫co
1- Déterminer les points de départ et d'arrivée entiers (x'_δ, y'_δ) et (x'_t, y'_t) les plus proches du cercle théorique à partir de (x_δ, y_δ) (x_t, y_t) en trouvant les "carrés-unités" respectifs c_δ et c_t qui les entourent grâce à l'expression :

$c(x',y') = \{(\lfloor x \rfloor, \lfloor y \rfloor), (\lfloor x \rfloor, \lceil y \rceil), (\lceil x \rceil, \lceil y \rceil)), (\lceil x \rceil, \lfloor y \rfloor)$
et en trouvant le point appartenant à c_δ (respectivement c_t) qui minimise : $|\lfloor (x'-a)^2 + (y^2-b)^2 \rfloor - R^2|$.

2- Translation du centre du cercle en (0,0)
$\bar{x}_\delta := x'_\delta - a, \bar{y}_\delta := y'_\delta - b$
$\bar{x}_t := x'_t - a, \bar{y}_t := y'_t - b$

3- Normalisation à partir de $(\bar{x}_\delta, \bar{y}_\delta)$ et $(\bar{x}_t, \bar{y}_t)$: on se ramène au premier quadrant et la direction D=1 pour déterminer :
$X_\delta = X_0$, $Y_\delta = Y_0, q_\delta$
$M1_\delta = M1_0$, $M2_\delta = M2_0$, $M3_\delta = M3_0$
X_t, Y_t, q_t
à l'aide de la table 2.

4- Calcul du nombre de quadrants traversés à partir de :
$Q^* := 4|(q_t - q_\delta)$ (c'est-à-dire $q_t - q_\delta$ modulo 4)
Si $Q^*=0$ et $X_t \leq Y_\delta$ et $Y_t \geq Y_\delta$
alors $Q_0 = 3$
sinon $Q_0 = Q^*-1$;

5- Calculer la valeur initiale de Δ_0
$\Delta_0 = (X_\delta+1)^2 + (Y_\delta-1)^2 - (x_\delta-a)^2 + (y_\delta-b)^2$
$= (x'_\delta-a)^2 + (y'_\delta-b)^2 - (x_\delta-a)^2 + (y_\delta-b)^2 + 2(X_\delta - Y_\delta + 1)$;

6- Sens de rotation :
D=1 ou -1

7- Premier point
$X := x'_\delta$;
$Y := y'_\delta$;

8- On a :
$X_i := X_0$; $Y_i := Y_0$;
$\Delta_i := \Delta_0$;
$Q_j := Q_0$;


```
co génération des points ∫co
    Fin := ∫aux;
    Tant que non(Fin) ∫faire
    Début
        Si Yi>0
          alors co calcul du mouvement ∫co
          Si Δi≤0
             alors ∂ := 2 Δi + 2 Yi - 1;
                   si ∂≤0 alors mouvement M1;
                          sinon mouvement M2 ∫si
             sinon ∂' := 2 Δi - 2 Xi - 1;
                   si ∂'≤0 alors mouvement M2;
                          sinon mouvement M3 ∫si
          ∫si
          si mouvement=M1
             alors Xi+1 := Xi + 1, Yi+1 := Yi; Δi+1 := Δi + 2 Xi+1 + 1;
             sinon si = M2
                   alors Xi+1 := Xi+1, Yi+1 := Yi - 1; Δi+1 := Δi + 2Xi+1 - 2Yi+1 + 2;
                   sinon Xi+1 := Xi, Yi+1 := Yi - 1, Δi+1 := Δi - 2Yi+1 + 1 co M3 ∫co ∫si
          ∫si
          sinon co réinitialisation ∫co
          X0 := -Yi; Y0 := Xi;
          Δ0 := Δi - 4 Xi;
          Qj+1 := Qj - 1;
          M1j+1 := M3j ; M2j+1 := D(M2(2)j, -M2(1)j);
          M3j+1 := D(M3(2)j, -M3(1)j).
        ∫si
        co Mise à jour de Fin ∫co
        si Qj<0
           alors si (Xt≤Xi) et (Yt≥Yi)
                 alors Fin := vrai;
        ∫si
    ∫tq
Fin
```

2.6. Génération des ellipses

2.6.1 - ELLIPSES SIMPLES (Algorithme de ROY)

Soit l'ellipse simple centrée en c(Xc,Yc) et d'équation $\frac{(X-Xc)^2}{a^2} + \frac{(Y-Yc)^2}{b^2} = 1$ (voir figure 2.9) où a,b∈N*
Xc,Yc∈Z.

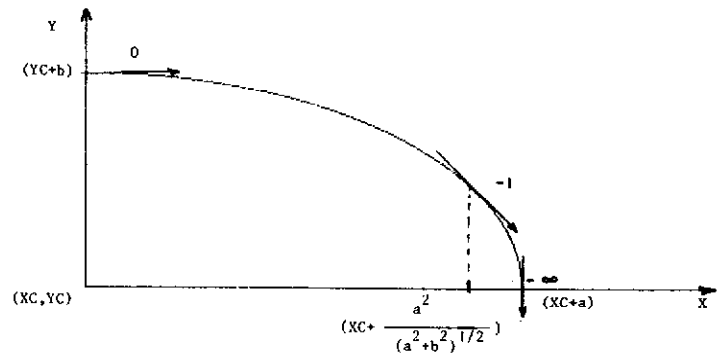


Figure 2.9.: Génération d'une ellipse simple (Méthode de ROY) (premier quadrant).

L'algorithme de génération de l'ellipse simple développé par ROY a la forme suivante :

```
Génération ELLIPSE SIMPLE (XC,YC,A,B:entier)
Début co méthode de ROY fco
     entier : A2,B2,DS,DT,DXT
              Xinc,Yinc,X,Y
              T,S,E,CA,CD

     A2 := A * A
     B2 := B * B
     DS := 4 * A2
     DT := 4 * B2
    DXT := ARRONDI (A2/SQR(A2+B2))

    Xinc:= +1
    Yinc:= -1
    TRACER-QUADRANT

    Yinc:= +1
    TRACER-QUADRANT

    Xinc:= -1
    TRACER-QUADRANT

    Yinc:= -1
    TRACER-QUADRANT

Fin
```



```
Procédure TRACER-QUADRANT
    T := 0
    S := -4 * A2 * B
    E := -S/2 - 2 * B2 - A2
   CA := -6 * B2
   CD := CA - 4 * A2

   X := XC
   Y := YC - Yinc * B
   Afficher (X,Y)

   Pour I := 1 pas 1 jusqu'à DXT faire
   Début
       X := X + Xinc
       Si E>=0
          alors début
                   E := E + T + CA
                   T := T - DT
                fin
          sinon début
                   Y := Y + Yinc
                   E := E + T - S + CD
                   T := T - DT
                   S := S + DS
                fin
       afficher (X,Y)
   fin
     DYT := ABS(Y-YC)
     E := E - T/2 - S/2 - B2 - A2 /* T et S pairs */
     CA := -6 * A2
     CD := CA - 4 * B2

     Pour I := 1 pas 1 jusqu'à DYT faire
     Début
         Y := Y + Yinc
         si E<=0
            alors début
                     E := E - S + CA
                     S := S + DS
                  fin
            sinon début
                     E := E - S + T + CD
                     S := S + DS
                     T := T - DT
                     X := X + Xinc
                  fin
         afficher (X,Y)
     fin
Fin procédure
```

2.6.2 - ELLIPSES QUELCONQUES (algorithme de ROY)

Lorsque les axes de l'ellipse sont parallèles aux axes du repère, l'équation de la courbe est donnée par :

$$b^2(X_o-X_c)^2+a^2(Y_o-Y_c)^2=a^2b^2$$

où $a,b \in N^*$
$X_c, Y_c \in Z$

Lorsque l'on effectue une rotation d'un angle avec $\theta \in]-\frac{\pi}{2}, \frac{\pi}{2}[$, nous obtenons l'équation :

$$(p^2b^2+m^2a^2)(X-X_c)^2+(b^2m^2+a^2p^2)(Y-Y_c)^2 +2(b^2-a^2)p\,m\,(X-X_c)(Y-Y_c) = a^2\,b^2\,(p^2+m^2) \quad (1)$$

avec $tg\theta = \frac{m}{p}$, $m \in Z$ et $p \in Z^*$

Nous posons :
$\alpha = p^2\,b^2 + a^2\,m^2$
$\beta = b^2\,m^2 + a^2\,p^2$
$\gamma_o = (b^2-a^2)pm$
$\gamma = 2\,\gamma o$
$k = a^2\,b^2\,(p^2+m^2)$

L'équation (1) devient alors :

$$\alpha(X-X_c)^2+\beta(Y-Y_c)^2+\gamma(X-X_c)(Y-Y_c)=k. \quad (2)$$

L'algorithme qui suit engendre l'ellipse quelconque (voir figure 2.10) définie par l'équation (2) (ellipse simple ayant subi une rotation).

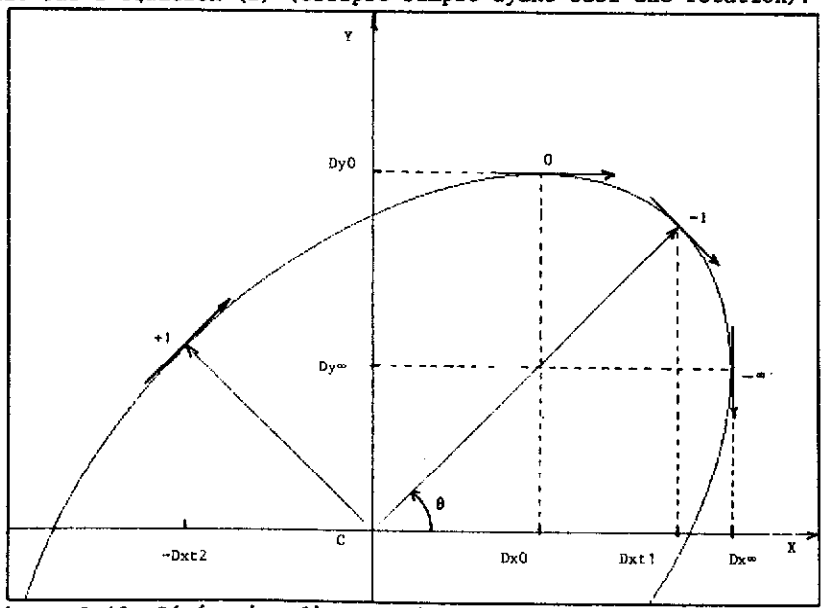


Figure 2.10: Génération d'une ellipse quelconque (méthode de ROY).


```
Génération ELLIPSE QUELCONQUE (ENTIER: XC, YC, A, B, M, P)
Début co méthode de ROY fco
    ENTIER α, β, γ0, γ, k, E, T, S, D
    ENTIER Ca, Cd, Dsa, Dsd, Dta, Dsd
    ENTIER Dxt1, Dxt2, DX0, DY0, DY∞
    REEL U
    /* Initialisation */
    A2 := A*A
    B2 := B*B
    M2 := M*M
    P2 := P*P
    α  := P2*B2+M2*A2
    β  := M2*B2+P2*A2
    γ0 := (B2-A2)*P*M
    γ  := 2*γ0
    k  := A2*B2*(P2+M2)
    Z  := α*β-γ0*γ0
    Dxt1 := ARRONDI(SQR(k+(β+γ0)²/((α+β+γ)*Z)))
    Dxt2 := ARRONDI(SQR(k+(β-γ0)²/((α+β-γ)*Z)))
    U  := γ0*SQR(k/(α*Z))
    Dx0 := ARRONDI(U)
    Si γ0 = 0 alors Dy0 := B
              sinon Dy0 := ARRONDI(α*U/γ0)
    Dy∞ := ARRONDI((γ0/β)*SQR(k*β/Z))
    D  := 2*(k-α*DX0²-β*DY0²+γ*DX0*DY0)
    T  := 4*α*DX0 - 2*γ*DY0
    S  := 4*β*DY0 - 2*γ*DX0
    /* 1er quadrant */
    X  := Xc-Dx0
    Y  := Yc+Dy0
    Dxt := Dxt2
    Xinc := +1
    Yinc := -1
    Sg := 1
    TRACER-QUADRANT
    /* 2ème quadrant */
    X  := Xc+Dx0
    Y  := Yc-Dy0
    Yinc := +1
    Dxt := Dxt1
    Sg := -1
    TRACER-QUADRANT
    /* 3ème quadrant */
    X  := Xc+Dx0
    Y  := Yc-Dy0
    Xinc := -1
    Dxt := Dxt2
    Sg := +1
    TRACER-QUADRANT
```

```
/* 4ème quadrant */
X := Xc - Dxo
Y := Yc + Dyo
Yinc := -1
Dxt := Dxt1
Sg := -1
TRACER-QUADRANT

PROCEDURE TRACER-QUADRANT
E := D + T + S/2 - 2 * α - β + Sg * γ
T := Sg*T
S := S
Ca:= -6*α+Sg*γ
Cd:= Ca-4*(β-Sg*γ)
Dta:=-4*α
Dsa:=2*Sg*γ
Dtd:=Dta+Dsa
Dsd:=Dsa-4*β

Afficher (X,Y)

Bt := ABS(X-Xc-Xinc*dxt)

Pour I := 1 à Bt faire
Début
    X := X+Xinc
    Si E>=0
       alors début
             E := E+T+Ca
             T := T+Dta
             S := S+Dsa
             fin
       sinon début
             Y := Y+Yinc
             E := E+T+S+Cd
             T := T+Dtd
             S := S+Dsd
             fin
    Afficher(X,Y)
fin
E := E-T/2+S/2+α-β

Ca := -6*β+Sg*γ
Cd := Ca-4*(α-Sg*γ)
Dta:= Dsa
Dsa:= -4*β
Dyt:= ABS(Y-Yc+Xinc*Dy∞)

Pour I :=1 à Dyt faire
Début
   y := Y+Yinc
   Si E<=0
      alors début
            E :=E+S+Ca
            T := T+Dta
            S := X+Dsa
            fin
      sinon début
            X := X+Xinc
            E := E+T+S+Cd
            T := T+Dtd
            S := S+Dsd
            fin
  Afficher (X,Y)
  fin
  FIN TRACER-QUADRANT
Fin
```


2.7. Génération des paraboles

2.7.1 - ARCS DE PARABOLE SIMPLE (*algorithme de ROY*)

La parabole est définie par l'équation $y=a(x-x_c)^2+b$ où (x_c,b) est l'extremum de la courbe (voir figure 2.11.).

Soient :

XA,YA les coordonnées de l'origine de la portion de courbe à tracer
XB,YB les coordonnées de son extrémité
XC détermine l'axe de symétrie de la courbe (i.e. de la droite d'équation $x=x_c$)

on a $a=(YB-YA)/((XB-XA)*(XB+XA-2*XC))$

$$b=\frac{(XB-XC)^2*YA-(XA-XC)^2*YB}{(XB-XA)*(XB+XA-2*XC)}$$

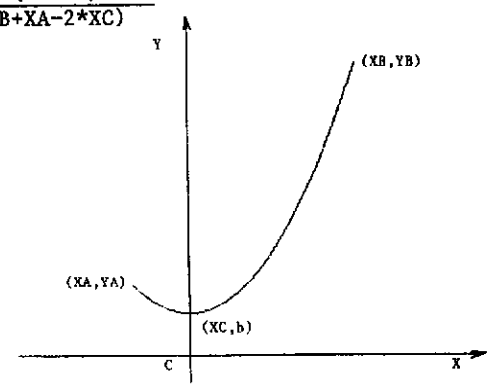


Figure 2.11.: Génération d'un arc de parabole (méthode de ROY).

L'algorithme de génération d'un arc de parabole développé par ROY, a la forme suivante :

```
    Génération ARC DE PARABOLE (Entier: XA,YA,XB,YB,XC)
Début co méthode de ROY fco
    Entier : YC,DSXO,CBO
    Si (XA<XC et XC<XB) ou (XB<XC et XC<XA)
    alors début
          si ABS(XA-XC)=ABS(XB-XC) alors lire YC
          sinon DSXO := (XB-XA)*(XB+XA-2*XC)
                CBO  := (XB-XC)²*YA-(XA-XC)²*YB
                YC   := ARRONDI (CBO/DSXO)
          finsi
                TRACER (XC,YC,XA,YA)
                TRACER (XC,YC,XB,YB)
       fin
       sinon début
             si (XC≤XA et XA<XB) ou
                (XC≥XA et XA>XB)
                alors TRACER (XA,YA,XB,YB)
                sinon TRACER (XB,YB,XA,YA)
          fin
Fin
```

```
   Procédure TRACER(Entier: XD,YD,XF,YF)
Début
   Entier : DX,DY,DELTAY,DT,DSX,DXT,
            X,Y,XINC,YINC,T,T,CA,CD
   DX := XF-XD
   DY := YF-YD
   DELTAY := ABS(DY)
   DT := 4*DELTAY
   DSX := DX*(XF+XD-2*XC)
   DXT := ARRONDI(2*DSX/DT)
   Si DX>0
      alors XINC := +1
      sinon XINC := -1
   Si DY>0
      alors YINC := +1
      sinon YINC := -1
   X := XD
   Y := YD
   Afficher (X,Y)
   T := DT*ABS(X-XC)
   E := T+DT/2-DSX  /* DT est pair */
   Si ABS (X-XC)<DXT
      alors début  /* cas des tangentes de pente ∈ [0,1] */
               CA := 6*DELTAY
               CD := CA-2*DSX
               BT := MIN(ABS(DX),-ABS(X-XC)+DXT)
               Pour I := 1 jusqu'à BT faire
               début
                 X := X+XINC
                 Si E<=0
                    alors  /* mvt axial */
                           E := E+CA+T
                    sinon  /* mvt diagonal */
                           Y := Y+YINC
                           E := E+CD+T
                 T := T+DT
                 Afficher (X,Y)
               fin
           fin
   DYT := ABS (YF-Y)
   Si DYT ≠ 0
      alors début /* cas des tangentes de pente ∈ ]1,+∞[
               CA := 2*DSX
               CD := CA-4*DELTAY
               T := -T
               E := -E-T/2+DELTAY+DSX  /* T est pair */
               Pour I := 1 pas 1 jusqu'à DYT faire
               début
                 Y := Y+YINC
                 Si E<=0
                    alors  /* mouvement axial */
                           E := E+CA
                    sinon  /* mouvement diagonal */
                           X := X+XINC
                           E := E+CD+T
                           T := T-DT
                 Afficher(X,Y)
               fin
           fin
      TRACEP.
Fin
```

N.B.: Dans le cas où |XA-XC|=|XB-XC| on choisira arbitrairement b, et on appliquera l'algorithme successivement à |(Xc,Yc),(XA,YA)| et |(Xc,Yc),(XB,YC)|.

2.7.2 - ARCS DE PARABOLE QUELCONQUE

Soit Y-Yc=a(X-Xc)² l'équation de la parabole de centre de symétrie (Xc,Yc).

Lorsque l'on effectue une rotation d'un angle θ avec $\theta \in]-\frac{\pi}{2}, \frac{\pi}{2}[$, nous obtenons :

$$a\,tg^2\theta\,(Y-Yc)^2+a(X-Xc)^2+2a\,tg\theta\,(X-Xc)(Y-Yc)$$
$$-(Y-Yc)\sqrt{1+tg^2\theta}+(X-Xc)\,tg\theta\sqrt{1+tg^2\theta}=0$$

Nous ne pouvons pas ici appliquer le même principe que pour les ellipses.

En effet, nous voyons apparaître les variables X et Y au degré 1 ailleurs que dans le double produit, ce qui introduit le coefficient $\sqrt{1+tg^2\theta}$.

La présence de ce coefficient nous empêche de nous ramener à une expression entière comme celle obtenue dans le cas des ellipses. De ce fait, les incrémentations effectuées à chaque étape des calculs ne seraient plus entières, ce qui provoquerait une accumulation des erreurs. Or, l'intérêt de toutes les méthodes présentées jusqu'ici est justement le fait qu'elles n'utilisent que des opérations entières.

2.8. Génération des hyperboles

A titre d'exemple, nous développons ci-dessous la généralisation de la méthode de BRESENHAM exposée au chapitre 2.3.3.2 pour la génération des hyperboles.

2.8.1 - GENERALISATION DE L'ALGORITHME DE BRESENHAM AUX HYPERBOLES "SIMPLES"

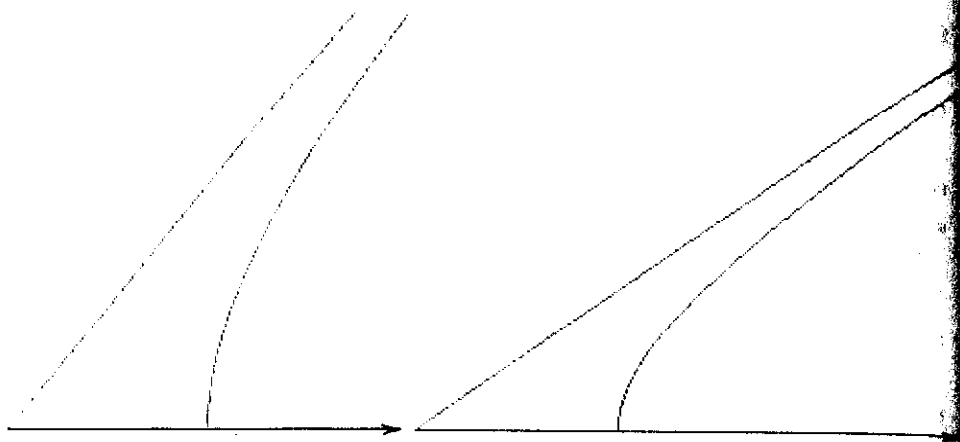
Nous considérons une hyperbole centrée à l'origine du repère et d'équation :

$$\frac{X^2}{a^2}-\frac{Y^2}{b^2}=1 \text{ où } a \text{ et } b \in N^*$$

La généralisation à une hyperbole de centre quelconque (Xc,Yc) est obtenue par simple translation.

L'hyperbole possède deux asymptotes de pente $+\frac{b}{a}$ et $-\frac{b}{a}$. Nous raisonnerons uniquement sur le premier quadrant ($X \geq 0$ et $Y \geq 0$) car les autres branches de l'hyperbole sont obtenues par symétrie. Dans ce quadrant, les pentes des tangentes à la courbe appartiennent à $]\frac{b}{a}, +\infty[$. On peut faire deux hypothèses :

- si $\frac{b}{a} > 1$, il n'y a pas de changement d'octant (figure 2.12.a)
- si $\frac{b}{a} \leq 1$, il y a un changement d'octant (figure 2.12.b.)



a) $b \leq a$ b) $b > a$

Figure 2.12.: Génération de l'hyperbole (premier quadrant).

Le quadrant considéré peut se décomposer en deux parties :

A) Les pentes des tangentes à la courbe appartiennent à $[1,+\infty[$, ce qui correspond aux points (x,y) tels que :

$x \in [a,+[$, si $b \geq a$,

ou

$x \in [a, \frac{a^2}{\sqrt{a^2-b^2}}[$, si $b < a$.

La progression se fera en Y.

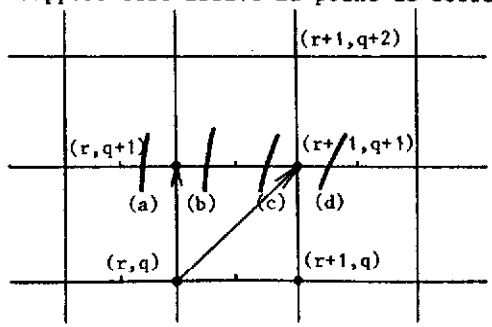
B) Les pentes des tangentes à la courbe appartiennent à $]\frac{b}{a},1[$, ce qui correspond aux points (x,y) tels que :

$x \in]\frac{a^2}{\sqrt{a^2-b^2}}, +\infty[$ si $b < a$.

La progression se fera en X.

a) Les pentes des tangentes $\in [1,+\infty[$

On suppose être arrivé au point de coordonnées (r,q)



. Soit $P_1(x_1,q)$ le point de la courbe à afficher, le point (r,q) étant le point entier le plus proche de la courbe. On a :

$$r - \frac{1}{2} < x_1 \leq r + \frac{1}{2}$$

Posons $D(r,q) = a^2q^2 + a^2b^2 - b^2r^2$.

Alors D(r,q) est un nombre entier de même signe que (x_1-r).

Soit $P_2(x_2,q+1)$ le point suivant de l'hyperbole à afficher.

Comme la pente $\in [1,+\infty[$ on aura :

$$r - \frac{1}{2} < x_2 \leq r + \frac{3}{2}$$

On montre que la quantité E définie par E=D(r,q+1)+D(r+1,q+1), détermine le choix entre le mouvement axial et le mouvement diagonal :

- si $E \leq 0$ (cas (a) et (b)) alors mouvement axial ;
- si $E > 0$ (cas (c) et (d)) alors mouvement diagonal.

On a :
$E = 2\,D(r,q) + 2*S(q) + T(r) + 2a^2 - b^2$.

Dans le cas du mouvement axial, on passe au point (r,q+1). La prochaine valeur de E à considérer sera donc :

$$Er,q + 2*S(q) + 6\,a^2$$

Dans le cas d'un mouvement diagonal, on passe au point (r+1,q+1). La prochaine valeur de E à considérer sera donc :

$$Er,q + 2*S(q) + 2*T(r) + 6\,a^2 - 4\,b^2$$

où $T(r) = \frac{d\,D(r,q)}{dr} = -2\,b^2\,r$

$S(q) = \frac{d\,D(r,q)}{dq} = +2\,a^2\,q$

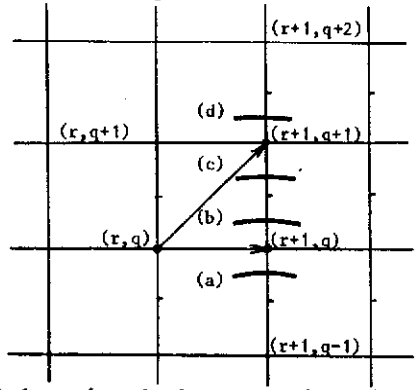
avec $T(r+1) = T(r) - 2\,b^2$
$S(q+1) = S(q) + 2\,a^2$

En résumé, le point le plus proche de la courbe suivant (r,q) sera donné par la séquence :

```
si E <= 0
   alors mouvement axial(en Y)
         E := E + 2*S + 6 a² ;
         S := S + 2 a² ;
   sinon mouvement diagonal
         E := E + 2*S + 2*T + 6 a² - 4 b² ;
         S := S + 2 a² ;
         T := T - 2 b² ;
```

b) Les pentes des tangentes $\theta \in]+\frac{b}{a}, +1[$

On suppose être arrivé au point de coordonnées (r,q).



. Soit $P_1(r\ y_1)$ le point de la courbe à afficher, le point (r,q) étant le point de coordonnées entières le plus proche de la courbe. On a :

$$q - \frac{1}{2} < y_1 \leqslant q + \frac{1}{2}$$

L'expression $D(r,q)=b^2r^2 - a^2b^2 - a^2q^2$ est du signe de (y_1-q).

. Soit $P_2(r+1,y2)$ le point suivant de l'hyperbole à afficher. Comme la pente $\theta \in]\frac{b}{a},1[$ on aura :

$$q - \frac{1}{2} < y2 < q + \frac{3}{2}$$

On montre que la quantité E définie par E=D(r+1,q)+D(r+1,q+1) détermine le choix du mouvement :

- si E≤0 (cas (a) et (b)) alors mouvement axial
- si E>0 (cas (c) et (d)) alors mouvement diagonal.

On a donc :
$E=2\ D(r,q) + 2\ T(r) + S(q) + 2\ b^2 - a^2$

Dans le cas du mouvement axial, on passe au point (r+1,q). La prochaine valeur de E à considérer sera donc

$E(r,q) + 2 * T(r) + 6\ b^2.$

Dans le cas d'un mouvement diagonal, on passe au point (r+1,q+1). La prochaine valeur de E sera donc :

$$E_{r,q} + 2*T(r) + 2*S(q) + 6\ b^2 - 4\ a^2$$

où $T(r) = \frac{dD(r,q)}{dr} = +2\ b^2r$

$$S(q) = \frac{dD(r,q)}{dq} = -2\ a^2q$$

avec $T(r+1) = T(r) + 2\ b^2$
$S(q+1) = S(q) - 2\ a^2.$

En résumé, le point suivant le plus proche de la courbe sera donné par la séquence :

```
si E<=0
  alors mouvement axial (en X) ;
        E := E + 2.T + 6 b²  ;
        T := T + 2 b²  ;
  sinon mouvement diagonal ;
        E := E + 2*T + 2*S + 6 b² - 4 a²  ;
        S := S - 2 a²  ;
        T := T + 2 b²  ;
```

L'algorithme :

. Initialisations -

A- Cas des tangentes de pente $\theta \in [1,+\infty[$
Le point de départ est donné par :
XD = XC + a
YD = YC où (XC,YC) est le centre de symétrie

Ce point appartient à l'hyperbole donc D(XD,YD)=0 et on a :
T(XD) = -2 b²(XD-XC) = -2 b²*a
S(YD) = 2 a² (YD-YC) = 0

On aura les initialisations suivantes :
T := -2 a*b²
S := 0
E := T + 2a² - b².

B- Cas des tangentes de pente $\theta \in]\frac{b}{a},+1[$ (i.e. b<a)
Dans le cas du déplacement en Y on avait
$D_y(r,q) = a^2(q-Y_c)^2 + a^2b^2 - b^2\ (r-X_c)^2$
$T_y(r) = -2\ b^2(r-X_c)$
$S_y(q) = 2\ a^2\ (q-Y_c)$
$E_y(r,q) = 2\ D_y(r,q)^2 + 2*S_y(q) + T_y(r) + 2a^2 - b^2.$

Dans le cas du déplacement en X, on a :
$D_x(r,q) = -D_y(r,q)$
$T_x(r) = -T_y(r)$
$S_x(q) = -S_y(q)$
$E_x(r,q) = -(E + T_y(r) - S_y(q) - b^2 - a^2).$

On aura donc les initialisations suivantes :
E := - E - T + S + b² + a²
T := - T
S := - S.

Le dernier point peut être défini par une borne en X ou en Y, ou bien par un point appartenant à l'hyperbole (XF,YF).

. L'algorithme suivant engendre une hyperbole admettant des axes de symétrie parallèles aux axes du repère et ayant pour équation

$\frac{(x-xc)^2}{a^2} - \frac{(y-yc)^2}{b^2} = 1$ (voir figure 2.13).

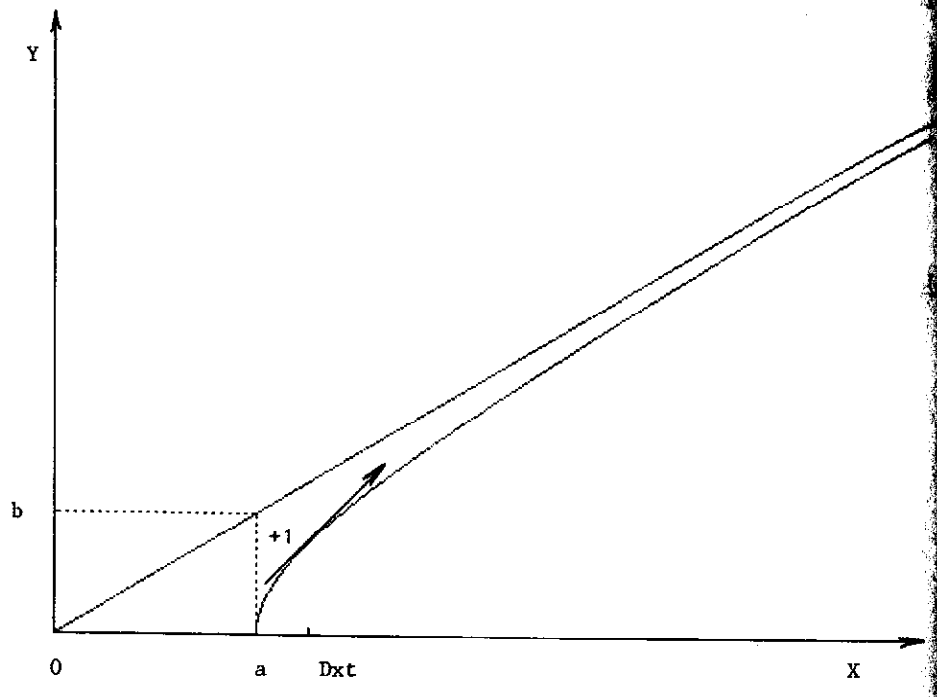


Figure 2.13.: Génération d'une hyperbole (premier quadrant) où b<a et (Xc,Yc)=(0,0) (généralisation de la méthode de BRESENHAM).

Nous obtenons, dans le cadre du premier quadrant où XF est l'abscisse maximale de la branche asymptotique l'algorithme suivant :

```
Génération HYPERBOLE (Xc,Yc,H,B,XF:entier)
Début co génération du premier quadrant fco
    entier : A2,B2,DS,DT
             CA,CD,S,T,E
             X,Y,DXT
    co initialisations fco
    A2 := A*A;
    B2 := B*B;
    DS := +2*A²;
    DT := -2*B²;
    si B<A alors DXT := ARRONDI (A²/SQR(A2-B2));
           sinon DXT := XF-XC   co pente ≥ 1, ∀ x fco
    X := XC+A;
    Y := YC;
```



```
    Affficher point (X,Y);
    T := -2*A*B2;
    S := 0;
    E := T+2*A2-B2;
    CA:= 6*A2;
    CD:= CA-4*B2;

    co on se déplace sur l'axe des ordonnées fco
    Tant que X < DXT faire
    début
        Y := Y+1;
        si E<=0
          alors co mouvement axial fco
                E := E+2*S+CA;
                S := S+DS;
          sinon co mouvement diagonal fco
                X := X+1;
                E := E+2*S+2*T+CD;
                S := S+DS;
                T := T+DT;
        fsi
        Affficher point (X,Y);
    fin
    si b<a
      alors DS := -DS;
            DT := -DT;
            E := -E-T+S+B2+A2;
            T := -T ; S := -S;
            CA:= 6*B2 ; CD := CA-4*A2;
            co on se déplace suivant l'axe des abscisses fco
            Tant que X ≤ XF faire
            Début
                X := X+1;
                si E≤0;
                  alors co mouvement axial fco
                        E := E+2*T+CA;
                        T := T+DT;
                  sinon co mouvement diagonal fco
                        Y := Y+1;
                        E := E+2*S+2*T+CD;
                        S := S+DS;
                        T := T+DT;
                fsi
                Affficher point (X,Y);
            fin
    fsi
fin
```

Dans le premier quadrant, les incréments en X et Y sont égaux à (1,1). Pour les second, troisième et quatrième quadrants, ils seront respectivement égaux à (1,-1),(-1,-1),(-1,1).

2.8.2 - HYPERBOLES AVEC ROTATION

Lorsque les axes de symétrie de l'hyperbole sont parallèles aux axes du repère, l'équation de la courbe est donnée par :

$$b^2 Xo^2 - a^2 Yo^2 = a^2 b^2$$

Après avoir effectué une rotation d'un angle $\theta\theta \in]-\frac{\pi}{2},+\frac{\pi}{2}[$ nous obtenons l'équation :

(1) $(p^2b^2-a^2m^2)X^2+(b^2m^2-a^2p^2)Y^2-2(a^2+b^2)p.m.XY = a^2b^2(p^2+m^2)$

avec $tg\theta = \frac{m}{p}$ où $m \in Z$ et $p \in Z^*$

En posant $\alpha = p^2b^2 - a^2m^2$
$\beta = b^2m^2 - a^2p^2$
$\gamma_o = (a^2+b^2)pm$
$\gamma = 2\ \gamma o$
$k = a^2b^2(p^2+m^2)$.

L'équation (1) devient alors :

$$\alpha X^2 + \beta.Y^2 + \gamma.XY = k$$

. Changement de quadrant : comme pour l'ellipse, le changement de quadrant s'effectue aux points où la pente de la tangente est soit nulle, soit infinie.

On montre qu'une tangente de pente infinie existe si et seulement si $\frac{m}{p} < \frac{a}{b}$. Dans ce cas, les coordonnées des points où la pente est infinie sont donnés par $(DX\infty,DY\infty)$ et $(-DX\infty,-DY\infty)$ où :

$$DY\infty = \sqrt{\frac{k\beta}{\alpha\beta-\gamma o^2}}$$

$$DY\infty = -\frac{\gamma o}{\beta} DX\infty$$

De même, une tangente de pente nulle existe si et seulement si $\frac{m}{p} > \frac{b}{a}$. Dans ce cas, les coordonnées des points sont données par (DXo,DYo) et $(-DXo-DYo)$ où

$$DXo = \gamma o\sqrt{\frac{k}{\alpha(\beta\alpha-\gamma o^2)}}$$

$$DYo = -\frac{\alpha}{\gamma o} DXo$$

. Changement de mouvement (octant). Le changement s'opère aux points tels que la pente de la tangente à la courbe est égale à +1 ou -1.

On montre qu'une tangente de pente égale à +1 existe si et seulement si $\frac{b}{a} < \frac{|m-p|}{|m+p|}$, et dans ce cas les coordonnées des points sont données par (DX_1,DY_1) et $(-DX_1,-DY_1)$ où

$$DX_1 = \sqrt{\frac{k(\beta+\gamma o)^2}{(\alpha+\beta-2\gamma o)(\alpha\beta-\gamma o^2)}}$$

$$DY_1 = -\frac{\gamma o + \alpha}{\beta + \gamma o} DX_1$$

De même, une tangente de pente égale à -1 existe si et seulement si $\frac{b}{a} < \left|\frac{m+p}{m-p}\right|$, et les coordonnées des points sont alors données par (DX_{-1},DY_{-1}) et $(-DX_{-1},-DY_{-1})$ où

$$DX_{-1} = \sqrt{\frac{k(\beta-\gamma o)^2}{(\alpha+\beta-2\gamma o)(\alpha\beta-\gamma o^2)}}$$

$$DY_{-1} = \frac{\alpha-\gamma o}{\beta-\gamma o} DX_{-1}$$

. Les asymptotes de l'hyperbole ont alors pour pentes

$$p_1 = \frac{am+bp}{ap-bm},\ p_2 = \frac{am-bp}{ap-bm}$$

et les conditions d'existence des pentes de la tangente à la courbe sont :

si $P_1>0$ alors les pentes infinies existent,
si $P_2>0$ alors les pentes nulles existent,
si $0<P<1$ alors les pentes = +1 existent,
si $-\infty<P_1<-1$ alors les pentes = -1 existent,
si $1<P_2<+\infty$ alors les pentes = +1 existent,
si $-1<P_2<0$ alors les pentes = -1 existent.

Les conditions $(0<P_1<1)$ et $(1<P_2<+\infty)$ sont exclusives, ainsi que $(-\infty<P_1<-1)$ et $(-1<P_2<0)$.

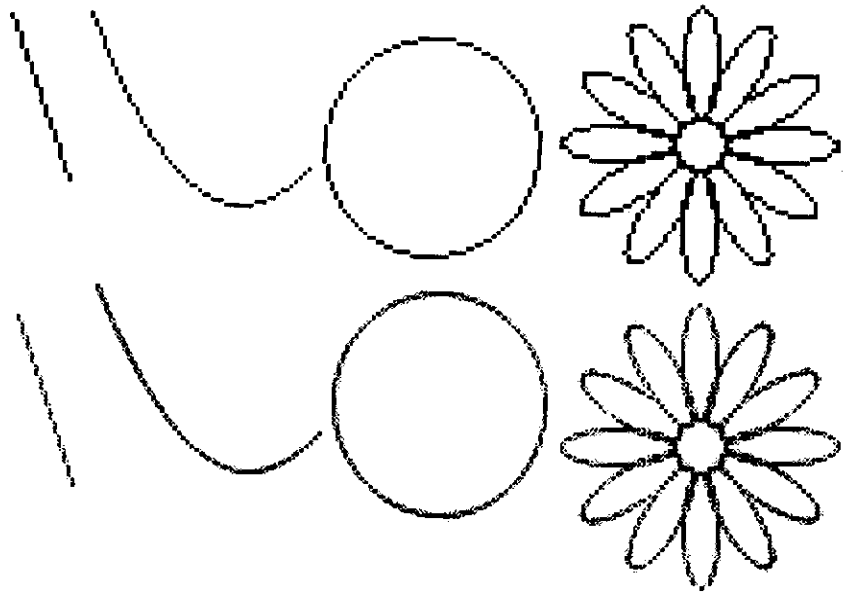
Ayant défini les quadrants et les changements de mouvement ainsi que leurs conditions d'existence, nous pouvons comme pour l'ellipse, déterminer pour chaque quadrant en tenant compte ici des branches asymptotiques et de la symétrie, les séquences qui calculeront les points les plus proches de la courbe.

2.9. Amélioration du dessin au trait

2.9.1 - INTRODUCTION

La quantification produit un "effet d'escalier" relativement désastreux du point de vue de l'esthétique des images. Plusieurs méthodes sont employées pour réduire ces défauts. En particulier, l'idée de BRESENHAM est de répartir l'intensité maximale sur deux points (un point principal sélectionné comme auraparavant et un point secondaire dans des proportions fonction de leurs distances respectives à la courbe). M.A.ROY (<ROY 83>), a généralisé cette solution pour l'amélioration du tracé des cercles, des ellipses et des paraboles (voir figure 2.14.) à partir des algorithmes de génération des coniques utilisant la méthode de base de BRESENHAM. En outre, elle a montré également que cette technique était simple et efficace par rapport aux autres propositions et en particulier par rapport à celles

qui utilisent des filtres numériques.



.segment de droite .arc de parabole .cercle .ellipses

Figure 2.14.: Amélioration du dessin au trait (grisés obtenus avec des cellules 3x3) (tiré de <ROY 83>).

2.9.2 - METHODE DE BRESENHAM

2.9.2.1 - Le Principe

Cette méthode d'élimination de l'effet d'escalier dérive directement de l'algorithme de génération des segments de droite de BRESENHAM. Cette méthode sélectionne le point principal et un point secondaire. Le point secondaire est tel que le segment passe entre ce point et le point principal. Ces deux points sont les points les plus proches du segment théorique. L'intensité attribuée à un point est fonction de la distance de ce point au segment. La somme des intensités de luminosité des points principal et secondaire est égale à l'intensité maximale (notée Imax) pouvant être attribuée à un pixel.

Aucune modification fondamentale n'est apportée à l'algorithme de BRESENHAM. On se contente d'exploiter toutes les informations données. En effet, les différents cas de figure avaient été regroupés en deux catégories : cas du mouvement axial et cas du mouvement diagonal. Il suffit donc

de faire "éclater" ces regroupements pour obtenir tous les cas de figure possibles.

Les opérations effectuées restent des opérations entières (addition ou soustraction) et des comparaisons. Seul le coefficient donnant la répartition de l'intensité entre les points principal et secondaire est un réel. Il est le résultat de la division de deux variables entières.

L'intérêt de cette méthode réside dans le fait que ces deux variables entières sont déjà calculées dans l'algorithme initial, et ne sont par conséquent pas introduites spécialement pour le calcul des intensités de luminosité.

2.9.2.2 - Amélioration du tracé des segments de droite

Nous redonnons la version de ROY, (<ROY 83>) où le segment de droite est défini par les coordonnées de ses extrémités (XD,YD),(XF,YF) et où Imax est l'intensité maximale et Ip et Is sont respectivement les intensités des points principaux et secondaires.

```
Génération améliorée SEGMENT (XD,YD,XF,YF,Imax:entier)
Début co méthode de ROY fco
    DX := XF-XD
    DY := YF-YD
    DELTAX := ABS(DX)
    DELTAY := ABS(DY)
    Si DX>0
       alors Xinc := +1
       sinon Xinc := -1
    Si DY>0
       alors Yinc := +1
       sinon Yinc := -1
    X := XD ; Y := YD
    Afficher point (X,Y,Imax)
```

```
Suivant cas faire
   DELTAX>=DELTAY
   Début
      D := 0
      Pour I := 1 pas 1 jusqu'à DELTAX faire
      Début
            X := X+Xinc
            D := D+DELTAY
            Si D<=0
               alors Y_δ :=Y-Yinc
               sinon début
                        Si D-DELTAX<0
                        alors début
                                 Si 2*D-DELTAX<=0
                                 alors Y_δ := Y+Yinc
                                 sinon début
                                          Y_δ := Y
                                          Y := Y+Yinc
                                          D := D-DELTAX
                                       fin
                              fin
                        sinon début
                                 Y := Y+Yinc
                                 Y_δ := Y+Yinc
                                 D := D-DELTAX
                              fin
                     fin
            Cp := ABS(D/DELTAX)
            I_δ := ARRONDI (Cp*Imax)
            Ip := Imax-Is
            Afficher point (X,Y,Ip)
            Afficher point (X,Y_δ,I_δ)
      fin
fin
```



```
DELTAY>DELTAX
Début
   D := 0
   Pour I := 1 pas 1 jusqu'à DELTAY faire
   Début
      Y := Y+Yinc
      D := D+DELTAX
      Si D<=0
         alors X_δ := X-Xinc
         sinon début
                  Si D-DELTAY<0
                  alors début
                           Si 2*D-DELTAY<=0
                           alors X_δ := X+Xinc
                           sinon début
                                    X_δ := X
                                    X := X+Xinc
                                    D := D-DELTAY
                                 fin
                        fin
                  sinon début
                           X := X+Xinc
                           Xs := X+Xinc
                           D := D-DELTAY
                        fin
               fin

      Cp := ABS(D/DELTAY)
      I_δ := ARRONDI(Cp*Imax)co Intensité attribuée au point
             secondaire fco
      Ip := Imax-Is co Intensité attribuée au point principal fco
      Afficher point (X,Y,Ip)
      Afficher point (Xs,Y,Is)
   fin
fin
Fincas
Fin
```

2.9.3 - AMELIORATION DU TRACE DES CONIQUES

2.9.3.1 - *La méthode de ROY*

Par application méthodologique du procédé d'élimination des effets d'escalier employé par BRESENHAM pour le tracé des segments de droite, ROY a développé de façon détaillée dans (<ROY 82>) des algorithmes visant à supprimer ces mêmes effets obtenus lors de la génération des ellipses et des arcs de parabole .

2.9.3.2 - Amélioration du tracé des ellipses

. Ellipses simples

Nous considérons une ellipse centrée en C(XC,YC) et d'équation :

$$\frac{(X-XC)^2}{A^2} + \frac{(Y-YC)^2}{B^2} = 1$$

L'algorithme donné par ROY, a la forme suivante :

```
Génération améliorée ELLIPSE SIMPLE        (XC,YC,A,B,Imax : entier)
Début co méthode de ROY fco
    A2 := A*A
    B2 := B*B
    DS := 2*A2
    DT := 2*B2
    DXT := ARRONDI (A2/SQR(A2+B2))
    Xinc := +1      /* 1er quadrant */
    Yinc := -1
    TRACER-QUADRANT
    Yinc := +1      /* 2è quadrant  */
    TRACER-QUADRANT
    Xinc := -1      /* 3è quadrant  */
    TRACER-QUADRANT
    Yinc := -1      /* 4è quadrant  */
    TRACER-QUADRANT

Procédure TRACER-QUADRANT
Début
    T := 0
    S := -2*A2*B
    D := 0
    X := XC
    Y := YC-Yinc*B
    Afficher(X,Y,Imax)
    Pour I := 1 pas 1 jusqu'à DXT faire
    Début
        X := X+Xinc
        D := D+T-B2
        Si D>=0
            alors YS := Y-Yinc
```



```
            sinon début
                Si D-S-A2>0
                  alors début
                      si 2*D-S-A2>=0
                        alors YS := Y+Yinc
                        sinon début
                            YS := Y
                            Y := Y+Yinc
                            D := D-S-A2
                            S := S+DS
                          fin
                    fin
                  sinon début
                      Y := Y+Yinc
                      YS:= Y+Yinc
                      D := D-S-A2
                      S := S+DS
                    fin
                fin
        T := T-DT
        Cp := ABS(D/S)
        Is := Cp*Imax
        Ip := Imax-Is
        Afficher point (X,Y,Ip)
    fin
    DYT := ABS(Y-YC)
    Pour I := 1 pas 1 jusqu'à DYT faire
    Début
        Y := Y+Yinc
        D := D-S-A2
        Si D<=0
            alors XS := X-Xinc
            sinon début
                    Si D+T-B2<0
                      alors début
                          Si 2*D+T-B2<=0
                            alors XS := X+Xinc
                            sinon début
                                XS := X
                                X := X+Xinc
                                D := T-B2+D
                                T := T-DT
                              fin
                        fin
                      sinon début
                          X := X+Xinc
                          XS:= X+Xinc
                          D := D+T-B2
                          T := T-DT
                        fin
                  fin
        S := S+DS
        Cp := ABS(D/T)
        Is := Cp*Imax
        Ip := Imax-Is
        Afficher point (X,Y,Ip)
        Afficher point (XS,Y,Is)
    fin
Fin TRACER-QUADRANT
```

Cet algorithme servira également pour la génération améliorée des cercles de rayon R avec A=B=R.

. Ellipses quelconques

Dans l'algorithme de ROY qui suit, nous considérons les ellipses simples auxquelles nous appliquons une rotation d'angle $\theta \in]-\frac{\pi}{2}, +\frac{\pi}{2}[$ dont la tangente est donnée par :

$$\operatorname{tg}\theta = \frac{M}{P} \text{ avec } M \in Z \text{ et } P \in Z^*$$

```
Génération améliorée ELLISPE QUELCONQUE      (XC,YC,A,B,M,P:entier)
Début co méthode de ROY foo
    ENTIER α,β,γo,γ,k,D,D1,T,S
    ENTIER Dsa,Dsd,Dta,Dtd
    ENTIER Dxt1,Dxt2,Dxo,Dyo,Dy∞
    REEL U

    /* Initialisation */

    A2 := A*A
    B2 := B*B
    M2 := M*M
    P2 := P*P

    α  := P2*B2+M2*A2
    β  := M2*B2+P2*A2
    γo := (B2-A2)*P*M
    γ  := 2*γo
    k  := A2*B2*(P2+M2)

    Dxt1 := ARRONDI(SQR(k*(β+γo)²/((α+β+2*γo)*(α*β-γo²))))
    Dxt2 := ARRONDI(SQR(k*(β-γo)²/((α+β-2*γo)*(α*β-γo²))))
    U := γo*SQR(k/(β*α²-α*γo²))

    DXO := ARRONDI (U)
    Si γo = 0 alors Dyo := β
              sinon Dyo := ARRONDI(α*U/γo)
    Dy∞ := ARRONDI((γo/β)*SQR(k*β/(β*α-γo²)))
    D1 := k-α*DXO²-β*DYO²+γ*DXO*DYO
    TO := 2*α*DXO-γ*DYO
    SO := 2*β*DYO-γ*DXO

    /* 1er quadrant */
    X := Xc-DXO
    Y := Yc+DYO
    Dxt := Dxt2
    Xinc := +1
    Yinc := -1

    Sg := 1
    TRACER-QUADRANT

    /* 2è quadrant */
    X := Xc+DXO
    Y := Yc-DYO
    Yinc := +1
    Dxt := Dxt1

    Sg := -1
    TRACER-QUADRANT

    /* 3è quadrant */
    X := Xc+DXO
    Y := Yc-DYO
    Xinc := -1
    Dxt := Dxt2

    Sg := 1

    TRACER-QUADRANT

    /* 4è quadrant */
    X := Xc-DXO
    Y := Yc+DYO
    Yinc := -1
    Dxt := Dxt1
    Sg := -1

    TRACER-QUADRANT
Fin
```



```
Procédure TRACER-QUADRANT
D := D1
Dta := -2*α
Dsa := Sg*γ
Dtd := Sg*γ
Dsd := -2*β

T := TO*Sg
S := SO

Si D≥0 alors Ys := Y-Yinc
       sinon Ys := Y+Yinc

Cp := ABS(D/S)
Is := Cp*Imax
Ip := Imax-Is

Afficher point (X,Y,Ip)
Afficher point (X,Ys,Is)

Bt := ABS(X-Xc-Xinc*Dxt)

Pour I := 1 à Bt faire
Début
    X := X+Xinc
    D := D+T-α

    Si D≥0
       alors Ys := Y-Yinc
       sinon début
                Si D+S-β+Dsa>0
                   alors début
                            Si 2*D+S-β+Dsa≥0
                               alors Ys := Y+Yinc
                               sinon début
                                        Ys := Y
                                        Y := Y+Yinc
                                        D := D+S-β+Dtd
                                        T := Dtd+T
                                        S := Dsd+S
                                     fin
                         fin
                   sinon début
                            Y := Y+Yinc
                            Ys:= Y+Yinc
                            D := D+S-β+Dtd
                            T := Dtd+T
                            S := Dsd+S
                         fin
    fin
```

```
        S := S+Dsa
        T := T+Dta

        Cp:= ABS(D/S)
        Is:= Cp*Imax
        Ip:= Imax-Is

        Afficher (X,Y,Ip)
        Afficher (X,Ys,Is)
    fin
    Dta := Sg*γ
    Dsa := -2*β
    Dtd := -2*α
    Dsd := Sg*γ

    Dyt := ABS(Y-Yc+Xinc*Dy∞)

    Pour I := 1 à Dyt faire
    Début
        Y := Y+Yinc
        D := D+S-β
        Si D≤0
            alors Xs := X-Xinc
            sinon début
                    Si D+T-α+Dta<0
                        alors début
                                Si 2*D+T-α+Dta≤0
                                    alors Xs := X+Xinc
                                    sinon début
                                            Xs := X
                                            X := X+Xinc

                                            D := D+T-α+Dta
                                            T := T+Dtd
                                            S := S+Dsd
                                          fin
                              fin
                        sinon début
                                X := X+Xinc
                                Xs:= X+Xinc

                                D := D+T-α+Dta
                                T := T+Dtd
                                S := S+Dsd
                              fin
                  fin
        T := T+Dta
        S := S+Dsa

        Cp := ABS(D/T)
        Is := Cp*Imax
        Ip := Imax-Is

        Afficher point (X,Y,Ip);Afficher point (Xs,Y,Is)
    fin
    FIN TRACER-QUADRANT
```


Remarque : Cas d'une courbure très accentuée.

Ceci se produit lorsque Dxt1 ou Dxt2 sont très proches de Dxo ou Dx∞ c'est-à-dire lorsque A<<B ou B<<A ou encore lorsque A et B ont des valeurs numériques relativement faibles.

On constate dans ce cas que les intensités Ip et Is associées aux points où la courbure est très prononcée, ne sont pas correctes. En fait, ces intensités sont inversées (c'est-à-dire que Ip est associée à S et Is est associée à P).Ceci provient du fait que dans ce cas la quantité T ou S diminue rapidement (jusqu'à devenir nulle dans certains cas). Cette décroissance rapide n'a aucune incidence dans le cas de la version "escalier".

Il est possible de remédier à ce phénomène en modifiant le calcul de Ip et Is de la façon suivante :

```
Si T=0 (resp. S=0)
    alors Cp := 0
    sinon début
            Cp := |D/T| (resp.|D/S|)
            Si Cp>1/2 alors Cp := 1-Cp
          fin
Is := Cp*Imax
Ip := Imax-Is
```

2.9.3.3. Amélioration du tracé des arcs de parabole

L'arc de parabole est défini par ses extrémités de coordonnées (XA,YA), (XB,YB) et sa courbe d'équation $Y=A(X-XC)^2+B$ où (XC,B) est l'extremum de la courbe (voir figure 2.11., chap. 2.7.).

L'algorithme développé par ROY a la forme suivante :

```
Génération améliorée ARC DE PARABOLE    (XA,YA,XB,YB,XC : entier)
Début co méthode de ROY fco
    Si (XA<XC et XC<XB) ou (XB<XC et XC<XA)
        alors début
                Si ABS(XA-XC)=ABS(XB-XC)
                    alors lire YC
                    sinon DSXO := (XB-XA)*(XB+XA-2*XC)
                          CBO := (XB-XC)²*YA-(XA*XC)²*YB
                          YC := ARRONDI (CBO/DSXO)
                TRACER(XC,YC,XA,YA)
                TRACER(XC,YC,XB,YB)
              fin
        sinon début
                Si (XC≤XA et XA<XB) ou (XC≥XA et XA>XB)
                    alors TRACER(XA,YA,XB,YB)
                    sinon TRACER(XB,YB,XA,YA)
              fin
Fin
```

```
TRACER(XD,YD,XF,YF)
DX := XF-XD
DY := YF-YD

DELTAY := ABS(DY)
DT     := 2*DELTAY
DSX    := DX*(XF+XD-2*XC)

DXT := ARRONDI(DSX/DT)

Si DX>0
   alors XINC := +1
   sinon XINC := -1

Si DY>0
   alors YINC := +1
   sinon YINC := -1

X := XD
Y := YD
Afficher point (X,Y,Imax)

T := DT*ABS(X-XC)
D := 0

Si ABS(X-XC)<DXT
   alors début /* cas des tangentes de pente ε[0,1]*/
          BT := MIN(ABS(DX),-ABS(X-XC)+DXT)
          Pour I := 1 pas 1 jusqu'à BT faire
          Début
              X := X+XINC
              D := D+T+DELTAY
              Si D<=0
                 alors YS := Y-YINC
                 sinon début
                       Si D-DSX<0
                          alors début
                                Si 2*D-DSX<=0
                                alors YS := Y+YINC
                                sinon
                                    début
                                       YS := Y
                                       Y := Y+YINC
                                       D := D-DSX
                                    fin
                                fin
                          sinon début
                                    Y := Y+YINC
                                    YS := Y+YINC
                                    D := D-DSX
                                fin
                       fin
              T := T+DT
              Cp := ABS(D/DSX)
              Is := Cp*Imax
              Ip := Imax-Is
              Afficher point (X,Y,Ip)
              Afficher point (X,YS,Is)
          fin
   fin
```



```
DYT := ABS(YF-Y)
Si DYT ≠ 0
   alors début /* cas des tangentes de pente ∈]1,+∞[*/
          T := -T
          D := -D

          Pour I := 1 pas 1 jusqu'à DYT faire
          Début
              Y := Y+YINC
              D := D+DSX
              Si D<=0
                 alors XS := X-XINC
                 sinon début
                       Si D+T-DELTAY<0
                          alors début
                                Si 2*D+T-DELTAY<=0
                                alors XS := X+XINC
                                sinon
                                    début
                                       XS := X
                                       X := X+XINC
                                       D := D+T-DELTAY
                                       T := T-DT
                                    fin
                                fin
                          sinon début
                                    X := X+XINC
                                    XS := X+XINC
                                    D := D+T-DELTAY
                                    T := T-DT
                                fin
                       fin
              Cp := ABS(D/T)
              Is := Cp*Imax
              Ip := Imax-Is
              Afficher point (X,Y,Ip)
              Afficher point (XS,Y,Is)
          fin
   fin
FIN TRACER
```

2.9.4 - SIMULATION DE GRISE

2.9.4.1 - *Introduction*

Les algorithmes traitant les effets d'escalier sont conçus pour être appliqués à des écrans à luminosité variable.

Si nous ne disposons pas d'un tel matériel, nous avons recours à la simulation.

Parmi les diverses techniques existantes (méthodes des seuils bruités, de Floyd et Steinberg,...,voir <JJN 76>,<NeS 79>,<Bre 79>) nous retenons l'utilisation des cellules prédéfinies, celle-ci étant la plus adaptée au cas du dessin au trait et la plus facile à mettre en oeuvre.

2.9.4.2 - *Utilisation des cellules prédéfinies*

. *Le principe*

L'idée de base de cette méthode est de remplacer un point-écran par des points contigüs formant une matrice carrée nxn. Une conséquence non négligeable de ce procédé sera la diminution du pouvoir de résolution de l'écran.

La technique de cette méthode est la suivante :

- à chaque point (r,q) que l'on désire afficher est associée une intensité de luminosité In.

- l'affichage des n^2 points-écran correspondants s'effectue par comparaison de l'intensité I_n avec la valeur donnée par la matrice de référence L_n :

Si $I_n > L_n(i,j)$
alors le point-écran (3*r+i,3*q+2-j) est allumé
sinon ce point-écran est éteint.

Soit par exemple une image où les intensités varient entre 0 et 9 et que l'on veut représenter à l'aide de cellules 3x3. La matrice L3 pourra être par exemple :

$$\begin{bmatrix} 6 & 8 & 4 \\ 1 & 0 & 3 \\ 5 & 2 & 7 \end{bmatrix}$$

La figure 2.15 montre pour chaque intensité quelle cellule lui est associée :

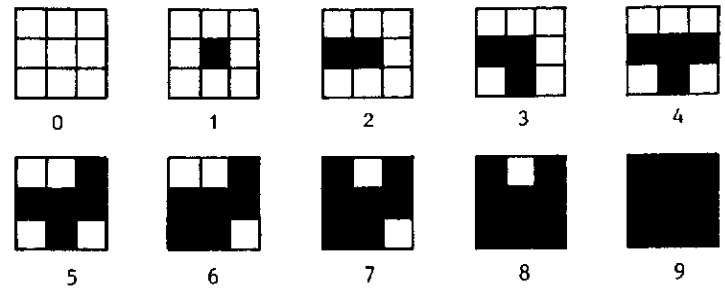


Figure 2.15 : Simulation des 9 niveaux d'intensité.

Un des problèmes que l'on rencontre en définissant une telle matrice est celui des figures parasites. En effet, certaines configurations de points dans une cellule peuvent donner un hachurage non désiré ou d'autres effets secondaires gênants. Une méthode connue sous le nom de "ordered dither" (<JJN 76>) donne un moyen d'obtenir des matrices L2, L4, L8,... satisfaisantes. Une matrice L_n est obtenue de la manière suivante :

$$L_n = \left[\begin{array}{c|c} 4\ L_m & 4\ L_m + 2\ U_m \\ \hline 4\ L_m + 3\ U_m & 4\ L_m + U_m \end{array}\right]$$

avec m=n/2 et U_m matrice carrée mxm dont tous les éléments sont égaux à 1.

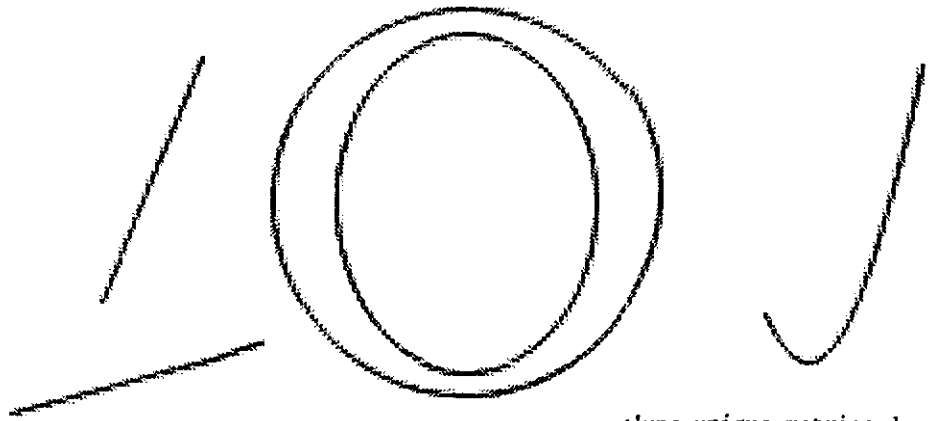
En notant L1=0, on obtient les matrices L2 et L4 suivantes :

$$L2 = \begin{bmatrix} 0 & 2 \\ 3 & 1 \end{bmatrix} \qquad L4 = \begin{bmatrix} 0 & 8 & 2 & 10 \\ 12 & 4 & 14 & 6 \\ 3 & 11 & 1 & 9 \\ 15 & 7 & 13 & 5 \end{bmatrix}$$

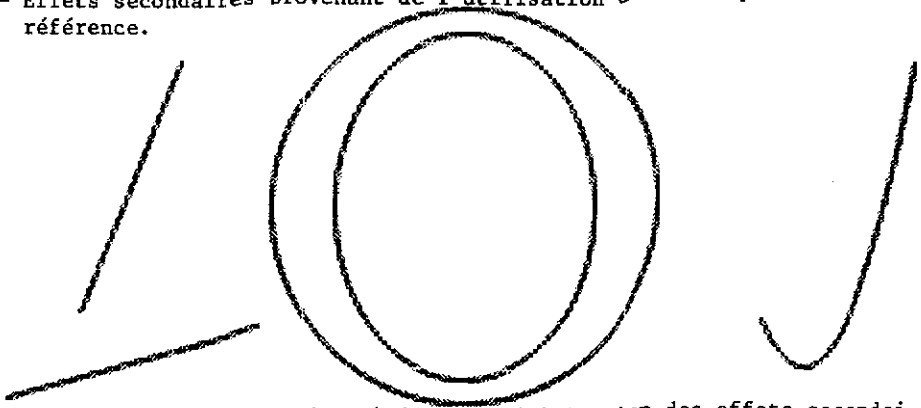
. *Elimination des effets secondaires*

De l'emploi d'une seule matrice résultent des effets de torsade et des effets qui détériorent la symétrie des courbes (notamment dans le cas des ellipses). La Figure 2.16.a montre les résultats peu concluants obtenus à partir de la matrice L.

$$L = \begin{bmatrix} 0 & 3 & 7 \\ 5 & 1 & 4 \\ 8 & 6 & 2 \end{bmatrix}$$



a- Effets secondaires provenant de l'utilisation d'une unique matrice de référence.



b-Simulation des niveaux de grisée après élimination des effets secondaires.

Figure 2.16.: Elimination des effets secondaires (tiré de <ROY83>).

Pour pallier cet handicap, ROY propose de tenir compte de l'"orientation des courbes tracées". En conséquence, nous passons de 1 à 4 matrices de référence, de telle sorte que chacune d'elles soit associée à une des 4 "orientations" possibles, à savoir :

$$L1=L=\begin{bmatrix}0 & 3 & 7\\ 5 & 1 & 4\\ 8 & 6 & 2\end{bmatrix}$$ associée aux 3ème et 4ème octants

$$L2=\begin{bmatrix}2 & 6 & 8\\ 4 & 1 & 5\\ 7 & 3 & 0\end{bmatrix}$$ associée aux 7ème et 8ème octants

$$L3=\begin{bmatrix}8 & 5 & 0\\ 6 & 1 & 3\\ 2 & 4 & 7\end{bmatrix}$$ associée aux 1er et 2ème octants

$$L4=\begin{bmatrix}7 & 4 & 2\\ 3 & 1 & 6\\ 0 & 5 & 8\end{bmatrix}$$ associée aux 5ème et 6ème octants

Mais ceci ne suffirait pas à préserver la symétrie des courbes. Aussi, nous utilisons le fait que les algorithmes de tracé utilisés nous donnent la position des points secondaires relativement aux points principaux. ROY a adopté la "stratégie" suivante (<ROY 83>) :

- au point principal P est associé l'indice de la matrice de référence correspondant à l'octant auquel il appartient (cet indice est noté I1b),

- au point secondaire S est associé l'un des 4 indices des matrices de référence, ceci en fonction de la position de S par rapport à P ainsi que de l'indice I1b.

Ces affectations s'effectuent de la manière suivante :

	3ème et 4ème octants	7ème et 8ème octants	1er et 2ème octants	5ème et 6ème octants
I1b indice associé à P	1	2	3	4
I1c S au-dessus de P	2	2	3	3
I1e S au-dessous de P	1	1	4	4
I1d S à droite de P	1	1	3	3
I1g S à gauche de P	2	2	4	4

Les mises à jour de I1b, I1c, I1e, I1d et I1g sont effectuées en même temps que celles des variables d'incrémentation des coordonnées Xinc et Yinc.

L'affichage des points suivra donc l'algorithme ci-dessous :

```
LUMINOSITE (ENTIER TAB(*,4),NBP,LI(3,3),L2(3,3),L3(3,3),L4(3,3))
Début
    ENTIER Xe,Ye,Xo,Yo
    ENTIER Lx(3,3),In,Il
    Pour I := 1 à NBP faire
    Début
        Xo := TAB(I,1)*3      /* TAB contient les coordonnées,
        Yo := TAB(I,2)*3+2       l'intensité et l'indice de la
        In := TAB(I,3)           matrice de référence des points
        Il := TAB(I,4)           à afficher */
        Suivant cas faire
        Début
            Il = 1 : Lx := L1
            Il = 2 : Lx := L2
            Il = 3 : Lx := L3
            Il = 4 : Lx := L4
        fin
        Ye := Yo
        Pour k := 1 à 3 faire
        Début
            Xe := Xo
            Pour J:=1 à 3 faire
            Début
              Si In>Lx(K,J) alors afficher(Xe,Ye)
              Xe:=Xe+1
            fin
            Ye:=Ye-1
        fin
    fin
Fin LUMINOSITE
```

Après élimination des effets secondaires, la simulation donne une représentation correcte de ce que serait l'image sur un écran à intensité variable. La figure 2.16.b montre l'amélioration obtenue après utilisation de la solution proposée initialement.

2.10. Conclusion

Sur la famille des courbes que nous venons d'étudier, nous pouvons conclure en disant que la méthode de BRESENHAM s'avère la plus avantageuse des techniques incrémentales :

- les points calculés sont les plus proches de la courbe théorique ;

- elle n'utilise qu'une arithmétique entière et les opérations d'addition et de soustraction pour le calcul des points ;

- sa complexité pratique est la plus intéressante.

Pour toutes ces raisons, il est donc intéressant de retenir la technique de BRESENHAM.

Cependant, si l'idée de BRESENHAM n'est pas généralisable pour toutes les familles de courbes, nous savons que la plupart d'entre elles peuvent être approchées par un ensemble de segments de droite, d'arcs de cercle ou d'arcs d'ellipse (voir "courbes évoluées" dans <SICO81> ou bien engendrées par l'une des méthodes incrémentales générales (JORDAN ou COHEN).

Différents auteurs ont généralement développé les techniques d'approximation polygonale des courbes et en particulier des coniques : on citera D. COHEN dans (<COHEN 71>) et ROGERS et ADAMS dans (<ROGERS-ADAMS 76>).

Chapitre 3

Remplissage de taches

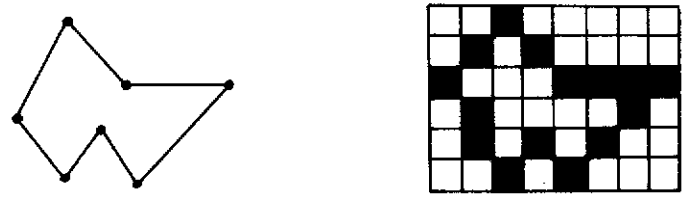
3.1. Introduction

Dans le terme générique de remplissage de taches, nous distinguons trois types de traitement :

- le coloriage,
- le remplissage proprement dit,
- le hachurage.

. Lorsque nous venons de dessiner point à point sur l'écran le contour d'une tache, l'idée première pour la peindre d'une certaine couleur est de la colorier à partir d'un point intérieur appelé germe ("seed"). Nous appelons cette technique le coloriage.

. Le remplissage de taches est une opération qui consiste à remplir une zone du plan dont la frontière est définie soit par un contour polygonal ou autre dans l'espace utilisateur (figure 3.1.a), soit par un contour préinscrit point par point dans une mémoire d'image représentant l'espace écran (voir figure 3.1.b).



a- Contour polygonal b- Contour défini point par point

Figure 3.1.: Représentations du contour.

Généralement, l'algorithme de remplissage se décompose en deux phases:

1- Préparation du contour : constitution d'une liste d'arêtes lorsque l'on raisonne dans l'espace utilisateur, ou inscription et codage du contour dans une matrice image représentant la surface de visualisation.

2- Remplissage proprement dit de la tache avec la couleur désirée.

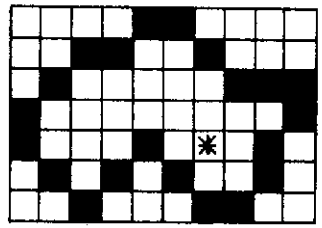
. Le problème du hachurage d'une tache et les solutions envisagées sont très proches voir semblables à certaines techniques de remplissage ligne par ligne et en particulier aux algorithmes de "suivi de contour". Par analogie avec ces méthodes, nous développerons une méthode de hachurage oblique sans rotation.

Dans ce qui suit, la notion de "contexte" attribuée à chaque ligne de balayage, est un ensemble de variables logiques, qui permet de connaître en tout point de la ligne certaines informations dont nous préciserons la nature pour chaque algorithme.

3.2. Le coloriage

3.2.1 - LE PRINCIPE DE BASE

Dans de nombreuses applications, le contour à remplir est fourni point par point en mémoire d'image. Le principe de base est ici le suivant : à partir d'un point intérieur à la tache qui sera connexe, il s'agit d'atteindre tous les points intérieurs à la tache qui sont "reliés" à ce point (voir figure 3.2.).



* point intérieur (x,y) ou germe ("Seed")

Figure 3.2.: Contour préinscrit et point intérieur.

L'algorithme de base est développé par SMITH, <SMITH 79> :

Ayant déterminé un point intérieur ou germe ('seed') de coordonnées (seedx,seedy) à l'intérieur de la tache, nous remplissons d'abord tout le segment horizontal auquel appartient ce point (c'est-à-dire tous les points de même couleur reliés entre eux et au germe, appartenant à la même ligne de balayage). Puis à partir de ce segment nous cherchons dessus et dessous les segments qui lui sont "reliés", c'est-à-dire possédant un point ('pixel') ayant un côté commun avec un des points du segment horizontal courant : c'est la propriété de la connexité. Ce point caractérise le nouveau segment et devient un nouveau germe. Pour chacun de ces nouveaux germes, nous répétons le processus décrit ci-dessus pour le premier jusqu'à ce que toute la tache soit remplie. Pour mémoriser et réutiliser les germes nous nous servons d'une pile.

La figure 3.3. montre les différentes étapes du remplissage décrit ci-dessus : les carrés noirs représentent le contour, l'étoile le point intérieur de départ, les points noirs indiquent les germes qui sont empilés.

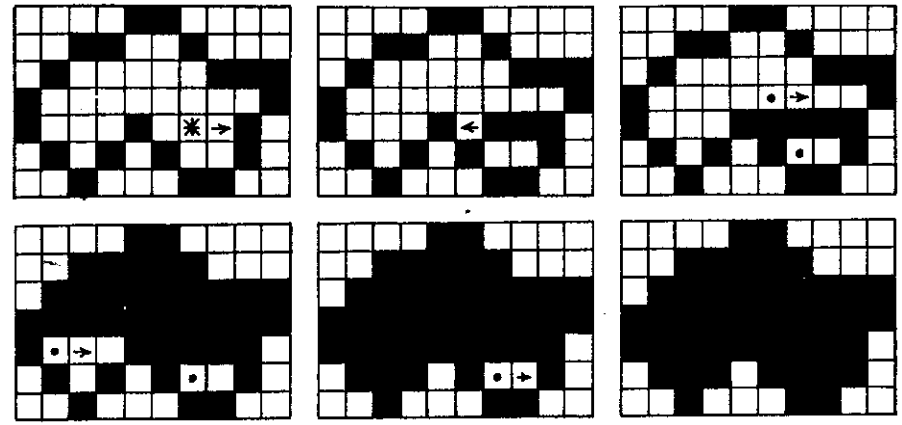


Figure 3.3.: Illustration du coloriage

Lors du coloriage, deux problèmes peuvent survenir :

. La génération du contour engendre parfois des points isolés (la tache n'est plus connexe) conduisant à des erreurs d'affichage plus ou moins importantes selon le nombre de ces points (voir figure 3.4.).

. La tache peut toucher les bords de l'écran provoquant pour certains algorithmes de coloriage (celui de PAVLIDIS en particulier) une sortie de la matrice de points. Nous envisageons deux solutions : soit tester à chaque fois si nous dépassons les limites de la mémoire, soit agrandir la mémoire sur les bords pour supprimer a priori tout risque d'erreur. Si la seconde idée nécessite plus de mémoire, elle n'implique aucun calcul supplémentaire.

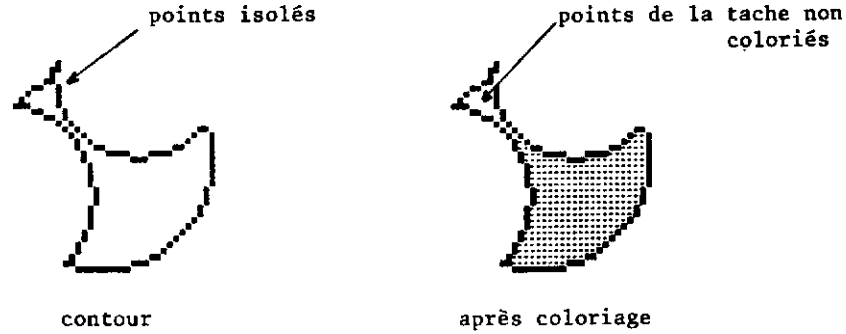


Figure 3.4.: Exemple de singularité rencontrée lors du coloriage (tache non connexe).

Deux idées ont été utilisées pour améliorer la complexité de l'algorithme de base :

- éviter l'exploration redondante des segments horizontaux contigüs (algorithme de SMITH, <SMITH 79>, et de LIEBERMAN, <LIEBERMAN 78>,

- modéliser la tache sous forme d'un graphe (algorithmes de SHANI, <SHANI 80>, et de PAVLIDIS <PAVLIDIS 81>).

Complexité :

Si NBI est le nombre de points intérieurs de la tache, et NBC le nombre de points du contour, la complexité de l'algorithme de PAVLIDIS est proportionnelle à NBI + 3 NBC. Les algorithmes de SMITH, LIEBERMAN ou SHANI ont une complexité proportionnelle à 3 NBI. Ainsi, si NBC < < NBI l'algorithme de PAVLIDIS s'avère a priori plus avantageux.

A cet effet, nous redonnons l'algorithme de base développé par SMITH, puis celui de PAVLIDIS.

3.2.2 - L'ALGORITHME DE SMITH

Pour éviter de rechercher systématiquement au-dessus et au-dessous de chaque segment horizontal ceux qui lui sont adjacents SMITH propose la solution suivante :

Si le segment horizontal courant Sc est adjacent au segment Sp précédemment traité, et si les extrémités de Sc sont à l'intérieur de celle de Sp, il n'existe pas de nouveaux segments au-dessus de Sc si Sc est situé dessous Sp (respectivement au-dessous de Sc si Sc est situé dessus Sp) qui lui sont "reliés" (voir figure 3.5.).

Figure 3.5.: positions relatives des segments adjacents Sp et Sc.

L'algorithme de SMITH a donc la forme suivante :

```
Algorithme SMITH(seedx,seedy,newpv)
Début
    x := seedx,y:=seedy
    new := newpv ; old := pvaleur(x,y)
    si old = new alors fin ;
    yref := y ; lxref := x, rxref = x ;
    Empiler(x,y) ;
    Tant que (pile non vide) faire ;
    Début
        Dépiler (x,y) ;
        Si pvaleur (x,y) ≠ new
            alors FILLINE
                  si HIVOISIN
                     alors SCANHI;
                     sinon si LOVOISIN
                              alors SCANLO;
                              sinon SCANHI ; SCANLO;
                           fsi
                  fsi
        fsi
        yref := y ; lxref = lx ; rxref := rx ;
    fin
Fin
```


Notations :
newpv = nouvelle valeur des points de la tache ;
old = ancienne valeur des points de la tache ;
(seedx,seedy)=coordonnées du point intérieur ;
lx et rx = extrémités gauches et droites des segments horizontaux ;
x min, x max et ymin, ymax sont les bornes de la mémoire d'image.

La fonction pvaleur(x,y) nous donne la valeur du point (x,y).

```
FILLINE co remplissage du segment horizontal fco
Début
    FILLRIGHT ; co remplissage à droite fco
    FILLLEFT  ; co remplissage à gauche fco
Fin ;

FILLRIGHT
Début
    sauver x ;
    Tant que (pvaleur(x,y)=old et x<xmax) faire
    Début
        pvaleur(x,y) := new ;
        x := x+1 ;
    fin
    rx := x-1 ;
    restaurer x ;
fin ;

FILLLEFT
Début
    sauver x ;
    x := x-1 ;
    Tant que (pvaleur(x,y) = old et x>xmin) faire
    Début
        pvaleur(x,y) := new ;
        x := x-1 ;
    fin
    lx := x+1 ;
    restaurer x ;
fin ;

SCANHI
Début
    si y+1 > ymax alors fin ;
    Sauver x et y ;
    x := lx ; y := y+1 ;
    Tant que x≤rx faire
    Début
        Tant que pvaleur(x,y) ≠ old et x≤rx faire
        Début x := x+1 fin ;
        Si x>rx alors fin ;
(1)     Empiler (x,y);
        Tant que pvaleur(x,y) = old et x≤rx faire ;
        Début x := x+1 fin ;
    fin ;
    Restaurer x,y ;
fin ;
```

```
SCANLO
Début
    si y-1 < ymin alors fin fsi
    Sauver x,y ;
    x := ℓx ; y := y-1 ;
    [1] de SCANHI ;
    Restaurer x et x ;
fin ;
```

La fonction logique HIVOISIN(LOVOISIN) vérifie si le segment horizontal courant est voisin du précédent, s'il est au-dessus (au-dessous) et détermine si aucun nouveau segment lui est relié au-dessous (au-dessus).

```
Logique HIVOISIN
Début
    si (y=yref+1 et ℓx≥ℓx ref-1 et rx≤rx ref+1)
       alors vrai ;
       sinon faux :
fin ;
```

On remplace yref+1 par yref-1 pour obtenir LOVOISIN .

Nous pouvons remarquer que cet algorithme de remplissage peut servir à remplir une tache monochrome déjà existante avec une nouvelle couleur ('newpv'), ou une tache définie par un contour préinscrit (avec la nouvelle couleur), dont l'intérieur est monochrome. Si l'intérieur du contour possède plusieurs couleurs cet algorithme est toujours utilisable après quelques modifications des tests sur la valeur des points.

3.2.3 - L'ALGORITHME DE PAVLIDIS

1- Le principe

Pour modéliser la tache, PAVLIDIS utilise deux graphes où les éléments sont les segments horizontaux formés de points ("pixels") de même couleur et où les arcs sont les couples de segments adjacents de même couleur. Ces deux graphes sont :

- celui associé au contour appelé C-LAG ("contour-line adjacency graph"),

- celui associé à l'intérieur du contour appelé I-LAG.

Si nous supposons que deux segments de l'intérieur sont reliés (ou adjacents) s'ils possèdent deux points ayant 1 côté commun, les segments du contour restent reliés même si ces 2 points n'ont qu'un sommet (ou coin) commun (voir figure 3.6.).

L'idée de l'algorithme de remplissage est d'utiliser le graphe C-LAG au lieu de I-LAG. La principale motivation de ce choix fut l'utilisation de structures identiques aux algorithmes de contrôle de parité et la possibilité d'améliorer la rapidité du remplissage.

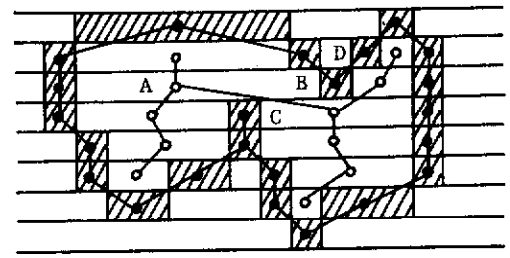


Figure 3.6.: Définition de C-LAG et de I-LAG.

A partir du noeud (segment horizontal) auquel appartient le point intérieur, nous définissons les germes des noeuds qui lui sont adjacents en examinant le contour à chacune de ses extrémités, puis ainsi de suite pour chaque segment déterminé (les germes sont mémorisés dans une pile).

Le nombre de noeuds du contour (ou degré) reliés au noeud du contour situé à l'une des extrémités du segment horizontal (noeud de I-LAG) permet de déterminer si nous sommes arrivés à un extrémum et de calculer les germes des segments de I-LAG adjacents au segment courant.

2- L'algorithme de PAVLIDIS

Avant d'exposer l'algorithme principal, nous devons définir les procédures auxiliaires.

. Les procédures auxiliaires :

Soit p définissant l'adresse (coordonnées (x,y) par exemple) d'un point de l'écran. L'expression p-1 nous donne le point situé à gauche de p sur la même ligne horizontale, et p+1 désigne le point situé à droite (figure 3.7.a).

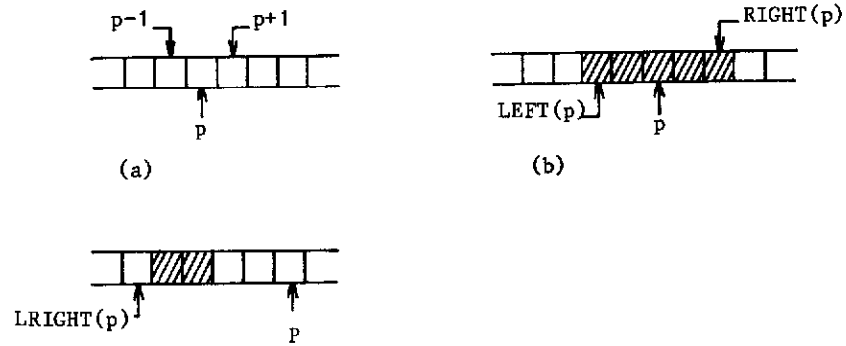


Figure 3.7.: Définitions.

La procédure LEFT(p) (resp$^{t}_{=}$ RIGHT(p)) nous délivre le point le plus à gauche (resp$^{t}_{=}$ le plus à droite) de l'intervalle de même couleur contenant p (figure 3.7.b).

On a aussi LRIGHT(p)=LEFT(p)-1)-1 (figure 3.7.c) qui calcule le point situé à gauche de p et de même couleur, immédiatement après une plage de couleur différente.

Les degrés du noeud (intervalle) d'un graphe, contenant le point p sont calculés par la procédure LINK (p,v) qui renvoie le vecteur v défini par : v=(a,b,p1,p2,e1,e2) où a et b sont les degrés au-dessus et au-dessous du noeud contenant p et p1,p2,e2 sont les points définis sur la figure 3.8. (si ces points ne sont pas définis on retourne la valeur NIL).

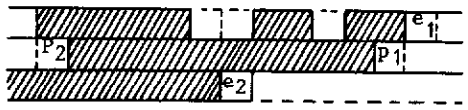


Figure 3.8.: Définition de p1,p2,e1,e2. Le point p ∈]p2,p1[

. Notations :

- pnext est le point qui sera utilisé pour remplir le prochain intervalle comme point de départ.
- Pother pourra avoir la valeur pabove ou pbelow et est le point situé immédiatement dessus ou dessous le point courant.
- empiler (S,p) signifie mettre le point p sur la pile S (p devient le sommet de la pile).
- pi est le point intérieur et 'couleur' est la couleur attribuée à la tache.

```
Algorithme de PAVLIDIS (pi,couleur)
Début
   co initialisation des piles fco
     si piabove ∈ contour
       alors LINK (piabove) ;
             si (a,b) ≠ (0,2)
                alors LINK (LEFT(pi)) ;
                      empiler (S,e1) ; Empiler(Sd,up);
             fsi
       sinon Empiler (S,LEFT(piabove) ; Empiler(Sd,up);
     fsi
     Empiler(S,LEFT(pi)) ; empiler(Sd,down);
   co Remplissage fco
     Tant que pile S non vide faire
     Début
          p next := dépiler(S) ; direction := dépiler Sd,Pright = X-max
          co RIGHT(P) fco
          si direction = down alors u := 2, other := below;
                              sinon u := 1, other := above;
          Répéter
              si (pnext=nil ou rempli) alors sortir de la boucle fsi
              p := pnext ; co pnext devient le germe courant fco
              LINK (p-1) ; co contour à gauche fco
          (1) si(direction=down et a>1) ou (direction=up et b>1)
                et p est à droite de pright alors sortir de la boucle fsi
          (2) si (a>0 et b>0)
                alors pnext := eu;
                sinon co extremum fco  si p non rempli alors
                      empiler (S,LEFT(LRIGHT(p))),empiler(Sd,-direction);
                      si (a=0 et b≠0 et direction=down) ou (a≠0 et b=0
                          et direction=up)
                          alors pnext := eu;
                          sinon si pother non rempli
                                alors pnext:=LEFT(pother);
                                sinon pnext:=nil;
                                fsi
                      fsi
          fsi
          Remplir la ligne de p à pright avec COULEUR;
          LINK(pright+1) ; co contour à droite fco
          (3) Si (a=0 ou b=0) et (p1 non rempli)
                alors empiler (S,p1)
                      si a>0 alors empiler(Sd,up) sinon empiler(Sd,down)
                      fsi
          fsi
          jusqu'à fin de boucle
     fin
fin
```


Remarques :

Dans l'algorithme de PAVLIDIS, le test (1) contrôle si nous avons atteint la fin de l'intérieur de la tache suivant la direction du balayage en cours. Si la direction est vers le bas et si a>1 alors nous avons atteint un "fond" (figure 3.9.a) où l'une des configurations montrées sur les figures 9aa et 9ab. On vérifie le premier cas si le point courant p est à droite de pright, et par conséquent hors du contour (sinon p est à l'intérieur).

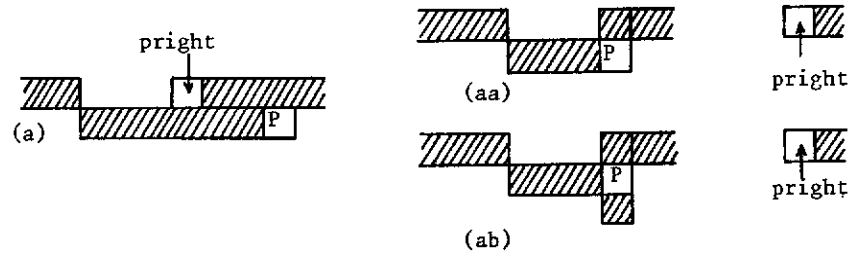
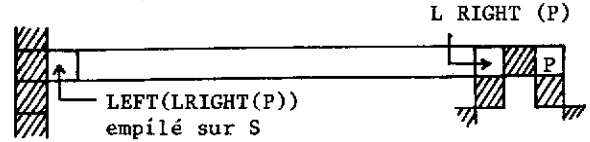


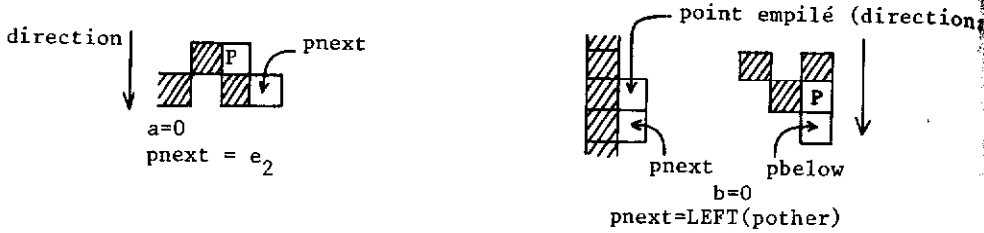
Figure 3.9.: Configurations détectées par (1).

En (2), nous vérifions si nous avons à faire à un extrémum, à gauche du germe p en cours de traitement :

Nous trouvons un exemple ci-dessous où a=0:



Le prochain point pnext est déterminé ci-dessous sur deux exemples :



En (3) nous traitons le cas où nous détectons un extrémum à droite du segment que l'on vient de remplir.

Pour la procédure LINK nous utilisons la couleur C du segment horizontal examiné donné par v(p)=couleur du point p.

```
LINK (p,(a,b,e1,e2,p1))
Début
      c := v(p);
      a := b := 0;
      e1:= e2 := p1 := nil;
      p := LEFT(p), pa := pabove ; pb := pbelow;
      si v (pa-1)=c alors a := a+1 fsi
      si v (pb-1)=c alors b := b+1 fsi
      tant que v(p)= c faire
      Début
          si (v(pa)=c et v(pa-1)≠c) alors a := a+1 fsi
          si (v(pb)=c et v(pb-1)≠c) alors b := b+1 fsi
          si (v(pa)≠c et v(pa-1)=c) alors e1:= pa fsi
          si (v(pb)≠c et v(pb-1)=c) alors e2:= pb fsi
      fsi
      si (v(pa)=c et v(pa-1)≠c) alors a := a+1 fsi
      si (v(pb)=c et v(pb-1)≠c) alors b := b+1 fsi
      si v(pa)=c alors e1 := RIGHT(pa)+1;
                 sinon si v(pa-1)=c alors e1 := pa ; fsi
      si v(pb)=c alors e2 := RIGHT(pb)+1;
                 sinon si v(pb-1)=c alors e2 := pb ; fsi
      p1 := p;
fin
```

3.2.4 - CONCLUSION

En conclusion, les algorithmes de coloriage ne remplissant que des taches connexes à partir d'un point intérieur (germe) souvent difficiles à déterminer automatiquement, sont avant tout utilisables de manière interactive.

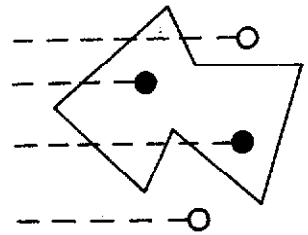
D'autre part, la comparaison de la complexité des méthodes nous a permis d'affirmer que l'algorithme de PAVLIDIS s'avère dans la plupart des cas, le plus avantageux, celui-ci raisonnant sur l'ensemble des points du contour plutôt que sur l'intérieur de la tache.

3.3. Le remplissage de taches

3.3.1 - INTRODUCTION

Le remplissage d'une tache monochrome est une opération qui consiste à remplir une zone du plan dont la frontière est définie par un ensemble de contours.

La bonne exécution de l'algorithme implique que l'on sache si on est à l'intérieur ou à l'extérieur de la tache. Sur une ligne joignant le point en partant de la gauche le nombre de frontières rencontrées indique si l'on est à l'intérieur du polygone ou à l'extérieur : c'est le "contrôle de parité" (figure 3.10).



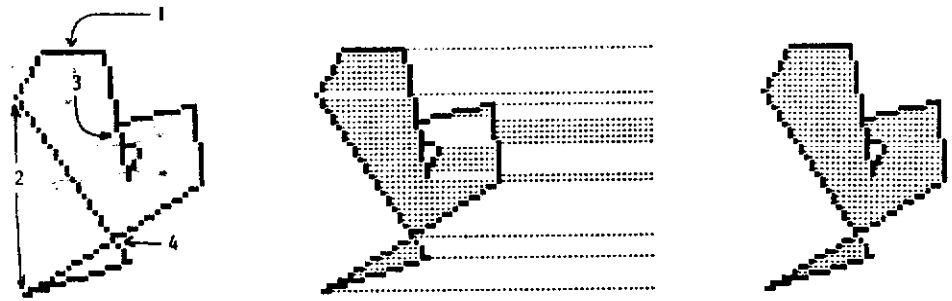
● point intérieur : nombre impair d'intersections.

○ point extérieur : nombre pair d'intersections.

Figure 3.10.: Contrôle de parité.

Mais l'application de cet algorithme donne des résultats erronés dans la majorité des cas, car il ne tient pas compte des singularités :

1- arêtes horizontales
2- sommets
3- recouvrements d'arêtes
4- points doubles, etc...



singularités du contour

erreurs de remplissage dues au "contrôle de parité"

remplissage correct

Figure 3.11.: Singularités rencontrées lors du remplissage.

Aussi, un certain nombre de solutions a été proposé pour remédier aux insuffisances de cet algorithme. Nous les avons classées suivant deux catégories. Nous distinguons :

- les techniques de "balayage ligne par ligne", basées essentiellement sur le codage du contour dans une matrice représentant la surface de visualisation : on raisonne donc dans l'espace image ;

- les techniques de "suivi de contour", pour lesquelles le remplissage de la tache est réalisé à partir d'une description structurée de son contour : tous les calculs se font dans l'espace utilisateur.

3.3.2 - LE BALAYAGE LIGNE PAR LIGNE

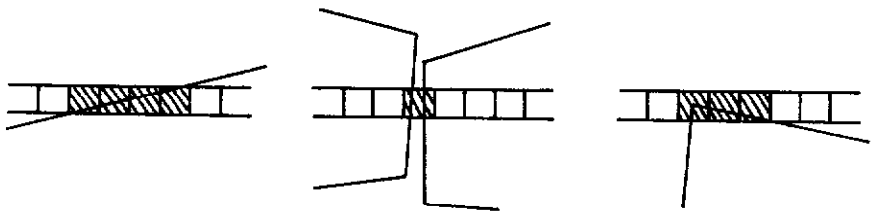
3.3.2.1 - *Introduction*

Les algorithmes de balayage par lignes utilisent tous une mémoire d'image dans laquelle s'effectuent le codage et le remplissage.

Le problème :

Si nous appliquons l'algorithme de parité sur le contour quantifié dans la matrice image, on obtient des résultats erronés dans la majorité des cas. Des erreurs interviennent dans les situations suivantes où la rencontre d'un point du contour n'implique pas de changement de contexte (à savoir si nous sommes à l'intérieur ou à l'extérieur de la tache) :

- la quantification engendre souvent plusieurs points successifs sur la même ligne (1)
- superposition d'arêtes, très voisines après quantification (2)
- certains sommets qui conservent le contexte (3).



(1) (2) (3)

Nous allons exposer les différentes techniques qui pallient ces singularités dans le cadre des taches polygonales, et la généralisation du balayage ligne par ligne aux taches quelconques.

3.3.2.2 - *Remplissage de taches polygonales*

1- *Classification des méthodes*

On peut regrouper les algorithmes existant en trois classes :

1- Codage des points des contours lors de la quantification

Chaque arête est engendrée point par point dans une matrice image, mais la perte d'information est compensée par un codage qui indique les changements de contexte. Puis on applique l'algorithme de parité.

Nous trouvons dans cette classe deux algorithmes :

. Celui de LUCAS, <LUCAS 77>, qui utilise une adaptation de son algorithme de génération de segments de droite pour engendrer et coder les arêtes en fonction des singularités rencontrées.

. Celui d'ACKLAND et WESTE, <ACKLAND-WESTE 81>, où pour la quantification des arêtes ils ne conservent qu'un seul point par ligne de balayage, ce qui entraîne une légère déformation du contour.

2- Inversions successives de l'image

Dans cette solution, définie par MARTINEZ, <MARTINEZ 82>, la quantification d'une arête s'accompagne d'une inversion des points situés à droite. La quantification ayant lieu au cours du remplissage la perte d'information demeure sans conséquence (voir figure 3.12).

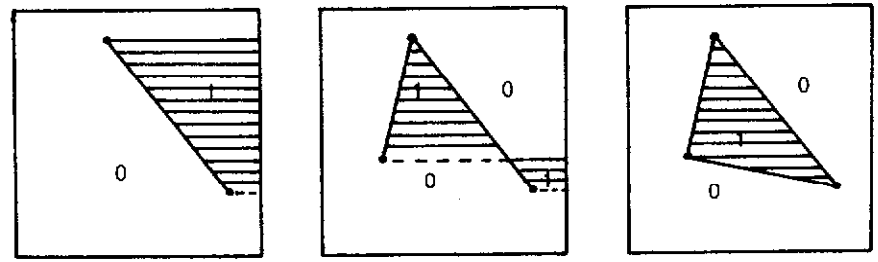


Figure 3.12: Remplissage d'un triangle par inversions successives de la mémoire.

3- Remplissage ligne par ligne d'un contour préinscrit point par point en mémoire d'image

Pour chaque ligne, lorsque nous rencontrons les points du contour, l'analyse du changement de contexte, à savoir si nous rentrons ou sortons de la tache, se fait à partir des points du contour des lignes au-dessus et au-dessous qui leur sont adjacents. PAVLIDIS a développé dans <PAVLIDIS 79> trois algorithmes de ce type qui tentent successivement de surmonter les points doubles, triples,... mais en vain dans certains cas. Ces algorithmes appartiennent plus au traitement d'image (reconnaissance du contour) qu'à la synthèse d'image, nous ne les inclurons donc pas dans cet ouvrage.

Comparaison des algorithmes

Si les algorithmes de LUCAS et de ACKLAND et WESTE sont similaires, la comparaison de leur complexité pratique permet de constater que la méthode de ACKLAND et WESTE est plus avantageuse. En effet, l'utilisation d'un seul code (1) au lieu de deux (1 et 2) engendre moins d'opérations (tests, accès tableau en particulier). Cependant, si ce codage du contour améliore la vitesse d'exécution, elle détériore l'aspect du contour et entraîne des effets désastreux au niveau de l'esthétique (problèmes de raccordements entre les taches par exemple).

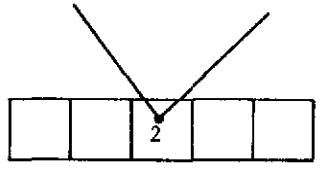
Le principal inconvénient de l'algorithme de MARTINEZ concerne le grand nombre de points traités inutilement. MARTINEZ argue dans <MARTINEZ 82> qu'une réalisation câblée, pour laquelle l'opération d'inversion est quasi-instantanée fait disparaître cet inconvénient.

2- L'algorithme de LUCAS

Les difficultés rencontrées sont surmontées de la façon suivante :

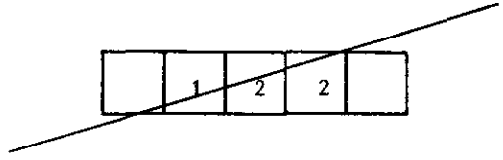
a- Segments horizontaux : on utilise un code disant qu'il faut colorier le segment mais ne changeant pas le contexte (code 2) ;

b- Sommet commun à deux côtés situés du même côté de l'horizontale passant par ce sommet : code impliquant un coloriage mais pas de modification du contexte (code 2) ;



c- Recouvrements : dus soit au croisement de deux côtés, soit à la précision moindre de la projection. Si ce nombre de recouvrements est pair, modifier le contexte, sinon ne pas le changer.

d- Segments horizontaux créés par l'approximation point par point : ne noter que les changements d'ordonnée (code 1) :



L'algorithme de LUCAS qui suit traite ces difficultés.


```
TACHE(entier XMIN,XMAX,YMIN,YMAX ; entier tableau X,Y,Z(*) ; entier COULEUR)
début {remplissage d'une tache inscrite dans le rectangle XMIN,YMIN,XMAX,
       YMAX les coordonnées des sommets de contour sont dans X,Y,Z ; la
       fonction SOMMETSUIVANT fournit l'indice d'un point de contour lors-
       qu'on suit le bord de la tache}
 logique UPL,UPM,POINTE ; entier L,LP,M;
 logique INTERIEUR ; logique ILYASOMMET;
 {initialisation de la couleur de fond}
 pour I:=XMIN pas 1 jusqua XMAX faire
 pour J:=YMIN pas 1 jusqua YMAX faire
 image(I,J):=0;

 {inscription du périmètre}
 LP:=PREMIER SOMMET ; L:=SOMMETSUIVANT;
 tantque Y(LP)=Y(L) faire {sauter les côtés horizontaux}
 début
     LP:=L;
     L:=SOMMETSUIVANT
 fin;

 {parcours du contour de la tache}
 UPL:=Y(LP)<Y(L) ; LP:=L;   {sommet de départ}
 ILYASOMMET:=vrai;
 tantque ILYASOMMET faire
 début
     M:=SOMMETSUIVANT;
     si Y(L)=Y(M)
     alors début    {segment horizontal}
           PA:=si X(L)<X(M) alors +1 sinon -1;
           pour I:=X(L) pas PA jusqua X(M) faire IMAGE(I,Y(L)):=2
           fin
     sinon début
           UPM:=Y(L)<Y(M) ; POINTE:=(UPM≠UPL);
           SEGMENT(X(L),Y(L),X(M),Y(M),POINTE)
           fin;
     L:=M ; ILYASOMMET:=(M≠LP)
 fin;

 {inscription de la surface}
 pour J:=YMIN pas 1 jusqua YMAX faire
 début
     INTERIEUR:=faux;
     pour I:=XMIN pas 1 jusqua YMAX faire
     début
         si IMAGE(I,J)=1  {point de contour}
         alors début
                 INTERIEUR:=¬ INTERIEUR;
                 IMAGE(I,J):=COULEUR
               fin
         sinon si INTERIEUR ou (IMAGE(I,J)=2)
               alors IMAGE(I,J):=COULEUR
     fin
 fin
fin TACHE;
```

```
SEGMENT(entier XA,YA,XB,YB ; logique POINTE);
début {tracé d'un segment en préparation au remplissage d'une tache}
      entier DELTAX,DELTAY,XINCR,YINCR,CUMUL,X,Y,YPREC;
      X:=XA ; Y:=YA ; (X,Y) point courant
      si POINTE {traitement d'un cas b}
      alors IMAGE(X,Y):=(si IMAGE(X,Y)≠1 alors 2 sinon 1)
      sinon IMAGE(X,Y):=(si IMAGE(X,Y)=1 alors 2 sinon 1);
      XINCR:=si XA<XB alors +1 sinon -1;
      YINCR:=si YA<YB alors +1 sinon -1;
      DELTAX:=ABS(XA-XB) ; DELTAY:=ABS(YA-YB) ; YPREC:=Y;
      si DELTAX>DELTAY
      alors début {traiter les cas d}
            CUMUL:=DELTAX/2;
            pour I:=1 pas 1 jusqua DELTAX-1 faire {pas le dernier point}
            début
              X:=X+XINCR ; CUMUL:=CUMUL+DELTAY;
              si CUMUL≥DELTAX
              alors début
                      CUMUL:=CUMUL-DELTAX;
                      Y:=Y+YINCR
                    fin;
              si (YPREC=Y) ou (Y=YB)
              alors début {même Y}
                      si IMAGE(X,Y)≠1 alors IMAGE(X,Y):=2;
                    fin
              sinon début {nouvel Y}
                      IMAGE(X,Y):=(si IMAGE(X,Y)=1 alors 2 sinon 1);
                      YPREC:=Y
                    fin
            fin
          fin
      sinon début {pas de problème si on se déplace en y}
            CUMUL:=DELTAY/2;
            pour I:=1 pas 1 jusqua DELTAY faire
            début
              Y:=Y+YINCR ; CUMUL:=CUMUL+DELTAX;
              si CUMUL≥DELTAY
              alors début
                      CUMUL:=CUMUL-DELTAY;
                      X:=X+XINCR
                    fin;
              IMAGE(X,Y):=(si IMAGE(X,Y)=1 alors 2 sinon 1)
            fin
          fin
fin segment
```


3.3.2.3 - Le remplissage de taches non-polygonales

Nous supposons que la tache est définie par un ensemble de contours non polygonaux. Chaque contour peut être décrit de deux façons différentes : soit par la liste ordonnée de primitives d'affichage comme les segments de droite, les arcs de cercle, soit par la liste ordonnée des points qui le constituent. C'est-à-dire que nous avons une structure d'anneaux.

Nous généralisons les algorithmes de LUCAS et de ACKLAND et WESTE en codant les points engendrés par le contour dans une matrice image puis en appliquant l'algorithme du contrôle de parité.

Les singularités à surmonter sont :
- les segments horizontaux,
- les extréma,
- les recouvrements, etc...

L'idée :

Pour chaque contour, le codage se fait pas à pas en analysant la position du point précédent par rapport au point courant qui vient d'être calculé.

Soient (xp,yp) les coordonnées du point précédent et (xc,yc) celles du point courant.

Soit IMAGE(*,*) la matrice image initialisée à "0".

Les variations des ordonnées permet de détecter les segments horizontaux et les extréma.

Nous utiliserons deux codes :
- "1" pour coder les changements de contexte,
- "2" pour coder les points du contour qui n'indiquent pas de changement de contexte.

Un seul code peut suffire, mais comme dans l'algorithme d'ACKLAND et WESTE la tache obtenue est légèrement déformée, ce qui engendre des problèmes de raccordement.

Les règles de la codification sont les suivantes :

On note ⊕ l'opération IMAGE(x,y)⊕ Code défini par la table qui suit :

Image \ Code	⊕1	⊕2
0	1	2
1	2	1
2	1	2

1- yp = yc (plage de points horizontaux)
IMAGE (xc,yc):=IMAGE(xc,yc)⊛2

2- yp<yc (courbe croissante)
IMAGE(xc,yc):=IMAGE(xc,yc)⊛1

3- yp>yc (courbe décroissante)
IMAGE(xp,yp):=IMAGE(xp,yp)⊛1
IMAGE(xc,yc):=IMAGE(xc,yc)⊛2

La codification rétrograde du point précédent lorsque la courbe décroît ainsi que l'opération ⊛, permettent de résoudre les problèmes dûs aux extrema. L'opération ⊛ supprime également les erreurs engendrées par les recouvrements (voir figure 3.13).

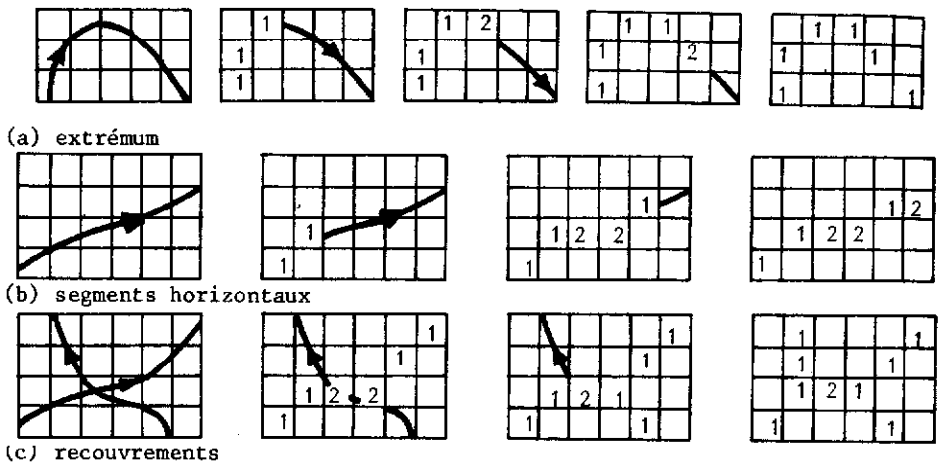
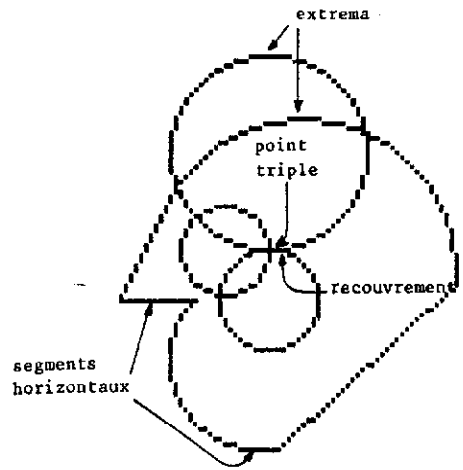


Figure 3.13 : Codage du contour.

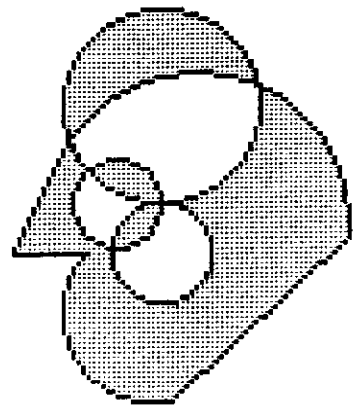
Remarque :

Le bon fonctionnement de l'algorithme suppose que la suite des points soit continue pour chaque contour (structure d'anneaux). Si le premier et le dernier point du contour sont différents, il faut coder le segment (dernier point, premier point).

Le remplissage proprement dit est réalisé en parcourant ligne par ligne la matrice codée IMAGE et en utilisant le contrôle de parité (voir Algorithme de LUCAS).



. Contour et singularités



. Remplissage correct

EXEMPLE : Illustration du remplissage d'une tache non polygonale par notre algorithme de codage du contour.

3.3.2.4 - *Conclusion*

En conclusion, le principal reproche que l'on puisse faire aux algorithmes de balayage par ligne concerne la nécessité de disposer d'une mémoire booléenne de travail. Nous devons nuancer cet argument. D'une part, il est possible de travailler directement dans la mémoire de trame du synthétiseur câblé si les taches ne se chevauchent pas. D'autre part, le prix de revient des composants va sans cesse décroissant.

Les méthodes de suivi de contour que nous allons développer dans le chapitre suivant pallient cet handicap.

3.3.3 - LE SUIVI DE CONTOUR

3.3.3.1 - *Le principe*

Il s'agit d'utiliser une description structurée du contour permettant de différencier les bords gauches des bords droits. On remplit la tache en déplaçant le long de ces bords les extrémités d'un segment de droite qui permet de peindre l'intérieur (voir figure 3.14).

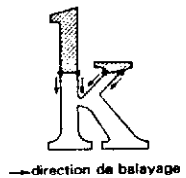


Figure 3.14: Affichage par suivi de contour.

On travaille ici uniquement dans l'espace utilisateur et au remplissage on ne sort pas de la tache.

3.3.3.2 - *Le remplissage des taches polygonales*

Deux variantes existent. La première consiste à calculer et à ordonner toutes les intersections du contour avec les lignes de balayage avant le remplissage : c'est l'algorithme YX présenté par NEWMAN et SPROULL dans <NEWMAN-SPROULL79>. La seconde solution que nous allons développer, permet de réaliser les calculs d'intersection et l'affichage ligne par ligne.

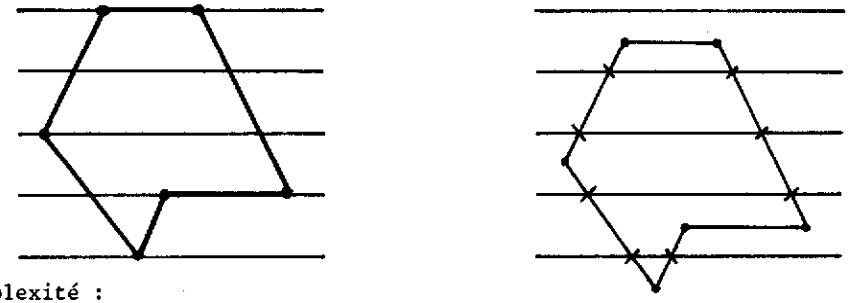
1- L'algorithme YX

1. Pour chaque arête du polygone, on calcule toutes les intersections de ce côté avec les lignes de balayage. A chaque calcul d'intersection, on enregistre ses coordonnées (x,y) dans une liste d'intersections.

2. On trie la liste d'intersections de telle façon que (x_1,y_1) précède (x_2,y_2) si et seulement si $(y_1>y_2$ ou $(y_1=y_2$ et $x_1<x_2))$.

3. On effectue le remplissage ligne par ligne en affichant pour chaque ligne de balayage y les segments $((x_{2k-1},y)(x_{2k},y))$ (où pour l'ensemble de la liste k=nbre d'intersections/2).

Pour éviter les erreurs dûes aux singularités (segments horizontaux, arêtes consécutives), NEWMAN et SPROULL préconisent de translater les ordonnées des sommets d'une demi unité comme suit :



Complexité :

Soients nbl le nombre de lignes de balayage coupant le contour, nbam le nombre moyen d'arêtes qui coupent chaque ligne (nbam=2 si le contour est convexe) et nba le nombre d'arêtes.

La mémoire nécessaire est proportionnelle à (nba+nbl*nbam) et la complexité des tris est en O ((nbl*nbam)Log(nbl*nbam)+nbl(nbam Log nbam)).

Nous constatons que pour des polygones importants, la mémoire nécessaire pour contenir la liste des intersections et le temps passé pour le tri de cette liste deviennent importants et coûteux. Aussi, d'autres algorithmes sont plus avantageux.

2- L'algorithme de suivi de contour

On remarque dans l'algorithme YX qu'à un moment donné nous n'avons besoin de connaître que les intersections des arêtes avec la ligne en cours de balayage. La solution consiste à conserver les contours non pas sous la forme d'une liste d'intersections mais par une liste d'arêtes. La quantification des arêtes se fait alors ligne par ligne au fur et à mesure du remplissage.

Pour chaque ligne, l'algorithme est le suivant :
1. Détermination de l'ensemble des arêtes qui coupent la ligne ;
2. Calcul des intersections ;
3. Tri des intersections ;
4. Affichage des segments horizontaux.

. Pour éviter de refaire tous les calculs à chaque ligne, il est intéressant de tirer parti de la cohérence du polygone qui existe entre deux lignes consécutives. Si Ei est l'ensemble des arêtes du contour qui coupent la ligne i alors

$E_{i+1}=E_i-$ {arêtes qui finissent sur la ligne i}+{arêtes qui commencent sur la ligne i+1}

Si x_i est l'abscisse de l'intersection d'une arête avec la ligne i, alors $x_{i+1}=x_i$ + Dxy où Dxy est l'accroissement des abscisses de l'arête entre les deux lignes i et i+1.

Nous utilisons deux structures de données pour mettre en oeuvre l'algorithme. Pour chaque ligne de balayage qui coupe le contour, nous constituons la liste des arêtes orientées vers le bas qui commencent sur celle-ci : c'est la structure du contour. Au cours du remplissage, nous établissons la liste E_i des arêtes qui coupent la ligne de balayage courante, en les ordonnant selon l'ordre croissant des abscisses des points d'intersection.

. Mais les problèmes dûs aux singularités se posent toujours. Des erreurs se produisent quand se trouvent sur la ligne de balayage :

- deux arêtes consécutives (ligne i_1 de la figure 3.15)
- certaines arêtes horizontales (ligne i_2 et i_3 de la figure 3.15).

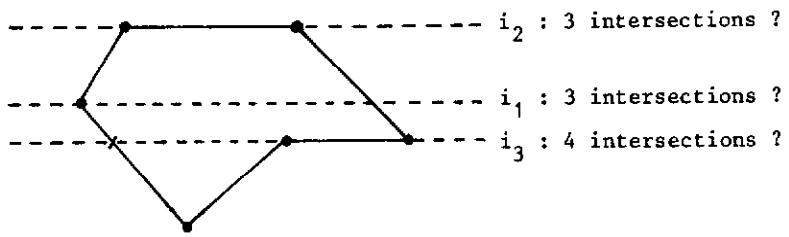
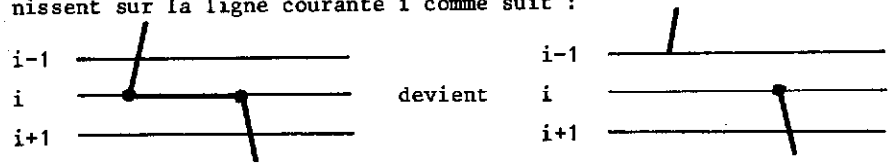


Figure 3.15.: Singularités rencontrées par l'algorithme de suivi de contour.

Pour y remédier, nous proposons de supprimer les arêtes horizontales de la liste des arêtes et de supprimer de la liste E_i les arêtes qui finissent sur la ligne courante i comme suit :



Si aucune des arêtes du contour ne se croise, nous pouvons supprimer le tri de la liste E_i en ordonnant dans la structure du contour les arêtes dans l'ordre croissant des abscisses de leur première extrémité et à abscisses égales selon l'ordre croissant des Dxy. L'ajout d'une arête dans la liste E_i deviendra alors une insertion qui respectera cet ordre.

. Mise en oeuvre :

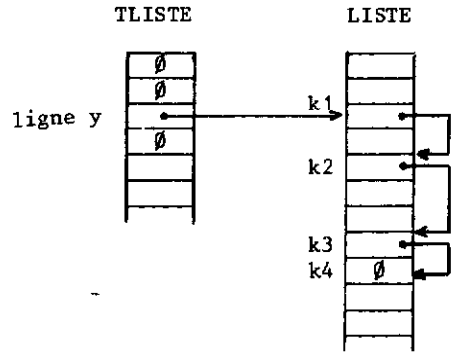
Pour réaliser la mise en oeuvre de l'algorithme de suivi de contour nous utilisons les structures de données suivantes :

. Le contour de la tache est formé d'un ensemble de polygones quelconques décrits par les suites ordonnées de leurs sommets. Nous disposons d'un tableau T(*) d'entiers contenant les coordonnées des sommets où pour chaque polygone la liste des sommets se termine par la valeur NIL. La variable N représente le nombre d'éléments contenus dans T(*).

. Liste des arêtes :

Chaque arête Ak sera définie par son abscisse début Axd(k), son ordonnée fin Ayf(k) et son accroissement des abscisses par rapport aux ordonnées Dxy(k)(réel).

Pour chaque ligne de balayage y, nous disposons de la liste des arêtes qui y commencent comme suit :



TLISTE(y)=Ø : liste vide, sinon la liste des arêtes qui commencent sur la ligne y est égale à $\{k_1,k_2,..,k_m\}$

. Liste des arêtes qui coupent la ligne de balayage courante :

Les numéros des Nbai arêtes coupant la ligne y seront contenus dans le tableau d'entiers Ei(*) et les abscisses des points d'intersection seront mémorisées dans le tableau de réels INTER(*).

L'algorithme peut alors s'écrire :

```
Remplissage SUIVI DE CONTOUR  (N,T(*):entier)
Début
    Constitution de la liste des arêtes;
    Remplissage;
Fin

Constitution de la liste des arêtes
Début
    Pour chaque arête a_k faire
    Début
        si a_k horizontale
            alors ne pas la prendre en compte;
            sinon l'orienter vers le bas et l'insérer dans la liste des
                  arêtes;
        fsi
    fin
Fin

Remplissage
Début
    ligne i := première ligne de balayage coupant le contour;
    Créer listes (INTER,E_i);

    REPETER
        trier(INTER,E_i), INTER étant le maître;
        Tracer les segments horizontaux(de INTER(2k-1)à INTER(2k));
        Ligne de balayage suivante : i=i+1;
        si ligne i ≠ dernière ligne de balayage
            alors enlever de (E_i,INTER) les arêtes qui finissent sur la li-
                  gne i ;
                  calcul des intersections avec la ligne i des arêtes ∈ E_i;
                  ajouter à (E_i,INTER) les arêtes qui commencent sur la
                  ligne i;
            sinon fin;
        finsi
    jusqu'à fin
```

Développons maintenant chaque module de l'algorithme général.

```
Constitution de la liste des arêtes
Début
    Initialiser Ymin et Ymax ; co utiliser respectivement le plus grand
    et le plus petit entier fco
    k := 0 ; co compteur d'arêtes non horizontales fco
    i := 1 ; co indice parcourant T(*) fco
    Tantque i<N faire
    Début
        i0 := i ; co premier sommet du polygone fco
        Xd := T(i) ; Yd := T(i+1);
        i := i+2 ; X:= T(i);
        Fin := faux;
        Répéter
            si X≠NIL alors Xf:=X ; Yf:=T(i+1);
                     sinon Xf:=T(i0) ; Yf:=T(i0+1)
                           fin:=vrai ; i:=i+1;
            co traiter l'arête ((Xd,Yd),(Xf,Yf)) fco
            si Yd≠Yf
               alors k:=k+1 ; co arête non horizontale fco
                    si Yd>Yf co on oriente l'arête vers le bas fco
                       alors Axd(k):=Xd ; Ayf(k):=Yf;
                             l:=Yd;
                             Dxy(k):=(Xf-Xd)/(Yd-Yf);
                       sinon Axd(k):=Xf ; Ayf(k):=Yd;
                             l:=Yf;
                             Dxy(k):=(Xd-Yf)/(Yf-Yd),
                    fsi
                    co insérer l'arête dans TLISTE fco
                    j:=TLISTE(l);
                    si j:=0
                       alors TLISTE(l):=k;
                       sinon Tantque LISTE(j)≠0 faire j:=LISTE(j) fait
                             LISTE(j):=k
                    fsi
                    si Ymax<Yf alors Ymax:=Yf fsi
                    si Ymin>Yf alors Ymin:=Yf fsi
            fsi
            co arête suivante fco
            si non(Fin) alors Xd:=Xf ; Yd:=Yf;
                              I:=I+2 ; X:=T(I);
        Jusqu'à Fin co fin du polygone fco
    fintq
Fin
```

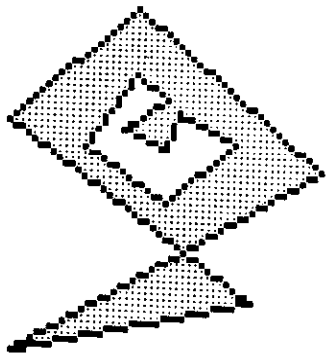
Nous parcourons les lignes de balayage selon l'ordre décroissant des ordonnées. Les autres modules s'écrivent par conséquent de la façon qui suit :

. Première ligne de balayage : $y:=Ymax$;
. Ligne de balayage suivante : $y:=y-1$;
. Dernière ligne de balayage : $y \leq Ymin$;


```
. Créer listes (Ei,INTER)
Début
    Nbai:=0 ; co nombre d'arêtes coupant la ligne de balayage fco
    j:=TLISTE(y);
    Tantque j≠0 faire
    Début
        Nbai:=Nbai+1;
        Ei(Nbai):=j;
        INTER(Nbai):=Axd(j);
        j:=LISTE(j);
    fin
Fin créer;

. Tracer les segments horizontaux
Début
    k1:=1;
    Tantque k1<Nbai faire
    Début
        k2:=k1+1;
        afficher segment((INTER(k1),Y),(INTER(k2),Y));
        k1:=k2+1;
    fin
Fin Tracer;

. Enlever les arêtes qui finissent sur la ligne y
Début
    ki:=1;
    Tantque ki<=Nbai faire
    Début
        k:=Ei(ki);
        si Ayf(k):=y
          alors j:=ki+1;
                Tantque j≤Nbai faire
                Début
                    Ei(j-1):=Ei(j);
                    INTER(j-1):=INTER(j);
                    j:=j+1;
                fin
                Nbai:=Nbai-1 ; ki:=ki-1;
        fsi
        ki:=ki+1;
    fin
Fin

. Calcul des intersections
Début
    Pour k:=1 jusqu'à Nbai faire INTER(k):=INTER(k)+Dxy(Ei(k));
Fin;
```

```
. Ajouter les arêtes qui commencent sur la ligne Y
Début
     j:=TLISTE[y];
     Tantque j≠0 faire
     Début
          Nbai:=Nbai+1;
          Ei[Nbai]:=j;
          INTER[Nbai]:=Axd[j];
          j:=LISTE[j];
     fin
Fin ajout;
```

Voici un exemple de tache engendrée par la méthode du suivi de contour :



3- Comparaison des algorithmes

Soient nbl le nombre de lignes de balayage qui coupent le contour polygonal, nbam le nombre moyen d'arêtes qui coupent chaque ligne de balayage (nbam=2 si le contour est convexe) et nba le nombre d'arêtes.

La mémoire nécessaire pour l'algorithme YX est proportionnelle à (nba+nbl*nbam), au lieu d'être proportionnelle en ce qui nous concerne à (nba+nbl+nbam).

La complexité du tri des intersections est en O((nbl*nbam)Log(nbl*nbam)) +nbl(nbam Log(nbam))) pour l'algorithme YX et en O(nbl(nbam Log(nbam))) pour notre version.

Cependant, la complexité de l'algorithme YX peut être améliorée si pour chaque ligne de balayage le tri et l'affichage sont effectués en parallèle. D'autre part, la complexité pratique des tris exécutés dans notre solution est inférieure à la complexité théorique. En effet, la liste E_i n'est plus ordonnée uniquement lorsque l'on ajoute une arête ou bien quand deux arêtes se sont croisées. Comme nbam n'est pas très grand et en utilisant un algorithme de tri simple comme le "tri à bulles", la complexité du tri sera en O(Nbam) pour toutes les lignes de balayage où un tel changement ne s'est pas produit.

3.3.3.3 - Le remplissage d'un ensemble de taches polygonales

1. Le problème

Il s'agit de remplir un ensemble de taches polygonales de telle sorte que le résultat final respecte l'ordre des priorités attribuées à chacune d'elle (voir figure 3.16).

Priorités :
A:1
B:2
C:4
D:3

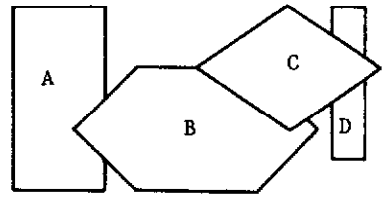
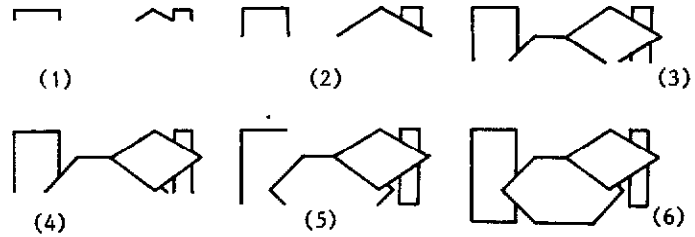


Figure 3.16: L'ensemble des taches et leurs priorités d'affichage

La première solution consiste à remplir chaque tache une par une en suivant l'ordre des priorités :

La seconde solution effectue le remplissage de toutes les taches simultanément et ligne par ligne en respectant l'ordre des priorités :



2. L'algorithme de suivi de contour

En utilisant les mêmes structures de données que précédemment (3.3.3.2) nous constituons pour chaque ligne de balayage la liste ordonnée des arêtes de toutes les taches polygonales qui la coupent suivant l'ordre croissant des abscisses des intersections puis à partir de celle-ci nous affichons les segments horizontaux en résolvant les priorités d'affichage des taches. Pour ce faire, nous devons attribuer à chaque arête son numéro de priorité correspondant.

Dans le cas particulier du remplissage simultané d'un ensemble de taches contigües (mêmes niveaux de priorité), E. SECHER a démontré dans <SECHER 83> que la complexité obtenue était plus avantageuse que celle engendrée par le remplissage successif de chaque tache par l'algorithme de

suivi de contour.

3.3.3.4 - *Le remplissage de taches non polygonales*

Nous supposons que les contours de la tache à remplir sont décrits par un ensemble de primitives d'affichage (segment de droite, arcs de cercle, ...).

La génération de la tache doit se faire en une seule passe suivant les ordonnées décroissantes. Pour cela, nous découpons les contours en un ensemble de bords gauches et droits appelés parois. Chaque paroi est une portion de contour monotone en Y et comprise entre un maximum local, ou point de naissance, et un minimum local, ou point de mort (voir figure 3.17).

Comme pour l'algorithme de suivi de contour développé pour les taches polygonales (chapitre 3.3.3.2), le remplissage s'effectue ligne par ligne en tenant parti de la cohérence entre deux lignes de balayage consécutives pour calculer les intersections et la liste des parois qui coupent la ligne. Les singularités sont surmontées de la même manière.

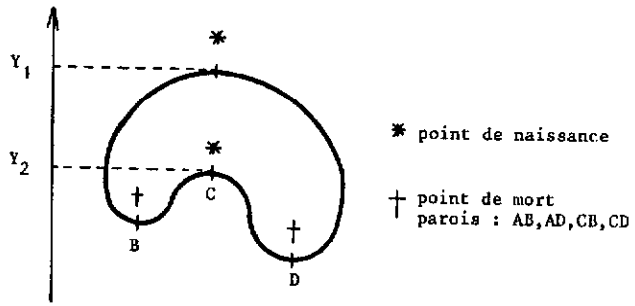
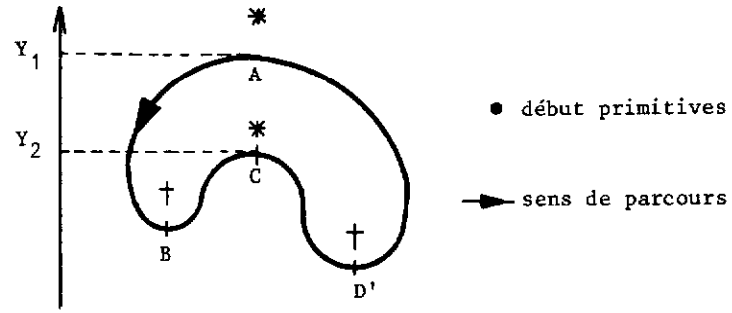


Figure 3.17.: Décomposition du contour en parois.

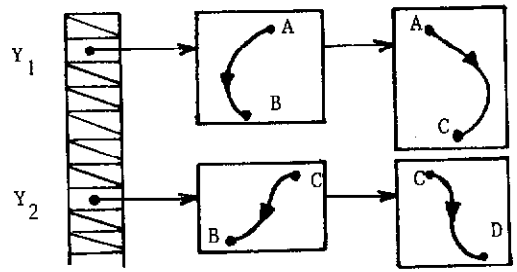
Cependant, nous pouvons envisager deux façons de structurer le contour:

- la première méthode est similaire à celle développée au chapitre 3.3.3.2. On constitue ligne par ligne la liste des primitives orientées vers le bas qui y commencent (voir figure 3.18.a).

- la seconde méthode, préconisée par GOUEIGNOUX et al. décrit les contours à l'aide de structures d'anneaux et d'arbres. Chaque anneau est la description d'une composante connexe découpée en une suite de droites et de cercle. Pour chaque anneau, l'arbre adresse dans l'anneau pour tous points de naissance de même abscisse l'ensemble des parois qui y commencent. (voir figure 3.18.b).

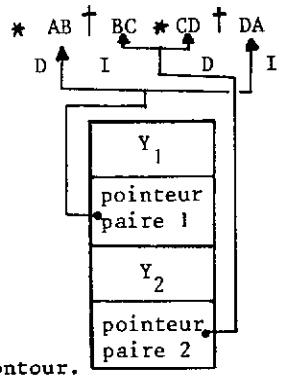


a) Liste des parois orientées vers le bas :



b) Structures d'anneau et d'arbre :

Description de l'anneau



I: paroi inverse
D: paroi directe

Figure 3.18.: Structures du contour.

Pour chacune des deux structures générales, les parois sont décomposées en sous-parois, c'est-à-dire en primitives élémentaires d'affichage. Les segments de droite et les arcs de cercle sont suffisants pour décrire la plupart des contours (voir "méthodologie de décomposition des courbes évoluées" dans <SICO 82>).

Le calcul des points d'intersection des primitives avec chaque ligne de balayage s'effectue de haut en bas. Celui-ci peut se faire à l'aide des algorithmes de génération de courbes de BRESENHAM en ne conservant qu'un seul point significatif par ligne. Le contexte qu'il faut mémoriser à chaque pas est différent selon les primitives (contexte : incrément en X suivant l'octant, expressions récurrentes). Les arcs de cercle seront décomposés en quadrant pour éviter les réinitialisations au cours de la généra-

tion réduisant par là même la taille du contexte. Lors de cette décomposition, les arcs obtenus seront orientés vers le bas, comme les segments de droite (voir figure 3.19.).

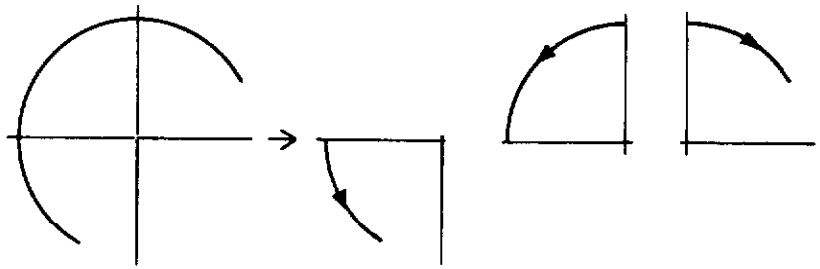
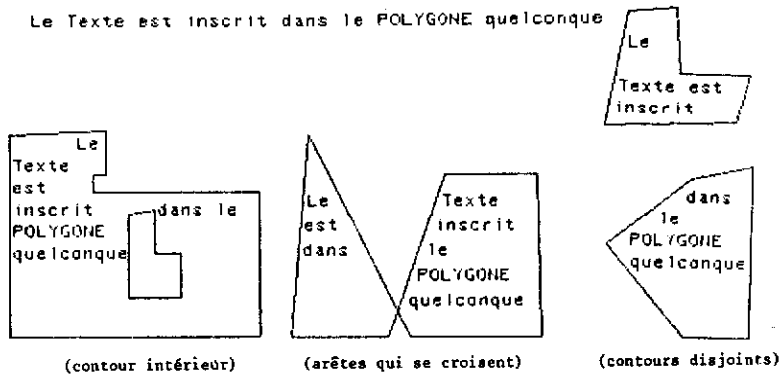


Figure 3.19.: Décomposition d'un arc de cercle par quadrant.

3.3.3.5 - *Une application : écriture d'un texte dans une tache polygonale quelconque* (<HEGRON 83b>).

1. *Le problème*

Nous désirons inscrire un texte dans un polygone quelconque. Nous rencontrons ce problème chaque fois que nous écrivons un texte dans un cartouche en dessin industriel par exemple, ou dans une bulle en bande dessinée, etc... Exemple :



2. *Définitions*

Nous inscrivons le texte sur une ligne d'écriture définie par sa base et son sommet. La hauteur de la ligne d'écriture varie selon la taille des caractères choisie (voir figure 3.20.).

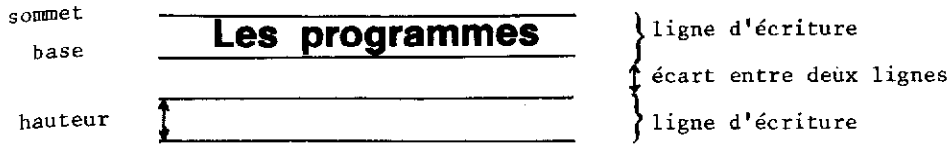


Figure 3.20.: Définition d'une ligne d'écriture.

L'ensemble des bords gauches et l'ensemble des bords droits formés par le polygone engendrent sur les lignes de balayage comprises entre le sommet et la base de chaque ligne d'écriture, un ensemble d'intervalles horizontaux (Voir figure 3.21.).

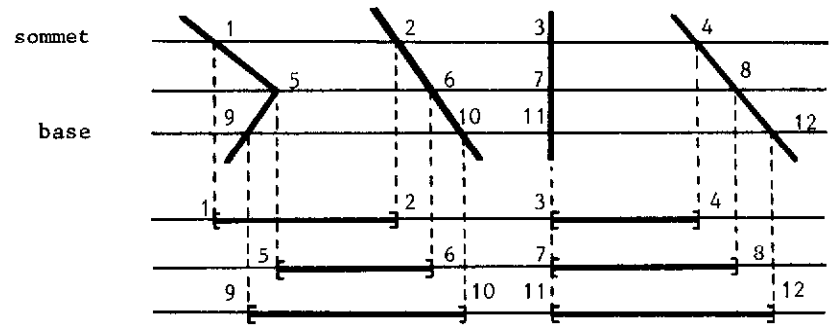
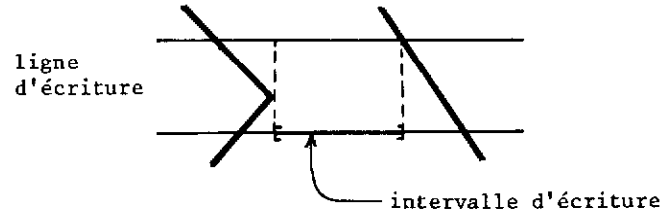
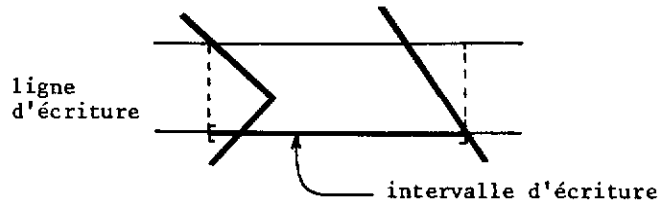


Figure 3.21.: Définition des intervalles horizontaux.

Sur chaque ligne d'écriture, le polygone engendre également des intervalles d'écritures dans lesquels nous pourrons insérer le texte (voir figure 3.22). Nous pouvons faire deux hypothèses : soit les caractères sont strictement contenus par le polygone (Figure 22.a), soit les caractères peuvent couper le contour (Figure 22.b). La seconde hypothèse est intéressante si nous voulons que le texte épouse la forme d'un polygone sans afficher ce dernier.



a) Les caractères sont strictement inclus dans le polygone.



b) Les caractères peuvent couper le contour.

Figure 3.22.: Définition des intervalles d'écriture.

3. *L'idée et l'algorithme*

Pour chaque ligne d'écriture, l'ensemble des intervalles d'écriture est obtenu en réalisant soit l'intersection des listes d'intervalles horizontaux formés par le contour polygonal sur les lignes de balayage comprises entre le sommet et la base de la ligne d'écriture dans le cadre de la première hypothèse, soit l'union de ces mêmes intervalles horizontaux entre chaque ligne de balayage si le texte peut couper le contour.

Pour le calcul des intervalles horizontaux, nous utiliserons l'algorithme de suivi de contour employé pour le remplissage (mêmes structures de données et même logique). Les intervalles d'écriture appartenant à chaque ligne d'écriture seront obtenus en calculant l'intersection ou l'union des intervalles horizontaux des lignes de balayage consécutives prises deux à deux.

Différentes stratégies peuvent être envisagées pour l'écriture du texte dans les intervalles d'écriture :

. Inscrire le texte au fur et à mesure que les lignes d'écriture sont calculées et s'arrêter lorsque tout le texte est affiché. En mode conversationnel, si tout le texte n'est pas écrit, il faut effacer le texte en recommençant le même processus, puis redéfinir une nouvelle taille des caractères et un autre écart entre les lignes d'écriture, et faire une nouvelle tentative d'inscription ;

. Afficher le texte quand tous les intervalles d'écriture sont calculés et si ce dernier peut y être entièrement contenu. Sinon on redéfinit une nouvelle taille des caractères et un nouvel écart, puis on réitère le processus.

La première solution est la plus avantageuse si tout le texte est inscriptible a priori. L'algorithme général suivant réalise cette méthode :


```
Ecriture d'un texte dans un polygone
Début
    obtenir la première ligne d'écriture (sommet,base);
    Tantque (ligne d'écriture non finale et écriture du texte non finie)
    faire
        Traiter la ligne d'écriture;
        Si écriture non finie alors ligne d'écriture suivante fco
    fait
fin

Traiter la ligne d'écriture
Début
    ligne courante := sommet;
    calcul des intersections de la ligne courante avec le contour(liste L1);
    liste des intervalles d'écriture (liste Li):=L1;
    Tantque ligne courante ≠ base faire
    Début
        ligne courante := ligne suivante;
        calcul des intersections (liste L1);
        Li:=Li∩L1 (ou Li∪L1) co intervalles d'écriture fco
    fin
    Si Li≠0 alors écrire le texte dans Li fsi
    Mettre à jour "écriture du texte non finie";
Fin
```

Les algorithmes d'union et d'intersection de deux listes d'intervalles sont développés au chapitre 4.

. Nous donnons une version de cet algorithme programmé dans le BASIC du HP 9345 qui permet d'écrire un texte strictement à l'intérieur d'un contour polygonal quelconque.

Les structures de données utilisées pour décrire le contour polygonal, la liste des arêtes et la liste des points d'intersection sur la ligne de balayage, sont identiques à celles employées par l'algorithme de remplissage par "suivi de contour" (chapitre 3.3.3.2 (2)).

```
10    SUB Texte_polygone(INTEGER N,T(*),Hauteur,Largeur,Ecart,Td,Tf,Texte$)
20    !
30    ! ECRITURE D'UN TEXTE DANS UNE TACHE  POLYGONALE
40    ! PAR LA METHODE DU SUIVI DE CONTOUR
50    !
60    ! Texte : défini par le premier caractère Td et le caractère final Tf
70    !       de la chaine de caractères contenue dans Texte$(*)
80    !                par la Largeur et la Hauteur des caractères
90    !                par l'Ecart entre chaque ligne d'écriture
100   ! Tache polygonale :définie par un ensemble de contours polygonaux
110   !        décrits par la suite ordonnée de leurs sommets terminée par
120   !            NIL dans (N,T(*))
130   !
140   ! DECLARATIONS
150   OPTION BASE 1
160   ! structures de donnees du polygone
170   INTEGER Axd(40),Ayd(40),Ayf(40),Nba,Ymax,Ymin
180   REAL Dxy(40)
190   INTEGER Tliste(454),Liste(40)
200   INTEGER Ei(20),Nbai
210   REAL Inter(20)
220   !
230   ! structures de données de l'écriture
240   INTEGER Y,Yp,Ys,Yb,Md,Mf,Nbcar,Long_mot,Fin_texte,Xpo,Ypo
250   REAL Intervalle(20),X1,X2
260   INTEGER Nbint,Longueur
270   !
280   ! divers
290   INTEGER I,I0,J,K,K1,K2,Ki,Fin
300   INTEGER X,Xd,Yd,Xf,Yf,Nil
310   REAL Dx,Dy
320   !
330   !
340   ! CONTRUCTION DE LA LISTE DES  ARETES ORIENTEES VERS LE BAS
350   !
360   Nil=9999
370   Ymin=Nil
380   Ymax=-Nil
390   Nba=0!nbre d'aretes
400   I=1
410   ! TANT QUE I<N FAIRE
420   IF I>N THEN 1000
430     I0=I
440     Xd=T(I)
450     Yd=T(I+1)
460     I=I+2
470     X=T(I)
480     Fin=0
490     ! REPETER
500       IF X=Nil THEN 550
510         Xf=X
520         Yf=T(I+1)
530         GOTO 590
540         ! dernière arête du contour courant
550         Xf=T(I0)
560         Yf=T(I0+1)
570         Fin=1
580         I=I+1
590       ! FSI
600       ! Traiter l'arête (Xd,Yd),(Xf,Yf)
610       ! afficher l'arête
620       PEN 1
630       MOVE Xd,Yd
640       DRAW Xf,Yf
650       !
```



```
660       IF Yd<=Yf THEN 750
670         Nba=Nba+1
680         Axd(Nba)=Xd
690         Ayd(Nba)=Yd
700         Ayf(Nba)=Yf
710         Dx=Xf-Xd
720         Dy=Yd-Yf
730         Dxy(Nba)=Dx/Dy
740         GOTO 860
750         IF Yd=Yf THEN 840
760           Nba=Nba+1
770           Axd(Nba)=Xf
780           Ayd(Nba)=Yf
790           Ayf(Nba)=Yd
800           Dx=Xd-Xf
810           Dy=Yf-Yd
820           Dxy(Nba)=Dx/Dy
830           GOTO 850
840           ! segment horizontal supprimé
850         ! FSI
860       ! FSI
870       IF Ymax<Yf THEN Ymax=Yf
880       IF Ymin>Yf THEN Ymin=Yf
890       ! fin traiter l'arête
900       !
910       ! arête suivante
920       IF Fin=1 THEN 980
930       Xd=Xf
940       Yd=Yf
950       I=I+2
960       X=T(I)
970       GOTO 490
980     ! Jusqu'a FIN contour
990     GOTO 420
1000  ! FINTQ
1010  !
1020  ! Construction de la liste d'arêtes
1030  !
1040  I=0  !indice de Tliste
1050  J=0  !indice de Liste
1060  K=0  !indice des aretes
1070  MAT Tliste=ZER
1080  MAT Liste=ZER
1090  FOR K=1 TO Nba
1100    I=Ayd(K)
1110    IF Tliste(I)<>0 THEN 1150
1120      ! Tliste(I)=0
1130      Tliste(I)=K
1140      GOTO 1230
1150      ! Tliste(I)#0
1160      J=Tliste(I)
1170      ! Tantque Liste(J)#0 faire
1180      IF Liste(J)=0 THEN 1210
1190        J=Liste(J)
1200        GOTO 1170
1210      ! FTQ
1220      Liste(J)=K
1230    ! FSI
1240  NEXT K
1250  !
1260  !
1270  !
1280  ! ECRITURE DU TEXTE DANS LE POLYGONE
1290  !
1300  ! Obtenir la première ligne d'écriture
1310  Yp=Ymax  !première ligne de balayage
1320  Ys=Yp-Ecart  !sommet de la ligne d'écriture
1330  Yb=Ys-Hauteur  !base de la ligne d'écriture
1340  Fin_texte=0
```

```
1360  ! Obtenir le premier mot
1370  Md=Td  !1er caractere du mot
1380  I=Md
1390  IF Texte$[I,I]<>" " THEN 1420
1400    I=I+1 !sauter les blancs
1410    GOTO 1390
1420  ! FTQ
1430  IF (I>Tf) OR (Texte$[I,I]=" ") THEN 1460
1440    I=I+1 !sauter les car. # du blanc
1450    GOTO 1430
1460  ! FTQ
1470  Mf=I-1 !dernier cart. du mot
1480  Nbcar=I-Md !nbre de caractères du mot
1490  Long_mot=Nbcar*Largeur
1500  !
1510  !
1520  Nbai=0
1530  ! Créer listes Ei,Inter
1540  J=Tliste(Yp)
1550  IF J=0 THEN 1610
1560    Nbai=Nbai+1
1570    Ei(Nbai)=J
1580    Inter(Nbai)=Axd(J)
1590    J=Liste(J)
1600    GOTO 1550
1610  ! FTQ
1620  !
1630  ! TANT QUE (ligne d'écriture non finale)et(non Fin_texte) FAIRE
1640  IF (Yb<=Ymin) OR (Fin_texte=1) THEN 3470
1650    !
1660    ! TRAITER LA LIGNE D'ECRITURE COURANTE
1670    !
1680    ! Constitution de la liste courante des intervalles:SOMMET
1690    ! enlever les arêtes qui finissent dans ]Yp,Ys]
1700    Ki=1
1710    IF Ki>Nbai THEN 1860
1720      K=Ei(Ki)
1730      IF Ayf(K)<Ys THEN 1830
1740        J=Ki+1
1750        IF J>Nbai THEN 1800
1760          Ei(J-1)=Ei(J)
1770          Inter(J-1)=Inter(J)
1780          J=J+1
1790          GOTO 1750
1800        ! FTQ
1810        Nbai=Nbai-1
1820        Ki=Ki-1
1830      ! FSI
1840      Ki=Ki+1
1850      GOTO 1710
1860    ! FTQ
1870    !
1880    ! calcul des intersections avec le sommet Ys
1890    FOR K=1 TO Nbai
1900      Inter(K)=Inter(K)+Ecart*Dxy(Ei(K))
1910    NEXT K
1920    !
1930    ! ajouter les arêtes qui commencent sans finir dans ]Yp,Ys]
1940    FOR Y=Yp-1 TO Ys STEP -1
1950      J=Tliste(Y)
1960      IF J=0 THEN 2060
1970        IF Ayf(J)>=Ys THEN 2030
1980          Nbai=Nbai+1
1990          Ei(Nbai)=J
2000          Dx=Axd(J)
2010          Dy=Y-Ys
2020          Inter(Nbai)=Dx+Dy*Dxy(J)
2030        ! FSI
```



```
2040        J=Liste(J)
2050        GOTO 1960
2060      ! FTQ
2070    NEXT Y
2080    !
2090    ! trier la liste des points d'intersection
2100    CALL Tri_bulle(Inter(*),Ei(*),Nbai)
2110    !
2120    !
2130    ! Création de la suite d'intervalles d'écriture
2140    FOR I=1 TO Nbai
2150      Intervalle(I)=Inter(I)
2160    NEXT I
2170    Nbint=Nbai
2180    !
2190    ! Calcul de la suite d'intervalles d'écriture
2200    !
2210    ! POUR chaque ligne de balayage de ]Ys,Yb] FAIRE
2220    FOR Y=Ys-1 TO Yb STEP -1
2230      ! enlever les arêtes qui finissent sur la ligne Y
2240      Ki=1
2250      IF Ki>Nbai THEN 2400
2260        K=Ei(Ki)
2270        IF Ayf(K)<>Y THEN 2370
2280          J=Ki+1
2290          IF J>Nbai THEN 2340
2300            Ei(J-1)=Ei(J)
2310            Inter(J-1)=Inter(J)
2320            J=J+1
2330            GOTO 2290
2340          ! FTQ
2350          Nbai=Nbai-1
2360          Ki=Ki-1
2370        ! FSI
2380        Ki=Ki+1
2390        GOTO 2250
2400      ! FTQ
2410      !
2420      ! calcul des intersections
2430      FOR K=1 TO Nbai
2440        Inter(K)=Inter(K)+Dxy(Ei(K))
2450      NEXT K
2460      !
2470      ! ajouter les arêtes qui commencent sur la ligne Y
2480      J=Tliste(Y)
2490      IF J=0 THEN 2550
2500        Nbai=Nbai+1
2510        Ei(Nbai)=J
2520        Inter(Nbai)=Axd(J)
2530        J=Liste(J)
2540        GOTO 2490
2550      ! FTQ
2560      !
2570      ! tri des intersections
2580      CALL Tri_bulle(Inter(*),Ei(*),Nbai)
2590      !
2600      ! intersection de Intervalle(*) ET Inter(*)
2610      CALL Inter_listes(Intervalle(*),Nbint,Inter(*),Nbai)
2620      !
2630    NEXT Y
2640    !
2650    !
2660    !
```

```
2670    ! Ecriture du texte dans la suite definie par Intervalle(*)
2680    IF Nbint=0 THEN 3370
2690      ! suite non vide
2700      ! premier intervalle d'ecriture
2710      K1=1
2720      K2=2
2730      X1=Intervalle(K1)
2740      X2=Intervalle(K2)
2750      Longueur=INT(X2-X1+.5)
2760      ! point origine du premier mot
2770      Xpo=INT(X1+.5)+3
2780      IF Md=Td THEN 2810
2790        Xpo=Xpo-Largeur
2800        Longueur=Longueur+Largeur
2810      ! FSI
2820      Ypo=Yb+INT(Hauteur/4)!base d'écriture
2840      ! REPETER
2850        ! Traiter l'Intervalle
2860        !
2870        ! Tant que(non fin_texte)ET(Longueur)=Long_mot)Faire
2880        IF (Fin_texte=1) OR (Longueur<=Long_mot) THEN 3180
2890          ! Ecrire le mot
2900          PEN 1
2910          CSIZE Hauteur/4.54,Largeur/Hauteur !hauteur donnée en unité GDU
2920          MOVE Xpo,Ypo !point origine du mot
2930          LABEL Texte$[Md,Mf]
2940          ! Nouveau point origine
2950          Xpo=Xpo+Long_mot
2960          ! Nouvelle longueur de l'intervalle
2970          Longueur=Longueur-Long_mot
2980          !
2990          ! Obtenir le mot suivant
3000          IF Mf<>Tf THEN 3030
3010            Fin_texte=1
3020            GOTO 3160
3030            Md=Mf+1
3040            I=Md
3050            IF (I>Tf) OR (Texte$[I,I]<>" ") THEN 3090
3060              I=I+1 !sauter les blancs
3070              GOTO 3050
3080            ! FTQ
3090            IF (I>Tf) OR (Texte$[I,I]=" ") THEN 3120
3100              I=I+1 !sauter les car. # du blanc
3110              GOTO 3090
3120            ! FTQ
3130            Mf=I-1
3140            Nbcar=I-Md
3150            Long_mot=Nbcar*Largeur
3160          ! FSI
3170          GOTO 2870
3180        ! FINTQ
3190        !
3200        IF Fin_texte=1 THEN 3330
3210          ! Intervalle suivant
3220          K1=K1+2
3230          IF K1>Nbint THEN 3310
3240            K2=K2+2
3250            X1=Intervalle(K1)
3260            X2=Intervalle(K2)
3270            Longueur=INT(X2-X1+.5)+Largeur-3
3280            ! point origine
3290            Xpo=INT(X1+.5)-Largeur+3
3300            IF Md<>Td THEN 3330
3310              Longueur=Longueur-Largeur
3320              Xpo=Xpo+Largeur
3330          ! FSI
3340        ! FSI
3350      IF (Fin_texte<>1) AND (K1<Nbint) THEN 2840
3360      ! JUSQU'A (Fin_texte) OU (Intervalle final)
3370    ! FSI
```



```
3380      !
3390      ! Ligne d'ecriture suivante
3400      Yp=Yb
3410      Ys=Yp-Ecart
3420      Yb=Ys-Hauteur
3430      !
3440      !
3450      GOTO 1640
3460      !
3470    ! FIN TANTQUE
3480    !
3490    SUBEND
3500    !
3540    SUB Inter_listes(REAL Intervalle(*),INTEGER Nbint,REAL Inter(*),
INTEGER Nbai)
3550    !
3560    ! INTERSECTION DES INTERVALLES DEFINIS DANS CHACUNE DES DEUX LISTES EN
3570    ! PRENANT LES ELEMENTS 2 à 2 ORDONNES DANS L'ORDRE CROISSANT.
3580    ! LE RESULTAT EST CONTENU DANS Intervalle(*)
3590    !
3600    OPTION BASE 1
3610    INTEGER Int1,Int2,I,J,L(20),N
3620    REAL Liste(20)
3630    !
3640    ! Interclassement de Intervalle[*] et de Inter[*]:resultat dans
3650    ! Liste[*]  et dans L[*] qui contiennent respt. les valeurs et
3660    ! le numéro de la liste correspondante
3670    CALL Fusion(Intervalle(*),Nbint,Inter(*),Nbai,Liste(*),L(*),N)
3680    !
3690    ! Extraire l'intersection de (Liste(*),L(*))
3700    !
3710    Int1=-1!faux
3720    Int2=-1!faux
3730    I=0
3740    J=0
3750    ! TANT QUE  I<N  FAIRE
3760    IF I>=N THEN 4030
3770      !
3780      ! Tantque (NON(Int1) OU NON(Int2)) ET I<N Faire
3790      IF (I>=N) OR (Int1=1) AND (Int2=1) THEN 3870
3800        I=I+1
3810        IF L(I)=2 THEN 3840
3820          Int1=-Int1
3830          GOTO 3850
3840          Int2=-Int2
3850        ! FSI
3860        GOTO 3780
3870      ! FTQ
3880      !
3890      IF (Int1=-1) OR (Int2=-1) THEN 4000
3900        J=J+1
3910        Intervalle(J)=Liste(I)
3920        I=I+1
3930        IF L(I)=2 THEN 3960
3940          Int1=-Int1
3950          GOTO 3970
3960          Int2=-Int2
3970        ! FSI
3980        J=J+1
3990        Intervalle(J)=Liste(I)
4000      ! FSI
4010      GOTO 3750
4020      !
4030    ! FINTQ
4040    Nbint=J
4050    !
4060    SUBEND !Inter_listes
```

```
4110  SUB Fusion(REAL L1(*),INTEGER N1,REAL L2(*),INTEGER N2,REAL L3(*),INTEG
L(*),N)
4130  ! INTERCLASSEMENT DES LISTES L1 ET L2 :RESULTAT DANS L3 ET L
4150  N=0
4160  I=1
4170  J=1
4180  ! TANTQUE I<=N1 ET J<=N2 FAIRE
4190  IF (I>N1) OR (J>N2) THEN 4310
4200    N=N+1
4210    IF L1(I)>L2(J) THEN 4260
4220      L3(N)=L1(I)
4230      L(N)=1
4240      I=I+1
4250      GOTO 4290
4260      L3(N)=L2(J)
4270      L(N)=2
4280      J=J+1
4290    ! FSI
4300    GOTO 4180
4310  ! FTQ
4330  IF I>N1 THEN 4390
4340    N=N+1
4350    L3(N)=L1(I)
4360    L(N)=1
4370    I=I+1
4380    GOTO 4330
4390  ! FTQ
4410  IF J>N2 THEN 4470
4420    N=N+1
4430    L3(N)=L2(J)
4440    L(N)=2
4450    J=J+1
4460    GOTO 4410
4470  ! FTQ
4480  !
4490  SUBEND !Fusion

4500  SUB Tri_bulle(REAL Ta(*),INTEGER Tb(*),N)
4510  ! TRI DANS L'ORDRE CROISSANT DES N VALEURS DES TABLEAUX Ta(*) ET Tb(*)
4520  ! Ta(*) étant le maitre
4540  OPTION BASE 1
4550  REAL Tai,Taj
4560  INTEGER B,Inv,I,J,N1,N2
4580  N1=N
4590  ! Repeter
4600  N2=N1-1
4610  Inv=0
4620  I=1
4630  IF I>N2 THEN 4780
4640    J=I+1
4650    Tai=Ta(I)
4660    Taj=Ta(J)
4670    IF Taj>=Tai THEN 4750
4680      Inv=1
4690      Ta(I)=Taj
4700      Ta(J)=Tai
4710      B=Tb(I)
4720      Tb(I)=Tb(J)
4730      Tb(J)=B
4740      N1=I
4750    ! FSI
4760    I=I+1
4770    GOTO 4630
4780  ! FTQ
4790  IF Inv=1 THEN 4590
4800  ! Jusqu'a Inv=0
4820  SUBEND !Tri_bulle
```


3.3.4 - CONCLUSION

Nous avons exposé deux approches générales utilisées pour le remplissage des taches polygonales. La première raisonne à partir du codage du contour dans une matrice de points. La seconde utilise une représentation structurée du contour pour effectuer le remplissage par suivi du contour.

Les algorithmes de remplissage par codage du contour présentent les inconvénients suivants :

- mémoire d'image supplémentaire,
- raisonnement effectué sur la surface du rectangle circonscrit à la tache (calculs supplémentaires),
- génération simultanée d'un ensemble de taches qui se chevauchent impossible.

Par contre, les algorithmes de suivi de contour possèdent les avantages suivants :

- peu de mémoire supplémentaire (proportionnelle au nombre d'arêtes),
- on ne travaille que sur l'intérieur de la tache,
- ils sont généralisables au remplissage simultané d'un ensemble de taches qui se chevauchent ou non,
- nous verrons également dans le chapitre 3.5. qu'ils sont facilement adaptables au hachurage.

3.4. Décomposition des taches polygonales en éléments simples

3.4.1 - INTRODUCTION

Pour le remplissage des taches polygonales, une autre approche consiste à décomposer le contour polygonal en un ensemble de régions cohérentes formées de trapèzes ou de triangles (voir figure 3.23.) puis à appliquer sur chaque région élémentaire un algorithme de remplissage très simple.

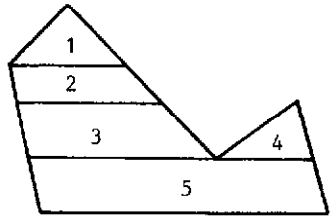


Figure 3.23.: Régions cohérentes d'un contour polygonal.

Les différences qui existent entre les méthodes proposées, résident essentiellement dans l'ordre par lequel les régions élémentaires sont remplies (glissement sur les contours, tri des arêtes suivant les ordonnées) (voir <NEWMAN-SPROULL 79>, <BRASSEL-FEGEAS 79>) et par la réduction éventuelle du nombre de régions.

Nous avons montré que, après la décomposition en triangles d'un polygone quelconque, la complexité des algorithmes de remplissage de taches appliqués sur cet ensemble, demeure dans la plupart des cas (convexité, nbam=2,...) équivalente ou supérieure à la complexité de ces mêmes algorithmes employés sur la tache polygonale initiale (voir <HEGRON 82b>). Pour les cas particuliers où ceci n'est plus vérifié, la complexité du découpage du polygone étant similaire à celle des techniques de remplissage la décomposition du polygone en figures élémentaires ne réduit pas la complexité initiale du remplissage.

Un autre inconvénient de la décomposition, est la perte du codage ligne par ligne de la tache initiale qui demeure intéressant pour la transmission de l'image en télévision.

La décomposition ne peut devenir avantageuse que dans la mesure où les algorithmes de remplissage des trapèzes ou des triangles sont réalisés de façon cablée ou micro-programmée (voir <MARTINEZ 82> et <LITTLE-HEUFT 79>). Nous pouvons envisager également le remplissage des régions élémentaires en parallèle, au fur et à mesure de leur création (parallélisme de type pipe-line).

Comme aucun des algorithmes de décomposition rencontrés ne traitent des contours polygonaux possèdant des arêtes qui se coupent, nous donnons ci-dessous une idée d'algorithme fondée sur la notion de suivi de contour.

3.4.2 - NOUVEL ALGORITHME DE DECOMPOSITION

La tache est formée d'un ensemble de contours polygonaux quelconques où les arêtes peuvent se croiser.

Notre idée d'algorithme repose sur celui de BENTLEY et OTTMAN qui recherche toutes les intersections dans un ensemble de segments. Les segments seront pour nous les arêtes des contours.

3.4.2.1 - La méthode de BENTLEY et OTTMAN

La technique de BENTLEY et OTTMAN <BENTLEY-OTTMAN 79> qui reprend les idées de SHAMOS et HOEY, <SHAMOS-HOEY 76>, recherche toutes les intersections dans un ensemble de N segments.

L'algorithme se décompose en deux parties :

1ère partie : tri des 2 N abscisses en ordre croissant. Une structure de données Q, contiendra cette séquence.

2ème partie : exploration de cette séquence Q obtenue en commençant par la plus petite abscisse. Celle-ci est stockée dans une structure intermédiaire R, réduite ici à un élément (c'est bien sûr l'abscisse début d'un segment S). Puis on passe à l'abscisse suivante dans la séquence Q :

. Si c'est une abscisse début (soit x_o): on calcule les ordonnées des points d'intersection de la droite $x=x_o$ avec les segments qu'elle rencontre, ce qui induit un ordre vertical. On insère le segment concerné dans R suivant l'ordre vertical. Si ce segment coupe celui au-dessus et au-dessous dans R, alors insérer l(es) abscisse(s) d(es) point(s) d'intersection dans Q.

. Si c'est une abscisse fin : si les segments au-dessus et au-dessous de ce segment dans R se coupent alors insérer l'abscisse de l'intersection dans Q. Enlever le segment de R.

. Si c'est l'abscisse d'une intersection, il faut échanger les positions des segments concernés et vérifier si après cet échange les segments nouvellement adjacents dans R se coupent. On insère les abscisses des points d'intersection dans Q.

. Si Q est vide alors fin ;

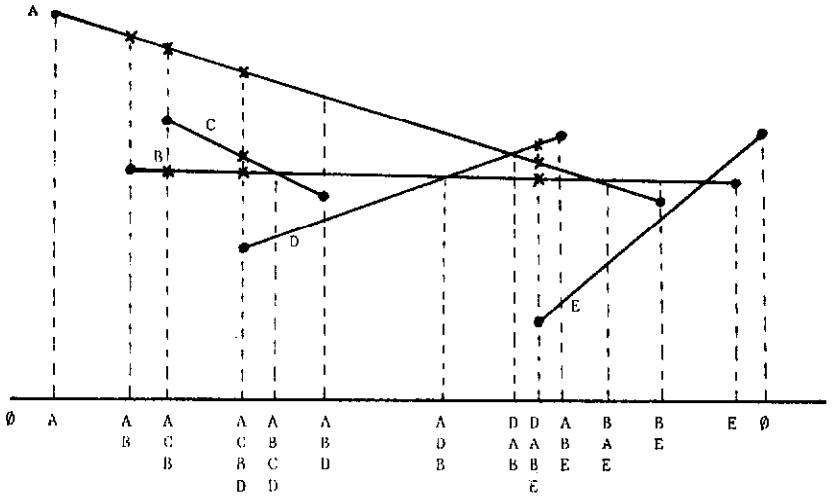


Figure 3.24.: Exemple de traitement par BENTLEY et OTTMAN (tiré de <LUCAS et al.79>).

En ce qui nous concerne, nous raisonnerons suivant les ordonnées décroissantes. Mais il faut noter que dans l'algorithme de BENTLEY et OTTMAN des difficultés apparaissent dans deux cas :

- segments horizontaux : dans notre application, cette difficulté disparaît car nous les supprimons ;

- intersections communes à plus de deux segments.

3.4.2.2 - Utilisation de l'algorithme de BENTLEY et OTTMAN

La détermination des régions élémentaires débute par le tri des extrémités des arêtes non horizontales suivant les Y décroissants.

La structure R induit sur les arêtes un ordre horizontal qui permettra de distinguer les bords gauches des bords droits.

Dans la structure R, trois types de points se présentent :

1. début d'arête,
2. fin d'arête,
3. point d'intersection.

Les modifications à apporter à l'algorithme de BENTLEY et OTTMAN sont les suivantes :

- dans le parcours de la structure Q, nous regroupons l'étude des points appartenant à la même ligne horizontale (c'est-à-dire de mêmes ordonnées). Pour chacune des lignes horizontales où se produit un ou plusieurs changements, nous obtenons la liste ordonnée des arêtes dans l'ordre croissant des abscisses des points d'intersection (structure R) ;

- la décomposition en trapèzes ou triangles se fait entre deux lignes horizontales à partir de la structure R de la ligne courante et l'ordonnée de la ligne suivante.

L'algorithme aura la forme générale suivante :

```
DECOMPOSITION
Début
    1. Tri des 2.N ordonnées des extrémités des arêtes non horizontales
       en ordre décroissant (structure Q) ;
    2. Répéter
           obtenir la première ligne horizontale (structure R);
           Tant que R non vide faire
           Début
               obtenir l'ordonnée de la ligne suivante;
               décomposition en éléments simples;
               obtenir la ligne horizontale suivante(structure R);
           fin
       jusqu'à Q vide
fin
```

Etudions chaque module :

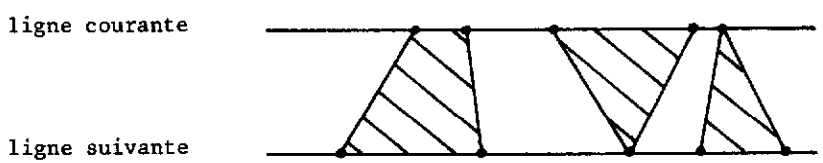
. Obtenir la ligne horizontale :

```
Pour tous points de Q de mêmes ordonnées faire
Début
    traiter le point comme dans la deuxième partie
    de l'algorithme de BENTLEY et OTTMAN
Fin
```

(en sortie, on obtient la structure R).

. Décomposition en éléments simples :

On calcule les abscisses des intersections des arêtes appartenant à R avec la ligne horizontale suivante. Nous obtenons alors un ensemble de bords gauches et droits suivant l'ordre horizontal :



- Décomposition en trapèzes :

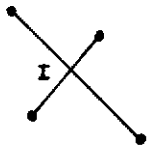
On extrait de R les arêtes deux par deux dont les extrémités correspondent aux points d'intersection sur la ligne courante et sur la ligne suivante.

- Décomposition en triangles :

On fait la même chose que précédemment, puis pour chaque couple, si aucune des extrémités n'est commune, on décompose le trapèze en deux triangles :

Mais le bon fonctionnement de l'algorithme nécessite une définition précise de l'intersection entre deux arêtes. Deux arêtes se coupent au point I si elles ne sont pas colinéaires et si aucune des extrémités n'est commune :

intersection au point I :

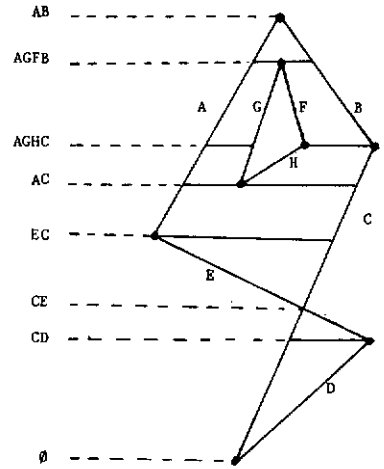


pas d'intersection :

deux extrémités communes — une extrémité commune — colinéarité

D'autre part, dans la structure R, les arêtes possédant le même point d'intersection avec la ligne horizontale courante seront ordonnées suivant l'ordre croissant de leurs pentes (les arêtes étant orientées vers le bas).

Exemple : Décomposition d'une tache polygonale quelconque :



Complexité théorique :

La méthode possède une complexité en O(NLogN + k LogN), tri sur les 2.N ordonnées inclus, où k est le nombre total d'intersections.

En terme d'appels à la fonction "intersection de deux segments de droite" le coût est en O(N+k).

3.4.3 - CONCLUSION

Pour conclure, nous pouvons faire deux remarques :

. Par rapport aux algorithmes existants, cette méthode permet de décomposer en éléments simples, des polygones quelconques admettant des arêtes qui se coupent et également des trous et des contours disjoints. Si aucune des arêtes ne se croisent, nous pouvons simplifier les calculs (pas de calcul d'intersection de segments de droite).

. L'efficacité de l'approche du remplissage par décomposition n'apparaît pas a priori par rapport aux approches directes.

3.5. Le hachurage de taches polygonales

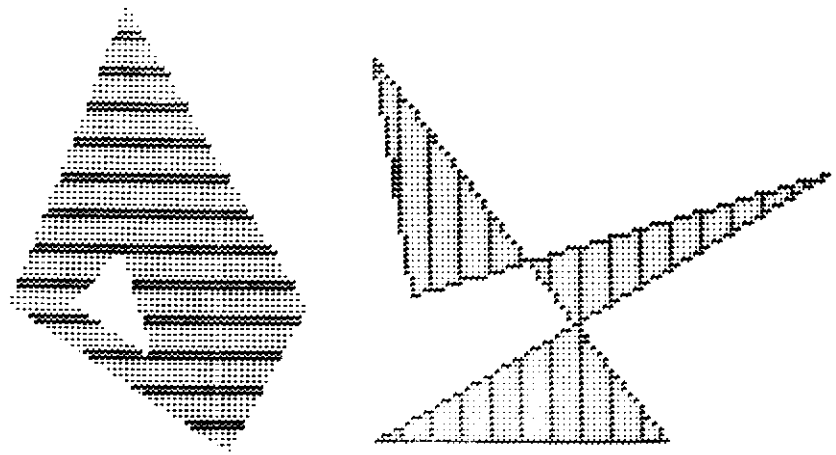
3.5.1 - LE HACHURAGE VERTICAL OU HORIZONTAL

Le problème du hachurage horizontal est équivalent à celui du remplissage par la méthode du suivi de contour.

La cohérence qui existe entre deux lignes de hachurages consécutives est la suivante : soit E_i l'ensemble des arêtes qui coupent la ligne i, alors $E_{i+1}=E_i+\{$arêtes qui commencent dans la bande $]$ligne i, ligne i+1$[\}-\{$arêtes qui finissent dans $]$ligne i, ligne i+1$]\}$.

D'autre part, le hachurage vertical est identique au hachurage horizontal en inversant le rôle des abscisses et des ordonnées.

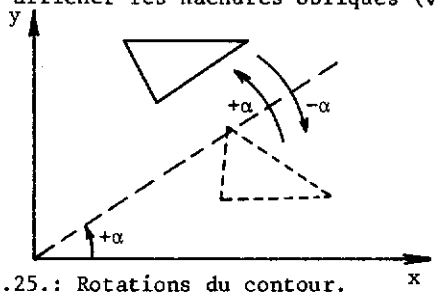
Exemples de hachurage : (grisés obtenus avec des cellules 3x3)



3.5.2 - HACHURAGE OBLIQUE DE CONTOURS POLYGONAUX

3.5.2.1 - *Introduction*

Puisque nous savons hachurer horizontalement, nous pouvons envisager de nous ramener à ce problème en opérant une rotation d'un angle $-\alpha$ du contour, α étant l'angle formé par les lignes de hachurage. Puis pour chaque segment horizontal calculé, nous effectuons une seconde rotation d'un angle α pour afficher les hachures obliques (voir figure 3.25.).



α = angle des lignes de hachurage

Figure 3.25.: Rotations du contour.

Cette méthode préconisée par BRASEL et FEGEAS (<BRASEL-FEGEAS 79>) demeure simple dans sa logique mais n'apparaît pas optimale car nous opérons deux rotations successives. A l'aide de la technique du suivi de contour, notre idée est de conserver le schéma général utilisé pour le hachurage horizontal sans avoir recours aux rotations successives.

3.5.2.2 - *Le hachurage oblique par "suivi de contour"*

1. *L'idée* :

Le problème à résoudre est de connaître l'ensemble des arêtes qui coupent chaque ligne de hachurage.

Soit i la ligne de hachurage courante et (i-1) la ligne précédente. Soit E_{i-1} l'ensemble des arêtes qui coupent la ligne (i-1). On a : $E_i = E_{i-1}$ + {arêtes qui commencent dans la bande]ligne i-1,ligne i]} - {arêtes qui finissent dans la bande]ligne i-1,ligne i]}.

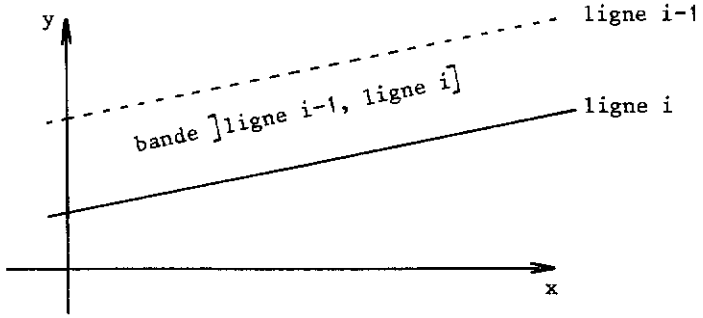


Figure 3.26.: Définition de la bande]ligne i-1,ligne i]

Une arête commence (ou finit) dans la bande]ligne i-1,ligne i] (voir figure 26) si son extrémité début (ou fin) appartient à ce plan : c'est le problème général de l'appartenance d'un point à un plan.

Soit a la pente des lignes de hachurage. Chaque ligne i est caractérisée par l'équation $y=ax+y_i$ où y_i=constante. Si (x_1,y_1) sont les coordonnées d'un point P, la droite de pente a passant par ce point est définie par l'équation $y=ax+y_p$ ou $y_p=y_1-ax_1$ (voir figure 27). Donc le point P appartient à la bande]ligne i-1,ligne i] si $y_{i-1} > y_p \geq y_i$.

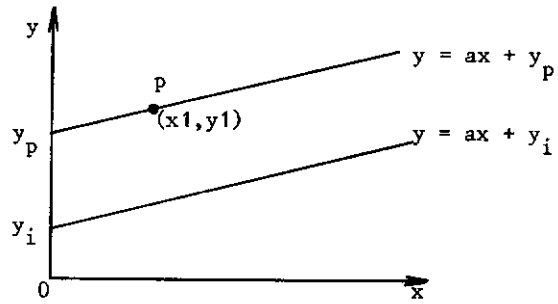


Figure 3.27.: Définition d'une ligne de hachurage et position relative d'un point.

2. *L'algorithme*

Soit un contour polygonal défini par la liste ordonnée S de ses n sommets et a la pente des lignes de hachurage. Tout point $s \in S$ est caractérisé par la valeur ys donnée par l'équation : $ys=y'-ax'$ où (x',y') sont les coordonnées de s.

Nous construisons une nouvelle liste de sommets ordonnée suivant l'ordre décroissant des ys correspondant. On obtient la suite : $(s_j,ys_j)_{j=1...n}$ où $ys_{j-1} \geq ys_j$ (voir figure 3.28.).

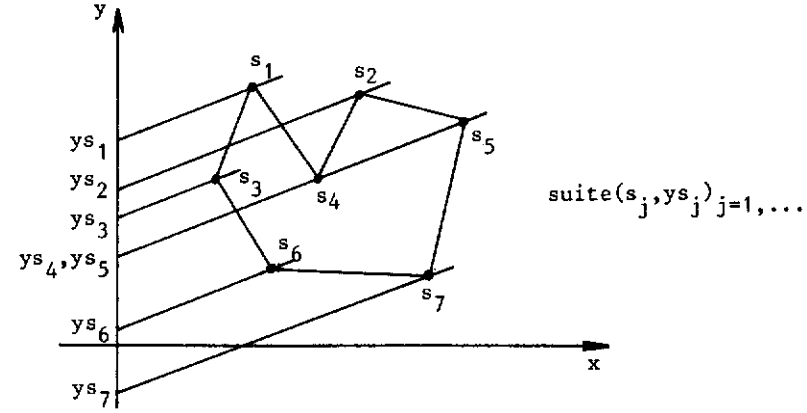


Figure 3.28.: Constitution de la liste ordonnée des sommets.

Chaque arête ayant pour extrémités les sommets sd(début) et sf(fin) est orientée de telle façon que ysd>ysf (voir figure 3.29.).

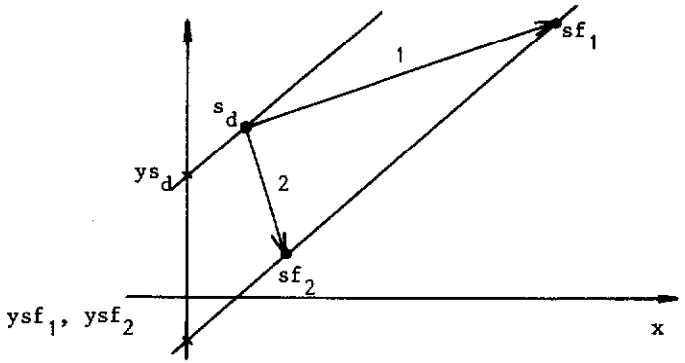


Figure 3.29.: Orientation des arêtes.

Dans l'algorithme de remplissage par suivi de contour, nous nous déplaçons sur le contour suivant la direction verticale. Ici, nous suivons le contour dans une direction perpendiculaire aux lignes de hachurage.

Le calcul des intersections tient compte de la cohérence qui existe entre deux lignes de hachurage consécutives. Pour ce faire, nous calculons pour chaque arête l'accroissement en X et en Y existant entre les lignes. Le premier point d'intersection est obtenu en réalisant le calcul d'intersection entre deux droites, la ligne de hachurage et la droite support de l'arête.

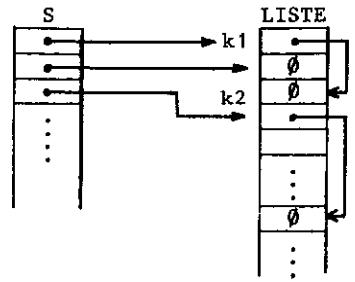
3. *Mise-en-oeuvre*

On se servira des structures de données suivantes :

- La liste des sommets définie par :
  - s(*) : tableau servant pour la numérotation des sommets
  - x(*),y(*) : coordonnées des sommets du contour
  - ys(*) : intersections des droites passant par les sommets avec l'axe oy (droites parallèles aux lignes de hachurage).

- La liste des arêtes orientées :
  - sd(*) : sommet début (donné par le numéro du sommet)
  - sf(*) : sommet fin
  - a(*) : accroissement en x
  - b(*) : accroissement en y
  - dax(*) : accroissement en x entre les lignes de hachurage
  - day(*) : accroissement en y entre les lignes de hachurage

- Liste des arêtes qui commencent sur chaque ligne de hachurage passant par les sommets (ordre décroissant des ys)



- S(*) : S(i) pointe sur la liste des arêtes $k_1,k_2,\ldots,k_p$ qui commencent au sommet i (les sommets i=1,2,...nbsom respectent l'ordre décroissant des ys_i correspondant).

- La liste des arêtes coupant la ligne de hachurage i :
  - Ei(*) : liste des arêtes
  - interx(*), intery(*) : coordonnées des intersections de chaque arête avec la ligne i.

Comme pour l'algorithme de remplissage, le contour de la tache est constitué d'un ensemble de polygones. Nous disposons également de l'Ecart qui existe entre deux lignes de hachurage consécutives et de leur Pente.

L'algorithme a la forme générale suivante :

```
HACHURAGE OBLIQUE (N,T[*])
Début
    Prétraitement;
    hachurage;
Fin;

Prétraitement;
Début
    liste des arêtes orientées;
    ordonner (s,ys);
    renumérotation des sommets constituants les arêtes;
    création de la liste des arêtes (S,LISTE)
Fin prétraitement;

Hachurage
Début
    Créer liste Ei, Inter ;
    obtenir la première ligne de hachurage ;
    co hachurage proprement dit fco
    Tant que ligne de hachurage non finale faire
    Début
        calcul des intersections des arêtes existantes dans Ei;
        ajouter les arêtes qui commencent dans la bande ]ligne i-1,ligne i]
        enlever les arêtes qui finissent dans la bande ]ligne i-1,ligne i]
        trier (Ei,Inter);
        tracer les hachurages;
        obtenir la ligne de hachurage suivante;
    fin;
Fin Hachurage
```

Développons les modules du prétraitement

```
Liste des arêtes orientées
Début
    co écart vertical entre les lignes de hachurage fco
    Dy := Ecart * SQRT(1+Pente²);
    co initialisations fco
    Nbs:=1 ; co nombre de sommets fco
    Nba:=0 ; co nombre d'arêtes fco
    i:=1 ; co indice parcourant T(*) fco
    Tant que i<N faire
    Début
      Nbso:=Nbs ; co premier sommet du polygone fco
      io:=i
      Xd:=T(i) ; Yd:=T(i+1);
      i:=i+2 ; Xo:=T(i);
      x(Nbs):=Xd ; y(Nbs):=Yd;
      Ysd:=Yd-Pente*Xd ; Nsd:=Nbs;
      s(Nbs):=Nbs ; Ys(Nbs):=Ysd;
      Nbs:=Nbs+1;
      Fin:=faux;
      Répéter
        si Xo≠NIL alors Xf:=Xo, Yf:=T(i+1);
                        x(Nbs):=Xf ; y(Nbs):=Yf;
                        Ysf:=Yf-Pente*Xf ; Nsf:=Nbs;
                        s(Nbs):=Nbs ; Ys(Nbs):=Ysf;
                        Nbs:=Nbs+1;
                   sinon Xf:=T(io) ; Yf:=T(io+1) ; i:=i+1;
                        Ysf:=Ys(Nbso), Nsf:=Nbso
                        Fin:=vrai co dernière arête du polygone fco
        co traiter l'arête ((Xd,Yd),(Xf,Yf)) fco
        Si Ysd>Ysf
          alors Nba:=Nba+1;
                sd(Nba):=Nsd, sf(Nba):=Nsf;
                Ax:=Xf-Xd ; Ay:=Yf-Yd;
                a(Nba):=Ax ; b(Nba):=Ay;
                dax(Nba):=(Dy*Ax)/(Ay+Pente*Ax);
                day(Nba):=Pente*dax(Nba)-Dy;
          sinon si Ysd<Ysf
                  alors Nba:=Nba+1;
                        sd(Nba):=Nsf ; sf(Nba):=Nsd;
                        Ax:=Xd-Xf ; Ay:=Yd-Yf;
                        a(Nba):=Ax ; b(Nba):=Ay;
                        dax(Nba):=(Dy*Ax)/(Ay+Pente*Ax);
                        day(Nba):=Pente*dax(Nba)-Dy;
                fsi
        fsi
        co arête suivante fco
        si non(Fin) alors Xd:=Xf ; Yd:=Yf ; Ysd:=Ysf ; Nsd:=Nsf
                          I:=I+2 ; Xo:=T(I);
      Jusqu'à Fin
    Fintq
    Nbsom:=Nbs-1;
Fin
```



```
ordonner(s,ys);
Début
    trier(ys(*),s(*),x(*),y(*),nbsom) dans l'ordre décroissant des ys;
Fin;

Rénumérotation des sommets constituants les arêtes;
Début

    Pour i:=1 jusqu'à nbsom faire
    début
      s1(s(i)):=i;
    fin
    Pour k:=1 jusqu'à nba faire
    Début
      sd(k):=s1(sd(k));
      sf(k):=s1(sf(k));
    fin
Fin renumérotation;

Créer liste des arêtes S,LISTE
Début
    i:=0 co indice de s(*) fco ; j:=0 co indice de liste(*) fco
    k:=0 co indice des arêtes fco
    s(*):=0 ; liste(*):=0 ; co initialisation des tableaux à zéro fco
    Pour k:=1 à nba faire
      i:=sd(k);
      si s(i)=0
        alors s(i):=k;
        sinon j:=s(i);
              Tantque liste(j)≠0 faire
              Début
                 j:=liste(j);
              fin
              liste(j):=k;
      fsi
    finpour
Fin

Développons les modules du Hachurage :
Créer liste Ei, Inter;
Début
    nbai:=0 co nbre arêtes sur ligne i fco
    ns:=1 co n° sommet fco
    ypred:=ys(ns)
    Tantque ypred=ys(ns) faire
    Début
      j:=s(ns);
      Tantque j≠0 faire
        nbai:=nbai+1;
        Ei(nbai):=k;
        (inter-x(nbai), inter-y(nbai)):=(x(ns),y(ns));
        j:=liste(j);
      fintq
      ns:=ns+1 ; co sommet suivant fco
    fin
Fin;
```

```
Obtenir première ligne de hachurage
Début
    y_cour :=ypred-dy co ligne i fco
    y-fin:=ys(nbsom)
Fin;
```

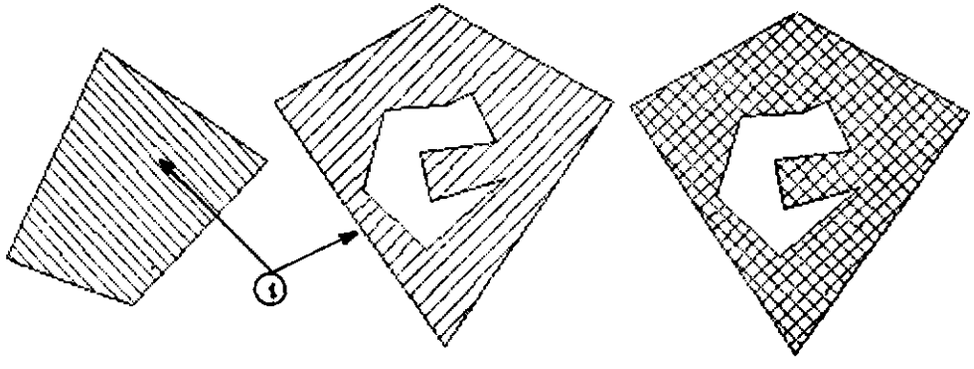
Hachurage proprement dit :
La ligne de hachurage est non finale si y_cour>y_fin.

```
Calcul des intersections des arêtes existantes dans Ei :
Début
    Pour k:=1 jusqu'à nbai faire
    Début
      inter_x(k):=inter_x(k)+dax(Ei(k));
      inter_y(k):=inter_y(k)+day(Ei(k));
    fin
Fin;

Ajouter les arêtes qui commencent dans la bande ]ligne i-1,ligne i]
Début
    Tantque (ns≤nbsom et ys(ns)≥y-cour) faire
    Début
      j=s(ns);
      Tantque j≠0 faire
        nbai:=nbai+1;
        Ei(nbai):=k;
        Inter-x (nbai):=(a(k)*y(ns)+b(k)*x(ns)-a(k)*ycour)/(b(k)-pente*a(k));
        Inter-y (nbai):=pente*inter-x (nbai)+y-cour;
        j:=liste(j);
      fintq
      ns:=ns+1;
    fin
Fin;

Enlever les arêtes qui finissent dans la bande ]ligne i-1, ligne i]
Début
    Pour i:=1 jusqu'à nbai faire
    Début
      k:=Ei(i);
      si ys(s(k))≤y-cour
        alors co supprimer Ei(i) fco
          si i≠nbai
            alors
                k:=i+1;
                Pour j=k à nbai faire
                Début
                    Ei(j-1):=Ei(j);
                    Inter(j-1):=inter(j);
          fsi     fin
          nbai:=nbai-1;
          i:=i-1
    fsi
    fin
Fin;
```



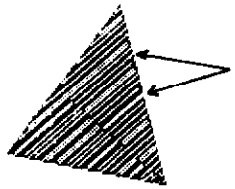
```
trier (Ei,inter);
Début
    trier(Ei,inter_x, inter_y) inter-x étant le maître
    (ordre croissant)
Fin tri;

Tracer les hachurages;
Début
    k1:=1;
    k2:=2;
    Tantque k2≤nbai faire
    Début
        afficher segment((inter-x (k1), inter-y(k1));
                         ((inter-x (k2), inter-y(k2));
        k1:=k1+2 ; k2:=k2+2;
    fin
Fin tracé;

Obtenir la ligne de hachurage suivante ;
Début
    y-pred:=y-cour ; co ligne i-1 fco
    y-cour:=y-pred-dy ; co ligne i fco
Fin;
```

Exemples de hachurage oblique :



Nous pouvons remarquer sur l'exemple ci-dessus que lors de l'affichage des lignes de hachurage, le passage de l'espace utilisateur (valeurs réelles) à l'espace écran (valeurs entières) produit des erreurs sur la distance comprise entre les lignes de hachurage ① (erreur variant de -1 à +1 unité écran).

Ce problème numérique ne nous permet pas d'effectuer un remplissage oblique correct, car comme nous le constatons sur l'exemple ci-après, tous les points de la tache ne sont pas affichés.



Nous retrouvons les erreurs commises sur la distance comprise entre les lignes de hachurage.

3.5.2.3 - *Conclusion*

La mise en oeuvre et l'évaluation de notre algorithme de hachurage oblique sans rotation nous permet de faire la remarque suivante :

- le principal inconvénient de cette méthode est la présence du tri des sommets dont la complexité demeure en 0 (nbsom.Log nbsom).

- l'inconvénient de l'algorithme de hachurage oblique avec rotations était les calculs supplémentaires effectués par les rotations successives, d'abord sur les sommets du contour puis sur les extrémités des lignes de hachurage.

Le problème est de connaître à partir de quel critère l'une ou l'autre des 2 solutions est la plus avantageuse.

Notre version du hachurage oblique, fondée sur la technique du suivi de contour, s'avère la plus performante dès que le nombre de lignes de hachurage est très supérieur au nombre d'arêtes (dès que nblh>2. nbam. nba où nbam est le nombre moyen d'arêtes coupant une ligne de hachurage, mais cette inégalité demande à être vérifiée dans la pratique).

3.6. Conclusion

De manière générale, les performances et les ressources requises par les algorithmes de remplissage ou de hachurage de taches dépendent de la complexité des contours. Mais le critère primordial demeure leur adaptation à la synthèse d'image dans un cadre d'application donné :

- intensité lumineuse variable ou constante,
- taches polygonales ou non,
- interactivité,
- mémoire disponible.

Pour chacune des classes d'algorithmes de remplissage, nous pouvons faire les remarques générales qui suivent.

. Les algorithmes de coloriage sont avant tout utilisables de manière interactive (détermination du point intérieur, connexité de la tache).

. La méthode de remplissage par codage du contour nécessite la plupart du temps une mémoire d'image supplémentaire et s'adapte mal à un ensemble de taches qui se chevauchent. Cependant, sa généralisation à des contours

quelconques est simple.

. Parmi ces diverses classes d'algorithmes, une méthode générale se dégage : il s'agit de la technique du suivi de contour. En effet, raisonnant à partir d'une représentation structurée du contour, elle permet non seulement de réaliser le remplissage d'une tache ou d'un ensemble de taches polygonales ou non, mais encore elle s'adapte facilement au hachurage, au découpage de taches polygonales en éléments simples et au parallélisme (voir <BLOCH 81>).

Nous montrerons dans le chapitre suivant que les principes du suivi de contour sont généralisables à certains problèmes de découpage et de nature géométrique.

Chapitre 4

Algorithmes de découpage et traitements de nature géométrique

4.1. Introduction

Il est extrêmement fréquent que, lors de la mise en page d'un dessin ou d'une image, on ait besoin de ne représenter sur l'écran qu'une partie de la scène considérée. Lorsqu'on travaille dans le plan, le problème classique consiste à utiliser une fenêtre, espace rectangulaire à bords parallèles aux axes (figure 4.1).

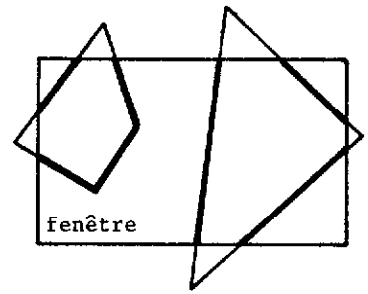


Figure 4.1.: Découpage d'une scène par une fenêtre.

Cependant, on peut avoir besoin de faire des calculs dans le cas où l'on ne dispose plus de la facilité donnée par le parallélisme aux axes. Le problème revient alors à découper une scène par une fenêtre constituée par un polygone convexe ou non, le nombre de ses côtés étant quelconque.

De façon plus générale, nous sommes parfois amenés à comparer deux polygones quelconques et à déterminer tous les polygones résultant de leur intersection (du découpage de l'un par rapport à l'autre). Nous utilisons ce découpage en particulier dans l'algorithme d'élimination des parties cachées d'une scène tridimensionnelle de ATHERTON et WEILER <ATHERTHON-WEILER 77>, ou à partir de l'ensemble des faces polygonales qui la composent, nous voulons obtenir un ordre total sur un ensemble de facettes. Ce découpage peut également s'avérer utile pour la manipulation d'image.

Ainsi, selon le type de la scène que nous manipulons (dessin ou image) et suivant le résultat que nous voulons obtenir (structures de données,

affichage sur l'écran) le problème du découpage peut être traité de manières très différentes. Dans le paragraphe suivant, nous ferons une classification des problèmes de découpage en dégageant les problèmes élémentaires de nature géométrique inhérents à chacun d'eux.

Pour l'ensemble des traitements abordés, nous avons constaté la disparité des méthodes proposées dans la littérature, provenant de la diversité des préoccupations qui ont conduit à la résolution du problème donné. A la lumière des chapitres précédents, nous montrerons dans le souci d'une démarche unitaire, comment nous avons utilisé les principes de la technique du suivi de contour pour résoudre les problèmes de découpage.

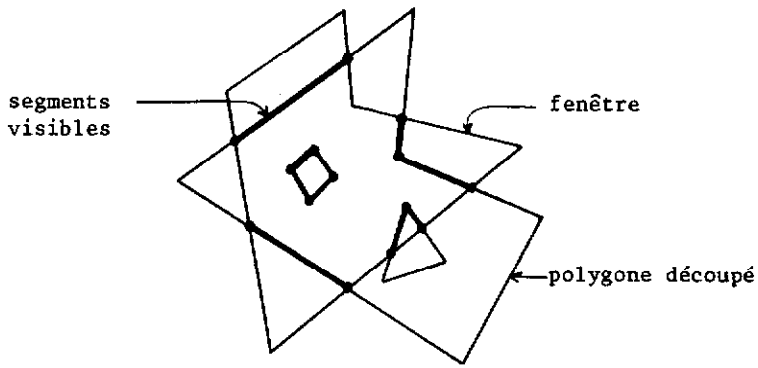
4.2. Classification des problèmes de découpage

Dans cette classification, nous ne raisonnerons que sur des contours polygonaux, mais elle peut être généralisée pour des contours quelconques.

1. Découpage d'un polygone par une fenêtre

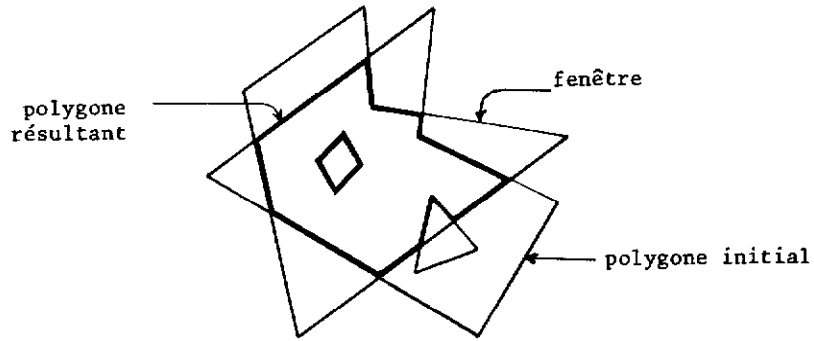
Le problème est de calculer la partie visible d'un polygone dans une fenêtre polygonale. Deux hypothèses peuvent être envisagées :

a) Si nous produisons simplement du dessin au trait sans tenir compte de la notion de contour, le problème élémentaire est le découpage d'un segment par un polygone :



On n'affiche que les parties des arêtes visibles dans la fenêtre.

b) Si nous voulons conserver la notion de contour, par exemple la notion de facette plane pour le dessin au trait, ou la notion de tache pour la synthèse d'image, le problème est de calculer le polygone résultant, situé à l'intérieur de la fenêtre qui résulte du découpage du polygone initial par celle-ci.



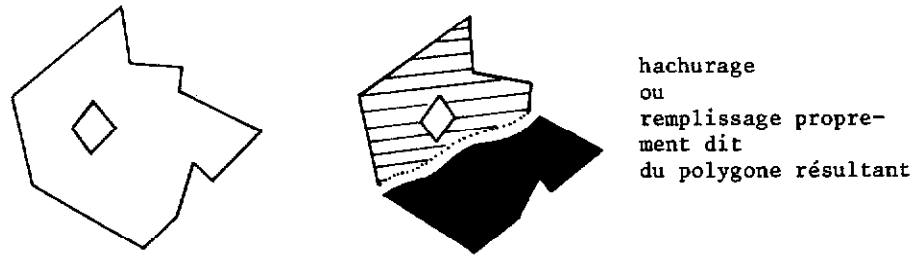
On calcule ici l'ensemble des contours (intérieurs et extérieurs) qui composent le polygone résultant du découpage.

2. Découpage d'une tache par une fenêtre polygonale

Nous voulons ici afficher la partie visible d'une tache (hachurée ou remplie) dans une fenêtre polygonale.

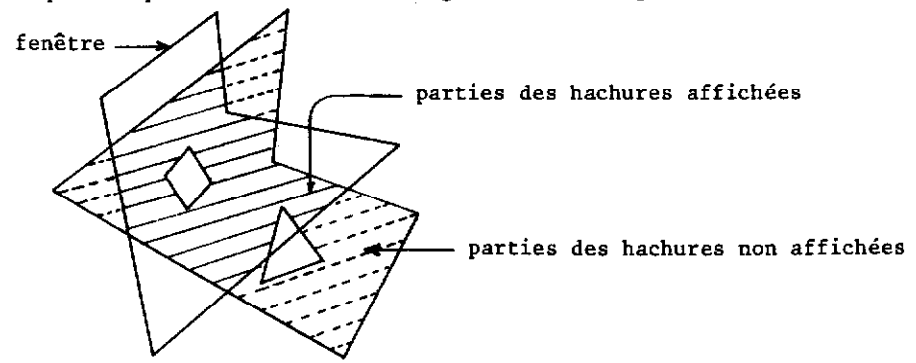
Deux solutions peuvent être envisagées :

a) Déterminer le polygone résultant du découpage du polygone initial par la fenêtre (comme en 1-b). Puis hachurer ou remplir ce polygone (on se ramène au problème élémentaire du remplissage ou du hachurage d'une tache).



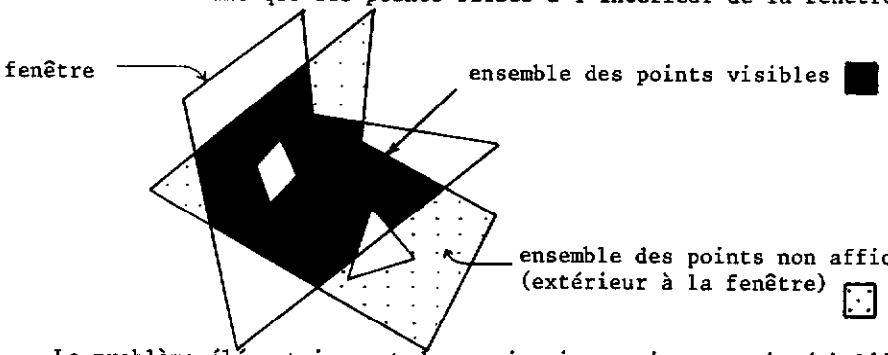
polygone résultant

b) Pour le hachurage, on hachure tout le polygone initial en n'affichant que les parties visibles des lignes de hachurage :



Le problème élémentaire demeure le découpage d'un segment de droite (hachure) par un polygone.

Pour le remplissage proprement dit, on remplit tout le polygone initial en n'affectant que les points situés à l'intérieur de la fenêtre :



Le problème élémentaire est de savoir si un point est situé à l'intérieur ou non du polygone (comparaison point-contour).

Dans le cas du hachurage (et peut-être aussi du remplissage),s'il s'avère nécessaire d'afficher les parties visibles du contour, nous nous ramenons alors au problème exposé en (1-a).

Pour la première solution (a) le temps de calcul et d'affichage est proportionnel à la complexité de l'image initiale et non à la complexité de l'image visible. Mais si le temps de calcul du remplissage ou du hachurage dans la seconde méthode (b), est proportionnel à la complexité de l'image visible, il faut lui ajouter le temps de calcul du découpage proportionnel à la complexité des contours. Le problème est de savoir, dans quels cas, l'une ou l'autre des deux solutions est la plus avantageuse.

En (b), nous proposons d'afficher la partie visible d'une tache, en effectuant le remplissage ou le hachurage de toute la tache initiale, mais en n'affectant que le sous-ensemble de l'ensemble des points ou des lignes de hachurage visibles dans la fenêtre.

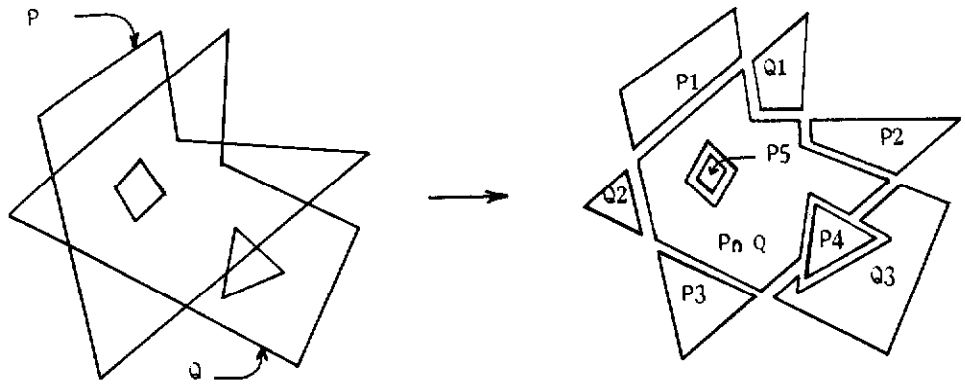
L'appartenance d'un point ou d'un segment à la fenêtre, peut être déterminé dans ce cas, de différentes façons :

- Soit en utilisant l'un des algorithmes de comparaison point-contour ou de découpage d'un segment par un polygone.

- Soit en effectuant le remplissage ou le hachurage de la fenêtre et de la tache simultanément et en n'affichant que les points ou les parties des lignes de hachurage qui appartiennent à l'intérieur de la tache et à l'intérieur de la fenêtre.

3. Découpage d'un polygone par un autre

Il s'agit de déterminer tous les polygones qui résultent de l'intersection de deux polygones quelconques.



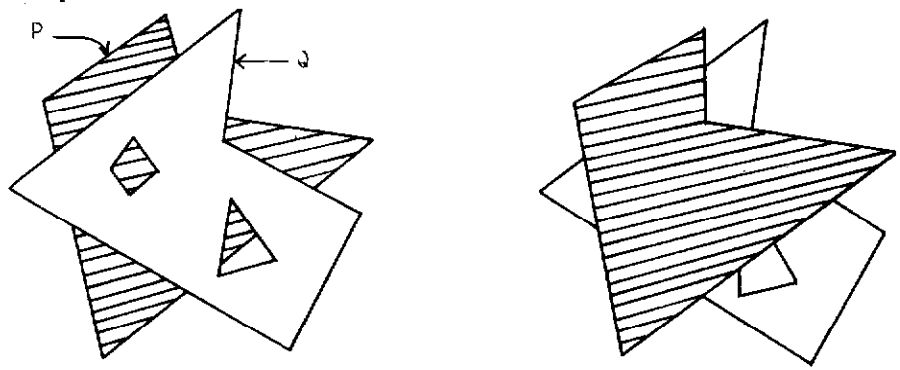
Si P découpe Q on obtient P∩Q, Q1, Q2, Q3

Si Q découpe P on obtient P∩Q, P1, P2, P3, P4, P5

On peut aussi vouloir obtenir P∩Q, P1, P2, P3, P4, P5 et Q1, Q2, Q3.

4. Découpage d'une tache par une autre

Pour composer une image en deux dimensions, nous manipulons un ensemble de taches. Les taches peuvent se superposer ou se combiner, suivant certaines règles de composition. Ainsi, quand deux tâches se chevauchent, nous devons découper l'une par rapport à l'autre, suivant le résultat escompté :



Q devant P ou P devant Q

Nous pouvons envisager deux solutions :

a) On découpe l'un des contours par le contour de la tache qui se trouve devant, en conservant les polygones résultants extérieurs à cette dernière : c'est le problème énoncé en (3).

Si Q est devant P on obtiendra P1, P2, P3, P4, P5

Si P est devant Q on obtiendra Q1, Q2, Q3.

Ensuite, on affiche les taches extérieures ainsi définies et la tache qui est devant avec leurs couleurs appropriées.

Dans ce cas, si une ambiguïté, illustrée sur la figure ci-dessous intervient, nous devons calculer le résultat du découpage des deux taches entre elles.

ambiguïté

Sur l'exemple de la figure du paragraphe (3), nous devrons déterminer P∩Q, P1, P2, P3, P4, P5, Q1, Q2, et Q3 où P∩Q sera de la couleur de P, si P est devant Q, ou de Q si Q est devant P, où P1, P2, P3, P4 et P5 seront de la couleur de P, et où Q1, Q2, et Q3 seront de la couleur de Q. Puis à partir de ce nouvel ensemble de taches, on pourra réaliser le découpage de la même façon avec les autres taches de la scène qui sont en conflit. Une fois le traitement de toute la scène effectué, on affichera ce nouvel ensemble de taches.

b) On découpe les taches non plus en raisonnant sur leurs contours, mais sur leurs surfaces. On retrouve ici le problème de la manipulation d'un ensemble de taches (voir <EDMONDS et al. 82>).

Avant d'entreprendre le découpage, on peut essayer tout d'abord de déterminer, si les deux contours polygonaux à comparer sont disjoints, si l'un contient l'autre ou s'ils se coupent (même chose pour les taches).

5. Conclusion :

Dans la classification des problèmes du découpage, nous avons dégagé quatre grandes classes :

1- Découpage d'un polygone par une fenêtre,
2- Découpage d'une tache par une fenêtre,
3- Découpage d'un polygone par un autre,
4- Découpage d'une tache par une autre.

Mais pour l'étude de chacune d'entre elles, il conviendra de distinguer les cas particuliers où les fenêtres, les polygones ou les taches sont rectangulaires aux bords parallèles aux axes, convexes ou quelconques, pour tirer parti dans chacun des cas de leurs propriétés intrinsèques.

D'autre part, nous avons mentionné l'utilisation de trois problèmes élémentaires :

1- La comparaison d'un point et d'un contour
2- Le découpage d'un segment de droite par un contour
3- L'intersection de deux segments de droite.

Le tableau récapitulatif qui suit, nous redonne toutes les opérations géométriques et de découpage citées plus haut, en y ajoutant des opérations élémentaires telles que comparaison entre deux points, entre un point et

	Point	Segment	Droite	Contour	Tache
Point	2 points sont-ils confondus ?	position d'un point par rapport à un segment	position d'un point par rapport à une droite	position d'un point par rapport à un contour	position d'un point par rapport à une tache
Segment		intersection de deux segments de droite	intersection d'une droite et d'un segment de droite	intersection d'un segment de droite et d'un contour	intersection d'un segment de droite et d'une tache
Droite			intersection de deux droites	intersection d'une droite et d'un contour	découpage d'une tache par une droite
Contour				intersection de deux contours	découpage d'une tache par un contour
Tache					intersection de deux taches

Table 3 : Traitements géométriques et problèmes élémentaires de découpage.

un segment ou une droite. En effet, un algorithme de découpage peut combiner plusieurs opérations élémentaires. Par exemple, l'algorithme d'intersection de deux contours convexes de SHAMOS, utilise la comparaison d'un point et d'un contour, la comparaison d'un point et d'une droite (ou segment de droite) et le calcul d'intersection de deux segments de droite.

4.3. Comparaison d'un point et d'un contour polygonal

4.3.1 - INTRODUCTION

Le problème à résoudre est de savoir si un point est situé à l'intérieur ou à l'extérieur d'un contour polygonal. La comparaison d'un point et d'un contour est très souvent rencontrée, en particulier, pour l'identification d'un contour ou d'une tache de manière interactive, dans le découpage d'un contour par un autre et dans certains algorithmes d'élimination de parties cachées.

Nous allons étudier les différents algorithmes qui résolvent ce problème en différenciant ceux qui s'appliquent uniquement aux contours convexes de ceux qui traitent les contours quelconques définis par la suite ordonnée de leurs nbsom sommets s_i où i=1,...,nbsom (voir Table 4).

4.3.2 - COMPARAISON POINT-CONTOUR CONVEXE

4.3.2.1 - Introduction

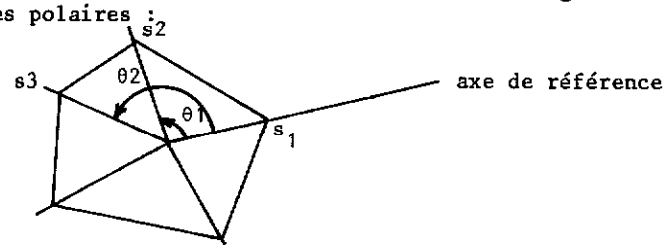
Il existe deux méthodes qui ne traitent que des contours convexes : la méthode de DAHAN LE TUAN, et celle de SHAMOS.

La complexité du premier algorithme est en O(Nbsom). Par contre, si dans l'algorithme de SHAMOS nous effectuons un prétraitement sur le contour convexe dont le comportement est en O(Nbsom), la comparaison effective du point et du contour convexe possède une complexité en O(Log Nbsom).

4.3.2.2 - Méthode de SHAMOS (voir <SHAMOS 75>)

Le principe :

A partir d'un point C intérieur au polygone convexe, on divise le plan en N secteurs angulaires. C est considéré comme l'origine d'un système de coordonnées polaires :



Soit p le point à comparer au contour.

Méthode	Nature du contour	Cas particuliers	Complexité théorique	Actions élémentaires	Remarques
DAHAN LE TUAN	Convexe		O(n)	équation d'une droite	si point extérieur, comparaison avec les arêtes non nécessairement exhaustive.
SHAMOS	Convexe orienté		.Prétraitement en O(n) .Traitement en O(logn) (recherche dichotomique)	fonction trigonométrique arc-tangente	méthode intéressante si contour utilisé plusieurs fois
CALCUL d'intersections	quelconque	1/2 droite passant par un ou plusieurs sommets	O(n)	intersection de segments	
ANGLES CAPABLES	quelconque	aucun renseignement fourni si le point est sur le contour	O(n)	fonction trigonométrique arc tangente	cas convexe : méthode exhaustive
CODAGE	quelconque	sommet appartenant au domaine entourant le point p	O(n)	code	cas convexe : méthode exhaustive
SUIVI DE CONTOUR	quelconque		O(n)	intersection de segments simplifiée	

n=nombre de sommets

Table 4 : Tableau récapitulatif des méthodes de comparaison POINT-CONTOUR

On détermine son angle polaire (cs_1,cp), puis suivant une recherche dichotomique, on détermine le secteur auquel il appartient. Une fois le secteur calculé, on regarde de quel côté de l'arête se situe le point p.

L'algorithme :

Soit (xc,yc) les coordonnées du point intérieur au contour, et Tangle (1:nbsom) le tableau des angles polaires ordonnés par valeur croissante. On obtient l'algorithme suivant :

```
Comparaison POINT CONTOUR CONVEXE
Début co méthode de SHAMOS fco
    frontière:=faux;
    vx1:=xc-x(1);
    vy1:=yc-y(1);
    vx2:=xc-xp;
    vy2:=yc-yp;
    angle:=arc tangente (vx1*vy2-vy1*vx2,vx1*vx2+vy1*vy2);
    Recherche dichotomique (angle, Tangle, nbsom, indice);
    co le point se situe dans le secteur défini par
    ((xc,yc),(x(indice),y(indice))),((xc,yc),(x(indice+1),y(indice+1)))fco

    a:=x(indice+1)-x(indice);
    b:=y(indice+1)-y(indice);
    c:=a.y(indice)+b.x(indice);
    prod:=(a.yp+b.xp-c)(a.yc+b.xc-c);
    si Prod <=0
       alors si Prod=0
                alors si (x(indice)-xp)(x(indice+1)-xp)
                         +(y(indice)-yp)(y(indice+1)-yp)<=0
                       alors frontière:=vrai;
                       sinon dehors:=vrai;
                    fsi
                sinon dehors:=vrai;
             fsi
       sinon dehors:=faux;
    fsi
Fin
```

Critique :

Le principal désavantage de cette méthode est l'utilisation de fonctions trigonométriques inverses très coûteuses en temps de calcul.

Par contre, dans la mesure où nous avons un très grand nombre de points à comparer à un contour convexe, le calcul des angles polaires, étant effectué une fois pour toutes, le comportement de cette solution demeure en O(Log Nbsom).

4.3.3 - COMPARAISON POINT-CONTOUR NON CONVEXE

4.3.3.1 - *Introduction*

Pour traiter la comparaison d'un point et d'un contour quelconque de nombreuses méthodes existent :

- dénombrement d'intersections,
- angles capables,
- codage.

Mais ces algorithmes ne surmontent pas facilement toutes les singularités (voir cas particulier de la Table 4). Pour pallier simplement ces difficultés, nous proposons un algorithme, fondé sur le raisonnement du "suivi de contour".

4.3.3.2 - *Méthode du suivi de contour*

Principe

On résoud le contrôle de parité pour le point P, en utilisant le raisonnement du remplissage de tache par la méthode du suivi de contour (voir figure 4.2).

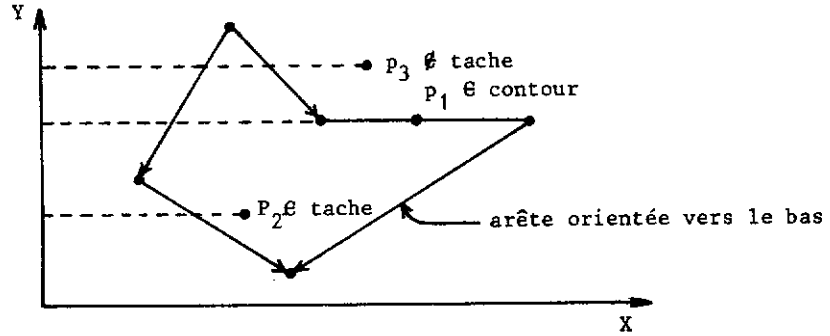


Figure 4.2.: Comparaison point-contour : "suivi de contour".

Pour ce faire, on dénombre les intersections de la demi-droite horizontale située à gauche du point avec l'ensemble des arêtes du contour en ne prenant pas en compte les arêtes horizontales et celles qui finissent sur la droite.

L'algorithme :

```
Comparaison POINT-CONTOUR NON CONVEXE
Début co méthode du "suivi de contour" fco
      co (xp,yp) point à tester,
         nba = nombre d'arêtes,
         ((xdeb,ydeb),(xfin,yfin)) arête de contour fco
      intérieur:=faux ; contour:=faux;
      xdeb:=x(premier sommet) ; ydeb:=y(premier sommet);
      yfin:=y(sommet suivant) ; yfin:=y(sommet suivant);
      Tant que (arête non finale et non contour) faire
      Début
           si ydeb≠yfin co on supprime les arêtes horizontales fco
             alors si ydeb>yfin co on oriente les arêtes vers le bas fco
                       alors si yfin<yp
                                 alors si ydeb>=yp
                                           alors si yfin<=yp ou xdeb<=xp
                                                     alors xinter:=(ydeb-y)*
                                                     ((xfin-xdeb)/(ydeb-yfin));
                                                        si xinter<=xp
                                                         alors si xinter=xp
                                                                 alors contour:=vrai;
                                                                 sinon intérieur
                                                                 :=¬intérieur;
                                                        fsi    fsi
                                                  fsi

                                           fsi
                                 sinon si yfin=yp alors si xfin=xp alors
                                 contour=vrai fsi
                       fsi
                  sinon (même chose que alors en inversant les extré-
                         mités)
             fsi
        sinon si (ydeb=yp et xdeb<=xp<=xfin) alors contour:=vrai fsi
      fsi
      xdeb:=xfin ; ydeb:=xfin;
      xfin:=x(sommet suivant) ; xfin:=y(sommet suivant);
    fin
Fin
```

Critique

Les avantages de la méthode de "suivi de contour" sont multiples :

- les singularités sont traitées simplement;
- le polygone peut être quelconque : en particulier les arêtes peuvent se croiser et le polygone peut avoir plusieurs contours, intérieurs ou extérieurs;

- on peut tirer parti de la convexité : on s'arrête dès que l'on a détecté deux intersections (point situé à l'extérieur).

La complexité théorique est en O(nbsom).

4.3.4 - CONCLUSION

Sur des contours convexes, l'algorithme de SHAMOS, tire le meilleur profit de cette information quand le contour est utilisé plusieurs fois et que le nombre d'arêtes est assez important.

Des quatre solutions qui traitent des contours quelconques (non convexes), la méthode du "suivi de contour" apparaît comme la plus intéressante :

- Les singularités ou les cas particuliers sont surmontés simplement sans calculs supplémentaires.

- Les contours peuvent posséder des trous (contours intérieurs), des contours extérieurs disjoints et des arêtes qui se croisent, ce qui est rarement le cas pour les autres méthodes.

- L'évaluation des divers algorithmes a montré que cette méthode s'avère la plus avantageuse car les calculs sont simples (voir <HEGRON 83c>).

- On peut tirer parti de la convexité.

4.4. Découpage d'un segment de droite par un contour polygonal

4.4.1 - INTRODUCTION

Le découpage d'un segment de droite par un contour est essentiellement utilisé pour la production de dessin au trait. Il s'agit dans la plupart des cas, de calculer la partie visible d'une scène dans une fenêtre. La scène étant formée d'un ensemble de segments de droite, seul le type de la fenêtre déterminera l'algorithme à employer, à savoir, rectangulaire (sous entendu aux bords parallèles aux axes), convexe ou non convexe .

Nous allons envisager dans les paragraphes suivants, les algorithmes de découpage, par rapport à ces trois types de fenêtre. La table 5 fait le bilan des caractéristiques des diverses méthodes.

4.4.2 - DECOUPAGE PAR RAPPORT A UNE FENETRE RECTANGULAIRE

4.4.2.1 - Introduction

Le problème du découpage d'une scène par rapport à une fenêtre rectangulaire, est le plus souvent rencontré, pour :

- la mise en page d'un dessin ou d'une image ;
- le zoom sur une partie de la scène ;
- les algorithmes d'élimination de parties cachées pour la visualisation de scènes tridimensionnelles : en particulier dans la méthode de WARNOCK.

METHODE	FENETRE	OPERATIONS ELEMENTAIRES	COMPORTEMENT	REMARQUES
SUTHERLAND (version dichotomique)	rectangulaire	-comparaison point contour (avec codage) -calcul du milieu d'un segment	$O(Log_2(taille\ segment))$	-adapté au câblage -parallélisme possible -généralisation à un contour convexe délicate
SUTHERLAND-COHEN	rectangulaire et convexe	-codage d'un point par rapport à un contour -calcul d'intersection de segments de droite	O(2n) n:nombre d'arêtes du contour	généralisation au contour convexe délicate (singularités)
PAVLIDIS	rectangulaire et convexe	-comparaison point-segment (coordonnées homogènes) -calcul d'intersection de segments de droite	O(n)	-comparaison avec les arêtes non nécessairement exhaustive et orientation du contour dans le cas convexe quelconque -généralisation aux contours non convexes très délicate et coûteuse(singularité)
SUIVI DE CONTOUR	non convexe quelconque	-comparaison d'un point par rapport à une droite -intersection d'une droite et d'un segment de droite	O(n)	-comparaison non nécessairement exhaustive dans le cas convexe -traitement très simple des singularités -contour non convexe quelconque : contours disjoints, trous, arêtes qui se croisent

Table 5 : Découpage d'un segment de droite par une fenêtre polygonale.

. La méthode triviale consiste à vérifier systématiquement si le point calculé est ou n'est pas visible. Cette méthode connue sous le nom de *fenêtrage par effacement* présente cependant l'inconvénient de demander un temps de calcul et d'affichage proportionnel à la complexité de l'image initiale et non à la complexité de l'image visible.

. Pour le découpage d'un segment de droite par une fenêtre rectangulaire, la réalisation câblée de l'algorithme de SUTHERLAND (version dichotomique) apparaît comme la méthode la plus adaptée. Du point de vue logiciel, si un grand nombre de segments sont horizontaux ou verticaux, la méthode de PAVLIDIS demeure plus performante que celle de SUTHERLAND-COHEN.

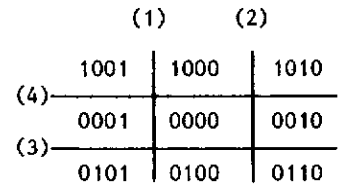
Nous verrons d'autre part, dans le chapitre suivant, que les techniques de SUTHERLAND et de PAVLIDIS, sont généralisables pour une fenêtre convexe.

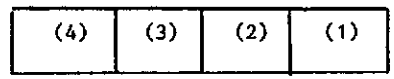
4.4.2.2 - *L'algorithme de SUTHERLAND-SPROULL*

La méthode proposée pour la première fois par SUTHERLAND et SPROULL dans <SUTHERLAND-SPROULL 68>, consiste à calculer exactement les parties de l'image qui sont visibles, la liste d'affichage ne comportant que les ordres relatifs à celles-ci. Le procédé de *découpage* s'applique essentiellement à des segments de droite, mais peut être généralisé à des symboles graphiques.

Le principe est le suivant :

On prolonge les bords de la fenêtre de manière à délimiter neuf régions dans l'espace utilisateur. A chacune des régions on associe un code binaire (4 bits), l'intérieur de la fenêtre étant codé par '0000' :





formation du code

Pour chaque droite, le bit associé est égal à 0 pour l'ensemble des points situés dans le demi-plan, auquel appartient l'intérieur du rectangle dont le code est nul ; ce bit est égal à 1 pour l'autre demi-plan.

L'algorithme de découpage consiste alors à trouver quelle portion d'une droite est visible dans la fenêtre ; pour cela, on commence par associer aux extrémités du segment le numéro de la région du plan dans lequel elle se trouve.

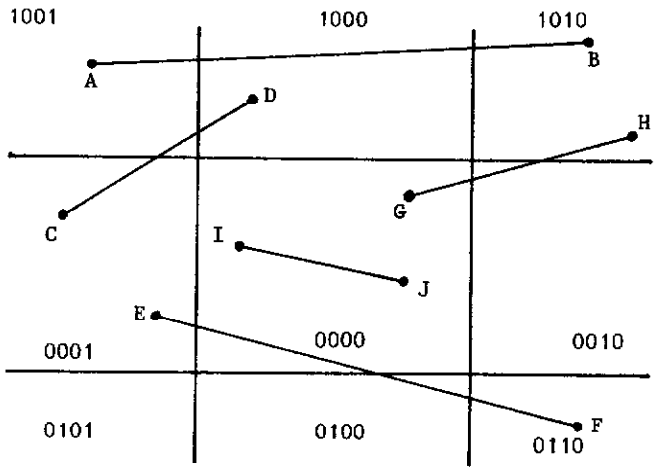


Figure 4.3.: Algorithme de découpage.

On distingue trois sortes de segments de droite (figure 4.3.):

- celles dont les deux extrémités sont en dehors de la fenêtre : segments AB, CD, EF,
- celles dont les deux extrémités sont dans la fenêtre (deux codes 0000): segment IJ,
- celles dont une seule extrémité est dans la fenêtre (un code 0000): segment GH.

Le principe d'élimination est alors simple :

- on fait l'intersection des deux codes binaires associés aux extrémités,
- si le résultat de l'intersection est non nul, le segment est hors de la fenêtre et non visible,
- si le résultat est nul, on divise le segment en deux parties et on recommence à appliquer le processus jusqu'à élimination des parties cachées et découverte d'un segment dont les codes soient tous les deux (0000).

Cet algorithme utilise trois fonctions élémentaires qui sont :

- déterminer le code d'un point,
- intersection de deux codes,
- calculer le milieu d'un segment.

Le milieu d'un segment (x1,y1),(x2,y2) est donné par ((x2+x1)/2, (y2+y1)/2) et peut être calculé pour des coordonnées entières avec deux additions et deux décalages de bits à droite.

Si cet algorithme est utilisé pour traiter des figures dans l'espace écran, où les coordonnées des points ont des valeurs entières et soit 2^k la taille maximale d'un segment,pour l'étude d'un segment nous ferons au plus k comparaisons d'un segment par rapport au contour.

Le comportement de l'algorithme est logarithmique en fonction de la taille du segment.

4.4.2.3 - L'algorithme de PAVLIDIS <PAVLIDIS 82>

PAVLIDIS décompose également l'espace en 9 régions (voir figure 4.4.). Il privilégie dans un premier temps le cas particulier des segments situés à gauche, à droite, dessus ou dessous la fenêtre, et le cas des segments verticaux et horizontaux. Dans le cas général, après avoir calculé les signes des 4 coins de la fenêtre par rapport à la droite support du segment, il vérifie si les coins consécutifs pris deux à deux sont de part et d'autre de la droite support du segment en comparant les signes relatifs des deux coins. Si oui (signes opposés), le segment peut couper le côté de la fenêtre joignant les deux coins concernés.

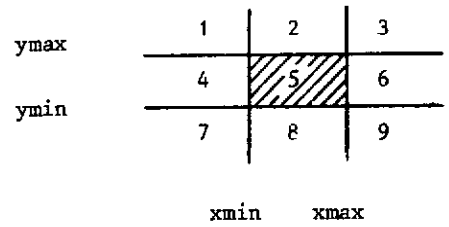


Figure 4.4.: Découpage avec une fenêtre rectangulaire.

L'algorithme de PAVLIDIS qui suit réalise le découpage du segment de droite d'extrémités (x1,y1) et (x2,y2) par une fenêtre rectangulaire d'abscisses xmin et xmax et d'ordonnées ymin et ymax.

```
DECOUPAGE.1 (xmin,xmax,ymin,ymax : entier)
co découpage d'un segment de droite par une fenêtre rectangulaire : algo-
   rithme de Pavlidis fco
Début
   1-si (x1<xmin et x2<xmin) ou (x1>xmax et x2>xmax) ou
        (y1<ymin et y2<ymin) ou (y1>ymax et y2>ymax)
        alors fin découpage . fsi
   2-si (y1>y2) alors co échange les extrémités fco
                       x:=x1, x1:=x2, x2:=x;
                       y:=y1, y1:=y2, y2:=y;
   3-si y1=y2
        alors si x1>x2 alors échanger les extrémités fsi
              si x1<xmin alors x1:=xmin fsi
              si x2>xmax alors x2:=xmax fsi
              afficher segment (x1,y1,x2,y2);
              fin découpage.
     fsi
   4-si x1=x2
        alors si y1<ymin alors y1:=ymin fsi
              si y2>ymax alors y2:=ymax fsi
              afficher segment (x1,y1,x2,y2);
              fin découpage.
     fsi
```

```
5-dy:=y1-y2 ; dx:=x1-x2 ; dxy:=x1.y2-y1.x2;
  qx:=xmin.dy ; qX:=xmax.dy ; qy:=ymin.dx ; qY:=ymax.dx;
  u1:=qx-qy+dxy ; u2:=qx-qY+dxy ; u3:=qX-qY+dxy ; u4:=qX-qy+dxy
6- j:=0;

7-si u1.u2<0
    alors co u1 et u2 sont de signes contraires fco
          j:=j+1;
       si x1<xmin
       alors y1:=(xmin.dy+dxy)/dx ; x1:=xmin;
             mettre à jour dx,dy et dxy;
       si x2<xmin
       alors y2:=(xmin.dy+dxy)/dx ; x2:=xmin;
             mettre à jour dx,dy et dxy;
       fsi
  fsi
8-si u2.u3<0
    alors co u2 et u3 sont de signes contraires fco
          j:=j+1;
       si y2>ymax
          alors x2:=(ymax.dx-dxy)/dy ;y2 :=ymax;
                mettre à jour dx,dy et dxy;
       fsi
  fsi
9-si u3.u4<0
    alors co u3 et u4 sont de signes contraires fco
          j:=j+1;
       si x1>xmax
          alors y1:=(xmax.dy+dxy)/dx ; x1:=xmax;
                mettre à jour dx,dy et dxy;
       fsi
       si x2>xmax
          alors y2:=(xmax.dy+dxy)/dx ; x2:=xmax;
                mettre à jour dx,dy et dxy;
       fsi
  fsi
10-si u4.u1<0
    alors co u4 et u1 sont de signes différents fco
          j:=j+1;
       si y1<ymin
         alors x1:=(ymin.dx-dxy)/dy ; y1:=ymin;
               mettre à jour dx,dy et dxy;
       fsi
  fsi
11-si j>0 alors afficher segment (x1,y1,x2,y2) fsi
fin découpage 1
```


4.4.3 - DECOUPAGE PAR RAPPORT A UN CONTOUR CONVEXE

4.4.3.1 - *Introduction*

La version de SUTHERLAND peut être généralisée aux contours convexes (voir <LUCAS et al.79>) mais beaucoup de cas particuliers rendent sa mise au point particulièrement délicate (cas des segments colinéaires avec les bords,...).

Nous redonnons ci-dessous l'algorithme de PAVLIDIS qui résoud plus facilement ces difficultés.

4.4.3.2 - *Algorithme de PAVLIDIS* (<PAVLIDIS 82>)

PAVLIDIS utilise pour son algorithme, la représentation d'un point P par ses coordonnées homogènes :

$$P=\begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

Pour rester cohérent avec les autres algorithmes développés ici, nous prendrons w=1 où (x,y) représente les coordonnées absolues du point P dans le plan xoy.

Le problème :

Soient $s_1, s_2, \ldots, s_n$ les sommets d'un polygone convexe. Et soit un segment de droite, défini par ses extrémités P1 et P2. Le problème est de calculer la partie du segment située à l'intérieur du polygone.

L'idée :

Soient (x_i, y_i) les coordonnées du sommet s_i et (x_j, y_j) celle du point $p_j (j=1,2)$.

Evaluons les n valeurs S_i tels que :

$$S_i = \det(s_i, p1, p2)$$
$$= X_i(y1-y2) + Y_i(x2-x1) + (x1.y2-y1.x2) \qquad (1)$$
$$i=1,2,\ldots n.$$

A partir de ce calcul, nous dégageons les cas suivants :

1- Les valeurs S_i sont de même signe et différentes de zéro : le polygone est situé du même côté du segment et le segment est extérieur au polygone.

2- Une des valeurs S_i est nulle et les autres sont de même signe : la droite passe par un sommet mais reste extérieure au polygone.

3- Deux d'entre elles sont nulles et les autres sont de même signes : la droite passe par l'un des côtés du polygone convexe sans couper d'autres côtés (convexité).

4- Une ou deux d'entre elles sont nulles et les autres ont des signes différents : la droite coupe le polygone en passant par un ou deux de ses sommets.

5- Aucune des valeurs S_i est nulle mais elles possèdent des signes différents : nous rencontrons alors deux changements de signe en parcourant le périmètre du polygone (comme en 4).

Soient s_j,s_{j+1} et s_k,s_{k+1} les paires de sommets où ce changement de signe intervient, et L1 et L2 les segments respectifs qu'ils définissent (S_i peut être nulle pour l'un des sommets de l'une ou des deux paires (cas 4)).

Connaissant maintenant les deux arêtes du polygone qui coupe la droite passant par le segment (p1,p2), il faut déterminer la position relative des points p1 et p2 par rapport aux deux segments L1 et L2 à partir du signe des quatre quantités :

$$u1 = \det(p1,s_j,s_{j+1}) \qquad (2\text{-a})$$
$$= x1\,(Y_j-Y_{j+1})+y1(X_{j+1}-X_j)+(X_jY_{j+1}-Y_jX_{j+1})$$
(position du point p1 par rapport à L1)

$$u2 = \det(p_2,s_j,s_{j+1}) \qquad (2\text{-b})$$
$$= x2\,(Y_j-Y_{j+1})+y2(X_{j+1}-X_j)+(X_jY_{j+1}-Y_jX_{j+1})$$
(position du point p2 par rapport à L1)

$$u3 = \det(p1,s_k,s_{k+1}) \qquad (2\text{-c})$$
$$= x1\,(Y_k-Y_{k+1})+y1(X_{k+1}-X_k)+(X_kY_{k+1}-Y_kX_{k+1})$$
(position du point p1 par rapport à L2)

$$u4 = \det(p2,s_k,s_{k+1}) \qquad (2\text{-d})$$
$$= x2\,(Y_k-Y_{k+1})+y2(X_{k+1}-X_k)+(X_kY_{k+1}-Y_kX_{k+1})$$
(position du point p2 par rapport à L2).

Si nous parcourons le contour dans le sens des aiguilles d'une montre, le signe de U sera négatif quand un point se situera à l'intérieur du polygone. Cette expression sera nulle si ce point appartient au contour.

Soit L le segment (p1,p2) et (q1,q2) les extrémités du segment à afficher résultant du découpage de L par le polygone. L'algorithme qui suit, calcule q_1 et q_2 en tenant compte des différentes configurations possibles des extrémités p_1 et p_2 du segment L.

L'algorithme :

Nous supposons utiliser les structures de données suivantes :

. Les tableaux X(1..N) et Y(1..N) contiennent les coordonnées des N sommets du polygone.

. (x1,y1) et (x2,y2) sont les coordonnées des points p1 et p2.

. Les tableaux B(0..2,0..2) et C(0..2,0..2) contiennent respectivement les premières et secondes coordonnées des paires de sommets (s_j,s_{j+1}), et (s_k,s_{k+1}).

. Le tableau v(0..4) contient les signes des valeurs u_i, i=1 à 4, représentant les signes des extrémités du segment par rapport aux 2 arêtes coupées par la droite support.
(qn)n=1,2 sont les coordonnées du segment à afficher.

```
DECOUPAGE-2
co découpage d'un segment de droite L par un polygone convexe.
   algorithme de PAVLIDIS fco
début
  co Parcours du contour et calcul des arêtes coupées par la droite fco
     X(n+1):=X(1) ; Y(n+1):=Y(1);
     dx:=x2-x1 ; dy:=y1-y2 ; dxy:=x1.y2-y1.x2;
     i:=1
     S:=X(i).dy+Y(i).dx+dxy ; co équation (1) fco
     u:=signe(S);
     k:=0 ; co nombre d'arêtes coupées fco
     i:=2;
     Tant que (i<=n+1 et k≠2) faire
     début
         Si:=X(i).dy+Y(i).dx+dxy;
         v:=signe(Si);
         si u≠v et u≠0
           alors co changement de signe fco
                 k:=k+1;
                 B(k,1):=X(i-1) ; B(k,2):=Y(i-1);
                 C(k,1):=X(i) ; C(k,2):=Y(i);
           sinon si (u=0 et v=0) alors fin découpage - 2.
         fsi
         u:=v;
     fin
  co calcul et affichage de la partie visible du segment L fco
     si k=2
        alors u1:=det (p1,B(1,*), C(1,*)) ; co équations 2 fco
              u2:=det (p2,B(1,*), C(1,*)) ;
              u3:=det (p1,B(2,*), C(2,*)) ;
              u4:=det (p2,B(2,*), C(2,*)) ;
              v(1):=signe(U1) ; v(2):=signe (U2);
              v(3):=signe(U3) ; v(4):=signe (U4);
              si (v(1)=v(3) et v(1)≠v(2) et v(1)≠v(3))
                alors fin découpage-2
                sinon Pour m:=1 à 2 faire
                      début
                          si v(m)≠v(m+2)
                            alors si v(m)<0
                                    alors qm:=intersection (L,L2);
                                    sinon qm:=intersection (L,L1);
                                  fsi
                            sinon qm:=pm;
                          fsi
                      fin
              fsi
        sinon fin découpage-2;
     fsi
     afficher segment(q1,q2);
fin découpage-2
```

Traitement des singularités :

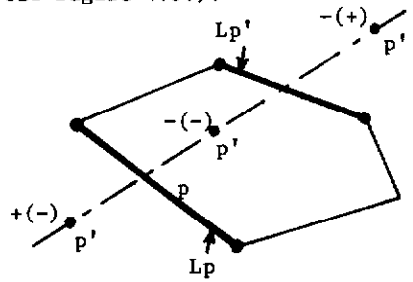
Nous rencontrons une singularité si la droite support du segment de droite passe par l'un des sommets du polygone ou bien si l'une des extrémités du segment appartient à l'une des arêtes du polygone.

- Parcours du contour : il y a une singularité s'il existe i tel que le signe (Si) est nul. Dans le cas où deux sommets consécutifs ont un signe nul (u=v=0) nous considérons alors le segment L invisible. Sinon, si un sommet s_i est de signe nul, nous prenons en compte uniquement l'arête (s_{i-1},s_i) (signe(S_{i-1})≠signe(S_i) et signe (S_{i-1})≠0).

- Calcul de la partie visible du segment L : il y a une singularité si une ou deux valeurs U_i est nulle.

Si deux valeurs U_i sont nulles alors $q_1=p_1$ et $q_2=p_2$ car p_1 et p_2 appartiennent au contour du polygone et L est contenu par ce polygone (convexité).

Si une seule des valeurs U_i est nulle, alors une seule des extrémités (p) du segment L appartient au contour. Dans ce cas, si le signe de l'autre extrémité (p') par rapport à l'arête Lp à laquelle appartient l'extrémité p, est positive, alors L est invisible. Sinon si ce signe est négatif, alors q1=p et q2=p' si signe de p par rapport à Lp' est négatif, q2=Lp'∩L sinon (voir figure 4.5.).



(entre parenthèses: signe du point p' par rapport à Lp'; sinon signe du point p' par rapport à Lp).

Figure 4.5.: Cas particulier où une des extrémités appartient au contour.

Critiques :

Dans la recherche des arêtes qui coupent la droite, le parcours des sommets du contour n'est pas nécessairement exhaustif (quand il y a intersection) et les singularités sont surmontées simplement. Cependant, nous devons parcourir le contour dans le sens des aiguilles d'une montre (orientation a priori).

4.4.4 - DECOUPAGE PAR RAPPORT A UN CONTOUR QUELCONQUE

4.4.4.1 - *Introduction*

Nous ne pouvons pas généraliser l'algorithme de SUTHERLAND pour le découpage d'un segment de droite par un polygone non convexe, parce que ce dernier peut chevaucher, de part et d'autre, les droites supports de ses arêtes.

Nous avons cherché à généraliser l'algorithme de PAVLIDIS pour des polygones non convexes (voir <HEGRON 83a>). La généralisation est possible mais complexe. En effet, nous avons montré que la mise en oeuvre du traitement des singularités est délicate et onéreuse car pour une arête (s_i,s_{i+1}), dès que le signe de l'un des sommets est nul (c'est-à-dire que $S_i=0$ ou $S_{i+1}=0$) il faut tenir compte des deux sommets précédents s_{i-2} et s_{i-1} ou du sommet précédent s_{i-1} et des deux sommets suivants s_{i+2} et s_{i+3} ou simplement du sommet suivant s_{i+2}.

Pour pallier ces difficultés, nous avons utilisé l'idée de notre algorithme de hachurage oblique sans rotation par la méthode du suivi de contour.

4.4.4.2 - *L'algorithme de "suivi de contour"*

Le problème est donc, de calculer le découpage d'un segment de droite par un polygone non-convexe (figure 4.6.).

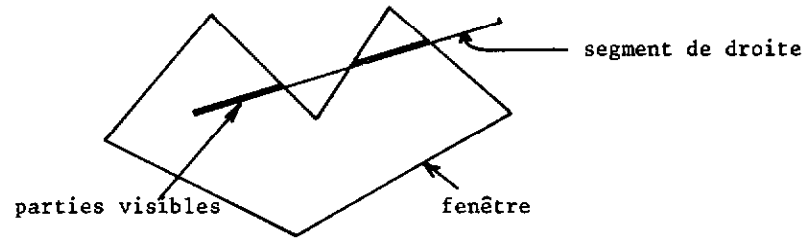


Figure 4.6.: Découpage d'un segment de droite par une fenêtre non convexe.

Le principe

Nous considérons l'équation de la droite passant par le segment. Soient a et b les accroissements respectifs en x et en y du segment.

Nous raisonnons dans le cas où |a|>=|b|. Dans l'autre cas, le raisonnement est équivalent en inversant x et y.

Soit $y=\frac{b}{a}.x+y0$ l'équation de la droite passant par le segment de coordonnées (xd,yd), (xf,yf) où $y0=yd\ \frac{b}{a}.xd$.

Chaque sommet s_i de coordonnées (x_i,y_i) est caractérisé par la valeur ys_i où $ys_i=y_i-\frac{b}{a}.x_i$.

Chaque arête (s_i, s_{i+1}) est orientée de telle sorte que $ys_i >= ys_{i+1}$ (voir figure 4.7.).

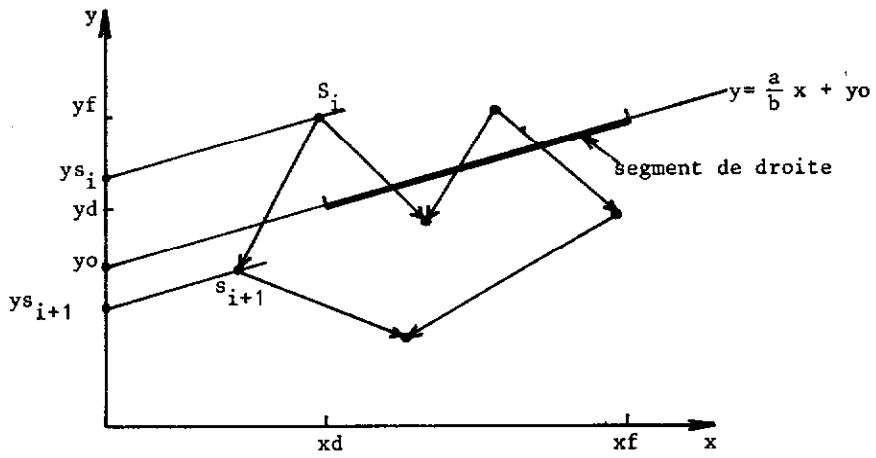
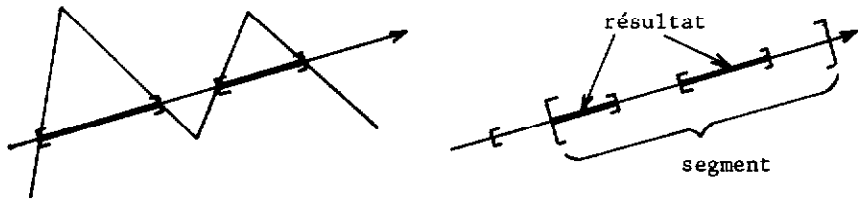


Figure 4.7.: Modélisation du polygone et du segment.

Pour déterminer l'intersection du segment avec le polygone, nous opérons de la façon suivante :

- on calcule l'ensemble des intersections du polygone avec la droite support en ordonnant cette liste, suivant l'ordre croissant des abscisses: nous obtenons ainsi, une suite de segments deux à deux, disjoints et intérieurs au polygone joignant un bord gauche à un bord droit (figure 4.8.a.).

- on calcule l'ensemble des parties du segment considéré situées à l'intérieur du polygone, en faisant l'intersection de ce segment avec la suite ordonnée des segments définis précédemment (figure 4.8.b.).



a) Liste ordonnée des segments

b) Intersection du segment à découper avec la liste ordonnée des segments.

Figure 4.8.: Intersection du segment avec le polygone.

Le traitement de la première phase se fait de la manière suivante :


```
Liste ordonnée des intersections
Début
    liste LINTER vide;
    Pour chaque arête (s_i, s_i+1) faire
    Début
        si ys_i>ys_i+1
          alors si (ys_i>=y0 et ys_i+1<y0)
                alors co l'arête coupe la droite (on ne prend pas en compte
                        les arêtes qui finissent sur la droite) fco
                     xinter:=(ys_i-y0).(x_i+1-x_i)/(y_i+1-y_i)- b/a(x_i+1-x_i));
                     yinter:= b/a.xinter+y0;
                     Ajouter le point d'intersection (xinter,yinter) à
                     LINTER en respectant l'ordre croissant des arêtes.
                fsi
          sinon si ys_i<ys_i+1 alors (même chose en inversant s_i et s_i+1);
                sinon co arête appartenant à la droite support fco
    Fin
Fin;
```

L'intersection du segment ((xd,yd),(xf,yf)) tel que xd<xf avec la liste ordonnée des segments définie par la liste des intersections se fait de la façon suivante :

```
Intersection du segment et de la liste des segments
Début
    si LINTER≠vide
      alors
        intérieur:=faux;
        obtenir première intersection (xint,yint);
        il-y-a-intersection:=vrai;
        Tant que (il-y-a-intersection et xint<xd) faire
        début
             intérieur:=¬intérieur;
             intersection suivante;
             mettre à jour il-y-a-intersection;
        fin
        si il-y-a-intersection
          alors xc:=xd;
                yc:=yd;
                tant que (il-y-a-intersection et xint<xf) faire
                début
                    xs:=xint ; ys:=yint;
                    si (intérieur et xc≠xs) alors sortir segment (xc,yc),
                       (xs,ys));intérieur:=¬intérieur;
                       xc:=xs ; yc:=ys;
                       intersection suivante;
                       mettre à jour    il-y-a-intersection;
                fsi
                si il-y-a-intersection
                  alors si (intérieur et xc≠xf) alors sortir segment
                                                   ((xc,yc),(xf,yf))fsi
                fsi
        fsi
    fsi
Fin.
```

Si le segment appartient en partie ou en totalité à une ou plusieurs arêtes du contour et si l'application le nécessite, il sera nécessaire de calculer les parties du segment strictement situées à l'intérieur du polygone en réalisant la différence des parties internes et des arêtes concernées.

Dans le cas particulier où le segment de droite est vertical, nous inversons le rôle des abscisses et des ordonnées.

Les avantages qu'offre cette méthode sont les suivants :

. comparaison avec l'ensemble des arêtes du contour non nécessairement exhaustive dans le cas convexe ;

. traitement très simple des singularités en supprimant du calcul d'intersection avec la droite support les arêtes qui lui sont parallèles et celles qui y finissent ;

. le contour peut être quelconque : contours disjoints, trous, arêtes qui se croisent.

4.4.5 - CONCLUSION

Nous avons dans ce chapitre, envisagé le découpage d'un segment de droite, en distinguant trois types de fenêtre : rectangulaire, convexe et non convexe (voir tableau récapitulatif : table 5).

Dans le cas particulier de la fenêtre rectangulaire, la méthode de SUTHERLAND (version dichotomique) est la mieux adaptée pour une réalisation câblée. Sinon, pour une réalisation programmée, l'algorithme de PAVLIDIS, demeure le plus performant, suivant les caractéristiques de la scène (pourcentage de segments verticaux et horizontaux).

Dans le cadre plus général, d'une fenêtre convexe ou non convexe, notre adaptation de la technique de suivi de contour, s'avère la plus avantageuse, que ce soit au niveau de la complexité, au niveau du traitement des singularités et de son adaptation au parallélisme.

D'autre part, nous avons exhiber, deux façons de comparer un point et un segment de droite :

- la première méthode consiste à calculer la valeur de l'équation de la droite passant par le segment en ce point ;

- l'autre possibilité utilise les coordonnées homogènes des points.

La seconde méthode permet d'éviter implicitement les divisions par zéro, lorsque la droite est verticale ou horizontale. Dans la première solution, nous devons envisager explicitement ce cas particulier.

4.5. Découpage d'un polygone par une fenêtre polygonale

4.5.1 - INTRODUCTION

Le problème est d'effectuer l'intersection entre un polygone et une fenêtre polygonale, de telle sorte que le résultat du découpage soit un contour polygonal. (figure 4.9.)

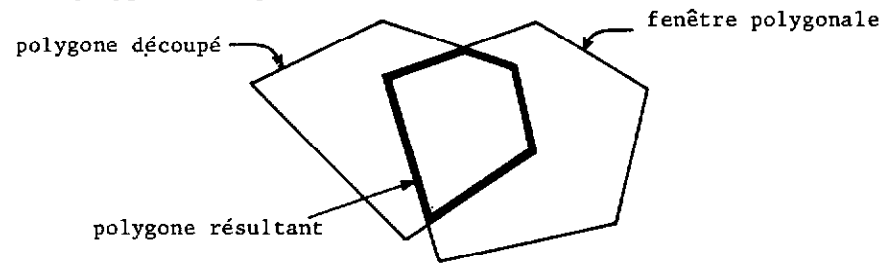


Figure 4.9.: Découpage d'un polygone par une fenêtre polygonale.

L'un des premiers algorithmes de découpage d'un polygone qui permit d'obtenir un contour, fut l'algorithme de découpage d'un polygone par une droite, proposé par SUTHERLAND et HODGMAN <SUTHERLAND-HODGMAN 74>.

Puis SHAMOS, <SHAMOS 75> développa un algorithme calculant l'intersection de deux polygones convexes, basé sur le découpage du plan en secteurs angulaires. Pour résoudre ce problème O'ROURKE <O'ROURKE 82>, proposa une nouvelle méthode qui recherche les points d'intersection entre les deux polygones en progressant pas à pas autour de chacun d'eux. La complexité de ces deux méthodes est linéaire, en fonction du nombre d'arêtes des deux contours, alors que la solution triviale, qui consiste à comparer tous les couples d'arêtes, possède une complexité proportionnelle au produit du nombre d'arêtes de chacun des contours.

4.5.2 - L'ALGORITHME DE SUTHERLAND ET HODGMAN

4.5.2.1 - *Découpage d'un polygone par une droite*

Il s'agit de déterminer la partie d'un polygone quelconque, situé dans l'un des deux demi-plans définis par une droite.

Le problème consiste donc à étudier, pour un polygone, une suite de sommets $s_1,\ldots,s_p$ pour en déduire une nouvelle suite (éventuellement vide) $s'_1,\ldots,s'_q$ de sommets tous situés dans le demi-plan à conserver. Pour ce faire, nous utilisons l'algorithme de SUTHERLAND et HODGMAN qui consiste à parcourir le polygone arête par arête en examinant la position de l'extrémité finale de chaque arête par rapport à la droite de découpage.

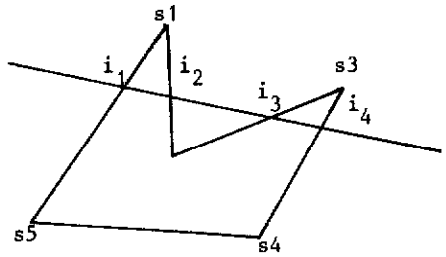
L'algorithme de traitement d'un polygone peut s'écrire :

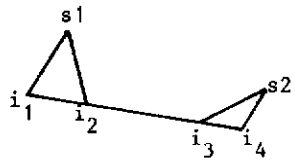
```
DECOUPE POLYGONE ;
Début (découpage d'un polygone par rapport à une droite ; les fonctions
       PREMIER SOMMET et SOMMET SUIVANT permettent de parcourir la suite
       d'arêtes du polygone ; cette suite est terminée par la valeur 0)
  entier P ; logique PREMIER;
  réel XP,YP,XS,YS,XJ,YJ,XI,YI;
  P:=PREMIER SOMMET ; PREMIER:=vrai;
  tant que P≠0 faire
  début
     XP:=X(P) ; YP:=Y(P) ; (coordonnées du sommet)
     si PREMIER
     alors début
              PREMIER:=faux;
              XJ:=XP ; YJ:=YP ; (conservation du premier sommet)
           fin
     sinon si ARETE COUPE (test par rapport à la droite)
           alors début
                    CALCUL INTERSECTION(XI,YI);
                    SORTIR(XI,YI);
                 fin
     XS:=XP ; YS:=YP ; (extrémité finale devient extrémité initiale)
     si S VISIBLE
     alors SORTIR(XS,YS);
     SOMMET SUIVANT;
  fin;
  (traitement de l'arête de fermeture)
  si SJ COUPE
  alors début
           CALCUL INTERSECTION(XI,YI);
           SORTIR(XI,YI);
        fin
fin DECOUPE POLYGONE;
```

Le découpage d'un polygone par une droite n'est pas considéré comme une simple extension de l'intersection de deux segments.

Deux types de problèmes sont à résoudre :

. Une attention particulière doit être portée aux polygones non convexes, qui peuvent produire des polygones dégénérés tels que :





liste des sommets :
$s1, i_2, i_3, i_4, i_1$

Or, nous voudrions obtenir les deux polygones définis par (s_1, i_2, i_1) et (s_2, i_4, i_3).

. Des singularités peuvent survenir :

- sommet appartenant à la droite
- colinéarité (arête appartenant à la droite : deux sommets consécutifs sont sur la droite).

Le problème des singularités est résolu en définissant de façon précise l'intersection entre un segment de droite (arête) et une droite. Soit P le demi-plan à conserver et D la droite. Soient deux sommets consécutifs s_i et s_{i+1}. Nous postulons que l'arête (s_i, s_{i+1}) coupe la droite si :

- s_i et s_{i+1} sont de part et d'autre de D avec s_i et $s_{i+1} \notin D$
- $s_i \in P$ et $s_{i+1} \in D$: il y a intersection au point s_{i+1}
- $s_i \in D$ et $s_{i+1} \in P$ et $s_{i-1} \notin P$: il y a intersection au point s_i.

4.5.2.2 - Généralisation au découpage d'un polygone par une fenêtre

Nous pouvons utiliser l'algorithme SUTHERLAND et HODGMAN, pour le découpage d'un polygone convexe ou non par une fenêtre convexe.

Dans un premier temps, nous découpons le polygone par la droite, passant par la première arête de la fenêtre : nous conservons la liste des sommets du polygone résultant, se trouvant dans le demi-plan contenant la fenêtre. Puis à partir du polygone ou des polygones résultants, nous exécutons de la même façon, le découpage avec la droite passant par la deuxième arête et ainsi de suite, jusqu'à la dernière arête de la fenêtre. Dans le cas de polygones non convexes, pour éviter la production de polygones non dégénérés, la mise au point de l'algorithme reste très délicate.

De plus, la complexité d'une telle généralisation, demeure en O(m*n) où m et n sont les nombres des arêtes des deux polygones. Dans le chapitre suivant, nous allons étudier deux algorithmes (SHAMOS et O'ROURKE) de découpage de polygones convexes, dont la complexité est en O(m+n).

4.5.3 - INTERSECTION DE DEUX POLYGONES CONVEXES

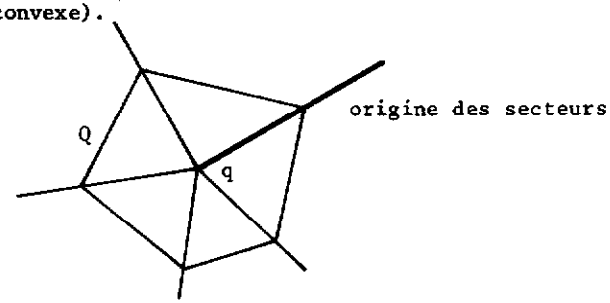
On considère deux polygones P et Q convexes ayant respectivement n et m arêtes. On veut calculer la liste des sommets du polygone $P \cap Q$.

4.5.3.1 - La méthode de SHAMOS

Les deux contours sont orientés dans un sens arbitraire (trigonométrique par exemple).

1- Le principe

A partir d'un point q intérieur au polygone Q, on le décompose en m secteurs angulaires (voir méthode de SHAMOS pour la comparaison d'un point et d'un contour convexe).

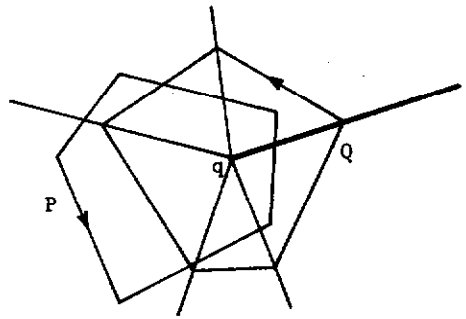


Le comportement de cette décomposition est en O(m).

Deux cas peuvent se présenter : soit le point q est à l'intérieur de P, soit il est à l'extérieur de P.

. Si le point q est à l'intérieur de P :

i) On cherche à quel secteur appartient chaque sommet de P.

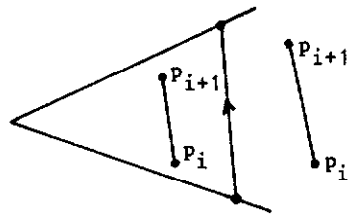


Les contours étant orientés et chaque secteur n'étant visité qu'une seule fois, cette recherche a un comportement en O(n+m).

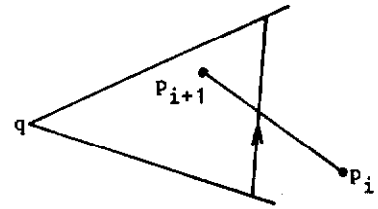
ii) On examine chaque arête (p_i,p_{i+1}) du polygone P en parcourant son contour. Différents cas de figure peuvent se présenter :

a) Les deux extrémités de l'arête (p_i,p_{i+1}) appartiennent au même secteur :

- si elles sont situées à l'intérieur ou à l'extérieur de Q il n'y a pas d'intersection.

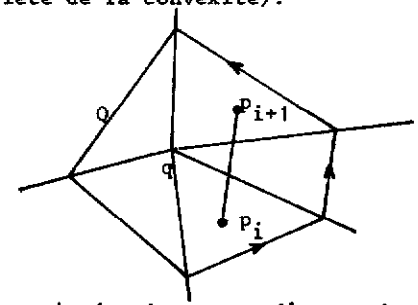


- si elles sont situées de part et d'autre du contour, il y a intersection avec l'arête du secteur correspondant.

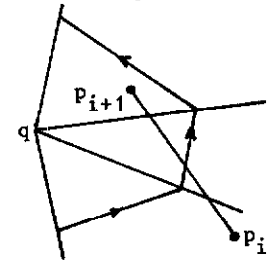


b) Les deux extrémités de l'arête appartiennent à des secteurs différents :

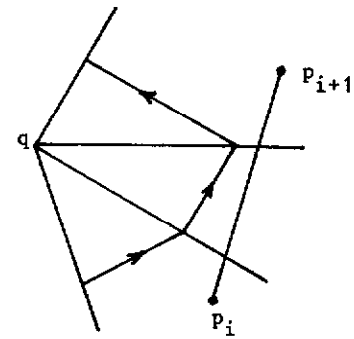
- si elles se trouvent à l'intérieur du polygone Q, il n'y a pas d'intersection (propriété de la convexité).



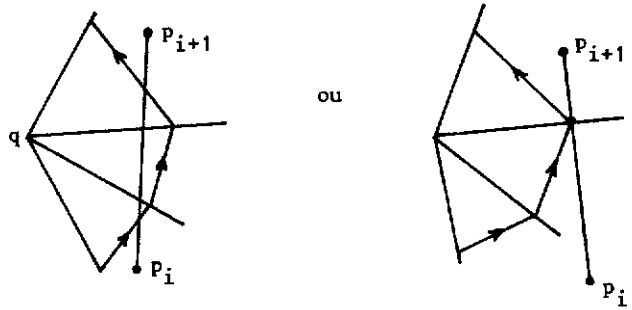
- si elles sont situées de par et d'autre du contour, il y a une intersection.



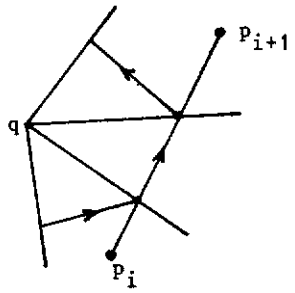
- si elles sont situées à l'extérieur du contour, il y a aucune intersection



deux intersections



ou une infinité d'intersections (colinéarité)

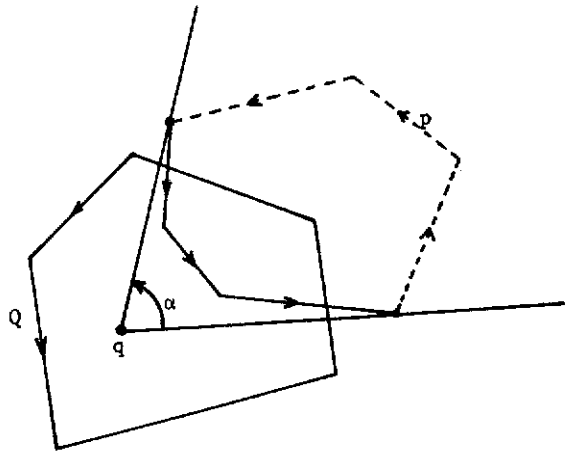


iii) Le polygone résultant de l'intersection, s'il existe, est obtenu en parcourant alternativement P et Q. Le changement se fait lors du passage sur un point d'intersection.

. Si le point q est à l'extérieur de P :

En q le contour P sous-tend un angle α. On décompose alors P en deux sous-suites d'arêtes qui seront traitées par la méthode développée ci-dessus.

Exemple :



P est décomposé en deux sous-suites (----) et (——).

2- L'algorithme

Soient deux polygones P et Q orientés dans le sens trigonométrique, définis par la liste ordonnée de leurs sommets $(sp_i)_{i=1 \text{ à } n}$ et $(sq_j)_{j=1 \text{ à } m}$ respectivement.

L'algorithme de SHAMOS peut avoir la forme générale suivante :

Algorithme de SHAMOS
Début co intersection de 2 polygones convexes P et Q fco
Calculer un point q intérieur à Q;
Décomposer Q en m vecteurs angulaires;
Si q est à l'intérieur de P
alors traitement 1;
sinon Calculer les deux sous-suites de sommets de P;
Pour chacune des deux sous-suites faire traitement 1;
fsi
Traiter P∩Q=vide;
Fin

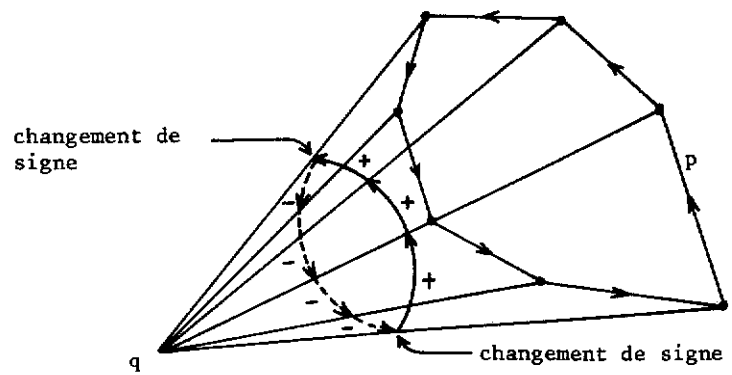
a) Nous déterminons si le point q est à l'intérieur du polygone P par la méthode des angles capables :

c'est-à-dire si $\sum_{i=1}^{n} \text{angle}\,(q\ sp_i, q\ sp_{i+1}) \cong 2\pi$

alors q ∈ P
sinon q ∉ P co somme des angles nulle fco

Pour ce faire, nous calculons d'abord les angles $(q\ sq_1, q\ sp_i)$ qui serviront ultérieurement à définir les secteurs auxquels appartiennent les sommets $(sp_i)_{i=1,\ldots,n}$ de P ((q sq_1) étant la droite de référence), puis, à partir de ces angles, nous déterminons les angles $(q\ sp_i, q\ sp_{i+1})$.

Si q n'appartient pas à P, nous devons définir les deux sous-suites de sommets. Nous obtenons le premier ou le dernier sommet de ces sous-suites quand les angles $(q\ sp_i, q\ sp_{i+1})$ changent de signe :



b) Le traitement 1 permet pour une sous-suite de sommets de P de calculer les sommets appartenant à P∩Q.

Quelques cas particuliers doivent être traités :

- sommets communs aux deux contours
- sommet d'un contour appartenant à une arête de l'autre contour
- arêtes des deux contours, colinéaires.

Ce sont des cas particuliers d'intersections que nous résolvons différemment, suivant que nous restons à l'extérieur ou à l'intérieur ou bien que nous passons de l'intérieur à l'extérieur ou de l'extérieur à l'intérieur du polygone Q.

Le problème est donc de définir quand est-ce qu'il y a intersection(s) entre une arête de P et le contour de Q.

c) Définition de l'intersection :

Soit (sp_i, sp_{i+1}) une arête de P. Pour chacune des extrémités, nous noterons respectivement :

- sect1 et sect2, les secteurs de Q associés

- int1 et int2, les variables logiques qui indiquent si les extrémités appartiennent ou n'appartiennent pas à l'intérieur de Q ;

- contour1 et contour2, les variables logiques précisant si les extrémités appartiennent ou n'appartiennent pas au contour de Q.

Nous devons distinguer le cas où les extrémités d'une arête de P sont situées dans le même secteur angulaire de Q du cas où elles appartiennent à deux secteurs distincts.

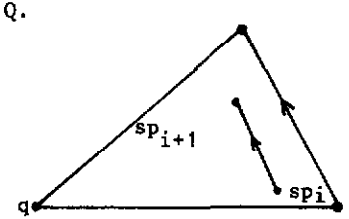
i) Les extrémités de l'arête appartiennent au même secteur.

1-Cas général (on a non (contour 1) et non (contour 2)):

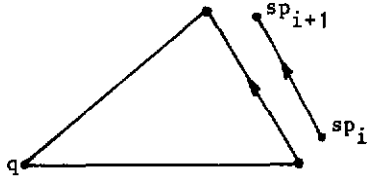
Quatre cas de figure peuvent se présenter :

. int1 et int2 (figure 4.10.a)
. non(int1) et non(int2) (figure 4.10.b)
. int1 et non(int2) (figure 4.10.c)
. non(int1) et (int2) (figure 4.10.d).

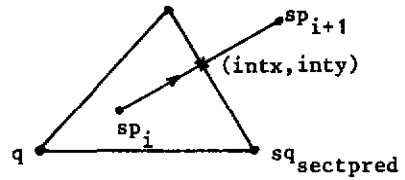
Sectpred correspond au dernier secteur dans lequel nous sommes sortis de Q.



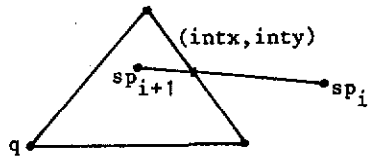
a) arête de P intérieure à Q : $sp_{i+1} \in P \cap Q$



b) arête de P extérieure à Q



c) On sort de Q :
. point d'intersection ∈ P∩Q



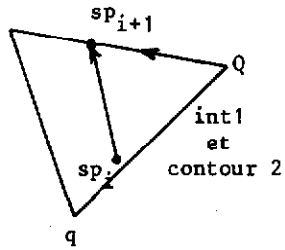
d) On entre dans Q :
. $sp_{i+1} \in P \cap Q$
. point d'intersection ∈ P∩Q.

Figure 4.10.: Cas où les extrémités d'une arête de P appartiennent au même secteur de Q.

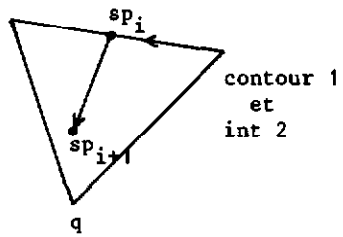
2-Cas particuliers (on a contour1 ou contour2)

Quatre cas de figure peuvent se présenter :

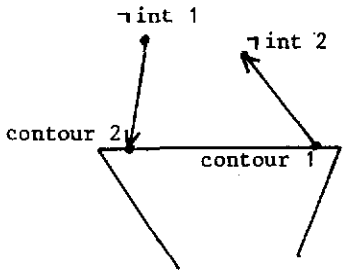
. int1 et contour2 (figure 4.11.a)
. contour1 et int2 (figure 4.11.b)
. (non(int1) et contour2), ou (contour 1 et non(int2)) (figure 4.11.c)
. contour1 et contour2 (figure 4.11.d).



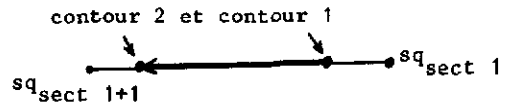
a) On sort de Q :
$sp_{i+1} \in P \cap Q$



b) On entre dans Q :
$sp_i \in P \cap Q$



c) On se considère à l'extérieur de Q (pas d'intersection).



d) Colinéarité : on se considère à l'extérieur de Q (pas d'intersection).

Figure 4.11.: Cas particuliers où les extrémités d'une arête de P appartiennent au même secteur de Q.

ii) Les extrémités de l'arête appartiennent à deux secteurs différents

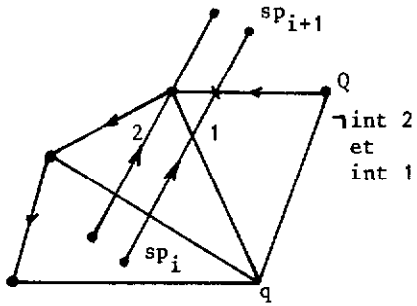
1- Cas général (on a non (contour1) et non (contour2)):

Nous pouvons rencontrer les cas de figure qui suivent.
. int1 et int2 : pas d'intersection car Q est convexe, $sp_{i+1} \notin P \cap Q$.

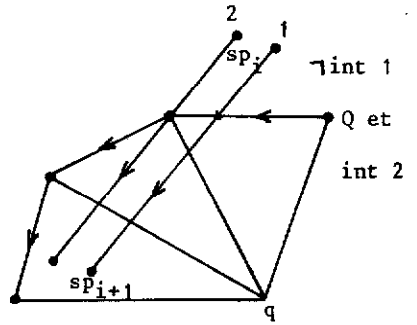
. s'il y a au moins une intersection, alors les 3 cas suivants peuvent se présenter :

- int1 et non(int2) (figure 4.12.a)
- non(int1) et int2 (figure 4.12.b)
- non(int1) et non(int2) (figures 4.12.c,d,e).

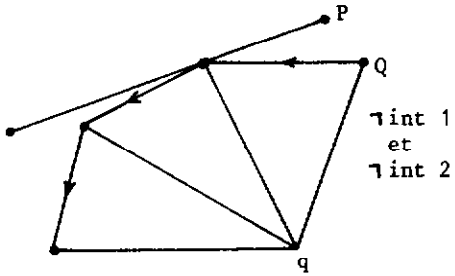
Les sommets de Q, situés à l'intérieur de P et entre deux points d'intersection consécutifs où l'on entre puis où l'on sort de Q, appartiennent à $P \cap Q$.



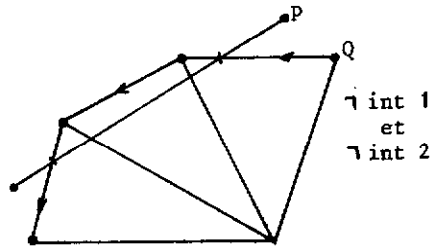
a) On sort de Q :
1-une seule intersection ($\notin P \cap Q$)
2-deux intersections identiques: on passe par un sommet de Q ($\notin P \cap Q$).



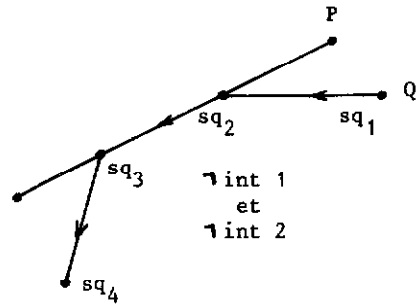
b) On rentre dans Q :
1-une seule intersection($\notin P \cap Q$)
2-deux intersections identiques : on passe par un sommet de Q ($\notin P \cap Q$).



c) extrémités extérieures à Q : on passe par un sommet de Q. (pas d'intersection)



d) extrémités extérieures à Q :
-1ère intersection : on entre dans Q,
-2ème intersection : on sort de Q (les deux points d'intersection $\notin P \cap Q$).



. il y a intersection ens q_2 avec l'arête (sq_1, sq_2), ens q_3 avec l'arête (sq_3, sq_4)
. il y a colinéarité avec l'arête (sq_2, sq_3).

e) colinéarité : on se considère à l'extérieur de Q (pas d'intersection).

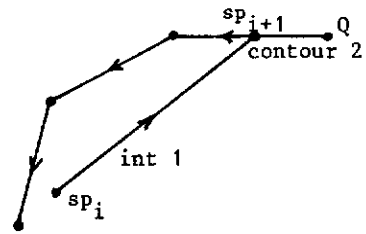
Figure 4.12.: Cas où les extrémités d'une arête de P n'appartiennent pas au même secteur.

2- Cas particulier (on a contour 1 ou contour 2)

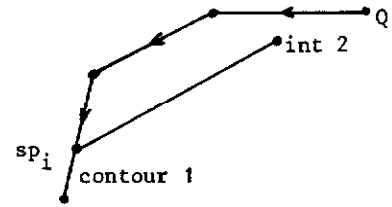
Nous pouvons rencontrer les cas de figures suivants :

. int1 et contour2 (figure 4.13.a)
. contour1 et int2 (figure 4.13.b)
. un point d'intersection et
  - non(int1) et contour2 (figure 4.13.c)
ou - contour1 et non(int2) (figure 4.13.d)

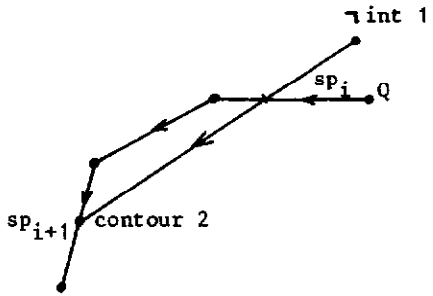
. pas de point d'intersection et
  - (non(int1) et contour2), ou (contour1 et non(int2))(figure 4.13.e)
ou - contour1 et contour2 (figure 4.13.f).



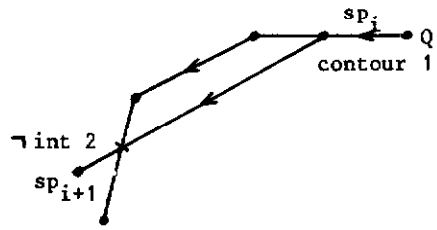
a) On sort de Q: $sp_{i+1} \notin P \cap Q$



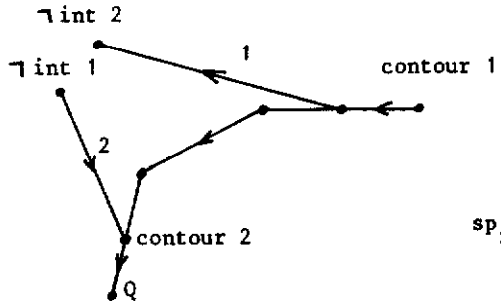
b) On entre dans Q: $sp_i \notin P \cap Q$



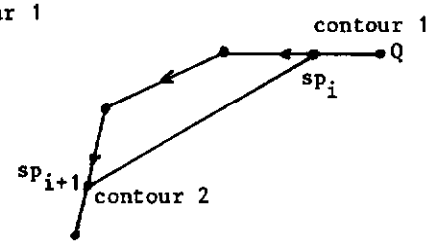
c) Une seule intersection :
. on entre au point d'intersection ($\in P \cap Q$)
. on sort en sp_{i+1} : $sp_{i+1} \in P \cap Q$.



d) Une seule intersection :
. on entre en sp_i : $sp_i \in P \cap Q$
. on sort au point d'intersection ($\in P \cap Q$)



e) On se considère à l'extérieur de Q (pas d'intersection)

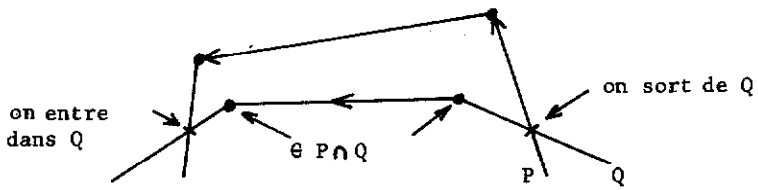


f) On entre en sp_i, on sort en sp_{i+1} :
sp_i et $sp_{i+1} \in P \cap Q$.

Figure 4.13.: Cas particuliers où les extrémités d'une arête de P n'appartiennent pas au même secteur.

Cette définition de l'intersection d'une arête du polygone P avec le contour du polygone Q, permet d'obtenir $P \cap Q$ en parcourant uniquement les arêtes de P.

Les sommets de Q appartenant à $P \cap Q$ sont situés entre deux points d'intersection consécutifs tels qu'au premier point nous sortons de Q et au second nous rentrons dans Q en parcourant le contour de P :



Pour le calcul de l'intersection de deux arêtes des polygones Q et P, nous utiliserons l'algorithme d'intersection de deux segments de droite (méthode paramétrique développée au chapitre 4.5.3.3.).

d) Si en fin de traitement $P \cap Q$ est vide, soit P et Q sont disjoints, soit $Q \subseteq P$. Cette alternative est résolue par la séquence suivante :

```
si q<P
   alors co Q ⊆ P fco
          P∩Q:=Q;
   sinon co P et Q sont disjoints
          P∩Q:=vide;
fsi
```

3- Conclusion

Le principal avantage de la méthode de SHAMOS est sa complexité en O(n+m) linéaire en fonction du nombre d'arêtes des contours.

En outre, nous avons remarqué que le nombre de sommets n du polygone P intervient dans une proportion environ quatre fois plus importante que le nombre de sommets m du polygone Q. Dans la pratique, si m>=4n environ, P sera le polygone coupant et Q le polygone coupé, sinon P deviendra le polygone coupé. L'inconvénient demeure l'utilisation des fonctions trigonométriques inverses (arctg).

4.5.3.2 - La méthode de O'ROURKE et al.

L'algorithme présenté par O'ROURKE et al. est fondamentalement différent de celui de SHAMOS.

1) Le principe

Nous utilisons les notations suivantes :

- p et q seront les sommets des polygones P et Q
- si p est le sommet courant, p_- sera le sommet précédent, et p_+ le sommet suivant ;
- $\dot{p}$ représente le segment de droite (p_-,p) ;
- $hp(\dot{p})=\{x : \dot{p} \wedge (x-p_-) >= 0\}$ représente le demi-plan qui contient le polygone P défini par la droite qui passe par p (où $\wedge$ est le produit vectoriel).

- On définit un prédicat pointer $(\dot{p},\dot{q})$ comme suit :
  pointer $(\dot{p},\dot{q})$=vrai si $p \in hp(\dot{q})$ et $\dot{q} \wedge \dot{p} < 0$
  ou
  $p \notin hp(\dot{q})$ et $\dot{q} \wedge \dot{p} >= 0$

C'est-à-dire que pointer $(\dot{p},\dot{q})$ est vrai si le vecteur $\dot{p}$ se dirige ou pointe vers la droite passant par $\dot{q}$.

Partant de deux arêtes $\dot{p}$ et $\dot{q}$ choisies arbitrairement, on progresse sur l'un ou l'autre des contours, de telle sorte que l'une des deux "pointe" vers l'autre. Ces règles sont résumées dans la table 6.

	pointer($\dot{p},\dot{q}$)	¬pointer($\dot{p},\dot{q}$)
pointer ($\dot{q},\dot{p}$)	avancer à partir de l'arête extérieure	q suivant
¬pointer($\dot{q},\dot{p}$)	p suivant	avancer à partir de l'arête extérieure

Table 6 : Règles de progression autour des contours.

A chaque pas, deux arêtes $\dot{q}$ et $\dot{p}$ sont comparées (test d'intersection) et un sommet de $P \cap Q$ est obtenu. Toutes les intersections peuvent être trouvées en parcourant au plus, deux fois chaque contour, ce qui permet à l'algorithme d'avoir un comportement en O(n+m).

2) L'algorithme

L'algorithme consiste principalement à utiliser les règles de progression, dans une structure répétitive, qui permettent d'avancer d'une arête sur l'un ou l'autre des contours à chaque pas. De plus, à chaque pas, deux arêtes sont comparées (test d'intersection) et un sommet est obtenu. Quand la boucle est terminée, tous les sommets du polygone $P \cap Q$ ont été calculés excepté lorsque $P \cap Q = \emptyset$.

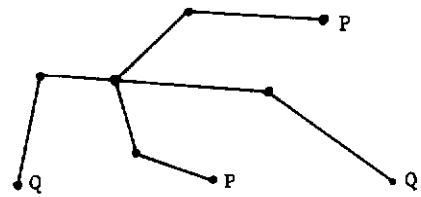
Sans prendre en compte les cas particuliers, l'algorithme a la forme générale suivante :

```
Algorithme de O'ROURKE
Début co intersection de 2 polygones convexes P et Q fco
    liste des sommets P∩Q = ∅;
    obtenir p et q artitrairement;
    répéter
       si p̊ et q̊ se coupent
         alors co traiter l'intersection fco
              si intersection = première intersection
                alors fin répéter;
                sinon P∩Q = (P∩Q) ∪ point d'intersection
                      si p ∈ hp (q̊)
                         alors intérieur := "P";
                         sinon intérieur := "Q";
                      fsi
              fsi
       fsi
    fsi
    co p suivant ou q suivant fco
    si q̊ ∧ ṗ>=0
      alors si p ∈ hp (q̊) alors q suivant sinon p suivant fsi
      sinon co q ∧ p < 0 fco
          si q ∈ hp (ṗ) alors p suivant sinon q suivant fsi
    jusqu'à boucle exécutée plus de 2(m+n) fois
    co si P∩Q = ∅ ou si P⊇Q ou P⊆Q fco
    choisir p et q arbitrairement
    si p∈Q alors P  Q:=P sinon si q∈P alors P∩Q:=Q sinon P∩Q=∅ fsi fsi
fin

p suivant
début
    si intérieur = "P" alors P∩Q:=(P∩Q)   p fsi
    p:=p+;
fin;

q suivant
début
    si intérieur="Q" alors P∩Q:=(P∩Q)  q fsi
    q:=q+
fin;
```

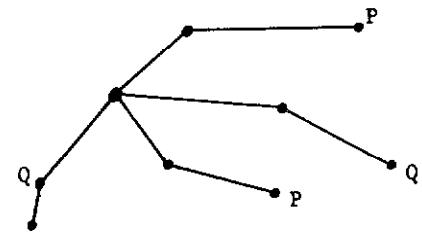

3) Singularités

Trois types particuliers d'intersections peuvent se présenter :

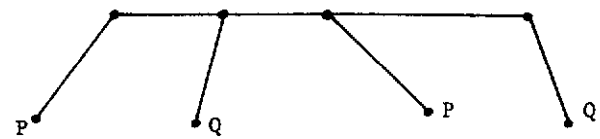
a) Un sommet de P appartient à une arête de Q :



b) P et Q ont deux sommets communs :



c) Une arête de P chevauche une arête de Q (colinéarité) :



Ces singularités sont surmontées grâce à notre définition de l'intersection qui suit.

Intersection de $\dot{p}$ et $\dot{q}$:

Soient $\dot{p}=(sp_i, sp_{i+1})$, $\dot{q}=(sq_j, sq_{j+1})$.

- si $\dot{p} \cap \dot{q} = sp_i$ ou sp_{i+1}, ou sq_j ou sq_{j+1} prenons-nous en compte cette intersection ? Si oui, nous obtenons deux points d'intersection identiques (voir singularité (a)), ou quatre points d'intersection identiques (voir singularité (b)). Pour obtenir un unique point d'intersection, dans ce cas particulier, il y aura intersection, uniquement si $\dot{p} \cap \dot{q}$ est égal à sp_{i+1} ou sq_{j+1}.

- si $\dot{p}$ et $\dot{q}$ sont colinéaires alors $\dot{p} \cap \dot{q} = \emptyset$ (singularité (c)).

4) Conclusion :

La complexité de l'algorithme de O'ROURKE et al., est proportionnelle à (n+m), au nombre d'intersections et au nombre de sommets de $P \cap Q$.

Le nombre de pas de la structure itérative est aléatoire car il dépend du choix arbitraire des deux premières arêtes.

Dans les cas particuliers où $P \subseteq Q$, $Q \subseteq P$ ou P et Q sont disjoints, le nombre de pas et de calculs d'intersection de deux segments de droite est maximal (c'est-à-dire 2(m+n)).

4.5.3.3 - *Intersection de deux segments de droite*

L'opération géométrique élémentaire commune aux algorithmes de découpage développés plus haut est le calcul de l'intersection de deux segments de droite.

Plusieurs algorithmes existent pour identifier les configurations géométriques de deux segments de droite :

- méthode analytique
- méthode vectorielle
- méthode min-max
- méthode paramétrique.

L'un d'entre eux a retenu notre attention, il s'agit de la méthode paramétrique. D'une part, ALLIAUME a montré dans <ALLIAUME 81> que cette technique s'avère l'une des plus avantageuse pour déterminer si deux segments se coupent et pour détecter les singularités (colinéarité, parallélisme,...). D'autre part, si les segments se coupent, le calcul du point d'intersection est immédiat.

La méthode paramétrique

Le principe :

Soient deux segments de droite AB et CD. Les droites support (AB) et (CD) sont définies par leurs équations paramétriques :

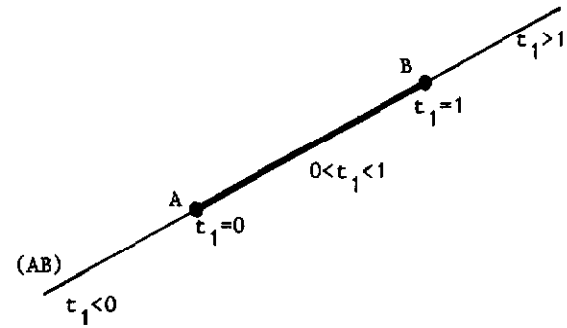
Pour (AB), on a :

$$X = XA + (XB-XA)\ t_1$$
$$Y = YA + (YB-YA)\ t_1$$

de même pour (CD), on a :

$$X = XC + (XD-XC)\ t_2$$
$$Y = YC + (YD-YC)\ t_2$$

La valeur du paramètre indique la position du point (X,Y) sur la droite :



Si les deux droites se coupent, le point d'intersection a pour coordonnées (X,Y) telles que

$$X = XA + (XB-XA)\ t_1 = XC + (XD-XC)\ t_2$$
$$Y = YA + (YB-YA)\ t_1 = YC + (YD-YC)\ t_2$$

Ce système a pour solution :

$$t_1 = \frac{(XC-XA)(YC-YD)-(XC-XD)(YC-YA)}{(XB-XA)(YC-YD)-(XC-XD)(YB-YA)}$$

$$t_2 = \frac{(XB-XA)(YC-YA)-(XC-XA)(YB-YA)}{(XB-XA)(YC-YD)-(XC-XD)(YB-YA)}$$

Les valeurs des paramètres t_1 et t_2 nous donnent tous les renseignements sur les positions relatives des deux segments de droite AB et CD.

L'algorithme :

L'algorithme qui suit traite l'ensemble des cas susceptibles d'être rencontrés (on supposera qu'aucun des segments n'est dégénéré, c'est-à-dire réduit à un point).

```
INTERSECTION de deux segments AB et CD
Début co méthode paramétrique fco
    XBA:=XB-XA;
    YBA:=YB-YA;
    XCD:=XC-XD;
    YCD:=YC-YD;
    D:=XBA.YCD-XCD.YBA;
    Si D=0
       alors segments parallèles ou colinéaires (droites supports confon-
             dues);
       sinon XCA:=XC-XA;
             YCA:=YC-YA;
             T1:=(XCA.YCD-XCD.YCA)/D;
             T2:=(XBA.YCA-YBA.XCA)/D;
             si (T1<0 ou T1>1) ou (T2<0 ou T2>1)
               alors pas d'intersection;
               sinon co intersection au point (X,Y) fco
                     X:=XA+T1.XBA;
                     Y:=YA+T1.YBA;
                     co cas particuliers co
                     si T1=0 alors intersection au point A;
                     si T1=1 alors        "            B;
                     si T2=0 alors        "            C;
                     si T2=1 alors        "            D;
             fsi
    fsi
fin
```

4.5.3.4 - Conclusion

La mise en oeuvre de l'algorithme de O'ROURKE est beaucoup plus simple que celle de SHAMOS et nécessite une structure de données minimale qui consiste en la liste des sommets des deux polygones et de leur intersection. D'autre part, dans la méthode de O'ROURKE et al., nous n'utilisons pas de fonction trigonométrique.

Cependant, dans la méthode de SHAMOS, le nombre de calculs d'intersection de deux segments de droite, sera toujours inférieur ou égal à celui dénombré dans la méthode de O'ROURKE.

4.5.4 - INTERSECTION DE DEUX POLYGONES NON CONVEXES

Contrairement au calcul de l'intersection de deux polygones convexes, dont le comportement est proportionnel à la somme du nombre d'arêtes, le traitement des contours non convexes a une complexité fonction du produit du nombre d'arêtes. Ce comportement est optimal car le nombre de points d'intersection peut être, dans ce cas, égal au produit du nombre d'arêtes.

4.5.4.1 - Une approche particulière

Une des possibilités pour accélérer la recherche des intersections, est de subdiviser les polygones en des formes plus simples. Soient R1 et R2 les deux rectangles circonscrits aux deux polygones respectifs P1 et P2 ; si R1 et R2 sont disjoints alors P1 et P2 le sont également. Si l'un des rectangles est contenu dans l'autre, par exemple $R2 \subsetneq R1$, nous pouvons alors considérer l'intersection de R2 et P1, puis l'intersection du dernier polygone ($R2 \cap P1$) avec P2. Si $R1 \cap R2$ est non vide, nous pouvons considérer l'intersection de chaque polygone P1 et P2 avec $R12 = R1 \cap R2$, ce qui définit deux nouveaux polygones, auxquels nous appliquons tout le même processus employé pour P1 et P2 de manière récursive. (voir figure 4.14.).

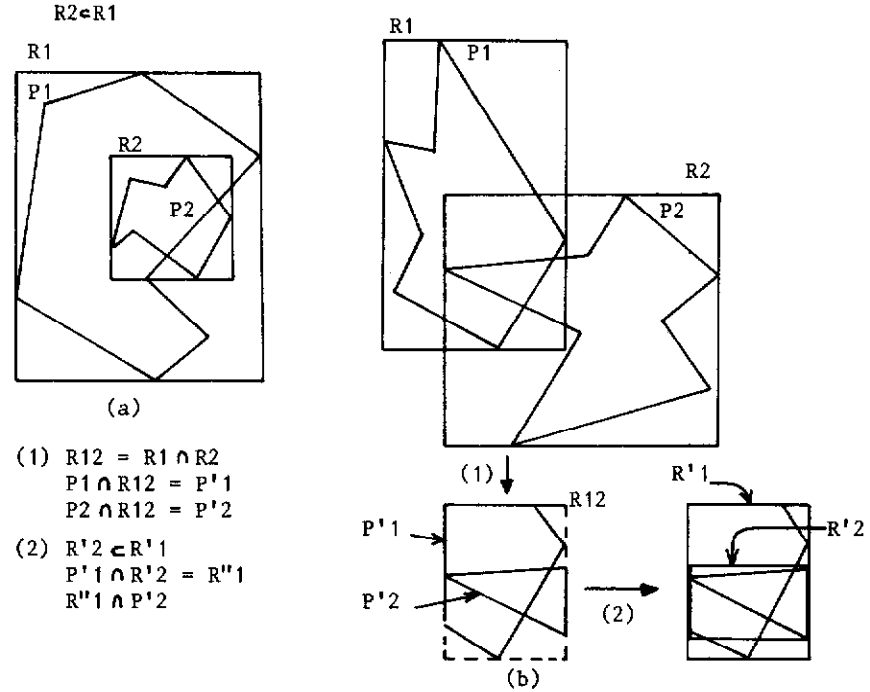


Figure 4.14.: Intersection de deux polygones non convexes par subdivisions successives.

Cependant, la réduction de la complexité n'est possible que si nous éliminons à chaque subdivision de l'un ou des deux polygones, un maximum d'arêtes. Ceci suppose également que le nombre d'arêtes des polygones soit suffisamment important (>>4). Cette approche ne résoud pas le problème de l'intersection des polygones non convexes car les contours obtenus par subdivisions successives, ne sont pas nécessairement convexes. A cet effet, quels sont les algorithmes de découpage que l'on peut employer pour les différentes opérations d'intersection ?

- Pour l'intersection d'un rectangle avec un polygone convexe ou non, nous pouvons appliquer la généralisation de l'algorithme de SUTHERLAND ET HODGMAN (voir chapitre 4.5.2.) qui réalisera ici le découpage d'un polygone

par une fenêtre rectangulaire.

- Pour l'intersection des deux polygones finaux, nous pouvons différencier trois grandes configurations :

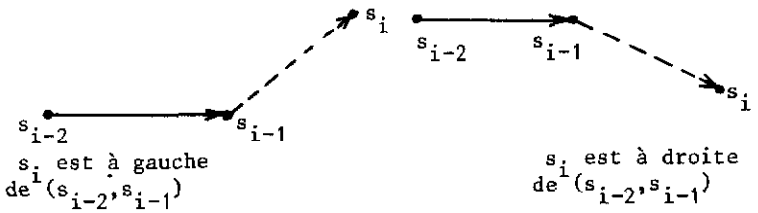
a- Les deux polygones sont convexes : on utilise soit l'algorithme de SHAMOS, soit l'algorithme de O'ROURKE.

b- L'un des deux polygones est convexe : on utilise la généralisation de la méthode de SUTHERLAND et HODGMAN.

c- Les deux polygones sont non convexes : nous devons utiliser une méthode générale comme l'algorithme de ATHERTON et WEILER.

Avant d'envisager l'étude d'une méthode générale réalisant l'intersection de deux polygones quelconques, nous devons pouvoir déterminer si un polygone est convexe ou non. Nous proposons la solution qui suit.

Soit un polygone P défini par la suite ordonnée de ses N sommets $(si)_{i=1}$ à N. Ayant choisi un sens de parcours arbitrairement, nous pouvons savoir de quel côté (à gauche ou à droite) se situe chaque sommet s_i par rapport à l'arête (s_{i-2},s_{i-1}) qui le précède :



Nous connaissons le demi-plan auquel appartient s_i en évaluant le signe de l'équation de la droite passant par (s_{i-2},s_{i-1}) au point s_i, ou le signe de det (s_i,s_{i-2},s_{i-1}) qui est le déterminant de la matrice 3x3 formée par les coordonnées homogènes des points s_i,s_{i-2},s_{i-1} (voir algorithme de PAVLIDIS, chapitre 4.4.3.2.).

Sachant que dans un polygone convexe, tous les sommets sont situés du même côté par rapport à l'arête précédente, nous détectons si un contour est non convexe en le parcourant dès que l'un des sommets change de côté (c'est-à-dire de signe (voir figure 4.15.).

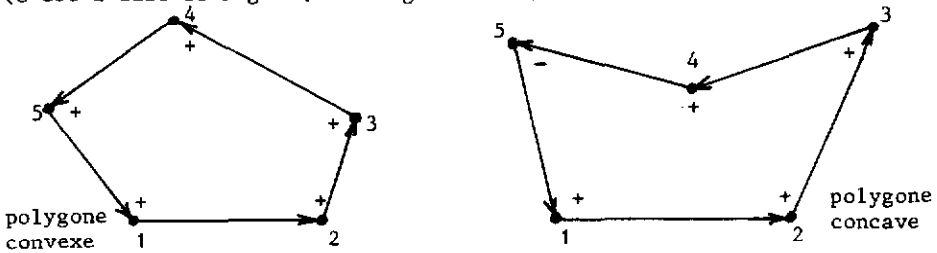


Figure 4.15.: Recherche de la convexité d'un polygone.

L'algorithme peut avoir la forme générale qui suit.

```
Déterminer si un polygone est convexe
co polygone défini par ses sommets (s_i)_{i=1,...,n} fco
début
    convexe:=vrai;
    si n>3
      alors co polygone non triangulaire fco
            calcul du signe du premier sommet s_1;
            i:=1;
            Répéter
                i:=i+1 ; co sommet suivant fco
                calcul du signe du sommet s_i;
                si signe (s_i) ≠ signe (s_1)
                  alors co changement de signe fco
                        convexité:=faux;
                fsi
            jusqu'à (¬ convexité ou i=n)
    fsi
fin
```

4.5.4.2 - L'algorithme de ATHERTON et WEILER

Le principal avantage de cet algorithme est de conserver en sortie, la même structure des données fournies en entrée, c'est-à-dire contour ou facette polygonale orientée, permettant dans un algorithme d'élimination de parties cachées, de conserver la notion de faces qui pourront être utilisées ultérieurement plus aisément par des algorithmes de remplissage ou de hachurage par exemple. Les polygones traités peuvent être quelconques, convexes, concaves et posséder des trous (contours intérieurs). Mais sa mise en oeuvre demeure délicate et sa complexité très élevée. Il s'avère fondamentalement différent des algorithmes réalisant le découpage de deux polygones convexes, comme ceux de SHAMOS ou de O'ROURKE.

Cet algorithme de découpage très complexe demeure le module de base d'un algorithme de visualisation réaliste de scènes tridimensionnelles développé par ATHERTON et WEILER (voir <ATHERTON-WEILER 77>,<WEILER 80>). Par conséquent, nous ne le considérons pas comme faisant partie de l'ensemble des traitements élémentaires.

4.5.5 - CONCLUSION

L'étude des différentes solutions qui permettent de réaliser l'intersection de deux polygones, appelée ici découpage d'un polygone par une fenêtre polygonale, a permis de dégager deux types d'algorithmes, fondamentalement différents de par la complexité de leur mise en oeuvre et de leur vitesse d'exécution. Nous distinguons les deux classes suivantes : (voir Table 7.).

1- Les algorithmes réalisant l'intersection de deux polygones convexes (SHAMOS, O'ROURKE) dont la complexité est en O(m+n) où m et n sont les nombres d'arêtes respectifs des deux contours ;

METHODE	POLYGONE/FENETRE	OPERATIONS ELEMENTAIRES	COMPORTEMENT	REMARQUES
SUTHERLAND-HODGMAN	non convexe/convexe	-intersection d'un segment de droite et d'une droite -découpage d'un polygone par une droite -comparaison point-droite	O(m*n) n: nbre d'arêtes du polygone m: nbre d'arêtes de la fenêtre	Dans certains cas, obtention de polygones dégénérés.
SHAMOS	convexe/convexe	-comparaison point-contour -calcul d'un angle(fonction trigonométrique arc-tangente) -intersection de deux segments de droite -calcul d'un point intérieur à un contour convexe	O(m+n)	-contours orientés -si n>=4 m environ, inverser les rôles polygone-fenêtre pour diminuer la complexité. -fonction trigonométrique (arctg)
O'ROURKE et al.	convexe/convexe	-comparaison point et droite -intersection de deux segments de droite	O(m+n)	Le nombre de pas et de comparaisons de deux segments de droite, dépend en partie des premières arêtes choisies pour parcourir les deux contours.
ATHERTON-WEILER	non convexe/non convexe	-intersections de segments de droite	O(m*n)	-Polygones non convexes orientés avec trous -Complexité du traitement des cas particuliers -Formats d'entrée et de sortie identiques, avec polygones extérieurs et intérieurs à la fenêtre.

Table 7 : Découpage d'un polygone par une fenêtre polygonale.

2- Les algorithmes réalisant l'intersection de deux polygones non convexes (SUTHERLAND-HODGMAN, ATHERTON-WEILER) dont la complexité est en O(m*n).

Il conviendra donc d'utiliser les informations intrinsèques à l'application traitée et aux figures géométriques manipulées afin de choisir l'algorithme qui convient :

- convexité ou non convexité des contours,
- nécessité ou non de conserver, après le découpage, la structure de données que nous avions en entrée ; c'est-à-dire contour ou facette polygonale orientée ou non.

Par exemple, dans l'algorithme d'élimination des parties cachées d'une scène tridimensionnelle d'ATHERTON et WEILER, l'utilisation de leur méthode générale de découpage de deux polygones non convexes est indispensable, puisque toute la logique et l'efficacité de leur algorithme repose sur celle-ci. Cependant, dans la simple optique du remplissage ou du hachurage de la partie visible d'un polygone dans une fenêtre, nous verrons dans les chapitres suivants que l'utilisation de la méthode générale de découpage peut être évitée.

4.6. Découpage d'une tache par une fenêtre polygonale

4.6.1 - INTRODUCTION

Le problème est d'afficher la partie visible d'une tache, soit remplie (coloriée) soit hachurée dans une fenêtre polygonale.

Nous pouvons envisager deux solutions :

1. Découpage du contour de la tache, puis remplissage ou hachurage du polygone résultant (découpage de la tache explicite) ;

2. Remplissage ou hachurage de toute la tache en n'affectant que les points ou les lignes de hachurage visibles (découpage de la tache implicite).

4.6.2 - DECOUPAGE EXPLICITE

Dans un premier temps, nous découpons le contour polygonal de la tache, par l'un des algorithmes de découpage étudié au paragraphe précédent. Puis, une fois le contour intérieur à la fenêtre obtenu, nous pouvons dans un deuxième temps, appliquer un algorithme de remplissage ou de hachurage de tache sur le polygone résultant.

La complexité du découpage explicite d'une tache, est égale à la somme des complexités du découpage par la fenêtre et du remplissage du contour produit.

4.6.3 - DECOUPAGE IMPLICITE

Deux méthodes peuvent être utilisées :

1. La première solution consiste à remplir ou hachurer la tache initiale, en calculant pour chaque point, ou pour chaque ligne de hachurage, leur appartenance à la fenêtre, à l'aide d'un algorithme de comparaison point-contour, ou respectivement du découpage d'un segment par un polygone.

2. La seconde solution consiste à remplir ou hachurer la tache et la fenêtre simultanément en n'affichant que les points ou les segments de droite qui sont situés à l'intérieur des deux contours.

Si la fenêtre est rectangulaire, pour le remplissage et le hachurage vertical ou horizontal, les deux solutions sont équivalentes. Sinon pour le remplissage (respectivement pour le hachurage), plus la surface ou le nombre de lignes de balayage (respectivement plus les lignes de hachurage) de la tache découpée sont importants, plus l'utilisation de la seconde solution devient avantageuse.

L'inconvénient majeur du découpage implicite est le temps passé à traiter la partie invisible de la tache dans la mesure où celle-ci est importante.

4.6.4 - CONCLUSION

Dans le cas de taches et de fenêtres convexes, la méthode explicite est la plus avantageuse. Mais dans le cas non convexe, l'utilisation du découpage implicite devient plus favorable car la complexité du découpage des contours est en O(n*m).

Nous pouvons cependant formuler deux remarques :

. Si nous traitons le découpage d'un ensemble de taches par une fenêtre, la complexité de la méthode implicite peut être considérablement améliorée en effectuant le remplissage ou le hachurage de la fenêtre une fois pour toutes. Seul l'espace mémoire nécessaire augmente dans ce cas.

. La conservation du polygone résultant dans la méthode explicite est par contre très intéressante, si nous voulons modifier la couleur ou le hachurage de la partie visible de la tache. Mais dans le chapitre suivant, nous verrons que les contours résultant du découpage peuvent être extraits et mémorisés lors du remplissage implicite de la partie visible d'une tache dans une fenêtre.

4.7. Découpage d'une tache par une autre

4.7.1 - INTRODUCTION

Par opposition au découpage d'une tache par une fenêtre, nous définissons le découpage d'une tache P par une tache Q comme l'opération qui calcule non seulement l'intersection ($P \cap Q$) mais encore l'union ($P \cup Q$) et les différences (P-Q) et (Q-P) (voir figure 4.15.).

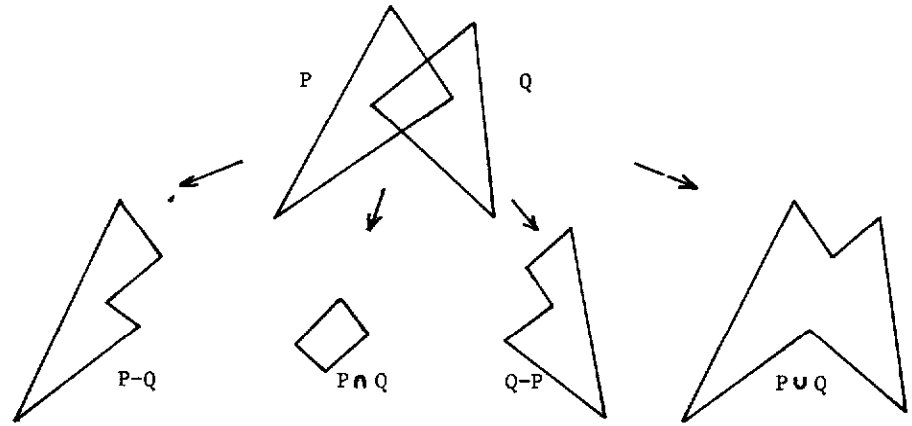


Figure 4.15.: Opérations de découpage entre deux taches P et Q.

Si P et Q sont convexes, nous pouvons réaliser de manière explicite uniquement les opérations d'intersection et d'union sur ces deux taches en utilisant l'algorithme de O'ROURKE. D'autre part, les algorithmes de SUTHERLAND-HODGMAN ou de ATHERTON-WEILER, peuvent être mis en oeuvre pour produire tous les polygones résultant (intérieurs et extérieurs) du découpage de deux polygones non convexes. Mais ces deux méthodes ne sont plus du type des techniques rencontrées dans le cas convexe et leurs complexités demeurent en O(m*n).

Nous proposons dans ce qui suit, de réaliser les opérations d'union, d'intersection et de soustraction de deux polygones P et Q à partir de ces mêmes opérations effectuées sur les deux taches qu'ils définissent : c'est-à-dire à partir du découpage implicite des deux taches.

Deux approches peuvent être envisagées : soit nous définissons la tache par l'ensemble des points qui la constituent (nous manipulons alors des matrices de points), soit nous regardons la tache comme un ensemble de segments horizontaux ou intervalles qui sont les parties des lignes de balayage intérieures à la tache (nous manipulons dans ce cas, une structure d'arêtes et d'intervalles).

4.7.2 - TACHE DEFINIE PAR UN ENSEMBLE DE POINTS

Soient deux taches A et B.

Après avoir rempli les taches A et B et mémorisé les ensembles des points qui les constituent dans des mémoires d'images (matrices de points) respectives, nous pouvons très facilement extraire les sous-ensembles désirés et les mémoriser si nécessaire.

Les opérations élémentaires sont très simples et limitées : pour chaque point, nous effectuons au plus deux tests et deux accès tableau. Mais en contre-partie, la mémoire utile demeure trop importante et il faut parcourir toute la matrice image.

Nous pouvons envisager une solution intermédiaire où nous n'utilisons qu'une seule mémoire de points. Celle-ci nous permet d'y inscrire pour chaque tache, l'ensemble des points qui la compose, puis d'y extraire ensuite la liste des intervalles de points ligne par ligne (codage par plages). (voir figure 4.16.)

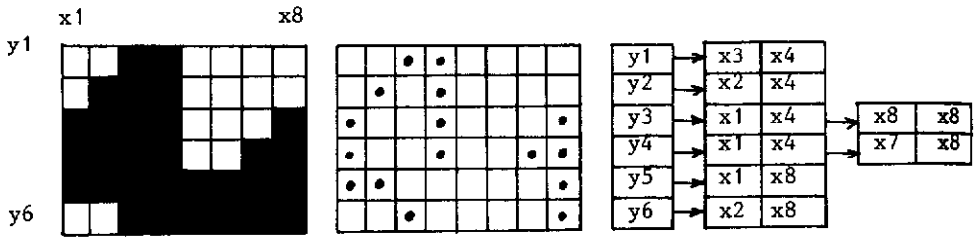


Figure 4.16.: Constitution de la liste des intervalles de points.

Les opérations ensemblistes sur les deux taches s'effectuent en comparant les deux listes d'intervalles, ligne par ligne. Chaque sous-ensemble désiré sera aussi défini par une liste d'intervalles. Pour l'union des taches, nous calculerons pour chaque ligne, l'union des intervalles, pour l'intersection, l'intersection des intervalles et pour la différence, la différence des intervalles. Si les opérations réalisées sont plus complexes que celles effectuées sur les matrices de points, le gain de mémoire utile est considérable.

Nous allons développer maintenant les différents algorithmes d'union, d'intersection et de différence de deux listes d'intervalles de points LISTE 1 et LISTE 2. Le résultat sera également une liste d'intervalles de points appelée LISTE 3.

Union : LISTE 3 = LISTE 1 ∪ LISTE 2.

Nous réalisons dans un premier temps, la fusion des deux listes ordonnées LISTE 1 et LISTE 2. Dans la nouvelle liste, chaque élément est marqué par le nom de la liste initiale correspondante. Sachant qu'un point appartient à l'union des taches, s'il appartient à l'une ou à l'autre, nous pouvons alors extraire la liste correspondante des intervalles.

Soient A et B, les taches correspondant à LISTE 1 et LISTE 2 respectivement, et int-A et int-B, deux variables logiques indiquant si nous sommes à l'intérieur des taches A et B.

Nous obtenons l'algorithme qui suit.

```
Union (données LISTE 1, LISTE 2 ; résultat LISTE 3)
Début
    Fusion(donnée LISTE 1, LISTE 2 ; résultat LISTE 4)
    co extraire l'union de LISTE 4 fco
    int-A:=faux;
    int-B:=faux;
    créer LISTE 3;
    ILYAELEMENT:=vrai ; co LISTE 4 non vide fco
    Tant que ILYAELEMENT faire
    début
       obtenir élément LISTE 4;
       si courant(LISTE 4) ∈ A
          alors int-A:=¬int-A;
          sinon int-B:=¬int-B;
       fsi
       écrire courant(LISTE 4) dans LISTE 3;
       Tant que (int A ou int B) faire
       début
            obtenir élément LISTE 4;
            si courant(LISTE 4) ∈ A
               alors int-A:=¬int-A;
               sinon int-B:=¬int-B;
            fsi
       fin
       écrire courant(LISTE 4) dans LISTE 3;
       mettre à jour ILYAELEMENT;
    fin
fin Union
```

Intersection[1] : LISTE 3 = LISTE 1 ∩ LISTE 2

Un point p ∈ A∩B si p ∈ A et p ∈ B.

En utilisant les mêmes notations que précédemment, on obtient alors l'algorithme suivant :

(1) Nous trouvons un exemple de la mise en oeuvre de cet algorithme dans le programme Texte-polygone (Ecriture d'un texte dans une tache polygonale quelconque) au Chapitre 3.3.3.5., pages 105-106.

```
Intersection (donnée LISTE 1, LISTE 2 ; résultat LISTE 3)
Début
    Fusion(donnée LISTE 1, LISTE 2 ; résultat LISTE 4);
    co extraire l'intersection de LISTE 4 fco
    int_A:=faux;
    int_B:=faux;
    créer LISTE 3;
    ILYAELEMENT:=vrai ; co LISTE 4 non vide fco
    Tant que ILYAELEMENT faire
    début
       Tant què (¬ int A ou ¬ int B) et ILYAELEMENT faire
       Début
            obtenir élément LISTE 4;
            mettre à jour int_A et int_B;
            mettre à jour ILYAELEMENT;
       fin
       si (int A et int B)
         alors écrire courant (LISTE 4) dans LISTE 3;
               obtenir élément LISTE 4;
               mettre à jour int_A et int_B;
               co on a ¬(int_A et int_B) fco
               écrire courant(LISTE 4) dans LISTE 3;
       fsi
       mettre à jour ILYAELEMENT;
    fin
Fin Intersection

Mettre à jour int_A et int_B
début
    si courant (LISTE 4) ∈ A;
      alors int_A:=¬int_A;
      sinon int_B:=¬int_B;
    fsi
fin;
```

DIFFERENCES :

1- LISTE 3=LISTE 1-LISTE 2(c'est-à-dire A-B)
Un point p ∈ A-B si p∈A et p∉B
On obtient l'algorithme qui suit.


```
Différence A-B(donnée LISTE 1, LISTE 2 ; résultat LISTE 3)
Début
    Fusion(donnée LISTE 1, LISTE 2 ; résultat LISTE 4);
    co extraire A-B de LISTE 4 fco
    int_A:=faux;
    int_B:=faux;
    créer LISTE 3;
    ILYAELEMENT:=vrai ; co LISTE 4 non vide fco
    Tant que ILYAELEMENT faire
    début
       Tant que (¬ int A ou int B) et ILYAELEMENT faire
       début
            obtenir élément LISTE 4;
            mettre à jour int_A et int_B;
            mettre à jour ILYAELEMENT;
       fin
       si (int_A et ¬ int_B)
         alors écrire courant (LISTE 4) dans LISTE 3;
               obtenir élément LISTE 4;
               mettre à jour int A et int B;
               co on a ¬(int_A et ¬ int_B) fco
               écrire courant (LISTE 4) dans LISTE 3;
       fsi
       mettre à jour ILYAELEMENT;
    fin
fin différence A-B.
```

2- LISTE=LISTE 2-LISTE 1 (c'est-à-dire B-A)

Un point p∈B-A si p∉A et p∈B.
On obtient l'algorithme "différence B-A" qui est identique à "Différence A-B" dans lequel on remplace la condition (¬int_A ou int_B) par (int_A ou ¬ int_B) et la condition (int_A et ¬ int_B) par (¬ int_A et int B).

4.7.3 - TACHE DEFINIE PAR UN ENSEMBLE D'INTERVALLES

4.7.3.1 - Introduction

Au lieu de construire la liste des intervalles en passant par le remplissage point à point de la tache dans une matrice de points, nous pouvons directement l'obtenir par un algorithme de SUIVI DE CONTOUR (voir chapitre 3.3.3.2).

Les différents traitements opérés sur les taches sont identiques à ceux exposés ci-dessus. Chaque nouvelle tache résultant de l'union, de l'intersection ou de la différence des deux premières, est caractérisée par la liste des intervalles qui la composent, ligne par ligne. A partir de cette structure de données, les opérations comme le remplissage de tache ou la translation sont triviales. Mais des transformations géométriques telles que la rotation, le changement d'échelle, nécessitent la description du contour de la tache, sous forme d'une liste d'arêtes. Cette description permettra d'autre part, d'économiser de la mémoire.

Nous présentons un algorithme, qui permet de réaliser de manière implicite (voir chapitre 4.7.3.2) les diverses opérations ensemblistes, entre deux taches polygonales, en calculant pour chacun des sous-ensembles obtenus, la liste des arêtes correspondantes. En outre, cet algorithme permet d'exécuter simultanément le remplissage ou le hachurage horizontal du sous-ensemble désiré.

4.7.3.2 - *Algorithme de découpage de deux taches*

Soient A et B, deux taches polygonales, définies par la liste ordonnée des coordonnées des sommets de leurs contours respectifs, dans le système écran.

Le problème : nous désirons établir les listes des arêtes, qui constituent les contours des taches polygonales obtenues à partir de A et B : c'est-à-dire $A \cap B$, $A \cup B$, A-B et B-A.

1) L'idée :

En utilisant la technique du suivi de contour, nous désirons calculer les arêtes q engendrées par A et B en les attribuant aux sous-ensembles correspondant, par un découpage implicite de A et B (voir figure 4.17).

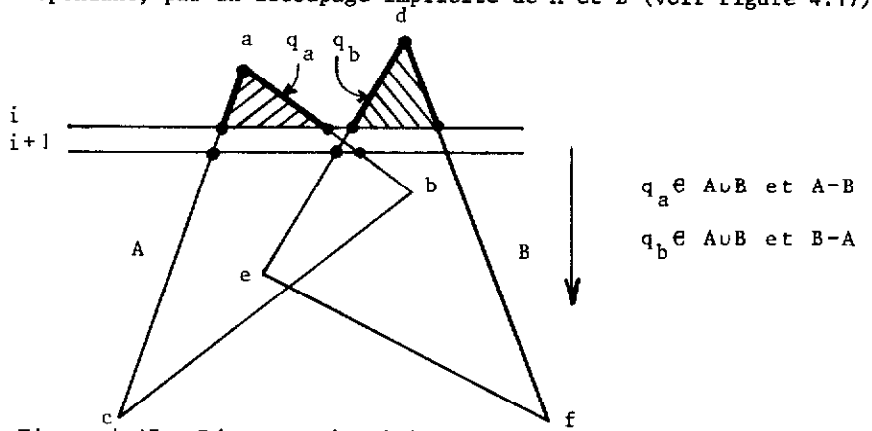


Figure 4.17.: Découpage implicite par suivi de contour.

2) L'algorithme :

Comme pour le remplissage simultané d'un ensemble de taches polygonales, nous constituons ligne par ligne la liste ordonnée des arêtes de A et de B qui la coupent.

Le calcul des arêtes q engendrées par A et B se fait de la manière suivante :

- nous détectons une arête q lorsqu'une arête change de contexte entre deux lignes de balayage consécutives. Ceci peut se produire lorsque deux arêtes se coupent ou lorsque des arêtes se terminent ou commencent. Pour ce faire, nous devons mémoriser les coordonnées du dernier changement de contexte de l'arête et son contexte sur la ligne précédente, à savoir si elle est à l'intérieur ou à l'extérieur de l'autre polygone correspondant;

- nous attribuons l'arête q aux sous-ensembles correspondants grâce à son contexte ; (voir Table 8).

- nous rencontrons une singularité si deux arêtes sont confondues (mêmes points d'intersection avec les lignes de balayage et mêmes pentes) (voir figure 4.18.). L'attribution de l'arête se fait à l'aide de son contexte par rapport aux deux polygones A et B (voir Table 9).

$q \in A$	$q \subset \overset{\circ}{B}$	$q \in A \cap B$ et B-A
	$q \notin \overset{\circ}{B}$	$q \in A \cup B$ et A-B
$q \in B$	$q \subset \overset{\circ}{A}$	$q \in A \cap B$ et A-B
	$q \notin \overset{\circ}{A}$	$q \in A \cup B$ et B-A

$q \in A$ (ou B): q appartient au contour de A (ou de B) ;
$q \in \overset{\circ}{A}$ (ou $\overset{\circ}{B}$): q appartient à l'intérieur de A (ou de B) ; c'est-à-dire à (A-(contour A)) (ou à (B-(contour B))).

Table 8

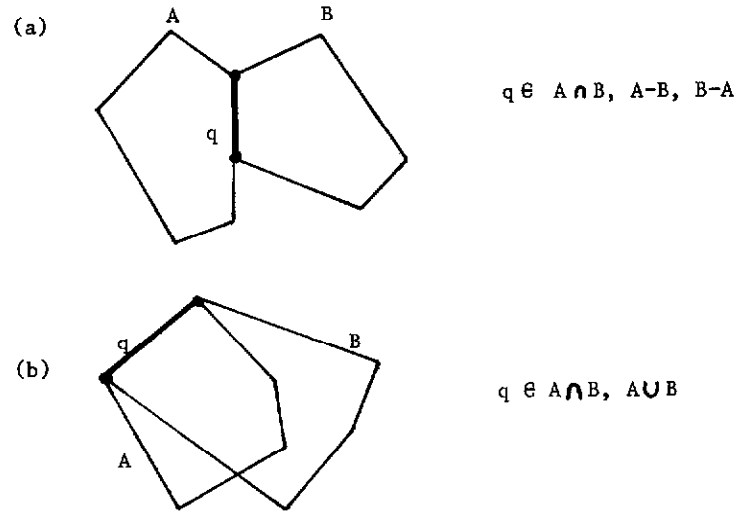


Figure 4.18.: Cas particulier où $q \in A$ et $q \in B$.

q∈A	q bord droit de A et bord gauche de B	q∈A∩B* et A-B
	q bord gauche de A et bord droit de B	q∈A∩B* et B-A
	q bord droit de A et de B ou q bord gauche de A et de B	q∈A∩B et A∪B
q∈B	q bord droit de A et bord gauche de B	q∈A∩B* et B-A
	q bord gauche de A et bord droit de B	q∈A∩B* et A-B
	q bord droit de A et de B ou q bord gauche de A et de B	q∈A∩B et A∪B

* cas dégénérés (ne pas affecter q à A∩B)

Table 9

3) Conclusion

Cette méthode présente les avantages suivants :

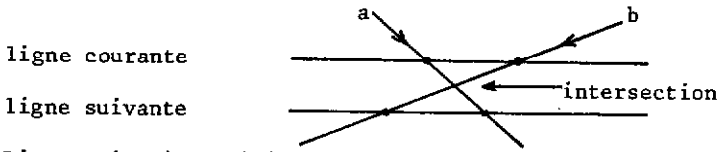
. Le découpage et le remplissage du ou des sous-ensembles désirés peuvent être exécutés simultanément. Nous remplissons A∩B, A∪B, B-A ou A-B en réalisant respectivement l'union, l'intersection ou les différences des listes d'intervalles horizontaux intérieurs aux taches A et B sur chaque ligne de balayage. Ce résultat demeure intéressant pour la manipulation d'images ;

. Les structures de données obtenues en sortie sont compatibles avec des traitements ultérieurs réalisés par la méthode du suivi de contour (remplissage, hachurage horizontal, comparaison point-contour quelconque) parce que nous avons un ensemble de bords gauches et un ensemble de bords droits.

Nous relevons cependant deux inconvénients :

. Il faudra tenir compte des arêtes horizontales pour le dessin au trait ou certains traitements ultérieurs sur les sous-ensembles obtenus tels que le hachurage vertical ou oblique et la rotation ;

. L'intersection de deux arêtes de A et B se produit rarement sur une ligne de balayage, mais le plus souvent entre deux lignes :



L'approximation utilisée engendre une erreur plus ou moins importante en fonction de la pente des arêtes concernées.

4.8. Généralisation aux taches non-polygonales

Nous supposons que les contours de la tache non polygonale sont formés de segments de droite et d'arcs de cercle .

4.8.1 - COMPARAISON POINT-CONTOUR NON POLYGONAL

Le problème est de savoir si un point appartient à l'intérieur d'une tache définie par un contour non polygonal et non convexe.

Parmi les algorithmes que nous avons recensés pour la comparaison d'un point et d'un contour polygonal, seule la méthode du suivi de contour peut être généralisée. Le principe est donc de résoudre le contrôle de parité pour le point à tester, en utilisant le raisonnement du remplissage de tache non polygonale par la méthode du suivi de contour (voir chapitre 3.3.3.2.).

Le problème élémentaire à résoudre est le calcul de l'intersection de la demi-droite horizontale située à gauche du point avec les primitives du contour : si la primitive est un segment, nous opérons comme dans l'algorithme du chapitre 4.3.3.2. Si la primitive est un arc de cercle, et si le point n'est pas situé au-dessus ou au-dessous du cercle entier, nous le décomposons en un ensemble de bords gauches et droits par quadrant (voir chapitre 3.3.3.4.). Puis pour chaque morceau d'arc de cercle obtenu, nous effectuons le traitement suivant :

Le calcul de l'intersection du morceau d'arc de cercle avec la demi-droite se fait de la façon qui suit.

Soient (Xd,Yd),(Xf,Yf) les coordonnées des extrémités début et fin du morceau d'arc, et D son sens de rotation (D:=si sens trigonométrique alors 1 sinon 0). Soient R le rayon et (Xc,Yc) le centre du cercle associé, et (Xp,Yp) les coordonnées du point à tester.

Si $Yp>Yd$ *ou* $Yp\leq Yf$ *ou* $Xp<min\ (Xd,Xf)$
alors pas d'intersection;
sinon intersection avec l'horizontale $Y=Yp$ au point d'abscisse Xi:
$Xi:=Xc+(R^2-(Yp-Yc)^2)^{1/2}$ *si* $D=0$;
$Xi:=Xc-(R^2-(Yp-Yc)^2)^{1/2}$ *si* $D=1$;
si $Xp=Xi$ *alors le point appartient au contour;*
si $Xp<Xi$ *alors* pas d'intersection avec la demi-droite;

La complexité est proportionnelle au nombre de primitives.

Nous pouvons cependant tirer parti de la convexité si le point est extérieur au contour : on s'arrête dès que l'on a détecté deux points d'intersection.

4.8.2 - DECOUPAGE D'UN SEGMENT PAR UN CONTOUR NON POLYGONAL

Le problème est de calculer la partie du segment de droite situé à l'intérieur du contour non-polygonal.

Nous proposons de généraliser la méthode du suivi de contour employée pour le découpage (voir chapitre 4.4.4.(2)) d'un segment par un contour polygonal.

Le problème élémentaire à résoudre est le calcul de l'intersection de la droite support avec les primitives. Si la primitive est un segment de droite, nous effectuons le même traitement qu'au chapitre 4.4.4.2. Si la primitive est un arc de cercle, nous pouvons opérer de la manière suivante :

Soient $y=mx+p$ l'équation de la droite support :
R le rayon et $C=(Xc,Yc)$ le centre du cercle ;
$Ed=(Xd,Yd)$ et $Ef=(Xf,Yf)$ les extrémités début et fin, et D le sens de parcours de l'arc de cercle.

Les points d'intersection de la droite support avec le cercle ont pour coordonnées :

$$X=Xc+\frac{-pm \pm \sqrt{\Delta}}{1+m^2} \qquad Y=mX+p$$

où $\Delta=p^2m^2-(1+m^2)(p^2-R^2)$

Nous pouvons rencontrer trois cas :

. $\Delta<0$, la droite ne coupe pas le cercle : pas d'intersection.

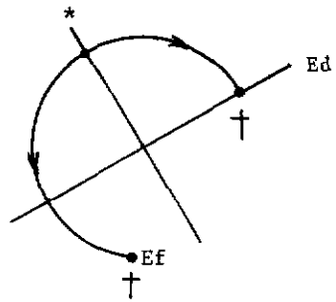
. $\Delta>0$, la droite coupe le cercle en deux points distincts $(I_k=(x_k,y_k))_{k=1,2}$.

On a $I_k \in$ arc(Ed,Ef) si $\widehat{(EdCI_k)}<\widehat{(EdCEf)}$, où l'orientation des angles respecte le sens de parcours D de l'arc de cercle.

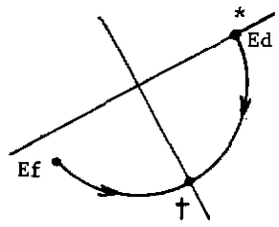
Si l'un ou les deux points d'intersection appartiennent à l'arc de cercle deux cas peuvent se présenter :

1- Le(s) point(s) d'intersection diffère(nt) des extrémités de l'arc : la droite support coupe l'arc de cercle en ce(s) point(s).

2- L'un ou les deux points d'intersection sont confondus avec les extrémités de l'arc : comme pour les segments de droite, on ne prend pas en compte l'extrémité où les sous-parois finissent :



pas d'intersection en Ed
(point de mort :†)



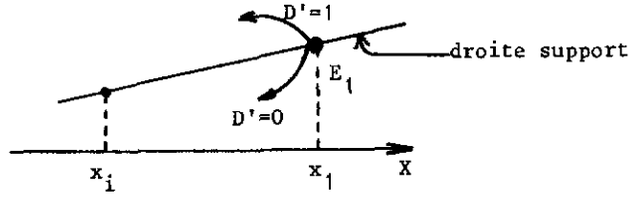
Intersection en Ed
(point de naissance :*)

Deux cas de figures se présentent :

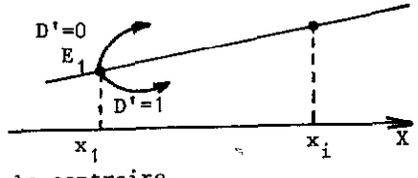
a) La droite passe par une seule des extrémités E_1

Soient $E_1=(x_1,y_1)$, $E_2=(x_2,y_2)$ l'autre extrémité et D' le sens de parcours de l'arc (E_1,E_2).

- la droite ne coupe l'arc qu'en E_1 :
  . $y_1-mx_1<y_2-mx_2$: pas d'intersection en E_1 (point de mort)
  . $y_1-mx_1>y_2-mx_2$: intersection en E_1 (point de naissance)
- la droite coupe l'arc en E_1 et au point $I=(x_i,y_i)$, avec $E_1=E_d$:
  . $x_1>x_i$: D'=0, intersection en E_1 (point de naissance)
  D'=1, pas d'intersection en E_1



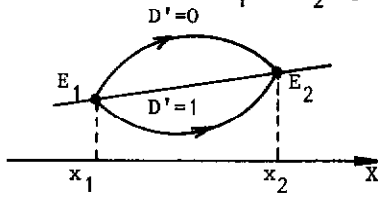
. $x_1<x_i$: D'=1, intersection en E_1 (point de naissance)
D'=0, pas d'intersection en E_1



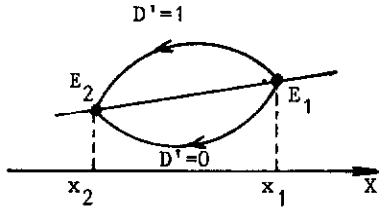
Si $E_1=E_f$, c'est le contraire.

b) La droite passe par les deux extrémités E_1 et E_2 avec $E_1=E_d$ (1) :

. $x_1<x_2$: D'=0, pas d'intersection en E_1 et E_2 (points de mort)
D'=1, intersection en E_1 et E_2 (points de naissance)



. $x_1>x_2$: D'=0, intersection en E_1 et E_2 (points de naissance)
D'=1, pas d'intersection en E_1 et E_2 (points de mort)



. $\Delta=0$, la droite est tangente au cercle au point T. Deux cas peuvent se présenter :

1- le point T diffère des extrémités de l'arc de cercle : il n'y a pas d'intersection.

2- Le point T est confondu avec l'une des extrémités de l'arc : soient $E_1=(x_1,y_1)$ cette extrémité et $E_2=(x_2,y_2)$ la seconde.

. $y_1-mx_1<y_2-mx_2$: pas d'intersection au point T (point de mort)

. $y_1-mx_1>y_2-mx_2$: intersection au point T (point de naissance).

L'ensemble des intersections de la droite support définit sur celle-ci un ensemble de segments de droite intérieurs à la tache et joignant ses bords. La partie visible du segment à découper sera calculée en effectuant l'intersection de celui-ci avec l'ensemble des segments intérieurs au contour non polygonal (voir chapitre 4.4.4.2.).

Si le segment de droite est vertical, on inversera le rôle des X et des Y.

La complexité de l'algorithme est proportionnelle au nombre de primitives. Nous pouvons également tirer parti de la convexité en arrêtant la comparaison de la droite support avec les primitives du contour dès que l'on a trouvé deux points d'intersection.

L'inconvénient de cette méthode est l'utilisation de la fonction trigonométrique inverse arctg pour le calcul des angles.

(1) Si $E_1=E_f$ c'est le contraire.

4.8.3 - DECOUPAGE D'UN ARC DE CERCLE PAR UN CONTOUR NON POLYGONAL

Le problème est de calculer la partie visible d'un arc de cercle situé à l'intérieur du contour non polygonal.

L'algorithme se décomposera comme précédemment en deux temps :

1. Calcul des points d'intersection du contour avec le cercle supportant l'arc.

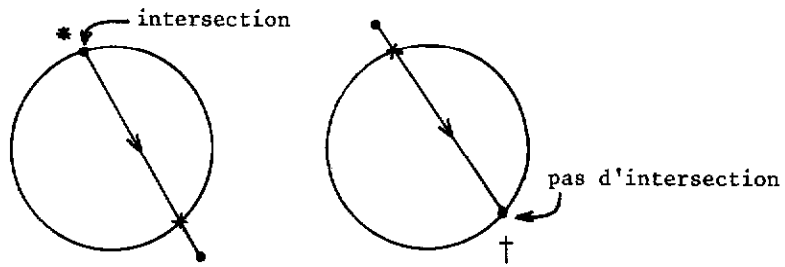
2. Extraction des parties visibles.

1- Calcul des points d'intersection

- intersection du cercle avec un segment de droite:

On calcule l'intersection de la droite support avec le cercle. Puis s'il y a deux intersections distinctes, on récupère les points d'intersection appartenant au segment de droite.

Comme pour le découpage d'un segment de droite, pour éviter les singularités (cercle passant par un sommet), on oriente les segments vers le bas et on ne prend pas en compte les points d'intersection confondus avec l'extrémité fin :

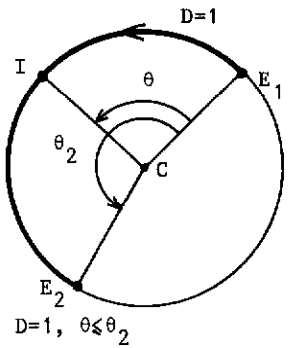


- intersection du cercle avec un arc de cercle:

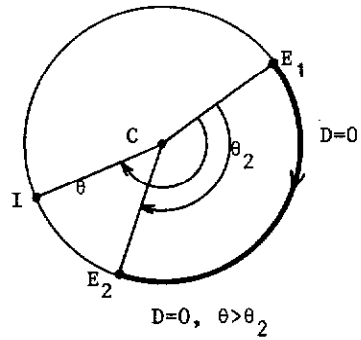
On calcule l'intersection des deux cercles. Puis s'il y a deux intersections distinctes, on récupère les points d'intersection qui appartiennent à l'arc de cercle (voir chapitre 4.8.2).

Si l'un des points d'intersection récupéré est confondu avec l'une des extrémités de l'arc de cercle, on vérifiera comme précédemment si c'est un point de naissance (on le garde) ou un point de mort (on le supprime).

Un point I du cercle support appartient à l'arc de cercle (E_1,E_2), si l'angle $\theta=(\widehat{E_1CI})$ est inférieur ou égal à l'angle $\theta_2=(\widehat{E_1CE_2})$ où C est le centre du cercle.



I ∈ arc de cercle



I ∉ arc de cercle

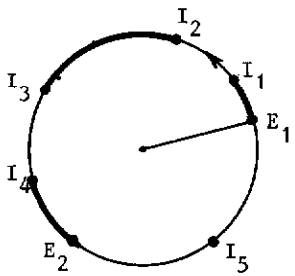
2- Extraction des parties visibles

Soit $(I_k)_{k=1,\ldots,N}$ l'ensemble de N points d'intersection.

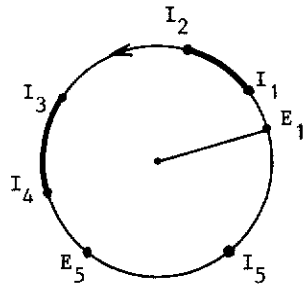
On ordonne cet ensemble dans le sens croissant des angles $(\widehat{E_1CI_k})_{k=1,\ldots,N}$ en tenant compte du sens de l'arc (E_1,E_2).

On regarde si E_1 est intérieur ou non au contour (voir chapitre 4.8.1.1.).

Sachant si en E_1 nous sommes à l'intérieur ou à l'extérieur, nous appliquons le contrôle de parité sur la liste ordonnée des intersections appartenant à l'arc (E_1,E_2) (c.a.d. tant que $(\widehat{E_1CI_k})\leq(\widehat{E_1CE_2})$). Chaque fois que nous rencontrons un point, nous entrons ou sortons de la tache (voir figure 4.19).



a) E_1 ∈ tache



b) E_1 ∉ tache

Figure 4.19.: Exemple d'extraction des parties visibles (D=1)

L'inconvénient de cet algorithme est l'utilisation de la fonction trigonométrique arctg pour le calcul des angles.

4.8.4 - CONCLUSION

Nous constatons une fois encore que l'application des idées du suivi de contour permet de surmonter simplement les singularités rencontrées dans la généralisation des trois problèmes élémentaires (comparaison point-contour, découpage d'un segment et d'un arc de cercle par un contour) aux contours non polygonaux. On remarque également que les contours peuvent être convexes ou non.

4.9. Conclusion

Sur les algorithmes élémentaires de découpage, nous pouvons faire plusieurs remarques :

. Nous constatons que la complexité des algorithmes dépend essentiellement de la nature des figures traitées : à savoir rectangulaires (bords parallèles aux axes), convexes ou non convexes. Dans toute application, il conviendra donc de choisir en fonction de la nature de la scène et de ses manipulations, l'algorithme adéquat.

. Nous remarquons également que le bon fonctionnement des algorithmes, et en particulier le traitement des singularités spécifiques à chaque problème, repose la plupart du temps sur une analyse rigoureuse de la définition de l'intersection de deux segments de droite, celle-ci variant d'une application à une autre.

. Lorsque nous traitons une tache par opposition au dessin au trait, deux sortes d'approches peuvent être envisagées :

- le découpage implicite : nous raisonnons ici sur les surfaces engendrées par les contours des taches. Les taches résultant du découpage, sont caractérisées par un ensemble de points (surface) et non par leurs contours.

- le découpage explicite : nous travaillons sur les contours des taches, comme pour le dessin au trait. Les taches produites sont définies par la description de leurs contours.

. Une méthode d'analyse générale se dégage de cette étude : il s'agit de la technique du "suivi de contour". En effet, elle permet de traiter indifféremment, des contours convexes ou quelconques, en surmontant simplement toutes les singularités. D'autre part, les analyses des contours effectués pour le découpage, pour le remplissage ou le hachurage à l'aide de cette technique étant identiques, nous pouvons effectuer une partie de ces traitements simultanément, en utilisant et en produisant des structures de données uniformes. Nous gagnons alors en espace mémoire et en vitesse d'exécution.

. Dans le cadre des contours non polygonaux, la logique du suivi de contour nous permet également de traiter simplement certains problèmes élémentaires (comparaison point-contour, découpage d'un segment et d'un arc de cercle). Il serait aussi intéressant d'envisager l'étude d'autres opérations comme l'intersection de deux contours non polygonaux convexes par exemple.

Bibliographie

<ACKLAND-WESTE 81>

B. ACKLAND, N. WESTE
The edge-flag algorithm. A fill method for raster scan displays. IEEE Trans. Comp., Vol. C-30, n° 1, January 1981.

<ALLIAUME 81>

F. ALLIAUME
Algorithmes pour identifier les configurations géométriques de 2 segments de droite.
Rapport de recherche, Université de Nantes, Mai 1981.

<ATHERTON-WEILER 77>

P. ATHERTON et K. WEILER
Hidden surface removal using polygon area sorting. Computer graphics, vol. 11(2) : 214, Summer 1977.

<BENTLEY-OTTMAN 79>

J.L. BENTLEY, S. OTTMAN
Algorithms for reporting and counting geometric intersections. IEEE Trans. Comp, Vol. C-28, n°9, September 1979.

<BLOCH 81>

M. BLOCH
Génération de taches bicolores - application aux caractères d'imprimerie - problèmes de nature ordinale.
Thèse Docteur-Ingénieur, St-Etienne, Juillet 1981.

<BRESENHAM 65>

J.E. BRESENHAM
Algorithm for computer control of a digital plotter
IBM System Journal, Vol. 4, n°1 pp. 25-30, 1965.

<BRESENHAM 77>

J.E. BRESENHAM
A linear algorithm for incremental digital display of circular arcs
CGIP, Vol. 20, n°2, Février 1977.

<LUCAS et al.79>

M. LUCAS et al.
La réalisation des logiciels graphiques interactifs
Ecole d'Eté du Bréau-sans-nappe - Juillet 1979.
Ed. Eyrolles 1981.

<BRASSEL-FEGEAS 79>

K.E. BRASSEL, R. FEGEAS
An algorithm for shading of regions on vector display devices.
Computer Graphics, Vol. 13, n°3, pp. 126-133, 1979.

<COUEIGNOUX-GUEDJ 80>

P. COUEIGNOUX, R. GUEDJ
Computer generation of colored planar patterns on TV-like rasters.
Proceedings of the IEEE, Vol. 68, n°7, July 1980.

<COUEIGNOUX 81>

P. COUEIGNOUX
Character generation by computer
CGIP, 16, pp. 240-269, 1981.

<EDMONDS et al. 82>

E.A. EDMONDS, A. SEHAPPO, S.A.R. SCRIVENER
Image handling in-two-dimensional design
IEEE Computer Graphics and Applications, Vol.2, n°5, Juillet 1982.

<FOLEY-VAN DAM 82>

J.D. FOLEY et A. VAN DAM
Fundamentals of Interactive Computer Graphics
Addison-Wesley Systems Programming Series, 1982.

<FRANKLIN 79>

N.R. FRANKLIN
Evaluation of algorithms to display vector plots on raster devices
CGIP, Vol. 11, pp. 377-397, 1979.

<HEGRON 82a>

G. HEGRON
Algorithmes élémentaires pour la production d'images par ordinateur.
Etude bibliographique.
Conv. P et T n°82 35068, RR IMI-info-3, Université de Nantes, Juin 1982.

<HEGRON 82b>

G. HEGRON
Techniques de remplissage de taches sur une surface à pointillage.
Conv. P et T n°82 35068, RR IMI-info-4, Université de Nantes, Septembre 1982.

<HEGRON 83a>

G. HEGRON
Algorithmes élémentaires de découpage et traitements de nature géométrique pour la production d'images par ordinateur.
Conv. P et T n°82 35068, RR IMI-info-7, Université de Nantes, Janvier 1983.

<HEGRON 83b>

G. HEGRON
La technique du suivi de contour en synthèse d'images et ses applications.
PI IRISA n° 209, Université de Rennes, Octobre 1983.

<HEGRON 83c>

G. HEGRON
Etude comparative d'algorithmes élémentaires pour la synthèse d'image par ordinateur.
Thèse de 3ème cycle, Université de Rennes, Septembre 1983.

<JARVIS et al. 76>

J.F. JARVIS, C.N. JUDICE, W.H. NINKE
A survey of techniques for the display of continuous tone pictures on bilevel displays.
CGIP, vol.5, pp. 13-40, 1976.

<LIEBERMAN 78>

H. LIEBERMAN
How to color in a coloring book
Comp. graphics, Vol. 12, August 1978.

<LITTLE-HEUFT 79>

W.D. LITTLE, R. HEUFT
An area shading display system
IEEE Trans. Comp., Vol. C-28, n°7, Juillet 1979.

<LUCAS 77>

M. LUCAS
Contribution à l'étude des techniques de communications graphiques avec un ordinateur. Eléments de base de logiciels graphiques interactifs.
Thèse d'Etat, IMAG Grenoble, Décembre 1977.

<MAXWELL-BAKER 79>

P.C. MAXWELL, P.W. BAKER
The generation of polygones representing circles, ellipses and hyperboles.
CGIP, Vol. 10, pp. 84-93, 1979.

<MARTINEZ 82>

F. MARTINEZ
Vers une approche systématique de la synthèse d'images.
Aspect logiciel et matériel. Thèse d'Etat, INPG, Grenoble, Novembre 1982.

<MERIAUX 79>

M. MERIAUX
Etude et réalisation d'un terminal graphique couleur tridimensionnel fonctionnant par taches.
Thèse de Docteur-Ingénieur, Université de Lille, Janvier 1979.

<MORVAN-LUCAS 76>

P. MORVAN, M. LUCAS
Images et ordinateur - Introduction à l'infographie interactive
Ed. Larousse, Série informatique, Paris, Septembre 1976.

<NEWMAN-SPROULL 73>

W. NEWMAN, R.F. SPROULL
Principles of interactive computer graphics
Mc. Graw-Hill, New-York, First Edition 1973.

<NEWMAN-SPROULL 79>

W. NEWMAN, R.F. SPROULL
Principles of interactive computer graphics
Mc. Graw-Hill, New-York, Second Edition 1979.

<O'ROURKE et al. 82>

J. O'ROURKE, C.B. CHIEN, T. OLSON et D. NADDOR
A new algorithm for intersection convex polygons.
CGIP, vol.19, n°4, pp. 384-391, Août 1982.

<PAVLIDIS 79>

T. PAVLIDIS
Filling algorithms for raster graphics
CGIP, Vol.10, pp. 126-141, June 1979.

<PAVLIDIS 81>

T. PAVLIDIS
Contour Filling in raster graphics
Comp. graphics, Vol. 15, n°3, August 1981.

<PAVLIDIS 82>

T. PAVLIDIS
Algorithms for graphics and Image Processing
Ed. Springer-Verlay, Bell Laboratories, 1982.

<ROGERS-ADAMS 76>

D.F. ROGERS, J.A. ADAMS
Mathematical elements for computer graphics
Mc Graw-Hill book company, New-York, 1976.

<ROY 83>

M.A. ROY
Etude comparative d'algorithmes pour l'amélioration de dessins au trait sur une surface point par point
IRISA Rennes, PI n°189, Université de Nantes, RR IMI-Info 10, Janvier 1983.

<SECHER 83>

E. SECHER
Conception et réalisation d'un logiciel de saisie et restitution de cartes élémentaires.
IRISA Rennes, PI n°188, Université de Nantes, RR IMI-Info 9, Janvier 1983.

<SHAMOS 75>

M. SHAMOS
Geometric complexity
Proceeding of the 7th ACM symposium on the theory of Computing, May 1985.

<SHANI 80>

H. SHANI
Filling regions in binary raster images : a graph-theorie approach
Comp. graphics, Vol. 14, n°3, pp. 321-327, July 1980.

<SHAMOS-HOEY 76>

M. SHAMOS, S. HOEY
Geometric intersection problems
17th FOCS Conference, 1976.

<SICO 82>

C. SICO
Génération de taches bicolores - Applications aux caractères d'imprimerie - Problème de nature géométrique.
Thèse de Docteur-Ingénieur, Saint-Etienne, Mars 1982.

<SMITH 79>

A.R. SMITH
Tint fill
Proc. SIGGRAPH-ACM, pp. 276-283, August 1979.

<SUTHERLAND-HODGMAN 74>

I.E. SUTHERLAND, G.W. HODGMAN
Reentrant polygon clipping
CACM, Vol. 17, n°1, Janvier 1974.

<SUTHERLAND-SPROULL 68>

I.E. SUTHERLAND, R.F. SPROULL
A clipping divider
FICC 1968, Thompson books, Washington, D.C., p.765.

<WEILER 80>

K. WEILER
Polygon comparison using a graphe representation
Comp. graphics SIGGRAPH-ACM, Vol. 14, n°3, Juillet 1980.

Notation : CGIP = Computer Graphics and Image Processing.

Liste des algorithmes

Index

N.B. Les noms d'algorithmes sont en majuscules.

Imprimerie GAUTHIER-VILLARS, France
7197-85 Dépôt légal, Imprimeur, n° 2816
Dépôt légal : août 1985 *Imprimé en France*

DUNOD
informatique

L'informatique

- **Ainsi naquit l'informatique**, par R. Moreau
- **Principes des ordinateurs**, par P. de Miribel
- **Emploi des ordinateurs**, par J.-C. Faure
- **Aide-mémoire d'informatique**, par Ch. Berthet
- **Let's talk D.P. ; lexique d'informatique**, par J.-P. Drieux et A. Jarlaud
- **D.P. words ; dictionnaire d'informatique anglais-français, français-anglais**, par G. Fehlmann

La microinformatique

- **Micro-informatique ; architecture, interfaces et logiciel**, par J.-D. Nicoud
- **Le choix d'un microordinateur**, par H.-P. Blomeyer-Bartenstein
- **Introduction aux microprocesseurs et aux microordinateurs**, par C. Pariot
- **Microprocesseurs : du 6800 au 6809, modes d'interfaçage**, par G. Révellin
- **Interfaçage des microprocesseurs**, par M. Robin et T. Maurin

Les applications de l'informatique

- **Analyse organique, tomes 1 et 2**, par C. Cochet et A. Galliot
- **Les systèmes d'information ; analyse et conception**, par Galacsi
- **Systèmes d'exploitation des ordinateurs**, par Crocus
- **Bases de données, méthodes pratiques sur maxi et mini-ordinateurs**, par D. Martin
- **Les fichiers**, par C. Jouffroy et C. Létang
- **Bases de données et systèmes relationnels**, par C. Delobel et M. Adiba
- **Systèmes informatiques répartis**, par Cornafion
- **Téléinformatique**, par C. Macchi et J.-F. Guilbert
- **Bases d'informations généralisées**, par C. Chrisment, J.-B. Crampes et G. Zurfluh

L'art de programmer

- **Initiation à l'analyse et à la programmation**, par J.-P. Laurent
- **Exercices commentés d'analyse et de programmation**, par J.-P. Laurent et J. Ayel
- **Les bases de la programmation**, par J. Arsac
- **Proverbes de programmation**, par H.-F. Ledgard
- **Programmation, tomes 1 et 2**, par A. Ducrin
- **Théorie des programmes**, par C. Livercy
- **Algorithmique**, par P. Berlioux et Ph. Bizard
- **Synchronisation de programmes parallèles**, par F. André, D. Herman et J.-P. Verjus
- **Algorithmique du parallélisme**, par M. Raynal
- **Synthèse d'image : algorithmes élémentaires**, par G. Hégron

Les langages de programmation

- **La programmation en assembleur**, par J. Rivière
- **Exercices d'assembleur et de macro-assembleur**, par J. Rivière
- **Basic : programmation des microordinateurs**, par A. Checroun
- **Introduction à A.P.L.**, par S. Pommier
- **Cobol. Initiation et pratique**, par M. Barès et H. Ducasse
- **Fortran IV**, par M. Dreyfus
- **La pratique du fortran**, par M. Dreyfus et C. Gangloff
- **Le langage de programmation PL/1**, par Ch. Berthet
- **Le langage ADA : manuel d'évaluation**, par D. Le Verrand

DUNOD

informatique

À l'heure où chacun peut admirer quotidiennement les images produites par ordinateur, il est étonnant de constater que des créations extraordinaires sont issues d'algorithmes souvent mal maîtrisés ou mal choisis.

Fruit de l'analyse, de la classification et de la comparaison des nombreuses techniques, ce livre recherche une sélection d'algorithmes dans trois domaines à la base de la synthèse d'image : la génération de courbes (droites, coniques, ...), le remplissage de surfaces (polygones, hachurages, ...) et les traitements de nature géométrique (découpage de droites, de polygones, ...).

Malgré la diversité des solutions offertes pour créer une image, l'auteur a su dégager un ensemble de procédés méthodologiques en traitant plus particulièrement de deux techniques : la génération de courbes avec le mécanisme de Bresenham et les traitements variés à partir du suivi de contours.

Tout en offrant aux lecteurs la possibilité de réaliser un véritable générateur personnel des algorithmes graphiques de base, cet ouvrage s'adresse plus particulièrement aux ingénieurs qui souhaitent connaître les meilleures méthodes pour les intégrer à leur machine. En outre, il met à la disposition des étudiants et des chercheurs un catalogue complet et synthétique des traitements classiques pour la synthèse d'image.