TABLE DES MATIERES

TABLE DES FIGURES

INTRODUCTION GENERALE

INTRODUCTION GENERALE

Aujourd'hui, grâce à la tomographie à rayon X, nous pouvons prendre une image volumique de l'espace poral dans un echantillon du sol. Un calcule mathématique basé sur la triangulation de Delaunay permet de déceler le squelette de la forme volumique donnée par l'image pour donner une modélisation de ce dernier par un ensemble de boules maximale.

En ayant comme données de départ un fichier de boule qui est le résultat de cette modélisation notre travail consiste à développer un programme dont les principales taches sont :

- Une simulation qui permet la visualisation, en 3D et sous différente angles, de notre model.
- Une amélioration du modél de départ par l'introduction d'autres primitives géométriques telles que cylindres et cônes.

Pour atteindre cet objectif nous avons développé un programme C++ composé de deux modules principales ; un module OpenGL fournissant un systeme de visualisation complet qui prend en charge la gesion de la camera ainsi que la possibilité de deplacer les objets graphiques au niveau de la scene de similation.

Un autre module C++ qui se charge de la meilleure stricturation des données ainsi que des calculs mathématiques necessaire pour detecter les differente relations qui peuvent exister entre les données.

Partie I : Etat de l'Art

VISION PAR ORDINATEUR

CHAPITRE 1

Chapitre I : Vision par ordinateur

INTRODUCTION

La vision naturelle est un phénomène merveilleux et complexe, elle a suscité l'intérêt de nombreux scientifiques et philosophes depuis déjà très longtemps.
Son étude concerne un vaste ensemble de disciplines indépendantes (optique, physiologie, neurologie, psychologie, … etc.).
Lorsque l'on demande à une personne de décrire les objets qui l'entoure elle n'éprouve aucune difficulté : table, chaise, stylo…etc., pourtant l'information disponible à la rétine (dans l'image) n'est qu'une collection de point (environ un million) ni plus ni moins. En chaque point ou pixel (*picture element*) il y'a tout simplement une information qui donne une indication quant à la quantité de lumière et la couleur qui proviennent de l'espace environnant et qui ont été projetées à cet endroit de la rétine.
Guidé à la fois par l'information codée dans l'image (ou la rétine) et par ses propres connaissances le processus visuel construit des percepts.
La reconnaissance des objets vus est le résultat final d'un processus d'interprétation qui fait partie intégrante du system de vision.
Avec la naissance de machines de calcul de plus en plus sophistiquées un certain nombre de scientifiques se sont attaquées au problème de la vision d'un point de vue quantitatif : est-il possible de construire un modèle computationnel pour la perception visuelle ?
Attention il ne s'agit pas de fournir une explication de comment marche la vision biologique mais de créer un modèle qui vu de l'extérieur possède des propriétés semblables.

Ce modèle artificiel peut-il être d'une utilité quelconque quant à la vision biologique ?

Peut-il constituer la base d'une nouvelle technologie -des machines qui voient ?

Pour répondre à cette question une nouvelle théorie est née « LA VISION PAR ORDINATEUR » un domaine actif de recherche depuis 40 ans environ, elle se réfère aux algorithmes variés pour restaurer ou interpréter les images numériques.
Voici comment la vision par ordinateur peut s'énoncer brièvement. « La vision est un processus de traitement de l'information. Elle utilise des stratégies bien définies afin d'atteindre ses buts. L'entrée d'un système de vision est constituée par une séquence d'images. Le système lui-même apporte un certain nombre de connaissances qui interviennent à tous les niveaux. La sortie est une description de l'entrée en termes d'objets et de relations entre ces objets. » [1]

I. Détection de contours

La détection de contours est une étape préliminaire à de nombreuses applications de l'analyse d'images. En effet les contours constituent des indices riches, au même titre que les points d'intérêts, pour toute interprétation ultérieure de l'image.
Les contours dans une image proviennent des :

- ☞ Discontinuités de la fonction de réflectance (texture, ombre),
- ☞ Discontinuités de profondeur (bords de l'objet).

et sont caractérisés par des discontinuités de la fonction d'intensité dans les images.
Le principe de la détection de contours repose donc sur l'étude des dérivéesde la fonction d'intensité dans l'image : les extrema locaux du gradient de la fonction d'intensité et les passages par zéro du laplacien. La difficulté réside dans la présence de bruit dans les images. **[1]**

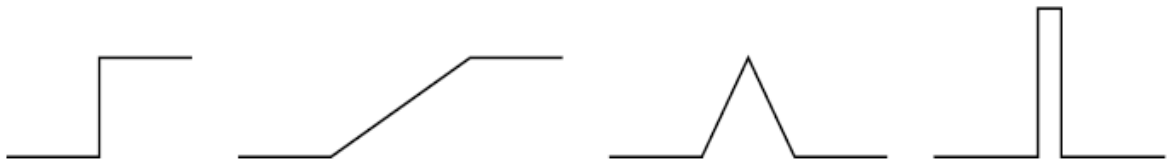


Figure1: Différents types de contours : marche, rampe, toit, pic.

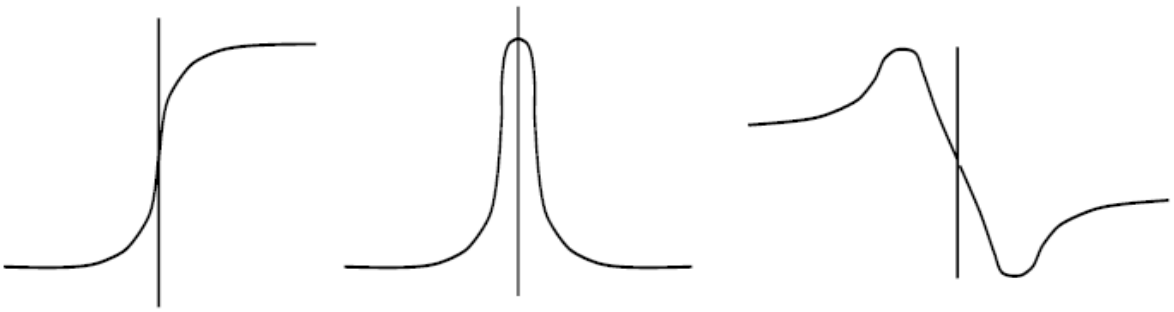


Figure 2 : La fonction d'intensité au voisinage d'un contour en marche et sec dérivées première et seconde.

1. Définitions

a. Le Filtrage linéaire d'une image

Filtrer une image consiste à convoluer sa fonction d'intensité *I(x,y)* avec une fonction *h(x,y)* appelée réponse impulsionnelle du filtre.

$$I'(x,y) = h(x,y) * I(x,y),$$

$$I'(x,y) = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} h(u,v)I(x-u,y-v)du\,dv,$$

$$I'(x,y) = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} h(x-u,x-v)I(u,v)du\,dv,$$

Dans le cas discret :

$$I'(x,y) = \sum_{u=-H/2}^{+H/2}\sum_{v=-H/2}^{+H/2} h(u,v)I(x-u,y-v).$$

où H correspond à la dimension du masque de filtrage.

b. Le gradient d'une image

Le gradient dune image est le vecteur $\nabla I(x,y)$ défini par :

$$\nabla I(x,y) = (\frac{\partial I(x,y)}{\partial x}, \frac{\partial I(x,y)}{\partial y})^t.$$

Il est donc caractérisé par un module G et une direction φ dans l'image :

$$m = \sqrt{(\frac{\partial I(x,y)}{\partial x})^2 + \frac{\partial I(x,y)}{\partial y})^2},$$

$$\phi = \arctan(\frac{\partial I(x,y)}{\partial y} / \frac{\partial I(x,y)}{\partial x}).$$

- La direction du gradient maximise la dérivée directionnelle
- La dérivée de *I(x,y)* dans une direction donnée **d** s'écrit : $\nabla I(x,y) \cdot d$
- Le gradient d'une image filtrée : $\nabla I'(x,y) = \nabla(I(x,y) * h(x,y)) = \nabla I(x,y) * h(x,y) = I(x,y) * \nabla h(x,y).$

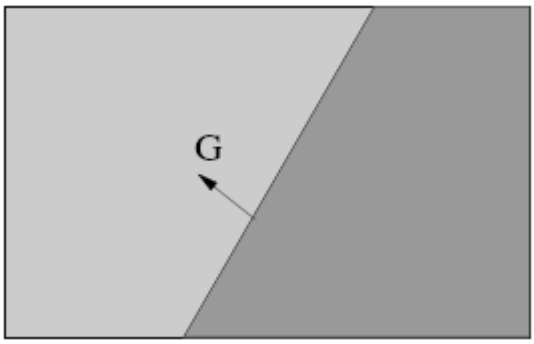


Figure 3 : Le gradient

c. Le laplacien d'une image

Le laplacien d'une image d'intensité *I(x,y)* est défini par :

$$\nabla^2 I(x,y) = \frac{\partial^2 I(x,y)}{\partial x^2} + \frac{\partial^2 I(x,y)}{\partial y^2}$$

- Invariant aux rotations de l'image.
- Le laplacien est souvent utilisé en amélioration d'images pour accentuer.
- l'effet de contour : $I'(x,y) = I(x,y) - c\nabla^2 I(x,y)$
- Sensibilité au bruit accrue par rapport au gradient.
- Le laplacien d'une image filtrée : $\Delta I'(x,y) = \Delta I(x,y) * h(x,y) = I(x,y) * \Delta h(x,y)$

d. Les filtres séparables

Un filtre à réponse impulsionnelle *h(x,y)* séparable selon _ et _ est un filtre pour lequel : h(x,y)=hx(x)*hy(y) ce qui se traduit pour le filtrage d'une image par

$$I'(x,y) = h(x,y) * I(x,y),$$

$$I'(x,y) = h_y(y) * (h_x(x) * I(x,y)),$$

et pour les dérivées :

$$\frac{\partial I'(x,y)}{\partial x} = I(x,y) * (\frac{\partial h_x(x)}{\partial x}\, h_y(y)),$$
$$\frac{\partial I'(x,y)}{\partial y} = I(x,y) * (h_x(x)\, \frac{\partial h_y(y)}{\partial y}),$$
$$\Delta I'(x,y) = I(x,y) * (\Delta h_x(x) h_y(y) + h_x(x) \Delta h_y(y)),$$

Les principaux intérêts des filtres séparables sont :

- ☞ Ramener le problème du filtrage d'un signal bidimensionnel à celui du filtrage d'un signal monodimensionnel,
- ☞ Réduire le temps de calcul,
- ☞ Possibilité d'implanter récursivement le filtre.

e. Dérivation par différence finies

Une image est discrète par nature. Les premières approches ont donc consisté à approximer les dérivées par différence :

$$\nabla_u I(u,v) = I(u,v) - I(u-n,v),$$

ou :

$$\nabla_u I(u,v) = I(u+n,v) - I(u-n,v),$$

avec, en général $n = 1$.

Ces dérivées sont calculées par convolution de l'image avec un masque de différences.

f. Dérivation par filtrage optimal

Les dérivations présentées consistent à convoluer l'image par des masques de petites dimensions. Ces approches sont donc dépendantes de la taille des objets traités, elles sont aussi très sensible au bruit. Un autre type d'approches plus récentes repose sur la définition de critères d'optimalité de la détection de contours ces critères débouchant sur des filtres de lissage optimaux.

g. De l'image des dérivées aux contours

Les différents filtres présentés permettent de calculer le gradient ou le laplacien d'une image mais ne donnent pas des points de contours. Un traitement ultérieur est nécessaire, ce traitement étant dépendant du type d'approche choisi.

2. Les approches de détection de contours

Deux approches sont utilisées:

- **Approche gradient :** détermination des extrema locaux dans la direction du gradient,
- **Approche laplacien :** détermination des passages par zéro du laplacien.

Ces approches reposent sur le fait que les contours correspondent des discontinuités d'ordre 0 de la fonction d'intensité.

Le calcul de dérivée nécessite un pré-filtrage des images. Filtrage linéaire pour les bruits de moyenne nulle (par exemple bruit blanc Gaussien, filtre Gaussien). Filtrage non-linéaire pour les bruits impulsionnels (filtre médian par exemple).

Les différentes approches existantes se classent ensuite suivant la manière d'estimer les dérivées de la fonction d'intensité :

- **Différences finies,**
- **Filtrage optimal,**
- **Modélisation de la fonction d'intensité.**

II. Détection de points d'intérêts

La détection de points d'intérêts (ou coins) est, au même titre que la détection de contours, une étape préliminaire à de nombreux processus de vision par ordinateur. Les points d'intérêts, dans une image, correspondent à des doubles discontinuités de la fonction d'intensités. Celles-ci peuvent être provoquées, comme pour les contours, par des discontinuités de la fonction de réflectance ou des discontinuités de profondeur. Ce sont par exemple : les coins, les jonctions en T ou les points de fortes variations de texture. **[1]**

Figure 4 : Différents types de points d'intérêt : coins, jonction en T et point de fortes variations de texture.

1. Les Avantages

Les Avantages des points d'intérêts sont :

- Sources d'informations plus fiables que les contours car plus de contraintes sur la fonction d'intensité,
- Robuste aux occultations (soit occulté complètement, soit visible),
- Pas d'opérations de chainage (=> contours !),
- Présents dans une grande majorité d'images (≠ contours !).

2. Les Différentes approches

De nombreuses méthodes ont été proposées pour détecter des points d'intérêts. Elles peuvent être classées grossièrement suivant trois catégories :

- **Approches contours :** l'idée est de détecter les contours dans une image dans un premier temps. Les points d'intérêts sont ensuite extraits le long des contours en considérants les points de courbures maximales ainsi que les intersections de contours,

- **Approches intensité :** l'idée est cette fois-ci de regarder directement la fonction d'intensité dans les images pour en extraire directement les points de discontinuités,
- **Approches à base de modèles :** les points d'intérêts sont identifiés dans l'image par mise en correspondance de la fonction d'intensité avec un modèle théorique de cette fonction des points d'intérêts considérés.

Les approches de la deuxième catégorie sont celles utilisées généralement. Les raisons sont : indépendance vis à vis de la détection de contours (stabilité), indépendance vis à vis du type de points d'intérêts (méthodes plus générales).

III. La segmentation

La segmentation d'images est un problème important dans le traitement numérique des images. La segmentation est une opération qui a pour objectif la description de l'information contenue dans l'image en donnant une représentation plus condensée et facilement exploitable. Il est difficile de définir d'une manière absolue, une 'bonne' segmentation. La segmentation, souvent, n'est pas une fin en soi, sa qualité influe directement sur les résultats obtenus par les traitements situés en aval de l'étape de segmentation. **[2]**

1. Définition

On peut définir la segmentation comme étant une partition de l'image I en sous ensembles R_i appelés régions, tels que :

$$\forall i \; R_i \neq \emptyset$$

$$\forall i, j ; i \neq j \quad R_i \cap R_j = \emptyset$$

$$I = \bigcup_i R_i$$

2. Les différentes approches de segmentation

La segmentation fait référence aux notions de différence et de similarité perçues par le système visuel humain. Ceci donne naissance à deux approches couramment qualifiés d'approche 'frontière' et d'approche 'région'. L'approche région s'attache à faire apparaître des régions homogènes selon un critère (niveaux de gris ou texture), alors que l'approche frontière tente de trouver des contours ou frontières de régions présentant une variation rapide du même critère.

Un algorithme de segmentation s'appuie donc sur :

- la recherche de discontinuités afin de mettre en évidence les contours,
- la recherche d'homogénéité locale pour définir les régions,
- ou encore sur la coopération des deux principes.

3. Les types de segmentation

Il existe deux types de segmentation à savoir :

a. Segmentation en contours

La recherche des contours dans une image numérique est un des problèmes les plus étudiés depuis l'origine des travaux sur l'imagerie numérique. Ceci est en grande partie dû à la nature très intuitive du contour qui apparaît très naturellement comme l'indice visuel idéal dans la plus grande partie des situations.

Très schématiquement, les contours sont les lieux de variations significatives de l'information niveaux de gris (il n'est pas question ici des travaux sur les contours dans les images couleurs ou multispectrales). Dans cette approche, on suppose que l'image est une mosaïque de régions parfaitement homogènes. C'est à dire que les contours recherchés sont de type créneaux. De plus, la transition étant stricte, un contour doit être une chaîne de pixels d'épaisseur 1. Cette restriction sur la nature du contour a été imposée dans un premier temps pour des raisons de formalisation mathématique. Il est possible de construire des processus capables d'extraire d'autres types de contours comme par exemple des vallées ou des toits. Cependant, il n'existe pas à l'heure actuelle de processus complet et général qui pourrait extraire tous les types de contour.

La notion de contour étant reliée à celle de variation, il est évident qu'une telle définition nous amène tout naturellement vers une évaluation de la variation en chaque pixel. Une variation existera si le gradient est localement maximum ou si la dérivée seconde (à définir dans un espace bi-dimensionnel) présente un passage par zéro. Les principaux algorithmes connus (Sobel, Prewitt, Kirsh, Canny, Dériche, ...) se focalisent sur ce premier aspect du contour.

Il existe moins de travaux sur la formalisation de la deuxième partie consistant à passer d'une mesure locale de variations à des chaines de points d'épaisseur 1. C'est pourtant cette deuxième partie qui fait souvent la différence et la qualité visuelle d'un résultat.

Nous allons d'abord présenter les méthodes d'extraction de points de contour, par des techniques de filtrage gradient ou laplacien. Puis nous verons quelques techniques de chaînage de ces points (suivi de contour) et de représentation des contours finaux ainsi obtenus

i. Calcul du gradient

Le gradient, en un pixel d'une image numérique, est un vecteur caractérisé par son amplitude et sa direction. L'amplitude est directement liée à la quantité de variation locale des niveaux de gris. La direction du gradient est orthogonale à la frontière qui passe au point considéré.

La méthode la plus simple pour estimer un gradient est donc de faire un calcul de variation monodimensionnelle, i.e. en ayant choisi une direction donnée. On a alors le schéma suivant :

$$\mathbf{Gd(x,y) = (I * Wd)(x,y)}$$

où Wd désigne l'opérateur de dérivation dans la direction d et * le produit de convolution.

$$\mathbf{Gd(x,y) = \Sigma\ i=-m,+m\ \Sigma\ j=-n,\ +n\ I(x+i,y+j)\ .\ Wd(i,j)}$$

Dans cette version discrète, la taille de cet opérateur est donnée par le couple (m, n). Sauf cas très particulier, on utilise toujours m=n.

Il existe de très nombreux opérateurs différents (Roberts, Sobel, Prewitt, Kirsch, ...) qui ont globalement les mêmes propriétés.

Le gradient étant un vecteur, l'approche la plus classique pour estimer le gradient consiste à choisir deux directions privilégiées (naturellement celles associées au maillage, i.e. ligne et colonne) orthogonales, sur lesquelles on projette le gradient. A partir de deux calculs identiques à celui présenté ci-dessus, on peut donc obtenir une connaissance parfaite du gradient :

$$G(x,y) = (GX(x,y),GY(x,y)\ ) = ((I * WX)(x,y),\ (I * WY)(x,y))$$

L'amplitude du gradient s'obtient alors par l'une des formules suivantes :

$$m(x,y) = (GX(x,y)2 + GY(x,y)2)\ ½$$
$$m(x,y) = |GX(x,y)| + |GY(x,y)|$$
$$m(x,y) = Max(|GX(x,y)| + |GY(x,y)|)$$

Et la direction du gradient est donnée par:

$$d(x,y) = Arctg(\ GY(x,y) / GX(x,y))$$

Cette approche présente comme principal inconvénient le fait que la direction peut prendre n'importe quelle valeur réelle. Ceci n'est pas en accord avec la nature discrète d'une image. Que représente une frontière orientée à 17° sur une grille ? C'est pourquoi on peut adopter un schéma différent adapté à la nature discrète de l'image. Il s'agit alors de calculer le gradient, non plus dans deux directions, mais dans toutes les directions possibles de l'image : 0°, 45°, 90°, 135°. On peut se contenter de ces 4 directions. On a alors :

$$m(x,y) = Maxd = 0°, 45°, 90°, 135°\ md(x,y)$$
$$d(x,y) = arg\ Maxd\ md\ (x,y)$$

Là encore, on peut imaginer plusieurs types de masques (Sobel, Prewitt, Kirsch, ...)

ii. Calcul du Laplacien Dérivée seconde et recherche des zero-crossing

Une frontière est un lieu de variation. On peut donc la localiser par la recherche du maximum de la dérivée première (approche basée sur le calcul du gradient). On peut aussi rechercher le passage par zéro de la dérivée seconde.

$$\nabla^2 I = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

Dans le cas d'une image, il n'existe pas une dérivée seconde unique mais 4 dérivées partielles (selon x^2, y^2, xy et yx). En pratique, on lève cette ambiguïté en ayant recours à l'opérateur Laplacien qui fait la somme des deux dérivées partielles principales.
Afin de limiter les réponses dues au bruit de l'image I, elle est préalablement filtrée par un filtre G. On obtient alors, grâce aux propriétés de l'opérateur produit de convolution :

$$\nabla^2(I * G) = (\nabla^2 I) * G = I * (\nabla^2 G)$$

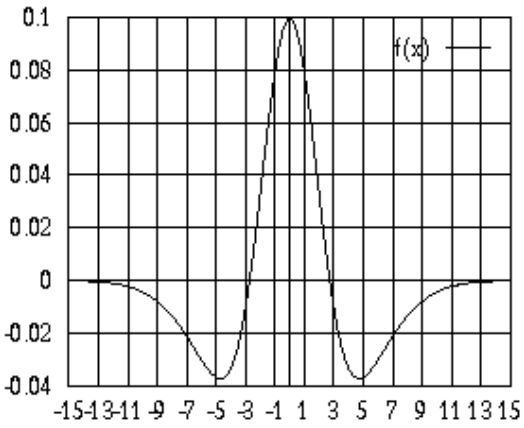
Les opérateurs de filtrage et de dérivation se font donc en une seule étape de calcul.
Le plus souvent, on fait appel à un filtrage Gaussien :

$$G(r) = -\frac{1}{\sqrt{2\pi}\sigma} exp(-\frac{r^2}{2\sigma^2}) \quad r = \sqrt{x^2 + y^2}$$

dont le Laplacien est

$$\nabla^2 G(r) = -\frac{1}{\pi\sigma^4}(1 - \frac{r^2}{2\sigma^2}) exp(-\frac{r^2}{2\sigma^2})$$

plus connu sous le nom de "chapeau mexicain".



Cette fonction sert de modèle au fonctionnement de la cellule ON/OFF, base de notre système visuel. Le paramètre sert a régler la résolution à laquelle les contours sont détectés. Son choix est très difficile a priori car il est peu probable qu'une valeur unique permette de représenter efficacement toutes les résolutions présentes dans une image (surtout pour les images naturelles).

Dans le cas discret, la plus simple approximation du Laplacien d'une Gaussienne est le filtre :

-1	**-1**	**-1**
-1	**+8**	**-1**
-1	**-1**	**-1**

connu sous le nom de masque Laplacien.

- **Avantages**

Cette approche est proche du mécanisme de la vision humaine :

- un seul paramètre,
- pas de seuil de significativité de l'amplitude,
- contours fermés.

- **Inconvénients**

- Plus grande sensibilité au bruit de par l'utilisation de la dérivée seconde,
- Pas d'information sur l'orientation du contour (i.e. direction du gradient).

Pour palier ces défauts, on combine souvent les approches gradient et passage par zéro en ne recherchant le passage par zéro que dans le voisinage des maxima du gradient.

iii. Chaînage des points de contour

Le calcul du gradient permet d'obtenir, en tout pixel d'une image, une information sur la présence de contours. Cependant, il est nécessaire d'enchaîner plusieurs traitements pour effectivement passer du gradient aux contours. L'algorithme qui suit permet la détection de contour basé sur la recherche de maxima de la dérivée première (gradient).

- Mesure, en chaque point, de l'amplitude, m(x,y), et de la direction, d(x,y), du gradient,
- Détermination des seuils sur l'amplitude,
- Suppression directionnelle des non maxima locaux,
- Extraction des éléments essentiels,
- Reconstruction des contours par hystérésis.

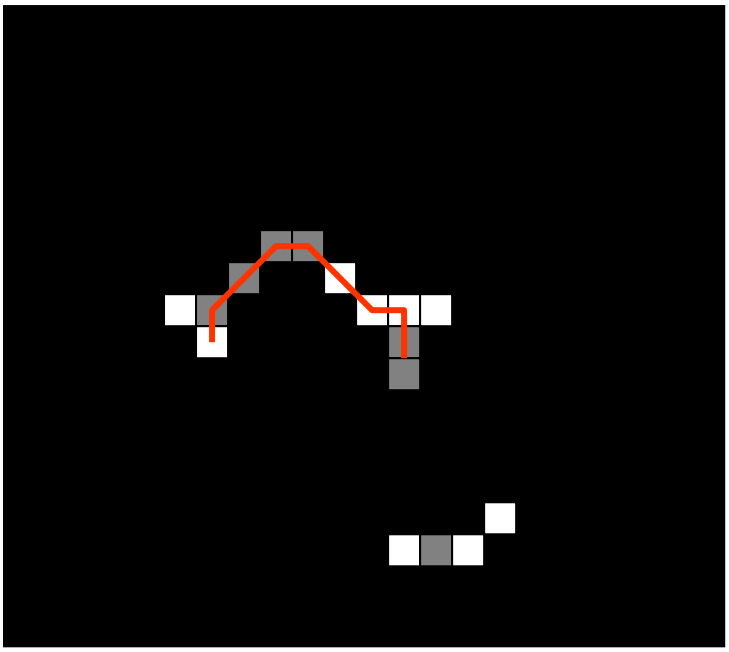


Figure 5 : chaînage des points de contour

iv. Détermination des seuils sur l'amplitude

Il est très difficile de fixer un seuil idéal pour toutes les applications d'extraction de contours. La meilleure méthode reste bien sûr la connaissance a priori du contenu de la scène. Si celle ci n'est pas accessible, on peut se référer à la technique proposée par Voorhees et Poggio. Dans cette approche, on détermine deux seuils sh (seuil haut) et sb (seuil bas) tels que :

si m(x,y) ‹ sb alors on est sûr que (x,y) n'est pas un point contour.
si m(x,y) › sh alors on est sûr que (x,y) n'est pas un point contour.
si sb ‹= m(x,y) ‹= sh alors l'appartenance à un contour sera fonction du contexte.

Les seuils sb et sh proviennent d'une modélisation de la distribution des amplitudes sous l'hypothèse suivante : soit I l'image, composée d'une structure s et perturbée par un bruit B. On pose :

$$I(x,y) = S(x,y) + B(x,y)$$

où le bruit B est un bruit additif gaussien de moyenne nulle et d'écart-type a.
Le gradient G est obtenu par ses projections sur les axes X et Y de l'image, GX et GY. On suppose que l'amplitude en (x,y) est calculée par

$$m(x,y) = \sqrt{G_X^2(x,y) + G_Y^2(x,y)}$$

Si la structure S est absente, I est une variable aléatoire et les composantes du gradient sont obtenues par convolution avec un filtre gaussien (donc une loi normale). Les composantes GX et GY sont donc des variables aléatoires de distribution normale. La distribution de m suit donc une distribution de Raleigh. On accède à cette distribution en calculant l'histogramme de l'image m des amplitudes du gradient et l'on a pour modèle :

$$H(i) = \frac{i}{2a^2} exp(-\frac{i^2}{2a^2})$$

Dans le cas d'une image réelle (S # 0), on admet que la présence du signal vient perturber cette distribution modèle pour les fortes valeurs de i uniquement (hypothèse d'un signal S dont les amplitudes sont plus fortes que celles du bruit). On approxime donc le paramètre a de la distribution H grâce à la position du maximum de l'histogramme h (en théorie, ce maximum est atteint pour z = a).
Une fois déterminée la valeur du paramètre a, on fixe (règle empirique) les deux seuils par :

$$sb = 3z \text{ et } sh = 6z$$

Le choix de 3 et 6 est lié à la notion de risque. En effet, sous ces hypothèses, Proba (amplitude du bruit > sb) < 1%. Le seuil sb permet donc de filtrer les réponses liées au bruit.

v. Suppression directionnelle des non maxima locaux

Dans la direction du gradient, la réponse à une "variation pure" permet de localiser parfaitement le lieu de cette variation, i.e. valeur maximale de l'amplitude du gradient.
Dans la pratique, il est nécessaire de faire en sorte qu'une seule réponse soit détectée. On ne conserve donc les maxima locaux dans la direction du gradient :

$$(x,y) \text{ est conservé si et seulement si}$$
$$m(x,y) > m \text{ et } m(x,y) > m$$

et sont les deux voisins immédiats de (x,y) dans la direction d=d(x,y) du gradient.
Dans un cas réel, il n'existe pas de pixel-image aux lieux et sauf si d est un multiple de 45°. On procède donc par approximation en assimilant et aux points les plus proches sur la grille. Par exemple, si d=11° alors on pose d=0°.
Il faut aussi choisir entre des inégalités strictes ou larges lors de la comparaison. Ceci n'a d'influence qu'en cas de pixels ayant strictement la même amplitude. Avec des inégalités strictes, on risque de ne sélectionner aucun point. Avec des inégalités larges, on risque de sélectionner plus d'un point. On obtient alors des contours épais. C'est cependant cette deuxième solution que l'on privilégie le plus souvent.

vi. Extraction des éléments essentiels

Les éléments essentiels sont les points pour lesquels on est sûr de l'appartenance à un contenu. Ils constitueront les germes de l'étape de reconstitution des contours. Ils sont obtenus par :

$$(x,y) : m(x,y) > sh$$

vii. Reconstitution des points par hystérésis

Partant des éléments essentiels, on extrait les chaînes (ensembles de points contours connexes). La reconstitution des points contours consiste à propager les extrémités des chaînes dans la direction locale du gradient, sous la contrainte que les points agglomérés aient une amplitude de gradient supérieure à sb.

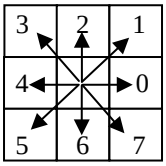
- **Variantes :**
  - Nous pouvons fixer une longueur maximale de propagation ,
  - Nous pouvons de ne pas tenir compte de la direction locale du gradient et propager une extrémité sur celui des voisins immédiats (il ne peut y en avoir plus de 3) ayant le plus fort gradient.

Cette étape est la plus longue car elle met en jeu des algorithmes plus complexes qu'un simple calcul. Sa complexité dépend fortement du nombre d'extrémités.

viii. Codage des contours

↳ Freeman

Codage le plus simple :



- ☞ Caractérise le passage d'un pixel à son successeur,
- ☞ La suite des codes locaux donne le codage du contour,
- ☞ Approximation du contour par une chaîne polygonale.

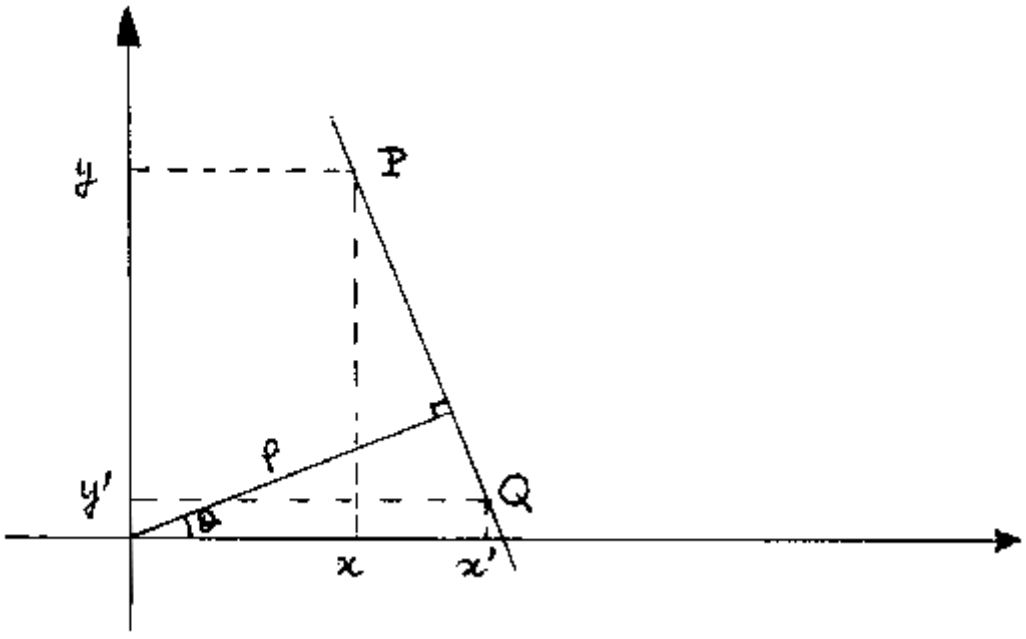
Double avantage : lissage et réduction de la quantité d'information à stocker

↳ Transformée de Hough

La transformée de Hough est un outil classique de l'analyse d'images qui permet de détecter la présence de courbes ayant une forme paramétrique (droite, conique...). Nous l'introduisons dans le cas particulier de la droite mais sa généralisation ne pose pas de problèmes mathématiques.

Considérons un pixel P(x,y) qu'un opérateur ODF (Opérateur de Détection de Frontière) a permis de détecter localement. Ce point P appartient à toutes les droites dont l'équation est, en coordonnées polaires :

$$\rho = x \cos\theta + y \sin\theta \quad (1)$$



L'équation (1) dépend du paramètre θ (ou ρ). Il y a donc une infinité de droites possibles. Cependant, nous avons une relation liant ρ à θ. Soit Q un autre point sélectionné. Grâce à lui, nous obtenons une seconde relation qui permet de ne trouver qu'un seul couple (ρ,θ) pour lequels P et Q sont sur la droite. Cette technique est appelée principe d'accumulation

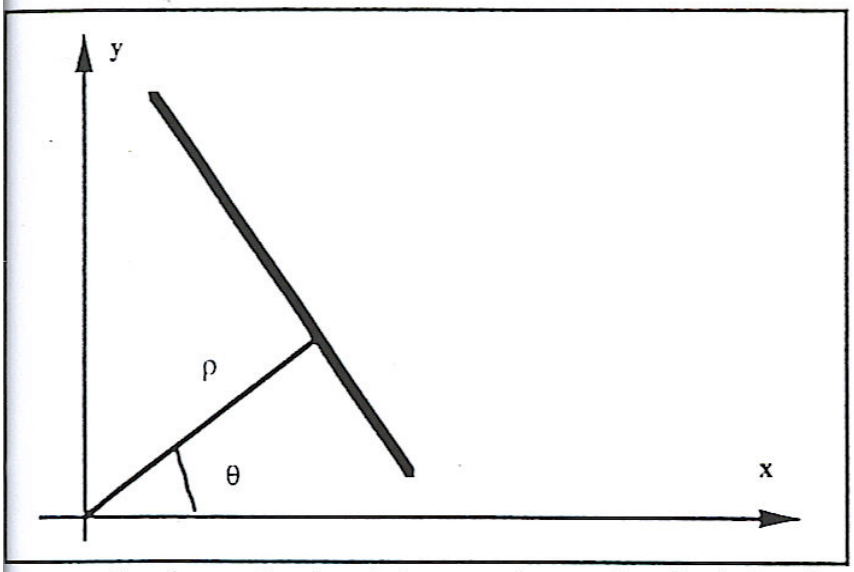
d'évidence. Pour déterminer les paramètres d'un modèle, on fait appel à un large ensemble de sources d'information (ici les pixels) et l'on procède à un recoupement de ces informations.
En pratique ce regroupement s'opère dans un accumulateur A(ρ,θ) qui est une matrice. Chaque élément de cette matrice correspond à une droite unique. Cette matrice ayant un nombre limité d'éléments, on ne peut espérer une précision infinie quand à la localisation de la droite. Soit N*N la taille de l'image I alors, les droites succeptibles d'avoir une intersection non vide avec I sont caractérisées par :

$$\rho \in [0, \frac{\sqrt{2}}{2}N]$$
$$\theta \in [0, 2\pi]$$

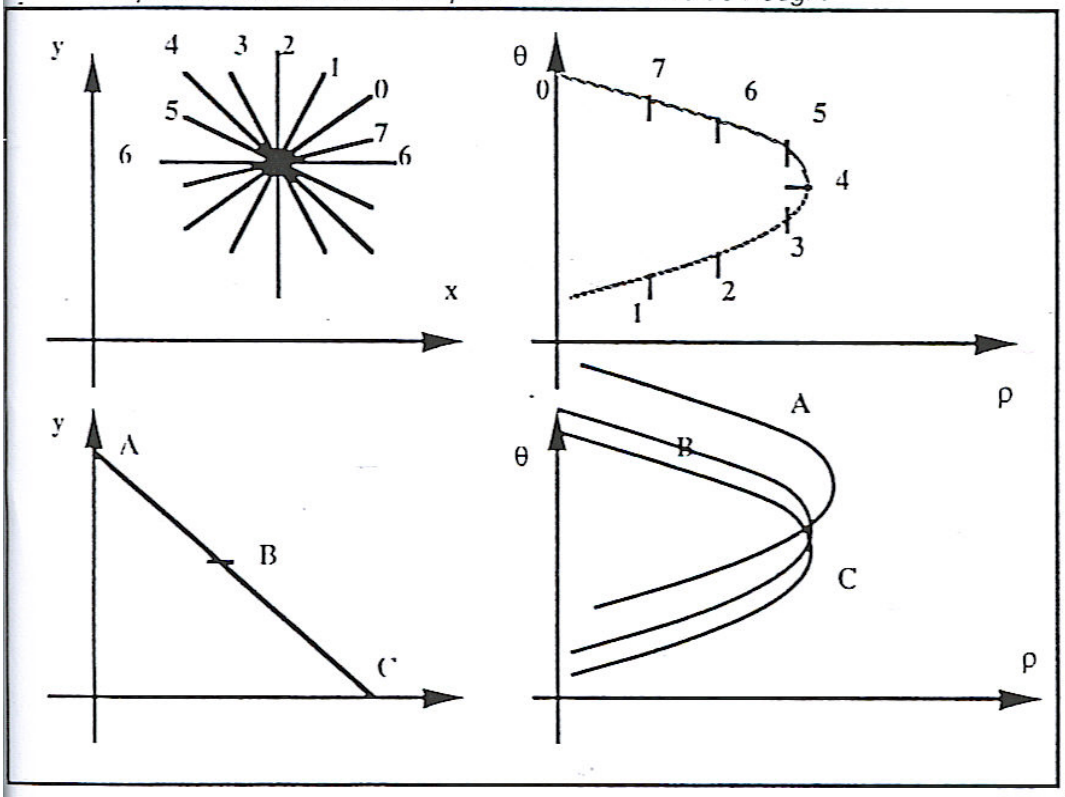
Si on utilise un accumulateur dont la taille est également N*N alors la précision sur le résultat sera au mieux :

$$(\Delta\rho\ ,\ \Delta\theta) = (\sqrt{2}\ ,\ \frac{2\pi}{N})$$

Le calcul de l'accumulateur : chaque élément (ρ,θ) de la matrice est un compteur incrémenté par les sources d'information lorsqu'elles votent pour la droite (ρ,θ). La (ou les) meilleure(s) droite(s) sera (seront) donc associée(s) aux maxima présents dans l'accumulateur.
Pour améliorer la détection des pics, il est souvent nécessaire d'employer une technique de vote non uniforme. En effet toutes les droites passant par le point (x,y) ne sont pas aussi probables. De même, tous les pixels ne sont pas équivalents. Grâce à un ODF, on peut pondérer le vote du pixel (x,y) par l'amplitude m(x,y) et orienter le choix des droites en fonction de la direction θ(x,y) (deuxième information fournie par l'ODF). On peut ainsi se contenter de considérer les droites telles que *abs (θ - θ(x,y))* < seuil (paramètre fixé a priori).
Très simple dans son principe, cette technique requiert un volume important de mémoire. De plus la présence de bruit dans l'accumulateur rend difficile l'extraction des meilleures droites. Enfin, cette technique ne fournit aucune information quant à la localisation de la droite dans l'image (une droite n'existe souvent que sur une partie de l'image).
Pour palier ces défauts, une nouvelle approche de la transformée de Hough a été développée en utilisant une structure pyramidale. Cette approche permet un calcul et une extraction rapide des meilleures droites.



Représentation des droites pour la transformée de Hough.



Recherche d'une droite par transformée de Hough.

Figure 6: chaînage de Transformé de Hough **[2]**

- **Algorithme**

- Quantifier l'espace des paramètres (ρ,θ) entre deux minima et deux maxima.
- Initialiser un tableau à deux entrées à 0.
- Pour chaque point de contour (xi,yi), incrémenter l'élément C(ρk,θl) du tableau tel que :

$$C(\rho_k,\theta_l) = C(\rho_k,\theta_l) + 1 \text{ si } \rho_k = x_i\cos\theta + y_i\sin\theta \text{ avec } \theta = \theta_l$$

- Les maxima locaux dans le tableau correspondent aux segments de droite.

Les valeurs des éléments du tableau indiquent le nombre de points alignés. Cette recherche à deux dimension peut être réduite à une dimension si l'orientation du gradient θ_i est connue pour chaque (xi,yi). En dérivant l'expression de la droite, on obtient :

$$dx/dy = \tan(\theta + \pi/2)$$

C (ρ,θ) ne doit plus être évalué que pour θ égal à ($\pi/2 - \theta$).
La transformée de Hough peut être utilisée pour détecter de courbes quelconques pourvu qu'on puisse les paramétrer : cela peut augmenter la taille du tableau. **[2]**

b. Segmentation en régions

La segmentation d'une image vis à vis d'un critère d'homogénéité H peut s'exprimer de la manière suivante :
La segmentation d'une image I en regard du critère H est une partition de l'image I en régions homogènes X1, ..., Xn telles que :

- Unioni=1,..., n Xi = I ,
- Pour tout i, Xi est connexe,
- Pour tout i, H[Xi] est vrai,
- Pour tout couple (Xi,Xj) de régions voisines, H[Xi,Xj] est faux.

Cette définition conduit à deux remarques très importantes. Tout d'abord, une segmentation dépend du critère employé. Le choix du critère est donc primordial. Ensuite, la décomposition obtenue n'est pas unique. Pour un critère donné, il existe plusieurs solutions.
Nous commencerons par évoquer un cas particulier de la segmentatione régions : le seuillage, avant de passer aux méthodes descendantes puis aux méthodes ascendantes **[2]**

i. Le seuillage

Le seuillage a pour objectif de segmenter une image en plusieurs classes en n'utilisant que l'histogramme. On suppose donc que l'information associée à l'image permet à elle seule la segmentation, i.e. qu'une classe est caractérisée par sa distribution de niveaux de gris. A chaque pic de l'histogramme est associée une classe.

Il existe de très nombreuses méthodes de seuillage d'un histogramme. La plupart de ces méthodes s'appliquent correctement si l'histogramme contient réellement des pics séparés. De plus, ces méthodes ont très souvent été développées pour traiter le cas particulier de la segmentation en deux classes (i.e. passage à une image binaire) et leur généralité face aux cas multi-classes n'est que très rarement garantie.

Détection de vallées

Cette technique est la plus intuitive. On suppose que chaque classe correspond à une gamme distincte de niveaux de gris. L'histogramme est alors m-modal. La position des minima de l'histogramme H permet de fixer les (m-1) seuils nécessaires pour séparer les m classes.
En termes mathématiques, les seuils s_i sont obtenus par

$$H(s_i) = Min\ [H(k)]\ pour\ k\ in]m_i,m_{i+1}[$$

où m_i et m_{i+1} sont les valeurs moyennes (ou les modes) de l'intensité lumineuse dans les classes C_i et C_{i+1}. Malgré le développement de techniques robustes visant à faciliter la détection des vallées, cette méthode, bien que simple, est très peu appliquée car les histogrammes traités sont le plus souvent bruités et unimodaux.
Sur l'exemple de la figure 8, deux seuils on été choisis correspondant aux deux premières vallées : 60 et 110. Si certains objets sont bien détectés, il aurait fallu utiliser plus de seuils dans les niveaux de gris clairs pour mieux mettre en évidence les différentes parties de cette image. C'est le principal inconvénient de beaucoup d'approche du seuillage : combien y-a-t-il a priori de classes à séparer ?

Minimisation de variance

La répartition des pixels en N classes est un problème classique de classification. Le choix des seuils s_i permet de détecter m classes auxquels on peut associer taille (t_i), moyenne (m_i) et variance V_i par :

$$t_i = \sum_{Dj} \{H(j)\}$$

$$m_i = \sum_{Dj} \{j \,.\, H(j) / t_i\}$$

$$V_i = \sum_{Dj} \{(j - m_i)^2 \,.\, H(j) / t_i\}$$

où H est l'histogramme normalisé (son intégrale est ramenée à l'unité) et $D_j = [s_{i-1},s_i[$ est la gamme de niveaux de gris correspondant à la classe C_i (par hypothèse, $s_0 = 0$).
A partir de ces indicateurs statistiques, on peut construire la variance intraclasse totale W par :

$$W = \sum_i \{t_i.V_i\}$$

Le meilleur seuillage dans cette approche correspond à une minimisation de la variance intraclasse (méthode de Fisher). Cette technique est difficilement applicable lorsque le nombre de classes est élevé. En effet, il faut tester exhaustivement tous les (N-1)-uples ($s_1,...,s_{N-1}$) possibles. De plus, il faut que chaque classe ait une taille significative en nombre de niveaux de gris pour que les indicateurs statistiques aient un sens. Dans le cas de la binarisation (N=2), cette méthode est performante.
Plus récemment, Otsu a proposé de réaliser une maximisation de la variance inter-classe qui, dans le cas de la binarisation, s'exprime par

$$B = t_0 \,.\, t_1\ (m_0 - m_1)^2$$

ce qui est rigoureusement équivalent puisque l'on a la relation :

$$W + B = cste$$

Cependant, la méthode d'Otsu est plus intéressante d'un point de vue calculatoire car elle ne nécessite pas de calcul de variances.
D'autres critères statistiques sont utilisables : test de *Student*, distance de Fisher (pour laquelle il existe un algorithme optimisé), ...
Sur l'exemple de la figure 7, un seuil a été déterminé par l'algorithme optimal de *fisher*. Le seuil trouvé est 127. Compte tenu du fait que l'on n'utilise qu'un seul seuil, toutes les composantes de l'image ne peuvent être séparées.

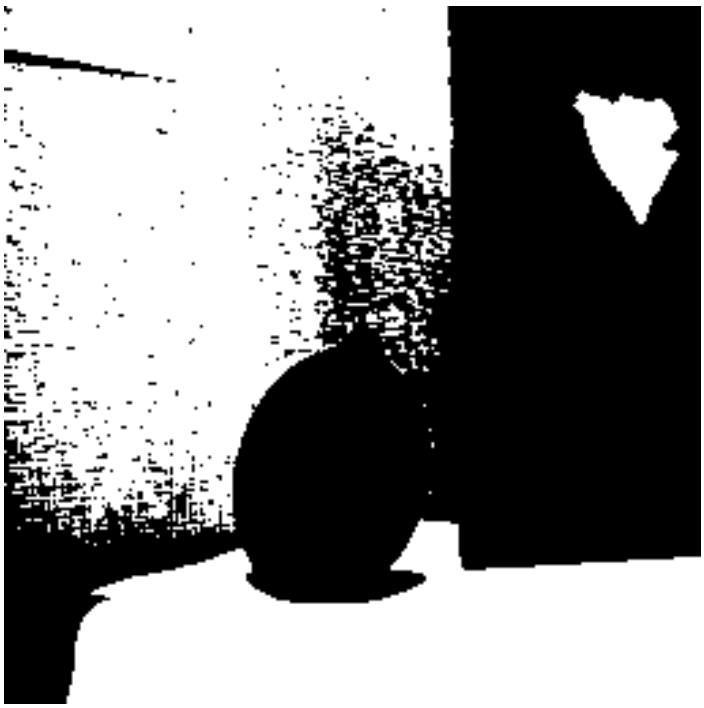


Figure 7 : seuillage part minimisation de variance

Seuillage entropique

Le seuillage entropique est une technique dérivée de la théorie de l'information. Les seuils sont déterminés de manière à maximiser l'entropie E résultant du découpage de l'histogramme H en plusieurs classes. En effet, l'entropie mesure la quantité d'information portée par un groupe. Pour un nombre de seuils fixe, on cherche à ce que les classes résultantes portent le maximum d'information.

L'entropie totale est calculée de la manière suivante:

$$E = \sum_i E(C_i)$$

où C_i désigne la classe No i,

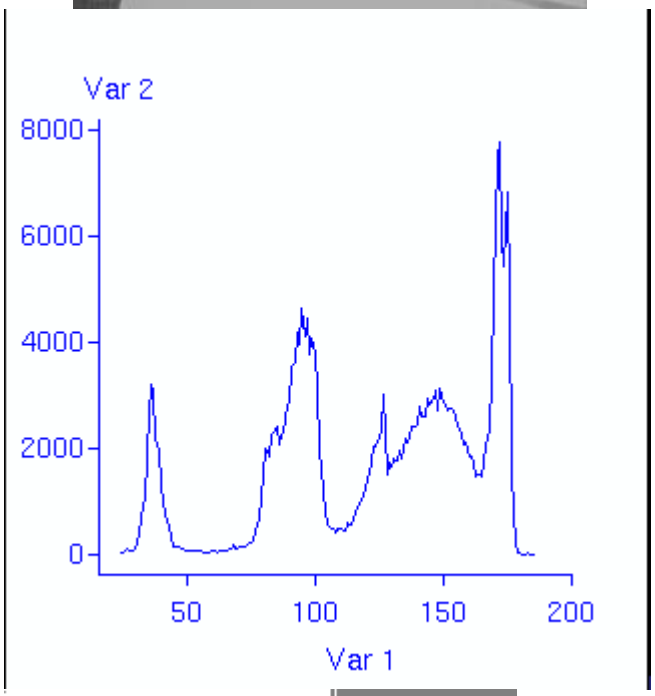
$$E(C_i) = - \sum_{Di} \{ p_j \, . \, log_2(p_j) \}$$

où D_i est l'ensemble des niveaux de gris j associés à la classe C_i et p_j la probabilité *a postériori* du niveau de gris j, estimée par

$$p_j = H(j) / taille\_image$$

La notion d'entropie n'est pas liée à une caractéristique visuelle. C'est pourquoi l'image résultat parait le plus souvent de moins bonne qualité (du point de vue de l'oeil humain) qu'une image obtenue par une des techniques présentées dans ce chapitre.

Image initiale



Histogramme

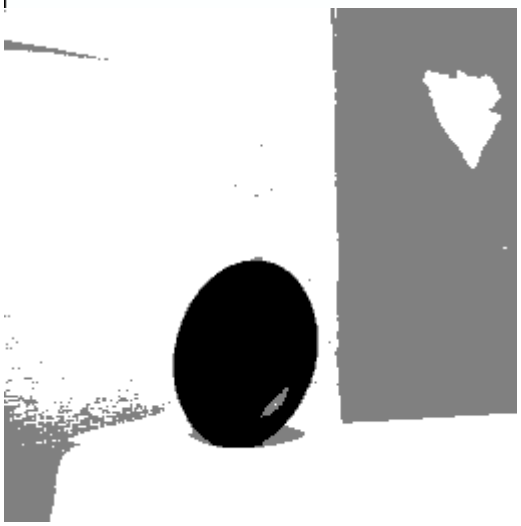


Figure 8: seuillage part détection de vallées

Erreur de seuillage

La minimisation de l'erreur de seuillage nécessite de faire des hypothèses quant à la nature des distributions de niveaux de gris correspondant aux classes C_i. Le plus souvent, on utilise le modèle gaussien. On construit donc un histogramme estimé T par :

$$T(x) = \sum_{i=1,N} A_i \, . \, N_i(x)$$

où A_i est l'amplitude de la distribution et N_i une distribution gaussienne de paramètres m_i (moyenne) et s_i (écart-type) :

$$N_i(x) = 1/s_i \, . \, e^{-½((x-m_i)/s_i)^2}$$

De plus, il est nécessaire de faire des hypothèses de séparabilité des distributions : Pour tout x dans $[m_i,m_{i+1}]$, $N_j(x)$ proche de 0 si j n'est pas égal à i ou i+1.

A partir de ce modèle, on peut raisonner dans le domaine du continu et exprimer les erreurs de classification en termes de probabilités pour un (N-1)-uple de seuils (S_1, ..., S_{N-1}). Dans le domaine du continu, on obtient les relations suivantes :

- Probabilité d'affecter un point de la classe C_i à la classe $C_{i+1} = E_{i,i+1} = Int_{Si,+inf} \{ N_i(z) \, dz \}$,
- Probabilité d'affecter un point de la classe C_{i+1} à la classe $C_i = E_{i+1,i} = Int_{-inf,Si} \{ N_{i+1}(z) \, dz \}$.

L'erreur de seuillage associée au choix du seuil S_i entre les classes C_i et C_{i+1} est donnée par :

$$A_i \, E_{i,i+1} + A_{i+1} \, . \, E_{i+1,i}$$

Cette erreur correspond à l'aire du domaine où les deux distributions se recouvrent. La valeur choisie pour le seuil est obtenue par minimisation de cette erreur de seuillage et est la solution de la forme quadratique:

$$s^2_{i+1} \, (S_i - m_i)^2 - s^2_i \, (S_{i+1} - m_{i+1})^2 = 2 \, s^2_i \, s^2_{i+1} \, ln[s^2_{i+1} \, A_i \, / \, (s^2_i \, A_{i+1})]$$

L'avantage de cette technique est la détermination rapide et indépendante de tous les seuils. Par contre, la qualité du résultat dépend de la qualité du modèle. Le problème d'ajustement peut être résolu par un algorithme de type Newton-Raphson. Cependant, bien qu'efficace, cette méthode est très coûteuse en temps de calcul. Il est possible d'utiliser des techniques d'estimation itératives reposant sur une hiérarchie des amplitudes. Si les distributions sont vraiment séparées, la recherche des pics facilite l'estimation des paramètres. Plus généralement, la minimisation de l'erreur de seuillage conduit à la résolution de l'équation : $A_i \, N_i(S_i) = A_{i+1} \, N_{i+1}(S_i)$. Cette équation est valable quelle que soit la nature des distributions N_i. Si celles-ci sont connues théoriquement, ou si l'on peut les estimer à partir de H, on pourra éventuellement, grâce à des méthodes numériques de recherche de racines d'une équation, déterminer la valeur optimale du seuil S_i. Cette approche est donc plus générale que celles évoquées dans ce chapitre et les résultats obtenus sont de bonne qualité. De plus, il est facile grâce au modèle de prendre en compte plusieurs classes (N > 2).

Méthode du pourcentage

Dans certains cas, une connaissance *a priori* de la scène, si celle-ci est simple, permet de choisir les seuils. En particulier, si l'image est constituée de deux entités, la connaissance de la fraction surfacique F de l'une des deux phases peut être reliée au seuil s par :

$$|d(s) - F| = Min \, | \, d(k) - F \, |$$ *où d(k) représente l'intégrale normalisée de l'histogramme entre les bornes 1 et k.*

Cette technique peut être appliquée au seuillage de pages de caractères si la densité dans une page est connue. Cependant, elle ne s'applique que dans des applications particulières et la

généralisation au cas de m classes est difficilement envisageable. Par contre, si la connaissance de F est rarement suffisante, elle peut faciliter la modélisation de l'histogramme.

↳ Maximisation du contraste

Bien que déterminant des seuils, cette technique ne fait pas appel à l'histogramme mais utilise directement la répartition spatiale des niveaux de gris g(i) dans l'image. Elle est ici présentée dans le cas de la binarisation. L'objectif est de trouver le seuil qui va introduire le maximum de contraste dans l'image résultat.
Soit k un seuil donné. On construit l'ensemble K par :

$$K(k) = \{(a,b) / g(a) <= k <= g(b)\}$$

où a et b sont deux pixels voisins dans l'image (pour un ordre de voisinage à fixer). A cet ensemble, on associe la mesure de contraste suivante :

$$C(k) = SumK(k) \{ Min (|g(a) - k|, |g(b) - k|) \}$$

La valeur retenue pour le seuil sera celle qui maximisera le ratio C(k)/Card[K(k)]. Cette méthode en apparence complexe est très intéressante car elle introduit la notion de contexte dans l'étude des couples de pixels voisins. L'histogramme des niveaux de gris n'est plus la seule source d'information. Cette idée sert de base aux méthodes contextuelles.
Les méthodes de segmentation basées sur le choix d'un ou plusieurs seuils sont aujourd'hui utilisées dans des applications très particulières. On les retrouvera en contrôle qualité où la maîtrise du milieu ambiant et la simplicité des scènes permet d'avoir de très bonnes conditions d'éclairage. On peut ainsi assurer au mieux l'hypothèse de séparabilité des phases nécessaire au bon fonctionnement de ce type de méthodes.

C'est dans ce type d'applications que l'on exploite le mieux les deux grands avantages du seuillage : la simplicité et la rapidité. Ces deux caractéristiques sont liées. En effet, la simplicité des algorithmes permet souvent la réalisation d'opérateurs câblés permettant la rapidité nécessaire dans le contexte du temps réel qui est le contexte habituel des applications industrielles.

Bien que nécessitant des paramètres, les techniques de seuillage peuvent fonctionner en mode autonome. Il est possible d'affiner un seuil par comparaison entre l'image initiale et l'image seuillée ou bien en rajoutant des contraintes sur la qualité (à définir) de l'image seuillée. Cette mesure peut être interprétée en terme d'évaluation de l'erreur de classification.

L'histogramme étant une fonction trop globale, on peut envisager un seuillage qui ne soit pas fixe. Pour cela, l'image est décomposée en sous-images. Sur chacune d'elles, une technique classique de seuillage est appliquée. Cependant, ce processus risque d'introduire des discontinuités. De plus, si les sous-images sont trop petites, les histogrammes ne contiendront pas assez de valeurs pour être statistiquement exploitables. Cette technique est à réserver à des configurations particulières (dérive lumineuse par exemple).

Cependant, ces méthodes ne sont pas suffisantes pour la plupart des applications où la complexité de l'information contenue dans l'image ne peut être résumée par l'histogramme des niveaux de gris sans trop de pertes d'information.

ii. Quelques critères d'homogénéité :

Cette liste n'est pas exhaustive, elle reprend cependant les critères les plus courants.

- **Contraste**

$H[X_i] = Variance(X_i) < seuil$

- **Similarité statistique**

Ce critère s'utilise principalement dans la phase d'analyse mais aussi pour tester la réunion de deux régions. Chaque région est assimilée à une population statistique dont on peut tracer la distribution de l'information portée. La similarité entre les deux régions est donc ramenée à une similarité entre deux distributions que l'on peut évaluer avec un test statistique comme par exemple le test de Kolmogorov-Smirnov (Muerle, Alen, 1968).
De nombreux paramètres sont utilisables pour décrire une région (radiométrie, forme, texture, couleur°. Nous reviendrons sur les paramètres de texture dans le chapitre 10, Analyse.

- **Frontière**

La validité de la réunion de deux régions dépend de leur frontière commune. Celle-ci doit être petite par rapport aux périmètres des deux régions. De plus, on ajoute souvent une contrainte de faible variation de l'information portée au voisinage de la frontière commune (on recherche ainsi une continuité dans la distribution spatiale de l'information portée) (Brice, Fennema, 1970).

- **Comparaison de modèles**

Chaque région est modélisée par une fonction paramétrique (le plus souvent des fonctions polynomiales telles que plan, hyper-quadrique, ...). La similarité entre deux régions est alors estimée par une distance entre les paramètres des modèles (Pavlidis, 1972).

- **Qualité de la segmentation**

Si l'on possède une référence permettant de construire un indicateur global de qualité de segmentation, cette mesure peut servir localement pour savoir si une fusion accroit significativement cette mesure (Feldman, Yakimovsky, 1974). Une telle mesure peut être la probabilité de bonne interprétation des régions et des contours.

- **Qualité de fusion :**

On peut dire qu'une région est homogène si aucune fusion locale ne peut l'améliorer (Hong, Rosenfeld, 1984).

- **Enchaînement de critères :**

Il est toujours possible de combiner les différents critères évoqués plus haut. Cette approche propose en particulier de réaliser une segmentation hiérarchique où à chaque itération correspond un critère donné. L'ordre des critères utilisé est très important. Il est nécessaire d'employer en premier des critères n'incluant que peu d'information. On peut ainsi utiliser la séquence suivante :

Soit $R = Union\ (X_k, X_l)$ et $g(P)$ l'information portée par le pixel M de l'image.
$C_1 : Max_R[g(P)] - Min_R[g(P)] < T_1$
$C_2 : | Moy_{Xk}[g(P)] - Moy_{Xl}[g(P)] | < T_2$
$C_3 : Var_R[g(P)] < T_3$
$C_4 : Moy_{Fk,l}|g(P_k) - g(P_l)| < T_4$

où F est l'ensemble des couples (P_k,P_l) dans X_k x X_l définissant la frontière commune entre ces deux régions.

$C_5 : a.C_3 + b.C_4$ *pour tout* X_l *tel que* $Card[X_l] < T_5$.

Tous les paramètres T_i désignent bien entendu des seuils à déterminer en fonction des caractéristiques générales de l'image à segmenter.

- **Les structures de contrôle :**

Les critères que nous venons d'évoquer peuvent être mis en œuvre de différentes manières :

- **Séquentiel :** On sélectionne la première région située en haut et à gauche de l'image. On teste ensuite une agglomération possible avec une de ses régions voisines dans le sens des aiguilles d'une montre. Si une agglomération est possible, elle est immédiatement effectuée. On passe alors à la première région disponible en conservant le même sens de parcours de l'image. Cette approche permet des exécutions rapides. Cependant, la solution obtenue est très fortement dépendante du sens de balayage,
- **Graphe :** Chaque région teste, en parallèle, son agglomération possible avec chacun de ses voisins. On peut donc construire un graphe valué (grâce à une mesure de la qualité de la fusion). La segmentation est obtenue en décomposant ce graphe. On recherche en particulier les cliques (i.e. graphe partiel complet). La valuation des arcs du graphe permet le plus souvent de réduire les effets de contradictions internes. L'algorithme de la pyramide stochastique proposé par Peter Meer permet entre autre d'assurer qu'une région ne puisse fusionner qu'avec au plus une autre région au cours d'une itération. Très séduisante, l'approche par graphe pose encore de nombreux problèmes d'implantation (coût mémoire) dans le cas où le nombre de régions est vaste (en particulier lorsque l'on veut initier le processus avec 1 région = 1 pixel).

IV. Reconnaissance d'objets

Le terme "reconnaissance d'objet" désigne l'ensemble des deux processus suivants :

- **Appariement** : établir une correspondance terme à terme entre des caractéristiques objets et des caractéristiques données,
- **Localisation** : déterminer la position et l'orientation de l'objet dans le repère des données (du capteur),
- **Complexité :** La tache qui consiste à établir les appariements est difficile : soit par exemple le cas ou les données et l'objet sont décrite par des caractéristiques du même type et soient **O** l'ensemble de caractéristiques objet et **D** l'ensemble de caractéristiques données

  *Si Card(**O**) = Card(**D**) = n dans ce cas le nombre d'appariement possible est :*

  $$n!$$

  Si Card(O) = n et Card(D) = m avec n≤m dans ce cas le nombre d'appariement possible est :

  $$C_m^n \quad n!$$

- **Appariement et isomorphisme de graphes :** Une première approche possible consiste à traiter le problème d'appariement comme un problème d'isomorphisme de graphes. Un objet peut être décrit comme un ensemble de caractéristiques et des relations entre ces caractéristiques. L'objet peut alors être décrit par un graphe dans lequel un nœud représente une caractéristique et une arête représente une relation entre deux caractéristiques. Les données peuvent être décrites par un graphe de même

nature. Le problème d'appariement de caractéristiques devient alors un problème d'appariement de graphes, soit un problème d'isomorphisme de graphes. **[1]**

- **Appariement et prédiction/vérification :** Cette deuxième approche consiste à résoudre les problèmes d'appariement et de localisation simultanément dans le cadre du paradigme prédiction et vérification. Ce paradigme consiste essentiellement à faire une hypothèse quant aux paramètres de position et d'orientation de l'objet par rapport au capteur et à vérifier que des appariements sont compatibles avec ces paramètres. Dans ses lignes les plus générales ce paradigme peut être décrit comme suit :

  Prédiction : établir quelques appariements objet/données,
  - Calculer les paramètres de la transformation objet/données sur la base de ces appariements ainsi que les intervalles de confiance associés à ces paramètres.
  - Appliquer cette transformation à l'ensemble des caractéristiques objets de façon à pouvoir prédire lesquelles parmi ces caractéristiques sont susceptibles d'être vues par le capteur et quelle est leur position et orientation dans le repère du capteur,

  Vérification : pour chaque caractéristique objet ainsi prédite,
  - sélectionner les caractéristiques données les plus proches et les plus ressemblantes.
  - Former ainsi des nouveaux appariements objet/données tout en rejetant les appariements qui s'avèrent incorrects.
  - Fusionner chaque nouvel appariement avec les appariements précédents afin de mettre à jour les paramètres de la transformation objet/données ainsi que les intervalles de confiance associés.

CHAPITRE 2

PRESENTATION DE L'OPENGL

Chapitre II : Présentation de l'OpenGL

INTRODUCTION

OpenGL signifie "*Open Graphics Library*". C'est une bibliothèque (spécification) qui définit une API (Interface de programmation) multi plateforme pour la conception d'application générant des images 2D et 3D. Et qui permet de faire du rendu en temps réel. C'est notamment pour cela qu'il est fortement utilisé pour les jeux vidéo et les applications.

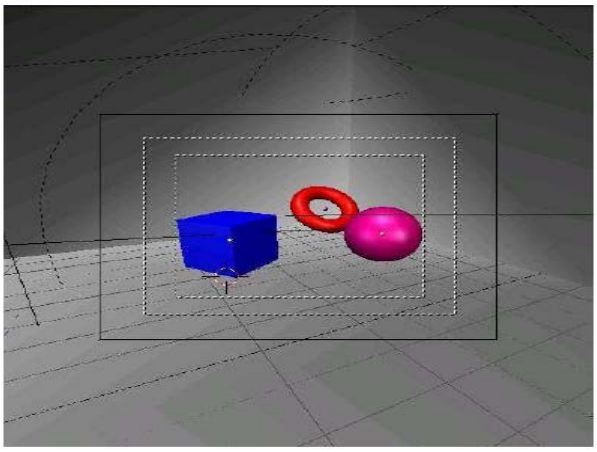


Figure 9: vue d'OpenGL

Fruit d'ARB (*Architecture Review Board working group*), consortium indépendant définissant les changements de spécifications pour OpenGL, cette spécification est indépendante de toute machine graphique et de tout système d'exploitation. Donc quelle que soit la plateforme de travail OpenGL aura toujours la même interface. C'est-à-dire que les appels de fonctions seront toujours identiques.
OpenGL fut créé initialement par SGI (Silicon Graphics, Inc), constructeur de station de travail pour les domaines de l'infographie de la 3D et du traitement vidéo.
La spécification OpenGL est actuellement surveillée par l'OpenGL Architecture Review Board (ARB), formé en 1992. L'ARB fait partie aujourd'hui du groupe Khronos.
L'ARB se compose d'entreprises ayant un profond intérêt pour la création d'une API cohérente et largement disponible. **[3]**

I.Fonctionnalités d'OpenGL

OpenGL permet de représenter à l'écran des scènes tridimensionnelles réalistes. Il est résolument orienté vers l'interaction 'temps réel', c'est à dire qu'un programme conçu avec OpenGL permet en général à l'utilisateur d'agir par l'intermédiaires de périphériques d'entrée (clavier, souris, trackball...) sur la scène (déplacement de caméra, changement de focale, déplacement des objets....), et d'en visualiser immédiatement l'effet. Ainsi, lorsque déplacez un objet dans la scène, avec la souris par exemple, le programme recalcule la nouvelle image et l'affiche à l'écran. On comprend aisément que pour que le jeu vidéo soit 'fluide', c'est-à-dire que les déplacements du joueur ne paraissent pas trop saccadés, le programme doit être capable de calculer et d'afficher une image extrêmement rapidement (moins d'un dixième de secondes). Dans le cas

d'un jeu vidéo on peut considérer que si l'image n'est pas recalculée une quinzaine de fois par secondes, le jeu ne sera pas fluide. Bien que non orienté photoréalisme, OpenGL permettent d'obtenir des rendus tout à fait satisfaisants. Voici une liste non exhaustive des principales fonctionnalités d'OpenGL.

L'OpenGL comprend les fonctionnalités suivantes :

- Affichage de maillages polygonaux,
- Rendu par projection orthogonale ou perspective,
- Placage de texture,
- Simulation d'éclairages réalistes,
- Ombre portées,
- Insertion d'objets bitmaps en avant plan (images, polices),
- Effet de brouillard, de flou,
- Gestion des courbes et surfaces gauches (Splines, Nurbs...),
- ...

1. Bibliothèques d'OpenGL

a. OpenGL Library (GL)

- Librairie standard proposant les fonctions de base pour l'affichage en OpenGL,
- Pas de fonction certaines transformations géométriques,
- triangulation des polygones,
- Rendu pour la construction d'une interface utilisateur (fenêtres, souris, clavier, ...).

Préfixe des fonctions: gl

b. OpenGL Utility Library (GLU)

Librairie standard proposant des commandes bas-niveau écrites en GL:

- des surfaces paramétriques et quadriques
- ...

Préfixe des fonctions: glu

c. OpenGL extension to X-Windows (GLX)

Utilisation d'OpenGL en association avec X Windows pour l'interface utilisateur Préfixe des fonctions: glX

d. Auxiliary Library

Bibliothèque écrite pour rendre simple la programmation de petites applications dans le cadre d'interfaces graphiques interactives simples:

- gestion d'une fenêtre d'affichage
- gestion de la souris
- gestion du clavier
- ...

Préfixe des fonctions: aux

e. OpenGL Tk Library (GLTk)

Équivalent à Aux mais avec l'utilisation de TclTk pour l'interface graphique. Préfixe des fonctions: tk.

f. OpenGL Utility Toolkit (GLUT)

Équivalent à Aux, Interface utilisateur programmable plus élaborée (multifenêtrage, menus popup) → plus grande complexité. Préfixe des fonctions: glut

- **L'Auxiliary Library: Aux**

Facilite la programmation d'OpenGL aux moyen d'un jeu de fonctions d'interfaçage vis à vis du système d'exploitation de l'ordinateur (Interface graphique).

- **L'Utility Toolkit: GLUT**

Programmation presque identique à celle réalisée au moyen de Aux.
Fonctionnalités supplémentaires:

- gestion des menus,
- gestion des environnements multifenêtrés,
- existence de polices de caractères bitmap et vectorielles intégrées,
- gestion de périphériques d'entrée supplémentaires
- ...

2. Syntaxe de la librairie GL

Les fonctions de la librairie GL commencent par le préfixe gl.
Les constantes (#define ou enum) sont données en majuscule et commencent par le préfixe GL_.
Certaines instructions finissent par un suffixe.
Exemple: glVertex2f
Ce suffixe indique le nombre et le type des arguments de la fonction.

-	*Type*	*Type C*	*Type OpenGL*
b	entier 8 bits	signed char	GLbyte
s	entier 16 bits	short	GLshort
i	entier 32 bits	long, int	GLint, GLsizei
f	réel 32 bits	float	GLfloat, GLclampf
d	réel 64 bits	double	GLdouble, GLclampd
ub	entier non signé 8 bits	unsigned char	GLubyte, GLboolean
us	entier non signé 16 bits	unsigned short	GLushort
ui	entier non signé 32 bits	unsigned long	GLuint, GLenum, GLbitfield

3. Processus de visualisation en OpenGL

Quatre transformations successives utilisées au cours du processus de création d'une image:

1. **Transformation de modélisation (Model) :** Permet de créer la scène à afficher par création, placement et orientation des objets qui la composent.

2e. **Transformation de visualisation (View) :** Permet de fixer la position et l'orientation de la caméra de visualisation.
3e. **Transformation de projection (Projection) :** Permet de fixer les caractéristiques optiques de la caméra de visualisation (type de projection, ouverture, ...).
4e. **Transformation d'affichage (Viewport) :** Permet de fixer la taille et la position de l'image sur la fenêtre d'affichage.

Les transformations *de visualisation* et de *modélisation* n'en forment qu'une pour OpenGL (transformation modelview). Cette transformation fait partie de l'environnement OpenGL.
La transformation de projection existe en tant que telle dans OpenGL, et fait elle aussi partie de l'environnement OpenGL.
Chacune de ces deux transformations peut être modifiée indépendamment de l'autre ce qui permet d'obtenir une indépendance des scènes modélisées vis à vis des caractéristiques de la " caméra " de visualisation.
La transformation d'affichage est elle aussi paramétrable en OpenGL. **[3]**

4. Transformations géométriques

a. Transformations d'utilité générale

- ***void glLoadIdentity(void) :*** Affecte la transformation courante avec la transformation identité,
- ***void glLoadMatrix {f d} (const TYPE *m) :*** Affecte la transformation courante avec la transformation caractérisée mathématiquement par la matrice m (16 valeurs en coordonnées homogènes),
- ***void glMultMatrix {f d} (const TYPE *m) :*** Compose la transformation courante par la transformation de matrice m (16 valeurs en coordonnées homogènes),
- ***void glTranslate {f d} (TYPE x,TYPE y,TYPE z) :*** Compose la transformation courante par la translation de vecteur (x,y,z). Très utilisé en modélisation,
- ***void glRotate{f d}(TYPE a,TYPE dx,TYPE dy,TYPE dz) :*** Compose la transformation courante par la rotation d'angle a degrés autour de l'axe (dx,dy,dz) passant par l'origine. Très utilisé en modélisation,
- ***void glScale {f d} (TYPE rx,TYPE ry,TYPE rz) :*** Compose la matrice courante par la transformation composition des affinités d'axe x, y et z, de rapports respectifs rx, ry et rz selon ces axes. Très utilisé en modélisation.

b. Transformation spécifique à la visualisation (view)

- ***void gluLookAt(GLdouble ex, GLdouble ey, GLdouble ez, GLdouble cx, GLdouble cy, GLdouble cz, GLdouble upx, GLdouble upy, GLdouble upz) :*** Compose la transformation courante (généralement MODELVIEW) par la transformation donnant un point de vue depuis (ex,ey,ez) avec une direction de visualisation passant par (cx,cy,cz). (upx,upy,upz) indique quelle est la direction du repère courant (fréquemment le repère global) qui devient la direction y (0.0, 1.0, 0.0) dans le repère écran. gluLookAt est une fonction de la bibliothèque GLU.
    - Dans la pratique, gluLookAt place et oriente la scène devant la caméra de telle manière que la caméra semble placée et orientée selon les paramètres d'entête.
    - gluLookAt doit être exécuté en mode MODELVIEW et doit être placé avant la création de la scène.

c. Transformations de projection

- ***void glFrustum(GLdouble g, GLdouble d, GLdouble b, GLdouble h, GLdouble cmin, GLdouble cmax) :*** Compose la transformation courante par la transformation de projection en perspective de volume de visualisation défini par la pyramide tronquée de sommet l'origine O, orientée selon l'axe -z et de plan supérieur défini par la diagonale (g,b,-cmin) (d,h,-cmin).
- ***void gluPerspective(GLdouble foc, GLdouble ratio, GLdouble cmin, GLdouble cmax) :*** Compose la transformation courante par la transformation de projection en perspective de volume de visualisation défini par la pyramide tronquée de sommet l'origine O, orientée selon l'axe -z, possédant l'angle foc comme angle d'ouverture verticale, l'apect-ratio ratio (rapport largeur/hauteur) et les plans de clipping proche et éloignés -cmin et -cmax.
- ***void glOrtho(GLdouble g, GLdouble d, GLdouble b, GLdouble h, GLdouble cmin, GLdouble cmax) :*** Compose la transformation courante par la transformation de projection orthographique selon l'axe -z et définie par le volume de visualisation parallélépipèdique (g,d,b,h,-cmin,-cmax). cmin et cmax peuvent être positifs ou négatifs et mais doivent respecter cmin < cmax.

Exemples de projections

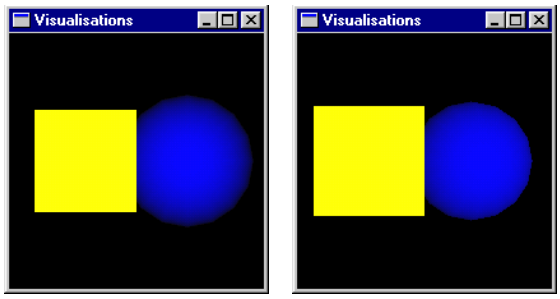


Figure 10: Projections orthogonales et perspectives (sans angle, pas de différence)

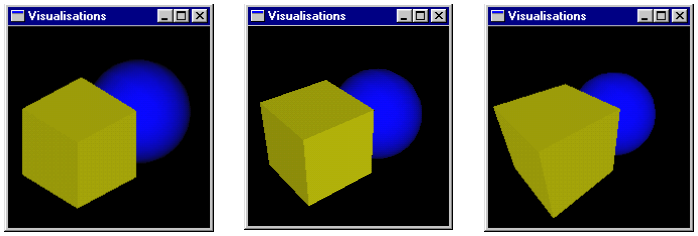


Figure 11: Projection en perspective avec rapprochement de scène

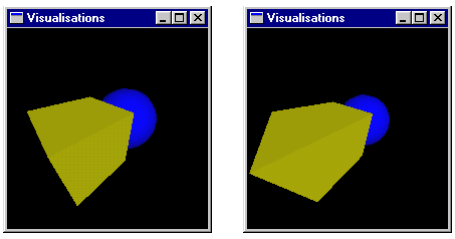


Figure 12 : Projection en perspective: très grosses déformations

5. Les listes d'affichage

Une liste d'affichage est une suite de commandes OpenGL stockées pour une utilisation future.
Il s'agit d'un mécanisme pour stocker des commandes OpenGL pour une exécution ultérieure, qui est utile pour dessiner rapidement un même objet à différents endroits.
Quand une liste d'affichage est invoquée, les commandes qu'elle contient sont exécutées dans l'ordre où elles ont été stockées.
Les commandes sont non seulement compilées dans la liste d'affichage, mais peuvent aussi être exécutées immédiatement pour obtenir un affichage.
Les commandes d'une liste d'affichage sont les mêmes que les commandes d'affichage immédiat.
Une fois qu'une liste a été définie, il est impossible de la modifier à part en la détruisant et en la redéfinissant.
Les listes d'affichages sont des macros et, à ce titre, ne peuvent pas être modifiées ou paramétrées.
Les listes d'affichage optimisent les temps de calcul et d'affichage car elles permettent la suppression de calculs redondants sur:

- Les opérations matricielles,
- Les conversions de formats d'image bitmap,
- Les calculs de lumière,
- Les calculs de propriétés des matériaux de surface,
- Les calculs de modèle d'éclairage,
- Les textures,
- Les motifs de tracé.

L'inconvénient principal des listes d'affichage est qu'elles peuvent occuper beaucoup de place en mémoire.

a. Commandes principales des listes d'affichage

- ***void glNewList(GLuint liste,GLenum mode) :*** Indique le début de la définition d'une liste d'affichage.
  - *liste:* entier positif unique qui identifiera la liste.
  - *mode:* GL_COMPILE ou GL_COMPILE_AND_EXECUTE suivant que l'on désire seulement stocker la liste ou la stocker et l'exécuter simultanément.
- ***void glEndList(void) :*** Indique la fin d'une liste d'affichage.
- Toutes les commandes depuis le dernier glNewList ont été placées dans la liste d'affichage (quelques exceptions).
- ***void glCallList(GLuint liste) :*** Exécute la liste d'affichage référencée par liste.
- Commandes auxiliaires
- ***GLuint glGenLists(GLsizei nb) :*** Alloue nb listes d'affichage contiguës, non allouées auparavant. L'entier retourné est l'indice qui donne le début de la zone libre de références.
- ***GLboolean glIsList(GLuint liste) :***Retourne vrai si liste est utilisé pour une liste d'affichage, faux sinon.
- ***void glDeleteLists(GLuint l,GLsizei nb) :*** Détruit les nb listes à partir de la liste l.

6. Les textures

Figure 13 : exemple de texture

- **Le placage de texture**

Le placage de texture consiste à placer une image (la texture) sur un objet. Cette opération se comprend aisément lorsqu'il s'agit de plaquer une image rectangulaire sur une face rectangulaire, tout au plus imagine-t-on au premier abord qu'il y a un changement d'échelle à effectuer sur chacune des dimensions. Mais peut-être au contraire le choix est de répéter l'image un certain nombre de fois sur la face.
Se pose ensuite la question d'objets qui ne sont pas des faces rectangulaires : comment définir la portion d'image et la manière de la plaquer ?
Etapes de l'utilisation d'une texture:

- spécification de la texture (1),
- spécification de la technique à utiliser pour appliquer la texture à chaque pixel (2),
- autorisation du plaquage,
- dessin de la scène qui sera implicitement texturée au moyen des paramètres définis en (1) et (2).

Partie II : Conception

CHAPITRE 3

CONCEPTION

Chapitre III : CONCEPTION

INTRODUCTION

Le but de notre travail est de développer une application qui, en entrée, prend un fichier de boules maximales[(1)] dont les centres représentent le squelette de la forme volumique et en sortie donne une visualisation (3D) de cette forme à travers l'affichage des boules et éventuellement des cylindres et cônes.

Pour la représentation des données (boules) nous avons choisi la structure de graphe qui se relève être efficace pour traduire les relations d'adjacences entre objets (boules) et permet en même temps le regroupement des boules selon des critères de choix au sein d'une seule entité

Dans ce cadre l'ensemble de boules est représenté ainsi :

- Chaque boule corresponde à un sommet du graphe,
- La relation « en contact » entre deux boules est représentée par un arc bidirectionnel entre les deux sommets correspondants.

I. Architecture des classes

L'architecture se compose de trois classes :

1. La classe graph

C'est la classe principale qui implémente un ensemble de boules elle contient les attributs suivant :

- **first :** c'est un pointeur sur le premier sommet du graphe
- **N :** le nombre de sommet du graphe

Et les méthodes suivantes :

- **AddVertex :** ajout d'un nouveau sommet au graphe,
- **AddEdge :** ajout d'un arc entre deux sommets,
- **Find :** cherche un sommet dans le graphe,
- **SetEdges :** ajout de tous les arcs existants au niveau du graphe elle fait appel à AddEdge,
- **Display :** visualise le graphe.

2. La classe Vertex

Elle implémente un sommet du graphe et contient les attributs suivant :

- **num :** un entier dont le rôle est d'identifier la sommet (la boule),
- **x,y,z,r :** des réels qui représente successivement les cordonnées et rayon de la boule qui correspond au sommet courant,
- **Edges :** pointeur sur la liste des arcs partant du sommet courant,
- **Next :** pointeur vers le sommet suivant du graphe.

Les méthodes de vertex :

- **getNum (), getX(), getY(), getZ() :** retournent successivement le numéro et les cordonnées de la boule correspondante au sommet courant.
- **connectTo (sommet S) :** crée un arc du sommet courant vers le sommet S passé un argument.
- **getNext () :** retourne un pointeur sur le sommet suivant dans le graphe.
- **getFirstEdg () :** retourne un pointeur sur la tête de la liste des arcs du sommet courant.

---

(1) une boule maximale est une boule contenue dans la forme volumique et qui n'est pas contenue dans autre boule incluse dans la forme.

3. La classe Edge

Implémente un arc du graphe, elle a les attributs suivants :

- **End :** pointeur sur le sommet arrivé de l'arc courant,
- **Next :** pointeur sur l'arc suivant dans la liste des arcs associés.

Elle a les méthodes suivantes :

- **Edge(Vertex *v, Edge *e) :** constructeur qui crée un nouveau arc dont le sommet d'arrivée est v et don l'arc suivant dans la liste d'arcs associée est e.
- **getEnd() :** retourne le sommet d'arrivée de l'arc courant.
- **getNext :** retourne un pointeur sur l'arc suivant dans la liste des arcs associée.

II. Structure de données

Autres structures et fonctions utilisé dans la fonction main() :

1. Types de données

- ***Le type file :*** type de données qui implémente une file

```
typedef struct file{
    int elt;
    struct file *suiv;
}File;
```

- ***le type listGraph :*** implémente une liste de graphes pour la manipulation des listes de sous-graphes

```
typedef struct listGraph{
    Graph *g;
    struct listGraph *suiv;
}ListGraph;
```

- ***le type listDroite :*** implémente une liste de droites pour la rétention et la manipulation des droites retenues.

```
typedef struct listDroite{
    float x1,y1,z1,x2,y2,z2;
    int nb;
    File *f;
    struct listDroite *suiv;
}myLIST;
```

2. Fonctions

Mise à part les différentes fonctions qui implémentent les opérations ordinaires d'ajout de suppression et de recherche d'un élément dans les trois structures de liste ci-dessus, on a les fonctions suivantes :

- ***float poinToline (float x,float y,float z,float X1,float Y1,float Z1,float X2,float Y2,float Z2) :*** calcule la distance d'un point, défini par ses coordonnées (x,y,z), à une droite

définie par deux points (X1,Y1,Z1) et (X2,Y2,Z2) le calcule se fait en utilisant le théorème de Pythagore,

- ***ListGraph *detectConnex(Graph *gr) :*** retourne une liste de sous-graphes connexes détectées à partir du graphe donné un argument,
- ***myLIST *droite(Graph *G, int Nbr_min_boul_par_cyl, float distance) :*** c'est la fonction qui implémente la méthode de détection des alignements proposé ci-dessous, elle a les paramètres suivants :
  - ****G :*** un pointeur sur le graphe correspondant au nuage de boules au sein duquel nous voulons détecter les alignements.
  - ***Nbr_min_boul_par_cyl :*** nombre minimum de points pour qu'un cylindre soit retenue.
  - ***Distance :*** rayon des cylindres (la précision).

En sortie, cette fonction donne la liste de droites sélectionnées avec, chacune, l'ensemble de points qui lui sont proches, chaque droite est définie par les deux points d'intersection avec la sphère.

III. Transformée de Hough

Nous cherchons à substituer des sous ensembles de boules simplement connexes et alignées par des cylindres et cônes, selon la variance et le rapport entre différent rayon qui constitue un sous ensemble de boules donné, de manière à optimiser la représentation de la forme initiale un problème est alors posé « comment, a partir d'un nuage de point (les centres des boules) dans un espace 3D, détecter tous les droite qui passe au moins par trois points ».
Dans un espace 2D ce problème est résolu notamment à l'aide de la transformée de Hough.
Vu qu'il n'y a, à nos jours, pas de solution mathématique pour ce problème nous avons proposé une méthode approximative.

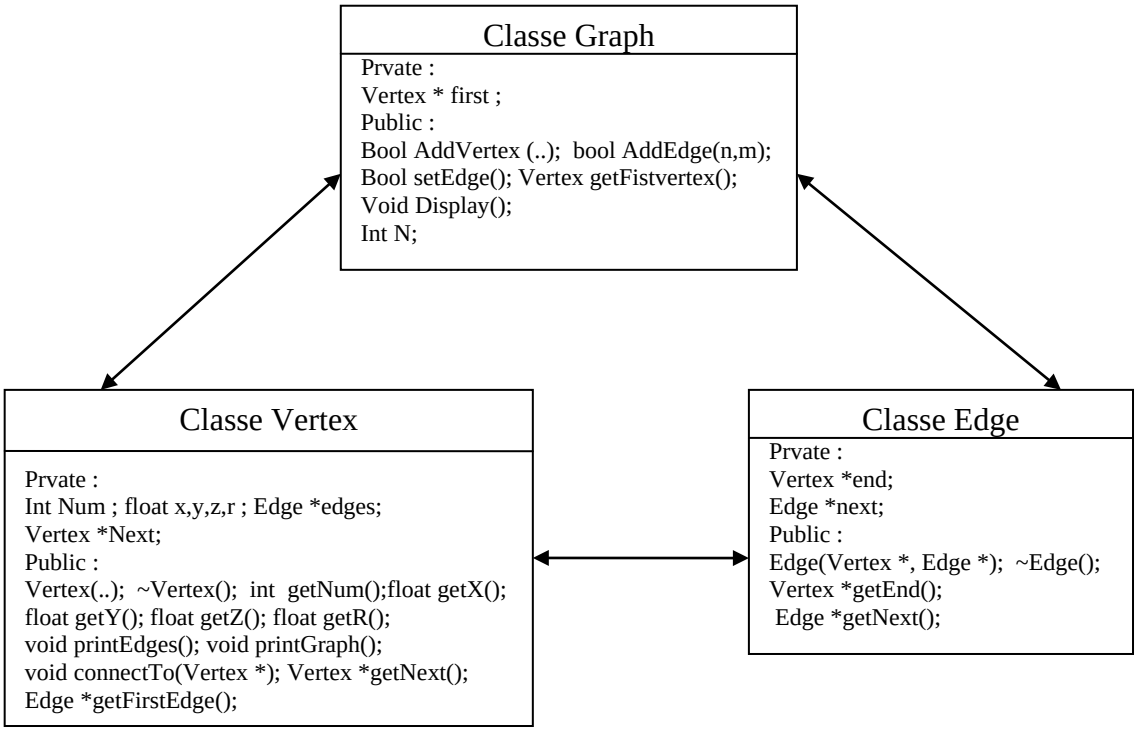
Méthode proposé

On peut résumer notre méthode ainsi :

- calculer le le barycentre G du nuage de points et faire un translation de l'origine du repère vers G,
- calculer r = Max GM pour tout M du nuage et considérer la sphère de centre G et de rayon R=r+5 (éviter l'effet de bord),
- procéder à un échantillonnage uniforme de la surface de sphère,
- Considérer tous les cylindres s'appuyant sur deux échantillons distincts et prendre l'axe de chaque cylindre comme représentant de toutes les droites qui passe par les deux échantillons on obtient ainsi une discrétisation de l'espace des droites qui traversent la sphère (susceptible de passer par un point du nuage),
- Pour tous les cylindres tester l'appartenance de tous les point en calculant la distance du point de l'axe du cylindre,
- Privilégier les cylindres avec un grand nombre de point.

Cette méthode permet de détecter toute les droites dans un nuage de points donné, mais présente aussi l'inconvénient d'être très lente pour une précision raisonnable; c.-à-d. pour un pas d'échantillonnage petit (pas $\leq 1°$).

IV. Diagramme de classe



Partie III : Partie Pratique

CHAPITRE 4

REALISATION

Chapitre IV : REALISATION

I. Présentation de l'application

Le programme développé comprend deux volets :

1. Volet lecture de données leurs traitement et calcul du résultat

Ce volet comporte plusieurs étapes :

- La première étape consiste à lire les données à partir d'un fichier de boules ensuite adapter ces données à la structure de graphe illustrée ci-dessus dans la partie conception, le résultat de cette étape est un graphe qui contient tous les données (boules),
- Entrée dans la boucle de discrétisation de l'espace des droites susceptible de passer par notre nuage de points, dans cette étape le programme parcours l'ensemble fini de droite, résultat de la discrétisation de l'espace, en associant à chacune la liste de points de son voisinage proche. Le résultat de cette étape est une liste de droite chacune avec une liste de points associés,
- Parcours de la liste de droites, résultat de l'étape précédente, et la mise des listes de points associé dans des sous-graphes. la sortie de cette étape est une liste de droite chacune avec un sous-graphe de points associées.
- Parcours de la liste résultante de l'étape précédente en divisant chaque sous-graphe en un ensemble de composante connexes (sous-graphe connexe) la sortie de cette étape est la liste de tous les composante connexes et alignées,
- Traiter chaque sous ensemble de boules de la liste résultante de l'étape précédente (sous ensemble connexes et alignées) pour décider, en fonction de la variation des rayons d'une extrémité à l'autre s'il convient de les substituer par un cylindre, un cône ou une combinaison de cylindres, cônes et boules.

2. Volet visualisation

Ce volet concerne la partie OpenGL et comporte les opérations suivant :

- **Initialiser la GLUT et créer la fenêtre**, le glut une, fois initié, gère les interactions entre l'OpenGL et le système pour le rendu visuel de la scène dans cette étape on crée aussi la fenêtre en choisissant ses dimensions et le mode d'affichage. Les fonctions appelées sont *glutInit, glutInitDisplayMode, glutInitWindowPosition, glutInitWindowSize, gMainWindow, glutCreateWindow*.
- **Mettre en place les fonctions de rappel :** c'est fonction sont associées au différente type d'événements envoyés par le système dans notre cas il s'agit des interruptions du clavier ou de la souris en plus de la fonction d'affichage.
- **Décrire la scène 3D par la création des objets** permettant l'affichage des formes volumiques, boules, cylindre et cône, qui composent notre scène à afficher ensuite le calcule des transformations géométriques telle que translations et rotations qui doivent s'appliquer sur chaque objet pour son positionnement exact.

- **Entrée dans la boucle principale d'affichage** par appel de la fonction glutMainLoop() qui gère les événements liés au fonctionnement du programme et exécute les fonction qu'on a associées à ces événement.

II. Environnement de développement

Le développement de cette application est réalisé dans l'environnement de développement intégrée Dev-C++ après avoir ajouté les bibliothèques et fichiers dll nécessaires pour la compilation et l'exécution du code OpenGL.

1. Le Dev-C++

Dev-C++ est un environnement de développement intégré permettant de programmer en C et en C++. Il utilise la version MinGW du compilateur gcc (GNU Compiler Collection) (venu du monde du logiciel libre) et permet d'exporter ses projets sous fichiers .dev. Dev-C++ peut aussi être utilisé en combinaison avec Cygwin ou tout autre compilateur basé sur GCC. Ce compilateur très complet comprend entre autre un "répertoire de classes", servant à localiser facilement les fonctions, classes et membres du code source, un "répertoire de fonctions incluses", fonctionnant comme le répertoire de classes mais pour chercher dans les fichiers inclus (header), et un débogueur qui permet de surveiller l'état des variables pendant l'exécution du programme. Il est multiplateforme et permet de générer des exécutables fonctionnels sous la majorité des plateformes.

2. GLUT

Pour créer un programme OpenGL dans un environnement fenêtré comme X-Window, il faut être en mesure de créer une fenêtre pour y afficher nos images, de la détruire, et de gérer les interactions avec l'utilisateur par le clavier, la souris et tous les autres types de périphériques d'entrée. OpenGL se voulant indépendant de toute plate-forme matérielle, l'API ne fournit pas de telles fonctions. C'est la bibliothèque annexe, Glut (GL Utility Toolkit) qui fournit ces fonctions. Développée par Mark Kilgard, Glut est fourni avec les dernières moutures de Mesa (une implémentation d'OpenGL).
Voici une liste de toutes les fonctionnalités proposées par Glut :

- Gestion de fenêtres.
- Gestion des événements par fonctions de rappel.
- Gestion de périphériques d'entrée exotiques (spaceballs...),
- Gestion des polices de caractères,
- Fonctions de création de menus.

III. Captures d'écran

Voici quelques captures d'écran pendant l'exécution du programme qui montre différentes vues de l'espace poral :

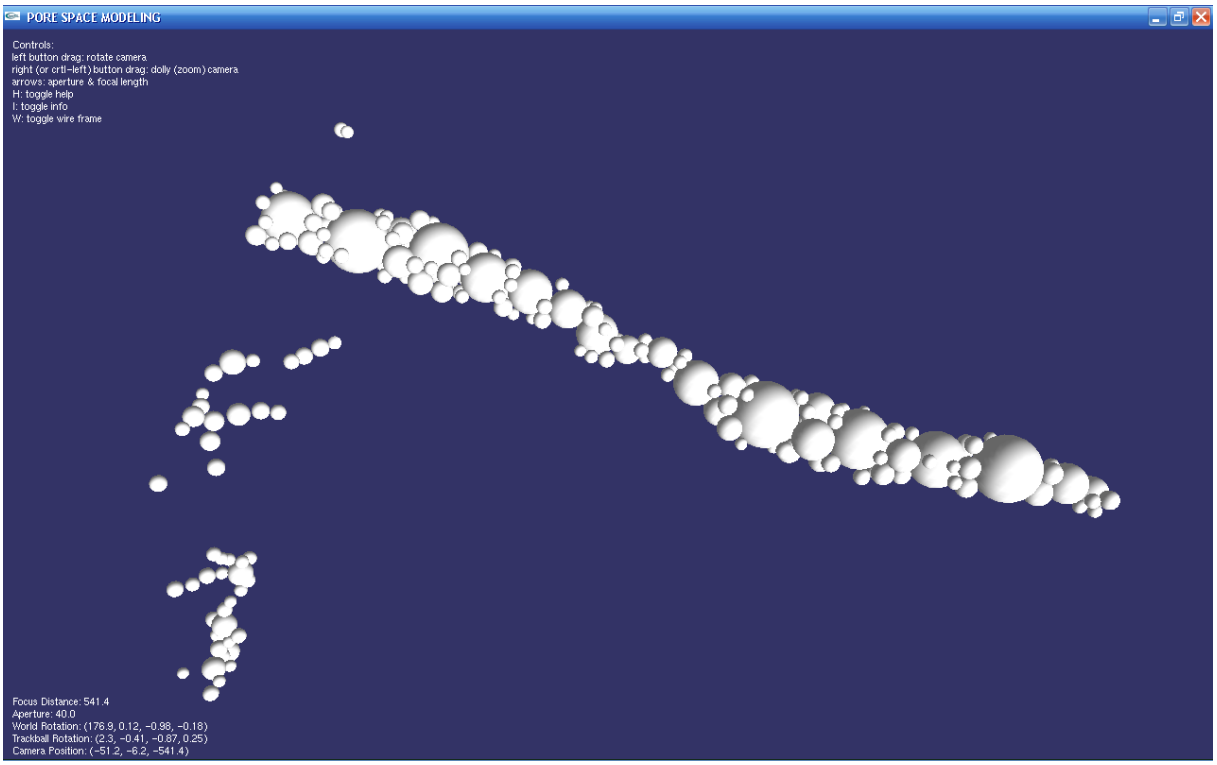


Figure 14: Capture 1

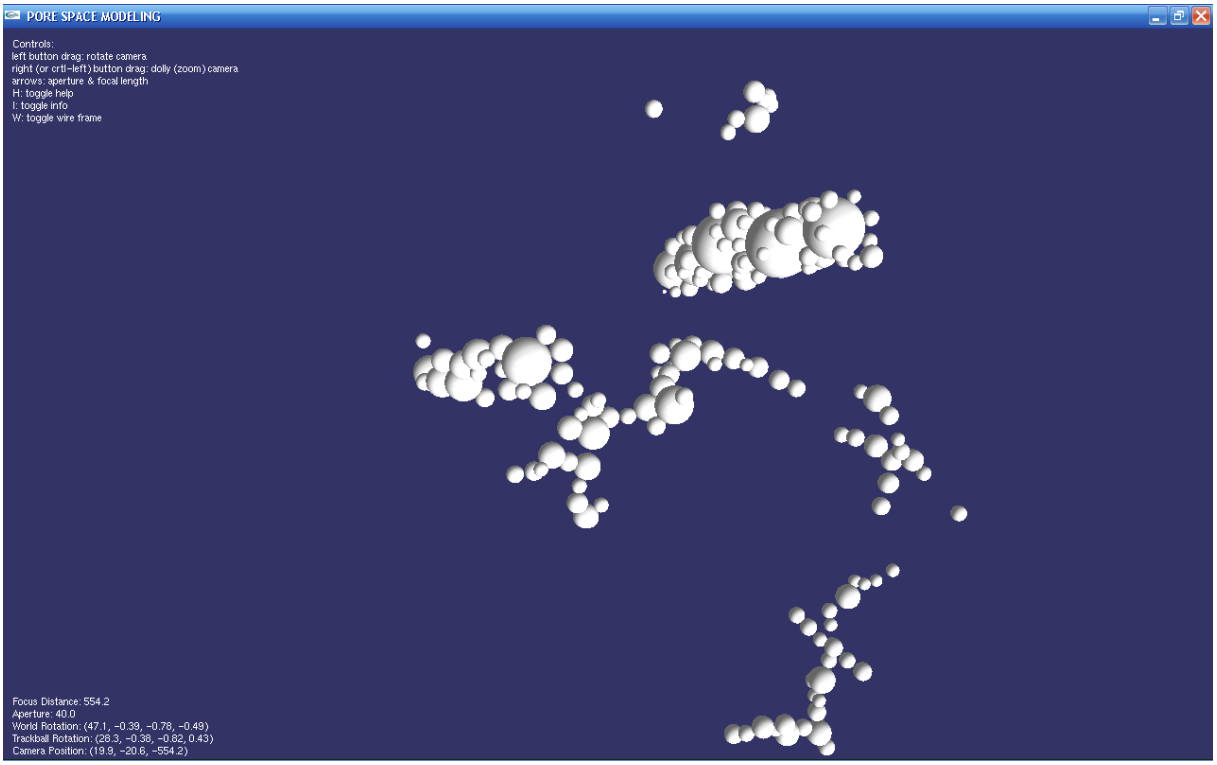


Figure 15: Capture 2

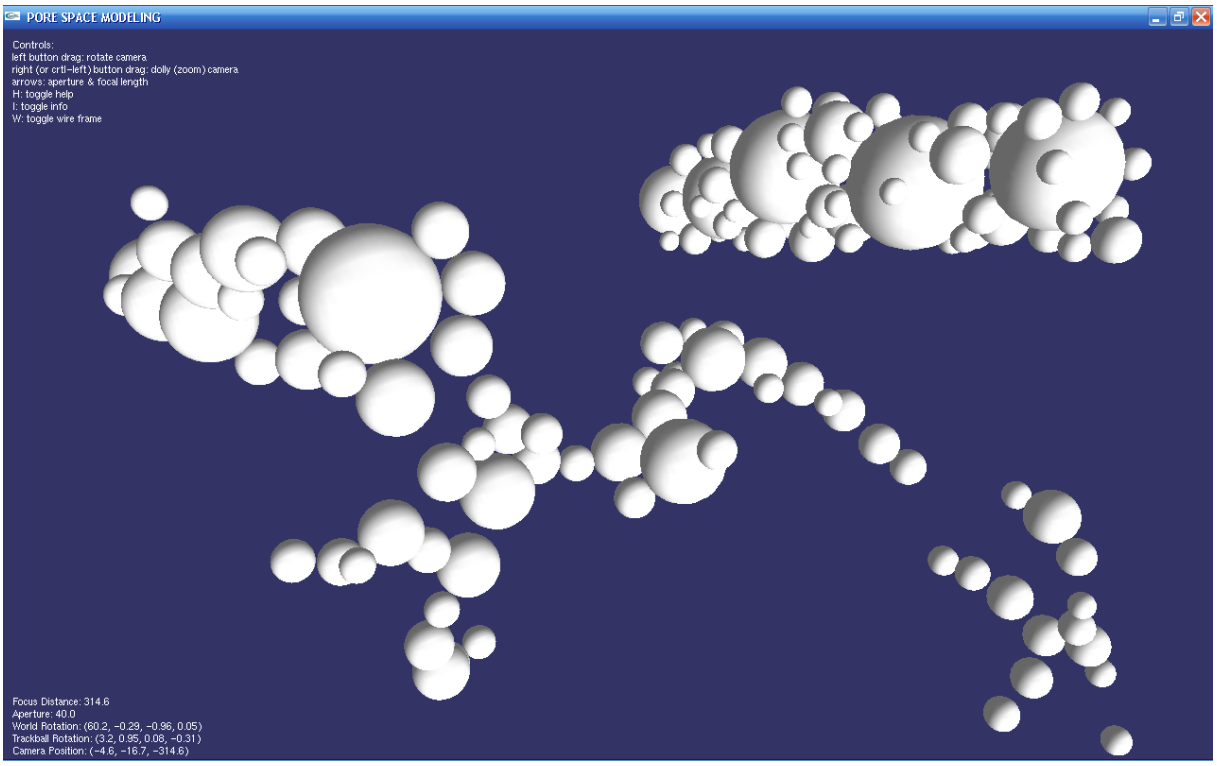


Figure 16: Capture 3

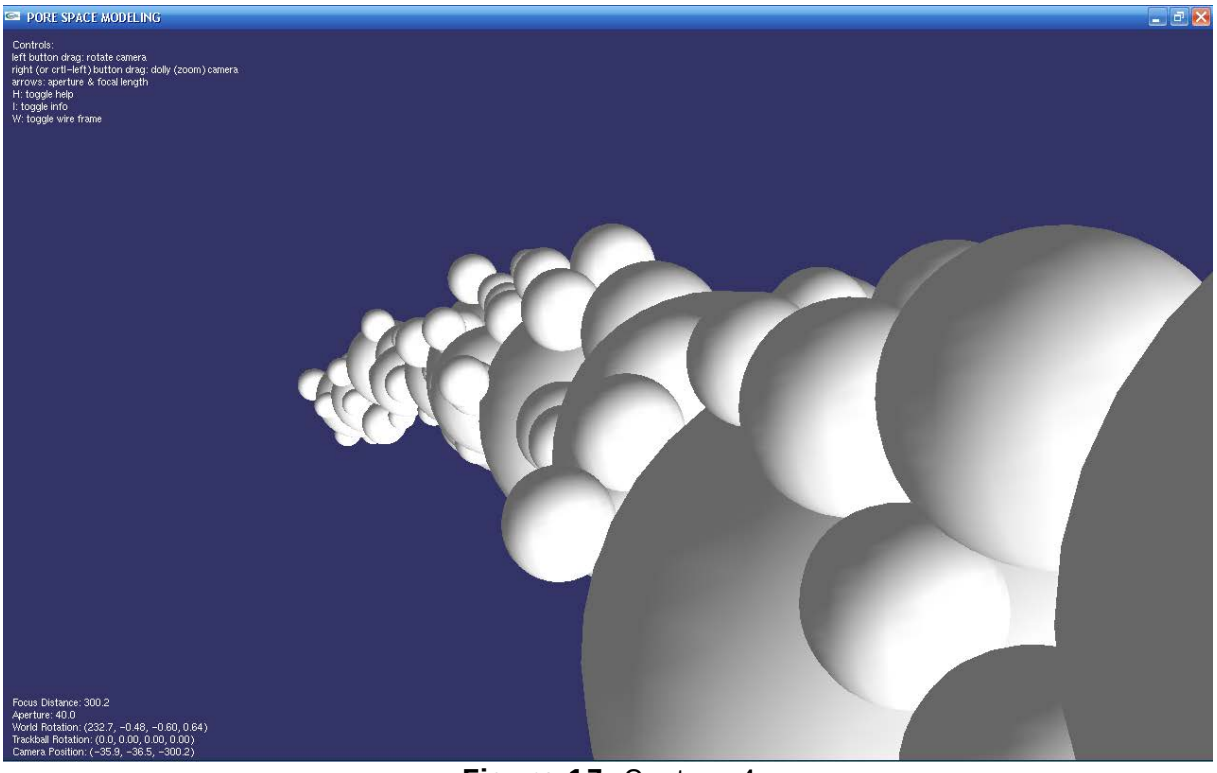


Figure 17: Capture 4

CONCLUSION & PERSPECTIVES

CONCLUSION & PERSPECTIVES

À partir d'une modélisation de l'espace porale par un ensemble de boules maximales, nous avons essayé d'introduire d'autres primitives géométriques à savoir les cylindres et les cônes pour révéler certain propriétés du sol.
Dans ce contexte nous avons proposé une méthode informatique pour étendre la transformée de Hough à un espace 3D, cette méthode se révèle efficace notamment dans le cas d'une connaissance à priori de l'existence des alignements important dans un nuage de point 3D, l'importance de cette méthode ne se limite pas à la science du sol mais peut s'appliquer également aux différente applications de la vision par ordinateur notamment la reconstruction volumique. Néanmoins, cette méthode comporte toujours des insuffisances quand à la précision et la rapidité de calcule.
Nous comptons dans un futur travail améliorer cette méthode et mettre au profil de notre travail une étude approfondie des propriétés mathématiques des graphes : les composantes connexes et simplement connexes ainsi que la théorie des chaines.

BIBIOLGRAPHIE

BIBLIOGRAPHIE

BIBLIOGRAPHIE

[1] Olivier MONGA et Radu HORAUD. «Vision par Ordinateur : outils fondamentaux »,
[2] Christine FERNANDEZ-MALOIGNE. «Introduction aux techniques de traitement et d'analyse d'images »,
[3] Samuel R. BUSS A mathematical Introduction with OpenGl.