
virtual Tracker

Projet de septième semestre réalisé au
LIG, laboratoire d'infographie, EPFL, Février 1997

Etudiant: Niklaus Hirt, section: Informatique

Professeur: D. Thalmann

Assistant: S. Rezzonico

TABLE DES MATIERES

| | | |
|-------|--|----|
| 1.0 | Introduction | 4 |
| 2.0 | Formules et données mathématiques | 7 |
| 2.1 | Introduction | 7 |
| 2.2 | Calcul de moments | 7 |
| 2.3 | Formule de Green discrétisée | 7 |
| 2.4 | Calcul des paramètres | 8 |
| 2.4.1 | Calcul du centre (x,y) | 8 |
| 2.4.2 | Calcul de l'angle de rotation $\text{rot}(z)$ | 8 |
| 2.4.3 | Moments centraux et calcul des deux axes de l'ellipse | 8 |
| 2.4.4 | Calcul de la position en z | 9 |
| 2.4.5 | Calcul des angles de rotation $\text{rot}(x)$ et $\text{rot}(y)$ | 9 |
| 3.0 | Localisation et poursuite d'un marqueur | 12 |
| 3.1 | Introduction | 12 |
| 3.2 | Approche choisi | 12 |
| 3.2.1 | Le point de départ | 12 |
| 3.2.2 | L'algorithme de détection du contour | 13 |
| 3.2.3 | Les quatre régions de l'image | 15 |
| 4.0 | Localisation et poursuite de deux marqueurs | 18 |
| 4.1 | Introduction | 18 |
| 4.2 | L'algorithme de détection des deux contours | 19 |
| 4.2.1 | Premier approche | 19 |
| 4.2.2 | Deuxième approche | 20 |
| 5.0 | Implémentation | 22 |
| 5.1 | Introduction | 22 |
| 5.2 | Structure de la boucle principale | 22 |
| 5.3 | Structure de l'image | 24 |
| 5.4 | Calibration des couleurs | 24 |
| 5.5 | Détection des contours | 24 |
| 5.6 | Recherche d'un cercle lors d'une perte de synchronisation. | 25 |
| 5.7 | Structure de la boucle principale et passage de paramètres. | 25 |
| 5.8 | Détection de la position des doigts | 26 |
| 5.9 | Montage de la caméra | 26 |
| 6.0 | Guide programmeur | 29 |
| 6.1 | Fonctions mises à disposition | 29 |
| 6.2 | Programme d'exemple | 30 |
| 6.3 | Le fichier de définition libTrace.h | 32 |
| 7.0 | Conclusion | 34 |
| 8.0 | Bibliographie | 36 |

1 Introduction

1.0 Introduction

Les applications multimedia deviennent de plus en plus exigeantes de nos jours. Pour tenir compte de cette évolution, on a besoin de dispositifs puissants pour capturer les informations nécessaires à une bonne exploitation de ces nouvelles possibilités. Les capteurs d'information les plus connus sont probablement le clavier et la souris.

En faisant appel à la réalité virtuelle, on arrive à assurer une facilité inconnue jusqu'à présent pour utiliser des programmes. Le problème qui se pose alors, c'est qu'on travaille souvent dans des espaces virtuels à trois dimensions. Dans ce domaine les dispositifs connus ne sont plus adaptés et on a été amené à en trouver des nouvelles solutions:

- Le SpaceBall qui consiste en une boule qui capte les forces dans trois dimensions et qui peut ainsi être considéré comme une sorte de Joystick 3D. L'inconvénient le plus gênant vient du fait que la boule reste fixe sur le support, on n'a donc pas le sentiment de mouvement dans l'espace.
- Le DataGlove: Une source fixe crée un champ bien défini. Un gant muni d'un capteur de champ magnétique génère des courants qui varient suivant sa position dans le champ. Cette méthode est très précise, mais malheureusement très coûteuse et elle demande un câblage considérable, ce qui limite la liberté de mouvement.

Le but de ce projet est de mettre à disposition une solution qui permet à l'utilisateur normal de profiter de la réalité virtuelle en trois dimensions, tout en étant aussi facile à utiliser que possible et ne demandant pas de matériel spécialisé.

La solution se base sur un article publié par C. Maggioni¹: L'utilisateur met un gant qui est équipé de deux cercles de couleur (Figure 1). Une caméra fournit l'image à un ordinateur qui en extrait les informations de position et de rotation dans les trois dimensions (Figure 2).

Figure 1.

Le gant utilisé

¹ C. Maggioni: "A Novel Gestural Input Device for Virtual Reality", 1993

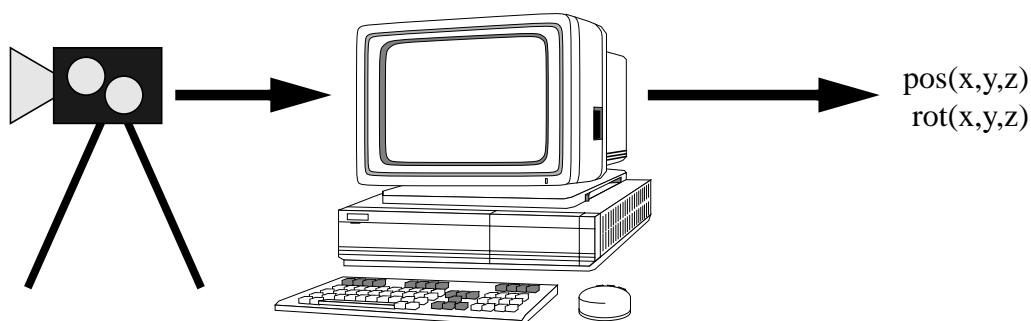


Figure 2.

Traitement des images

Tout d'abord quelques notions mathématiques de base sont présentées. Elles servent à comprendre le déroulement des calculs qui seront effectués. Ensuite seront expliquées les possibilités et l'approche choisie pour localiser et poursuivre UN marqueur qui se trouve sur le gant.

Enfin, la solution avec deux marqueurs est présentée en exposant deux méthodes possibles dans le cadre de l'algorithme choisi.

L'implémentation des algorithmes est une partie très importante, car la qualité de détection en dépend fortement. Une analyse de l'image en un temps très court est primordial. On est alors obligé à trouver des solutions fortement optimisées pour assurer que l'utilisation soit fluide.

Les programmeurs qui comptent intégrer la librairie dans leurs applications trouveront les explications nécessaires à l'utilisation des fonctions mises à disposition, ainsi qu'un petit exemple d'implémentation.

Finalement, quelques idées concernant l'évolution possible du projet réalisé seront présentées, ainsi que des observations sur le travail accompli.

2 Formules et données mathématiques

2.0 Formules et données mathématiques

2.1 Introduction

Comme l'idée pour calculer les paramètres de position et de rotation est basée sur les calculs des moments de l'aire de l'objet, on donnera dans ce chapitre les formules et procédés théoriques qui sont nécessaires.

On suppose qu'on connaît la surface A ou le contour B de l'objet. On verra plus tard comment on peut les déterminer.

Toutes les formules dans ce chapitre sont tirées de l'article "A Novel Gestural Input Device for Virtual Reality".

2.2 Calcul de moments

Pour calculer ces moments on a besoin de connaître l'aire du cercle.

$$m_{(p,q)} = \frac{1}{p+1} \cdot \int_{(x,y) \in A} x^{p+1} \cdot y^q \cdot dy$$

Formule 1.

Calcul du moment $m(p,q)$

L'application de cette formule pose deux problèmes:

- Premièrement il nous faudrait une formule qui calcule les moments à partir d'un ensemble de points discrets.
- Deuxièmement le temps de calcul devient trop important si on intègre sur l'aire A.

2.3 Formule de Green discrétisée

Pour éviter les deux inconvénients de la formule 3, on choisit la formule de Green discrétisée, qui nous calcule les mêmes moments à partir du contour B de l'aire considérée. On a donc:

$$m_{(p,q)} = \frac{1}{p+1} \cdot \sum_{(x_i, y_i) \in B} x_i^{p+1} \cdot y_i^q \cdot \Delta y_i$$

$$\Delta y_i = \begin{cases} 1, & \text{si } y_{i+1} < y_i \\ 0, & \text{si } y_{i+1} = y_i \\ -1, & \text{si } y_{i+1} > y_i \end{cases}$$

Formule 2.

Formule de Green discrétisée

2.4 Calcul des paramètres

Pour effectuer le calcul des paramètres recherchés, on aura besoin d'un ensemble de points, sous forme d'une liste, qui constituent le contour B de la région. Cette liste contient les N points stockés en tuple.

2.4.1 Calcul du centre (x,y)

En calculant les moments $m(0,0)$, $m(1,0)$ et $m(0,1)$ on peut directement trouver le centre de masse, qui représentera le centre (x,y) du cercle recherché, dans le plan normal à l'axe de vue.

$$x = \frac{m_{1,0}}{m_{0,0}} \quad \left| \quad y = \frac{m_{0,1}}{m_{0,0}}$$

Formule 3. Centre (x,y) d'un cercle

2.4.2 Calcul de l'angle de rotation $rot(z)$

En appliquant ce procédé à deux cercles concentriques aux contours B1 et B2 on obtient deux centres, (x_1,y_1) et (x_2,y_2) . A partir de la distance vectorielle, on peut calculer l'angle de rotation autour de l'axe Z.

$$rot_z = \arctan\left(\frac{x_2 - x_1}{y_2 - y_1}\right)$$

Formule 4. Angle de rotation $rot(z)$ autour de l'axe Z

2.4.3 Moments centraux et calcul des deux axes de l'ellipse

En faisant des rotations autour des axes X et Y, le cercle prend la forme d'une ellipse.

Pour calculer les deux axes M et N, on passe par le calcul des moments centraux, tout en connaissant le centre de masse, calculé en 2.4.1.

$$\tilde{m}_{(p,q)} = \frac{1}{p+1} \cdot \sum_{(x_i, y_i) \in B} (x_i - \hat{x})^{p+1} \cdot (y_i - \hat{y})^q \cdot \Delta y_i$$

$$\hat{x} = \frac{m_{1,0}}{m_{0,0}} \quad \left| \quad \hat{y} = \frac{m_{0,1}}{m_{0,0}}$$

Formule 5. Moment central (p,q) d'un cercle

On calcule les deux axes M et N avec:

$$M = a + \sqrt{a-b}$$

$$N = a - \sqrt{a-b}$$

où

$$a = \frac{\tilde{m}_{2,0} + \tilde{m}_{0,2}}{2}$$

$$b = \tilde{m}_{2,0} \cdot \tilde{m}_{0,2} - \tilde{m}_{0,2}^2$$

Formule 6. Axes de l'ellipse

2.4.4 Calcul de la position en z

Connaissant les données particulières de la caméra, on peut tirer de ces deux axes la position en Z, tout en respectant les rotations dans l'espace. En gros on peut dire que la distance entre le cercle et la caméra est déterminé par la taille des axes.

2.4.5 Calcul des angles de rotation rot(x) et rot(y)

L'angle entre l'axe M et l'axe X est:

$$\tan(2 \cdot \theta) = \left[\frac{2 \cdot \tilde{m}_{1,1}}{\tilde{m}_{2,0} + \tilde{m}_{0,2}} \right]$$

Formule 7. L'angle entre M et l'axe X

Avec cet angle on calcule la surface (A,B,C) normale au plan du cercle:

$$A = -B \cdot \tan(\theta)$$

$$B = \sqrt{\frac{1}{(-\tan(\theta) \cdot \sin(\theta) - \cos(\theta))^2} \cdot \left(1 + (\tan(\theta))^2 + \left(\left(\frac{M}{N} \right)^2 - 1 \right) \right)}$$

$$C = \sqrt{1 - A^2 - B^2}$$

Formule 8. Plan (A,B,C) normal au cercle

On applique un changement de repère en faisant une transformation inverse, connaissant l'angle de rotation $rot(z)$.

$$A' = A \cdot \cos(rot_z) - B \cdot \sin(rot_z)$$

$$B' = B \cdot \cos(rot_z) + A \cdot \sin(rot_z)$$

Formule 9. Transformation inverse selon l'angle $rot(z)$

Finalement on obtient les deux paramètres manquants:

$$rot_x = \arccos\left(\frac{C}{\sqrt{B'^2 + C^2}}\right)$$

$$rot_y = \arcsin(-A')$$

Formule 10. Calcul des angles de rotation $rot(x)$ et $rot(y)$

Ainsi on réussit à calculer tous les six paramètres nécessaires pour déterminer la position et la rotation dans l'espace.

Mais ce ne sont que les théories mathématiques, alors que dans la pratique il y a des problèmes plus complexes à résoudre, comme on verra dans les deux chapitres suivants.

3 Localisation et poursuite d'un marqueur

3.0 Localisation et poursuite d'un marqueur

3.1 Introduction

Comme on a vu dans le chapitre précédent, le problème principal consiste à extraire le contour B de l'objet en question. Dans notre cas il s'agit d'un cercle, ce qui rend plus facile la tâche de détection.

Partant du choix qu'on utilisera un cercle d'une couleur différente de l'environnement, on peut facilement retrouver les bords du cercle, en utilisant une fonction de seuil.

Les points seront stockés dans une liste de taille variable. Un premier problème consiste à insérer les points trouvés dans le bon ordre dans cette liste. A cause de la Formule de Green, le parcours du contour doit se faire dans le sens des aiguilles d'une montre.

Un autre problème est la précision de la recherche, c'est à dire qu'on n'enregistre pas de points qui ne font pas partie du contour de l'objet, car ces points parasites mènent à des résultats erronés.

Dans une première approche, un algorithme de traçage de contours élaboré par T.Pavlidis² a été choisi. Malheureusement, cet algorithme n'était pas bien adapté à ce problème, car il était trop gourmand en temps de calcul, dû au fait que la liste de points devait être triée avant de passer aux calculs de moments.

3.2 Approche choisi

Pour une deuxième approche, on a abouti à une solution inspirée par l'algorithme de Pavlidis. On trace recursivement le contour en utilisant une fonction de seuil, ce qui semble être mal adapté, mais en pratique très efficace.

Cet algorithme sera décrit par la suite.

3.2.1 Le point de départ

Le premier problème consiste à trouver l'objet dans l'image. Alors on fait l'hypothèse de connaître un point à l'intérieur du cercle, qu'on appellera (x',y') par la suite.

Comment trouver ce point cherché?

L'approche la plus évidente, c'est de parcourir l'image ligne par ligne, jusqu'à ce qu'on trouve un point de la couleur du cercle. Malheureusement, c'est trop lent et pas précis, surtout si on enregistre d'autres objets de la même couleur.

² T. Pavlidis: "Structural Pattern Recognition", Springer-Verlag 1977

Alors on suppose que le centre de l'image précédente se trouve encore dans le cercle. Si on met cette solution en oeuvre, on constate qu'elle est très rapide, mais elle limite la vitesse de mouvement considérablement, si les temps entre deux parcours de l'algorithme devient trop long.

On a donc:

$$\frac{d_{min}}{T} \leq v_{max} \leq \frac{d_{max}}{T}$$

Formule 11.

Bornes de la vitesse maximale de mouvement

où d_{min} et d_{max} sont les distances minimales respectivement maximales entre l'ancien centre (x',y') et le contour, et T est le temps de traitement pour une itération de la boucle effectuant tous les calculs.

Comme la pratique l'a montré, on arrive à une Frame Rate d'environ 10 images par seconde, donc on a des temps de l'ordre de la dixième de seconde, ce qui est largement suffisant pour suivre des mouvements qui ne sont pas trop irréguliers.

3.2.2 L'algorithme de détection du contour

Comme on peut supposer connaître un point à l'intérieur du cercle, le traçage du contour se fait facilement.

En pseudo code on a le programme suivant:

```
tant que Couleur = Couleur du cercle cherché, alors
  avancer sur la ligne jusqu'à ce que Couleur < Seuil
  mettre le point dans la liste
  aller à la prochaine ligne et reculer de N pas
```

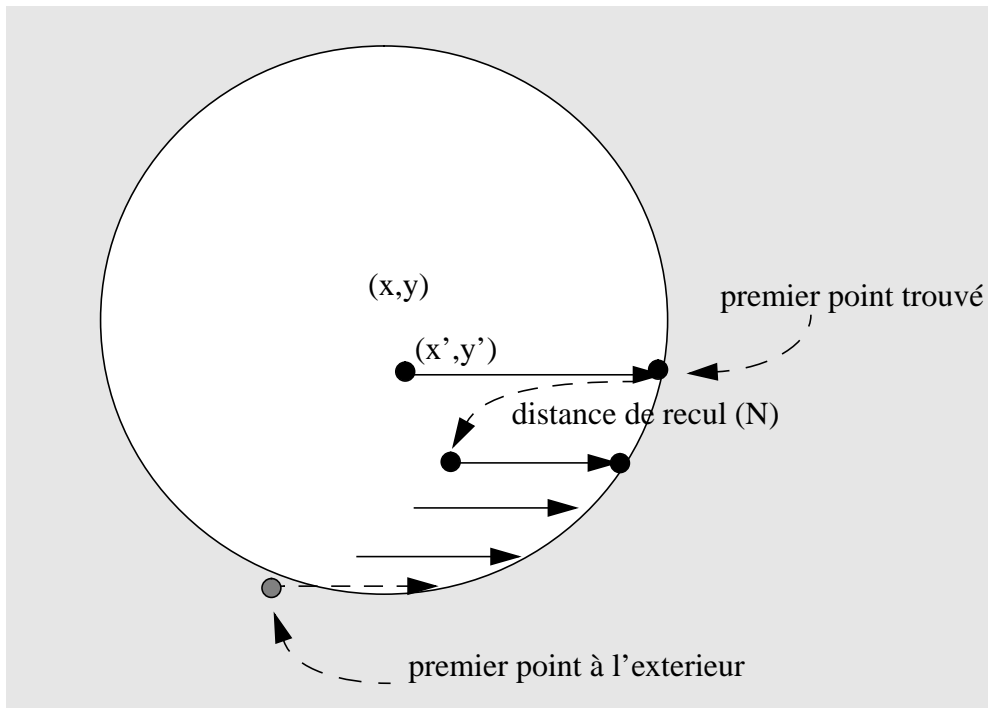


Figure 3.

Algorithme de traçage de contour

On n'a obtenu qu'une partie du contour. En appliquant cet algorithme quatre fois pour toutes les quatre possibilités, on reçoit un contour quasiment fermé.

En appliquant cet algorithme, le contour détecté ne contient pas de structures horizontales. A première vue ca paraît être un inconvénient, mais en vérifiant dans la formule de Green au chapitre précédent, on remarque que dans les calculs des moments les composantes horizontales sont également négligées (deuxième cas, $y(i)=y(i+1)$).

3.2.3 Les quatre régions de l'image

La solution présentée divise l'image en quatre régions R1...R4, ou pour R2 et R4 le sens de parcours du contour est malheureusement inversé.

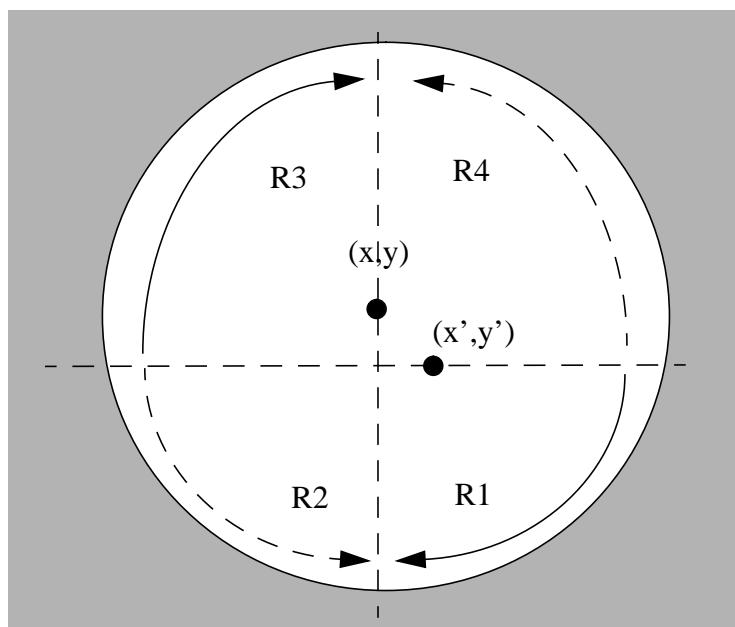


Figure 4. Les quatre régions

Pour résoudre ce problème, on divise la liste des points du contour en deux. Donc une liste pour les points de R1 et R3, qui sont dans le bon ordre, et une pour les points de R2 et R4, qui sont placés dans l'ordre inverse.

Pour les calculs des moments on inverse le signe du coefficient lors des calculs avec la liste inversée

pour R1 et R3 pour R2 et R4, inversés:

$$\Delta y_i = \begin{cases} 1, & \text{si } y_{i+1} < y_i \\ 0, & \text{si } y_{i+1} = y_i \\ -1, & \text{si } y_{i+1} > y_i \end{cases} \quad \Delta y_i = \begin{cases} -1, & \text{si } y_{i+1} > y_i \\ 0, & \text{si } y_{i+1} = y_i \\ 1, & \text{si } y_{i+1} < y_i \end{cases}$$

Formule 12. Nouvelles conditions pour le calcul de moments

En pratique la qualité de la détection du contour peut subir des dégradations à cause d'autres objets qui bougent dans l'image ou de mouvements trop brusques de l'objet qu'on essaie de suivre. On le remarque quand le centre commence à "danser" autour du point réellement cherché.

Pour stabiliser les résultats des calculs on peut stocker les N derniers centres dans une liste circulaire et en calculer la moyenne. Cela baisse la vitesse maximale de mouvement et augmente le temps de réaction à des changements de direction de mouvement. Par contre, le centre arrive à suivre le cercle à une certaine distance, même s'il ne se trouve plus dans l'objet.

En plus on peut négliger les points qui se trouvent à l'extérieur d'un carré de côté S et centre $(x'y')$.

Un autre problème se pose lorsqu'on veut tracer deux marqueurs en même temps, comme c'est nécessaire dans le cadre de ce projet.

4 Localisation et poursuite de deux marqueurs

4.0 Localisation et poursuite de deux marqueurs

4.1 Introduction

Pour le problème de la localisation on peut partir du même principe que dans le chapitre 3, en supposant que l'ancien centre (x',y') se trouve à l'intérieur des deux cercles.

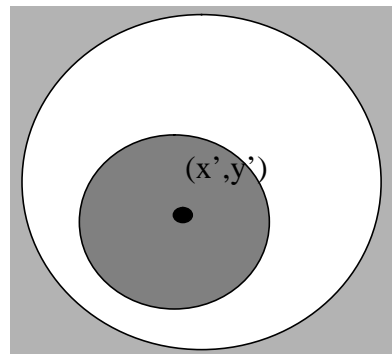


Figure 5.

L'ancien centre (x',y')

En partant de l'ancien centre du petit cercle, on est toujours sûr de se trouver à l'intérieur des deux cercles. Mais en pratique, cette solution ne mène pas à un traçage stable, car ce cercle est trop petit pour en extraire un contour assez précis.

Alors, il vaut mieux, partir de l'ancien centre du grand cercle, et on est toujours sûr de bien suivre le mouvement. Par contre on arrive pas toujours à calculer le centre du petit cercle, et donc l'angle de rotation $\text{rot}(z)$. Mais en gardant l'ancienne valeur de $\text{rot}(z)$, on obtient des résultats tout à fait satisfaisants, en ce qui concerne stabilité et précision.

4.2 L'algorithme de détection des deux contours

4.2.1 Premier approche

Partant alors du principe expliqué, on essaie de déterminer les deux contours de manière suivante.

```

/*Chercher les points des contours des deux cercles */
tant que Couleur = Couleur du cercle intérieur, alors
    avancer sur la ligne jusqu'à ce que Couleur != Couleur du cercle intérieur
    mettre le point dans la liste intérieure
tant que Couleur = Couleur du cercle extérieur, alors
    avancer sur la ligne jusqu'à ce que Couleur < Seuil
    mettre le point dans la liste extérieure
    aller à la prochaine ligne et reculer de N pas
/*Sortie du petit cercle, chercher que pour le grand cercle */
tant que Couleur = Couleur du cercle extérieur, alors
    avancer sur la ligne jusqu'à ce que Couleur < Seuil
    mettre le point dans la liste extérieure
    aller à la prochaine ligne et reculer de N pas
    
```

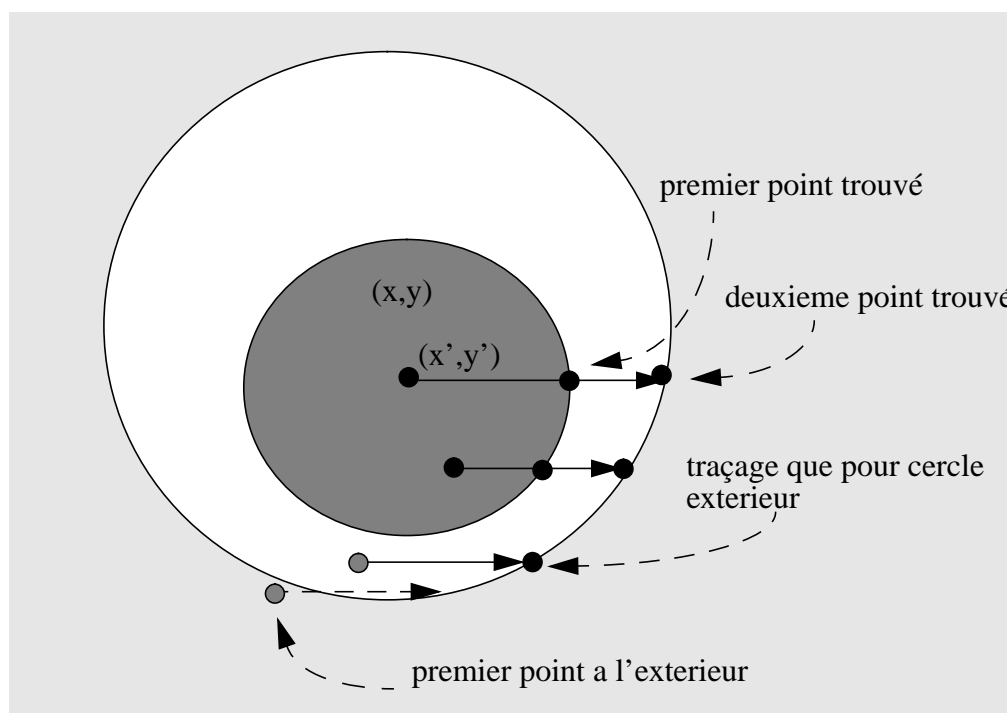


Figure 6.

Algorithme de traçage de contour: Premier approche

Cette méthode était très lente, car il fallait tracer toute la ligne à partir du point de départ, ce qui faisait baisser la Frame Rate à environ une image par seconde, ce qui rend l'utilisation impossible.

4.2.2 Deuxième approche

La deuxième approche consistait à faire tracer les deux cercles séparément, avec des conditions de seuil différentes, mais avec le même point de départ (x',y') .

```
/*Chercher les points du contour du petit cercle*/  
tant que Couleur = Couleur du cercle intérieur, alors  
    avancer sur la ligne jusqu'à ce que Couleur != Couleur du cercle intérieur  
    mettre le point dans la liste intérieure  
    aller à la prochaine ligne et reculer de N pas
```

```
/*Chercher les points du contour du grand cercle*/  
tant que Couleur < Seuil , alors  
    avancer sur la ligne jusqu'à ce que Couleur > Seuil  
    mettre le point dans la liste extérieure  
    aller à la prochaine ligne et reculer de N pas
```

On remarque qu'en pratique c'est une très bonne solution, donc elle a été retenue.

Finalement on possède toutes les connaissances théoriques pour mettre en oeuvre ce qui manque encore: le code de la librairie qui servira à développer des applications profitant des idées présentées.

5 Implémentation

5.0 Implémentation

5.1 Introduction

Le programme est implémenté de manière à éviter au programmeur de devoir s'occuper des détails. Cela n'assure pas une très grande flexibilité, mais par contre l'utilisation est très facile (cf. Chapitre 6).

Les problèmes principaux qui se sont posés lors de l'implémentation étaient des problèmes d'optimisation, pour que le temps de traitement de la boucle de traitement principale soit minimal. Tout cela pour assurer une Frame Rate plus élevée, ce qui nous assure un traçage précis par une poursuite rapide.

5.2 Structure de la boucle principale

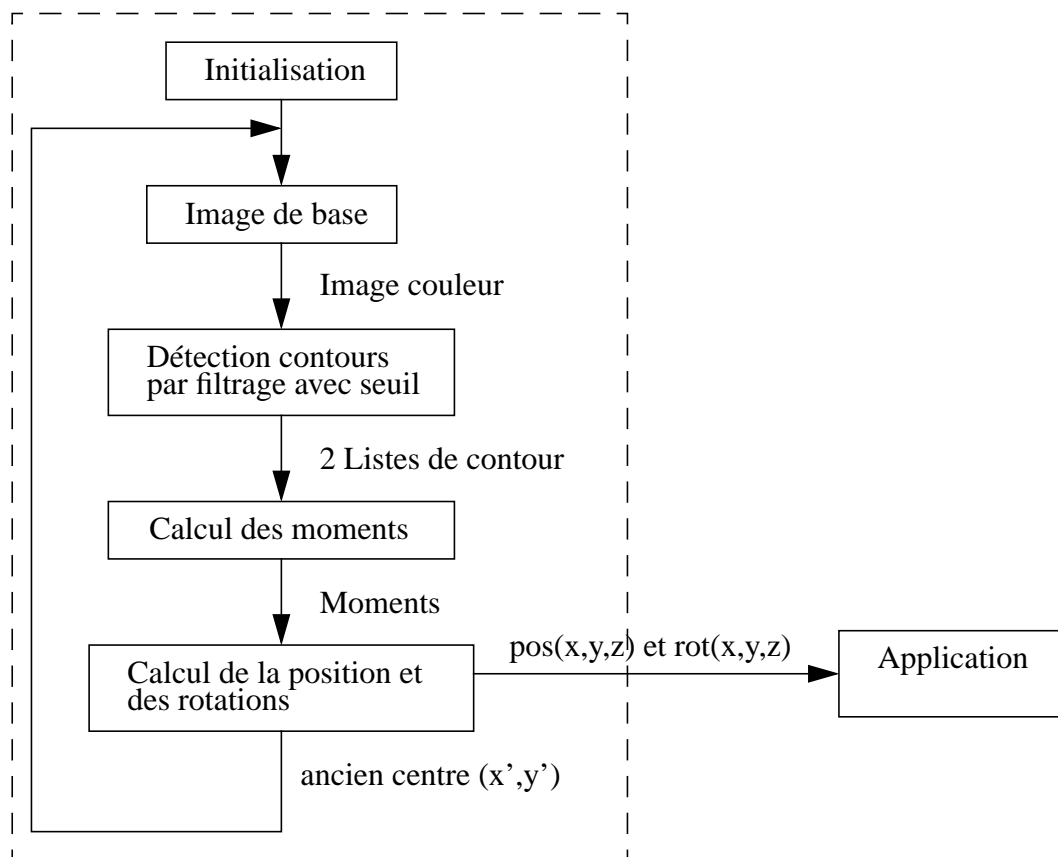


Figure 7. Schéma de la boucle principale

Il est évident que la plupart du temps est perdue pour la détection des contours. A cause de cela on s'est concentré à l'optimisation de cette partie du programme. La partie *Initialisation* contient l'initialisation des paramètres pour la caméra, l'ouverture d'une fenêtre X, l'initialisation des listes de contour et la calibration optionnelle pour la couleur du cercle intérieur. Après on acquiert une *Image* à partir de la caméra et on la passe à la phase de *Détection de Contours*, qui fournit quatre listes de contour, deux pour chaque cercle. Ces quatre listes passent à la phase *Calcul des Moments* d'où on sort avec les différents moments dont on a besoin, et avec lesquels finalement la phase de *Calcul de Position et des Rotations* calcule les paramètres désirés.

Figure 8.

Image perçue par la caméra

Figure 9.

Image des contours filtrés

Figure 10.

Image d'écran avec marqueurs et contours

Pour des raisons de vitesse et de disponibilité du langage, ce projet a été programmé en C, sur une station de travail Indy de Silicon Graphics. Comme caméra on utilise une IndyCam.

5.3 Structure de l'image

Pour l'utilisation de l'IndyCam on utilise la librairie vidéo <dmedia/vl.h>.

Ce qui fait que l'image fournie par la caméra est un tableau unidimensionnel d'indice I. En plus on connaît la résolution en X et en Y (res_x,res_y).

Pour trouver un pixel (X,Y) il suffit de faire les calculs suivants:

$$X = I \text{ mod } \text{res_x}$$
$$Y = I / \text{res_x} \quad \text{division entière}$$

Formule 13. Transformations de coordonnées

Les valeurs des couleurs sont stockées en paquets de quatre valeurs, dont la première a toujours une valeur de 255 et les trois suivantes représentent la couleur du point dans les modèle RGB.

| - | Rouge | Vert | Bleu |
|-----|-------|------|------|
| 255 | R | G | B |

Figure 11. Format d'un pixel de l'image

5.4 Calibration des couleurs

La méthode proposée est très sensible aux conditions extérieures. Surtout la lumière ambiante joue un rôle très important dans la reconnaissance des couleurs. Pour les couleurs du gant et du cercle extérieur on a pas de problème, car on ne travaille qu'avec du seuillage faisant intervenir la moyenne des trois valeurs RGB. Mais pour le cercle intérieur de couleur rouge on est obligé de faire une calibration. Cela permet de connaître les rois valeurs RGB de la couleur dans la situation présente.

5.5 Détection des contours

Comme on peut facilement voir, la partie de Détection de Contours est la partie la plus importante du codage. On pourrait même dire que ce projet se réduit quasiment à trouver et programmer cette partie de manière fortement optimisée. Si on arrive à extraire les deux contours à tout moment il ne faut qu'appliquer les formules données au chapitre 2 et on obtient le résultat cherché.

Pour l'implémentation de l'algorithme, il fallait donc trouver de nombreuses petites solutions astucieuses pour assurer une précision optimale.

- Premièrement il fallait implémenter un test de dépassement sur les bords de l'image. Comme l'image est codée dans une table unidimensionnelle, il faut appliquer les deux formules montrées plus haut. Si le traçage dépasse les bords il faut absolument que le point qui se trouve à la limite soit enregistré dans la liste, car sinon on a des contours qui ne sont théoriquement pas fermés, ce qui donne des résultats erronés. Comme la taille des listes de contour est variable on stocke sa taille à la position 0.

- On a déjà présenté la possibilité d'introduire une liste circulaire comme buffer pour les valeurs de centres, ainsi qu'une limitation de la validité des points trouvés à l'aide d'une fenêtre définie par son centre (normalement (x',y')) et sa longueur de côté (cf. chapitre 4).
- Pour appliquer la fonction de seuil à des couleurs comme le noir ou le blanc on compare la moyenne des trois composantes RGB avec une valeur de seuil. Pour l'appliquer à des couleurs (comme le rouge) il faut tester les trois composantes séparément. Cela augmente le temps de calcul d'un facteur trois environ.
- Dans les boucles de recherche du contour il faut éviter tous les calculs qui peuvent être faits au préalable, donc il faut aussi éviter tous les appels à des fonctions.
- Pour rendre l'utilisation plus facile on affiche à chaque passage les contours, ainsi qu'un réticule, qui indique la position actuelle du centre.
- La fonction de détection de contour a besoin de connaître six paramètres:
 - Deux pointeurs sur des listes, qui contiendront la partie de la liste non-inversée et la partie de la liste inversée.
 - Un paramètre de seuil, *Threshold*, qui fixe le seuil pour la détection des bords.
 - Un paramètre, *Edge*, qui détermine de combien l'algorithme recule en allant sur la prochaine ligne (cf. Chapitre 3).
 - Un paramètre, *Speed*, qui détermine la taille du carré pour le filtrage des valeurs non-admissibles.
 - Et finalement un paramètre *Buffer*, qui indique la taille du buffer qui permet de stabiliser le centre.
 - En plus elle a besoin de la valeur de seuil pour la couleur du cercle intérieur, qui est connue globalement dans le programme.

5.6 Recherche d'un cercle lors d'une perte de synchronisation

Si on perd le cercle, on ne connaît plus de point à l'intérieur de l'objet, l'hypothèse qu'on avait faite n'est plus valable. Le traçage est interrompu.

Pour résoudre ce problème, on considère qu'il n'y a pas d'objets de couleur rouge dans l'image. Donc on peut faire une recherche d'un point de cette couleur. Dès qu'on en a trouvé un, il est utilisé comme nouveau point de départ.

Comme le temps pour parcourir tous les points d'une image est considérablement long, on ne vérifie que un certain nombre de points par ligne avec un espacement fixe.

Cette solution prend toujours beaucoup de temps, mais comme elle est seulement utilisée lors d'une perte de synchronisation cela ne se fait pas trop remarquer.

5.7 Structure de la boucle principale et passage de paramètres

La structure du programme est telle que la boucle principale est une tâche de fond, ce qui est réalisé grâce à l'appel `fork()`.

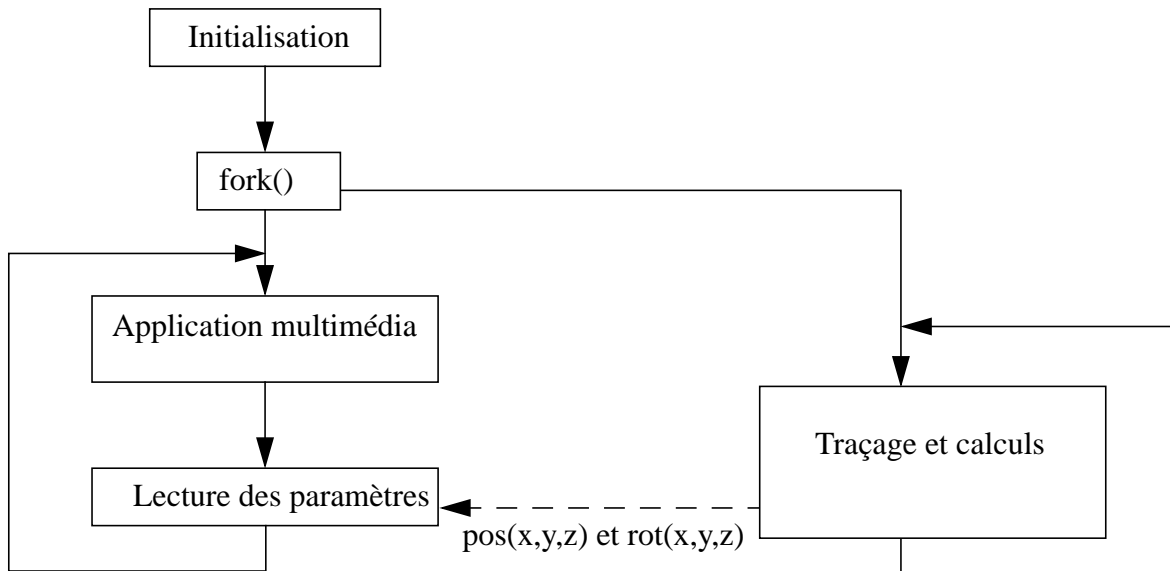


Figure 12.

Passage de paramètres

Le passage de paramètres par des variables ou le passage de valeurs par référence n'est pas possible, car le `fork()` sépare les deux espaces d'adressage du programme principal et de la tâche qui fait les calculs. Alors on a été amené à utiliser les mécanismes de mémoire partagée.

Pour récupérer les paramètres il existe une fonction qui renvoie les valeurs acquises, et qui se charge des mécanismes de mémoire partagée.

5.8 Détection de la position des doigts

Jusqu'à maintenant on a été limité à manipuler des objets en les touchant. Il serait très confortable de pouvoir manipuler des objets en les prenant dans la main.

Pour être capable de faire cela, on doit pouvoir connaître la position des doigts. La solution la plus facile, c'est de ne considérer que deux positions: main ouverte et main fermée.

L'idée consiste à chercher un troisième marqueur qui est situé sur un doigt. Si ce marqueur est visible, alors la main est ouverte, sinon elle est fermée. La première approche était de chercher un cercle de couleur verte (ou bleue), mais comme la caméra fournit des images de mauvaise qualité, ceci n'était pas possible.

Pour la deuxième approche on a choisi à chercher trois bandes noires qui seront collées sur trois doigts. Ainsi on passe à la reconnaissance de motif. Le procédé de recherche est le même que celui décrit en 5.5.

Figure 13.

Le gant finalement utilisé avec les trois marqueurs de grasping.

Cette méthode s'est montrée efficace et très stable. Le seul problème c'est que la main est considérée fermée dès qu'on sort de l'image avec au moins un des marqueurs.

5.9 Montage de la caméra

La caméra devrait être placée à côté du moniteur comme le montre la figure 14.

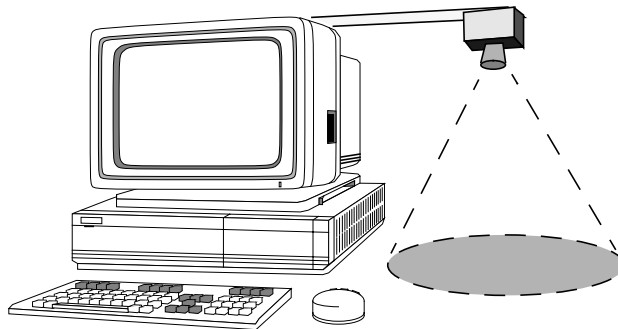


Figure 14.

Montage optimal de la caméra

6 Guide programmeur

6.0 Guide programmeur

6.1 Fonctions mises à disposition

Les fonctions mises à disposition par la librairie libTrace.a sont au nombre de trois, décrites ci-dessous.

Start_Trace(Threshold, Edge, Speed, Buffer, Grasping, Calibrate);

Cette fonction initialise toutes les fonctions et lance la détection comme tâche de fond. Les paramètres sont:

Threshold, seuil pour la détection des bords. (integer)

Edge, points de recul en allant sur la prochaine ligne (cf. Chapitre 3). (integer)

Speed, taille du carré pour le filtrage des valeurs non-admissibles.(integer)

Buffer, indique la taille du buffer, pour stabiliser le centre.(integer)

Grasping, indique, si la détection de la position des doigts aura lieu. (0: pas de détection; 1: détection)

Calibration, indique au programme d'éventuellement faire une calibration de la couleur du cercle interieur.(0: pas de calibration; 1: calibration)

Get_Coord(*x,*y,*z,*rx,*ry,*rz,*grasped);

Cette fonction transmet les paramètres de position (x,y,z) et les paramètres de rotation (rx,ry,rz).

*x, *y, *z, *rx, *ry, *rz, pointeurs sur des variables integer, pour le passage des paramètres par référence.

*grasped, vaut 1, si la main est ouverte, 0 sinon. Si la détection n'a pas lieu, la valeur renvoyée vaut toujours 99.

Stop_Trace();

Cette fonction termine la tâche, ferme la fenêtre et libère les ressources.

6.2 Programme d'exemple

Ce programme trace les mouvements de la main, et affiche la position et les angles de rotation. Pour actualiser les valeurs pressez <Enter> et pour sortir pressez <q>.

Pour lancer la démonstration:

```
demo.x Threshold, Edge, Speed, Buffer
```

Pour des explications des différents paramètres referez vous au point 6.1.

Essayez avec des valeurs de:

Threshold: 120

Edge: 40

Speed: 200

Buffer: 5

```
/*
 * File:      demo.c
 * Author:    Niklaus Hirt
 * Date:      21.11.96
 * Last Modified: 21.01.97
 *
 * Usage:     demo Threshold, Edge, Speed, Buffer
 *
 * Description: demo traces the 3dimensional motions of a hand from pictures
               provided by an Indycam and displays the positionnal and rotational
               values.
               Press <Return> to update
               Press <q> to quit
 */
#include "libTrace.h"

void main(int argc, char *argv[]){

    int Threshold;
    int Edge;
    int Speed;
    int Buffer;
    int Grasp;
    int Calibrate;
    int dx;
    int dy;
    int dz;
    int drx,dry,drz;
    int grasped;
    char test;

    Threshold = atoi(argv[1]);
    Edge = atoi(argv[2]);
    Speed = atoi(argv[3]);
    Buffer = atoi(argv[4]);
    Calibrate = 1;
    Grasp = 1;

    if (argc<5) {
        printf("Utilisation:      \n\n");
        printf("demo Threshold,Edge_Size,Speed,Buffer_Size\n");
        printf("good values are: demo.x 120 80 200 5 \n");
    }
    else{

        Start_Trace(Threshold,Edge,Speed,Buffer,Grasp,Calibrate);

        for(;;){
            Get_Coords(&dx,&dy,&dz,&drx,&dry,&drz,&grasped);

            printf("Vals :(%i,%i,%i), Rot :(%i,%i,%i) Grasp:(%i) \n",dx,dy,dz,drx,dry,drz,grasped);

            test=getchar();
            } while ((char)test!='q');
            printf("Bye Bye \n");
        }
        printf("Have a nice Day \n");
        Stop_Trace();
    }
}
```

6.3 Le fichier de définition libTrace.h

```
/*
 * File:          libTrace.h
 * Author:       Niklaus Hirt
 * Date:        21.11.96
 * Last Modified: 22.01.97
 *
 *
 * Description: Contains all functions to use the positional
 *             and rotational tracking.
 *
 * Utilisation: Just call Start with the following parameters:
 *
 * Threshold : Threshold for edgedetection
 * Edge      : "Size" of the Edges (number of pixels to go back
 *             on the next line to scan)
 * Speed     : Maximum possible distance between two
 *             consecutive centers
 * Buffer     : Size of the stabilization buffer (0..19)
 * Grasp     : Detect Grasping 0->no, 1->yes
 * Calibrate : Red Calibration 0->no, 1->yes
 *
 * As default use : Threshold :110-180
 *                 Edge       :60
 *                 Speed      :200
 *                 Buffer      :5-15 (depends on the speed of
 *                 your machine)
 *                 Grasp      :1 (yes)
 *                 Calibrate  :1 (yes)
 *
 * Then call Get_Coords everytime you want to get the actual
 * parameters.
 */

/* Get actual positional and rotational parameters */
void Get_Coords( int *ex, int *ey, int *ez, int *erx, int *ery, int *erz, int *grsp);
/* Start the contour following process */
void Start_Trace(int T, int E, int S, int B, int G, int Calibrate);
/* Stop the contour following process (ends all tasks and closes the window */
void Stop_Trace();
```

7 Conclusion

7.0 Conclusion

On a réussi à trouver une solution qui permet de suivre les mouvements d'une main à partir d'une image prise par une caméra. On réussit à suivre les déplacements ainsi que les rotations de la main.

On remarque que les choix qui ont été faits pour optimiser le temps de calcul n'ont pas d'effet négatifs sur le confort d'utilisation. En plus, la simplification de l'interface pour programmeurs contenant seulement trois méthodes avec peu de paramètres apporte une grande facilité pour l'implémentation dans des futurs programmes, tout en permettant d'assurer une stabilité maximale dans des conditions externes très différentes.

Malgré tout, il faut respecter quelques conditions pour un fonctionnement optimal:

- Le fond devrait être d'un blanc uniforme.
- On devrait utiliser le gant fourni avec cette documentation.
- Une illumination ambiante assez puissante est préférable. Si celle-ci ne suffit pas, on peut aussi utiliser un spot tout en essayant d'éviter de créer des ombres dans la zone perçue par la caméra.

En ce qui concerne le projet, tous les buts ont été atteints:

- On calcule la position de la main dans l'espace 3D.
- On calcule les rotations autour des trois axes.

En plus l'implémentation de la détection de l'ouverture de la main (grasping), élargit considérablement le domaine d'utilisation en donnant la possibilité de pouvoir saisir des objets lors d'une manipulation.

La version présente assure une utilisation confortable et surtout adaptable à des différentes exigences des utilisateurs. L'interface de programmation est complète, fournissant des routines de démarrage et de terminaison de la détection, ainsi qu'une routine de lecture des valeurs pour les paramètres calculés.

La librairie permet au programmeur d'intégrer facilement toutes ces possibilités dans ses applications. Malgré tout, cela ne présente pas la meilleure solution au problème posé au départ. La détection des angles par exemple est assez restreint. Mais cependant on a trouvé une solution tout à fait satisfaisante en ce qui concerne le compromis entre coût, câblage, précision et facilité à l'utilisation.

Pour le futur on pourrait envisager d'améliorer le traçage des cercles en passant par exemple par la reconnaissance de formes. Avec des caméras de plus haute gamme, fournissant des images plus précises, la détection des bords serait plus efficace. Finalement il faudrait améliorer le dispositif de grasping qui dans cette première approche est assez rudimentaire.

Le possibilités futures sont encore imprévisibles, étant donnée que la reconnaissance de formes est un domaine de recherche jeune, vaste et très prometteur.

8 Bibliographie

8.0 Bibliographie

C. Maggioni, *A Novel Gestural Input Device for Virtual Reality*, 1993

T. Pavlidis, *Structural Pattern Recognition*, 1977 Springer-Verlag

T. Rauber, *Algorithmen in der Computergrafik*, 1993 B.G Teubner Stuttgart

C.A. Lindley, *Practical Image Processing in C*, 1991 John Wiley & Sons