Université de Liège - Faculté des sciences appliquées
Institut Montefiore - Service de télécommunications et d'imagerie

Travail de fin d'études

Génération automatique d'une base de données de silhouettes humaines au moyen d'un avatar humain tridimensionnel

Sébastien Piérard, 3ELIN

Jury : Professeur Marc Van Droogenbroeck
Professeur Bernard Boigelot
Professeur François-Xavier Litt

Promoteur : Professeur Marc Van Droogenbroeck

---

18 mai 2007

Remerciements

C'est avec un réel plaisir que je remercie tous ceux qui par leur enseignement et leurs conseils ont contribué à ce travail de fin d'études. Je remercie tout d'abord le Professeur M. Van Droogenbroeck qui m'a accordé sa confiance et m'a permis de travailler sur un sujet aussi intéressant qu'attrayant. Je remercie également mes autres Professeurs pour tout ce qu'ils m'ont appris. J'adresse mes plus vifs remerciements à Monsieur O. Barnich pour le temps qu'il m'a consacré, son aide précieuse et les perspectives intéressantes qu'il a données à mon travail. Ma gratitude va également à Monsieur S. Jodogne pour ses suggestions fructueuses. Enfin, j'adresse mes pensées sincères à tous ceux qui m'ont soutenu et encouragé pour l'élaboration de ce travail, mes amis et mes parents.

Table des matières

Table des figures

Chapitre 1

Introduction

1.1 Énoncé

Dans le cadre de la conception de systèmes de vidéo-surveillance, il est intéressant de disposer d'un système permettant la détection et la localisation automatique des êtres humains présents devant les caméras. De tels systèmes existent et une partie d'entre-eux sont basés sur la reconnaissance de la forme de silhouettes humaines.

Si on souhaite faire usage d'une technique d'apprentissage automatique pour la reconnaissance de ces silhouettes, on est amené à devoir constituer une base de données comportant un grand nombre d'exemples de telles silhouettes. C'est là un travail fastidieux et répétitif qu'il serait très intéressant de pouvoir automatiser.

Le but de ce travail est d'explorer les possibilités offertes par l'utilisation d'un avatar humain tridimensionnel afin de générer automatiquement une telle base de données ou d'enrichir une base de données de silhouettes réelles.

Après avoir fait une étude bibliographique afin de prendre connaissance des travaux existants dans le domaine, l'étudiant devra choisir une méthode et procéder à la génération de bases de données comportant soit des silhouettes exclusivement synthétiques, soit un mélange de silhouettes synthétiques et de silhouettes réelles. Il devra ensuite étudier l'impact de l'utilisation de ces bases de données sur les performances d'un système de détection de personnes existant qui lui sera fourni.

1.2 Petit historique du travail

Après avoir effectué plusieurs jours de recherche sur plusieurs sites internet proposant des bases de données d'articles scientifiques, nous avons dû nous résoudre à abandonner. En effet, soit la documentation sur le sujet est inexistante, soit la génération de silhouettes est un problème exploré dans d'autres disciplines auxquelles nous n'avons pas pensé. Nous espérions cependant obtenir de la documentation sur quelques sujets ponctuels.

Nous avons donc été contraints d'explorer le problème de manière relativement empirique. Rapidement, quelques questions fondamentales se sont posées. Outre la question de la modélisation de l'avatar, notre attention a été attirée par le nombre important de degrés de liberté du problème. Il nous a semblé impossible de modéliser de manière réaliste un avatar humain avec moins d'une vingtaine de paramètres, auquels il faut ajouter trois degrés de liberté pour la position et trois autres pour l'orientation[1]. Supposons un instant faire varier chaque paramètre par dix pas, il nous faudrait effectuer 10^{26} projections. À titre de comparaison, à raison de 30 projections par seconde, on obtiendrait 10^9 silhouettes par an. Un tel ensemble de silhouettes est donc impossible à obtenir, mais également à stocker et à exploiter.

D'ores et déjà, nous pouvons dire qu'il est nécessaire de réduire la dimensionnalité du problème au maximum. De plus, la pose de l'avatar ainsi que sa projection doivent être rapides.

Réduction de la dimensionnalité

On peut arriver à réduire la dimensionnalité du problème en jouant sur plusieurs tableaux. Voyons tout d'abord s'il est possible d'éliminer quelques degrés de liberté. Si nous nous limitons aux poses «debout», l'avatar ne peut que pivoter autour de l'axe vertical, ce qui limite les degrés de liberté relatifs à l'orientation à un seul. Ensuite, une personne marchant se déplace sur un plan (plus généralement une surface), et non dans l'espace tridimensionnel, ce qui permet également de gagner un degré de liberté supplémentaire. Cependant, au vu de la comparaison précédente, diminuer de trois le nombre de degrés de liberté ne permet pas de rendre le problème envisageable.

Une autre piste, consiste à essayer de limiter le nombre de projections à effectuer sans pour autant éliminer de dimensions au problème. On peut par exemple tenter de ne générer que des silhouettes suffisamment différentes les unes des autres, en utilisant comme critère la distance de Hausdorff.

[1] Le minimum acceptable est une dizaine de paramètres, mais dans ce cas, la modélisation ne permettra pas à l'avatar de simuler la majorité des poses humaines.

On peut également être indifférent à l'échelle et à la position des silhouettes générées pour en limiter le nombre. Ceci est intéressant dans la mesure où la méthode d'apprentissage utilisée pour reconnaître les silhouettes humaines est indépendante de ces variations. Pour finir, une piste plus intéressante consiste à se limiter aux silhouettes relatives aux poses dites «de marche». Dans ce cas, un grand nombre de poses évoquées précédemment seront éliminées.

Compromis temps / généralité

Un autre sujet de préoccupation concerne la projection donnant naissance aux silhouettes. Nous y voyons un compromis entre, d'une part, un mécanisme de projection général, permettant de changer facilement d'avatar, et, d'autre part, un mécanisme optimisé pour le type d'objet tridimensionnel utilisé. Par exemple, le logiciel Pov-Ray permettrait aussi bien de projeter un avatar composé de cylindres et de sphères déformés que de projeter un volume défini par un maillage. La projection, réputée lente avec de tels logiciels utilisant le lancé de rayons, peut être fortement accélérée. En effet, il n'y a nul besoin de déterminer quel objet est situé à l'avant plan et sa couleur pour construire la silhouette[2], et les effets atmosphériques peuvent être désactivés. En revanche, si nous nous restreignons à n'utiliser que des avatars sous forme de maillages, nous pouvons écrire du code spécialisé pour en obtenir la silhouette très rapidement. La vitesse de traitement est donc obtenue au détriment de la généralité de la méthode.

Compromis temps / réalisme

Un deuxième compromis concerne le type d'avatar utilisé. D'un côté, nous avons les maillages souples et réalistes, mais nécessitant la reprojection totale à chaque nouvelle pose. De l'autre côté, les avatars composés de parties rigides sont moins réalistes, mais permettent d'optimiser le rendu. L'idée est de prendre chaque partie séparément, de la déplacer dans l'espace tridimensionnel et de l'orienter de diverses manières pour obtenir un ensemble suffisamment dense de ses silhouettes. Pour chaque pose, placement et orientation de l'avatar, il n'y aurait plus qu'à récupérer les silhouettes partielles correspondant aux diverses parties, la silhouette globale étant obtenue en appliquant l'opérateur booléen *ou* à l'ensemble des silhouettes partielles. Une opération de fermeture serait alors utile pour éliminer les imperfections au niveau des joints. Un raffinement supplémentaire de la méthode consisterait à regrouper récursivement deux silhouettes partielles, on obtiendrait alors un arbre des silhouettes. Ainsi n'y aurait-il plus besoin, par exemple,

[2] aux environs de la ligne 1792 de render.cpp

de recalculer la silhouette d'une jambe à chaque fois qu'elle se trouve dans la même position. De plus, nous pourrions exploiter les symétries humaines pour limiter les parties à projeter, la jambe gauche et la jambe droite étant par exemple identiques à première vue.

Donnons, à titre d'illustration du gain, les complexités algorithmiques des diverses approches. Notons ddl le nombre de degrés de liberté, n le nombre de pas par paramètre, p le nombre de paramètres, m le nombre de morceaux, T le temps total d'exécution, T_a le temps nécessaire à la projection de l'avatar, T_e le temps nécessaire à la projection d'un élément de l'avatar et T_{ou} le temps requis pour fusionner deux silhouettes partielles. On a :

$$\begin{aligned}
\text{Algorithme naïf :} \quad & T = \mathcal{O}(T_a n^{ddl}) \quad \text{avec} \quad ddl = p + 6 = m + 5 \\
\text{Avec segmentation :} \quad & T = \mathcal{O}(T_e n^6 + p T_{ou} n^{ddl}) \\
\text{Avec arborescence :} \quad & T = \mathcal{O}(T_e n^6 + T_{ou} n^{ddl})
\end{aligned}$$

La dernière complexité mérite une petite explication. A chacun des m morceaux sont associées n^6 silhouettes. A chaque fois que l'on fusionne deux ensembles de s_1 et s_2 silhouettes, on obtient $n\frac{s_1 s_2}{n^6}$ silhouettes. Un arbre à m feuilles ayant $m-1$ noeuds internes, on effectuera $m-1=p$ regroupements. Au total, $\frac{n^{6m}}{n^{5p}} = n^{m+5} = n^{ddl}$ silhouettes seront ainsi obtenues. Pour connaître le nombre de *ou* à effectuer, il faut sommer le nombre de silhouettes associées aux noeuds internes. La racine ayant nettement plus de silhouettes que les autres noeuds, nous pouvons les négliger.

Quoiqu'il en soit, nous voyons que la complexité reste exponentielle en le nombre de degrés de liberté. Seul un facteur constant a été gagné, certes important, mais il ne nous permet pas de rendre le problème soluble. Le faible gain s'inscrit donc comme opposant au réalisme.

Plan du travail

Dans un premier temps nous allons nous concentrer sur le mécanisme de projection, en choisissant une modélisation et en mettant en évidence quelques-unes de ses propriétés. Puis, nous ferons un essai d'étude du mécanisme de projection en vue d'évaluer l'impact de changements dans le mode tridimensionnel sur les modifications subies par la projection. L'impact sera mesuré en termes de la distance de Hausdorff. Cette étude sera menée sans jamais faire d'hypothèses restrictives sur l'objet manipulé. Elle nous permettra d'obtenir des bornes sur le nombre d'endroits en lesquels doit être placé un objet pour garantir l'obtention d'une base de données

représentative des diverses silhouettes observables au travers d'une caméra donnée. Elle permettra également d'appréhender la question du choix des orientations et poses à faire prendre à l'avatar.

Globalement, cette approche est déterministe. On essaye d'obtenir toutes les silhouettes, à un seuil de ressemblance près. La caractérisation se fait dans le monde bidimensionnel de la projection. À cette approche s'oppose une démarche aléatoire dans laquelle on essaye de répartir les différentes poses selon une distribution voulue, par tirages aléatoires. La caractérisation se fait alors dans un espace de dimensionnalité importante. Comme principal avantage de cette démarche, on note la facilité triviale à obtenir un nombre voulu de silhouettes, puisqu'il suffit de faire autant de tirages que l'on souhaite de silhouettes. Cependant, la méthode utilise une distribution probabiliste qu'il n'est pas aisé de définir pour obtenir un certain nombre de caractéristiques souhaitées de la base de données. Comparativement, l'approche déterministe ne permet pas de déterminer un seuil de ressemblance tel que l'on obtienne un certain nombre de silhouettes, elle permettrait seulement de l'estimer. Nous verrons comment y parvenir lors de l'étude de l'impact des changements dans le mode tridimensionnel sur les modifications subies par la projection.

Dans un deuxième temps, nous nous concentrerons sur l'implémentation d'un générateur de silhouettes réalistes. Nous choisirons d'utiliser un avatar sous forme de maillage pour son réalisme et un mécanisme de projection optimisé pour garantir une certaine efficacité. Afin d'obtenir les silhouettes, nous utiliserons l'approche aléatoire. Notons à ce sujet que nous pourrions aussi obliger l'avatar à jouer une scène prédéfinie, par exemple marcher en rond. Des logiciels tels que Poser ou le script python Walk.O.Matic pour Blender permettent d'y arriver.

Chapitre 2

Le mécanisme de projection

Sommaire

Dans ce chapitre, nous commençons par étudier le mécanisme de projection d'une caméra et le modélisons avant d'étudier comment obtenir les paramètres du modèle pour une caméra quelconque. Ensuite, nous verrons brièvement comment convertir le modèle en un ensemble de paramètres fréquemment utilisés dans les logiciels de graphisme tridimensionnel.

Deux modèles ont été envisagés. Le modèle de caméra affine n'est valable que si l'objet observé est situé à grande distance de la caméra. Un tel modèle ne semble pas adéquat, car les caméras observant les personnes sont souvent placées dans des pièces et couloirs de petites tailles. Les personnes ne se trouvent donc pas à grande distance des caméras de surveillance. À ce modèle, nous avons préféré le modèle du sténopé.

2.1 Le modèle du sténopé

Nous étudions ici une des modélisations du mécanisme de formation d'image appelée modèle du sténopé[1]. Celui-ci est particulièrement adapté lorsque les conditions de Gauss[2] sont respectées par le système optique de la caméra[3]. Ce mécanisme peut se scinder en trois étapes dont chacune est mathématiquement un changement de base. On distingue en effet 4 repères :

- $\begin{pmatrix} x & y & z \end{pmatrix}$ le repère tridimensionnel *"monde"*
- $\begin{pmatrix} x & y & z \end{pmatrix}^{(c)}$ le repère tridimensionnel centré sur le centre optique de la caméra, dont l'axe des $z^{(c)}$ pointe dans la direction du regard et l'axe des $y^{(c)}$ pointe vers le dessus de la caméra
- $\begin{pmatrix} u & v \end{pmatrix}^{(f)}$ le repère bidimensionnel dans le plan focal
- $\begin{pmatrix} u & v \end{pmatrix}$ le repère bidimensionnel dans le plan *"écran"*

2.1.1 Passage dans le monde de la caméra

La première étape consiste à exprimer les coordonnées d'un point du monde tridimensionnel dans un repère associé à la caméra. Il s'agit de la composition d'une rotation et d'une translation.

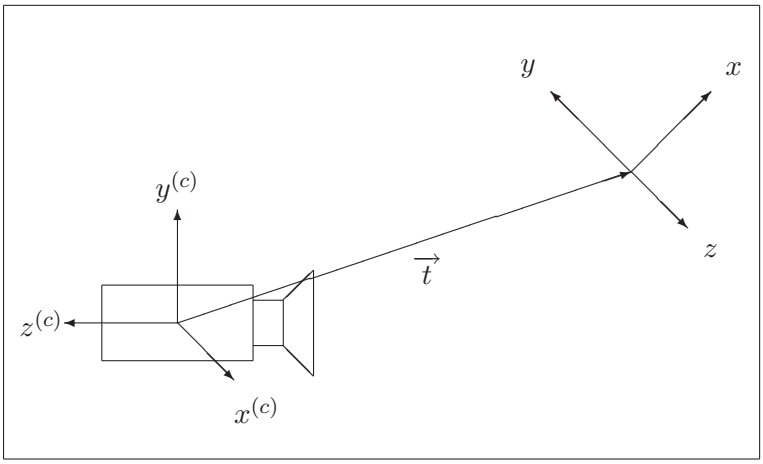


Fig. 2.1 – Passage dans le monde de la caméra.

[1] connu dans le monde anglo-saxon sous l'appellation *pin-hole model*

[2] Les rayons traversent la lentille près du centre, et font un angle faible avec l'axe.

[3] Les objectifs dits *"grand angle"* ne respectent pas ces conditions.

On a :

$$\begin{pmatrix} x^{(c)} \\ y^{(c)} \\ z^{(c)} \end{pmatrix} = R_{3\times 3} \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix} \tag{2.1}$$

R est une matrice exprimant la rotation que doit subir le repère *"monde"* pour coïncider avec le repère *"caméra"*. Les colonnes de R sont les vecteurs de base du repère *"monde"* exprimés dans la base *"caméra"*. Dans le cas où ces deux repères sont orthonormés, les colonnes de R sont orthogonales deux à deux, c'est à dire, $R^T R = I \Leftrightarrow R^{-1} = R^T$. Cette propriété est très intéressante, c'est pourquoi nous allons nous restreindre à des repères orthonormés dans la suite.

2.1.2 Projection sur le plan focal

Ensuite, on peut exprimer la projection du point sur un plan situé à une distance f de la caméra (la distance focale).

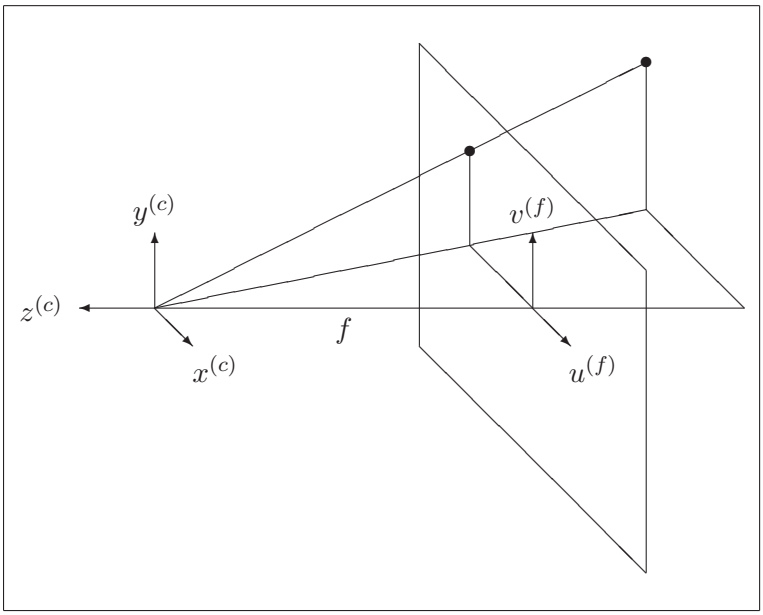


FIG. 2.2 – Projection sur le plan focal.

On a :

$$u^{(f)} = x^{(c)} \frac{f}{z^{(c)}} \qquad\qquad v^{(f)} = y^{(c)} \frac{f}{z^{(c)}} \tag{2.2}$$

2.1.3 Passage aux coordonnées discrètes

La dernière étape exprime le changement de repère dans le plan pour aboutir aux coordonnées sur l'écran.

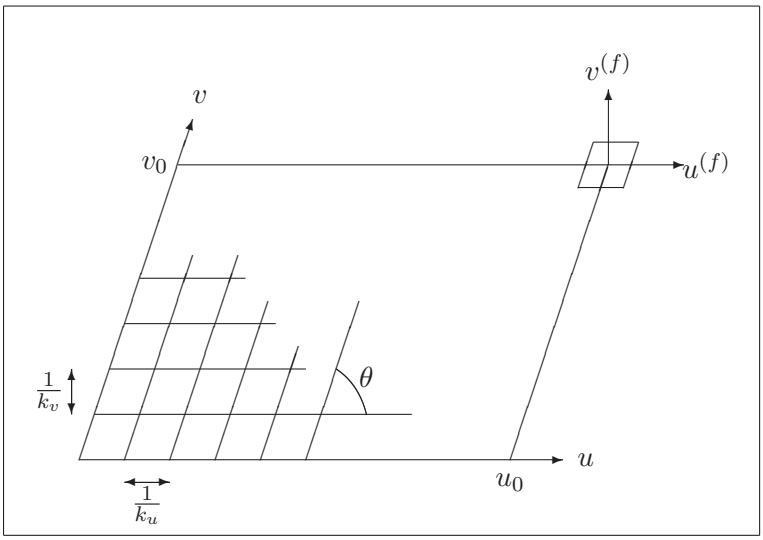


FIG. 2.3 – Passage aux coordonnées discrètes.

On a :

$$u = (u^{(f)} - \frac{v^{(f)}}{\tan\theta})k_u + u_0 \qquad\qquad v = v^{(f)}k_v + v_0 \tag{2.3}$$

Les paramètres k_u et k_v sont appelés *facteurs d'agrandissement* et (u_0, v_0) est la coordonnée de la projection du centre optique sur le plan image.

En posant $s_{uv} = -\frac{k_u}{\tan\theta}$, on obtient :

$$\begin{pmatrix} su \\ sv \\ s \end{pmatrix} = \begin{bmatrix} k_u & s_{uv} & u_0 \\ 0 & k_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} su^{(f)} \\ sv^{(f)} \\ s \end{pmatrix} \tag{2.4}$$

Souvent, la matrice de cellules photosensibles peut être considérée quasi-rectangulaire. Le paramètre s_{uv} est alors négligé et devient nul.

2.1.4 Conclusion

L'utilisation de coordonnées homogènes permet d'écrire le modèle du sténopé sous forme de relation linéaire :

$$\begin{pmatrix} su \\ sv \\ s \end{pmatrix} = \begin{bmatrix} k_u & s_{uv} & u_0 \\ 0 & k_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} & R_{3\times3} & & \begin{matrix} t_x \\ t_y \\ t_z \end{matrix} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{2.5}$$

$$\Leftrightarrow \begin{pmatrix} su \\ sv \\ s \end{pmatrix} = \begin{bmatrix} \alpha_u & S_{uv} & u_0 & 0 \\ 0 & \alpha_v & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} & R_{3\times3} & & \begin{matrix} t_x \\ t_y \\ t_z \end{matrix} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{2.6}$$

$$\Leftrightarrow \begin{pmatrix} su \\ sv \\ s \end{pmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{2.7}$$

avec :

$$\alpha_u = f\ k_u \qquad \alpha_v = f\ k_v \qquad S_{uv} = f\ s_{uv} \tag{2.8}$$

Les équations se simplifient fortement lorsque l'on travaille dans le repère associé à la caméra. En effet, la matrice R devient unité et le vecteur t s'annule. De plus, il est souvent acceptable de négliger le facteur de non orthogonalité s_{uv}. Le mécanisme de projection prend alors la forme suivante :

$$u = u_0 + \alpha_u \frac{x}{z} \qquad v = v_0 + \alpha_v \frac{y}{z} \tag{2.9}$$

2.2 Étalonnage de caméra

2.2.1 Détermination de la matrice de projection

Étalonner la caméra[4] consiste à retrouver la matrice de projection M. La méthode que nous utiliserons se base sur une mise en relation manuelle de points de l'image avec les points correspondants de l'espace tridimensionnel. D'autres méthodes sont possibles.

$$\begin{pmatrix} su \\ sv \\ s \end{pmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

La détermination de la matrice M ne requiert la détermination que de 11 paramètres[5]. En effet, on montrera que le vecteur $(m_{31}m_{32}m_{33})$ est unitaire,

[4] connu dans le monde anglo-saxon sous l'appellation *camera calibration*

[5] cinq correspondances et demi suffisent donc pour déterminer la matrice M

ce qui élimine un degré de liberté. La matrice M n'est donc pas définie à un facteur près, comme les équations suivantes pourraient le faire penser :

$$u = \frac{m_{11}\ x + m_{12}\ y + m_{13}\ z + m_{14}}{m_{31}\ x + m_{32}\ y + m_{33}\ z + m_{34}} \tag{2.10}$$

$$v = \frac{m_{21}\ x + m_{22}\ y + m_{23}\ z + m_{24}}{m_{31}\ x + m_{32}\ y + m_{33}\ z + m_{34}} \tag{2.11}$$

Ces équations peuvent se réécrire sous la forme suivante :

$$\begin{aligned}(m_{31}\ u - m_{11})x + (m_{32}\ u - m_{12})y + (m_{33}\ u - m_{13})z + (m_{34}\ u - m_{14}) &= 0\\ (m_{31}\ v - m_{21})x + (m_{32}\ v - m_{22})y + (m_{33}\ v - m_{23})z + (m_{34}\ v - m_{24}) &= 0\end{aligned}$$

Ces deux équations sont linéaires en les paramètres de la matrice de projection. L'utilisation de n correspondances procure $2n$ équations qui peuvent être mises sous forme matricielle :

$$\begin{bmatrix} x_1 & y_1 & z_1 & 1 & 0 & 0 & 0 & 0 & -u_1x_1 & -u_1y_1 & -u_1z_1 & -u_1 \\ x_2 & y_2 & z_2 & 1 & 0 & 0 & 0 & 0 & -u_2x_2 & -u_2y_2 & -u_2z_2 & -u_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_n & y_n & z_n & 1 & 0 & 0 & 0 & 0 & -u_nx_n & -u_ny_n & -u_nz_n & -u_n \\ 0 & 0 & 0 & 0 & x_1 & y_1 & z_1 & 1 & -v_1x_1 & -v_1y_1 & -v_1z_1 & -v_1 \\ 0 & 0 & 0 & 0 & x_2 & y_2 & z_2 & 1 & -v_2x_2 & -v_2y_2 & -v_2z_2 & -v_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & x_n & y_n & z_n & 1 & -v_nx_n & -v_ny_n & -v_nz_n & -v_n \end{bmatrix} \begin{pmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{14} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{24} \\ m_{31} \\ m_{32} \\ m_{33} \\ m_{34} \end{pmatrix} = 0$$

2.2.2 Factorisation de la matrice de projection

Ensuite, cette matrice est factorisée afin de mettre en évidence les paramètres intrinsèques et les paramètres extrinsèques. Alors que les premiers décrivent la caméra elle-même, les seconds décrivent sa position. Cette écriture permet de donner plus aisément une signification aux différents paramètres, et indique qu'il n'est pas obligatoire de recalibrer complètement une caméra lorsqu'elle est déplacée (seules trois correspondances sont alors nécessaires).

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} = \underbrace{\begin{bmatrix} \alpha_u & S_{uv} & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{paramètres intrinsèques}} \underbrace{\begin{bmatrix} & & t_x \\ & R_{3\times3} & t_y \\ & & t_z \end{bmatrix}}_{\text{paramètres extrinsèques}}$$

Une première façon de mener à bien cette factorisation repose sur le théorème d'orthonormation de Graham-Schmidt. Posons :

$$\begin{array}{lll} m_1 = (m_{11} & m_{12} & m_{13}) \\ m_2 = (m_{21} & m_{22} & m_{23}) \\ m_3 = (m_{31} & m_{32} & m_{33}) \end{array} \qquad \begin{array}{lll} r_1 = (r_{11} & r_{12} & r_{13}) \\ r_2 = (r_{21} & r_{22} & r_{23}) \\ r_3 = (r_{31} & r_{32} & r_{33}) \end{array}$$

La factorisation peut alors se décomposer en deux :

$$\begin{bmatrix} m_1 \\ m_2 \\ m_3 \end{bmatrix} = \begin{bmatrix} \alpha_u & S_{uv} & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_1 \\ r_2 \\ r_3 \end{bmatrix} \qquad \begin{bmatrix} m_{14} \\ m_{24} \\ m_{34} \end{bmatrix} = \begin{bmatrix} \alpha_u & S_{uv} & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

Que l'on réécrit sous la forme suivante :

$$\begin{array}{ll} m_1 = \alpha_u r_1 + S_{uv} r_2 + u_0 r_3 & m_{14} = \alpha_u t_x + S_{uv} t_y + u_0 t_z \\ m_2 = \alpha_v r_2 + v_0 r_3 & m_{24} = \alpha_v t_y + v_0 t_z \\ m_3 = r_3 & m_{34} = t_z \end{array}$$

En utilisant le fait que :

$$|r_1| = |r_2| = |r_3| = 1$$
$$r_1 \bullet r_2 = r_2 \bullet r_3 = r_3 \bullet r_1 = 0$$

Nous obtenons l'algorithme suivant :

```
r3  := m3
v0  := m2 . r3
av  := +/- | m2 - v0 r3 |
r2  := ( m2 - v0 r3 ) / av
u0  := m1 . r3
suv := m1 . r2
au  := +/- | m1 - suv r2 - u0 r3 |
r1  := ( m1 - suv r2 - u0 r3 ) / au
tz  := m34
ty  := ( m24 - v0 tz ) / av
tx  := ( m14 - suv ty - u0 tz ) / au
```

L'autre façon de mener à bien cette factorisation repose sur une décomposition QR. En effet,

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} = \begin{bmatrix} \alpha_u & S_{uv} & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$
$$\Leftrightarrow \begin{bmatrix} m_{33} & m_{23} & m_{13} \\ m_{32} & m_{22} & m_{12} \\ m_{31} & m_{21} & m_{11} \end{bmatrix} = \begin{bmatrix} r_{33} & r_{23} & r_{13} \\ r_{32} & r_{22} & r_{12} \\ r_{31} & r_{21} & r_{11} \end{bmatrix} \begin{bmatrix} 1 & v_0 & u_0 \\ 0 & \alpha_v & S_{uv} \\ 0 & 0 & \alpha_u \end{bmatrix}$$

Cette dernière équation est une décomposition matricielle en un produit d'une matrice orthogonale et d'une matrice triangulaire supérieure, par définition appelée décomposition QR et dont il existe des algorithmes pour la réaliser.

2.3 La caméra virtuelle de Pov-Ray

La description de la caméra dans un logiciel de rendu 3D ne se fait pas nécessairement en spécifiant la matrice de projection. A titre d'exemple, nous voyons ici le cas du logiciel Pov-Ray (version 3.6). Comment passer des paramètres intrinsèques et extrinsèques à ceux du logiciel ?

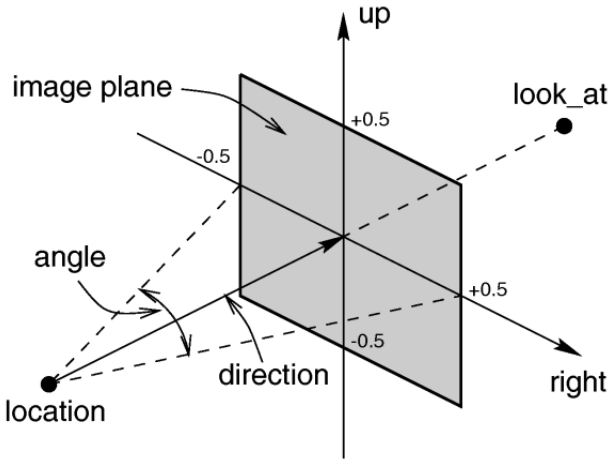


FIG. 2.4 – La caméra de Pov-Ray, *http ://www.povray.org/documentation/*

2.3.1 Description des paramètres extrinsèques

La position de la caméra

La commande `location <x,y,z>` permet d'introduire la position de la caméra. Cette position peut se calculer aisément. En effet, on sait que la caméra est située en $\begin{pmatrix}0 & 0 & 0\end{pmatrix}^{(c)}$. Il suffit donc d'inverser le premier changement de repère évoqué :

$$\begin{pmatrix}0\\0\\0\\1\end{pmatrix} = \begin{bmatrix} & & & t_x\\ & R_{3\times3} & & t_y\\ & & & t_z\\ 0 & 0 & 0 & 1\end{bmatrix} \begin{pmatrix}x\\y\\z\\1\end{pmatrix} \Leftrightarrow R\begin{pmatrix}x\\y\\z\end{pmatrix} + \begin{pmatrix}t_x\\t_y\\t_z\end{pmatrix} = 0$$

$$\Leftrightarrow \begin{pmatrix} x \\ y \\ z \end{pmatrix} = -R^{-1} \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix} = -R^T \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix}$$

La direction de la caméra

La commande `look_at <x,y,z>` permet de spécifier les coordonnées d'un point vers lequel la caméra est pointée. Le lieu de ces points est la droite supportée par l'axe z du repère associé à la caméra. Seule la demi-droite correspondant aux z négatifs nous intéresse, car il s'agit des points situés devant la caméra.

$$\begin{pmatrix} 0 \\ 0 \\ k \\ 1 \end{pmatrix} = \begin{bmatrix} & & & t_x \\ & R_{3\times 3} & & t_y \\ & & & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \Leftrightarrow R \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ k \end{pmatrix}$$

$$\Leftrightarrow \begin{pmatrix} x \\ y \\ z \end{pmatrix} = R^{-1} \left[\begin{pmatrix} 0 \\ 0 \\ k \end{pmatrix} - \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix} \right] = R^T \left[\begin{pmatrix} 0 \\ 0 \\ k \end{pmatrix} - \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix} \right] \text{ avec } k < 0$$

Une autre manière de spécifier la direction suivie par le regard de la caméra consiste à donner à Pov-Ray un vecteur donnant la direction. La syntaxe est `direction <x,y,z>`. La longueur de ce vecteur est la distance focale f, distance entre le foyer de la caméra et le plan de projection. On a :

$$\Leftrightarrow \begin{pmatrix} x \\ y \\ z \end{pmatrix} = R^T \left[\begin{pmatrix} 0 \\ 0 \\ -f \end{pmatrix} - \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix} \right] + R^T \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix} = R^T \begin{pmatrix} 0 \\ 0 \\ -f \end{pmatrix}$$

2.3.2 Description des paramètres intrinsèques

La définition du plan de projection

Le plan de projection est, en Pov-Ray, un losange centré sur la direction du regard. Il se définit grâce aux deux vecteurs $\overrightarrow{right}$ et $\overrightarrow{up}$ qui représentent les côtés de celui-ci. La longueur de ceux-ci définit le ratio largeur/hauteur de l'image produite. Leur sens permet de retourner l'image par symétrie axiale.

$$\overrightarrow{right} = R^T \overrightarrow{\begin{pmatrix} 1 & 0 & 0 \end{pmatrix}} \frac{w_{\text{rendu}}}{\alpha_u} \tag{2.12}$$

$$\overrightarrow{up} = R^T \overrightarrow{\begin{pmatrix} \cos\theta & \sin\theta & 0 \end{pmatrix}} \frac{h_{\text{rendu}}}{\alpha_v \sin\theta} \tag{2.13}$$

$$= R^T \overrightarrow{\begin{pmatrix} \frac{1}{\tan\theta} & 1 & 0 \end{pmatrix}} \frac{h_{\text{rendu}}}{\alpha_v} \tag{2.14}$$

$$= R^T \overrightarrow{\begin{pmatrix} -\frac{S_{uv}}{\alpha_u} & 1 & 0 \end{pmatrix}} \frac{h_{\text{rendu}}}{\alpha_v} \tag{2.15}$$

La distance focale

Ce qui importe, ce sont les angles d'ouverture horizontal et vertical ainsi que le format de rendu. Ceux-ci sont caractérisés par les rapports suivants :

$$\frac{|\overrightarrow{right}|}{f} \quad , \quad \frac{|\overrightarrow{up}|}{f} \quad , \quad \frac{|\overrightarrow{right}|}{|\overrightarrow{up}|}$$

Les grandeurs $|\overrightarrow{right}|$, $|\overrightarrow{up}|$ et f sont ainsi définies à un facteur d'échelle près. Par commodité, on prend $f = 1$.

Chapitre 3

Placement de l'avatar

Sommaire

3.1 Définition du problème

La question fondamentale étudiée dans ce chapitre concerne la détermination des différents points de l'espace tridimensionnel E^3 en lesquels l'avatar doit être placé. En effet, ni déplacer légèrement l'avatar lorsqu'il est loin de la caméra, ni le déplacer amplement lorsqu'il en est proche ne sont des stratégies acceptables. La densité des points en lesquels l'avatar devrait être placé n'est pas uniforme dans l'espace.

Malheureusement, aucun résultat satisfaisant n'a pu être trouvé dans la littérature. La théorie qui suit a été établie pour pallier à ce manque. Elle n'a nullement la prétention d'être parfaite. D'ailleurs, elle donne clairement des résultats sous-optimaux.

Nous nous attacherons à déterminer les positions dans quatre situations différentes. Nous étudierons les conséquences de l'ajout d'une indifférence à la translation ainsi qu'à la dilatation des silhouettes. Le but étant d'obtenir un ensemble de positions où l'avatar doit être placé afin d'obtenir une base de données de silhouettes à la fois riche et la moins redondante possible pour limiter sa taille et le temps requis pour la générer.

3.1.1 Définition de la distance d_π

Pour répondre à la question posée, on commence par définir une distance $d_\pi(\Pi, V, p_1, p_2)$ comme étant la distance euclidienne maximale parcourue par les projections des points du volume V lorsqu'il est translaté de p_1 à p_2, le système projectif étant défini par la fonction de projection $\Pi : \mathbb{R}^3 \to \mathbb{R}^2$. Lorsqu'aucune confusion n'est possible, on la notera simplement $d_\pi(p_1, p_2)$ ou encore d_π.

$$d_\pi(\Pi, V, p_1, p_2) = \max_{p \in V} d_e(\Pi(p + p_1), \Pi(p + p_2)) \tag{3.1}$$

Le fait que d_π soit ou non une distance au sens mathématique dépend de Π et de V. En effet, pour que la propriété de séparation soit remplie, il est nécessaire et suffisant que

$$\forall p_1, p_2 \in \mathbb{R}^3, \forall p \in V, \Pi(p + p_1) = \Pi(p + p_2) \Rightarrow p_1 = p_2 \tag{3.2}$$

Démonstration. Commençons par remarquer que la distance euclidienne est une distance au sens mathématique.

1. positivité

$$\forall p \in V, d_e(\Pi(p+p_1), \Pi(p+p_2)) \geq 0 \text{ par positivité de } d_e$$

$$\Rightarrow \max_{p \in V} d_e(\Pi(p+p_1), \Pi(p+p_2)) \geq 0 \Leftrightarrow d_\pi \geq 0$$

2. symétrie

$$d_\pi(\Pi, V, p_1, p_2) = \max_{p \in V} d_e(\Pi(p+p_1), \Pi(p+p_2))$$

$$d_\pi(\Pi, V, p_2, p_1) = \max_{p \in V} d_e(\Pi(p+p_2), \Pi(p+p_1))$$

$$\text{par symétrie de } d_e \text{ , } d_\pi(\Pi, V, p_1, p_2) = d_\pi(\Pi, V, p_2, p_1)$$

3. séparation
   - $p_1 = p_2 \Rightarrow d_\pi(\Pi, V, p_1, p_2) = 0$

     par séparation de d_e , $p_1 = p_2 \Rightarrow \forall p \in V, d_e(\Pi(p+p_1), \Pi(p+p_2)) = 0$

     $$d_\pi(\Pi, V, p_1, p_2) = \max_{p \in V} 0 = 0$$

   - $d_\pi(\Pi, V, p_1, p_2) = 0 \Rightarrow p_1 = p_2$

     $$\begin{aligned} & d_\pi(\Pi, V, p_1, p_2) = \max_{p \in V} d_e(\Pi(p+p_1), \Pi(p+p_2)) = 0 \\ \Rightarrow \quad & \forall p \in V, d_e(\Pi(p+p_1), \Pi(p+p_2)) = 0 \text{ par positivité de } d_e \\ \Rightarrow \quad & \forall p \in V, \Pi(p+p_1) = \Pi(p+p_2) \text{ par séparation de } d_e \\ \Rightarrow \quad & p_1 = p_2 \text{ par hypothèse} \end{aligned}$$

4. inégalité triangulaire

$$\begin{aligned} & d_\pi(\Pi, V, p_1, p_2) + d_\pi(\Pi, V, p_2, p_3) \\ = \quad & \max_{p \in V} d_e(\Pi(p+p_1), \Pi(p+p_2)) + \max_{p \in V} d_e(\Pi(p+p_2), \Pi(p+p_3)) \\ \geq \quad & \max_{p \in V} (d_e(\Pi(p+p_1), \Pi(p+p_2)) + d_e(\Pi(p+p_2), \Pi(p+p_3))) \\ \geq \quad & \max_{p \in V} d_e(\Pi(p+p_1), \Pi(p+p_3)) \text{ par inégalité triangulaire de } d_e \\ \geq \quad & d_\pi(\Pi, V, p_1, p_3) \end{aligned}$$

□

Sous l'hypothèse 3.2, nous avons donc

$$\forall x, y \in \mathbb{R}^3 : d_\pi(x, y) \in \mathbb{R}^+ \quad \text{(positivité)} \tag{3.3}$$
$$\forall x, y \in \mathbb{R}^3 : d_\pi(x, y) = d_\pi(y, x) \quad \text{(symétrie)} \tag{3.4}$$
$$\forall x, y \in \mathbb{R}^3 : d_\pi(x, y) = 0 \Leftrightarrow x = y \quad \text{(séparation)} \tag{3.5}$$
$$\forall x, y, z \in \mathbb{R}^3 : d_\pi(x, z) \leq d_\pi(x, y) + d_\pi(y, z) \quad \text{(inégalité triang.)} \tag{3.6}$$

Voyons à présent sous quelles conditions la relation 3.2 est satisfaite avec le modèle du sténopé. Ayant négligé le facteur de non-orthogonalité et travaillant dans le repère associé à la caméra, la fonction de projection prend la forme suivante :

$$u = u_0 + \frac{\alpha_u p_x}{p_z} \qquad u' = u_0 + \frac{\alpha_u p'_x}{p'_z} \tag{3.7}$$
$$v = v_0 + \frac{\alpha_v p_y}{p_z} \qquad v' = v_0 + \frac{\alpha_v p'_y}{p'_z} \tag{3.8}$$

Sous quelles conditions la relation 3.2 est-elle satisfaite ? On a :

$$\Pi(p + p_1) = \Pi(p + p_2) \Leftrightarrow \begin{cases} u_0 + \alpha_u \frac{p_x + p_{1x}}{p_z + p_{1z}} = u_0 + \alpha_u \frac{p_x + p_{2x}}{p_z + p_{2z}} \\ v_0 + \alpha_v \frac{p_y + p_{1y}}{p_z + p_{1z}} = v_0 + \alpha_v \frac{p_y + p_{2y}}{p_z + p_{2z}} \end{cases}$$
$$\Leftrightarrow \begin{cases} \alpha_u = 0 \text{ ou } \frac{p_x + p_{1x}}{p_z + p_{1z}} = \frac{p_x + p_{2x}}{p_z + p_{2z}} \\ \alpha_v = 0 \text{ ou } \frac{p_y + p_{1y}}{p_z + p_{1z}} = \frac{p_y + p_{2y}}{p_z + p_{2z}} \end{cases}$$

Si la matrice de projection M n'est pas dégénérée, $\alpha_u \neq 0$ et $\alpha_v \neq 0$. Alors,

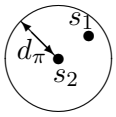
$$\Pi(p + p_1) = \Pi(p + p_2) \Leftrightarrow (p + p_1) \propto (p + p_2) \tag{3.9}$$

Tenant compte que cette relation doit être valable pour tout p de V, on conclut que soit tous les points de V sont alignés, soit $p_1 = p_2$. La seule façon que tous les points d'un parallélépipède soient alignés est que deux de ses dimensions soient nulles. Autrement dit, d_π sera une distance au sens mathématique si la projection est non dégénérée, et que V n'est pas un segment de droite. Dans le cas contraire, V peut être aligné vers la caméra et toute translation radiale de V ne produira aucun déplacement de la projection.

3.1.2 Lien avec la distance de Hausdorff d_H

La distance d_π borne la distance parcourue par le bord de la silhouette de V. En effet, considérons deux silhouettes S_1 et S_2 du même avatar placé en des endroits différents. A chaque point de V sont associés, par projection, un point s_1 de S_1 et un point s_2 de S_2. Si la distance euclidienne les séparant est au maximum d_π alors, $\forall s1 \in S1$ et $\forall s2 \in S2$,

$d_e(s_1, s_2) \leq d_\pi$
$S_2 \subseteq S_1 \oplus D_{d_\pi}$



$d_e(s_2, s_1) \leq d_\pi$
$S_1 \subseteq S_2 \oplus D_{d_\pi}$

On a ainsi,

$$d_H(S_1, S_2) = min\left\{r \mid S_1 \subseteq S_2 \bigoplus D_r, S_2 \subseteq S_1 \bigoplus D_r\right\} \leq d_\pi \tag{3.10}$$

Autrement dit, la distance de HAUSDORFF caractérisant la différence entre les deux silhouettes S_1 et S_2 est bornée par la distance maximale parcourable par les projections des points de V.

$$d_H(\Pi(V, p1), \Pi(V, p2)) \leq d_\pi(\Pi, V, p1, p2) \tag{3.11}$$

où

$$\Pi(V, p) = \bigcup_{p1 \in V} \Pi(p + p1) \tag{3.12}$$

3.1.3 Construction d'un ensemble S de positions

On cherche à construire un sous-ensemble S de points de R^3 tel que, si on se donne une tolérance de ε pixels,

$$\forall x \in \mathbb{R}^3, \exists y \in S : d_H(\Pi(V + x), \Pi(V + y)) \leq \varepsilon \tag{3.13}$$

Nous avons vu que cette condition sera remplie notamment si

$$\forall x \in \mathbb{R}^3, \exists y \in S : d_\pi(\Pi, V, x, y) \leq \varepsilon \tag{3.14}$$

Dans ce but, on va générer un ensemble de surfaces tel que tout point de $\mathbb{R}^3$ soit distant au plus de $\frac{\varepsilon}{3}$ d'une surface. Au sein de chaque surface, on va construire un ensemble de courbes tel que tout point de la surface soit distant au plus de $\frac{\varepsilon}{3}$ d'une de ces courbes. Enfin, sur chaque courbe, on choisira un ensemble de points garantissant que tout point de la courbe soit au maximum éloigné de $\frac{\varepsilon}{3}$ d'un des points de l'ensemble. L'union de tous les ensembles de points choisis formera le sous-ensemble de $\mathbb{R}^3$ recherché.

Démonstration. On peut trouver un chemin, allant de tout point de $\mathbb{R}^3$ à un des points choisis, tel qu'il se décompose en les trois étapes suivantes :

1. aller du point à la surface la plus proche ;
2. se déplacer sur la surface vers la courbe la plus proche ;
3. se déplacer sur la courbe vers le point le plus proche.

La distance parcourue dans chacune de ces étapes est au maximum de $\frac{\varepsilon}{3}$. Par la propriété d'inégalité triangulaire, on a ainsi construit, pour tout point de $\mathbb{R}^3$, un chemin de longueur inférieure ou égale à ε, le menant à un des points de l'ensemble construit. □

Il nous reste à voir comment construire un ensemble de surfaces tel que décrit ci-dessus. La construction des ensembles de courbes et de points est un problème analogue. La construction d'un ensemble de surfaces distantes les unes des autres de $2\frac{\varepsilon}{3}$ est incorrecte. En revanche, la création d'un ensemble de surfaces distantes les unes des autres de $\frac{\varepsilon}{3}$ dont on ne garde qu'une surface sur deux est une bonne manière de procéder.

Remarque. La façon de procéder décrite ci-dessus n'utilise que les propriétés d'inégalité triangulaire et de symétrie de la distance. En pratique, pour d_π, il n'y a pas besoin de montrer 3.2 pour que cette méthode soit applicable.

Dans la suite, nous ne ferons plus la distinction entre surfaces, courbes et points. En effet, nous considérerons les courbes comme étant l'intersection de deux "familles" de surfaces et les points comme étant obtenus par l'intersection de trois "familles" de surfaces. Nous parlerons ainsi exclusivement de "familles" de surfaces.

La méthode employée dans la suite peut se résumer comme suit. On choisit arbitrairement une direction $\begin{pmatrix} dx & dy & dz \end{pmatrix}$ de déplacement du volume V. Les différentes surfaces seront construites itérativement. Pour tout point $\begin{pmatrix} x & y & z \end{pmatrix}$ de la surface, on va rechercher une borne de la modification subie (donnée par la distance d_π) par la silhouette du volume V lorsqu'il est déplacé du point $\begin{pmatrix} x & y & z \end{pmatrix}$ au point $\begin{pmatrix} x+dx & y+dy & z+dz \end{pmatrix}$:

$$d_\pi \left(\Pi, V, \begin{pmatrix} x \\ y \\ z \end{pmatrix}, \begin{pmatrix} x+dx \\ y+dy \\ z+dz \end{pmatrix} \right) \leq F(dx, dy, dz) \tag{3.15}$$

Souhaitant garantir que la nouvelle surface construite ne soit pas distante de plus de $\frac{\varepsilon}{3}$ de la surface courante,

$$d_\pi \left(\Pi, V, \begin{pmatrix} x \\ y \\ z \end{pmatrix}, \begin{pmatrix} x+dx \\ y+dy \\ z+dz \end{pmatrix} \right) \leq \frac{\varepsilon}{3} \tag{3.16}$$

On choisira donc le point de la nouvelle surface $\begin{pmatrix} x+dx & y+dy & z+dz \end{pmatrix}$ en adaptant la norme de $\begin{pmatrix} dx & dy & dz \end{pmatrix}$ pour qu'elle soit telle que $F(dx, dy, dz)$ vaille $\frac{\varepsilon}{3}$.

3.1.4 Hypothèses de travail

Pour définir complètement la distance d_π, il faut spécifier la relation de projection Π et le volume V avec lesquels nous travaillons.

Afin de simplifier au maximum l'expression de la fonction Π, on va travailler dans le monde associé à la caméra. Il ne s'agit nullement d'une restriction, un changement de base permettant aisément d'obtenir les positions exprimées dans le repère mondial. Ensuite, on va négliger le paramètre S_{uv} de la caméra. Souvent, la valeur de ce paramètre sera faible et une telle approximation n'aura pas de conséquences.

En ce qui concerne V, un choix pratique est une boîte parallélépipédique centrée à l'origine et de dimensions $2\Delta x \times 2\Delta y \times 2\Delta z$. Ses dimensions seront choisies de telle manière que l'avatar soit complètement inscrit dans celle-ci.[1] Ce choix a notamment pour conséquence que la théorie décrite ci-dessous est générale, et peut s'appliquer à d'autres avatars que celui utilisé dans le cadre de ce travail.

On notera (x, y, z) le centre de la boîte dans sa position de référence et (p_x, p_y, p_z) un de ses points. De même, (x', y', z') sera le centre de la boîte dans sa position testée et (p'_x, p'_y, p'_z) son point correspondant à (p_x, p_y, p_z). Enfin, (dx, dy, dz) est le vecteur déplacement entre les deux positions.

$$p_x = x + \delta x \quad \text{et} \quad p'_x = p_x + dx \quad \text{avec} \quad -\Delta x \leq \delta x \leq \Delta x \tag{3.17}$$
$$p_y = y + \delta y \quad \text{et} \quad p'_y = p_y + dy \quad \text{avec} \quad -\Delta y \leq \delta y \leq \Delta y \tag{3.18}$$
$$p_z = z + \delta z \quad \text{et} \quad p'_z = p_z + dz \quad \text{avec} \quad -\Delta z \leq \delta z \leq \Delta z \tag{3.19}$$

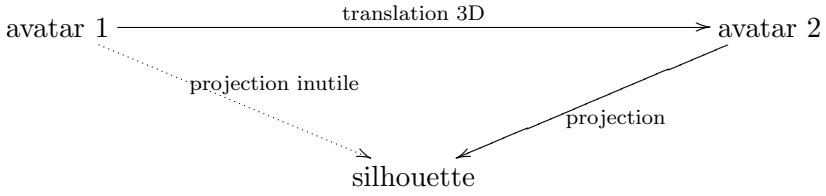
Ayant négligé le facteur de non-orthogonalité, le mécanisme de projection devient symétrique par rapport aux plans $x = 0$ et $y = 0$. On peut se contenter de rechercher les positions dans $x \geq 0$, $y \geq 0$, les autres étant générées par symétrie. De plus, l'avatar ne sera visible que s'il est situé devant la caméra. On se restreint un peu plus en imposant que $z + \Delta z \leq 0$.

$$x \geq 0 \qquad y \geq 0 \qquad z \leq -\Delta z \tag{3.20}$$

[1] V est connu sous le nom de *bounding box* de l'avatar.

3.2 Premier ensemble de positions

Considérons, pour commencer, le problème de base où l'on ne tient pas compte d'une invariance à la translation ni à la dilatation.



La fonction de projection est la suivante :

$$u = u_0 + \frac{\alpha_u p_x}{p_z} \qquad u' = u_0 + \frac{\alpha_u p'_x}{p'_z} \tag{3.21}$$

$$v = v_0 + \frac{\alpha_v p_y}{p_z} \qquad v' = v_0 + \frac{\alpha_v p'_y}{p'_z} \tag{3.22}$$

La distance d_π prend la forme suivante :

$$d_\pi = \max_{p \in V+x} \sqrt{(u'-u)^2 + (v'-v)^2} \tag{3.23}$$

$$= \max_{p \in V+x} \sqrt{\left(\alpha_u \left(\frac{p'_x}{p'_z} - \frac{p_x}{p_z}\right)\right)^2 + \left(\alpha_v \left(\frac{p'_y}{p'_z} - \frac{p_y}{p_z}\right)\right)^2} \tag{3.24}$$

3.2.1 Première famille de surfaces

Étudions la modification de silhouette subie par une translation selon l'axe des x. Les surfaces seront obtenues itérativement. On a $dx \geq 0$, $dy = 0$ et $dz = 0$.

$$u = u_0 + \frac{\alpha_u p_x}{p_z} \qquad u' = u_0 + \frac{\alpha_u (p_x + dx)}{p_z} \tag{3.25}$$

$$v = v_0 + \frac{\alpha_v p_y}{p_z} \qquad v' = v_0 + \frac{\alpha_v p_y}{p_z} \tag{3.26}$$

$$\begin{aligned} d_\pi &= \max \sqrt{(u'-u)^2 + (v'-v)^2} \\ &= \max \left| \frac{\alpha_u dx}{p_z} \right| = \frac{|\alpha_u||dx|}{\min |p_z|} = \frac{|\alpha_u| dx}{-(z + \Delta z)} = \frac{\varepsilon}{3} \end{aligned} \tag{3.27}$$

$$x_n = x_{n-1} + dx = x_{n-1} - \frac{\varepsilon}{3} \frac{z + \Delta z}{|\alpha_u|} \tag{3.28}$$

La résolution de cette équation aux récurrences donne, si $x_0 = 0$,

$$x_n = -\frac{\varepsilon}{3} \frac{n}{|\alpha_u|} (z + \Delta z) = a(z + \Delta z) \qquad \text{avec} \qquad a \leq 0 \tag{3.29}$$

3.2.2 Deuxième famille de surfaces

Étudions la modification de silhouette subie par une translation selon l'axe des y. Les surfaces seront obtenues itérativement. On a $dx = 0$, $dy \geq 0$ et $dz = 0$. En procédant de manière analogue à ce qui a été fait pour la première famille, on obtient si $y_0 = 0$,

$$y_n = -\frac{\varepsilon}{3}\frac{n}{|\alpha_v|}(z + \Delta z) = b(z + \Delta z) \qquad \text{avec} \qquad b \leq 0 \tag{3.30}$$

3.2.3 Troisième famille de surfaces

Étudions la modification de silhouette subie par une translation selon l'axe des z, mais sous les contraintes $x = a(z + \Delta z)$ et $y = b(z + \Delta z)$ avec a et b négatifs. Les surfaces seront également obtenues itérativement. On a $dx = a\ dz$, $dy = b\ dz$ et $dz \leq 0$.

$$\begin{aligned} u' - u &= \alpha_u \left(\frac{p'_x}{p'_z} - \frac{p_x}{p_z}\right) = \alpha_u \frac{p'_x p_z - p_x p'_z}{p'_z p_z} \\ &= \alpha_u \frac{(p_x + a\ dz)p_z - p_x(p_z + dz)}{p'_z p_z} = \alpha_u \frac{a\ dz\ p_z - p_x\ dz}{p'_z p_z} \\ &= dz\ \alpha_u \frac{a\ p_z - p_x}{p'_z p_z} = dz\ \alpha_u \frac{a\ \delta z - a\ \Delta z - \delta x}{(z + \delta z + dz)(z + \delta z)} \end{aligned} \tag{3.31}$$

$$\max|u' - u| \leq -dz\ |\alpha_u| \frac{\max|a\ \delta z - a\ \Delta z - \delta x|}{\min|(z + \delta z + dz)(z + \delta z)|} \tag{3.32}$$

Pour maximiser le numérateur, il faut que les signes de chaque terme soient identiques et que δx et δz soient d'amplitude maximale. Le deuxième terme étant positif, on obtient $\begin{pmatrix}\delta x & \delta z\end{pmatrix} = \begin{pmatrix}-\Delta x & -\Delta z\end{pmatrix}$. Minimiser le dénominateur revient à prendre un δz de signe opposé à celui de z et de $z + dz$. D'où $\delta z = \Delta z$. On a donc,

$$\max|u' - u| \leq -dz\ |\alpha_u| \frac{\Delta x - 2a\Delta z}{(z + \Delta z + dz)(z + \Delta z)} \tag{3.33}$$

De façon similaire, on obtient :

$$\max|v' - v| \leq -dz\ |\alpha_v| \frac{\Delta y - 2b\Delta z}{(z + \Delta z + dz)(z + \Delta z)} \tag{3.34}$$

Il s'en suit que,

$$\begin{aligned} d_\pi &= \max\sqrt{(u' - u)^2 + (v' - v)^2} \\ &\leq \sqrt{(\max|u' - u|)^2 + (\max|v' - v|)^2} \\ &\leq |dz| \frac{\sqrt{\alpha_u^2 (\Delta x - 2a\Delta z)^2 + \alpha_v^2 (\Delta y - 2b\Delta z)^2}}{(z + \Delta z)(z + \Delta z + dz)} \\ &\leq |dz| \frac{K(a,b)}{(z + \Delta z + dz)(z + \Delta z)} = \frac{\varepsilon}{3} \end{aligned} \tag{3.35}$$

Et donc,

$$z_n = z_{n-1} + dz = z_{n-1} - \frac{(z_{n-1} + \Delta z)^2}{z_{n-1} + \Delta z + \frac{K(a,b)}{\varepsilon/3}} \tag{3.36}$$

dz devant être négatif, nous avons la condition limite suivante :

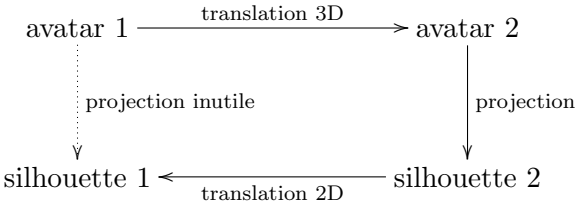
$$z > -\frac{K(a,b)}{\varepsilon/3} - \Delta z \tag{3.37}$$

Il est montré en annexe B que la solution de cette équation aux récurrences est donnée par :

$$z_n = z_0 - \frac{n(z_0 + \Delta z)^2}{n(z_0 + \Delta z) + C} = \qquad \text{avec} \qquad C = \frac{K(a,b)}{\varepsilon/3} \tag{3.38}$$

3.3 Deuxième ensemble de positions

Le but recherché ici étant de minimiser le nombre de projections à effectuer, il est possible de raffiner le résultat précédent. La projection d'un objet étant plus lente qu'une translation de silhouette, nous souhaiterons faire abstraction de la position lors de l'évaluation de la différence entre deux silhouettes.



A cette fin, modifions la relation de projection pour que les silhouettes soient toujours centrées, c'est à dire que les silhouettes sont translatées pour que le point central de V soit projeté à l'origine.

$$u = \left(u_0 + \alpha_u \frac{p_x}{p_z}\right) - \left(u_0 + \alpha_u \frac{x}{z}\right) = \alpha_u \left(\frac{p_x}{p_z} - \frac{x}{z}\right) \tag{3.39}$$

$$v = \left(v_0 + \alpha_v \frac{p_y}{p_z}\right) - \left(v_0 + \alpha_v \frac{y}{z}\right) = \alpha_v \left(\frac{p_y}{p_z} - \frac{y}{z}\right) \tag{3.40}$$

3.3.1 Première famille de surfaces

Étudions la modification de silhouette subie par une translation selon l'axe des x. Les surfaces seront obtenues itérativement. On a $x_0 = 0$, $dx \geq 0$, $dy = 0$ et $dz = 0$.

$$\begin{aligned} u' - u &= \alpha_u \left(\frac{p'_x}{p'_z} - \frac{p_x}{p_z} + \frac{x}{z} - \frac{x'}{z'} \right) = \alpha_u dx \left(\frac{1}{p_z} - \frac{1}{z} \right) = -\frac{\alpha_u dx \delta z}{z(z+\delta z)} \\ v' - v &= \alpha_v \left(\frac{p'_y}{p'_z} - \frac{p_y}{p_z} + \frac{y}{z} - \frac{y'}{z'} \right) = 0 \end{aligned}$$

$$d_\pi = \max \sqrt{(u'-u)^2 + (v'-v)^2} = \max \frac{|\alpha_u| dx |\delta z|}{z(z+\delta z)} = \frac{|\alpha_u| dx \Delta z}{z(z+\Delta z)} = \frac{\varepsilon}{3} \tag{3.41}$$

$$x_n = x_{n-1} + dx = x_{n-1} + \frac{\varepsilon/3}{|\alpha_u| \Delta z} z(z+\Delta z) \tag{3.42}$$

$$x_n = \frac{\varepsilon}{3} \frac{n}{|\alpha_u| \Delta z} z(z+\Delta z) = cz\ (z+\Delta z) \qquad \text{avec} \qquad c = -\frac{a}{\Delta z} \geq 0 \tag{3.43}$$

3.3.2 Deuxième famille de surfaces

Étudions la modification de silhouette subie par une translation selon l'axe des y. Les surfaces seront obtenues itérativement. On a $y_0 = 0$, $dx = 0$, $dy \geq 0$ et $dz = 0$. Par analogie avec la première famille,

$$y_n = \frac{\varepsilon}{3} \frac{n}{|\alpha_v| \Delta z} z(z+\Delta z) = d\ z(z+\Delta z) \qquad \text{avec} \qquad d = -\frac{b}{\Delta z} \geq 0 \tag{3.44}$$

3.3.3 Troisième famille de surfaces

Étudions la modification de silhouette subie par une translation selon l'axe des z, mais avec les contraintes $x = cz(z+\Delta z)$ et $y = dz(z+\Delta z)$. Les surfaces seront également obtenues itérativement. On a $dz \leq 0$.

$$\begin{aligned} \frac{u'-u}{\alpha_u} &= \left(\frac{p'_x}{p'_z} - \frac{x'}{z'} \right) - \left(\frac{p_x}{p_z} - \frac{x}{z} \right) \\ &= \left(\frac{x' + \delta x}{z' + \delta z} - \frac{x'}{z'} \right) - \left(\frac{x + \delta x}{z + \delta z} - \frac{x}{z} \right) \\ &= \frac{z'\delta x - x'\delta z}{z'(z'+\delta z)} - \frac{z\delta x - x\delta z}{z(z+\delta z)} \\ &= \frac{\delta x - c(z'+\Delta z)\delta z}{z'+\delta z} - \frac{\delta x - c(z+\Delta z)\delta z}{z+\delta z} \\ &= (\delta x - c\delta z \Delta z) \left(\frac{1}{z'+\delta z} - \frac{1}{z+\delta z} \right) - c\delta z \left(\frac{z'}{z'+\delta z} - \frac{z}{z+\delta z} \right) \\ &= \frac{(\delta x - c\delta z \Delta z)(z-z') + c\delta z^2 (z-z')}{(z'+\delta z)(z+\delta z)} \\ &= dz \frac{c\delta z \Delta z - c\delta z^2 - \delta x}{(z+\delta z+dz)(z+\delta z)} \end{aligned} \tag{3.45}$$

$$\max\left|u'-u\right| \leq -dz|\alpha_u|\frac{\max|c\delta z\Delta z - c\delta z^2 - \delta x|}{\min|(z+\delta z+dz)(z+\delta z)|} \tag{3.46}$$

Pour maximiser le numérateur, il faut que les signes de chaque terme soient identiques et que δx et δz soient d'amplitude maximale. Le deuxième terme étant négatif, on obtient $\begin{pmatrix}\delta x & \delta z\end{pmatrix} = \begin{pmatrix}\Delta x & -\Delta z\end{pmatrix}$. Minimiser le dénominateur revient à prendre un δz de signe opposé à celui de z et de $z+dz$. D'où $\delta z = \Delta z$. On a donc,

$$\max\left|u'-u\right| \leq -dz|\alpha_u|\frac{\Delta x - 2a\Delta z}{(z+\Delta z+dz)(z+\Delta z)} \tag{3.47}$$

et par un développement similaire,

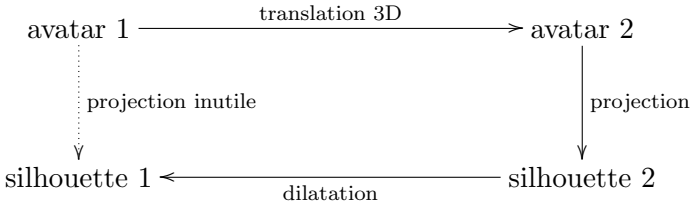
$$\max\left|v'-v\right| \leq -dz|\alpha_v|\frac{\Delta y - 2b\Delta z}{(z+\Delta z+dz)(z+\Delta z)} \tag{3.48}$$

Ces résultats sont identiques à ceux obtenus au point 3.2.3. Le reste du développement est identique et on conclut en rappelant que

$$z_n = z_0 - \frac{n(z_0+\Delta z)^2}{n(z_0+\Delta z)+C} = \qquad \text{avec} \qquad C = \frac{K(a,b)}{\varepsilon/3} \tag{3.49}$$

3.4 Troisième ensemble de positions

Dans les sections précédentes, nous avons construit un ensemble de positions et l'avons réduit en tenant compte de l'invariance vis-à-vis de la position. Dans cette section, on va tenter l'ajout d'une invariance à l'échelle.



Une question qui se pose concerne l'expression de l'invariance à la dilatation. Faut-il mettre les deux silhouettes à une taille donnée avant de les comparer ou est-il préférable de ne changer la taille que d'une seule ? Si l'on considère que le but est de générer un ensemble de silhouettes à partir desquelles il est possible par dilatation de retrouver toutes les silhouettes à une certaine tolérance près, la deuxième proposition est à choisir. On introduit donc un nouveau paramètre σ (par facilité, on travaille avec u_0 et v_0 nuls) :

$$\frac{u'-u}{\alpha_u} = \frac{p'_x}{p'_z} - \sigma\frac{p_x}{p_z} \tag{3.50}$$

$$\frac{v' - v}{\alpha_v} = \frac{p'_y}{p'_z} - \sigma \frac{p_y}{p_z} \tag{3.51}$$

Le choix du facteur σ n'est pas trivial. On peut par exemple imaginer de calculer la distance d_π séparant deux silhouettes en fonction de ce paramètre, puis de le choisir tel qu'il la minimise. Cette technique appliquée aux deux premières familles donne $\sigma = 1$, mais son application à la troisième famille complique considérablement les calculs et semble en pratique inutilisable.

Une autre stratégie consiste à considérer la taille d'une projection comme étant inversement proportionnelle à l'éloignement de l'objet. Celle-ci vaut, dans notre cas, où V est un parallélépipède rectangle, en moyenne z. Choisir pour σ la valeur $\frac{z}{z'}$ devrait diminuer d_π, mais sans pour autant la minimiser. Remarquons que dans le cas des deux premières familles, $z' = z \Rightarrow \sigma = 1$, ce qui correspond à l'optimum. Cette stratégie a l'avantage de ne pas compliquer excessivement les calculs.

$$\sigma = \frac{z}{z'} \tag{3.52}$$

Il reste à étudier la modification de la troisième famille de surfaces.

$$\begin{aligned}
u' - u &= \alpha_u \left(\frac{p'_x}{p'_z} - \frac{z}{z'} \frac{p_x}{p_z} \right) = \alpha_u \frac{p'_x z' p_z - p_x z p'_z}{z' p'_z p_z} \\
&= \alpha_u \frac{(p_x + a\ dz)(z + dz)p_z - p_x z (p_z + dz)}{z' p'_z p_z} \\
&= \alpha_u \frac{a\ dz\ (z + dz)p_z + p_x\ dz p_z - p_x\ dz z}{z' p'_z p_z} \\
&= \alpha_u dz \frac{a(z + dz)(z + \delta z) + (a(z + \Delta z) + \delta x)\delta z}{z' p'_z p_z} \\
&= \alpha_u dz \frac{az(z + dz) + a((z + dz) + (z + \Delta z))\delta z + \delta x \delta z}{z' p'_z p_z}
\end{aligned} \tag{3.53}$$

$$\max |u' - u| \leq -dz\ |\alpha_u| \frac{\max |\overbrace{az(z + dz)}^{\leq 0} + \overbrace{a((z + dz) + (z + \Delta z))}^{\geq 0} \delta z + \delta x \delta z|}{\min |z' p'_z p_z|}$$

Le numérateur sera maximisé pour $\begin{pmatrix} \delta x & \delta z \end{pmatrix} = \begin{pmatrix} \Delta x & -\Delta z \end{pmatrix}$. La minimisation du dénominateur se fait en prenant $\delta z = \Delta z$. On a donc,

$$\begin{aligned}
\max |u' - u| &\leq dz\ |\alpha_u| \frac{-az(z + dz) + a(2z + dz + \Delta z)\Delta z + \Delta x \Delta z}{(z + dz)(z + \Delta z)(z + dz + \Delta z)} \\
&\leq |\alpha_u| \frac{dz}{z'} \left(\frac{\Delta x \Delta z - 2az(z + dz)}{(z + \Delta z)(z + dz + \Delta z)} + a \right)
\end{aligned} \tag{3.54}$$

Symétriquement, on a :

$$\max|v'-v| \leq |\alpha_v|\frac{dz}{z'}\left(\frac{\Delta y\Delta z - 2bz(z+dz)}{(z+\Delta z)(z+dz+\Delta z)} + b\right) \tag{3.55}$$

Et donc,

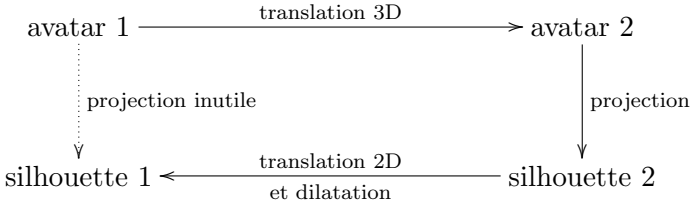
$$d_\pi \leq dz\frac{\sqrt{\begin{array}{r}\alpha_u^2(\Delta x\Delta z + a(z\Delta z + z'\Delta z + \Delta z^2 - zz'))^2 \\ + \quad \alpha_v^2(\Delta y\Delta z + b(z\Delta z + z'\Delta z + \Delta z^2 - zz'))^2\end{array}}}{z'(z'+\Delta z)(z+\Delta z)} = \frac{\varepsilon}{3} \tag{3.56}$$

$$d_\pi \leq dz\frac{\sqrt{\begin{array}{r}\alpha_u^2(\Delta x\Delta z + a(2z\Delta z + dz\Delta z + \Delta z^2 - z^2 - zdz))^2 \\ + \quad \alpha_v^2(\Delta y\Delta z + b(2z\Delta z + dz\Delta z + \Delta z^2 - z^2 - zdz))^2\end{array}}}{(z+dz)(z+dz+\Delta z)(z+\Delta z)} = \frac{\varepsilon}{3} \tag{3.57}$$

La résolution de cette équation pour obtenir dz en fonction de ε nécessite la résolution d'une équation du quatrième degré. Bien entendu, seules les solutions réelles négatives nous intéressent.

3.5 Quatrième ensemble de positions

Pour finir, tentons d'utiliser simultanément l'indifférence vis-à-vis de la position et de la dilatation.



$$\frac{u'-u}{\alpha_u} = \left(\frac{p'_x}{p'_z} - \frac{x'}{z'}\right) - \sigma\left(\frac{p_x}{p_z} - \frac{x}{z}\right) \tag{3.58}$$

$$\frac{v'-v}{\alpha_v} = \left(\frac{p'_y}{p'_z} - \frac{y'}{z'}\right) - \sigma\left(\frac{p_y}{p_z} - \frac{y}{z}\right) \tag{3.59}$$

Tout comme dans la section précédente, nous choisissons $\sigma = \frac{z}{z'}$. Pour les deux premières familles, $z' = z \Rightarrow \sigma = 1$. Celles-ci ne seront donc pas affectées par l'ajout du paramètre σ et seront identiques à celles obtenues pour le deuxième ensemble de positions. Il reste à étudier la modification de la troisième famille de surfaces.

$$\begin{aligned}
\frac{u'-u}{\alpha_u} &= \left(\frac{p'_x}{p'_z}-\frac{x'}{z'}\right)-\sigma\left(\frac{p_x}{p_z}-\frac{x}{z}\right)\\
&= \left(\frac{x'+\delta x}{z'+\delta z}-\frac{x'}{z'}\right)-\sigma\left(\frac{x+\delta x}{z+\delta z}-\frac{x}{z}\right)\\
&= \frac{z'\delta x-x'\delta z}{z'(z'+\delta z)}-\sigma\frac{z\delta x-x\delta z}{z(z+\delta z)}\\
&= \frac{\delta x-c(z'+\Delta z)\delta z}{z'+\delta z}-\sigma\frac{\delta x-c(z+\Delta z)\delta z}{z+\delta z}\\
&= (\delta x-c\delta z\Delta z)\left(\frac{1}{z'+\delta z}-\sigma\frac{1}{z+\delta z}\right)-c\delta z\left(\frac{z'}{z'+\delta z}-\sigma\frac{z}{z+\delta z}\right)\\
&= \frac{(\delta x-c\delta z\Delta z)(\delta z(1-\sigma)+z-\sigma z')-c\delta z(zz'(1-\sigma)+\delta z(z'-\sigma z))}{(z'+\delta z)(z+\delta z)}
\end{aligned}$$

Puisque $\sigma=\frac{z}{z'}$, $1-\sigma=\frac{dz}{z'}$ et $z-\sigma z'=0$ et $z'-\sigma z=\frac{z'^2-z^2}{z'}=dz\frac{z+z'}{z'}$.

$$\begin{aligned}
\frac{u'-u}{\alpha_u} &= \frac{(\delta x-c\delta z\Delta z)(\delta z dz)-c\delta z(zz'dz+\delta z dz(z+z'))}{z'(z'+\delta z)(z+\delta z)}\\
&= dz\frac{-czz'\delta z+\delta x\delta z-c(z+z'+\Delta z)\delta z^2}{z'(z'+\delta z)(z+\delta z)} \qquad (3.60)
\end{aligned}$$

$$\begin{aligned}
\max|u'-u| &\le |dz||\alpha_u|\frac{\max|\overbrace{-czz'}^{\le 0}\delta z+\delta x\delta z\overbrace{-c(z+z'+\Delta z)}^{\ge 0}\delta z^2|}{\min|z'(z'+\delta z)(z+\delta z)|}\\
&\le dz|\alpha_u|\frac{czz'\Delta z+\Delta x\Delta z-c(z+z'+\Delta z)\Delta z^2}{z'(z'+\Delta z)(z+\Delta z)}\\
&\le dz|\alpha_u|\frac{\Delta x\Delta z+a(z\Delta z+z'\Delta z+\Delta z^2-zz')}{z'(z'+\Delta z)(z+\Delta z)}\\
&\le |\alpha_u|\frac{dz}{z'}\left(\frac{\Delta x\Delta z-2azz'}{(z'+\Delta z)(z+\Delta z)}+a\right) \qquad (3.61)
\end{aligned}$$

Symétriquement, on a :

$$\max|v'-v|\le|\alpha_v|\frac{dz}{z'}\left(\frac{\Delta y\Delta z-2bzz'}{(z'+\Delta z)(z+\Delta z)}+b\right) \qquad (3.62)$$

Ces bornes étant similaires à celles obtenues pour le troisième ensemble, le résultat sera identique.

3.6 Analyse des performances

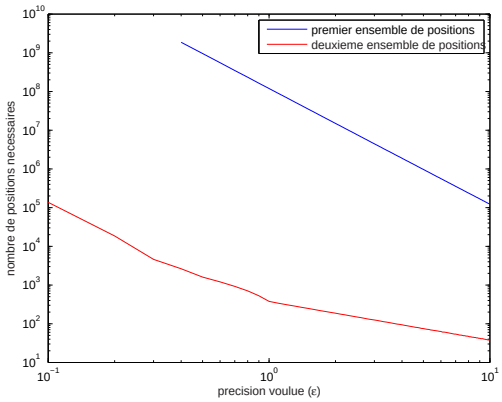


FIG. 3.1 – Impact de l'indifférence à la translation

La figure 3.1 illustre l'impact de l'ajout d'une indifférence à la translation. Le premier ensemble de positions que nous avons obtenu, sans cette indifférence, est vraiment très grand et inutilisable. L'ajout de l'indifférence (deuxième ensemble de positions) donne à l'ensemble une taille raisonnable. Il est très probable que l'indifférence à l'échelle limite également fortement la taille de l'ensemble, et que la combinaison de ces deux désintéressements donne un ensemble de petite taille.

Les données illustrées à la figure 3.1 ont été obtenues pas la méthode décrite précédemment. Nous y avons considéré une boîte limite de 50 cm $\times$ 180 cm $\times$ 20 cm, correspondant à celle d'une personne debout et face à la caméra. Concernant cette dernière, nous avons utilisé des valeurs typiques pour ses paramètres : une image produite de 640 $\times$ 480 pixels, l'image étant centrée sur l'axe optique. La taille typique d'une cellule photosensible d'un capteur CCD variant entre 4 μm et 16 μm, une valeur moyenne donne $k = 10^5$ m. Si l'on considère une distance focale de 10 cm comme raisonnable, $\alpha_u = \alpha_v = 10000$ est un bon choix. Enfin, nous avons uniquement comptabilisé les poses telles que la projection de la boîte soit entièrement incluse dans l'image de la caméra.

Avant de passer à l'étude de l'orientation de l'avatar, une remarque s'impose. Dans ce chapitre, nous avons toujours borné les diverses valeurs pour garantir que la base de données contienne toutes les silhouettes possibles, à une certaine tolérance près. Le nombre de positions réellement nécessaires peut dès lors être de loin inférieur au nombre de positions proposées par la méthode présentée.

Chapitre 4

Orientation et pose de l'avatar

Sommaire

4.1 Définition du problème

Nous considérons à présent que l'avatar est situé à position fixe et cherchons à caractériser les différentes orientations dans lesquelles il est nécessaire de le placer pour obtenir un ensemble suffisamment varié, mais le moins dense possible, de silhouettes. La théorie établie s'étend également aux choix des poses.

4.2 Orientation de l'avatar

L'orientation d'un corps rigide dans un espace tridimensionnel nécessite trois degrés de libertés. Si nous plaçons, par la pensée, l'avatar dans une sphère, nous pouvons l'orienter partiellement en choisissant un point de la sphère pour indiquer la direction dans laquelle se situe la tête. Il faut ensuite spécifier un angle, pour l'empêcher de tourner librement autour de l'axe choisi.

Si nous ne faisons aucune hypothèse concernant la géométrie de l'avatar, les différents points choisis sur la sphère se doivent d'être répartis équitablement. Il s'agit là d'un problème complexe. Plusieurs critères de

répartition existent. L'un d'entre eux consiste, une fois le nombre de points fixés, à maximiser la plus petite distance séparant deux points sur la sphère. Il s'agit du problème de TAMMES. Comme pour bon nombre d'autres critères, on observe expérimentalement que les points sont les centres d'hexagones, sauf pour quelques-uns qui sont le centre de pentagones. Ceci est illustré à la figure 4.1

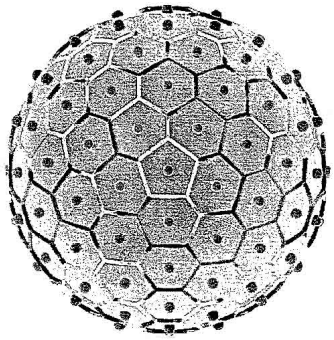


FIG. 4.1 – Exemple de distribution de points sur une sphère

Vu la difficulté du problème, nous allons supposer que l'avatar est dans une position "debout". Ainsi, l'orienter consiste à le faire tourner autour de son axe "vertical" et à choisir quelques-uns des angles de rotation. Répartir ces angles est un problème plus simple, que l'on peut comparer au choix de points répartis uniformément sur un cercle.

Encore faut-il connaître de nombre d'orientations à utiliser. L'idée de base est de déterminer une distance maximale que peut parcourir un point quelconque de l'avatar pour garantir que sa silhouette de subisse pas un changement trop important. Notre raisonnement se base sur la figure 4.2. On y considère un déplacement borné et on étudie son impact sur la projection d'un point pour diverses positions. On constate que l'influence maximale est obtenue pour le point ayant une abscisse et une ordonnée maximales (en valeurs absolues) et ayant une profondeur la plus faible possible.

Nous allons, dans un premier temps, fixer l'amplitude du déplacement à d. Nous étudierons un déplacement du point $\begin{pmatrix} x & y & z \end{pmatrix}$ au point $\begin{pmatrix} x' & y' & z' \end{pmatrix}$. Nous chercherons ensuite une borne du déplacement d_π subi par la projection du point. La valeur de l'amplitude du déplacement sera obtenue en exprimant l'infériorité du déplacement de la projection par rapport à la distance de HAUSDORFF souhaitée. Pour ce faire, commençons par utiliser les coordonnées sphériques pour exprimer que la distance entre $\begin{pmatrix} x & y & z \end{pmatrix}$ et $\begin{pmatrix} x' & y' & z' \end{pmatrix}$ vaut d :

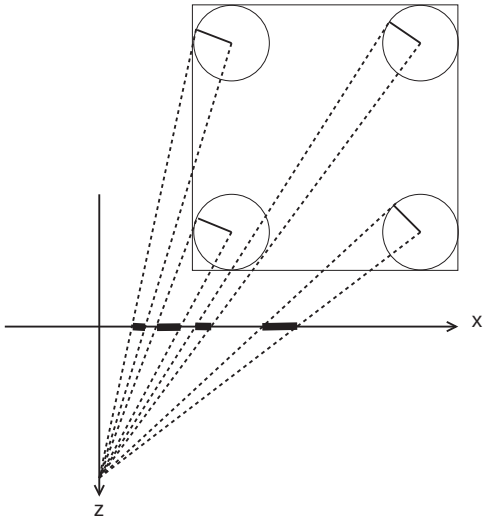


FIG. 4.2 – Influence maximale, d'un déplacement borné d'un point, sur sa projection. Le carré représente la *bounding box* de l'objet étudié.

$$\begin{cases} x' = x + d\sin\phi\cos\theta \\ y' = y + d\sin\phi\sin\theta \\ z' = z + d\cos\phi \end{cases} \tag{4.1}$$

Par inégalité triangulaire,

$$d_\pi \begin{pmatrix} x & y & z \\ x' & y' & z' \end{pmatrix} \leq d_\pi \begin{pmatrix} x & y & z \\ x' & y & z \end{pmatrix} + d_\pi \begin{pmatrix} x' & y & z \\ x' & y' & z \end{pmatrix} + d_\pi \begin{pmatrix} x' & y' & z \\ x' & y' & z' \end{pmatrix} \tag{4.2}$$

$$\begin{aligned}
d_\pi \begin{pmatrix} x & y & z \\ x' & y & z \end{pmatrix} &= \sqrt{\alpha_u^2 \left(\frac{x + d\sin\phi\cos\theta}{z} - \frac{x}{z} \right)^2 + 0} \\
&= |\alpha_u| \frac{d}{|z|} |\sin\phi\cos\theta| = d_\pi \begin{pmatrix} x & y & z \\ x-d & y & z \end{pmatrix} |\sin\phi\cos\theta| \\
d_\pi \begin{pmatrix} x' & y & z \\ x' & y' & z \end{pmatrix} &= \sqrt{0 + \alpha_v^2 \left(\frac{y + d\sin\phi\sin\theta}{z} - \frac{y}{z} \right)^2} \\
&= |\alpha_v| \frac{d}{|z|} |\sin\phi\sin\theta| = d_\pi \begin{pmatrix} x & y & z \\ x & y-d & z \end{pmatrix} |\sin\phi\sin\theta| \\
d_\pi \begin{pmatrix} x' & y' & z \\ x' & y' & z' \end{pmatrix} &= \sqrt{\alpha_u^2 \left(\frac{x'}{z'} - \frac{x'}{z} \right)^2 + \alpha_v^2 \left(\frac{y'}{z'} - \frac{y'}{z} \right)^2} \\
&= \sqrt{\alpha_u^2 x'^2 + \alpha_v^2 y'^2} \left| \frac{1}{z'} - \frac{1}{z} \right|
\end{aligned}$$

Étant donné que le point $\begin{pmatrix} x & y & z \end{pmatrix}$ a une abcisse maximale, une ordonnée maximale et une profondeur minimale, on a les inégalités suivantes :

$$z - d \leq z' \leq z < 0 \qquad 0 \leq |x'| \leq |x| \qquad 0 \leq |y'| \leq |y| \tag{4.3}$$

Appliquons-les :

$$\alpha_u^2 x'^2 + \alpha_v^2 y'^2 \leq \alpha_u^2 x^2 + \alpha_v^2 y^2 \tag{4.4}$$

$$0 \leq \frac{1}{z'} - \frac{1}{z} \leq \frac{1}{z-d} - \frac{1}{z} \tag{4.5}$$

Ainsi, on arrive à la conclusion que :

$$d_\pi \begin{pmatrix} x' & y' & z \\ x' & y' & z' \end{pmatrix} \leq \sqrt{\alpha_u^2 x^2 + \alpha_v^2 y^2} \left| \frac{1}{z-d} - \frac{1}{z} \right| = d_\pi \begin{pmatrix} x & y & z \\ x & y & z-d \end{pmatrix} \tag{4.6}$$

La relation 4.2 devient donc,

$$\begin{aligned} d_\pi \begin{pmatrix} x & y & z \\ x' & y' & z' \end{pmatrix} \quad &\leq \quad d_\pi \begin{pmatrix} x & y & z \\ x-d & y & z \end{pmatrix} |\sin\phi\cos\theta| \\ &+ \quad d_\pi \begin{pmatrix} x & y & z \\ x & y-d & z \end{pmatrix} |\sin\phi\sin\theta| \\ &+ \quad d_\pi \begin{pmatrix} x & y & z \\ x & y & z-d \end{pmatrix} \end{aligned} \tag{4.7}$$

Cette expression peut être vue comme le produit scalaire des vecteurs

$$\left(d_\pi \begin{pmatrix} x & y & z \\ x-d & y & z \end{pmatrix} \quad d_\pi \begin{pmatrix} x & y & z \\ x & y-d & z \end{pmatrix} \quad d_\pi \begin{pmatrix} x & y & z \\ x & y & z-d \end{pmatrix} \right)$$

et

$$\begin{pmatrix} |\sin\phi\cos\theta| & |\sin\phi\sin\theta| & 1 \end{pmatrix}$$

Le produit scalaire étant donné par le produit des normes des vecteurs multiplié par le cosinus de l'angle entre ceux-ci,

$$\begin{aligned} d_\pi \begin{pmatrix} x & y & z \\ x' & y' & z' \end{pmatrix} \quad &\leq \quad \sqrt{d_\pi^2 \begin{pmatrix} x & y & z \\ x-d & y & z \end{pmatrix} + d_\pi^2 \begin{pmatrix} x & y & z \\ x & y-d & z \end{pmatrix} + d_\pi^2 \begin{pmatrix} x & y & z \\ x & y & z-d \end{pmatrix}} \sqrt{2} \\ d_\pi^2 \begin{pmatrix} x & y & z \\ x' & y' & z' \end{pmatrix} \quad &\leq \quad 2 \left(d_\pi^2 \begin{pmatrix} x & y & z \\ x-d & y & z \end{pmatrix} + d_\pi^2 \begin{pmatrix} x & y & z \\ x & y-d & z \end{pmatrix} + d_\pi^2 \begin{pmatrix} x & y & z \\ x & y & z-d \end{pmatrix} \right) \\ &= \quad 2 \left(\alpha_u^2 \frac{d^2}{z^2} + \alpha_v^2 \frac{d^2}{z^2} + (\alpha_u^2 x^2 + \alpha_v^2 y^2) \frac{d^2}{z^2 (z-d)^2} \right) = F^2(d) \end{aligned} \tag{4.8}$$

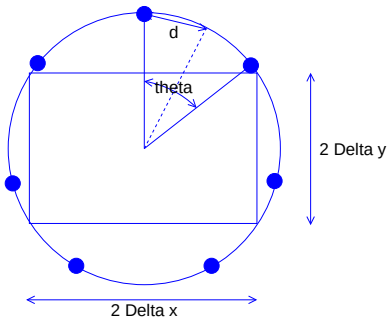


FIG. 4.3 – Détermination du nombre d'orientations nécessaires en fonction du déplacement local maximal d.

Une simulation numérique a montré que le facteur 2 serait inutile. Nous souhaitons garantir que $d_\pi \leq \varepsilon$. Nous avons obtenu une borne du type $d_\pi \leq F(d)$, il nous reste donc à monter que $\frac{\partial}{\partial d}F(d) > 0$ et imposer que $d \leq F^{-1}(\varepsilon)$. En effet,

$$d \leq F^{-1}(\varepsilon) \underset{\frac{\partial}{\partial d}F(d)>0}{\longleftrightarrow} F(d) \leq \varepsilon \underset{d_\pi \leq F(d)}{\longrightarrow} d_\pi \leq \varepsilon \tag{4.9}$$

Montrons que $\frac{\partial}{\partial d}F(d) > 0$:

$$\frac{\partial}{\partial d}F(d) = \frac{\alpha_u^2+\alpha_v^2}{z^2}2d + \frac{\alpha_u^2x^2+\alpha_v^2y^2}{z^2}\frac{2d(z-d)^2+d^2 2(z-d)}{(z-d)^4} \tag{4.10}$$

$$= \underbrace{\frac{\alpha_u^2+\alpha_v^2}{z^2}}_{>0}\underbrace{2d}_{>0} + \underbrace{\frac{\alpha_u^2x^2+\alpha_v^2y^2}{z^2}}_{\geq 0}\underbrace{\frac{2d(z-d)z}{(z-d)^4}}_{>0} > 0 \tag{4.11}$$

La plus grande valeur de d telle que $d \leq F^{-1}(\varepsilon)$ est obtenue en résolvant l'équation $F(d) = \varepsilon$. Il s'agit d'une résolution d'une équation du quatrième degré. Une fois d connu, on détermine le nombre d'orientations nécessaires. Le raisonnement est illusté à la figure 4.3, pour une rotation autour de l'axe des z. Nous avons :

$$4(\Delta_x^2+\Delta_y^2)\sin^2\frac{\theta}{4} \leq d^2 \qquad \Leftrightarrow \qquad \theta \leq 4\arcsin\frac{d}{2\sqrt{\Delta_x^2+\Delta_y^2}} \tag{4.12}$$

$$n_\angle = \left\lceil \frac{\pi}{2\arcsin\frac{d}{2\sqrt{\Delta_x^2+\Delta_y^2}}} \right\rceil \tag{4.13}$$

On peut également se servir du déplacement local maximal d pour choisir les poses.

4.3 Choix des poses

Commençons par émettre une hypothèse. A chaque paramètre i de pose correspond une rotation rigide. L'angle de rotation est donné sous forme d'un intervalle : $\begin{bmatrix}\theta_{i,min} & \theta_{i,max}\end{bmatrix}$. On suppose également la connaissance de la distance maximale r_i des points mobilisés à l'axe de rotation.

Nous travaillerons avec la distance curviligne maximale a_i parcourue par les points de l'objet :

$$a_i = |\theta_{i,max} - \theta_{i,min}| r_i \tag{4.14}$$

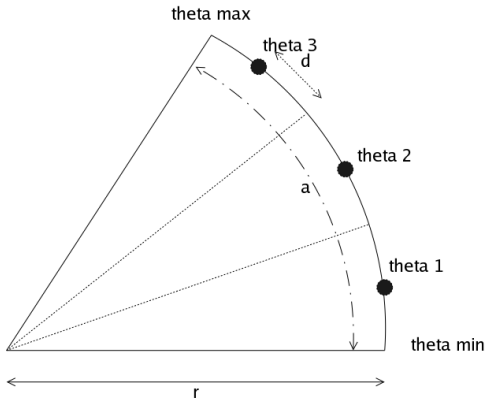
Supposons que nous souhaitions faire prendre n_i valeurs au paramètre i. Les $n_i(\geq 1)$ angles correspondants sont donnés par :

$$\theta_{i,j} = \theta_{i,min} + \frac{\theta_{i,max} - \theta_{i,min}}{n_i}\left(j - \frac{1}{2}\right) \qquad \text{avec} \qquad j \in [1, n_i] \tag{4.15}$$

Maintenant, imaginons que l'on ait un ensemble de silhouettes de l'avatar mis dans les diverses poses, comme décrit ci-dessus. La question qui se pose est l'obtention d'une borne de la modification de la silhouette si on observe le même avatar au même endroit, orienté de la même manière mais dans une pose quelconque. Si l'on veut ramener l'avatar dans une pose connue, il faudra modifier consécutivement le paramètre i de chaque articulation pour lui faire prendre un des angles $\theta_{i,j}$. La distance totale parcourue par les points de l'objet est obtenue en sommant les distances parcourues lors de la modification de la valeur de chaque paramètre. Calculons-en une borne majorante :

$$d_i = \frac{a_i}{2n_i} \tag{4.16}$$

Les différentes grandeurs évoquées jusqu'à présent sont schématisées ci-dessous :



Nous avons maintenant tous les éléments nécessaires pour poser le problème de la détermination des n_i :

$$\sum_i d_i \leq \varepsilon \Leftrightarrow \sum_i \frac{a_i}{n_i} \leq 2\varepsilon \qquad \prod_i n_i \text{est minimal} \tag{4.17}$$

$$\varepsilon = cste \geq 0 (\in \mathbb{R}) \qquad a_i = cste \geq 0 (\in \mathbb{R}) \qquad n_i \geq 1 (\in \mathbb{Z}) \tag{4.18}$$

Il s'agit d'un problème d'optimisation à valeurs entières. C'est donc un problème complexe. Nous allons essayer établir une heuristique à partir de la solution en valeurs réelles. Voici comment établir cette dernière, en notant N le nombre de paramètres de pose :

$$\sum_{i=1}^{N} \frac{a_i}{n_i} = 2\varepsilon \Leftrightarrow n_1 = \frac{a_1}{2\varepsilon - \sum_{i=2}^{N} \frac{a_i}{n_i}} \tag{4.19}$$

$$\prod_{i=1}^{N} n_i = \frac{a_1 \prod_{i=2}^{N} n_i}{2\varepsilon - \sum_{i=2}^{N} \frac{a_i}{n_i}} \tag{4.20}$$

$$\begin{aligned}
\frac{\partial}{\partial n_j} \prod_{i=1}^{N} n_i = 0 \quad &\Leftrightarrow \quad \frac{\partial}{\partial n_j} \left(a_1 \prod_{i=2}^{N} n_i \right) \left(2\varepsilon - \sum_{i=2}^{N} \frac{a_i}{n_i} \right) = \left(a_1 \prod_{i=2}^{N} n_i \right) \frac{\partial}{\partial n_j} \left(2\varepsilon - \sum_{i=2}^{N} \frac{a_i}{n_i} \right) \\
&\Leftrightarrow \quad a_1 \left(\prod_{i=2, i\neq j}^{N} n_i \right) \left(2\varepsilon - \sum_{i=2}^{N} \frac{a_i}{n_i} \right) = a_1 \left(\prod_{i=2}^{N} n_i \right) \frac{a_j}{n_j^2} \\
&\Leftrightarrow \quad \frac{a_j}{n_j} = 2\varepsilon - \sum_{i=2}^{N} \frac{a_i}{n_i} = K \qquad \forall j \in [1, N] \\
&\Leftrightarrow \quad \frac{a_j}{n_j} = K = 2\varepsilon - (N-1)K \\
&\Leftrightarrow \quad \frac{a_j}{n_j} = K = \frac{2\varepsilon}{N} \\
&\Leftrightarrow \quad n_i = \frac{N}{2\varepsilon} a_i \qquad \forall i \in [1, N]
\end{aligned}$$

Pour obtenir rapidement le nombre de valeurs à considérer pour chaque paramètre, on utilisera l'heuristique suivante :

- prendre $n_i = \max\left\{1, \left\lceil \frac{N}{2\varepsilon} a_i \right\rceil \right\}$;
- trier les paramètres par ordre croissant de a_i (ou de n_i) ;
- pour chaque paramètre, décrémenter n_i tant que $\sum_i \frac{a_i}{n_i} \leq 2\varepsilon$;
- calculer les $\theta_{i,j}$ en fonction des n_i.

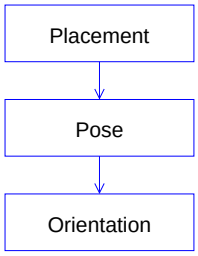
Cette heuristique permet d'automatiquement déterminer les paramètres qui ont le moins d'influence. Les paramètres i négligeables se verront attribuer une valeur n_i égale à 1. Malheureusement, pour l'avatar que nous utiliserons dans la suite, qui possède 105 paramètres de poses, le nombre de

poses demandées ($\prod_i n_i$) devient gigantesque.

L'heuristique précédente n'est pas idéale. Pour le comprendre, considérons un point de la main gauche de l'avatar. Lorsqu'on modifie la pose de l'avatar pour le placer dans une pose connue, le point considéré n'est pas déplacé par toutes les articulations ; entre autres, les articulations du bras gauche, de la nuque et des jambes ne doivent pas rentrer en compte. Le squelette de l'avatar, présenté au chapitre 6, permettrait de déterminer une hiérarchie entre les articulations et ainsi diminuer fortement le nombre de poses.

4.4 Ordre des choix

Il serait donc théoriquement possible de choisir les poses, positions et orientations à faire prendre à l'avatar. Il reste à savoir dans quel ordre doivent être faits les choix. Le choix de la pose et de l'orientation conditionne la taille de la boîte limite ; le nombre de positions peut ainsi être réduit pour certaines poses. En revanche, le choix de la position influence fortement le nombre de poses et d'orientations à faire prendre à l'avatar. Clairement, le choix des positions doit se faire en premier lieu en tenant compte de la plus grande boîte limite, puis pour chaque position, on doit déterminer les poses et orientations. L'orientation ne limite pas les poses, mais le choix d'une pose peut réduire la taille de la boîte limite et ainsi le nombre d'orientations nécessaires. On privilégiera donc les choix dans l'ordre :



L'organisation globale proposée est illustrée à la figure 4.4. On se sert des paramètres de projection obtenus par étalonnage d'une caméra. Ces paramètres servent à la détermination des positions, poses, et orientations à faire prendre à l'avatar. L'avatar est défini par son apparence. On en calcule les dimensions pour déterminer la taille de la boîte limite, ainsi qu'un squelette permettant de déterminer un bon ensemble de poses. On commencerait alors par déterminer un ensemble de positions ; pour chacune d'entre elles, on spécifierait les poses. On obtiendrait alors un objet tridimensionnel représentant l'avatar qu'il ne reterait plus qu'à orienter. Pour toutes les configurations, la projection serait effectuée en vue d'obtenir les silhouettes qui seraient triées pour éliminer celles redondantes.

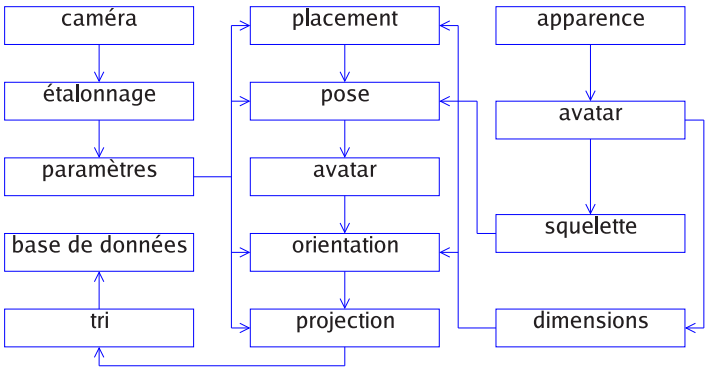


FIG. 4.4 – Organisation globale proposée.

Chapitre 5

MakeHuman et l'avatar

Sommaire

Le but du travail étant de générer des silhouettes à partir d'un avatar humain tridimensionnel, nous avions besoin d'une telle modélisation de personnes, de préférence la plus réaliste possible. En particulier, nous souhaitions éviter des silhouettes anguleuses. Réaliser une telle modélisation par nous-mêmes était impensable en un temps raisonnable.

Les modélisations humaines disponibles se répartissent en trois catégories. D'un côté, on a des modèles sous forme de fichiers importables dans des environnements de modélisation 3D tels que Pov-Ray, Blender, Maya, etc ... Ces modèles peuvent alors être retouchés, associés à des squelettes et animés suivant une séquence prédéterminée. Evolver[1] permet de créer de tels modèles. D'un autre côté se trouvent des applications proposant une modélisation interactive de l'avatar dont il ne reste alors plus qu'à effectuer le rendu. Parmi ces logiciels, notons Poser[2], MakeHuman, ... Enfin, il existe la norme H-Anim[3], standard permettant la description de personnages 3D animés et destiné à rendre compatibles les personnages virtuels avec les différents

[1] http ://www.darwindimensions.com/content/product.jsp
[2] http ://www.e-frontier.com/
[3] http ://www.h-anim.org/

systèmes de modélisation, animation et visualisation tels que VRML[4]

Assez rapidement, nous nous sommes focalisés sur le projet *open-source* MakeHuman dont l'annonce de la version 0.9 était prometteuse. Nous n'avons pas été déçus. Après avoir démontré des possibilités offertes, cet avatar a été choisi et nous n'avons guère investigué les autres possibilités.

5.1 MakeHuman

L'avatar choisi est celui du projet MakeHuman[5].

FIG. 5.1 – L'avatar de MakeHuman 0.9

5.1.1 Introduction

MakeHuman est historiquement une suite de scripts python pour réaliser des personnages avec Blender. Aujourd'hui, il est devenu un programme C++ *open-source*, existant pour plusieurs systèmes d'exploitation, destiné à modéliser des humanoïdes tridimensionnels réalistes.

Outre les possibilités de modélisation offertes, il permet de faire prendre à l'avatar une pose voulue et d'en obtenir un rendu ou d'exporter le maillage au format WaveFrontObject *.obj*. L'apparence et la pose peuvent toutes deux être sauvegardées sous forme de fichiers *.bs* et restaurées à partir de ceux-ci. Il s'agit de fichiers textes associant les valeurs à chaque paramètre utilisé.

[4]Virtual Reality Markup Language
http ://www.web3d.org/x3d/specifications/vrml/ISO-IEC-14772-VRML97/
[5]Portail : http ://www.makehuman.org
Documentation : http ://makewiki.aleppax.it/pmwiki/pmwiki.php
Code source : http ://sourceforge.net/project/showfiles.php ?group_id=150931

La version 0.9 ne contient pas moins de 2972 paramètres d'apparence et 105 paramètres de pose.

FIG. 5.2 – Le programme MakeHuman 0.9 et l'avatar de base, dans sa pose initiale.

5.1.2 Le modèle

Le modèle de MakeHuman est ce que l'on peut qualifier d'humanoïde universel, un maillage pouvant être déformé pour obtenir quasiment n'importe quelle apparence imaginable.

MakeHuman permet ainsi de créer une large panoplie d'humanoïdes, de l'enfant à l'adulte, de l'homme à la femme, du maigre à l'obèse en passant par de très nombreuses variantes et options de fantaisie telles que cornes, dentitions et asymétries dignes de bandes dessinées.

Le maillage d'origine avait été optimisé pour limiter le nombre de faces et de points. Depuis lors, il a été corrigé et le nombre de facettes et de noeuds a augmenté pour devenir humanoïde universel. Voici quelques caractéristiques du maillage actuel :

Nombre de noeuds	:	10936
Nombre de faces triangulaires	:	470
Nombre de faces quadrangulaires	:	10387

La question de la pose initiale de l'avatar n'est apparemment pas triviale. La première version de MakeHuman adopta partiellement la position foetale, en raison de l'angle intermédiaire pris par les diverses articulations, l'espoir étant de limiter l'erreur de pose via la réduction des angles parcourus. En pratique, cependant, ce choix s'est révélé inadéquat. C'est pourquoi aujourd'hui, la pose classique de l'homme de Vitruve est utilisée : debout, les pieds joints et les bras écartés à l'horizontale.

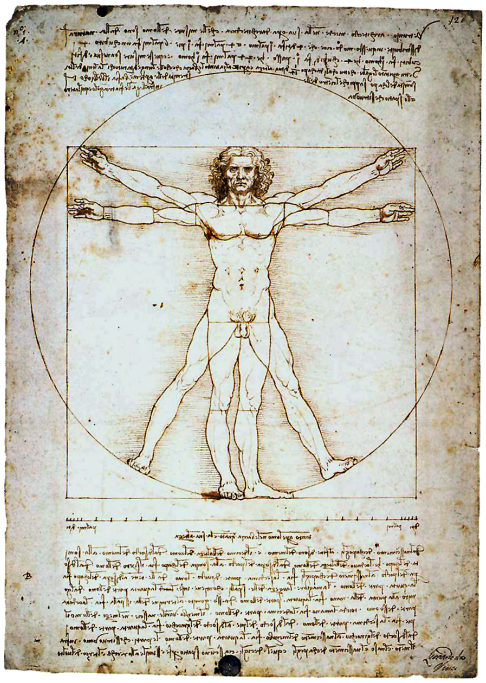


FIG. 5.3 – d'après Wikipedia : ≪ *L'Homme de Vitruve (ou homme vitruvien) est le nom communément donné au croquis Étude de proportions du corps humain selon Vitruve réalisé par Léonard de Vinci aux alentours de 1492*≫.

5.1.3 Le système de pose

Le système de pose actuel (QuickPose) n'est pas idéal pour réaliser des animations, mais devrait être remplacé dans le futur par un système plus complexe le permettant. Cependant, le système actuel est suffisant pour travailler pose par pose. La façon actuelle de procéder est principalement une application d'une matrice de rotation tridimensionnelle rigide autour de l'articulation. Cependant, pour obtenir un résultat satisfaisant, le maillage est préalablement déformé par transformation linéaire.

Cette transformation est établie par le moteur musculaire de MakeHuman. Il s'agit d'un "système expert" capable d'apprendre, à partir de

FIG. 5.4 – Fonctionnalités futures de MakeHuman : habits.
source : Blog des développeurs de MakeHuman
http ://www.dedalo-3d.com/public/blog/index.php

maillages créés par des artistes, le comportement musculaire lors de mouvements de l'ossature du corps. Le degré de réalisme et la précision ne sont donc pas fixés, mais augmenteront avec le nombre et la qualité des exemples servant à l'apprentissage.

5.1.4 Les évolutions prévues

A terme, le logiciel devrait inclure un moteur de pose hautement réaliste basé sur une simulation quasi-parfaite du système musculaire et du squelette. De plus, il devrait se compléter par un ensemble d'outils permettant d'habiller les personnages et de leur créer une chevelure.

FIG. 5.5 – Fonctionnalités futures de MakeHuman : cheveux.
source : Blog des développeurs de MakeHuman
http ://www.dedalo-3d.com/public/blog/index.php

FIG. 5.6 – Fonctionnalités futures de MakeHuman : des paramètres d'apparence permettant d'obtenir des personnages encore plus fantaisistes.
source : Blog des développeurs de MakeHuman
http ://www.dedalo-3d.com/public/blog/index.php

FIG. 5.7 – Fonctionnalités futures de MakeHuman : nouveau moteur de pose basé sur une modélisation fine du squelette et de la musculature humaine.
source : Blog des développeurs de MakeHuman
http ://www.dedalo-3d.com/public/blog/index.php

Chapitre 6

Squelette de l'avatar

Sommaire

6.1 Apprentissage du squelette

Dans sa version actuelle, MakeHuman ne fournit pas de squelette au maillage. De même, bien que les paramètres soient clairement nommés, l'ensemble et les noms de ceux-ci ne sont pas fixés. Afin de permettre une évolution du logiciel, on se doit donc de considérer le programme MakeHuman comme étant une boîte noire dont l'unique entrée est une liste de paramètres avec les valeurs associées et dont l'unique sortie fournit un maillage de l'avatar. Afin de contrôler cette boîte noire de manière utile, il convient tout d'abord d'apprendre son comportement.

L'avatar va être modélisé par un modèle articulé à partir duquel on va déterminer un squelette, permettant de calculer certaines valeurs ca-

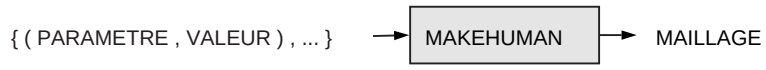


FIG. 6.1 – Interaction avec la boîte noire MakeHuman

ractérisant les paramètres. Il nous faut donc, dans un premier temps, définir ces deux notions.

6.1.1 Notion de modèle articulé

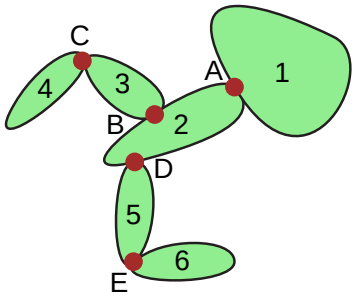


FIG. 6.2 – Exemple de modèle articulé

Définition. Nous appellerons modèle articulé un ensemble de parties rigides interconnectées par des charnières. Dans la suite, nous noterons les différentes parties par des naturels et les charnières par des lettres majuscules. Nous nous restreindrons aux modèles articulés acycliques et connexes, c'est-à-dire tels qu'entre deux parties il existe un et un seul chemin.

La motivation à définir cette notion tient en la similitude qu'on remarque en première approximation entre un humanoïde et un tel modèle. Les articulations sont mises en correspondance avec autant de charnières qu'elles ont de degrés de liberté. Les parties rigides sont quant à elles à comparer avec les parties du corps sans articulation. Le point le plus délicat concerne les mouvements de torsion qui peuvent être observés, par exemple, aux avant-bras.

De tels modèles sont présents dans la littérature. Le lecteur trouvera sans aucun doute un parallélisme intéressant entre la notion de modèle articulé et la modélisation en boîtes articulées élastiquement qu'utilisent FELZENSZWALB et HUTTENLOCHER.

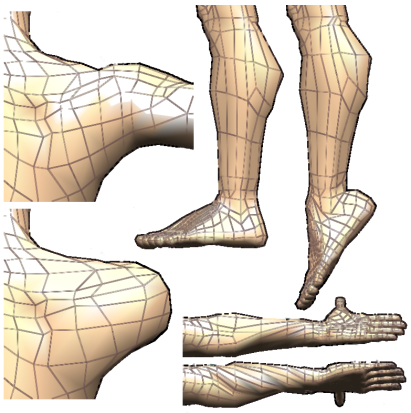


FIG. 6.3 – Différents mouvements de l'avatar de MakeHuman qu'il est impossible de modéliser par un modèle articulé : mouvements des points de l'omoplate suite au mouvement du bras ; torsion de l'avant-bras ; contraction simulée des muscles jumeaux lors de l'étirement du pied, qui ne peut être modélisée par la rotation au niveau de la cheville.

6.1.2 Notion de squelette

Définition. Le squelette d'un modèle articulé est une arborescence binaire dont les feuilles correspondent aux parties rigides et les noeuds internes aux charnières du modèle articulé. L'arborescence est telle que le sous-arbre de droite d'une charnière forme le squelette du sous-modèle articulé maximal mu par celle-ci.

6.1.3 Apprentissage

La liste des paramètres peut être récupérée dynamiquement en listant les fichiers .target du répertoire `share/makehuman/`. Le sous-répertoire `targets/` contient les définitions des paramètres d'apparence et `rotations/` contient les définitions des paramètres de pose. Les paramètres de MakeHuman varient de 0 à 1 ou de -1 à 1. Dans le premier cas, on trouvera 1 fichier .target dont le nom concorde avec le nom du paramètre. Dans le deuxième cas, il y a en plus un deuxième fichier .target dont le nom correspond au nom du paramètre suivi du caractère '-'.

Dans une première phase, on détermine pour chaque paramètre de pose l'ensemble des points du maillage qu'il influence. Ceci est réalisé en fixant la valeur des autres paramètres (par exemple à 0) et en récupérant le maillage pour les deux valeurs extrêmes du paramètre considéré. La comparaison des coordonnées des points des deux maillages donne l'information désirée.

Dans un deuxième temps, on va faire l'hypothèse que l'information récupérée concerne un modèle articulé et que les paramètres de pose correspondent aux charnières de ce modèle. Le modèle articulé est alors récursivement scindé en deux sous-modèles par le choix d'une charnière, l'un d'eux étant mobilisé par celle-ci et l'autre étant fixe.

Nous organisons les paramètres dans un arbre binaire exprimant la hiérarchie entre les articulations correspondantes. La racine de l'arbre correspond à l'articulation principale. Celle-ci est choisie en prenant le paramètre influençant le plus grand nombre de points.

Une fois l'articulation principale choisie, nous classons les autres en deux catégories : celles influencées par l'articulation principale et les autres. Afin que notre démarche soit robuste, nous utilisons l'algorithme suivant :

Soient deux articulations A et B :

1. notons N_A le nombre de points influencés par l'articulation A ;
2. notons N_B le nombre de points influencés par l'articulation B ;
3. notons N_I le nombre de points influencés par les articulations A et B.

Trois cas peuvent se présenter :

1. $N_A < N_B$ et $N_A < N_I \Rightarrow A$ est mobilisée par B.
2. $N_B < N_A$ et $N_B < N_I \Rightarrow B$ est mobilisée par A.
3. $N_I < N_A$ et $N_I < N_B \Rightarrow A$ et B sont indépendantes.

Une fois la répartition des articulations restantes en deux catégories effectuée, on sélectionne les articulations qui ne sont pas influencées par l'articulation principale que l'on attribue au fils de gauche. Les articulations influencées seront, quant-à elles, attribuées au fils de droite. Chaque fils qui se voit attribuer au moins une articulation devient la racine du sous-squelette appris à partir de cet ensemble d'articulations.

Exemple. Considérons le modèle articuléde de la figure 6.2 à 5 charnières et 6 parties rigides tel que la dépendance des points par rapport aux articulations est donnée dans le tableau ci-dessous :

	groupes de points					
articulations	1	2	3	4	5	6
A	$\mathcal{X}$	$\mathcal{V}$	$\mathcal{V}$	$\mathcal{V}$	$\mathcal{V}$	$\mathcal{V}$
B	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{V}$	$\mathcal{V}$	$\mathcal{X}$	$\mathcal{X}$
C	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{V}$	$\mathcal{X}$	$\mathcal{X}$
D	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{V}$	$\mathcal{V}$
E	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{V}$

L'application de l'algorithme précédent aboutira à la conclusion que :

1. E est moblisée par D qui est elle-même mobilisée par A
2. C est moblisée par B qui est elle-même mobilisée par A

Une indétermination subsiste sur l'arbre binaire construit, selon qu'on choisit B ou D en premier lieu comme articulation principale du sous-squelette. Les deux solutions sont présentées à la figure 6.4.

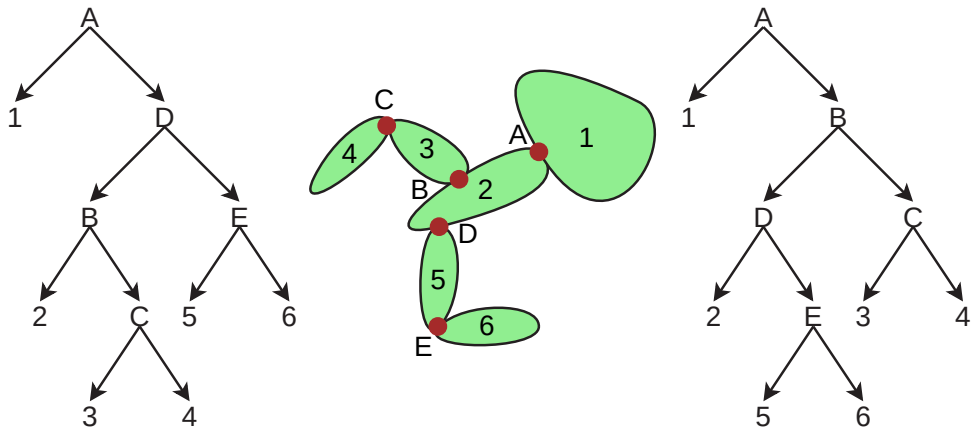


FIG. 6.4 – Squelettes associés au modèle articulé de la figure 6.2

6.1.4 Enrichissement du squelette

Nous allons placer dans l'arbre représentant le squelette diverses données numériques utiles à sa caractérisation : taille des parties mobilisées par les diverses articulations et angles de rotations associés. Ces données sont nécessaires pour estimer les articulations les plus importantes et leur attribuer un poids.

Taille des parties mobilisées

Nous souhaitons connaître la distance maximale entre l'axe de rotation et les points mobilisés. Si nous ne voulons pas calculer l'axe de rotation, nous pouvons utiliser la taille de la partie mobilisée par une articulation, définie comme étant la plus grande distance séparant deux points mobilisés par l'articulation. Si le maillage est suffisamment dense, et qu'il y a des points à proximité de l'axe de rotation, cette taille borne la quantité recherchée. Dans bon nombre de cas, l'exagération sera faible.

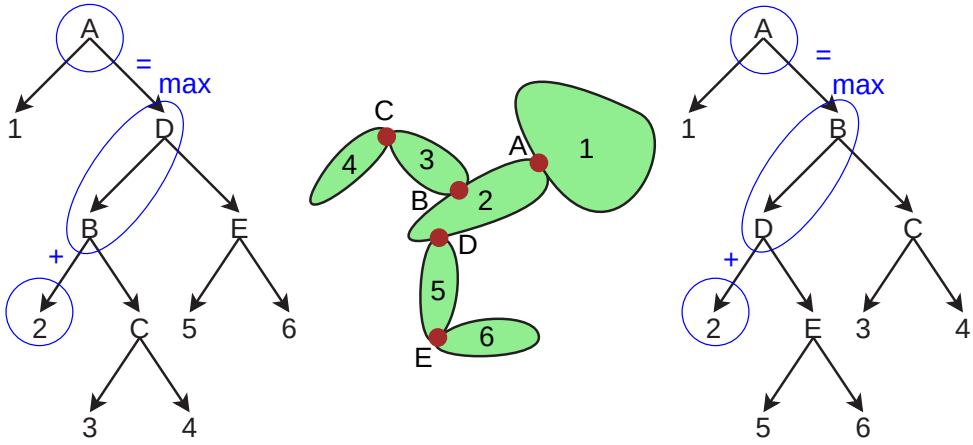


FIG. 6.5 – Calcul des tailles du modèle articulé de la figure 6.2

Lorsque la partie mobilisée contient des articulations, nous nous plaçons dans le cas le plus défavorable, où toutes les articulations sont alignées. Nous pouvons alors calculer la taille récursivement en partant des feuilles de l'arbre. Pour chaque partie rigide, on calcule sa taille. La taille d'une articulation est obtenue en sommant la taille de la partie rigide qu'elle est la seule à mobiliser avec le maximum des tailles des articulations qu'elle mobilise.

Exemple. Par exemple, pour le modèle articulé des figures 6.2 et 6.4, nous obtenons :

$$\begin{aligned} taille(A) &= taille(2) + \max(taille(B), taille(D)) \\ taille(B) &= taille(3) + taille(C) \\ taille(C) &= taille(4) \\ taille(D) &= taille(5) + taille(E) \\ taille(E) &= taille(6) \end{aligned}$$

Nous voyons que la valeur s'obtient en parcourant systématiquement l'arbre à partir de l'articulation dont on souhaite connaître la taille de la partie mobilisée, en allant au fils de droite puis toujours au fils de gauche. On obtient la taille en sommant la taille de la feuille avec le maximum des tailles des noeuds internes par lesquels on passe. Cela est illustré à la figure 6.5. Nous remarquons que le résultat obtenu est identique pour les divers arbres équivalents.

Angle de rotation des articulations

Pour chaque articulation, nous allons calculer l'angle qu'elle est capable de décrire. Rappelons que nous avons obtenu, lors du calcul du squelette,

FIG. 6.6 – Squelette de l'avatar de MakeHuman. Pour la clarté, les paramètres de droite ont été omis (ils sont obtenus par symétrie avec leur homologue de gauche).

les coordonnées des points du maillage lorsque la valeur du paramètre correspondant à l'articulation était minimale et maximale. C'est grâce à ces données que nous allons estimer l'angle de rotation.

Nous commençons par ne garder que les points influencés. Nous faisons l'hypothèse que la modification subie par la partie du maillage sélectionnée est une combinaison d'une rotation, d'une translation et d'une mise à l'échelle. Cette transformation est donc linéaire, représentée par la matrice M :

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{6.1}$$

Soit, en supprimant de la matrice M les valeurs connues, et en tenant compte de tous les points :

$$\begin{bmatrix} x'_1 & x'_2 & \dots \\ y'_1 & y'_2 & \dots \\ z'_1 & z'_2 & \dots \end{bmatrix} = \begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{pmatrix} \begin{bmatrix} x_1 & x_2 & \dots \\ y_1 & y_2 & \dots \\ z_1 & z_2 & \dots \end{bmatrix} \tag{6.2}$$

La résolution de cette équation au sens des moindres carrés nous donne les valeurs de la matrice M. En pratique, nous n'observons pas une rotation pure (cf. figure 6.3). Afin d'épurer l'ensemble de points à partir desquels la transformation est calculée, nous appliquons l'algorithme d'*Expectation-Maximization*. La transformation M est appliquée à chaque point $\begin{pmatrix} x & y & z \end{pmatrix}$ et nous regardons si le résultat est plus proche du point $\begin{pmatrix} x & y & z \end{pmatrix}$ ou du point $\begin{pmatrix} x' & y' & z' \end{pmatrix}$. Nous ne considérons alors plus comme influencés que les seconds. Nous répétons l'opération d'estimation de la matrice M avec ce nouvel ensemble de points tant qu'il n'y a pas convergence de l'ensemble de points.

Une fois la convergence atteinte, nous allons essayer de mettre en évidence la matrice de rotation. Or, puisque nous avons supposé que modification est une combinaison d'une rotation, d'une translation et d'une mise à l'échelle, la matrice M peut se décomposer comme ceci :

$$\begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix} = \underbrace{\begin{pmatrix} & & & 0 \\ & R & & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\text{rotation}} \underbrace{\begin{pmatrix} 1 & 0 & 0 & \delta_x \\ 0 & 1 & 0 & \delta_y \\ 0 & 0 & 1 & \delta_z \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\text{translation}} \underbrace{\begin{pmatrix} \alpha_x & 0 & 0 & 0 \\ 0 & \alpha_y & 0 & 0 \\ 0 & 0 & \alpha_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\text{dilatation}}$$

où R est une matrice de rotation (matrice orthogonale). En regroupant les matrices de translation et de dilatation,

$$\begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} & & & 0 \\ & R & & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \alpha_x & 0 & 0 & \delta_x \\ 0 & \alpha_y & 0 & \delta_y \\ 0 & 0 & \alpha_z & \delta_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{6.3}$$

La décomposition écrite sous cette forme n'est rien d'autre qu'une décomposition QR. La matrice R est donc obtenue en calculant la décomposition QR de

$$\begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{pmatrix}$$

La rotation ayant trois degrés de liberté, elle est définie par un angle θ autour d'un vecteur $\overrightarrow{\begin{bmatrix} x & y & z \end{bmatrix}}$. C'est l'idée du quaternion $\begin{bmatrix} i & j & k & \omega \end{bmatrix}$:

$$i = x \sin \frac{\theta}{2} \qquad j = y \sin \frac{\theta}{2} \tag{6.4}$$

$$k = z \sin \frac{\theta}{2} \qquad \omega = \cos \frac{\theta}{2} \tag{6.5}$$

On peut montrer que la matrice de rotation R équivalente est donnée par :

$$R = \begin{pmatrix} 1 - 2j^2 - 2k^2 & 2ij - 2k\omega & 2ik + 2j\omega \\ 2ij + 2k\omega & 1 - 2i^2 - 2k^2 & 2jk - 2i\omega \\ 2ik - 2j\omega & 2jk + 2i\omega & 1 - 2i^2 - 2j^2 \end{pmatrix} \tag{6.6}$$

Pour obtenir l'angle de rotation, il faut passer de la matrice R au quaternion. Pour ce faire, on utilise l'algorithme suivant :

```
/*
 * Trace ( R ) = R11 + R22 + R33 + 1
 *             = 4 - 4 x^2 - 4 y^2 - 4 z^2
 *             = 4 - 4 sin^2 ( t / 2 )
 *             = 4 cos^2 ( t / 2 )
 * R21 - R12 = ( 2yz + 2wx ) - ( 2yz - 2wx )
 *           = 4 w x
 *           = 4 cos ( t / 2 ) x
 * S = 1 / 2 sqrt ( tr ( R ) ) = 1 / 4 | cos ( t / 2 ) |
 * ( R21 - R12 ) S = +- x
 */

final double trace = m00 + m11 + m22 + 1.0 ;

// NOTE : La trace R11 + R22 + R33 + 1 vaut 4 cos^2 ( t / 2 ).
//        Elle est donc toujours positive mais peut être petite.
//        Pour éviter les erreurs numériques, on procède autrement
//        si elle est petite ...

if ( trace > 0.005 ) {
        s = 0.5 / sqrt ( trace ) ;
        x = ( m21 - m12 ) * s ;    y = ( m02 - m20 ) * s ;
        z = ( m10 - m01 ) * s ;    w = 0.25 / s ;
}

/*
 * 1 + m00 - m11 - m22 = 1 + (1-2y2-2z2) - (1-2x2-2z2) - (1-2x2-2y2)
 *                     = 4 x2
 * s = 2 sqrt ( 1 + m00 - m11 - m22 ) = 4 x
 * m01 + m10 = ( 2xy - 2wz ) + ( 2xy + 2wz ) = 4 x y
 */

// NOTE : La valeur 1.0 + m00 - m11 - m22 vaut 4 x^2.
//        La valeur 1.0 - m00 + m11 - m22 vaut 4 y^2.
//        La valeur 1.0 - m00 - m11 + m22 vaut 4 z^2.
//        Elles sont donc toujours positives mais peuvent être petites
//        On choisit la plus grande ...

else if ( m00 > m11 && m00 > m22 ) {
        s = 2.0 * sqrt ( 1.0 + m00 - m11 - m22 ) ; // 4 x
        x = 0.25 * s ;             y = ( m01 + m10 ) / s ;
        z = ( m02 + m20 ) / s ;    w = ( m12 - m21 ) / s ;
}
else if ( m11 > m22 ) {
        s = 2.0 * sqrt ( 1.0 - m00 + m11 - m22 ) ; // 4 y
        x = ( m01 + m10 ) / s ;    y = 0.25 * s ;
        z = ( m12 + m21 ) / s ;    w = ( m02 - m20 ) / s ;
}
else {
        s = 2.0 * sqrt ( 1.0 - m00 - m11 + m22 ) ; // 4 z
        x = ( m02 + m20 ) / s ;    y = ( m12 + m21 ) / s ;
        z = 0.25 * s ;             w = ( m01 - m10 ) / s ;
}
```

6.2 Détection des poses illicites

Nous regrettons que MakeHuman puisse mettre l'avatar dans une pose incohérente telle que présentée à la figure 6.7. *A priori*, cela n'est pas gênant puisqu'il y a de grandes chances que la silhouette résultante soit proche de celle d'une pose correcte. Il serait cependant intéressant de pouvoir détecter une telle situation, afin de ne pas enregistrer de silhouettes quand cela se produit.

FIG. 6.7 – Pose illicite

La méthode proposée tente de détecter lorsqu'un maillage s'auto-intersecte. Les facettes à plus de trois côtés sont décomposées en triangles, de telle manière à transformer le maillage en un maillage ne contenant que des faces triangulaires. On se ramène ainsi au problème de la détection d'intersection entre deux triangles dans un espace tridimensionnel.

Deux triangles peuvent être dans quatre positions relatives différentes. Elles sont représentées à la figure 6.8. Seules les situations (a) et (b) sont considérées comme étant des intersections. Cette figure nous inspire le critère suivant :

Proposition. Deux triangles s'intersectent si et seulement si le point de percée d'une arrête de l'un dans de plan supporté par l'autre y est inclus.

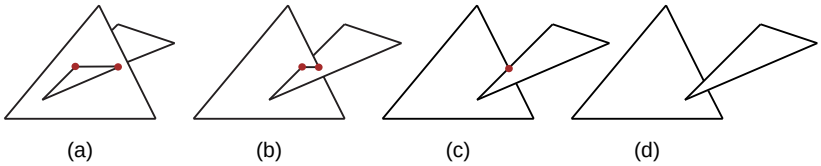


FIG. 6.8 – Les quatre positions relatives de deux triangles

Le problème auquel nous nous sommes ramenés consiste en la détection des intersections entre un segment de droite et un triangle.

6.2.1 Première approche

Exprimons l'existence d'une intersection entre le triangle ABC et le segment PQ. Ce sera le cas s'il existe un point X appartenant aux deux. Leurs équations sont établies à l'annexe D :

$$\begin{cases} X \in ABC \Leftrightarrow \quad X = aA + bB + cC & \text{avec} \quad \begin{cases} a \geq 0, b \geq 0, c \geq 0 \\ a + b + c = 1 \end{cases} \\ X \in PQ \Leftrightarrow \quad X = pP + qQ & \text{avec} \quad \begin{cases} p \geq 0, q \geq 0 \\ p + q = 1 \end{cases} \end{cases}$$

Il y aura donc intersection si :

$$pP + (1-p)Q = aA + bB + (1-a-b)C \qquad \text{avec} \qquad \begin{cases} 0 \leq p \leq 1 \\ a \geq 0, b \geq 0 \\ a + b \leq 1 \end{cases}$$

Résolvons ce système pour trouver a, b, c, p, q :

$$\begin{aligned} p(P-Q) + a(C-A) + b(C-B) &= C - Q \\ \Leftrightarrow \begin{pmatrix} P_x - Q_x & C_x - A_x & C_x - B_x \\ P_y - Q_y & C_y - A_y & C_y - B_y \\ P_z - Q_z & C_z - A_z & C_z - B_z \end{pmatrix} \begin{pmatrix} p \\ a \\ b \end{pmatrix} &= \begin{pmatrix} C_x - Q_x \\ C_y - Q_y \\ C_z - Q_z \end{pmatrix} \end{aligned} \tag{6.7}$$

Il s'agit d'un système de trois équations à trois inconnues. Il possède donc zéro ou une solution. On obtient ainsi le crière suivant : *Le segment PQ intersecte le triangle ABC si et seulement si le système 6.7 a une solution telle que* $0 \leq p \leq 1$, $a \geq 0$, $b \geq 0$ *et* $a + b \leq 1$.

6.2.2 Deuxième approche

Nous proposons d'utiliser une transformation linéaire $\mathcal{L}$ ramenant le triangle dans le plan xy.

$$\begin{cases} \overrightarrow{p_1} \xrightarrow[\mathcal{L}]{} \begin{pmatrix} 0 & 0 & 0 \end{pmatrix}^T \\ \overrightarrow{p_2} \xrightarrow[\mathcal{L}]{} \begin{pmatrix} 1 & 0 & 0 \end{pmatrix}^T \\ \overrightarrow{p_3} \xrightarrow[\mathcal{L}]{} \begin{pmatrix} 0 & 1 & 0 \end{pmatrix}^T \end{cases} \Leftrightarrow M * \begin{pmatrix} p_{1_x} & p_{2_x} & p_{3_x} \\ p_{1_y} & p_{2_y} & p_{3_y} \\ p_{1_z} & p_{2_z} & p_{3_z} \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \tag{6.8}$$

Il y aura intersection entre le segment et le triangle si et seulement si il y a intersection entre le segment et le plan contenant le triangle, et la transformation linéaire appliquée à ce point est dans le triangle $(0,0)$,$(0,1)$,$(1,0)$. L'équation du plan contenant le triangle est donnée par :

$$ax + by + cz + d = 0 \quad \text{avec} \quad \overrightarrow{\begin{pmatrix} a & b & c \end{pmatrix}} = (\overrightarrow{p_2} - \overrightarrow{p_1}) \wedge (\overrightarrow{p_3} - \overrightarrow{p_1}) \tag{6.9}$$

La détermination des coefficients de l'équation linéaire $\mathcal{L}$ nécessite l'inversion de la matrice $\begin{pmatrix} p_1 & p_2 & p_3 \end{pmatrix}$ dont le déterminant peut s'annuler lorsque les trois sommets du triangle sont alignés ou lorsque le plan de support du triangle passe par l'origine (apparition d'une relation linéaire entre les coordonnées des sommets). Pour éviter cela, nous commençons par translater le plan suivant sa normale :

$$\begin{cases} \overrightarrow{p'_1} = \overrightarrow{p_1} - k\overrightarrow{\begin{pmatrix} a & b & c \end{pmatrix}} \\ \overrightarrow{p'_2} = \overrightarrow{p_2} - k\overrightarrow{\begin{pmatrix} a & b & c \end{pmatrix}} \\ \overrightarrow{p'_3} = \overrightarrow{p_3} - k\overrightarrow{\begin{pmatrix} a & b & c \end{pmatrix}} \end{cases} \tag{6.10}$$

On cherche alors la matrice M telle que

$$\begin{cases} \overrightarrow{p'_1} \xrightarrow[\mathcal{L}]{} \begin{pmatrix} 0 & 0 & 0 \end{pmatrix}^T \\ \overrightarrow{p'_2} \xrightarrow[\mathcal{L}]{} \begin{pmatrix} 1 & 0 & 0 \end{pmatrix}^T \\ \overrightarrow{p'_3} \xrightarrow[\mathcal{L}]{} \begin{pmatrix} 0 & 1 & 0 \end{pmatrix}^T \end{cases} \Leftrightarrow M * \begin{pmatrix} p'_{1_x} & p'_{2_x} & p'_{3_x} \\ p'_{1_y} & p'_{2_y} & p'_{3_y} \\ p'_{1_z} & p'_{2_z} & p'_{3_z} \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \tag{6.11}$$

Comme démontré dans l'annexe C, la valeur de k rendant unitaire le déterminant de la matrice $\begin{pmatrix} p'_1 & p'_2 & p'_3 \end{pmatrix}$ est donnée par

$$\begin{vmatrix} p'_{1_x} & p'_{2_x} & p'_{3_x} \\ p'_{1_y} & p'_{2_y} & p'_{3_y} \\ p'_{1_z} & p'_{2_z} & p'_{3_z} \end{vmatrix} = 1 \Leftrightarrow k = \frac{D-1}{a^2+b^2+c^2} \quad \text{avec} \quad D = \begin{vmatrix} p_{1_x} & p_{2_x} & p_{3_x} \\ p_{1_y} & p_{2_y} & p_{3_y} \\ p_{1_z} & p_{2_z} & p_{3_z} \end{vmatrix} \tag{6.12}$$

Dans ce cas, la matrice $\begin{pmatrix} p'_1 & p'_2 & p'_3 \end{pmatrix}$ est inversible et on a :

$$M = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} p'_{1_x} & p'_{2_x} & p'_{3_x} \\ p'_{1_y} & p'_{2_y} & p'_{3_y} \\ p'_{1_z} & p'_{2_z} & p'_{3_z} \end{pmatrix}^{-1} \tag{6.13}$$

$$\Leftrightarrow M = \begin{pmatrix} p'_{1_z}\ p'_{3_y} - p'_{3_z}\ p'_{1_y} & p'_{1_x}\ p'_{3_z} - p'_{3_x}\ p'_{1_z} & p'_{1_y}\ p'_{3_x} - p'_{3_y}\ p'_{1_x} \\ p'_{1_y}\ p'_{2_z} - p'_{2_y}\ p'_{1_z} & p'_{1_z}\ p'_{2_x} - p'_{2_z}\ p'_{1_x} & p'_{1_x}\ p'_{2_y} - p'_{2_x}\ p'_{1_y} \\ 0 & 0 & 0 \end{pmatrix} \tag{6.14}$$

L'algorithme naïf déterminant si un maillage s'auto-intersecte est ainsi :

1. *transformer le maillage en un maillage triangulaire ;*
2. *pour chaque facette, déterminer l'équation du plan la contenant, le facteur de translation k et les 6 coefficients de la matrice M ;*
3. *pour chaque couple de faces f_a et f_b, déterminer si les deux facettes s'intersectent*
   (a) *en calculant pour chaque arête son intersection avec le plan supporté par le triangle duquel elle ne fait pas partie ;*
   (b) *en appliquant au point obtenu la translation $-k\ \overrightarrow{(a \quad b \quad c)}$;*
   (c) *en projetant le point translaté dans le plan xy par la relation $\mathcal{L}$;*
   (d) *en déterminant si le point obtenu est dans le triangle $(0,0)$,$(0,1)$,$(1,0)$;*
4. *conclure en se demandant si on a trouvé deux facettes qui s'intersectent.*

Cet algorithme est relativement efficace. En effet, le point 1 peut être effectué une fois pour toutes, puisque seuls les noeuds sont susceptibles d'être modifiés. De plus, les opérations coûteuses ont été reportées dans la phase linéaire (point 2), ne laissant qu'une charge de calcul réduite lors de la phase quadratique (point 3). Il y a cependant moyen de faire mieux.

6.2.3 Optimisation

En pratique, les diverses facettes du maillage ne sont que rarement proches les unes des autres. Tester si deux facettes sont proches est nettement moins côuteux que de tester si les facettes s'intersectent. Pour chaque facette, en plus de ce qui est mentionné au point 2 ci-dessus, on calcule sa boîte limite[1]. Au point 3, on commence par vérifier si les deux boîtes limites ont une intersection. Si tel n'est pas le cas, il ne sert à rien de tester si les deux facettes s'intersectent.

Grouper les facettes proches les unes des autres en paquets est un raffinement supplémentaire de la méthode. Si l'on pouvait couper le maillage en n parties de même taille de sorte qu'il soit impossible de trouver deux faces s'intersectant sans être dans la même partie, alors seuls les couples de facettes au sein d'une même partie resteraient à considérer. Le troisième point de l'algorithme décrit ci-dessus serait alors n fois plus rapide.

Jung, Shin et Choi proposent dans leur article l'utilisation d'une structure en *buckets*. Ils calculent la boîte limite du maillage qu'ils scindent récursivement en deux selon sa plus grande dimension. Les facettes à cheval sur la limite sont dupliquées afin de se retrouver dans les deux *buckets*. Le mécanisme s'arrête dès qu'il ne reste qu'un certain nombre arbitrairement prédéfini de faces au sein d'un *bucket*, ou lorsque toutes les facettes

[1] Axis Aligned Bounding Box

résultantes sont à cheval sur la limite. Les tests d'intersections sont alors réalisés entre facettes appartenant au même groupe (*bucket*). Nous avons testé cette méthode, qui s'est révélée inadaptée pour plusieurs raisons :

- la répartition en *buckets* est à recommencer chaque fois que le maillage change ;
- le nombre d'ensembles de facettes peut devenir important, nécessitant beaucoup de mémoire, un temps important pour leur allocation et libération ;
- le fait que certaines facettes se retrouvent dans plusieurs groupes implique que certaines intersections seront détectées plusieurs fois, rendant difficile un comptage de celles-ci ;
- les groupes ainsi formés sont sans signification, ce qui est regrettable puisque que cela rend difficile l'interprétation de l'ensemble d'intersections trouvées ;
- un nombre important d'intersections parasites entre facettes voisines est détecté.

L'approche choisie se base quant à elle sur l'information contenue dans le squelette appris. L'avatar étant piloté par 105 paramètres, le squelette classe les noeuds du maillage en 106 classes, les feuilles du graphe, suivant l'ensemble d'articulations les mouvant. Par rapport au modèle articulé, ces classes correspondent aux parties indéformables. Ces classes peuvent très bien convenir pour décrire les différents groupes de facettes. Celles étant définies sur des points appartenant à plusieurs classes[2] sont ignorées lors de la détection d'intersections car elles sont situées au niveau d'une articulation, zone du maillage susceptible d'être détériorée et donnant naissance à des intersections parasites. A l'opposé de la méthode de Jung, aucune intersection n'est recherchée au sein d'un même groupe, une partie d'un modèle articulé ne pouvant pas avoir d'intersection avec elle-même. Les intersections entre facettes sont donc recherchées entre groupes différents, pour autant que les boîtes limites des groupes ne soient pas dissociées[3].

Comparativement à l'approche de Jung et al., la répartition des facettes triangulaires en groupes peut se faire une fois pour toutes, le squelette étant invariant à la pose. Ici, il n'y a que 106 groupes de facettes, et aucune facette ne se retrouve dans plusieurs groupes. Il est ainsi possible de dire combien d'intersections ont été trouvées[4], ainsi que les zones entre lesquelles il y a

[2] 3710 facettes triangulaires sont dans ce cas.

[3] Cette précision est importante. Sans elle, l'algorithme serait $\mathcal{O}(n^2)$, tout comme l'algorithme naïf testant les intersections entre chaque paire de facettes.

[4] Bien évidemment, dès qu'une intersection a été trouvée entre deux zones, il n'est plus nécessaire de vérifier s'il y a intersection entre d'autres facettes de ces zones. Cependant, il peut être intéressant d'avoir un estimateur de l'importance de la collision.

intersection.

Enfin, remarquons que le rejet des silhouettes correspondant aux poses illicites peut poser problème, dès que l'on restreint les poses prenables par l'avatar. Ainsi, le troisième orteil est par défaut en collision avec les deux doigts voisins. Fixer les paramètres des orteils à leur valeur par défaut (0), conduirait à un rejet de toutes les silhouettes. On voit ici une application directe de la possibilité de connaître les parties entres lesquelles il y a intersection. Une autre application sera étudiée à la section 6.3.2.

Avant de passer à l'étude de techniques permettant de réduire le taux de rejet, une dernière remarque s'impose. L'algorithme que nous venons de décrire ne garantit rien. Certaines situations de collisions peuvent paraître licites, notamment si l'intersection se situe au niveau d'une articulation ou lorsque les limites des facettes d'une partie correspondent tout juste à la surface de l'autre partie. De telles situations seront cependant exceptionnelles. De plus, certaines situations licites peuvent être détectées comme illicites, de par la dégradation engendrée par la transformation appliquée au maillage pour le rendre triangulaire. Ces situations seront cependant proches de situations de collisions, de telle sorte que ces erreurs sont sans gravité.

6.3 Diminution du taux de rejet

La détection des poses illicites permet de n'enregistrer une silhouette que lorsque l'avatar est dans une pose correcte. Cependant, avant de déterminer si la pose envisagée est licite, il est nécessaire de récupérer le maillage, ce qui prend un temps non négligeable. De plus, il se peut que le taux de rejet des poses soit important. Dans ce cas, le débit de silhouettes produites se réduit de manière inacceptable.

Afin de limiter la quantité de poses illicites, deux méthodes ont été envisagées. La première tente de n'essayer une pose que si ses chances d'être licite sont suffisamment élevées. La seconde essaye de corriger la position de l'avatar lorsque celle-ci est impossible. Les deux méthodes peuvent être éventuellement combinées.

6.3.1 Méthode *a priori*

L'observation du caractère licite ou non des poses essayées précédemment fournit une information utile pour guider le choix d'une nouvelle pose. La création d'une base de données regroupant les paramètres des poses essayées précédemment ainsi que leur classe (licite ou non) est alors nécessaire.

Afin de pouvoir comparer une pose avec celles enregistrées dans la base de données, il est nécessaire de définir une distance entre poses. Nous noterons d_+ la distance de la pose envisagée par rapport à la pose licite la plus similaire et d_- la distance de la pose envisagée par rapport à la pose illicite la plus semblable.

Le but sera d'augmenter la densité de tuples licites tout en évitant de choisir des poses trop similaires. Un compromis doit ainsi être établi entre d_+ et d_-. Cela peut se faire en maximisant l'entropie[5] suivante :

$$E = \min\{\tau d_+, d_-\} \tag{6.15}$$

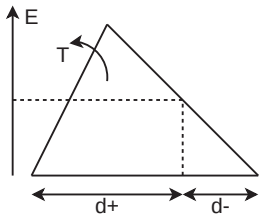


FIG. 6.9 – Illustration de la fonction d'entropie E (équation 6.15)

Le paramètre τ permet de régler la distance moyenne entre les poses licites par rapport à celle entre les poses illicites. Un exemple est illustré à la figure 6.10. On peut remarquer que l'utilisation de la méthode pour $\tau = 1$ force les différents points à s'écarter les uns des autres. La croissance de τ a manifestement l'effet escompté. Cependant, une valeur trop élevée de τ risque de provoquer l'oubli de certaines régions trop petites d'instances positives. Le choix d'une valeur de τ est donc également un compromis.

Cette méthode a trois inconvénients majeurs. Premièrement, le choix de la distance à utiliser n'est pas évident. Alors que le choix d'une distance euclidienne est naturel pour des points dans un espace bidimensionnel (exemple de la figure 6.10), une telle distance ne semble pas adaptée à l'espace à 105 dimensions des poses de l'avatar. Deuxièmement, aucun moyen pour fixer τ automatiquement n'a été trouvé. Troisièmement, il n'est pas raisonnable de mémoriser une base de données telle qu'envisagée ci-dessus. En effet, sa taille risque d'augmenter très rapidement, rendant son stockage difficile et rendant la détermination de d_+ et de d_- de plus en plus lente. Il est donc nécessaire de limiter la taille de la base de données ainsi que de prévoir un mécanisme de remplacement des tuples.

[5]On parle d'entropie car cette quantité est d'autant plus faible que la pose est proche d'une pose dont on connaît la classe. La valeur de E reflète donc l'incertitude sur la légitimité de la pose.

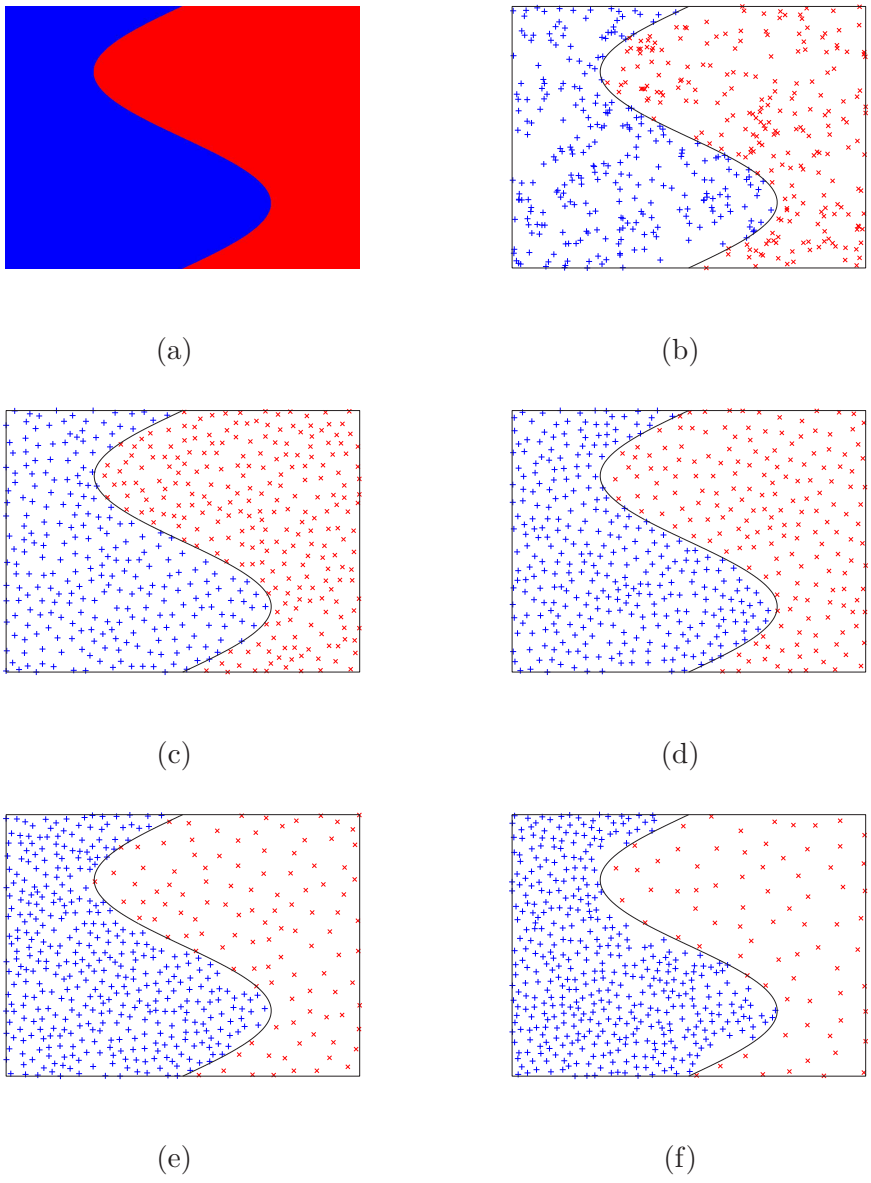


FIG. 6.10 – Illustration de la méthode *a priori* dans un espace euclidien bidimensionnel : (a) limite des deux classes ; (b) résultat d'un tirage aléatoire [54 % de +] ; (c) résultat de la méthode pour $\tau^2 = 1$ [51 % de +] ; (d) résultat de la méthode pour $\tau^2 = 2$ [63 % de +] ; (e) résultat de la méthode pour $\tau^2 = 5$ [77 % de +] ; (f) résultat de la méthode pour $\tau^2 = 10$ [85 % de +].

6.3.2 Méthode *a posteriori*

On peut essayer de corriger une pose détectée comme étant illicite en une pose licite. La motivation de cette démarche tient en la constatation suivante. Considérons deux tirages aléatoires consécutifs des paramètres de pose. Imaginons qu'à la suite du premier, l'avatar ait ses deux mains s'intersectant, il est ainsi en une position prohibée. Si l'on refait un nouveau tirage aléatoire d'une pose, on a de grandes chances d'aboutir de nouveau dans une position illicite, mais cette fois par exemple à cause des pieds. Il est clair que les chances d'aboutir à une position licite seraient augmentées si on pouvait garder les pieds dans la position du premier tirage. Il convient donc de ne refaire un tirage aléatoire que des paramètres associés aux poignets, coudes et épaules.

Algorithmiquement, il faut déterminer les parties de l'avatar entre lesquelles il y a collision, puis déterminer quels sont les paramètres ayant une influence sur la position relative de ces parties et leur associer une nouvelle valeur par tirage aléatoire. Les articulations mobilisant une certaine partie sont obtenues en parcourant l'arbre représentant le squelette depuis la feuille correspondant à cette partie jusqu'à la racine et en ne gardant que les noeuds internes atteints par la droite.

Illustrons ceci au travers d'un exemple. Considérons qu'une intersection a été détectée entre les parties 4 et 5 du modèle articulé présenté à la figure 6.2. Il est clair que seules des modifications apportées à la position des articulations C, B et D ont une chance de ramener le modèle articulé dans une position licite. Ces articulations sont celles mobilisant soit la partie 4, soit la partie 5, exclusivement. En effet, A, B, C sont les articulations mobilisant la partie 4 et A, D sont celles mobilisant la partie 5. L'articulation A n'est d'aucun intérêt puisqu'elle mobilise les deux parties, ne modifiant alors pas leur position relative.

En théorie, l'algorithme décrit ci-dessus doit toujours amener, après quelques itérations, le modèle articulé dans une position licite. L'impossibilité d'y arriver impliquerait l'inexistence d'une configuration séparant les deux parties en collision. Il n'y aurait alors aucune configuration licite du modèle articulé.

En pratique cependant, on observe que l'algorithme décrit ci-dessus peut de temps en temps aboutir à un blocage. La nature du problème n'est pas claire. On peut remettre en cause la précision de l'algorithme de détection des collisions, tout autant que l'adéquation du modèle articulé avec l'avatar (les articulations ne sont pas ponctuelles et les mouvements de torsion sont approximés comme des rotations).

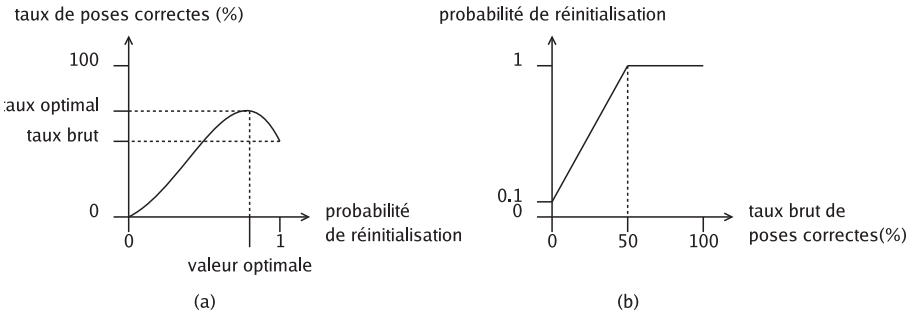


FIG. 6.11 – Illustration du raisonnement menant à la formule empirique 6.16.

Afin de garantir l'impossibilité de blocage du programme, une possibilité de repartir d'un nouveau tirage aléatoire a été introduite à chaque itération. Le programme pourra la choisir avec une certaine probabilité de réinitialisation déterminée dynamiquement.

Si cette probabilité est trop faible, l'algorithme risque d'aboutir à une situation de quasi blocage dont il aura du mal à sortir, mettant ainsi l'avatar dans beaucoup de poses incorrectes. Le taux de silhouettes générées sera alors faible, et tendra vers zéro si la probabilité de réinitialisation tend elle-même vers zéro. Si elle est grande, l'algorithme aura rarement le temps de corriger la pose avant de se réinitialiser. A la limite, quand la probabilité devient certaine, le taux de silhouettes générées tend vers le taux brut, la méthode étant alors désactivée. Le taux optimal est obtenu pour une probabilité de réinitialisation intermédiaire. Ce raisonnement est illustré graphiquement à la figure 6.11.a.

Intuitivement, si l'avatar se trouve dans une pose illicite et que la probabilité que cela arrive est inférieure à 0.5, on ne souhaitera pas corriger sa pose, mais on préférera réinitialiser le processus avec une nouvelle pose aléatoire. A l'opposé, plus la probabilité que cela arrive est grande, plus on préférera corriger la pose, tout en veillant à pouvoir repartir d'une nouvelle pose de temps en temps. La figure 6.11.b illustre ce raisonnement, pour la règle empirique suivante :

$$p_{\text{réinitialisation}} = \min \{ \ 1 \ , \ 0.1 + 1.8 \text{ taux brut } \} \tag{6.16}$$

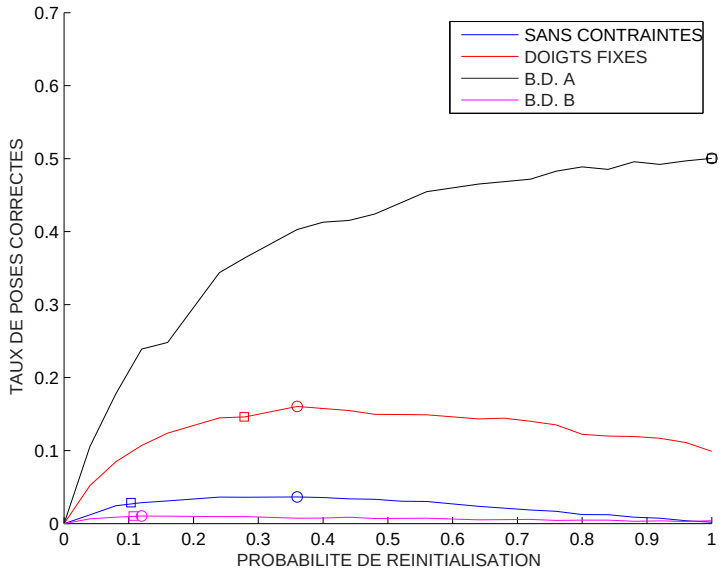


FIG. 6.12 – Taux de poses licites obtenus en fonction de la probabilité de réinitialisation pour quatre ensembles de poses. Les cercles pointent les taux maximaux, et les carrés pointent les taux obtenus grâce à la formule 6.16. Remarquez la similitude entre ces taux (ordonnée) qui constitue une justification *a posteriori* de la formule 6.16.

6.3.3 Résultats des deux méthodes

Les proportions données dans les tableaux ci-dessous correspondent aux taux de poses correctes rencontrées (1 - taux de rejet). Ils ont été obtenus en comptant le nombre de poses licites obtenues parmi 10000 poses tirées aléatoirement. Nous avons testé les taux pour diverses restrictions de l'ensemble de poses. Elles seront décrites en détails au chapitre 8.

Explicitons le paramétrage utilisé pour la méthode *a priori*. Nous avons utilisé une base de données de maximum 2500 tuples, et une valeur de τ égale à $2,0$. La distance entre les tuples est une distance euclidienne pondérée dans l'espace à 105 dimensions des paramètres de poses. Les poids sont choisis comme étant le produit entre l'angle parcouru par l'articulation et la taille de la partie mobilisée, par unité du paramètre. Ainsi, plus une articulation est importante, plus elle sera associée à un poids élevé.

Avatar complètement libre

	sans méth. *a posteriori*	avec méth. *a posteriori*
sans méth. *a priori*	0.25 %	2.61 %
avec méth. *a priori*	0.15 %	2.36 %

Doigts contraints à une pose fixe

	sans méth. *a posteriori*	avec méth. *a posteriori*
sans méth. *a priori*	10.51 %	15.22 %
avec méth. *a priori*	8.93 %	13.95 %

Cas de la base de données $\mathcal{A}$

	sans méth. *a posteriori*	avec méth. *a posteriori*
sans méth. *a priori*	49.99 %	51.43 %
avec méth. *a priori*	52.33 %	52.61 %

Cas de la base de données $\mathcal{B}$

	sans méth. *a posteriori*	avec méth. *a posteriori*
sans méth. *a priori*	0.40 %	0.98 %
avec méth. *a priori*	0.37 %	1.09 %

De manière évidente, la méthode *a posteriori* a toujours un impact positif. Dans certains cas, il est même significatif : elle permet de décupler la vitesse d'obtention des silhouettes voulues lorsque l'avatar est complètement libre.

Les résultats sont nettement moins encourageants en ce qui concerne la méthode *a priori*. On observe un faible gain dans certains cas, mais surtout un impact négatif lorsque les taux sont initialement faibles. Comme mentionné ci-dessus, la détermination du paramètre τ est problématique. Cependant, nous sommes enclins à penser que le problème provient du choix de la distance utilisée. Pour l'illustrer, considérons une pose illicite à cause d'un chevauchement entre deux doigts. Les articulations fautives ont malheureusement un poids faible. Si les autres articulations sont dans une configuration proche d'une pose précédemment détectée comme licite, la nouvelle pose sera erronément classée comme étant licite. La distance utilisée n'est donc pas adéquate.

En conclusion, nous conseillons de toujours utiliser la méthode *a posteriori*. En revanche, nous déconseillons l'utilisation de la méthode *a priori* tant qu'une meilleure distance n'a pas été trouvée, bien que le principe ait été démontré correct par la figure 6.10.

Chapitre 7

Le programme *Silhouette 1.0*

Sommaire

Décrivons à présent le programme mis au point. Nous l'avons appelé *Silhouette 1.0*.

7.1 Approches aléatoire et déterministe

A l'opposé de l'approche déterministe étudiée aux chapitres 3 et 4, nous approchons le problème selon une vue aléatoire. Les différentes poses seront tirées au sort suivant une distribution uniforme des différents paramètres dans un sous-intervalle. L'avantage principal de cette approche est sa simplicité à générer un nombre donné de silhouettes, car il suffit de faire autant de tirages aléatoires que l'on souhaite de silhouettes. Comparativement, avec l'approche déterministe, il aurait fallu estimer la tolérance en terme de la distance de Hausdorff entre les différentes silhouettes, telle qu'on obtienne le nombre souhaité de silhouettes, ce qui est beaucoup plus complexe à mettre en oeuvre. On pourrait reprocher à la méthode de se baser sur une distribution uniforme. Il ne s'agit cependant pas d'une restriction, car nous pouvons combiner divers ensembles de silhouettes, obtenus avec des intervalles différents pour les paramètres, dans la proportion adéquate pour

simuler une distribution voulue.

Il convient de remarquer la différence fondamentale existant entre les approches déterministes et aléatoires. Alors que la première tente de répartir uniformément les poses dans l'espace bidimensionnel des silhouettes et ainsi fournir certaines garanties quant aux caractéristiques des bases de données produites, la seconde cherche à espacer les poses prises dans un sous-espace de l'espace à 105 dimensions des paramètres de poses. Les caractéristiques des bases de données produites seront différentes, les utilisations qui peuvent en être faites sont donc également différentes.

L'approche déterministe fournit une base de données contenant toutes les silhouettes, à une certaine tolérance près. L'approche aléatoire ne le garantit pas, mais tend à y mettre les silhouettes de "toutes" les poses. L'approche déterministe serait donc idéale pour servir de base à l'apprentissage automatique de la caractérisation des silhouettes, si seulement la base de données était générable en un temps raisonnable. En revanche, l'approche aléatoire est idéale pour l'apprentissage de la relation liant la pose à la silhouette. Nous espérons en plus qu'elle puisse également remplacer dignement l'approche déterministe pour l'apprentissage automatique de la caractérisation des silhouettes.

7.2 Position et orientation de l'avatar

Afin de simplifier davantage le problème, nous allons faire l'hypothèse que nous ne sommes pas intéressés par les silhouettes obtenues lorsque l'avatar est très proche de la caméra, ni lorsqu'il est sur le côté du champ de vision. Dans ce cas, l'approximation classique considère que les silhouettes sont simplement translatées et dilatées dans le plan image. C'est pourquoi nous considérerons l'avatar à une position fixe.

Si besoin en est, l'approche aléatoire pourrait être également appliquée aux positions. Il faudrait alors déterminer une densité de probabilité adéquate. Une piste serait d'utiliser les résultats obtenus lors de l'étude de l'approche déterministe. On mettrait en correspondance la densité de probabilité recherchée avec la densité de positions souhaitée en chaque point de l'espace tridimensionnel. Celle-ci pourrait être approximée par le nombre de poses et d'orientations nécessaires en ce point ainsi que par la densité de positions.

En ce qui concerne l'orientation, nous considérons l'avatar dans une position "debout". Il tournera donc uniquement autour de l'axe vertical z. Afin de ne pas devoir manipuler chaque point du maillage, on préférera déplacer la caméra pour obtenir le même résultat. Si l'on souhaite une rotation d'angle

θ, et l'avatar placé en $\begin{pmatrix} x & y & z \end{pmatrix}$, il suffit de calculer la nouvelle matrice de projection par :

$$\begin{pmatrix} su \\ sv \\ s \end{pmatrix} = M' \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} \qquad \text{avec} \qquad M' = M \begin{pmatrix} \cos\theta & \sin\theta & 0 & x \\ -\sin\theta & \cos\theta & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

car

$$\begin{pmatrix} su \\ sv \\ s \end{pmatrix} = M \begin{pmatrix} p'_x \\ p'_y \\ p'_z \\ 1 \end{pmatrix} \qquad \text{avec} \qquad \begin{pmatrix} p'_x \\ p'_y \\ p'_z \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta & 0 & x \\ -\sin\theta & \cos\theta & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix}$$

7.3 Interaction avec MakeHuman

Nous souhaitions interagir avec MakeHuman pour répétitivement placer l'avatar dans une pose voulue et récupérer le maillage. Malheureusement, MakeHuman est uniquement pilotable à la souris. Il n'accepte aucun argument au démarrage, il n'est pas scriptable et il passe par des fichiers pour toutes ses opérations d'entrées/sorties. MakeHuman n'est donc pas utilisable tel quel.

À l'époque où nous avons débuté ce projet, MakeHuman 0.9 (la première version réellement utilisable) venait de voir le jour. Aucune documentation n'était disponible, le code-source n'était absolument pas commenté et le forum des développeurs n'existait pas encore. Nous avons donc dû nous plonger assidûment dans le code-source de MakeHuman pour comprendre comment interagir avec celui-ci.

Le programme MakeHuman se base sur deux bibliothèques, *animorph* et *mhgui*. Nous avions besoin des fichiers de données fournis avec MakeHuman, ainsi que de la bibliothèque *animorph*. C'est cette dernière qui contient le code nécessaire à transformer le maillage pour faire prendre à l'avatar l'apparence voulue, ainsi que le code du moteur de pose. Le programme MakeHuman ainsi que la bibliothèque *mhgui* ne nous sont d'aucune utilité ; cette dernière sert seulement pour l'interface graphique.

Il nous fallait donc créer une couche logicielle pour pouvoir communiquer entre les deux applications. Le programme MakeHuman étant écrit en C++ et souhaitant développer Silhouette en Java, l'interface entre les deux programmes devait être composée de deux modules. Nous avons testé trois mécanismes d'interfaçage.

La première interface mise au point était rudimentaire. Du côté natif, il s'agissait d'un programme prenant en argument les noms des fichiers .bs de description de la pose et de l'apparence ainsi que celui du fichier .obj dans lequel le maillage doit être enregistré. Du côté Java, nous écrivions les deux fichiers .bs, faisions un exec pour lancer le programme avec les bons paramètres, et une fois celui-ci terminé, nous lisions le fichier .obj pour obtenir le maillage. Cette première interface était particulièrement inefficace, car le temps nécessaire à l'initialisation du maillage est important comparativement au temps requis pour placer l'avatar dans la pose voulue.

La deuxième interface mise au point est une amélioration de la première. Cette fois, le module natif est initialisé une fois pour toutes et la communication entre les deux modules se fait via sockets unix. Les différentes données transitent toujours via fichiers, mais nous n'enregistrons plus les normales aux facettes dans le fichier .obj, faisant passer la taille des fichiers enregistrés de 1.8 Mo à 775 Ko. Cette façon de procéder est suffisamment rapide. Nous l'avons nommée MakeHumanLight. On peut cependant vouloir transférer les données via la communication réseau, mais il y a moyen de faire encore mieux.

La troisième interface réalisée se base sur JNI (Java Native Interface). L'appel de fonctions définies en Java exécute du code natif. Ainsi, il n'y a plus besoin de lancer un autre programme. La bibliothèque contenant le code natif utilise directement la bibliothèque *animorph*. Les données transitant entre la partie native et la partie Java ne passent par fichiers. Nous ne devons plus écrire le maillage sous forme d'un fichier .obj. Cette interface ne nécessite donc plus la conversion des nombres en chaînes de texte, d'où un gain important en rapidité et précision. Cette interface est appelée MakeHumanJni, elle est nettement plus rapide que MakeHumanLight grâce au fait que nous ne devons plus décoder le fichier .obj.

7.4 Calcul des silhouettes

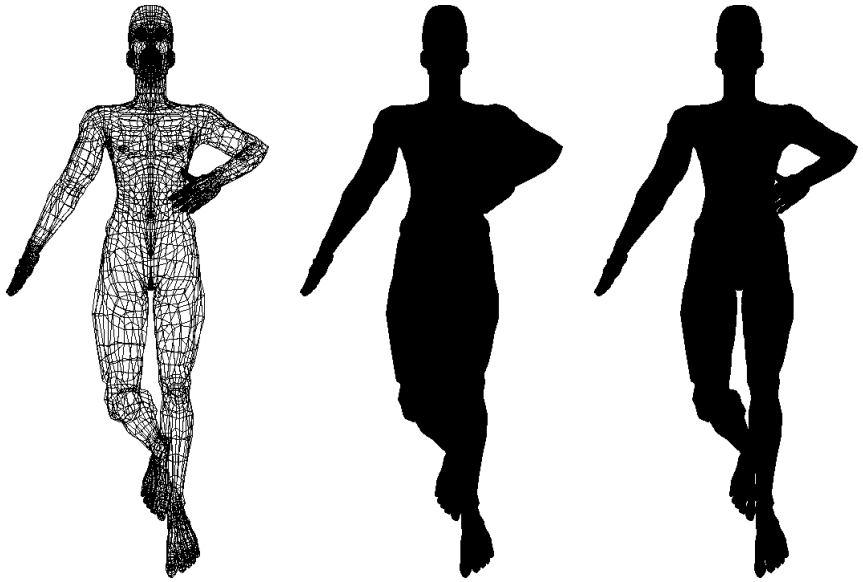
Le calcul de la silhouette se fait à partir du maillage. Chaque noeud est d'abord projeté. Ensuite, la boîte limite des projections est calculée, ce qui permet de s'allouer l'image dans laquelle la silhouette sera dessinée.

La première technique envisagée consistait en le dessin du polygone[1] reliant les points projetés de chaque facette. Dans une deuxième phase, un remplissage était réalisé. Les figures 7.1.a et 7.1.b illustrent respectivement ces deux étapes. On peut remarquer que cette méthode est inadéquate, bouchant des trous qui ne devraient pas l'être.

[1] L'algorithme utilisé pour le tracé des segments est celui de BRESENHAM.

C'est pourquoi la technique a été raffinée. Les silhouettes de chaque facette sont dessinées séparément par la première méthode, et l'ensemble est superposé. Le résultat est illustré à la figure 7.1.c.

Dans le cas où l'on souhaite obtenir des silhouettes annotées, 8 silhouettes sans annotation sont créées par la deuxième méthode, une pour chaque super-classe de facettes. Chaque silhouette binaire produit alors un bit du code de la silhouette annotée.



(a) (b) (c)

FIG. 7.1 – Trois modes de dessin : (a) dessin en fil de fer ; (b) dessin en fil de fer rempli ; (c) silhouette.

7.5 Annotation des silhouettes

La connaissance du squelette a, comme nous l'avons vu, de nombreuses applications dans le cadre de l'interaction avec MakeHuman. Une autre application, beaucoup plus intéressante en pratique bien qu'initialement inattendue, est l'annotation des silhouettes produites.

Rappelons que le squelette appris permet de classer les noeuds du maillage de l'avatar en 106 classes. Celles-ci correspondent aux parties rigides du modèle articulé associé à l'avatar. Par extension, le squelette permet la classification des facettes du maillage d'après les classes associées aux sommets.

Une facette peut se trouver dans autant de classes qu'elle a de sommets.

Lors de la génération de la silhouette, on peut ainsi associer à chaque pixel les classes des facettes projetées dont il fait partie[2]. Chaque pixel doit alors être codé sur au moins 106 bits, chaque bit spécifiant l'appartenance du pixel à une classe. Sans compression, une silhouette de 640x480 pixels nécessiterait 3975 ko, ce qui est beaucoup trop important.

FIG. 7.2 – Quelques annotations de silhouettes

Il nous semble raisonnable de coder chaque pixel sur un octet. Ce choix nous impose une restriction à 8 super-classes, qui peuvent être obtenues en groupant les classes entre elles. Pour les applications classiques, c'est suffisant. On peut en effet choisir des super-classes telles que tête, torse, bras gauche, main gauche, bras droit, main droite, jambe gauche et jambe droite qui sont les différentes parties susceptibles d'être souhaitées mises en évidence.

A titre d'illustration, considérons une hypothétique application à l'annotation des silhouettes. Supposons que l'on ait à notre disposition une base de données contenant des silhouettes annotées par les 8 super-classes citées ci-dessus. La mise au point d'un système avec lequel l'utilisateur peut interagir avec les mains nécessite leur situation dans la scène filmée. Un moyen d'y parvenir réside en l'extraction de la silhouette par soustraction d'arrière

[2]Nous adoptons une approche dans laquelle plusieurs classes peuvent être associées à un pixel, prenant en considération toutes les facettes, même celles à l'arrière du maillage. Une autre approche aurait été de ne considérer que les facettes du devant de l'avatar, par rapport à la caméra, mais aurait nécessité un algorithme supplémentaire d'élimination des facettes cachées.

plan, identification de la silhouette la plus ressemblante contenue dans la base de données et construction du masque adéquat en sélectionnant les pixels dont les bits correspondant aux mains sont actifs. Bien sûr, des techniques plus évoluées pourraient être utilisées, comme l'apprentissage d'un modèle donnant la position et la taille des zones correspondant à chaque main à partir de caractéristiques de la silhouette.

Remarque :

Du point de vue pratique, un détail reste à considérer. L'enregistrement des silhouettes, dans un format tel que BMP, permet de coder chaque pixel sur un octet, et une table associe à chaque code la couleur à afficher. Comment choisir cette table ?

Premièrement, il serait souhaitable de ne pas être obligé de relire le fichier soi-même pour récupérer l'annotation (code du pixel). On souhaite que l'annotation soit facilement récupérable lorsque la lecture du fichier se fait grâce à une fonction fournissant le code couleur RVB de chaque pixel. Cela implique que les bits de code se retrouvent tels quels dans le code RVB.

Deuxièmement, il serait pratique que l'annotation soit visiblement marquée. Ça requiert l'espacement maximal des couleurs ; autrement dit que chaque bit du code influence fortement la couleur.

Afin de générer la table efficacement tout en respectant les deux points de vue précédents, nous avons choisi de générer la table à partir de la règle suivante, n'effectuant que des décalages binaires :

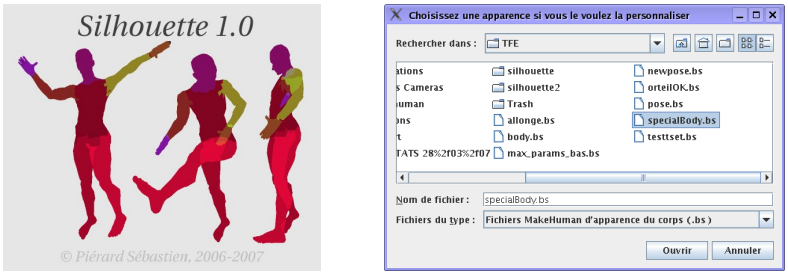
$$\begin{aligned} code &= c_7\ c_6\ c_5\ c_4\ c_3\ c_2\ c_1\ c_0 \\ R &= c_7\ c_6\ c_5\ c_4\ c_3\ c_2\ c_1\ c_0 \\ V &= c_4\ c_3\ c_2\ c_1\ c_0\ c_7\ c_6\ c_5 \\ B &= c_1\ c_0\ c_7\ c_6\ c_5\ c_4\ c_3\ c_2 \end{aligned}$$

7.6 Mode d'emploi

Après avoir affiché un panneau de démarrage, le programme demande à l'utilisateur de sélectionner un fichier de description de corps. Il s'agit d'un fichier *.bs* obtenu en exportant les paramètres d'apparence de l'avatar modélisé grâce à MakeHuman[3]. Si vous souhaitez simplement utiliser l'apparence par défaut, faites *annuler*.

Une fois les caractéristiques d'apparence connues, le programme effectue un apprentissage du squelette de l'avatar. Cette opération prend moins d'une dixaine de secondes, un peu plus si l'utilisateur laisse la case *preview* cochée (cette case est située au bas de la fenêtre). Lorsqu'elle est cochée, les différentes poses prises par l'avatar au cours de l'apprentissage sont dessinées. Une barre de progression indique l'état d'avancement de l'apprentissage.

[3]Veillez à être dans le mode *morphing* pour obtenir ce fichier. Si vous êtes en mode *pose*, vous obtiendrez un fichier *.bs* avec les paramètres de pose et non d'apparence !



(a) (b)

FIG. 7.3 – (a) Le panneau de démarrage du programme.
(b) L'invite de sélection du fichier d'apparence.

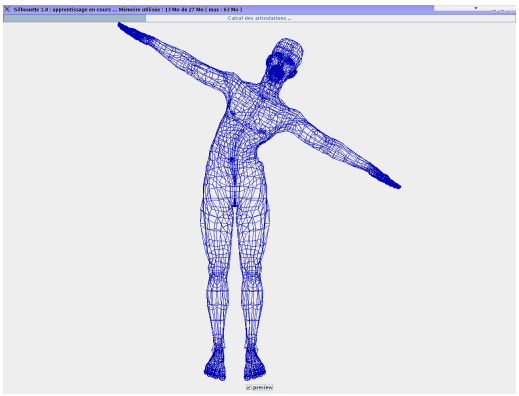


FIG. 7.4 – L'interface graphique de l'étape d'apprentissage.

Une fois l'apprentissage du squelette effectué, il est affiché sous la forme d'un arbre rétractile dans une nouvelle fenêtre (figure 7.5 a). On y retrouve également, outre les noms des paramètres, les différentes parties rigides ainsi que quelques données quantitatives telles que l'intervalle de valeurs admissibles pour chaque paramètre, le nombre de points contenu dans chaque partie, le nombre de points mobilisés par chaque articulation, les tailles des parties rigides, une borne sur les tailles des parties mobilisées par les différentes articulations ainsi qu'une estimation numérique de l'importance des paramètres.

De plus, l'utilisateur peut attribuer une classe aux différentes parties rigides. Ce sont elles qui définissent la couleur dans laquelle apparaîtra la partie dans la silhouette si elle est annotée. Un clic sur une case colorée fait apparaître un petit menu (figure 7.5 b) avec les huit choix possibles. Par défaut, les parties rigides ont été classifiées comme tête, torse, bras gauche,

main gauche, bras droit, main droite, jambe gauche et jambe droite.

Enfin, cette interface permet aussi de restreindre l'intervalle de valeurs parmi lesquelles les paramètres seront tirés au sort. Pour ce faire, un clic sur le bouton situé en regard d'un paramètre fait apparaître une petite fenêtre dans laquelle il est possible de restreindre l'intervalle au moyen de deux curseurs (figure 7.5 c).

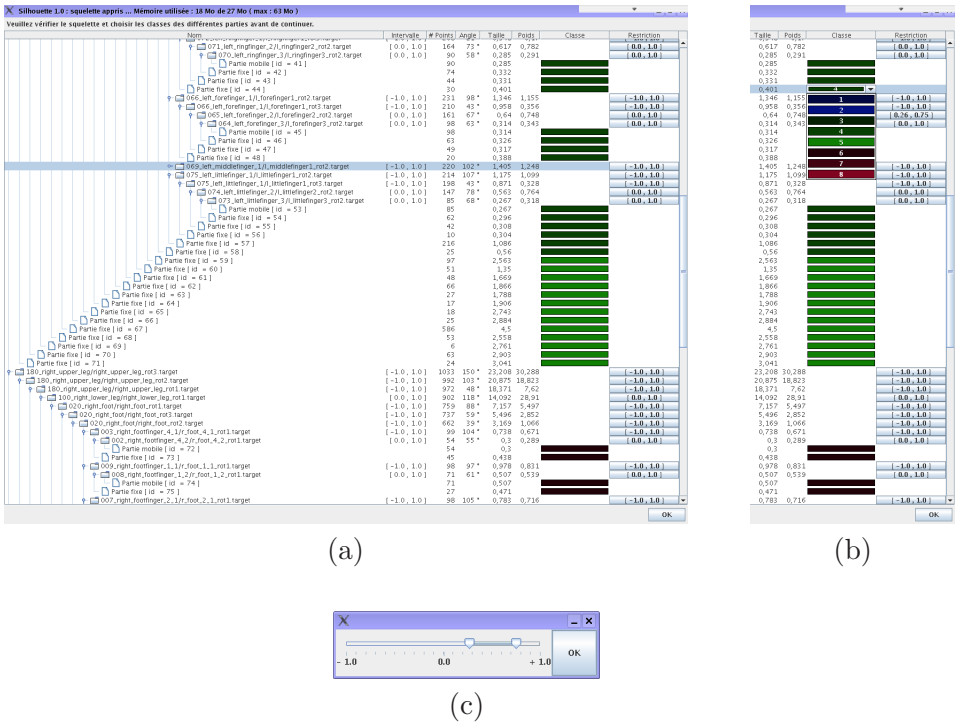


FIG. 7.5 – (a) L'interface graphique montrant le squelette.
(b) Le menu de choix de classe pour une partie rigide.
(c) La fenêtre permettant de régler l'intervalle du paramètre.

Une fois les différents réglages souhaités, une nouvelle fenêtre apparaît (figure 7.6). Elle permet de réaliser la configuration de l'étape de génération de la base de données. Les différents paramètres sont classés en trois catégories.

Premièrement viennent les choix concernant les poses. L'utilisateur a le choix entre les restrictions ayant été utilisées pour réaliser les quatre bases de données évoquées précédemment et la restriction personnalisée ayant été configurée à l'étape précédente.

Ensuite viennent les paramètres décrivant la caméra. Elle est spécifiée

par sa matrice de projection. Veillez à ce qu'elle regarde vers l'origine, car l'avatar y sera placé. De plus, veillez à ce qu'il n'y ait pas de paramètres infinis ou NaN dans sa définition sous peine d'avoir un comportement indéfini. Enfin, sachez qu'il faut écarter suffisamment la caméra de l'origine pour qu'elle ne puisse jamais se retrouver à l'intérieur de l'avatar. En effet, la silhouette serait alors de taille infinie et un manque de mémoire surviendrait. Un bouton permet d'obtenir la matrice de projection via un étalonnage de caméra. Nous y reviendrons dans la suite. Enfin, notez qu'une matrice de projection par défaut est proposée. C'est elle qui a été utilisée pour obtenir les différentes silhouettes tout au long du travail. Elle convient pour l'avatar d'apparence par défaut.

Enfin, arrivent les paramètres spécifiant les caractéristiques de la base de données à produire. On y retrouve notamment le nombre de silhouettes souhaitées, l'option permettant ou non d'annoter les silhouettes, la possibilité d'enregistrer toutes les silhouettes ou seulement les silhouettes correspondant aux poses licites, et dans ce cas, la possibilité d'activer ou non les méthodes d'accélération *a priori* et *a posteriori*.

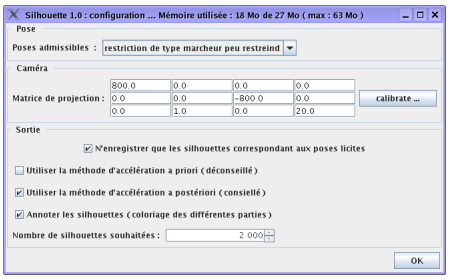
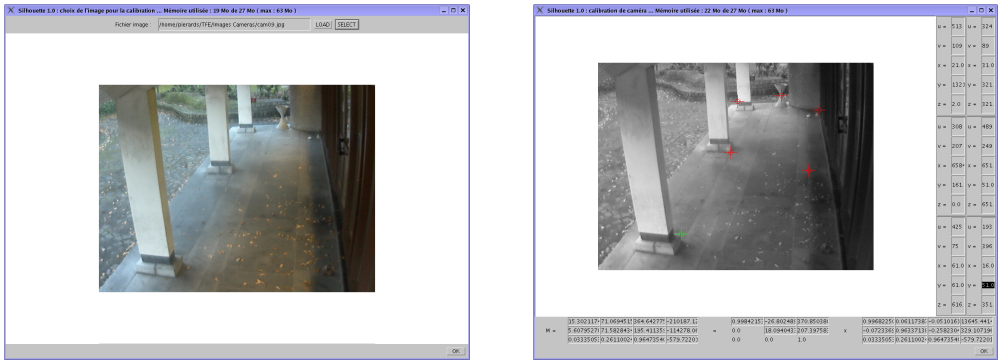


FIG. 7.6 – La fenêtre de configuration.

Dans le cas où l'utilisateur souhaite obtenir la matrice de projection via un étalonnage de caméra, une première fenêtre (figure 7.7 a) permet de sélectionner l'image à partir de laquelle l'étalonnage doit être effectué. Veillez à sélectionner un fichier qui est est une image dans un format classique, tel que jpg ou gif[4]. Un fois l'image choisie, une deuxième fenêtre (figure 7.7 b) permet de spécifier les coordonnées bidimensionnelles et tridimensionnelles des six points participant à l'étalonnage. A chaque point est associé un curseur pointant sur le pixel correspondant de l'image. Ces curseurs peuvent être déplacés avec la souris. La matrice de projection est mise à jour et factorisée en temps réel. Une fois l'étalonnage effectué, vous reviendrez au

[4]La restriction provient des capacités du *toolkit* par défaut de Java à comprendre les différents formats.

panneau de configuration.



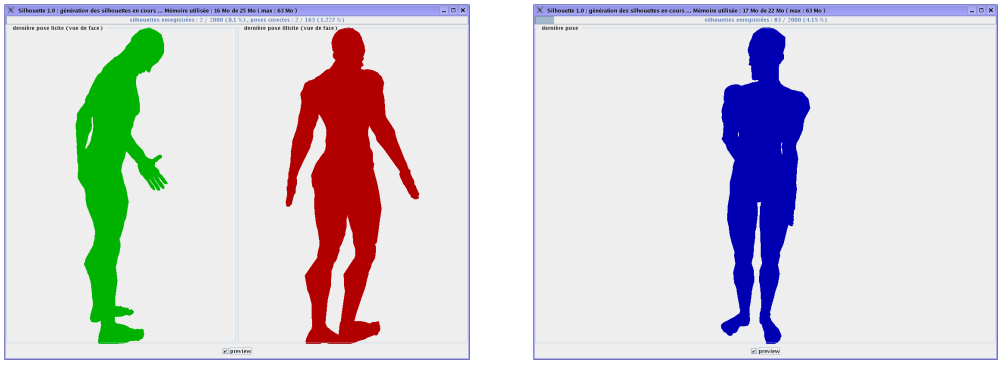
(a) (b)

FIG. 7.7 – Interface graphique de l'étalonnage de caméra :
(a) sélection de l'image à partir de laquelle travailler ;
(b) choix des positions des 6 points servant à l'étalonnage.

Dès que la configuration est terminée, la génération de la base de données commence. Les fichiers seront enregistrés dans `/tmp/output/`. Aucun fichier ne sera produit si ce répertoire n'existe pas. Il est préférable qu'il soit initialement vide, mais le programme ne devrait normalement écraser aucun fichier s'y trouvant. Un fichier *.xml* représentant le squelette est généré ainsi qu'un autre regroupant les différents paramètres (*targets*) avec les valeurs des bornes extrêmes et des bornes restreintes. Pour chaque silhouette produite, un fichier *.bs* est généré avec la pose correspondante, et la silhouette est enregistrée dans un fichier *.bmp*. Ainsi, il est possible de retrouver toutes les données de base à partir desquelles les silhouettes ont été produites.

Suivant que les poses illicites soient ou non exclues, une fenêtre différente apparaîtra (figures 7.8 a et b). L'interface permet de visualiser les silhouettes produites, si demandé. Une case à cocher située en bas de la fenêtre permet d'acter la prévisualisation. Les silhouettes montrées ne reflètent pas l'orientation de l'avatar lorsque l'interface utilisée est celle correspondant au souhait d'écarter les poses illicites. Dans ce cas, les silhouettes produites peuvent différer des silhouettes affichées. Aucune des deux interfaces ne permet de montrer les silhouettes annotées. Une barre de progression indique l'avancement de la tâche.

D'une manière générale, on passe d'une fenêtre à l'autre grâce à un bouton *OK* situé en bas à droite. La fermeture d'une fenêtre provoque l'arrêt du programme. Une jauge de la mémoire utilisée par les différents objets Java



(a) (b)

FIG. 7.8 – Interface graphique de la génération de silhouettes :
(a) lorsque seules les poses valides sont gardées ;.
(b) lorsque toutes les poses sont gardées.

est régulièrement mise à jour dans la barre de titre des fenêtres. Notons au passage que la quantité de mémoire consommée ne tient pas compte de la mémoire utilisée par MakeHuman.

Chapitre 8

Application, résultats et perspectives

Sommaire

Dans ce chapitre, nous nous intéresserons à une application pratique du générateur de silhouettes mis au point, nous démontrerons son utilité et présenterons quelques pistes de recherche potentiellement intéressantes. Ce chapitre sera également l'occasion de décrire en détails les différentes bases de données avec lesquelles nous avons travaillé.

8.1 Application

La génération automatique de silhouettes humaines synthétiques a notamment une forte utilité dans le domaine de la vidéo-surveillance. En effet, de tels systèmes impliquent la détection de personnes présentes devant des caméras. Un tel système est développé par le service de *Télécommunications et d'Imagerie* de l'ULG.

Dans une première étape, les différentes images composant le flux vidéo sont binarisées, mettant en évidence les pixels appartenant à l'avant-plan.

Ensuite, un éclatement en parties connexes fait apparaître les silhouettes des différents objets de l'avant-plan. Celles-ci sont alors finalement classifiées en silhouettes humaines ou non.

Pour ce faire, les silhouettes sont remplies avant de leur appliquer un opérateur morphologique donnant l'ensemble des rectangles inscrits maximaux non recouvrants. A ce niveau, aucune information n'est perdue. Un certain nombre de ces rectangles sont alors choisis aléatoirement, typiquement entre 100 et 200. Ils sont caractérisés par 6 grandeurs avant d'être classifiés au moyen d'une forêt d'arbres extrêmement aléatoires (*extra-trees*). La décision est prise selon que la majorité des arbres votent ou non pour une silhouette humaine.

Le modèle classificateur est construit par apprentissage d'une base de données de silhouettes préalablement classifiées. Puisque cette méthode est invariante à la fois à la position de la silhouette au sein de l'image et à sa taille, la base de données de silhouettes utilisée peut être obtenue pas un générateur indifférent à la translation et à la dilatation des silhouettes. La méthode utilisée pour obtenir un ensemble de positions au chapitre 3 est ainsi adéquate.

8.2 Les différentes bases de données

Dans le cadre du travail, nous avons été amenés à travailler avec plusieurs bases de données de silhouettes, que nous décrivons dans cette section. Elles se répartissent principalement en deux catégories : celles contenant des silhouettes synthétiques, et celles contenant des silhouettes réelles.

8.2.1 Les bases de données de silhouettes réelles

Dans le cadre de la détection de personnes dans un flux vidéo, les bases de données utilisées contiennent deux ensembles de silhouettes :

- un ensemble de silhouettes humaines (positifs) ;
- et un ensemble de silhouettes non humaines (négatifs).

Deux bases de données de silhouettes réelles avaient été constituées manuellement :

- une base de données d'apprentissage (1245 positifs, 1924 négatifs) ;
- et une base de données de test (1535 positifs, 1212 négatifs).

8.2.2 Les bases de données de silhouettes synthétiques

Afin d'évaluer l'impact de notre générateur pour le problème considéré, nous avons constitué quelques bases de données. L'application du modèle appris à la base de données de test, évoquée ci-dessus, a fourni des résultats objectifs. Pour constituer les différentes bases de données, nous avons regroupé, d'une part, les silhouettes humaines fournies par notre générateur (en utilisant l'avatar par défaut) avec, d'autre part, les silhouettes non humaines de la base de données de référence. Voyons à présent quels ensembles de silhouettes ont été testés.

Nous avons commencé par générer une base de données contenant des silhouettes de l'avatar non contraint. Chaque paramètre est choisi aléatoirement suivant une densité de probabilité uniforme dans l'intervalle des valeurs admises par MakeHuman pour celui-ci. Puis, en vue de diminuer le taux de rejet, nous avons contraint les doigts des mains et des pieds à une position fixe. Un échantillon de cette base de données est représenté à la figure 8.4. Clairement, la distribution de ces silhouettes ne correspond pas du tout à une distribution naturelle de silhouettes humaines. L'impact de cette base de données n'a donc pas été testé.

Afin d'y remédier, une nouvelle base de données a été constituée. Comme auparavant, chaque paramètre est choisi suivant une densité de probabilité uniforme, mais cette fois, dans un sous-intervalle des valeurs possibles. Les sous-intervalles ont été déterminés manuellement grâce à l'interface graphique de MakeHuman, pour contraindre l'avatar à se trouver dans une position de marche. Les plages acceptables pour les paramètres des jambes, du torse et de la tête ont donc été restreintes. Celles correspondant aux articulations des bras restent inchangées dans la mesure où un marcheur peut placer librement ses bras sans pour autant perdre l'équilibre. Un échantillon de cette base de données est représenté à la figure 8.5. Dans la suite, nous noterons cette base de données $\mathcal{A}$.

Une fois testé en "*live*" le modèle obtenu par apprentissage de cette dernière base de données, nous nous sommes rendus compte que les silhouettes de personnes ayant les bras le long du corps n'étaient jamais reconnues comme étant des silhouettes humaines. C'est pourquoi nous avons décidé de contraindre encore plus les poses de l'avatar pour obtenir de telles silhouettes. De plus, le torse nous semblait avoir une souplesse trop importante. Cette fois, ce sont donc les paramètres du torse, des épaules et des bras qui ont vu leur plage se réduire. La figure 8.6 montre un échantillon de la nouvelle base de données. Dans la suite, nous la noterons $\mathcal{B}$.

Un comparatif des valeurs pouvant être prises par les différents pa-

ramètres dans les diverses bases de données est fourni à la figure 8.1.

8.2.3 Les bases de données hybrides

Utilisée seule, la base de données $\mathcal{B}$ avait de meilleures performances que la la base de données $\mathcal{A}$. Cependant, le fait d'avoir contraint davantage la position des bras a une conséquence néfaste. Plus aucune silhouette avec les bras en l'air ne se trouve dans l'ensemble d'apprentissage. Si l'on rencontre de telles silhouettes, les chances de les classer comme humaines sont faibles.

Nous avons donc eu la curiosité d'essayer des bases de données hybrides, construites en mélangeant dans diverses proportions les bases de données $\mathcal{A}$ et $\mathcal{B}$. Théoriquement, il s'agit de changer la distribution statistique des poses. En combinant une infinité de bases de données, il y aurait moyen d'obtenir toute distribution voulue.

8.3 Résultats

Nous qualifions de positives les silhouettes humaines et de négatives les silhouettes non humaines. Rappelons que $\mathcal{A}$ est la base de données un peu restreinte et que $\mathcal{B}$ est la base de donnée fort restreinte. Voici les résultats obtenus :

	% de positifs reconnus	% de négatifs reconnus
b.d. de référence	83.06 %	99.01 %
2000 $\mathcal{A}$	16.16 %	94.72 %
1000 $\mathcal{A}$ + 1000 $\mathcal{B}$	31.73 %	97.19 %
2000 $\mathcal{B}$	39.61 %	99.17 %
2000 $\mathcal{A}$ + 1000 $\mathcal{B}$	43.13 %	92.57 %
2000 $\mathcal{A}$ + 2000 $\mathcal{B}$	60.46 %	91.17 %
3500 $\mathcal{B}$	61.43 %	98.27 %
2000 $\mathcal{A}$ + 3500 $\mathcal{B}$	73.63 %	88.37 %

Au vu de ces résultats, plusieurs conclusions peuvent être données.

Premièrement, nous remarquons que la base de données $\mathcal{B}$ utilisée seule a permis de construire un meilleur modèle que la base de données $\mathcal{A}$. Nous mettons ceci en relation avec le fait que la base de données de test contient des silhouettes de poses proches de celles utilisées pour construire la base de données $\mathcal{B}$. On aurait ainsi, à première vue, intérêt à construire une base de données d'apprentissage reflétant la distribution probabiliste des poses que le système devra reconnaître.

paramètre	libre	doigts fixes	b.d. $\mathcal{A}$	b.d. $\mathcal{B}$
360_upper_torso/torso2_rot1				
340_lower_torso/torso3_rot3				
360_upper_torso/torso2_rot3				
340_lower_torso/torso3_rot2				
360_upper_torso/torso2_rot2				
320_neck/neck_rot1				
300_head/head_rot3				
300_head/head_rot1				
300_head/head_rot2				
280_left_collar/l_collar_rot2				
280_left_collar/l_collar_rot3				
280_left_collar/l_collar_rot1				
240_left_upper_arm/l_upper_arm_rot2				
240_left_upper_arm/l_upper_arm_rot3				
240_left_upper_arm/l_upper_arm_rot1				
160_left_lower_arm/l_lower_arm_rot1				
080_left_hand/l_hand_rot3				
080_left_hand/l_hand_rot1				
080_left_hand/l_hand_rot2				
063_left_pollex_1/l_pollex1_rot1				
063_left_pollex_1/l_pollex1_rot3				
062_left_pollex_2/l_pollex2_rot{1,2}				
061_left_pollex_3/l_pollex3_rot1				
072_left_ringfinger_1/l_ringfinger1_rot{2,3}				
071_left_ringfinger_2/l_ringfinger2_rot2				
070_left_ringfinger_3/l_ringfinger3_rot2				
066_left_forefinger_1/l_forefinger1_rot{2,3}				
065_left_forefinger_2/l_forefinger2_rot2				
064_left_forefinger_3/l_forefinger3_rot2				
069_left_middlefinger_1/l_middlefinger1_rot{2,3}				
068_left_middlefinger_2/l_middlefinger2_rot2				
067_left_middlefinger_3/l_middlefinger3_rot2				
075_left_littlefinger_1/l_littlefinger1_rot{2,3}				
074_left_littlefinger_2/l_littlefinger2_rot2				
073_left_littlefinger_3/l_littlefinger3_rot2				
200_left_upper_leg/l_upper_leg_rot3				
200_left_upper_leg/l_upper_leg_rot2				
200_left_upper_leg/l_upper_leg_rot1				
120_left_lower_leg/l_lower_leg_rot1				
040_left_foot/l_foot_rot1				
040_left_foot/l_foot_rot3				
040_left_foot/l_foot_rot2				
024_left_footfinger_4_1/l_foot_4_1_rot1				
023_left_footfinger_4_2/l_foot_4_2_rot1				
030_left_footfinger_1_1/l_foot_1_1_rot1				
029_left_footfinger_1_2/l_foot_1_2_rot1				
028_left_footfinger_2_1/l_foot_2_1_rot1				
027_left_footfinger_2_2/l_foot_2_2_rot1				
026_left_footfinger_3_1/l_foot_3_1_rot1				
025_left_footfinger_3_2/l_foot_3_2_rot1				
022_left_footfinger_5_1/l_foot_5_1_rot1				
021_left_footfinger_5_2/l_foot_5_2_rot1				

FIG. 8.1 – Les différents paramètres de pose de l'avatar, avec leurs plages dans les différentes bases de données. Les intervalles sont tous représentés par rapport à [-1 , +1]. Pour la clarté de la table, les paramètres de droite ont été omis (ils sont obtenus par symétrie avec leur homologue de gauche) et certaines lignes identiques ont été regroupées. Remarquez que le paramètre *026_left_footfinger_3_1/l_foot_3_1_rot1* est forcé à la valeur 0.085 pour rendre le maillage licite.

Deuxièmement, nous constatons la faible influence (11 %) de la base de données d'apprentissage utilisée sur la reconnaissance des silhouettes non humaines. On peut se demander si ce n'est pas dû à la similitude des objets non-humains testés avec ceux utilisés afin de réaliser l'apprentissage. Quoi qu'il en soit, il en résulte que l'on peut tenter diverses bases de données en ne se souciant que de l'impact sur la reconnaissance des silhouettes humaines.

Troisièmement, il est clair que l'augmentation du nombre de silhouettes humaines utilisées lors de l'apprentissage a un impact positif. Ce phénomène est plus marqué pour la base de données $\mathcal{B}$ que $\mathcal{A}$. Manifestement, nous n'avons pas atteint le seuil de surapprentissage. Il serait intéressant de faire davantage de tests pour déterminer la proportion idéale des bases $\mathcal{A}$ et $\mathcal{B}$ dans la base hybride, ainsi que pour mettre en évidence le nombre optimal de silhouettes à utiliser.

Il faut relativiser les résultats obtenus, qui paraissent à première vue excellents, et ce pour plusieurs raisons.

D'un côté, sans aucune connaissance *a priori* de la proportion de silhouettes humaines contenues dans la séquence de test, un modèle identifiant aléatoirement une silhouette comme humaine, avec une probabilité de 50 %, reconnaîtrait correctement les silhouettes humaines dans 50 % des cas. Du point de vue du nombre de silhouettes humaines reconnues comme telles, les quatre premiers ensembles de silhouettes donnent des résultats médiocres. La méthode utilisée avec un ensemble inadéquat d'apprentissage se révèle donc particulièrement mauvaise.

De l'autre côté se pose la question de la génération de l'ensemble de silhouettes non humaines nécessaire à l'apprentissage. L'idéal serait d'avoir l'ensemble complémentaire de celui contenant les silhouettes humaines. Cependant, il comporterait alors un nombre prohibitif d'éléments. D'autre part, essayer de mettre au point un générateur de silhouettes non humaines semble irréalisable, vu les difficultés rencontrées lors de la mise au point d'un générateur de silhouettes humaines, ensemble qui est bien mieux caractérisable.

Heureusement, quelques pistes existent pour améliorer le processus de détection. Voyons-en deux, qui peuvent éventuellement être combinées.

Premièrement, le classificateur utilisé prend sa décision sur base du vote majoritaire des différents arbres. Si l'utilité visée est l'enregistrement des seules parties de séquences vidéo qui sont susceptibles de contenir une personne, on pourrait caractériser une silhouette avec une probabilité d'être humaine, en fonction du nombre relatif d'arbres qui la considèrent comme

telle. Lors de la classification, on pourrait garder les parties de séquences contenant des silhouettes dont la probabilité d'être humaine est supérieure à un seuil arbitraire. Ce seuil est d'autant plus inférieur à la moitié que l'on souhaite une haute probabilité de garder toutes les parties de séquences contenant une personne.

Deuxièmement, nous avons travaillé en classifiant chaque silhouette séparément. Or, pour un système de surveillance, l'intérêt serait de déterminer globalement si la séquence observée contient une personne. La décision devrait ainsi être prise sur l'ensemble des silhouettes observées pour un même élément de l'avant plan. L'utilisation d'une méthode de suivi (*tracking*) devrait ainsi gommer les quelques poses non reconnues, du moins si la personne n'est pas immobile.

Enfin, quelques différences sont visibles entre les silhouettes obtenues lors de la reconnaissance et celles générées et utilisées lors de l'apprentissage. Par exemple, les premières sont plus "épaisses" que les secondes, de par la présence de vêtements sur la personne passant devant la caméra. L'utilisation d'une version future de MakeHuman devrait pallier à cet inconvénient puisque celui-ci devrait inclure des outils pour habiller l'avatar. Une autre piste d'investigations concerne l'apparence de l'avatar, que nous avons gardée initiale. Inclure des variations entre homme et femme et faire varier la corpulence enrichirait la base de données d'apprentissage.

8.4 Perspectives

Tout au long du projet, nous avons essayé de rester les plus indépendants possible par rapport à MakeHuman. Il est donc envisageable de substituer à l'avatar de MakeHuman un avatar personnalisé. Aujourd'hui, il est possible d'obtenir un avatar ressemblant à une personne en quelques minutes. Il y aurait donc moyen d'obtenir un meilleur modèle si les personnes à reconnaître sont connues à l'avance.

D'autres perspectives intéressantes concernent l'information qui peut être extraite des silhouettes humaines. Nous avons clairement illustré l'utilité de l'annotation des silhouettes, permettant de retrouver efficacement n'importe quelle partie souhaitée d'une silhouette (par exemple la tête ou les mains) en identifiant la silhouette la plus ressemblante de la base de données. Bien évidemment, on peut imaginer faire utilisation de méthodes d'apprentissage à ce niveau.

Dans le même ordre d'idées, le logiciel Silhouette enregistre systémati-

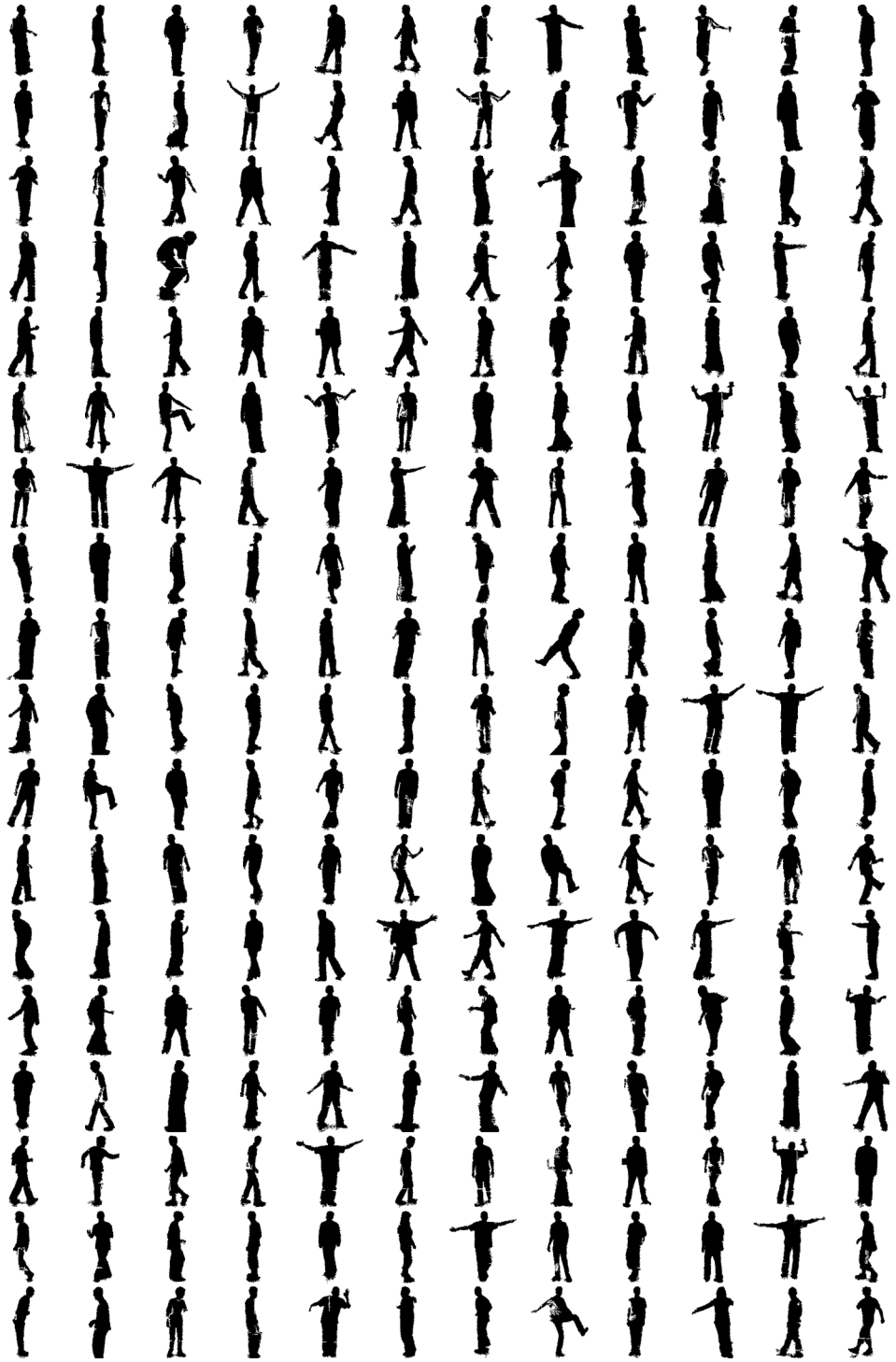


FIG. 8.2 – Échantillon de la base de données de référence

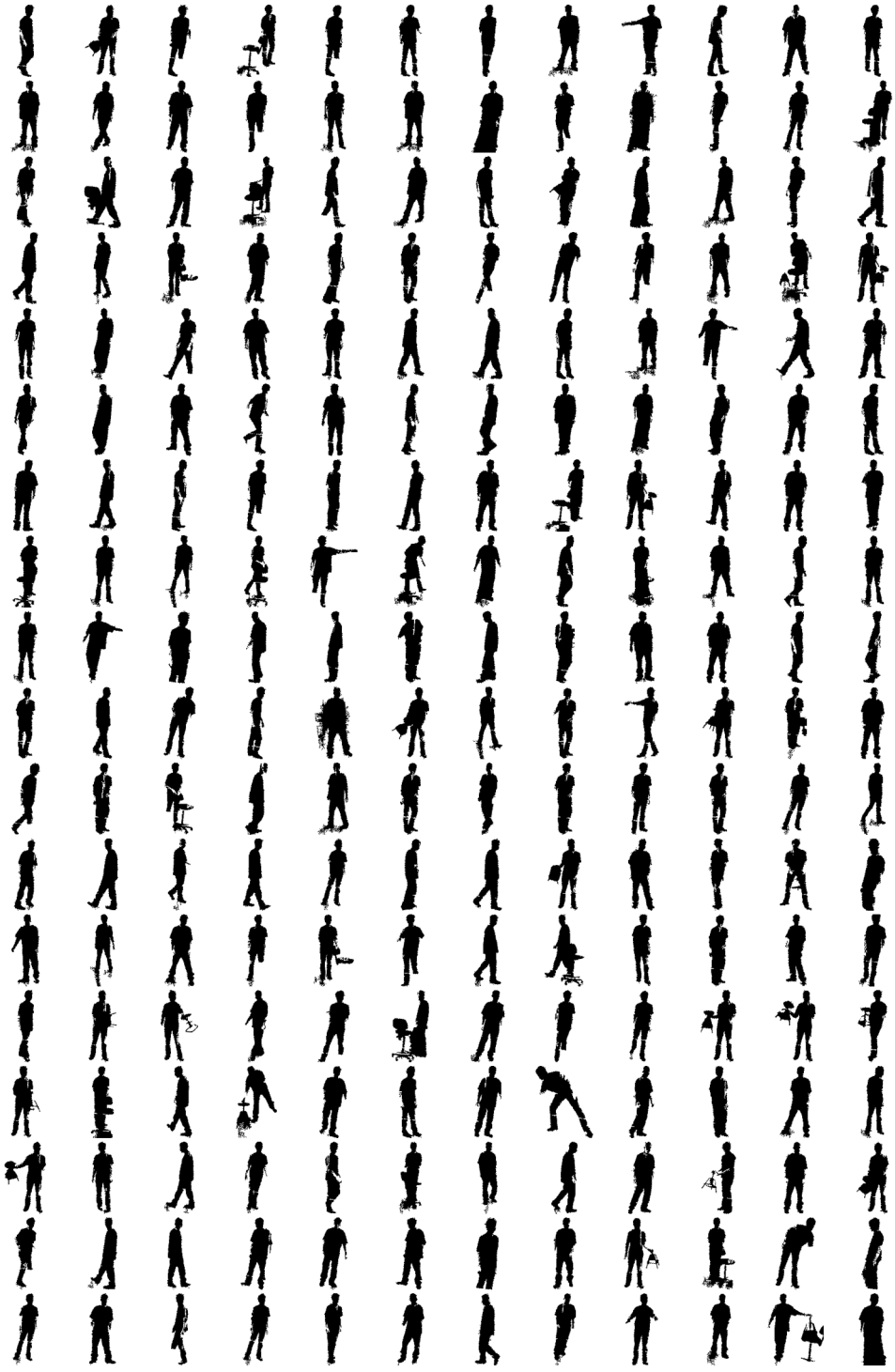


FIG. 8.3 – Échantillon de la base de données de test

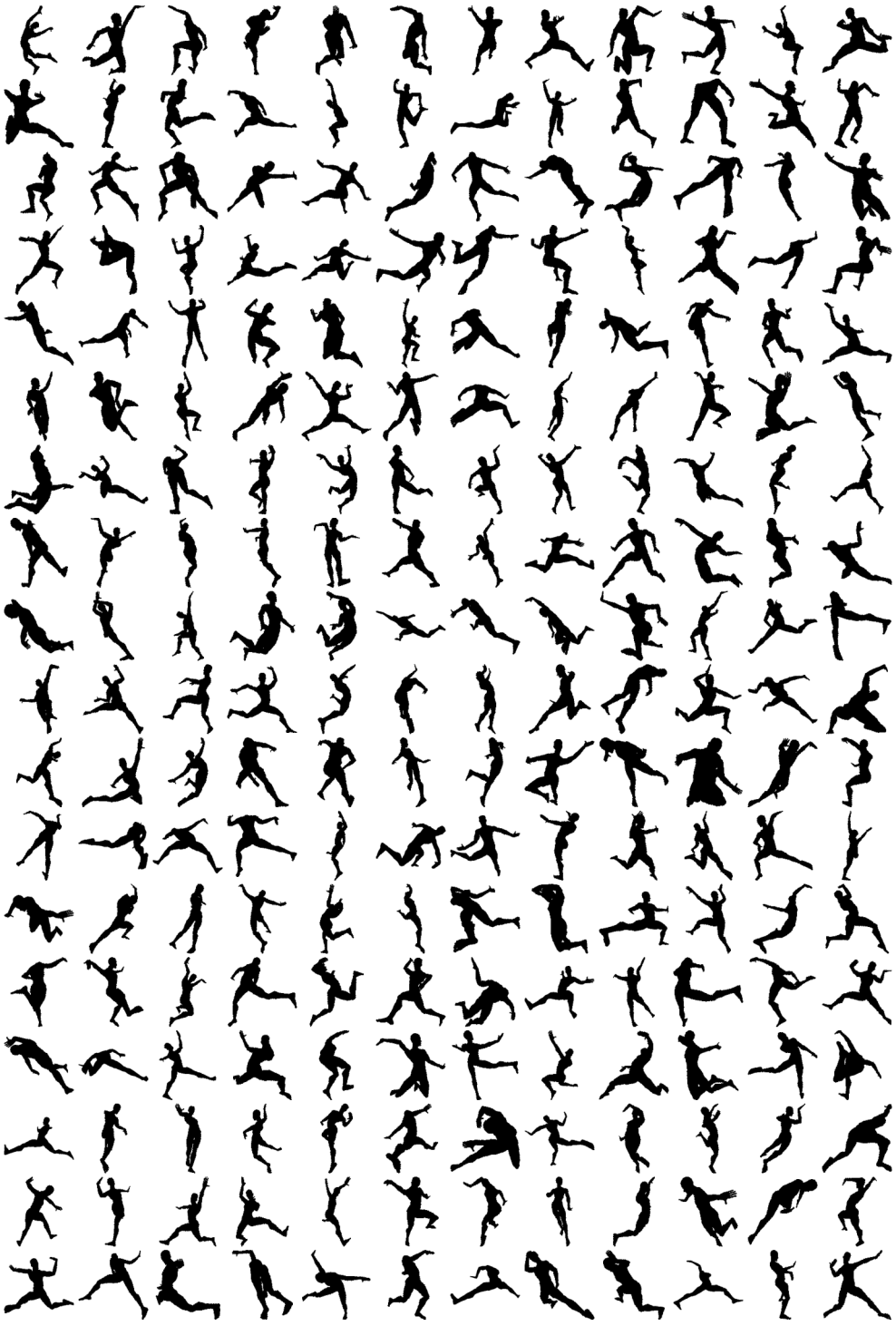


FIG. 8.4 – Échantillon de la base de données où seuls les doigts sont fixés

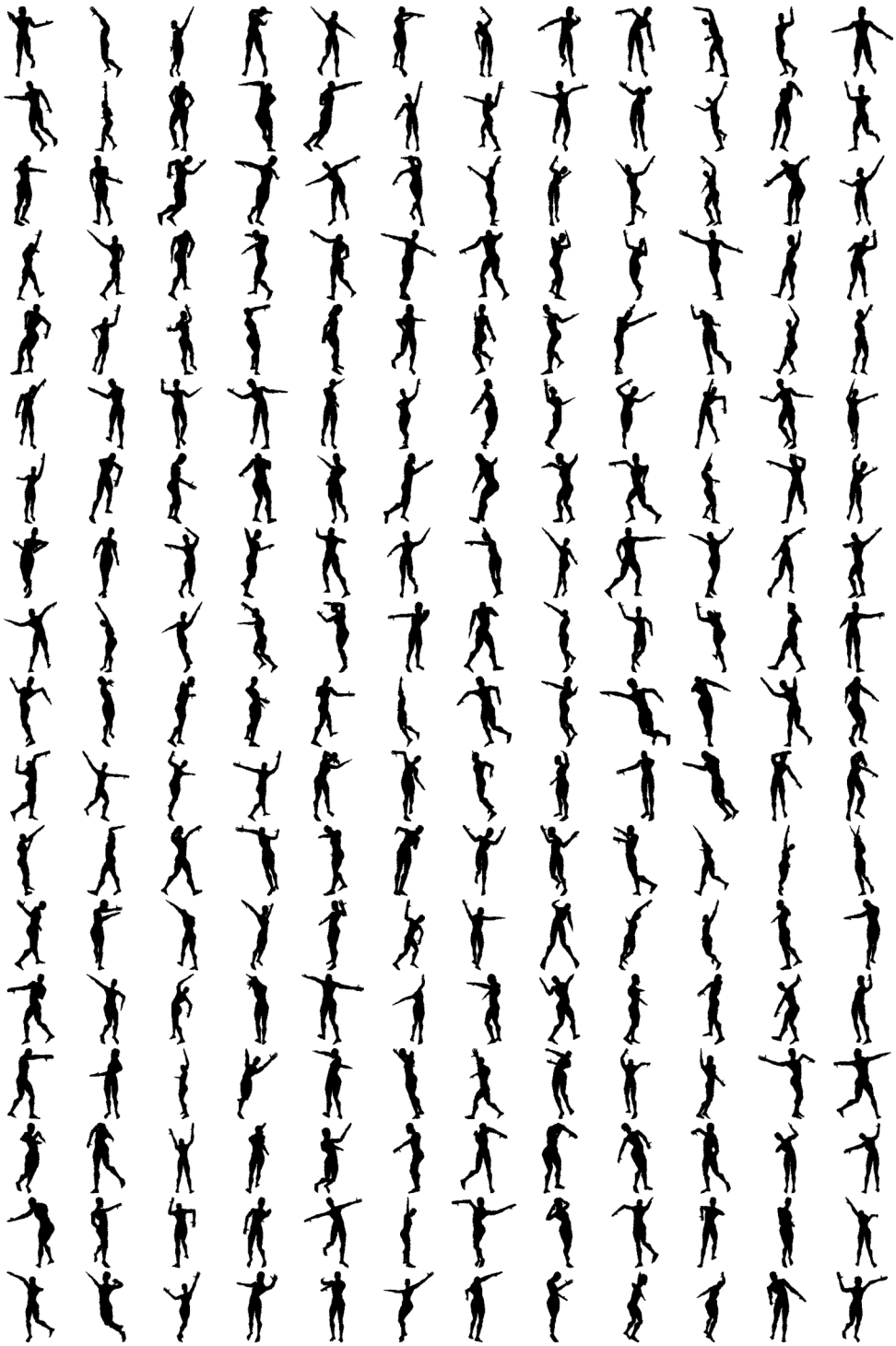


FIG. 8.5 – Échantillon de la base de données $\mathcal{A}$

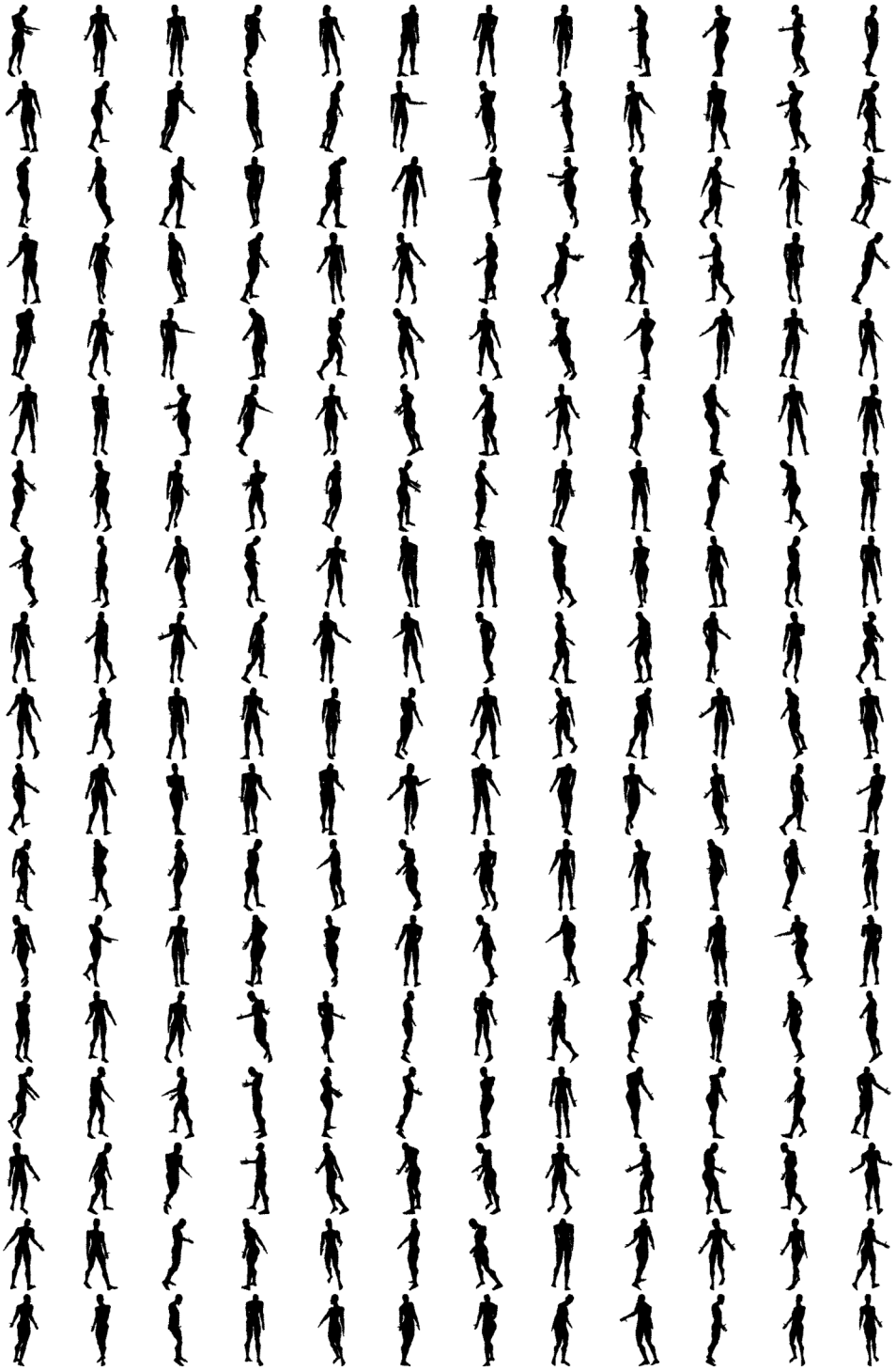


FIG. 8.6 – Échantillon de la base de données $\mathcal{B}$

quement les données concernant la pose à partir de laquelle la silhouette a été générée. L'inversion de cette relation permettrait de retrouver les poses possibles correspondantes à une silhouette donnée. On voit ici un intérêt énorme à l'élimination des poses illicites lors de la génération de la base de données. Les possibilités offertes par cette démarche inverse sont nombreuses. En effet, elle permettrait d'aider à l'analyse des mouvements d'une personne.

Enfin, soulignons l'utilité actuelle du travail dans la recherche.

D'un côté, la création d'une base de données de silhouettes avec des caractéristiques données est devenue faiblement coûteuse en temps humain, bien que le temps machine nécessaire puisse être important lorsque seule les poses licites sont souhaitées. Obtenir 2000 silhouettes était hier un vrai travail titanesque ; aujourd'hui automatisé, il est devenu facilement réalisable. Ainsi pourrons-nous créer des bases de données plus riches et obtenir de meilleurs modèles.

De l'autre côté, l'analyse des performances d'un système de reconnaissance était jusqu'aujourd'hui une évaluation plutôt qualitative, voire même biaisée. En effet, les silhouettes servant à l'apprentissage étaient acquises par le même mécanisme que celles devant être classées. Aujourd'hui les deux procédés sont devenus distincts, rendant ainsi l'analyse des performances des systèmes de reconnaissance rigoureusement quantitative.

Bibliographie

[1] Olivier Barnich. *Détection de personnes dans des séquences vidéo.* Travail de fin d'études de DEA en Sciences Appliquées. Université de Liège, 2006

[2] Olivier Barnich, Sébastien Jodogne, and Marc Van Droogenbroeck. *Robust analysis of silhouettes by morphological size distributions.* Advanced Concepts for Intelligent Vision Systems (ACIVS 2006) of Lecture Notes on Computer Science, 12 pages. Springer Verlag, Antwerpen, Septembre 2006.

[3] Pedro F. Felzenszwalb, Daniel P. Huttenlocher. *Pictorial Structures for Object Recognition*

[4] E.B. Saff et A.B.J.Kuijlaars. *Distributing many points on a sphere.* Mathematical Intelligencer, Springer-Verlag New York, Volume 19, n°1, pp 5-11, 1997

[5] Wonhyung Jung, Hayong Shin et Byoung K. Choi. *Self-intersection Removal in Triangular Mesh Offsetting.* Computer-Aided Design & Applications, Vol. 1, pp 477-484, 2004

[6] David A. Forsyth, Jean Ponce. *Computer Vision : A Modern Approach.* Pearson Prentice Hall, International Edition, 2003

[7] Radu Horaud et Olivier Monga. *Vision par ordinateur, outils fondamentaux* Editions Hermès, deuxième édition, 1995
`http ://perception.inrialpes.fr/people/Horaud/livre-hermes.html`

[8] Wikipédia. *Étalonnage de caméra*
`http ://fr.wikipedia.org/wiki/Calibration_de_cam%C3%A9ra`

[9] William H. Press, Brian P. Flannery, Saul A. Teukolsky, William T. Vetterling. *Numerical Recipes in C : The Art of Scientific Computing* Cambridge University Press, 2ème édition, 1992.
`http ://www.nrbook.com/b/bookcpdf.php`

[10] *POV-Ray : Documentation*
`http ://www.povray.org/documentation/`

[11] *MakeHuman v0.9 : un générateur d'humains en 3D.* Linux Pratique, n°40 pp. 50-56 et n°41 pp. 46-49, 2007

[12] Manuel Bastioni. *The MakeHuman Muscle Engine System : Introduction*
`http ://www.dedalo-3d.com/public/dev/articles/05_10_13_EN_MH_Muscle_Engine_System_Intro.pdf`

[13] Manuel Bastioni. *MakeHuman 0.8 Beta Release : The Humanoid 3D Model*
`http ://makewiki.aleppax.it/pmwiki/pmwiki.php/User/TheHumanoid3DModel`

[14] Developer Notes. *description Of The Quick Pose System*
`http ://makewiki.aleppax.it/pmwiki/pmwiki.php/Developer/DescriptionOfTheQuickPoseSystem`

[15] Naveed Ahmed, Edilson de Aguiar, Christian Theobalt, Marcus Magnor, HansPeter Seidel. *Automatic Generation of Personalized Human Avatars from Multiview Video* ACM Symposium on Virtual Reality Software and Technology (VRST), 2005

Annexe A

Rappels d'algèbre linéaire

A.1 Décomposition en valeurs singulières

Toute matrice A de taille $m \times n$ avec $m \geq n$ peut être décomposée sous la forme $A = UWV^T$ avec :

- U de taille $m \times n$ dont les colonnes sont orthonormées ($U^T U = I$)
- W une matrice diagonale de taille n dont les éléments sont les valeurs singulières de A
- V de taille $n \times n$ telle que $V^T V = VV^T = I$

On peut montrer que les carrés des valeurs singulières de la matrice A égalent les valeurs propres de AA^T et que les colonnes de V sont les vecteurs propres correspondants.

A.2 Moindres carrés et pseudo-inverse

Nous serons amenés à résoudre des systèmes linéaires $Ax = y$. Envisageons la résolution au sens des moindres carrés. On cherche à minimiser la fonction d'erreur $E = \mid Ax - y \mid^2 = (Ax - y)^T (Ax - y)$. On souhaite avoir :

$$0 = \frac{\partial E}{\partial x_i} = 2\frac{\partial (Ax - y)}{\partial x_i}^T (Ax - y) = 2(Ae_i)^T (Ax - y) = 2a_i^T (Ax - y)$$

où a_i est la ième colonne de A. Cette relation est valable pour $i = 1, \ldots q$. Regroupant ces q équations, on obtient :

$$A^T (Ax - y) = 0 \Leftrightarrow A^T Ax = A^T y \Leftrightarrow x = (A^T A)^{-1} A^T y$$

La matrice $(A^T A)^{-1} A^T$ est appelée la pseudo-inverse de U. La décomposition en valeurs singulières permet de calculer cette matrice de manière numériquement plus stable. Si $A = UWV^T$, on a :

$$(A^TA)^{-1}A^T = (VW^TU^TUWV^T)^{-1}VW^TU^T = (VW^2V^T)^{-1}VWU^T$$
$$= (V^{-T}W^{-2}V^{-1})VWU^T = VW^{-1}U^T$$

A.3 Les équations homogènes

Dans le cas des équations homogènes $Ax = 0$, la solution $x = 0$ est triviale. Minimiser la fonction d'erreur présentée ci-dessus n'a de sens que si on impose une contrainte sur x. On peut montrer que sous la contrainte $|x| = 1$, la solution du problème est donnée par le vecteur propre associé à la valeur propre minimale de A^TA. Une fois encore, la décomposition en valeurs singulières permet de calculer la solution de manière numériquement plus stable. En effet, on peut montrer que les carrés des valeurs singulières de la matrice A égalent les valeurs propres de A^TA et que les colonnes de V sont les vecteurs propres correspondants. La solution est donc simplement le vecteur colonne de V associé à la valeur singulière minimale.

Annexe B

Solution de l'équation 3.36

Proposition. Si, pour tout $n \in Z \cap [1, m]$,

$$z_n = z_{n-1} - \frac{(z_{n-1} + \Delta z)^2}{(z_{n-1} + \Delta z) + K}$$

Alors, pour tout $n \in Z \cap [0, m]$,

$$z_n = z_0 - \frac{n(z_0 + \Delta z)^2}{n(z_0 + \Delta z) + K}$$

Démonstration. On procède par induction. La proposition est trivialement vérifiée pour $n = 0$. Pour $1 \leq n \leq m$,

$$\begin{aligned}
z_n &= z_{n-1} - \frac{(z_{n-1} + \Delta z)^2}{z_{n-1} + \Delta z + K} \\
&= z_0 - \frac{(n-1)(z_0 + \Delta z)^2}{(n-1)(z_0 + \Delta z) + K} - \frac{\left(\frac{K(z0+\Delta z)}{(n-1)(z_0+\Delta z)+K}\right)^2}{\frac{K(z0+\Delta z)}{(n-1)(z_0+\Delta z)+K} + K} \\
&= z_0 - \frac{(n-1)(z_0 + \Delta z)^2}{(n-1)(z_0 + \Delta z) + K} - \frac{1}{K}\frac{1}{(n-1)(z_0 + \Delta z) + K}\frac{K^2(z0 + \Delta z)^2}{n(z_0 + \Delta z) + K} \\
&= z_0 - \frac{(z_0 + \Delta z)^2}{(n-1)(z_0 + \Delta z) + K}\left((n-1) + \frac{K}{n(z_0 + \Delta z) + K}\right) \\
&= z_0 - \frac{(z_0 + \Delta z)^2}{(n-1)(z_0 + \Delta z) + K}\left(\frac{(n-1)n(z_0 + \Delta z) + nK}{n(z_0 + \Delta z) + K}\right) \\
&= z_0 - \frac{n(z_0 + \Delta z)^2}{n(z_0 + \Delta z) + K}
\end{aligned}$$

□

Annexe C

Solution de l'équation 6.11

Proposition. La valeur de k qui rend unitaire la déterminant

$$\begin{vmatrix} x_1 - k\ a & x_2 - k\ a & x_3 - k\ a \\ y_1 - k\ b & y_2 - k\ b & y_3 - k\ b \\ z_1 - k\ c & z_2 - k\ c & z_3 - k\ c \end{vmatrix} \tag{C.1}$$

lorsque

$$\overrightarrow{\begin{pmatrix} a \\ b \\ c \end{pmatrix}} = \overrightarrow{\begin{pmatrix} x_2 - x_1 \\ y_2 - y_1 \\ z_2 - z_1 \end{pmatrix}} \wedge \overrightarrow{\begin{pmatrix} x_3 - x_1 \\ y_3 - y_1 \\ z_3 - z_1 \end{pmatrix}} \neq \overrightarrow{0} \tag{C.2}$$

vaut

$$k = \frac{D - 1}{a^2 + b^2 + c^2} \quad \text{avec} \quad D = \begin{vmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ z_1 & z_2 & z_3 \end{vmatrix} \tag{C.3}$$

Démonstration. On souhaite avoir

$$1 = \begin{vmatrix} x_1 - k\ a & x_2 - k\ a & x_3 - k\ a \\ y_1 - k\ b & y_2 - k\ b & y_3 - k\ b \\ z_1 - k\ c & z_2 - k\ c & z_3 - k\ c \end{vmatrix}$$

En développant le déterminant, cette équation peut se réécrire sous la forme

$$\begin{aligned} \Leftrightarrow 1 = \quad & (x_1 - k\ a)(y_2 - k\ b)(z_3 - k\ c) - (x_1 - k\ a)(y_3 - k\ b)(z_2 - k\ c) \\ + \quad & (x_2 - k\ a)(y_3 - k\ b)(z_1 - k\ c) - (x_2 - k\ a)(y_1 - k\ b)(z_3 - k\ c) \\ + \quad & (x_3 - k\ a)(y_1 - k\ b)(z_2 - k\ c) - (x_3 - k\ a)(y_2 - k\ b)(z_1 - k\ c) \end{aligned}$$

En distribuant les différents termes et en remarquant que les termes en ab, bc, ca et abc s'annulent deux par deux, on obtient

$$\begin{aligned} \Leftrightarrow 1 = D \quad & -k\ a\ y_2\ z_3 - k\ b\ x_1\ z_3 - k\ c\ x_1\ y_2 + k\ a\ y_3\ z_2 + k\ b\ x_1\ z_2 + k\ c\ x_1\ y_3 \\ & -k\ a\ y_3\ z_1 - k\ b\ x_2\ z_1 - k\ c\ x_2\ y_3 + k\ a\ y_1\ z_3 + k\ b\ x_2\ z_3 + k\ c\ x_2\ y_1 \\ & -k\ a\ y_1\ z_2 - k\ b\ x_3\ z_2 - k\ c\ x_3\ y_1 + k\ a\ y_2\ z_1 + k\ b\ x_3\ z_1 + k\ c\ x_3\ y_2 \end{aligned}$$

Divisons les deux membres par k,

$$\begin{aligned}\Leftrightarrow \frac{1-D}{k} = \quad & -a\ y_2\ z_3 - b\ x_1\ z_3 - c\ x_1\ y_2 + a\ y_3\ z_2 + b\ x_1\ z_2 + c\ x_1\ y_3 \\ & -a\ y_3\ z_1 - b\ x_2\ z_1 - c\ x_2\ y_3 + a\ y_1\ z_3 + b\ x_2\ z_3 + c\ x_2\ y_1 \\ & -a\ y_1\ z_2 - b\ x_3\ z_2 - c\ x_3\ y_1 + a\ y_2\ z_1 + b\ x_3\ z_1 + c\ x_3\ y_2\end{aligned}$$

et regroupons les termes en a, ceux en b et ceux en c,

$$\begin{aligned}\Leftrightarrow \frac{1-D}{k} = \quad & a(y_3\ z_2 + y_1\ z_3 + y_2\ z_1 - y_2\ z_3 - y_3\ z_1 - y_1\ z_2) \\ + \quad & b(x_1\ z_2 + x_2\ z_3 + x_3\ z_1 - x_1\ z_3 - x_2\ z_1 - x_3\ z_2) \\ + \quad & c(x_1\ y_3 + x_2\ y_1 + x_3\ y_2 - x_1\ y_2 - x_2\ y_3 - x_3\ y_1)\end{aligned}$$

$$\begin{aligned}\Leftrightarrow \frac{1-D}{k} = \quad & -a\,[(y_2-y_1)(z_3-z_1) - (z_2-z_1)(y_3-y_1)] \\ & -b\,[(z_2-z_1)(x_3-x_1) - (x_2-x_1)(z_3-z_1)] \\ & -c\,[(x_2-x_1)(y_3-y_1) - (y_2-y_1)(x_3-x_1)]\end{aligned}$$

Enfin, remarquons que

$$\begin{cases} a = (y_2-y_1)(z_3-z_1) - (z_2-z_1)(y_3-y_1) \\ b = (z_2-z_1)(x_3-x_1) - (x_2-x_1)(z_3-z_1) \\ c = (x_2-x_1)(y_3-y_1) - (y_2-y_1)(x_3-x_1) \end{cases}$$

ce qui permet de réécrire l'équation sous la forme

$$\Leftrightarrow \frac{1-D}{k} = -a^2 - b^2 - c^2$$

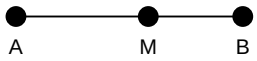
$$\Leftrightarrow k = \frac{D-1}{a^2+b^2+c^2}$$

□

Annexe D

Segments et triangles

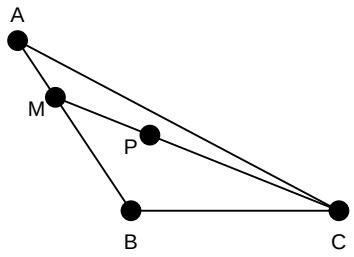
Les équations d'un segment de droite et d'un triangle sont obtenues en utilisant les coordonnées barycentriques. Commençons par établir la première.



Le point M appartiendra au segment si et seulement si :

$$\begin{aligned} & \overrightarrow{AM} = k\overrightarrow{AB} \quad \text{avec} \quad k \in [0,1] \\ \Leftrightarrow \quad & \overrightarrow{M} - \overrightarrow{A} = k(\overrightarrow{B} - \overrightarrow{A}) \\ \Leftrightarrow \quad & \overrightarrow{M} = (1-k)\overrightarrow{A} + k\overrightarrow{B} \end{aligned} \tag{D.1}$$

Utilisons ce résultat pour établir l'équation d'un triangle.



Pour tout point P du triangle ABC, il doit exister un point M appartenant au segment AB tel que P appartienne au segment MC :

$$\begin{aligned} & \begin{cases} \overrightarrow{M} = (1-\alpha)\overrightarrow{A} + \alpha\overrightarrow{B} & \text{avec} \quad \alpha \in [0,1] \\ \overrightarrow{P} = (1-\beta)\overrightarrow{M} + \beta\overrightarrow{C} & \text{avec} \quad \beta \in [0,1] \end{cases} \\ \Leftrightarrow \quad & \overrightarrow{P} = (1-\beta)(1-\alpha)\overrightarrow{A} + (1-\beta)\alpha\overrightarrow{B} + \beta\overrightarrow{C} \\ \Leftrightarrow \quad & \overrightarrow{P} = a\overrightarrow{A} + b\overrightarrow{B} + c\overrightarrow{C} \quad \text{avec} \quad \begin{cases} a \geq 0, b \geq 0, c \geq 0 \\ a+b+c = 1 \end{cases} \end{aligned} \tag{D.2}$$