3m11.3213.4

Université de Montréal

Rendu non-photoréaliste de chevelures

par

Martin Côté

Département d'informatique et de recherche opérationnelle

Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures
en vue de l'obtention du grade de
Maître ès sciences (M.Sc.)
en informatique

Mai 2004

© Martin Côté, 2004

QA
76
U54
2004
v.013

Université de Montréal

Direction des bibliothèques

AVIS

L'auteur a autorisé l'Université de Montréal à reproduire et diffuser, en totalité ou en partie, par quelque moyen que ce soit et sur quelque support que ce soit, et exclusivement à des fins non lucratives d'enseignement et de recherche, des copies de ce mémoire ou de cette thèse.

L'auteur et les coauteurs le cas échéant conservent la propriété du droit d'auteur et des droits moraux qui protègent ce document. Ni la thèse ou le mémoire, ni des extraits substantiels de ce document, ne doivent être imprimés ou autrement reproduits sans l'autorisation de l'auteur.

Afin de se conformer à la Loi canadienne sur la protection des renseignements personnels, quelques formulaires secondaires, coordonnées ou signatures intégrées au texte ont pu être enlevés de ce document. Bien que cela ait pu affecter la pagination, il n'y a aucun contenu manquant.

NOTICE

The author of this thesis or dissertation has granted a nonexclusive license allowing Université de Montréal to reproduce and publish the document, in part or in whole, and in any format, solely for noncommercial educational and research purposes.

The author and co-authors if applicable retain copyright ownership and moral rights in this document. Neither the whole thesis or dissertation, nor substantial extracts from it, may be printed or otherwise reproduced without the author's permission.

In compliance with the Canadian Privacy Act some supporting forms, contact information or signatures may have been removed from the document. While this may affect the document page count, it does not represent any loss of content from the document.

Université de Montréal
Faculté des études supérieures

Ce mémoire intitulé

Rendu non-photoréaliste de chevelures

présenté par

Martin Côté

a été évalué par un jury composé des personnes suivantes:

Pierre Poulin, président du jury
Victor Ostromoukhov, directeur de recherche
Philippe Langlais, membre du jury

Mémoire accepté le
21 juin 2004

Sommaire

Les chevelures sont depuis plusieurs années un sujet très chaud en infographie. Les défis que les chevelures procurent sont nombreux : rendu difficile dû à la fine géométrie des cheveux et aux nombreux phénomènes physiques s'y rattachant, animation difficile due aux mouvements complexes, etc. Plusieurs chercheurs ont relevé le défi avec succès en produisant des chevelures qui se confondent avec la réalité en utilisant des modèles physiques généralement fort complexes.

Néanmoins, très peu d'ouvrages ont été réalisés dans le rendu non-réaliste (ou non-photoréaliste) des chevelures, notamment en ce qui concerne les dessins animés. Pourtant, les chevelures de style *cartoon* sont devenues au fil du temps étonnamment complexes et sont toujours produites manuellement dans l'industrie du dessin animé. Ce présent travail a pour but de combler cette lacune.

L'approche utilisée est fort simple : nous allons étudier les chevelures produites par des artistes experts du domaine et tenter d'en extraire un style puissant et couramment utilisé. Nous allons ensuite trouver un algorithme permettant de reproduire le ou les styles voulus par un modèle simple et efficace. Nous aurons toujours la ferme intention de simplifier le travail de l'artiste au minimum lors de la production de chevelures. Nous allons donc proposer une interface simple inspirée des techniques traditionnelles de dessins pour produire les chevelures. Finalement, nous allons démontrer que le modèle choisi, ainsi que l'interface proposée seront facilement intégrables dans des systèmes d'animation déjà existants.

Mot-clés : chevelures, rendu non-photoréaliste, animation.

Abstract

Hair has been a very popular topic in computer graphics since many years. The challenges that follow from it are numerous : difficult rendering caused by the fine geometry of hair and by many physical interactions, difficult animation caused by subtle movements, etc. Many researchers have proposed interesting solutions to these problems by creating realistic hair based on physical models, generally highly complex.

However, very few authors have attacked the problem of non-realistic (or non-photorealistic) rendering of hair, particularly in regards to cartoon animation. Today's cartoon renderings are highly complex and always achieved manually in the industry. The purpose of this work is to fill this gap.

The approach we will use is conceptually simple : we will study different hair models created by professional artists in order to find a common style that we could use. Then, we will develop an algorithm to reproduce the style with a simple procedural model. Our ultimate goal will be to simplify the artists' work during the hair creation process. We will therefore propose a simple interface inspired by traditional movements used by artists to create their work. Finally, we will show that our technique is suitable to fit in already existing animation software.

Keywords : hair, non-photorealistic rendering, animation.

Table des matières

Liste des tableaux

Table des figures

Liste des algorithmes

À Léon et Marième

Ainsi qu'en mémoire de ma mère, Bibiane

Remerciements

Trop rapides. Trop rapides furent ces deux années que je qualifierais des plus riches et des plus stimulantes de mon existence. J'ai la nette impression d'avoir plus appris en deux ans qu'il est physiquement possible, non seulement sur l'infographie, mais également sur la recherche en général.

Plusieurs personnes ont enrichi mon existence durant ma maîtrise. Le plus évident est mon directeur de recherche, Victor Ostromoukhov, qui m'a appris énormément de choses concernant la recherche et l'infographie. Le plus étonnant avec Victor est sa capacité de transmettre des connaissances et de partager son expérience gratuitement et en toute simplicité. Il va également de soit de remercier Pierre Poulin. Ses conseils pratiques et son humour sont toujours appréciés. Il y a également d'autres membres du laboratoire qu'il est essentiel de nommer. Ma maîtrise n'aurait pas été pareille sans la frénésie de musique avec Jean-François, ainsi que nos "soupers santés". Également, j'ai toujours eu des discussions intéressantes avec Emric et Martin (il s'agit ici d'une seule et même entité). Mathieu a beaucoup enrichi (et surtout critiqué) mon *C++*, en plus d'avoir rempli mon *iPod* de musique rythmée, il mérite donc mes remerciements. Il y a aussi d'autres membres avec qui j'ai eu des discussions constructives : Philippe, François (merci pour les animés japonais), l'autre Jean-François, Yannick et Simon pour en nommer quelques-uns.

À l'extérieur du laboratoire, il est essentiel de remercier mon père, Léon, ma soeur, Mariève, ainsi que le reste de ma famille pour leurs encouragements continus. Ils sont tous à la source de ma réussite. Je tiens également à remercier de tout coeur ma copine

et conjointe, Émilie, pour sa patience et son écoute durant ces deux années. Tout aurait été différent sans elle.

Sur une note plus technique, je remercie la compagnie *Apple* pour avoir réalisé la machine et le système d'exploitation de rêve pour un étudiant en maîtrise, et je remercie l'auteur du langage *C++* et les auteurs des logiciels *Emacs* et LaTeX pour avoir rendu possible la réalisation de ce travail.

Conventions d'écriture

Bien que ce travail soit écrit en français, plusieurs termes techniques anglophones sont utilisés soit pour simplifier la lecture, ou encore parce qu'aucun terme francophone équivalent n'est satisfaisant. Les termes anglophones seront toujours en *italique*.

Dans les formules, tous les vecteurs seront implicitement de dimension 3 et seront des vecteurs colonne, à moins d'indication contraire. Un vecteur v sera noté $\vec{v}$ et sera toujours explicitement transposé ($\vec{v}^T$) lorsque nécessaire. Une matrice sera représentée par une lettre majuscule, par exemple M, et les scalaires par une lettre minuscule, par exemple s. Les éléments des vecteurs ou matrices peuvent être spécifiés avec des indices, par exemple, $\vec{v}_x$ ou M_{ij}. Le produit scalaire entre deux vecteurs $\vec{v}$ et $\vec{w}$ est représenté par $\vec{v} \cdot \vec{w}$ et le produit vectoriel par $\vec{v} \times \vec{w}$. Le produit matriciel entre deux matrices M et N est représenté par $M \cdot N$.

Droits d'auteur

Plusieurs oeuvres artistiques sont illustrées dans cet ouvrage. Les noms des auteurs de ces oeuvres sont clairement indiqués sur les figures (avec références, selon le cas). Également, plusieurs images sont tirées de travaux de publication ACM. Les références aux travaux originaux sont alors visiblement indiquées sur les figures en question. Voici leur note de droits d'auteur :

Copyright© 2004 by the Association for Computing Machinery, Inc. Permission to make digital or hard copies of part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page or intial screen of the document. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Publications Dept., ACM Inc., fax +1 (212) 869-0481, or permissions@acm.org.

Chapitre 1

Introduction

In the beginning the Universe was created. This has made a lot of people very angry and been widely regarded as a bad move.

Douglas Adams,
The Restaurant at the End of the Universe

La synthèse d'images réalistes, ou encore photoréalistes, est généralement l'objectif ultime à atteindre en informatique graphique. Tenter de reproduire les phénomènes naturels par ordinateur peut offrir au développeur un sentiment de contrôle qui se compare au Créateur lui-même. Bien que cette affirmation semble un peu exagérée, elle reste néanmoins véridique et totalement justifiée. En essayant de comprendre ces phénomènes physiques complexes et en réussissant à les reproduire fidèlement, le chercheur peut avoir l'impression de reproduire la réalité.

Cependant, cette recherche absolue du réalisme, ainsi que l'obsession qu'ont eu les scientifiques d'hier et d'aujourd'hui à jouer au Créateur, ont eu pour conséquence une divergence des priorités de recherche et de développement en infographie. Depuis déjà plusieurs décennies, les étudiants et chercheurs s'acharnent à reproduire une multitude

(a) (b) (c)

FIG. 1.1 – Exemples de dessins style *cartoon*. Images réalisées par (a) Brian Stelfreeze, (b) Michael Turner, Joe Weems V and Jonathan D. Smith, (c) Keu Cha, D-Tron et Steve Firchow.

de phénomènes physiques complexes dans l'espoir d'éblouir la population entière avec leur réalisme alors que ces mêmes chercheurs n'avaient qu'à sortir la tête dehors pour admirer ces phénomènes. Nous pouvons alors nous poser la question suivante : étant donnée une image photoréaliste, quelles informations supplémentaires nous apporte-t-elle par rapport à la réalité ?

Depuis quelques années, une nouvelle vague de recherches a pris naissance : le rendu non-photoréaliste (NPR). Il s'agit non seulement d'une nouvelle mode, mais également d'une réorientation majeure des voies de recherche en infographie. Le rendu non-photoréaliste, contrairement au rendu photoréaliste, tente de reproduire l'information visuelle sous une forme plus facile à digérer pour un être humain. Pour y arriver, les chercheurs tentent de trouver une représentation plus simple et plus compréhensible des données à afficher.

Il peut sembler étrange, voire même inapproprié de nommer un domaine d'étude par ce qu'il n'est pas. Stanislaw Ulam aurait d'ailleurs déjà fait la remarque suivante : *"The study of non-linear physics is like the study of non-elephant biology."* Nous verrons que ce type de rendu englobe effectivement un large éventail de styles différents.

Ceci étant dit, nous pouvons désormais nous intéresser à un type particulier de rendu non-photoréaliste (puisqu'il en existe évidemment plusieurs). Les dessins animés *(cartoons)* sont un de ces types. Ils sont également très répandus et connus de tous. Essentiellement, les dessins animés tentent de reproduire la réalité d'une manière sim-

FIG. 1.2 – Les chevelures véhiculent les émotions. Images tirées du film *Princess Mononoke.*

plifiée. Le style a cependant évolué et touche maintenant plusieurs artistes accomplis qui tentent, à leur manière, de faire passer différentes émotions à travers les couleurs, le style, et les personnages. Les éléments que possèdent les artistes pour véhiculer ces informations sont évidemment bien limités à travers un dessin, c'est pourquoi ils devraient faire usage de tous les moyens nécessaires pour y arriver. La Fig. 1.1 montre différents exemples de dessins *cartoons* ainsi que leur impact visuel.

Parmi les différents éléments que les artistes possèdent, les chevelures s'avèrent particulièrement puissantes. Leur fine géométrie ainsi que leur visuel frappant donnent de la vie aux dessins. Les chevelures sont également utiles pour faire passer différentes émotions aux personnages. Ce fait est particulièrement remarquable dans le film *Princess Mononoke* dans lequel la chevelure du héros *Ashitaka* se déforme relativement à ses sentiments (Fig. 1.2).

Tenter de reproduire les chevelures créées par les artistes en dessin animé est un sujet de rendu non-photoréaliste parmi tant d'autres. Pourquoi s'intéresser particulièrement au rendu *cartoon* des cheveux ? Pourquoi ne pas s'intéresser au rendu au fusain ou à la peinture qui comportent souvent un niveau artistique plus élevé ? Il y a plusieurs réponses à ces questions :

- *Les dessins animés sont mondialement répandus.* Tout le monde a déjà lu une bande dessinée, ou encore regardé un dessin animé à la télévision. S'intéresser au rendu *cartoon*, c'est s'intéresser à tout un monde.
- *Le travail effectué dans l'industrie est long et laborieux.* Les technologies utilisées en dessin animé sont généralement démodées et comportent une grande quantité de manipulations encore exécutées à la main. Diminuer la charge de travail manuel permettrait d'augmenter considérablement la productivité dans l'industrie.

(a) (b) (c) (d)

FIG. 1.3 – Exemples de chevelures complexes. Images réalisées par (a) et (b) J.Scott Campbell et Alex Garner, (c) et (d) Ed Benes et Vince Russell.

– *Les dessins animés sont facilement automatisables.* L'industrie du dessin animé comporte de nombreuses facettes répétitives à pratiquement tous les stades de la production (Section 3.1). Une automatisation totale ou partielle pourrait amener d'importantes économies.

Nous connaissons l'importance des cheveux dans un dessin, et l'importance des dessins dans l'industrie. Cependant, quels défis procurent le rendu *cartoon* des chevelures ? Pourquoi s'attaquer aux chevelures plutôt qu'à n'importe quelle autre partie d'un dessin ? Bien sûr, nous connaissons l'impact des chevelures dans les dessins animés, mais comportent-elles un problème majeur dans l'industrie ? La réponse à cette question se trouve à la Fig. 1.3, dans laquelle toute la complexité des chevelures est explicitée. Puisque les cheveux sont si fins, ils sont extrêmement difficiles à dessiner pour les artistes, et encore plus difficiles à animer, les mouvements des cheveux étant subtils et imprévisibles. Nous pouvons donc affirmer hors de tout doute que les chevelures procurent un défi intéressant en infographie, et plus particulièrement les chevelures non-photoréalistes.

Il est étonnant de constater que, jusqu'à ce jour, très peu de recherche a été effectuée sur les chevelures non-photoréalistes. Un des seuls travaux sur le sujet a été produit par Kowalski *et al.* [KMN+99] qui utilisaient des textures procédurales nommées *graftals* qui permettent le rendu non-photoréaliste de différentes matières, notamment la fourrure, qui changeait d'apparence selon l'angle de vue. Cependant, leur approche n'est pas particulièrement intéressante pour produire des chevelures (Chapitre 2).

Même si peu de travail a été effectué en chevelures non-photoréalistes, il n'en est pas de même pour les chevelures photoréalistes où une grande quantité de littérature est disponible sur leur rendu et leur animation. Les auteurs de ces travaux étaient généralement inspirés par des modèles physiques complexes permettant de reproduire l'aspect anisotropique et les mouvements subtils des chevelures. Néanmoins, ces auteurs étaient souvent influencés par leur éducation scientifique et avaient du mal à trouver des solutions intéressantes au problème de rendu non-photoréaliste de chevelures. Le Chapitre 2 fait une revue de ces différentes solutions.

L'approche utilisée dans ce travail est totalement différente. Au Chapitre 3, nous étudierons en détail les différentes techniques utilisées par les artistes en dessin animé, ainsi que l'importance des chevelures dans ce type de rendu. Nous tenterons également de découvrir des méthodes procédurales permettant de reproduire fidèlement les techniques à partir d'une généralisation des résultats. Nous verrons que certaines de ces techniques reviennent régulièrement d'une oeuvre à une autre et d'un artiste à un autre. Parmi ces techniques, nous verrons que le *feathering* est très souvent utilisé et procure des résultats d'une qualité hautement supérieure. De plus, nous verrons que cette technique est en fait un cas particulier de rendu en demi-ton et qu'il est possible de trouver des méthodes procédurales et interactives pour la reproduire.

Cependant, trouver des méthodes procédurales pour produire des chevelures n'est pas suffisant pour convaincre un artiste d'utiliser un système informatisé. Le Chapitre 4 propose certaines solutions afin de simplifier au maximum l'apprentissage d'un système pour un artiste. Essentiellement, la procédure utilisée tente de produire des mèches de cheveux à partir du même mouvement caractéristique que les artistes exécutent en pratique. Nous verrons que cette méthode permet de créer rapidement des mèches cohérentes par rapport aux techniques traditionnelles.

Les mèches positionnées sur un modèle doivent être rendues d'une manière ou d'une autre. La technique utilisée pour faire le rendu de ces mèches déterminera le style du dessin final. Dans le Chapitre 5, une méthode simple pour imiter le *feathering*, décrit précédemment, est explicitée. Le rendu sera temps-réel avec OpenGL [WNDS99].

Un autre aspect primordial de l'industrie du dessin animé est l'animation. Encore aujourd'hui, trop de travail est effectué manuellement par les artistes. Les différentes images composantes des animations sont générées manuellement par des *in-betweeners*. Ce travail est long, épuisant et peu valorisant. De plus, cette répétition manuelle est très encline aux erreurs de production. Le Chapitre 6 démontre que les solutions choisies précédemment s'adaptent facilement à différents modèles d'animation, simplifiant ainsi le travail manuel laborieux.

Le Chapitre 7 décrira en détail les avantages et inconvénients de chacune des techniques proposées au cours des chapitres et propose également des solutions alternatives à certains problèmes. Deux annexes ont également été ajoutées à ce travail : l'Annexe A décrit les détails techniques de l'implémentation du système et l'Annexe B explicite les temps de calcul requis pour faire le rendu.

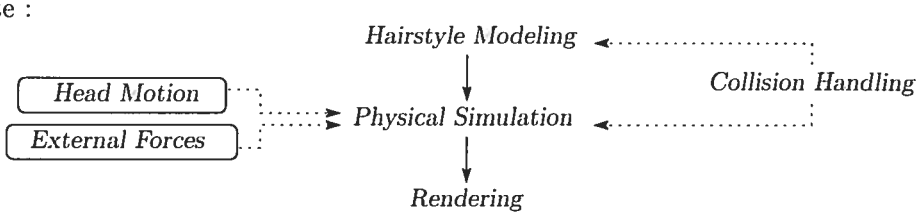
Chapitre 2

Travaux antérieurs

I love mathematics, it's the mathematicians
I cannot stand.

Edsger Dijkstra

Plusieurs auteurs ont étudié les chevelures. La plupart d'entre eux ont tenté, avec succès, de reproduire l'apparence et le mouvement des cheveux à partir de modèles physiques. Les différentes propriétés subtiles des chevelures font de ces modèles de véritables défis dans plusieurs branches de l'infographie moderne. Tous ces auteurs ont cependant un aspect en commun : leur but est de reproduire des chevelures réalistes avec des modèles réalistes. Aucun d'entre eux n'a tenté de reproduire les chevelures sous un aspect non-photoréaliste. Un aperçu des systèmes classiques est démontré dans l'illustration suivante :



Les travaux effectués en chevelures photoréalistes se divisent donc en trois parties distinctes : la modélisation, l'animation et le rendu.

2.1 Chevelures photoréalistes

2.1.1 Modélisation des chevelures

Très peu d'ouvrages avant l'an 2000 traitent de la modélisation des chevelures. Visiblement, les auteurs souhaitent d'abord fournir des rendus de chevelures de qualité avant de simplifier leur modélisation. Le travail de Daldegan *et al.* en 1993 [DTKT93] était donc fort avant-gardiste. Leur approche propose un système complet de modélisation, d'animation et de rendu de chevelures. Un module spécial de leur système nommé *HairStyler* permet une modélisation simplifiée sur des modèles de tête 3D. Ce module place des cheveux individuels sur les polygones formant le cuir chevelu (*scalp*). Chaque polygone possède un certain nombre de paramètres :

- une référence vers une courbe 3D ;
- un identificateur de matériel pour les cheveux ;
- un déplacement (*jitter*) pour la position initiale des cheveux ;
- l'orientation des cheveux ;
- l'échelle (*scaling*) appliquée aux cheveux ;
- la densité, *i.e.* le nombre de cheveux par unité de distance ;
- etc.

La plupart des travaux subséquents en modélisation de chevelures sont inspirés de cette méthode.

La contribution suivante majeure en modélisation fut publiée en 2001 par Chang [Yu01]. Sa solution tente de simplifier la création des cheveux frisés par quelques approches totalement différentes :

- une approche pour simuler le flot des cheveux (localement et globalement) par des champs vectoriels ;
- une fonction générique permettant de simuler les «frisottis» naturels des cheveux ;
- une méthode pour regrouper les cheveux.

Essentiellement, l'auteur utilise l'analogie entre les cheveux et les champs vectoriels pour faciliter la création des chevelures. L'usager peut superposer différents champs vectoriels (créés manuellement) pour donner la forme générale des cheveux. Ensuite, les frisottis

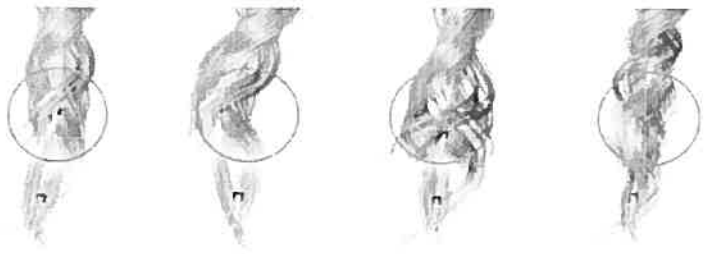
sont ajoutés avec une fonction de décalage (*offset*) paramétrique. Finalement, les cheveux sont regroupés pour former des agrégats à partir de quelques cheveux représentatifs et d'une région sélectionnée.

Xu et Yang [XY01], toujours en 2001, proposent un système complet pour faire le *design* et la modélisation de chevelures basé sur un modèle de agrégats, nommé *V-HairStudio*. Leur système est en fait un outil permettant la modélisation de cylindres généralisés (*generalized cylinders*) qui sont définis de la manière suivante :

$$\overline{x}(s,a) = \overline{y}(s) + R(s,a),$$

où $\overline{y}(s)$ est une courbe générale paramétrisée sur la longueur s et $R(s,a)$ est une fonction de rayon centrée en $\overline{y}(s)$ et ayant un angle a (permettant d'effectuer un *twist* sur le cylindre). Les cheveux seront distribués uniformément dans ce cylindre pour créer un *cluster*. Le système permet donc la création et l'édition de cylindres généralisés autour d'un modèle 3D de tête, permettant la modélisation de chevelures complètes avec un *feedback* interactif.

Kim et Neumann [KN02] ont également développé un système de modélisation et d'édition basé sur les cylindres généralisés, mais ils poussent leur système un peu plus loin en utilisant un modèle multirésolutions. Effectivement, le modèle de Xu et Yang ne permettait pas la modélisation de plusieurs styles de chevelures étant donné l'aspect uniforme des cylindres. Kim et Neumann ont résolu ce problème en créant une hiérarchie de noeuds à l'intérieur des cylindres. De cette manière, l'utilisateur peut raffiner l'aspect visuel des agrégats en hiérarchisant chaque mèche de cheveux, tel que démontré dans l'illustration suivante (tirée de [KN02]) :



Ce travail apporte également des contributions significatives concernant la sélection de branches et autres outils de haut niveau (tel que le *copy/paste*) pour manipuler la hié-

rarchie qui peut s'avérer fort souvent très complexe. D'autre part, les auteurs montrent également quelques solutions pour obtenir des rendus de chevelures hautement réalistes. Ils utilisent les *deep shadow maps* [LV00] pour calculer les ombres à l'intérieur même des chevelures (*self-shadowing*). Également, ils permettent un antialiassage temps-réel en ordonnant les cheveux individuels de derrière à avant et en activant l'antialiassage de lignes d'OpenGL.

Ces quelques articles sont les ouvrages représentatifs en modélisation de chevelures. Comme nous pouvons le constater, les contributions proposées restent préliminaires et ne nous seront pas d'une grande utilité dans le cadre de chevelures non-photoréalistes. Nous allons maintenant changer de direction et regarder le travail effectué concernant l'animation des chevelures.

2.1.2 Animation des cheveux

L'animation des cheveux a vu le jour dans la recherche infographique essentiellement au même moment que la modélisation. Ceci s'explique par la même raison : les chercheurs désiraient tout d'abord produire des rendus de bonne qualité avant de s'attaquer à l'animation.

Le travail de Anjyo *et al.* [iAUK92] en 1992 fut probablement la contribution la plus significative et avant-gardiste en ce qui concerne l'animation des cheveux. Ils proposent un modèle physique très simplifié afin d'obtenir des mouvements réalistes de chevelures, mais avec un temps de calcul fortement réduit. La méthode suggère d'évaluer séquen-

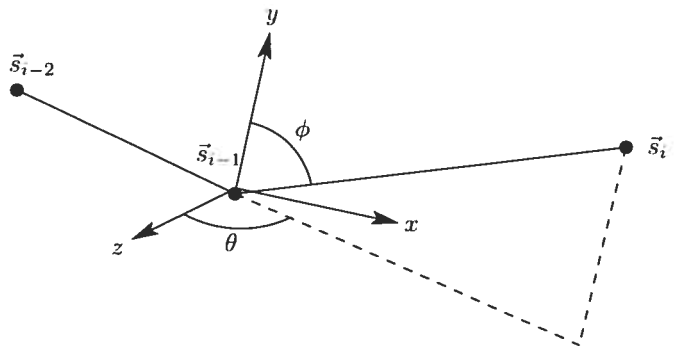


FIG. 2.1 – Représentation utilisée par Anjyo *et al.* [iAUK92].

tiellement la position de chaque segment composant un cheveu. La position du segment courant dépend de la position du segment précédent et le premier segment est immobile (fixé à l'avance sur le modèle de tête).

L'évaluation de la position de $\vec{s_i}$ (où $\vec{s_i}$ est le vecteur déterminé par le point distant du segment $i-1$ vers le point distant du segment i)* se fera par rapport à un système de coordonnées dont $\vec{s}_{i-1}$ est l'origine. La position de $\vec{s_i}$ dans ce système est exprimée par deux angles, ϕ et θ, tel qu'illustré à la Fig. 2.1. Lorsqu'une force est appliquée sur le cheveu, elle est projetée dans les plans $\vec{s_i}y$ et xz du système de coordonnées pour permettre la modification des angles ϕ et θ. Cette modification utilisera les deux derniers pas de l'animation, $\phi_{t_0-\Delta t}, \theta_{t_0-\Delta t}, \phi_{t_0}$ et θ_{t_0}, ainsi que les valeurs projetées de la force appliquée, F_ϕ et F_θ, pour évaluer la position du segment au prochain pas de l'animation, $\phi_{t_0+\Delta t}$ et $\theta_{t_0+\Delta t}$, de la manière suivante :

$$\begin{aligned} \phi_{t_0+\Delta t} &= \Delta t^2 c_i u_i F_\phi + 2\phi_{t_0} - \phi_{t_0-\Delta t} \\ \theta_{t_0+\Delta t} &= \Delta t^2 c_i v_i F_\theta + 2\theta_{t_0} - \theta_{t_0-\Delta t} \end{aligned}$$

où u_i et v_i sont les demi-longueurs de la projection de $\vec{s_i}$ sur les plans $\vec{s_i}y$ et xz, et où c_i est une constante permettant de contrôler le mouvement des cheveux. Ce même ouvrage propose également des solutions pour détecter les collisions entre les cheveux et le corps humain, mais pas les collisions entre les cheveux, ceci demandant beaucoup trop de calculs, les cheveux étant traités individuellement. La plupart des travaux qui suivront en animation [KiAT93, DTKT93] seront inspirés par cette contribution.

La prochaine contribution significative fut proposée en 2000 par Koh et Huang [KH00]. Les auteurs proposent d'utiliser des surfaces 2D texturées (*strips*) afin de simplifier la modélisation et l'animation des chevelures. Les textures utilisées sont combinées avec des cartes de transparence (*alpha maps*) pour donner un aspect plus réaliste aux cheveux. Les résultats obtenus d'une telle approche ne seront pas particulièrement impressionnants (Fig. 2.2), mais le rendu pourra se faire en temps-réel sans problème. La Fig. 2.2(a) montre la carte de transparence ainsi que la texture utilisée pour les *strips*.

*Le point distant d'un segment est le point le plus éloigné de la racine du cheveu sur le segment.

(a) (b) (c)

FIG. 2.2 – Utilisation de *strips* texturées pour représenter les chevelures. Images tirées de Koh et Huang [KH00].

La Fig. 2.2(b) montre les *strips* non texturés et la Fig. 2.2(c) montre les *strips* texturés.

Néanmoins, cette contribution n'apporte rien en particulier en ce qui a trait à l'animation. Les auteurs démontrent uniquement que leur approche peut être utilisée pour animer et en font la démonstration avec une animation par *keyframes* interpolés. Leur prochaine contribution [KH01] proposera un modèle physique simplifié pour animer les chevelures avec la représentation par *strips*. Leur approche est fortement inspirée du travail de Anjyo *et al.* [iAUK92], mais propose d'utiliser les points de contrôle des surfaces 2D plutôt que les segments formant les cheveux individuels. Leur procédure est explicitée à l'Algorithme 2.1.

```
foreach timestep dt do
    foreach hair strip do
        foreach control point P_i, from top to bottom do
            Compute force F_1 due to head movement and gravity;
            Compute force F_2 due to wind;
            Compute force F_3 due to springs;
            Compute F_total = F_1 + F_2 + F_3;
            Break F_total into F_φ and F_θ;
            Compute torques M_φ and M_θ;
            Compute new position;
            Collision detection with ellipsoid (head approximation);
            if collision occured then Compute collision response;
        end
    end
end
```

Algorithme 2.1: Algorithme de Koh et Huang [KH01] pour simuler le mouvement des cheveux.

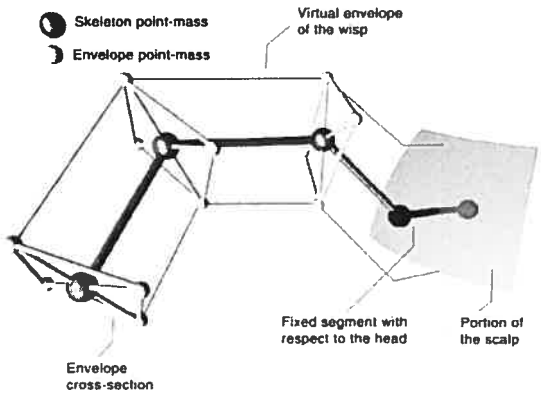


FIG. 2.3 – Volume pour simuler les interactions entre les cheveux. Image tirée de Plante *et al.* [PCP02].

Plante *et al.* [PCP01, Pla99] proposent en 2001 une nouvelle approche pour traiter l'interaction complexe entre les cheveux sur un modèle. Ils suggèrent de regrouper les cheveux ensemble avec des volumes, dans lesquels les interactions visqueuses seront simulées. Dans un autre ouvrage [PCP02], ils proposent une méthode pour animer les cheveux longs tout en traitant les interactions décrites précédemment. Pour y arriver, ils suggèrent d'utiliser un système de couches (*layers*) pour contrôler les mouvements à plusieurs niveaux. Ils définissent également le concept de *wisp* avec la structure suivante :

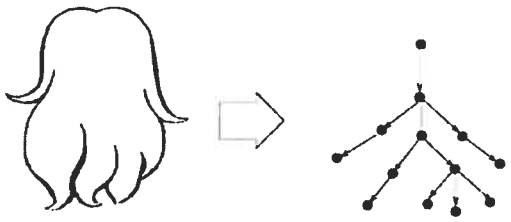
- une courbe squelette, pour définir les mouvements et déformations à grande échelle ;
- une enveloppe, englobant le squelette, pour définir les interactions avec les mèches voisines ;
- un certain nombre de cheveux individuels distribués dans le *wisp*.

Une telle structure est représentée à la Fig. 2.3. L'algorithme pour animer les chevelures avec ce modèle ressemble à ceci :

- calculer les forces appliquées sur les cheveux (gravité, friction de l'air, etc.) ;
- détecter les interactions entre les mèches et avec le modèle 3D ;
- calculer les nouvelles vitesses ;
- calculer les nouvelles positions.

À chaque étape, tous les points de masse sont calculés avant de procéder à l'étape suivante. La position d'une *wisp* est toujours calculée de la racine à la pointe.

Plus récemment, Bertails *et al.* [BKCN03] proposent une extension au modèle multirésolution de Kim et Neumann [KN02] pour l'animation. Cette dernière est réalisée avec le modèle de Plante *et al.* [PCP02]. Ils proposent également une nouvelle structure pour représenter les chevelures : l'*Adaptive Wisp Tree* (AWT). Cette structure permet le raffinement des chevelures de la racine à la pointe, facilitant ainsi la simulation physique et le calcul des interactions entre les mèches. L'illustration suivante (tirée de [KN02]) montre comment la structure peut être créée à partir d'une chevelure.



Pour les animations, la structure évoluera selon les forces externes appliquées sur celle-ci. Par exemple, une branche pourra se diviser en deux (*split*) si trop de changements de vitesses se produisent. Inversement, plusieurs branches peuvent se regrouper (*merge*) si elles sont suffisamment proches et que les vitesses sont suffisamment lentes. Évidemment, la réponse aux collisions est également traitée à chaque niveau de la structure.

L'animation des chevelures a suscité l'intérêt des chercheurs pendant plusieurs années. Plusieurs bonnes solutions ont été proposées, et les résultats obtenus sont très convainquants. Nous allons maintenant étudier le travail réalisé dans le dernier aspect des chevelures photoréalistes : le rendu.

2.1.3 Rendu des cheveux

Le rendu des chevelures (et de la pilosité en général) a attiré l'attention des chercheurs très tôt en infographie. En 1989, Kajiya et Kay [KK89] proposent de faire le rendu de fourrure avec des textures tridimensionnelles nommées *texels*. Un *texel* est un tableau tridimensionnel contenant des paramètres approximant l'aspect visuel de la microgéométrie sous-jacente. Ils possèdent une densité scalaire $\rho(x, y, z)$, une orientation locale $\vec{o} = \left[\begin{array}{ccc} \vec{n}(x,y,z) & \vec{t}(x,y,z) & \vec{b}(x,y,z) \end{array} \right]$ ainsi qu'une fonction d'illumination bidirectionnelle $\Psi(x, y, z, \theta, \phi, \psi)$.

FIG. 2.4 – Rendu de fourrure avec *texels*. Image tirée de Kajiya et Kay [KK89].

FIG. 2.5 – Rendu de chevelure. Image tirée de Anjyo *et al.* [iAUK92].

La densité ρ détermine en tout point du *texel* la quantité de microsurfaces présente. Il s'agit d'une approximation grossière mais généralement suffisante pour obtenir des résultats d'une bonne qualité visuelle. Le vecteur $\vec{o}$ détermine l'orientation locale en tout point du *texel*, avec une normale $\vec{n}$, une tangente $\vec{t}$ ainsi qu'une binormale $\vec{b}$. Finalement, la fonction d'illumination Ψ permet de déterminer comment la lumière interagit à l'intérieur du *texel*. Cette fonction permettra d'approximer les interactions complexes qui se produisent réellement dans les microsurfaces.

Le rendu de ces *texels* peut être lourd (particulièrement à cette époque), mais les résultats obtenus sont impressionnants (Fig. 2.4). Le travail de Kajiya et Kay est considéré comme un classique dans la littérature et a inspiré plusieurs chercheurs, notamment en ce qui concerne l'anisotropie.

Anjyo *et al.* [iAUK92] ont proposé une méthode simple pour faire le rendu des chevelures. Ils considèrent chaque cheveu comme étant un cylindre très mince (Fig. 2.6) où les interactions avec la lumière sont produits. Ils se basent sur le modèle de Blinn [Bli77] pour calculer la composante spéculaire du cheveux, soit :

$$\Phi_s = k_s(\vec{n} \cdot \vec{h})^n$$

où k_s est le coefficient de réflexion spéculaire, $\vec{n}$ est la normale sur le cheveu, $\vec{h}$ est le vecteur bisecteur entre le vecteur vers la lumière (*light*) et le vecteur vers l'oeil (*eye*) et

n est un scalaire déterminant la taille du *highlight* spéculaire. Il s'agit probablement de la manière la plus simple pour faire le rendu d'une chevelure : on rend chaque cheveu individuellement avec un modèle d'illumination de base. Néanmoins, les résultats obtenus sont tout de même impressionnants (Fig. 2.5).

Une autre contribution intéressante est proposée par Yang et Ouhyoung [YO97], qui suggèrent de faire le rendu de chevelures en incluant l'éclairage de derrière (*back-lighting*). Pour y arriver, ils utilisent une table de densité (*density map*, Fig. 2.7) qui accumule chaque cheveu rendu. Cette technique permettra de connaître la densité de cheveux pour toute la scène. Il utilisent la fonction d'illumination proposée par Leblanc *et al.* [LTT91] :

$$h = l_a k_a + \sum s_i l_i [k_d \sin\theta + k_s cos^n(\phi + \theta - \pi)], \tag{2.1}$$

où h est l'intensité du cheveu résultante, l_a est l'intensité de la lumière ambiante, k_a est le coefficient de réflectance ambiante, s_i est la quantité de lumière i qui doit être atténuée par les ombres, l_i est l'intensité émise de lumière, k_d est le coefficient de réflectance diffuse, k_s est le coefficient de réflectance spéculaire, ϕ est l'angle entre le cheveu et l'oeil et θ est l'angle entre le cheveu et la lumière (Fig. 2.8). Une modification à ce modèle est nécessaire pour tenir compte du *density map* :

$$\begin{aligned} h = l_a k_a + & \sum_i s_i l_i [k_d \sin\theta + k_s cos^n(\phi + \theta - \pi)] + \\ & \sum_i (1 - D_i) L_i f(\omega), \end{aligned}$$

$$f(\omega) = \begin{cases} 1 & \text{si } \omega > 160^\circ \text{ (angle où le } \textit{back-lighting} \text{ est actif)} \\ 0 & \text{sinon} \end{cases}$$

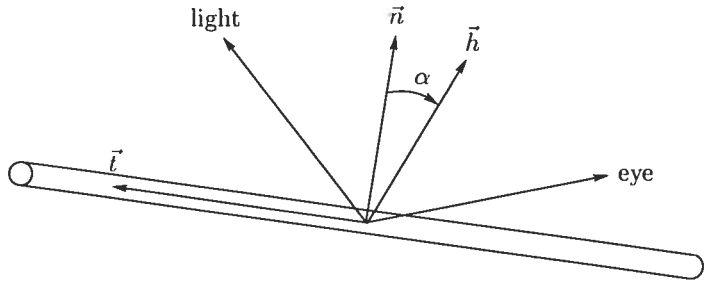


FIG. 2.6 – Modèle d'illumination de Anjyo *et al.* [iAUK92].

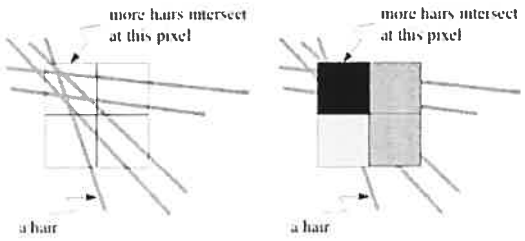


FIG. 2.7 – *Density map.* Images tirées de Yang et Ouhyoung [YO97].

où $(1-D_i)$ représente la fraction de l'intensité de la lumière i qui atteint l'oeil. Cette valeur est obtenue à partir du *density map*. La fonction $f(\omega)$ permet de s'assurer que le *back-lighting* ne survient que lorsque l'oeil regarde vers la lumière, tel qu'illustré à la Fig. 2.8. Cette approche est particulièrement intéressante puisqu'il est possible d'accélérer le rendu avec le matériel graphique et OpenGL. Les résultats obtenus sont également intéressants, mais nous verrons une technique plus moderne qui procure de meilleurs résultats proposée par Kim et Neumann un peu plus loin.

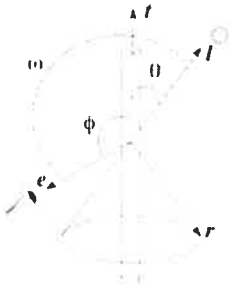


FIG. 2.8: Angles utilisés par Yang et Ouhyoung [YO97].

En 1999, Chen *et al.* [CSDI99] proposent un système de synthèse de chevelures 3D basé sur les mèches décrites à la Section 2.1.2. Ils définissent leurs mèches comme étant l'ensemble des cheveux générés sur un triangle. Ce triangle est déprojeté et modélisé afin de donner une forme géométrique de base à la chevelure. Le rendu des mèches est effectué avec la formule de l'équation 2.1. Également, les cheveux produisent des ombres avec un *shadow map* standard. La contribution principale de ce travail est de produire des chevelures réalistes à moindre coût et avec une représentation simple.

Kim et Neumann [KN02] proposent également une solution pour rendre les chevelures avec leur modèle multirésolutions. Ils dessinent chaque cheveu individuellement en utilisant la primitive de ligne d'OpenGL. Néanmoins, l'aliassage des lignes est problématique. Pour résoudre ce problème, ils utilisent le processus d'antialiassage des lignes d'OpenGL. Cependant, il est requis que les primitives soient triées de derrière à devant

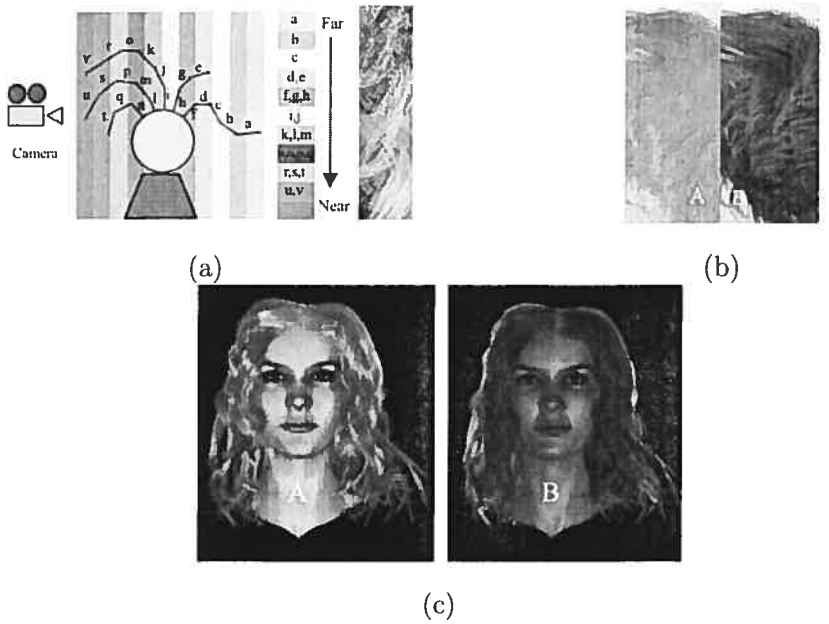


FIG. 2.9 – Rendus réalisés avec la primitive de ligne d'OpenGL et avec utilisation de *deep shadow maps* pour la génération d'ombres. Images tirées de Kim et Neumann [KN02].

(*back-to-front*) pour que l'antialiassage fonctionne. Ils proposent alors de trier chaque segment formant les cheveux (Fig. 2.9(a)). Pour obtenir un maximum de réalisme, des ombres de qualité doivent être générées par les cheveux. Pour y arriver, les auteurs proposent d'utiliser les *deep shadow maps* [LV00]. Un *deep shadow map* est sensiblement identique à un *shadow map* standard, mais l'information de profondeur de toutes les primitives est conservée pour chaque pixel (au lieu de conserver uniquement la primitive la plus près de la lumière pour chaque pixel). De cette manière, il est possible de générer des ombres pour des objets semi-transparents. Cette solution s'adapte bien pour les ombres de chevelures : les cheveux étant très minces et translucides, ils laissent partiellement traverser la lumière. Un résultat avec et sans *deep shadow map* est illustré à la Fig. 2.9(b). Notons que le calcul des ombres avec cette technique peut être très long, mais une fois qu'il est terminé, la chevelure peut être examinée sous n'importe quel angle sans recalcul des ombres (tant que le modèle et la source de lumière reste inchangés). La Fig. 2.9(c) montre une chevelure entière avec illumination de face et de derrière (*backlighting*).

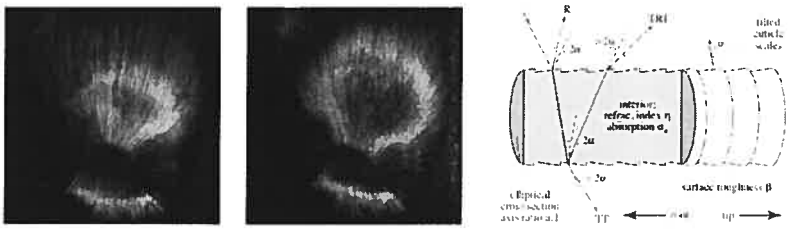
La contribution la plus récente en rendu de chevelures est proposée par Marschner *et al.* [MJC+03]. Ils démontrent qu'un modèle anisotropique d'illumination peut fournir des rendus de meilleure qualité (Fig. 2.10(a)). Évidemment, les calculs nécessaires sont plus lourds à exécuter. La Fig. 2.10(b) permet de voir à haut niveau comment il est possible de produire de tels rendus. Sur l'illustration, les lignes pointillées montrent comment la lumière réfléchie sur un cylindre parfait, utilisé notamment dans le modèle de Anjyo *et al.* [iAUK92]. Les lignes pleines montrent comment la lumière réfléchie sur une micro-structure similaire à celle d'un cheveu humain.

Jusqu'à maintenant, nous avons étudié plusieurs contributions concernant la modélisation, l'animation et le rendu des chevelures. Cependant, nous avons seulement exploré l'aspect photoréaliste des chevelures, alors que nous avons besoin d'un modèle non-photoréaliste. Dans la prochaine section, nous explorerons les travaux effectués en rendu non-photoréaliste qui pourraient nous être utiles.

2.2 Non-photoréalisme

Durant la dernière décennie, plusieurs dizaines de contributions ont été apportées concernant le rendu non-photoréaliste. Il est donc surprenant de constater qu'aucune d'entre elles ne s'est intéressée au rendu et à l'animation de chevelures alors que plusieurs travaux ont été effectués sur le sujet en rendu et animation photoréaliste.

En fait, une poignée d'auteurs se sont intéressés, de près ou de loin, au rendu de pilosité non-photoréaliste. Parmis ces auteurs se distinguent Kowalski et Markosian,



(a) (b)

FIG. 2.10 – Rendu anisotropique de chevelures. Images tirées de Marschner *et al.* [MJC+03].

qui se sont attardés au rendu à partir de silhouettes des objets 3D. Les silhouettes dans une image sont une source inestimable d'informations : nous identifions la plupart des objets de notre environnement par les contrastes que leurs contours provoquent. Kowalski et Markosian sont partis de cette idée pour proposer un système de rendu basé sur les contours des objets. Nous allons étudier leurs travaux avec plus de détail dans la prochaine section.

2.2.1 Rendu basé sur les silhouettes

En 1999, Kowalski *et al.* [KMN+99, MMK+00] proposent une approche pour faire du rendu basé sur les silhouettes des éléments d'une scène. Pour y arriver, ils utilisent des textures procédurales nommées *graftals*. Les *graftals* sont distribués un peu partout sur les modèles 3D, et leur apparence est dépendante de leur position par rapport à la silhouette courante. Nous allons maintenant discuter des *graftals* plus en détails et discuterons éventuellement de leurs avantages et inconvénients.

La Fig. 2.11 (tirée de [KMN+99]) montre un exemple d'image rendue avec les *graftals*. Nous voyons que l'apparence des structures formant les poils (les *graftals*) change en fonction de leur position sur le modèle. En fait, elles changent d'apparence selon leur position relative à la silhouette du modèle, qui est elle-même dépendante de la position de la caméra. Les *graftals* sont en fait des banques de structures géométriques, tel qu'illustré sur la figure suivante (également tirée de [KMN+99]) :

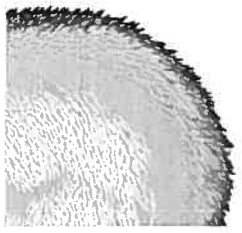
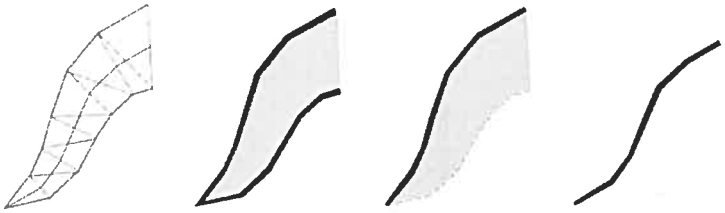


FIG. 2.11: Example de rendu par *graftals*.



Sur cette illustration, nous voyons les polygones originaux utilisés pour faire le rendu original du *graftal* ainsi que les autres structures utilisées pour les changements d'apparences. La Fig. 2.12 montre le processus utilisé pour choisir les différentes structures

dépendamment de la position de celles-ci.

Le rendu des *graftals* nécessite la connaissance des silhouettes de la scène. Les silhouettes sont définies comme étant tous les segments communs à un polygone *front-facing* et à un polygone *back-facing*[†]. Évidemment, ceci est seulement vrai si les polygones sont triangulaires, c'est pourquoi nous considérons les scènes 3D comme étant triangularisées a priori.

Le positionnement des *graftals* requiert un peu de travail supplémentaire. Nous désirons obtenir une densité constante (en espace image) des *graftals*. Pour y arriver, les auteurs se sont inspirés du travail de Salisbury *et al.* [SWHS97] qui arrivent à produire des dessins à l'encre avec une densité contrôlable. À chaque trait rendu, le trait est également soustrait d'une image de différence (*difference image algorithm* — DIA). Le prochain trait est donc placé en analysant le DIA pour essayer de trouver un endroit en besoin de noirceur et en générant le nouveau trait à cet endroit. L'image résultante sera un ensemble de marques dont la densité reflète l'intensité en ton de gris originale.

Un processus similaire est utilisé pour positionner les *graftals* dans l'image. Chaque *graftal* rendu sera également dessiné dans une image de référence de couleurs (*color reference image*). Les régions plus sombres dans l'image de référence nécessiteront une plus grande densité de *graftals*. Cette image sera nommée *desire image*, puisqu'elle indique le

[†]Pour déterminer si un polygone est *front-facing*, il suffit de calculer le produit scalaire (*dot product*) entre la normale du polygone $\vec{n}$ et le vecteur orientation de la caméra $\vec{v}$, i.e. un polygone est *front-facing* si et seulement si $\vec{n} \cdot \vec{v} < 0$.

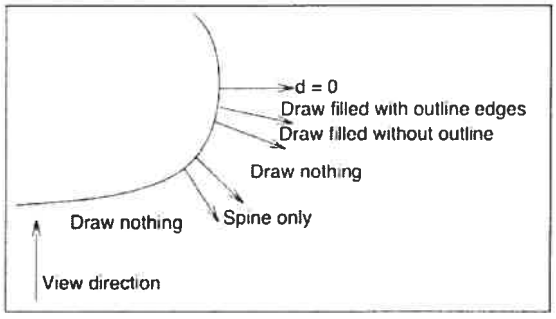


FIG. 2.12 – Rendu des *graftals* à partir de leurs positions. Image tirée de Kowalski *et al.* [KMN+99]

niveau de besoin d'un *graftal* à chaque endroit. Également, l'usager peut volontairement noircir certaines régions de l'image de référence pour forcer une plus grande densité de *graftals*.

Pour donner l'impression que les *graftals* sont collés sur le modèle 3D, un *mapping* de la position 2D de l'écran vers la scène 3D est nécessaire. Ceci est effectué par un lancement d'un rayon dans la scène à partir de la position image. Notons que le rayon est intersecté avec une seule primitive qui est connue à l'avance, ce qui permet d'obtenir la position en $O(1)$ (par *graftal*). Les résultats obtenus avec les *graftals* sont très intéressants. Ils sont particulièrement efficaces pour donner un effet de fourrure avec un aspect très *cartoon*. Nous pourrions donc être tentés de les utiliser pour faire le rendu de chevelures avec un style *cartoon*. Nous allons voir plus loin jusqu'à quel point ils pourront nous être utiles dans ce contexte.

Chapitre 3

Analyse des chevelures NPR

Get the habit of analysis — analysis will in time enable synthesis to become your habit of mind.

Frank Lloyd Wright

Analyser les chevelures produites par les artistes est la première étape à franchir avant de tenter de les reproduire. Il est important de bien comprendre les techniques utilisées ainsi que leurs origines avant de produire des outils intuitifs à utiliser. Dans ce chapitre, nous allons parcourir plusieurs styles de chevelures non-photoréalistes produits par différents artistes et allons tenter d'en extraire le plus d'informations possibles dans l'espoir de les reproduire fidèlement avec des outils puissants.

Pour commencer, regardons la Fig. 3.1. Une étude attentive de l'illustration nous indique que les visages de chaque personnage sont identiques. Cependant, au premier coup d'oeil, il nous semble évident que les personnages sont totalement différents : certains ont un air masculin, d'autres un air féminin, chacun d'eux a un style bien défini, etc. Les chevelures sont néanmoins l'unique différence sur tous ces personnages. Ceci est une bonne indication que les chevelures véhiculent plusieurs aspect avec eux, notamment la personnalité et les émotions, comme nous le verrons plus loin. Également, bien que

FIG. 3.1 – Impact des cheveux sur la personnalité. Images tirées de [bak].

les dessins sur la figure soient relativement simplistes, les cheveux ajoutent un niveau de complexité intéressant aux illustrations. Ils apportent donc un outil efficace aux artistes pour ajouter des effets subtils à leurs oeuvres. D'autres exemples plus intéressants viendront au cours de ce chapitre. Pour l'instant, nous allons voir comment les dessins animés sont généralement réalisés dans l'industrie.

3.1 Étapes de production

Afin de bien cerner comment les chevelures sont produites, il est important de comprendre la structure générale de production des dessins. Il existe beaucoup de variance dans l'industrie à savoir comment les dessins sont produits, mais des approches similaires sont tout de même présentes. Les sections qui suivront discuteront avec plus de détails les étapes standards de production (Fig. 3.2), c'est-à-dire le dessin au crayon, l'encrage et le coloriage.

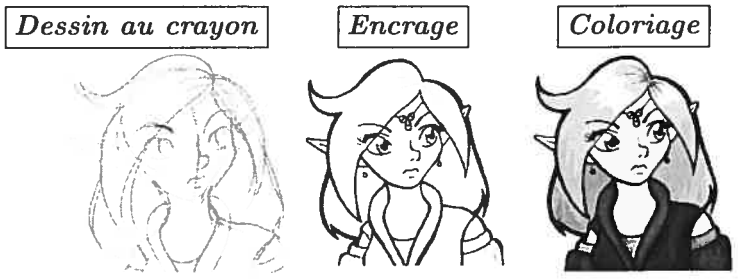


FIG. 3.2 – Étapes de production couramment utilisées dans l'industrie. Images tirée de [Tae].

3.1.1 Dessin au crayon

Tel qu'illustré à la Fig. 3.2(a), le dessin au crayon (*penciling*) consiste à produire un premier croquis de base. Cette étape est généralement faite au plomb pour permettre des corrections mineures facilement. Cette étape est presque toujours produite à la main, sans aucun outil informatisé. Elle est généralement considérée comme étant celle comportant un plus haut niveau artistique, bien que les étapes subséquentes, qui sont moins bien connues du public, soient également essentielles à la création d'images de qualité supérieure.

3.1.2 Encrage

Probablement la moins bien connue des étapes de production, l'encrage (*inking*), tel qu'illustré à la Fig. 3.2(b), consiste à redéfinir les contours à l'encre et à définir nettement les reflets. Il s'agit d'une étape essentielle à la production d'images, et plus particulièrement de chevelures, de haute qualité. De préférence, cette étape sera produite par le même artiste qui a effectué le dessin au crayon, mais les contraintes de temps et d'argent de l'industrie ne le permettent pas toujours. Pour ces raisons, l'artiste effectuant le dessin au crayon se doit de transmettre de l'information sur la structure de la scène à l'artiste qui effectuera l'encrage. Ceci est généralement réalisé par des marques sur le dessin au plomb, indiquant les régions qui doivent être totalement noircies. L'illustration de la Fig. 3.3 en est un exemple. Il est important de noter que cette étape est exclusivement effectuée à la main avec des outils d'encrage difficiles à manipuler et très enclin aux erreurs.

3.1.3 Coloriage

Le coloriage (*coloring*, Fig. 3.2(c)) consiste évidemment à ajouter les couleurs à l'image produite à l'étape d'encrage. Il s'agit probablement de la seule étape généralement réalisée avec des outils informatisés, tel que *Adobe Photoshop*. Ceci permet à l'artiste de choisir parmi différentes configurations de couleurs et de les tester, favorisant ainsi de meilleurs résultats. Ceci élimine également le risque de détruire complètement

FIG. 3.3 – Exemple d'encrage à partir d'un dessin au plomb. Images tirées de [Ala].

le travail effectué aux étapes précédentes, risque qui est toujours présent à l'étape de l'encrage.

3.2 Production de chevelures

La création des chevelures forme généralement un *bottleneck* dans la production. La structure complexe des chevelures les rendent particulièrement difficiles à dessiner. La solution adoptée par les artistes est d'approximer l'effet que les chevelures produisent lorsque vues de loin. Plusieurs techniques ont été développées pour réussir à produire cet effet. Une des plus populaires se nomme le *feathering* et sera discutée plus en détail dans cette section.

Le *feathering* est une technique de dessin très générale qui permet aux artistes de produire différents tons de gris avec des couleurs opaques, comme de l'encre noir. La Fig. 3.4 montre un exemple simple de *feathering*, et comment il est produit manuellement. Cette technique est utilisée dans plusieurs contextes, comme pour produire des ombres floues (Fig. 3.3), mais elle est particulièrement puissante pour créer des chevelures de qualité supérieure. Les artistes ont adopté cette technique pour sa simplicité et la qualité des images qu'elle permet de produire.

Pour les chevelures, les artistes traceront des lignes dans la direction générale des cheveux, et créeront des reflets avec des taches d'encre pour faire le *feathering*. En utilisant cette approche, l'aspect anisotropique des cheveux se reflètera naturellement, et

FIG. 3.4 – Exemple simple de *feathering* produit manuellement.

FIG. 3.5 – Exemples de chevelures produites avec la technique du *feathering*. Images réalisées par Ed Benes et Vince Russell.

ce, avec une technique très simple. Quelques résultats obtenus avec cette approche sont illustrés à la Fig. 3.5. Nous pouvons remarquer la qualité supérieure de ces chevelures par rapport à d'autres techniques. Une contribution majeure dans le cadre de ce travail sera de proposer une approche permettant de reproduire fidèlement le *feathering* avec une technique simple et flexible (Chapitre 5).

3.3 Méthode procédurale de production de chevelures

La Section 2.2 a présenté l'approche de Kowalski *et al.* pour générer des *graftals* basés sur les silhouettes d'une image. Cette technique pouvait très bien être adaptée au rendu de poils ou de fourrure, mais qu'en est-il pour les chevelures ? Quels sont les avantages et inconvénients de cette approche dans ce contexte ? Cette section tentera de répondre à ces questions, et proposera un *framework* pour permettre la modélisation et le rendu de chevelures non-photoréalistes.

3.3.1 *Graftals* : révision

Les *graftals* peuvent être utilisés pour modéliser les chevelures. Un graftal est un ensemble de structures dont les propriétés changent selon l'angle de celles-ci par rapport à la caméra. Cette approche est parfaitement abordable pour créer des chevelures de type

cartoon : l'artiste pourrait définir des traits minimals lorsque la normale du modèle pointe vers la caméra, et définir des mèches plus développées sur les silhouettes. Cette approche donne des résultats très intéressants (illustration à gauche, tirée de [KMN+99]).

Cependant, il y a un problème majeur avec cette approche : l'interface. En fait, celle-ci est inexistante dans le modèle proposé par Kowalski *et al.*, ce qui rend le système inutilisable pour des artistes, peu importe la qualité artistique des rendus obtenus.

La première approche proposée pour créer un nouveau *graftal* est de générer une sous-classe *C++* [Str00] du modèle de base, et de réécrire les fonctions de rendu pour celui-ci. Cette approche lourde s'avère un échec total pour l'utilisabilité du système, les auteurs ont donc proposé une autre approche plus flexible et plus facile d'utilisation : les *graftal description files* [MMK+00]. Ces fichiers permettent de décrire le comportement voulu des *graftals* par un certain nombre de paramètres bien définis. Parmi ces paramètres se distinguent la géométrie, la couleur à chaque sommet ainsi que les multiples résolutions des *graftals*.

Malgré les améliorations apportées à la modélisation, il n'en demeure pas moins que les artistes traditionnels sont dans l'incapacité d'utiliser le système. Les artistes désirent un *feedback* interactif de leurs manipulations. Plus important encore, ils désirent créer leurs oeuvres avec les mêmes mouvements qu'ils le feraient dans la réalité. Les *graftals* ne sont d'aucune utilité dans ce contexte puisqu'ils sont principalement basés sur les silhouettes des objets 3D, et non sur les traits des images 2D.

Ce qu'il nous faut, c'est un système permettant aux artistes traditionnels de créer simplement et intuitivement des mèches de cheveux, en leur permettant de répéter les mouvements qu'ils utiliseraient dans la réalité. Permettre exactement ces mouvements est très difficile étant donné les contraintes imposées par le matériel informatique, mais nous pouvons simplifier la modélisation par un système de dessin par traits.

3.3.2 *Framework* de base

Nous allons maintenant proposer un *framework* pour un système permettant la modélisation, la génération et le rendu de chevelures. Les chapitres qui suivront discuteront

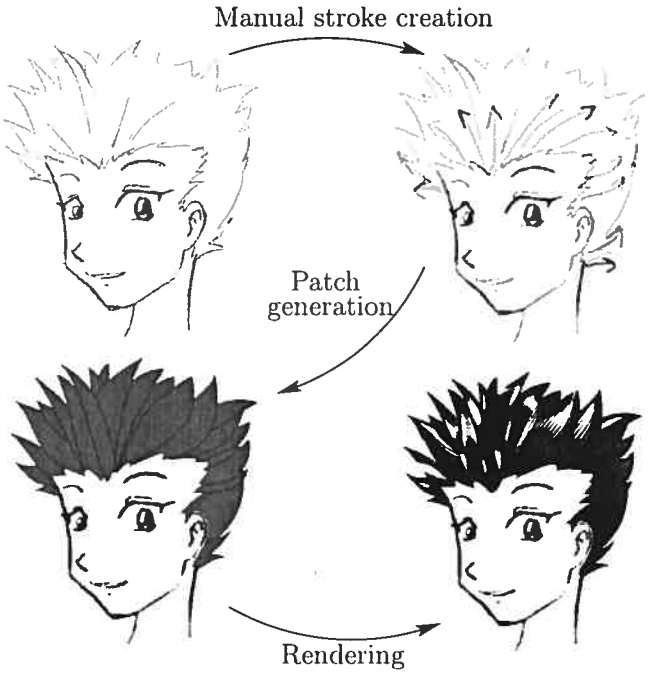


FIG. 3.6 – *Framework* général pour la production de chevelures NPR. Image originale tirée de [bak].

de chacune des étapes avec plus de détails. L'illustration de la Fig. 3.6 montre schématiquement les différentes étapes de la production, *i.e.* la création de traits, la génération des mèches et le rendu.

La première étape du sytème est de dessiner manuellement des traits dans l'orientation des chevelures. Les traits seront utilisés pour générer des *patches* 3D planaires (deuxième étape). La dernière étape consiste à faire le rendu de ces *patches* en utilisant un certain nombre de paramètres permettant de définir le style du *feathering*.

Ce chapitre a tenté de cerner les problèmes à résoudre concernant la production de chevelures NPR. Les chapitres qui suivent proposent des solutions à ces problèmes. Plus particulièrement, le Chapitre 4 discute de la génération de *patches* à partir de traits produits par l'artiste alors que le Chapitre 5 propose une solution au rendu de *feathering*.

Chapitre 4

Modélisation des mèches

Do not quench your inspiration and your imagination ;
do not become the slave of your model.

Vincent Van Gogh

Tel que mentionné au Chapitre 3, nous désirons permettre la modélisation des chevelures avec une technique se rapprochant le plus possible des méthodes traditionnelles utilisées par les artistes. Ce chapitre sera consacré à cet effet. Nous verrons comment il est possible à partir de traits de générer des *patches* (*NURBS*) en conservant l'orientation et la forme, simplifiant ainsi la modélisation de chevelures complexes. Le chapitre aura la structure suivante : tout d'abord, nous étudierons la théorie de base de la représentation des courbes et des surfaces (Section 4.1) qui nous sera nécessaire éventuellement pour nous permettre de les générer à partir de traits. Nous débuterons en révisant les courbes paramétriques pour étendre le sujet vers les surfaces paramétriques. Ensuite, nous proposerons une interface simple pour permettre aux artistes de générer des chevelures en utilisant cette représentation (Section 4.2).

4.1 Représentation des courbes et surfaces

Il existe plusieurs modèles mathématiques pour représenter des courbes. L'approche classique est de fournir une fonction du type $y = f(x)$ et de dessiner, pour chaque x, l'équivalent en y^* . Cependant, cette approche est très limitée, particulièrement dû au fait qu'il ne doit exister qu'un seul y pour chaque x. Dessiner un cercle doit donc se faire en deux temps : dessiner le demi-cercle supérieur et dessiner le demi-cercle inférieur. De plus, dessiner des lignes verticales (avec une pente infinie) est difficile et doit être traité séparément. Cette méthode est donc beaucoup plus contraignante qu'on l'imagine : il faut trouver une solution plus flexible.

La deuxième solution qui nous vient à l'esprit est d'utiliser une fonction implicite du type $f(x, y) = 0$. Cette méthode ne règle aucun problème. Si nous reprenons l'exemple du cercle, nous voulons dessiner la fonction $x^2 + y^2 - r^2 = 0$. Hors, cette fonction possède plusieurs solutions valides, il faudrait donc ajouter des contraintes du genre $y \geq 0$ qui ne peuvent être intégrées avec élégance dans la fonction. Encore une fois, cette approche n'est pas suffisamment flexible.

Les problèmes rencontrés peuvent être éliminés en utilisant une représentation paramétrique des courbes. Une courbe paramétrique est une fonction du type $Q(t)$ avec $0 \leq t \leq 1$, de telle manière que

$$Q(t) = \begin{bmatrix} x(t) & y(t) & z(t) \end{bmatrix}.$$

Chacune des fonctions qui composent $Q(t)$ peut être une fonction de degré quelconque, mais des fonctions de degré 3 sont généralement utilisées pour plusieurs raisons :

- les fonctions de plus bas degré manquent de flexibilité ;
- les fonctions de plus haut niveau sont plus difficiles à contrôler, et sont plus coûteuses à calculer ;
- les fonctions de degré 3 sont les premières à s'étendre à la troisième dimension, permettant la création de courbes non-planaires.

*Pour l'instant, nous allons nous concentrer sur des fonctions à deux variables, nous généraliserons à trois variables pour le modèle paramétrique.

Les fonctions qui composent $Q(t) = \begin{bmatrix} x(t) & y(t) & z(t) \end{bmatrix}$ ont donc la forme suivante :

$$\begin{aligned} x(t) &= a_x t^3 + b_x t^2 + c_x t + d_x, \\ y(t) &= a_y t^3 + b_y t^2 + c_y t + d_y, \\ z(t) &= a_z t^3 + b_z t^2 + c_z t + d_z, \quad 0 \leq t \leq 1. \end{aligned}$$

Cette forme est plutôt lourde à traiter. Elle est généralement simplifiée sous la forme matricielle suivante : $Q(t) = T \cdot C$ avec

$$T = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \quad \text{et} \quad C = \begin{bmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \\ d_x & d_y & d_z \end{bmatrix},$$

cette notation étant plus compacte et plus élégante. Néanmoins, il reste toujours un problème : le choix des coefficients est toujours laissé à découvert. L'usager ne peut pas choisir intuitivement les coefficients de manière cohérente pour donner une forme voulue à la courbe. La solution utilisée pour régler ce problème est de séparer la matrice des coefficients C en deux matrices distinctes : une matrice de base (*basis matrix*, M) et un vecteur de géométrie (*geometry vector*, G). La matrice de base est fixée à l'avance pour un type de courbe donné, et le vecteur géometrie est utilisé par l'usager pour déterminer le comportement de la courbe. La forme finale de la courbe est donc $Q(t) = T \cdot M \cdot G$, avec

$$T = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix}, \quad M = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix} \quad \text{et} \quad G = \begin{bmatrix} G_1 \\ G_2 \\ G_3 \\ G_4 \end{bmatrix}.$$

Les techniques utilisées pour spécifier des matrices M ainsi que la forme des vecteurs G dépassent le cadre de ce travail, mais elles sont discutées en détail dans l'ouvrage de Foley *et al.* [FvDFH96] ainsi que dans le livre de Piegl et Tiller [PT95]. Nous allons tout de même résumer les principales courbes paramétriques dans la prochaine section.

4.1.1 Courbes paramétriques standards

Les courbes d'*Hermite* proposent d'utiliser un vecteur géométrie de la forme $G_H = \begin{bmatrix} P_1 & P_4 & \vec{R}_1 & \vec{R}_4 \end{bmatrix}^T$ où P_1 et P_4 sont les points déterminant les extrémités de la courbe et les vecteurs $\vec{R}_1$ et $\vec{R}_4$ déterminent les "vitesses" à ces points, tel qu'illustré sur la Fig. 4.1(a). Cette forme est très simple, mais un peu contre-intuitive à utiliser puisque l'usager doit spécifier deux points ainsi que deux vecteurs. Il serait préférable de transformer les vecteurs en points, pour fournir un vecteur géométrie à quatre points. C'est précisement ce que font les courbes de *Bézier*, qui proposent un vecteur géométrie de la forme $G_B = \begin{bmatrix} P_1 & P_2 & P_3 & P_4 \end{bmatrix}^T$. Les relations entre les nouveaux points introduits et les vecteurs d'*Hermite* sont $\vec{R}_1 = 3(P_2 - P_1)$ et $\vec{R}_4 = 3(P_4 - P_3)$. Cette nouvelle forme est non seulement plus simple à utiliser, mais possède également certaines propriétés intéressantes. En particulier, les courbes de *Bézier* offrent la garantie que la courbe rendue sera toujours comprise dans l'enveloppe convexe défini par les points de contrôles.[†] Un exemple de courbe de *Bézier* est illustré sur la Fig. 4.1(b).

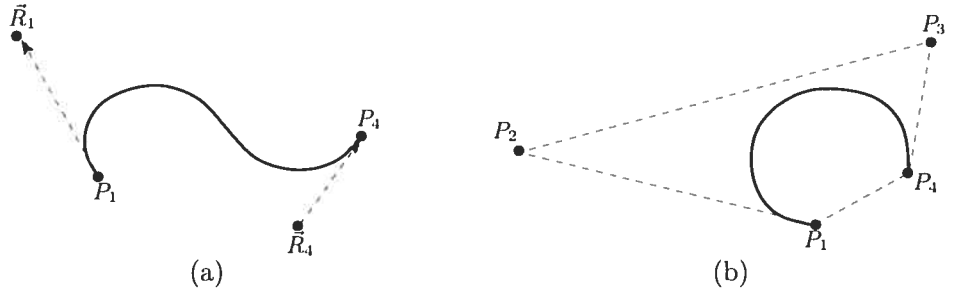


FIG. 4.1 – Exemple de courbes (a) d'*Hermite* et (b) de *Bézier*.

Dessiner de telles courbes est particulièrement simple : il s'agit essentiellement de dessiner des traits linéaires entre chaque paire de points évaluée sur la courbe à des valeurs de t choisies.

Jusqu'à maintenant, nous avons seulement étudié des courbes paramétriques simples qui peuvent être dessinées en utilisant le paramètre t entre 0 et 1. Néanmoins, il s'agit toujours de courbes produites avec des polynômes de degré 3, ce qui est assez restrictif. Plusieurs usagers désireront plus de contrôle pour permettre des courbes plus longues

[†]Les points de contrôles sont les points spécifiés dans le vecteur géométrie G.

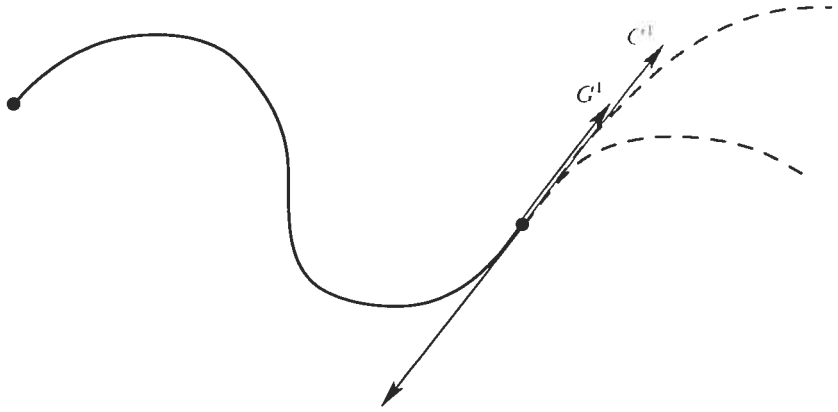


FIG. 4.2 – Différence entre les continuités G^1 et C^1.

par exemple. Pour y arriver, nous aimerions pouvoir joindre ensemble plusieurs de ces courbes de degré 3 pour faire un trait plus long et plus flexible. Chacune des courbes deviendrait alors un segment de la courbe totale. Le problème revient maintenant à joindre ensemble les segments et de garantir une bonne continuité.

La continuité la plus simple est lorsque deux segments se touchent à leurs extrémités ; on dira alors qu'il s'agit d'une continuité géométrique G^0. Si la direction (mais pas nécessairement la longueur) des vecteurs tangents aux extrémités est la même, on dira qu'il s'agit d'une continuité géométrique G^1. Similairement, si les vecteurs tangents sont les mêmes (direction et longueur), on dira qu'il s'agit d'une continuité C^1. Ainsi de suite, si les directions et longueurs de $\frac{d^n}{dt^n}Q(t)$ sont les mêmes aux extrémités, on dira que la courbe a une continuité C^n. Bien que les continuités G^1 soient moins restrictives que C^1, elles auront l'air tout aussi douces, comme le démontre la Fig. 4.2.

Les courbes d'*Hermite* et de *Bézier* n'offrent implicitement aucune garantie de continuité, le programmeur doit donc s'en assurer en apportant de multiples contraintes plus difficiles à gérer, notamment :

- modifier la position de l'extrémité d'un segment doit également changer la position des points de contrôle précédents et suivants ;
- modifier la position d'un point de contrôle intermédiaire (entre les extrémités d'un segment) doit également changer la position d'un point de contrôle intermédiaire du segment précédent ou suivant ;

– les extrémités de la courbe entière doivent être traitées séparément.

Toutes ces contraintes sont difficiles à gérer. Des problèmes peuvent facilement survenir, et nous n'avons pas encore discuté des *patches* paramétriques, où ces problèmes sont autrement plus difficiles à gérer. Les courbes *BSpline* offrent une alternative intéressante aux problèmes de continuité. En particulier, les *BSplines* assurent une continuité C^2 partout sur la courbe, ce qui les rend plus douce que *Hermite* ou *Bézier*. Pour y arriver, l'affichage doit s'effectuer avec des masques sur les points de contrôle. Par exemple, le premier segment s'affiche avec les points $P_1, \ldots, P_4$, le deuxième segment avec les points $P_2, \ldots, P_5$, ainsi de suite. Chacun des segments se dessine normalement, à une exception près : le paramètre t pourra être séquentiel d'un segment à l'autre, avec un simple changement de variable. Par exemple, le premier segment pourra se dessiner avec $0 \leq t \leq 1$, le deuxième avec $1 \leq t \leq 2$, ainsi de suite. La connexion entre deux segments s'appelle un noeud (*knot*) dans les *BSplines*, et la valeur du paramètre t à ces endroit s'appelle la valeur du noeud (*knot value*). Dans les *BSplines*, les noeuds sont également espacés, c'est-à-dire que la différence entre deux valeurs de noeud successives est la même sur toute la courbe, on dira alors que la *BSpline* est uniforme. Une autre variété de *BSplines* non-uniformes procure une flexibilité accrue, il s'agit des *NURBS* (*Non-Uniform Rational BSpline*) [PT95].

4.1.2 *BSpline* non-uniformes et rationnelles (*NURBS*)

Une forme plus générique des courbes paramétriques est les ratios de polynômes (d'où le terme "rationnel") :

$$Q(t) = \left[\; x(t) = \frac{X(t)}{W(t)} \quad y(t) = \frac{Y(t)}{W(t)} \quad z(t) = \frac{Z(t)}{W(t)} \; \right].$$

Cette forme peut être considérée comme une représentation homogène des courbes, avec la forme

$$Q(t) = \left[\; X(t) \quad Y(t) \quad Z(t) \quad W(t) \; \right].$$

Évidemment, toute courbe non-rationnelle peut être transformée en forme rationnelle en ajoutant $W(t) = 1$ comme quatrième élément. N'importe quel type de courbe peut

être utilisé à la base des courbes rationnelles, mais il s'agit d'une *NURBS* uniquement si la courbe à la base est une *BSpline*.

Un des avantages majeurs des courbes rationnelles est qu'elles sont invariantes aux transformations de rotations, mises à échelle, translations et transformations perspectives des points de contrôle. Ceci permet d'accélérer le rendu et d'attendre après la projection des points de contrôle avant de dessiner la courbe.

Les *NURBS* sont également non-uniformes, ce qui signifie que l'espacement entre deux *knots* n'est pas nécessairement le même. Les courbes non-uniformes sont plus flexibles que les courbes uniformes : les joints de continuité C^2 peuvent être réduits à C^1 ou C^0 sans problème, permettant une forme arbitraire aux courbes. Cependant, des paramètres supplémentaires sont nécessaires pour contrôler cette flexbilité accrue. Comme toujours, la courbe paramétrique est approximée par une suite de segments de type *BSpline*, mais une séquence de *knot values* doit également être spécifiée, déterminant les valeurs de t aux différents joints sur la courbe. Il y a toujours quatre *knots* de plus que de points de contrôle sur la courbe. Par exemple, une courbe d'un seul segment aura huit *knots* (de $t_0, \ldots, t_7$), et la courbe sera définie dans l'intervalle t_3 à t_4. Plus d'information à ce sujet est disponible dans le livre de Foley *et al.* [FvDFH96] et dans le livre consacré aux *NURBS* de Piegl et Tiller [PT95].

Les *NURBS* seront utilisées dans le cadre de ce travail. La raison principale de ce choix est basée sur la flexibilité qu'elles procurent. Cette flexibilité sera particulièrement utile pour regrouper ensemble des paires de courbes afin de générer des *patches* qui sont discutées dans la prochaine section.

4.1.3 Relation entre courbes et *patches*

Les *patches* (surfaces paramétriques) sont en fait une généralisation des courbes paramétriques. Essentiellement, le vecteur géométrie sera une matrice 4×4 plutôt qu'un vecteur de 4 éléments. La disposition des points de contrôle dans cette matrice dépend du modèle choisi, mais la Fig. 4.3 montre un exemple de *patch* de *Bézier* avec l'organisation des points. La surface sera fonction de deux paramètres, s et t, un pour la longueur et

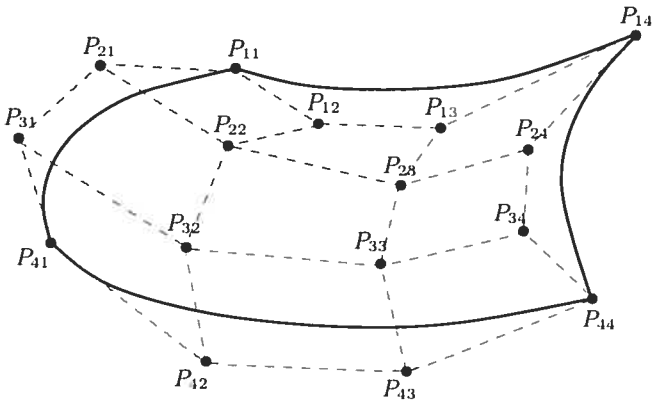


FIG. 4.3 – *Patch* de *Bézier* avec points de contrôle.

l'autre pour la largeur. Pour un t fixé, la fonction se résume à une courbe :

$$Q(s,t) = S \cdot M \cdot G(t) = S \cdot M \cdot \begin{bmatrix} G_1(t) \\ G_2(t) \\ G_3(t) \\ G_4(t) \end{bmatrix}.$$

Puisque $G_i = \begin{bmatrix} g_{i1} & g_{i2} & g_{i3} & g_{i4} \end{bmatrix}^T$ (les g minuscules représentent les points de contrôle) et que $G_i(t) = T \cdot M \cdot G_i$, il est facile de montrer que la forme finale d'une *patch* est la suivante [FvDFH96] :

$$Q(s,t) = S \cdot M \cdot \begin{bmatrix} g_{11} & g_{12} & g_{13} & g_{14} \\ g_{21} & g_{22} & g_{23} & g_{24} \\ g_{31} & g_{32} & g_{33} & g_{34} \\ g_{41} & g_{42} & g_{43} & g_{44} \end{bmatrix} \cdot M^T \cdot T^T,$$

S ayant évidemment la forme $\begin{bmatrix} s^3 & s^2 & s & 1 \end{bmatrix}$.

Afficher de telles surfaces est relativement simple : la procédure proposée par Foley *et al.* est explicitée à l'Algorithme 4.1. Cependant, la procédure dessine une grille à la résolution voulue et nous ne désirons pas toujours dessiner une *patch* comme un ensemble de lignes ; le Chapitre 5 propose une technique pour faire le rendu de *patches* pour le *feathering*.

input : *Coefficients de* $Q(s,t)$
input : n_s (nombre de lignes en s)
input : n_t (nombre de lignes en t)
input : n (nombre de segments pour dessiner chaque ligne)

$\delta = 1/n$;
$\delta_s = 1/(n_s - 1)$;
$\delta_t = 1/(n_t - 1)$;
for $s \leftarrow 0$ ***to*** *1* ***by*** δ_s **do**
    (Les fonctions X, Y et Z évaluent un point aux coordonnées données) ;
    move_to($X(s,0), Y(s,0), Z(s,0)$) ;
    for $t \leftarrow \delta$ ***to*** *1* ***by*** δ **do** line_to($X(s,t), Y(s,t), Z(s,t)$) ;
end
for $t \leftarrow 0$ ***to*** *1* ***by*** δ_t **do**
    move_to($X(0,t), Y(0,t), Z(0.t)$) ;
    for $s \leftarrow \delta$ ***to*** *1* ***by*** δ **do** line_to($X(s,t), Y(s,t), Z(s,t)$) ;
end

Algorithme 4.1: Procédure pour dessiner une surface paramétrique.

Connecter plusieurs segments de surface ensemble, dans le but de générer des surfaces plus grandes et plus flexibles, induit les mêmes problèmes que la continuité des courbes, mais ils sont plus difficiles à traiter étant donné la dimension supplémentaire. Les *patches* de *BSpline*, et particulièrement les *NURBS* résolvent ces problèmes élégamment. Dessiner une grande surface paramétrique composée de plusieurs segments de *patches* revient à faire des masques 2D sur les points de contrôle, et de faire le rendu de ces segments indépendamment.

4.2 Interface de modélisation

Jusqu'à maintenant, nous avons discuté de la théorie de base des courbes et surfaces paramétriques, de l'élégance et de la puissance qu'elles procurent, mais nous n'avons aucune idée à savoir comment les utiliser dans un contexte de modélisation de chevelures. Cette section aura pour but de combler cette lacune. Plus particulièrement, nous verrons comment il est possible de convertir un trait dessiné manuellement par un artiste en

courbe paramétrique. Nous verrons également comment il est possible d'utiliser deux de ces courbes et de les joindre ensemble pour former une *patch*.

4.2.1 Génération de courbes

À partir d'une courbe dessinée à la main par un artiste, nous devons trouver un moyen de générer une courbe paramétrique. La courbe dessinée originalement sera en fait un ensemble de points qui sont générés par un dispositif d'entrée (*input device*), typiquement une souris. Nous pouvons décrire ces points comme l'ensemble des n couples (x, y) suivants[‡] :

$$\mathcal{P} = \left\{ \ (x_0, y_0), \quad (x_1, y_1), \quad \ldots, \quad (x_{n-1}, y_{n-1}) \ \right\}.$$

Notre but sera de trouver une *NURBS* (points de contrôle et *knots*) dont la courbe passe le plus près possible de ces points. Il existe plusieurs techniques pour trouver une telle courbe ; elles sont d'ailleurs décrites en détail dans le livre de Piegl et Tiller [PT95]. La plupart des procédures permettant d'adapter une courbe sur des points se divisent en deux catégories : globales ou locales. Dans l'approche globale, on utilise un ensemble d'équations ou encore un problème d'optimisation qui sera résolu. Les calculs requis peuvent être relativement simples si les points de contrôle sont les seuls inconnus (les *knots* et les poids sont connus à l'avance). L'approche locale est basée sur la géométrie de la courbe : les points de l'ensemble $\mathcal{P}$ sont utilisés séquentiellement pour placer les points de contrôle ainsi que les autres variables. L'approche utilisée dans le cadre de ce travail est globale et approxime l'ensemble des points de $\mathcal{P}$ avec une erreur permise E.[§] Nous débutons avec une courbe contenant plusieurs points de contrôle (typiquement tous les points de $\mathcal{P}$) avec lesquelles nous générons une première courbe. Ensuite, des points sont rejetés si l'erreur induite sur la courbe est plus faible que E. La déviation entre les courbes est évaluée avec les moindres carrés.

Bien que cette approche soit simpliste, elle a démontré être capable de reproduire fidèlement les courbes dessinées manuellement, peu importe la complexité de celles-ci.

[‡]La coordonnée z est ignorée puisque les points sont spécifiés en espace image, donc en 2D.

[§]L'implémentation de l'algorithme est réalisée par Lavoie par l'intermédiaire de la librairie *NURBS++* [Lav].

De plus, la géométrie produite sur la courbe, particulièrement les points de contrôle, est grandement simplifiée et procure souvent des résultats ne nécessitant aucune modification. Nous avons donc une technique robuste pour générer des courbes de type *NURBS* à partir de points dessinés par un artiste. Nous devons maintenant produire des mèches à partir d'un ensemble de traits.

4.2.2 Génération de *patches*

Notre but est toujours de simplifier au maximum le travail que l'artiste aura à effectuer pour générer des chevelures. Nous avons choisi de représenter des mèches de cheveux par des *patches* de *NURBS* et désirons maintenant les générer à partir de traits produits par l'artiste.

Piegl et Tiller [PT95] proposent d'utiliser une opération de *skinning* qui permet de produire une *patch* à partir d'un ensemble $\mathcal{C}$ de m courbes :

$$\mathcal{C} = \left\{ C_k \right\}, \quad 0 \leq k \leq m-1.$$

Il y a cependant certaines contraintes :

- le nombre de points de contrôle doit être le même pour chaque courbe ;
- les valeurs des *knots* doivent être les mêmes pour chaque courbe.

Néanmoins, il existe des techniques pour changer le nombre de points de contrôle et la valeur des *knots* sans changer l'apparence des courbes, permettant de joindre ensemble des courbes qui pouvaient être incompatibles en premier lieu. Un exemple de *skinning* est illustré à la Fig. 4.4.

Le nombre de points de contrôle dans un sens de la *patch* est défini par les courbes de $\mathcal{C}$, mais une résolution doit être définie dans l'autre sens également. La solution utilisée dans ce travail est de prendre la grandeur de $\mathcal{C}$ comme résolution. Ensuite, la matrice de géométrie (ou les matrices, s'il s'agit d'une *patch* à plusieurs segments) est générée à partir des points de contrôle ainsi produits.

Dans notre situation, nous désirons permettre à l'usager de générer une *patch* à partir de deux courbes. Cependant, deux courbes ne sont pas suffisantes pour permettre une opération de *skinning*. En réalité, une telle opération serait possible, mais ne permettrait

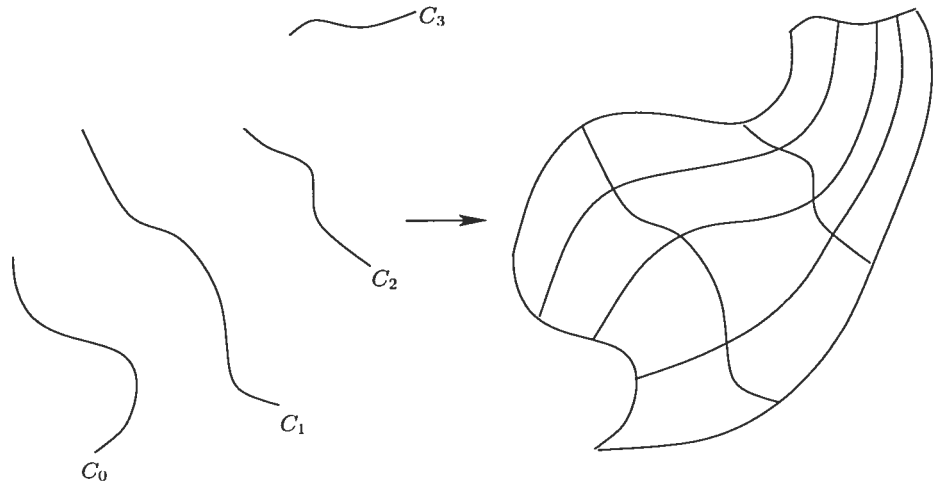


FIG. 4.4 – Opération de *skinning* sur un ensemble de courbes *NURBS*.

aucune modification dans le sens longitudinal de la surface.[¶] L'usager qui dessine des mèches de cheveux devrait pouvoir faire des modifications sur la surface comme il le désire. Pour permettre cette flexibilité, nous allons générer automatiquement des courbes supplémentaires pour permettre un *skin* sur quatre courbes. Le processus de création des deux courbes supplémentaires est illustré à la Fig. 4.5. En (a), des vecteurs $\vec{v}$ sont

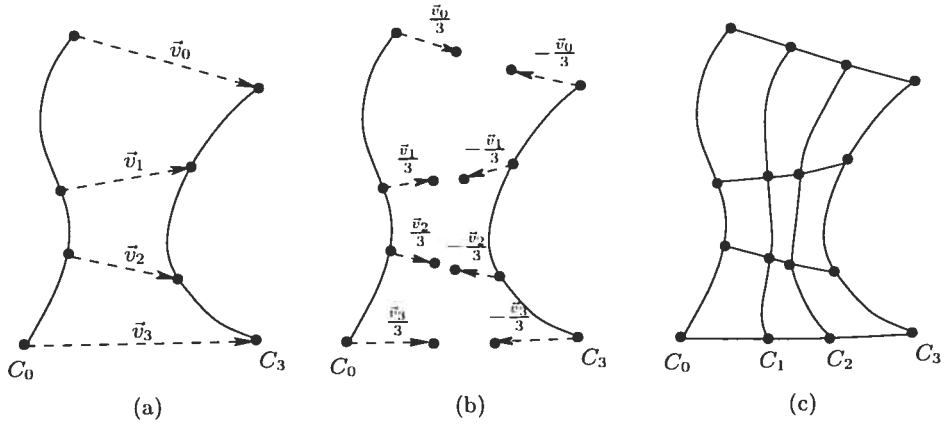


FIG. 4.5 – Création de deux courbes supplémentaires pour le *skinning*.

[¶]Un *skin* avec deux courbes s'appelle une *ruled surface*. Il s'agit essentiellement de deux courbes paramétriques reliées ensemble par des traits linéaires. Une opération de *skinning* à plus de deux courbes est une généralisation des *ruled surfaces*.

produits à partir des points de contrôle de la première courbe C_0 vers les points de contrôle de la deuxième courbe C_3. En (b), ces vecteurs sont réduits aux tiers de leurs longueurs afin de produire une nouvelle série de points. Les vecteurs sont également inversés pour produire une deuxième nouvelle série à partir de la deuxième courbe C_3. En (c), les nouvelles courbes C_1 et C_2 sont produites. Cette approche s'est démontrée suffisamment flexible dans la majorité des cas et permet la création simple de *patches* avec une opération de *skinning* tout en conservant un degré de liberté dans la largeur de la *patch*.

4.2.3 Outils de modélisation

La *patch* générée à partir des traits ne sera pas toujours placée comme le souhaiterait l'artiste. Quelques retouches supplémentaires sont parfois nécessaires et les courbes définissant la largeur doivent être modifiées *a posteriori* puisque les courbes intermédiaires sont interpolées linéairement à partir des traits. Pour ces raisons, il est nécessaire de fournir des outils de modélisation à l'usager. Fournir de tels outils permet également à l'usager de définir l'apparence d'une *patch* sans utiliser le système de traits décrit plus tôt, mais de modifier une *patch* de base déjà existante. Nous voulons essentiellement fournir les outils suivants :

- permettre le changement de position de groupes de points ;
- permettre des transformations (rotations, mises-à-échelle) sur les points ;
- permettre des modifications de haut niveau et de bas niveau.

Il est possible d'intégrer tous ces outils d'une manière élégante et intuitive. L'usager peut, par exemple, sélectionner un sous-ensemble des points de contrôle de la *patch*, choisir une transformation à appliquer sur ces points et appliquer ces transformations. De cette manière, l'usager pourra déformer la surface selon ses goûts. Néanmoins, cette approche demeure locale : modifier l'apparence globale d'une *patch* de plusieurs points de contrôle demeure difficile et il faudrait proposer une solution à de tels problèmes.

Une des premières solutions au problème de déformation d'objets complexes est le *free form deformation* (FFD [Bar84]). Cette approche consiste à envelopper l'objet à déformer d'un cube paramétrique (ou d'une surface paramétrique, en 2D) et d'associer

la géométrie à l'intérieur de ce cube. Les déformations appliquées sur le cube se réfléteront sur l'objet par l'intermédiaire de cette association. Par exemple, considérons la transformation d'un point unique qui se trouve en plein centre du cube paramétrique. Dans l'espace du cube, le point se trouve à la position $(0.5, 0.5)$ et conservera toujours cette position. La modification du cube change la position de son centre dans l'espace monde, et modifiera la position du point dans l'espace monde également (qui suit toujours la coordonnée $(0.5, 0.5)$ de l'espace cube). Dans notre cas, nous n'avons pas besoin d'un cube paramétrique mais plutôt d'une surface paramétrique puisque nos points de contrôle sont pour l'instant planaires. Nous utiliserons une surface de *Bézier* pour englober nos points. Il s'agit d'une décision arbitraire principalement basée sur la facilité d'implémentation de ces surfaces.

Un problème potentiel qui pourrait survenir est la détermination des coordonnées originale dans l'espace de la surface. Effectivement, déterminer les coordonnées (s, t) d'une *patch* à partir des coordonnées (x, y) de l'espace monde peut être difficile à résoudre. Cependant, nous n'avons pas ce problème puisque nous pouvons supposer notre *patch* originale comme étant rectangulaire : les coordonnées (s, t) ne sont alors qu'une interpolation linéaire à l'intérieur de ce rectangle. Les illustrations de la Fig. 4.6 montrent plus en détail ce processus. Pour déformer nos surfaces, nous allons associer les points

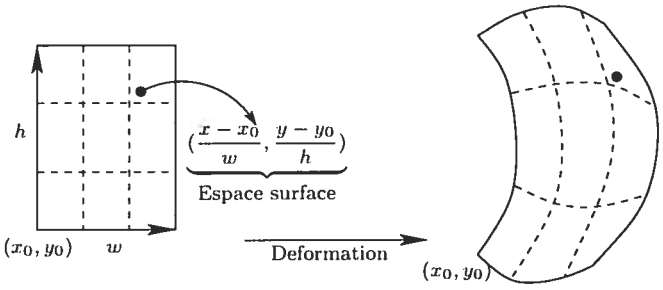


FIG. 4.6 – Déformation par FFD d'un point.

de contrôle de celles-ci uniquement. Ainsi, la déformation de la *patch* FFD modifiera les points de contrôle de la surface *NURBS*. Afin de conserver une certaine généralité, nous pouvons permettre d'effectuer un FFD sur un sous-ensemble de ces points de contrôle.

Tous les outils présentés dans ce chapitre permettent la création rapide de surfaces avec une technique similaire à celle utilisée par les artistes. Ils permettent également la modification générique de ces surfaces pour avoir un niveau de contrôle supplémentaire. Cependant, créer des surfaces n'est que la première étape dans la génération de mèches de cheveux : il faut maintenant les rendre de manière à leur donner un aspect de chevelure.

Chapitre 5

Technique de rendu

Make everything as simple as possible, but not simpler.

Albert Einstein

Il existe plusieurs techniques pour générer de la géométrie à partir d'une surface. La méthode généralement utilisée est de produire un polygone à chaque paramètre (s, t) évalué sur la surface. Par la suite, il est facile d'y apposer une texture ou d'effectuer un *shading* quelconque. Cependant, notre but est de produire des chevelures cohérentes dans un contexte de dessins animés et plus particulièrement de produire un effet de *feathering* (Section 3.2) de bonne qualité. Ce chapitre aura pour but d'expliquer une solution possible de rendu de *feathering* basée sur la structure des surfaces paramétriques *NURBS*.

Lorsqu'on parle de rendu de *feathering*, nous devons tout d'abord savoir où il apparaît sur la surface. Le *feathering* est en fait une technique de rendu en demi-ton [Kan99, LA01, Uli87] où différents tons de gris sont créés avec uniquement deux couleurs (typiquement noir et blanc). Le rendu en demi-ton tente généralement de reproduire des images en ton continu avec ces deux couleurs en utilisant l'intensité des images sources. Notre situation est cependant différente puisque nous ne désirons pas convertir une image

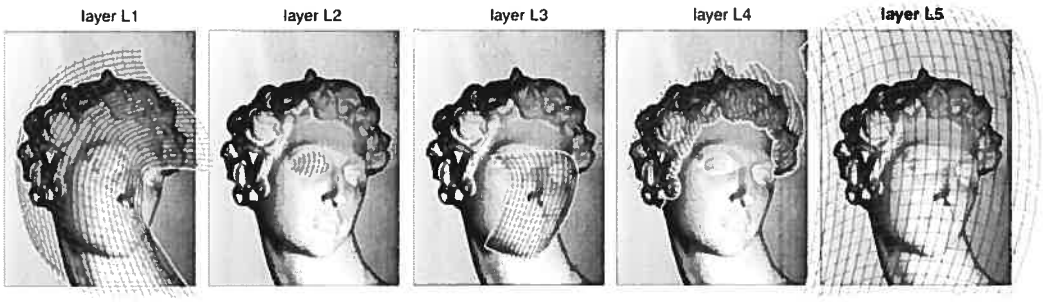


FIG. 5.1 – Simplification de modèles pour gravures à partir de *patches*. Images tirées de Ostromoukhov [Ost99].

en ton continu, mais plutôt générer un demi-ton selon des propriétés externes, comme l'éclairage par exemple.

Une solution à un problème similaire est proposée par Ostromoukhov [Ost99] qui suggère une méthode de rendu pour les gravures, originalement faites à la main. Sa méthode utilise également les surfaces paramétriques pour générer les gravures ; nous devrions donc pouvoir nous en inspirer dans notre technique. L'idée générale de sa méthode est de placer des *patches* au-dessus de l'image avec laquelle nous désirons produire une gravure, tout en épousant les formes de celle-ci. L'idée est de simplifier la structure complexe du modèle sous-jacent avec quelques *patches* simples, permettant ainsi de simplifier la modélisation et le rendu (Fig. 5.1). L'intensité de la portion de l'image sous-jacente à la *patch* sera analysée et des traits (*hatches*) seront produits à partir de celle-ci. Les traits auront une largeur qui varie en fonction de l'intensité de l'image, reproduisant ainsi le même ton que l'image originale. Les résultats obtenus par cette technique sont forts impressionnants (illustration à droite, tirée de [Ost99]) et nous fournissent des images inspirantes pour le rendu de chevelures. Particulièrement, le rendu de gravures ressemble sensiblement au rendu de *feathering* puisqu'ils sont tous deux basés sur la variation de largeur de traits pour générer différents tons. La différence majeure est la source de l'intensité. Dans le *framework* que nous avons proposé (Section 3.3.2), nous avons décidé de produire les chevelures à partir d'un dessin typiquement fait au plomb par un artiste, ne nous laissant aucune

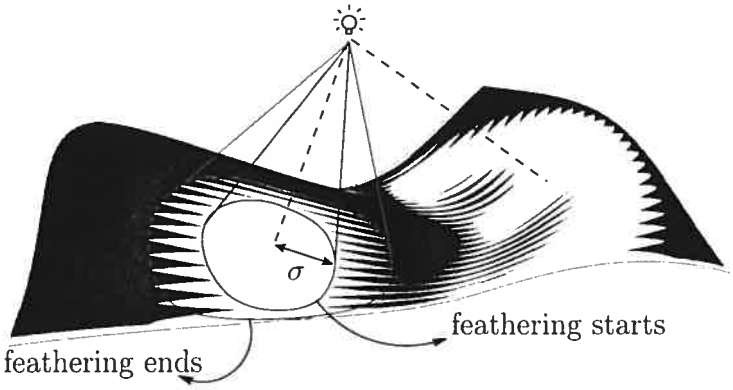


FIG. 5.2 – Position des reflets par rapport à une pseudo-lumière.

intensité de gris nous permettant de déduire la largeur des traits. Nous devons proposer une autre solution à ce sujet.

Le *feathering* est utilisé pour simuler les reflets (*highlights*) sur les surfaces. Les artistes les positionnent intuitivement sur les images, et les résultats ainsi obtenus sont généralement visuellement satisfaisants. Nous allons donc faire confiance à l'artiste en ce qui concerne le placement des reflets. Ce que nous allons faire, c'est fournir un outil pour leur permettre de spécifier une région dans laquelle un reflet sera visible. Pour décrire une telle région, nous allons définir une nouvelle entité : une *pseudo-lumière* (*pseudo-light*). Les pseudo-lumières seront des sources génératrices de reflets qui seront utilisées au courant de ce chapitre. La Fig. 5.2 montre l'endroit où apparaîtront les reflets par rapport à une pseudo-lumière donnée. Essentiellement, le *feathering* apparaîtra entre les reflets et les régions aucunement éclairées (totalement noires). Évidemment, la taille des reflets devra être paramétrée afin d'offrir une plus grande flexibilité à l'artiste. Le reste de ce chapitre sera utilisé pour décrire ce processus en détail.

Définitions

Notre but étant de produire des *feathers* à partir d'une pseudo-lumière, nous devons trouver un moyen de dessiner des traits de largeurs variables en utilisant la représentation paramétrique des surfaces décrites au chapitre précédent. Pour y arriver, nous avons

besoin de définir quelques éléments qui nous seront utiles. Une *patch* utilisée pour le rendu sera représentée par la lettre caligraphique $\mathcal{P}$, et ses indices représentent des coordonnées sur la surface, par exemple $\mathcal{P}_{s,t}$, où s est la dimension de largeur et t est la dimension de longueur. Si les valeurs de s et t sont fixées, $\mathcal{P}_{s,t}$ représente alors un point sur la surface. Les surfaces seront toujours des *NURBS* et auront une taille arbitraire (en terme de points de contrôles). Un vecteur $\vec{l}_{s,t}$ part du point $\mathcal{P}_{s,t}$ vers la pseudo-lumière. Également, le vecteur $\vec{n}_{s,t}$ est la normale au point $\mathcal{P}_{s,t}$. Tous les vecteurs sont considérés normalisés.

5.1 Génération de traits

Pour permettre le rendu avec OpenGL, il nous faut utiliser une approche polygonale pour dessiner les traits. Ceci peut être facilement effectué en générant une paire de quadrilatéraux à chaque point de la *patch* le long de la coordonnée t. Pour simuler un effet de *feathering*, nous devons trouver un moyen de produire des traits sur la *patch* avec une fonction de largeur $w(s,t)$ qui varie pour donner une impression de reflet généré par la pseudo-lumière. Intuitivement, nous pourrions opter pour une fonction du style $\vec{n}_{s,t} \cdot \vec{l}_{s,t}$ pour évaluer l'angle de la pseudo-lumière par rapport au point $\mathcal{P}_{s,t}$. Cette fonction est effectivement très utile dans ce contexte, mais il faut plutôt utiliser la fonction inverse pour évaluer la largeur du trait. Par exemple, lorsqu'un point $\mathcal{P}_{s,t}$ se trouve directement sous la pseudo-lumière, les vecteurs $\vec{n}_{s,t}$ et $\vec{l}_{s,t}$ sont les mêmes, et donc $\vec{n}_{s,t} \cdot \vec{l}_{s,t} = 1$. Or, à cet endroit, nous désirons une largeur de trait nulle pour éclaircir la région. Nous allons donc utiliser la fonction $w(s,t) = 1 - \vec{n}_{s,t} \cdot \vec{l}_{s,t}$ pour connaître l'épaisseur du trait (en pourcentage) à la coordonnée (s,t) sur la *patch*.

Nous devons maintenant déterminer la largeur absolue du trait à partir du pourcentage $w(s,t)$. Pour y arriver, nous allons utiliser la valeur $w(s,t)$ par rapport au trait voisin. Pour éviter la superposition des traits, nous allons plutôt les restraindre à mi-chemin. Ceci permettra aux traits voisins de se joindre ensemble sans se toucher, et ainsi former une surface uniforme*. La Fig. 5.3 montre comment les polygones sont générés

*Cette approche permet également d'éviter un problème courant avec OpenGL appelé le *z-fighting*, où les polygones à la même profondeur tentent de s'afficher simultanément.

pour les traits. En (a), les lignes représentent des traits non-rendus (il s'agit de lignes

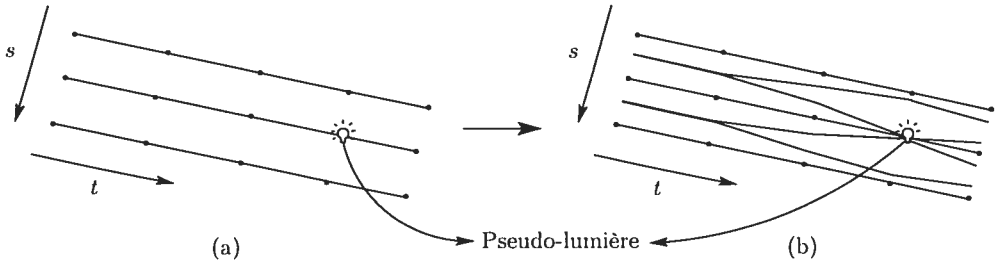


FIG. 5.3 – Génération de polygones pour les traits à partir d'une pseudo-lumière.

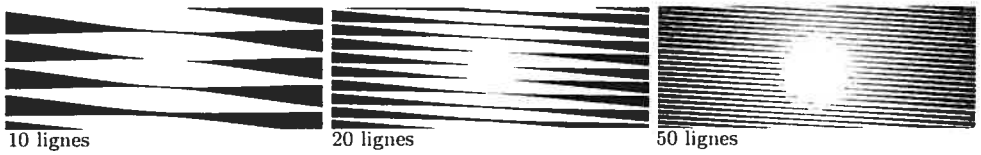
paramétriques sur la surface, choisies à des intervalles réguliers sur le paramètre s). En (b), la géométrie est produite en utilisant la fonction

$$\vec{m}_{s,t} = \frac{1}{2}\vec{v}_{s,t}w(s,t),$$

où $\vec{v}_{s,t} = \mathcal{P}_{s-1,t} - \mathcal{P}_{s,t}$ (*i.e.* un vecteur vers la courbe voisine). Évidemment, il y certains paramètres configurables. Ils sont clairement identifiables sur la Fig. 5.3(b) :

1. le nombre de lignes (nombre d'intervalles sur s) ;
2. la vitesse de croissance des traits ;
3. le décalage par rapport à la pseudo-lumière.

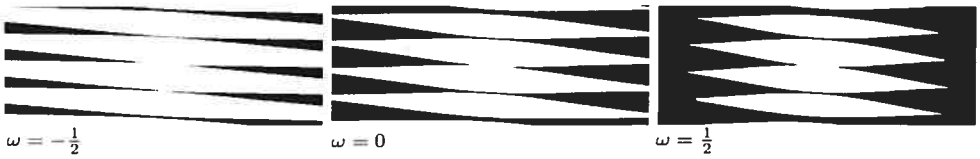
Le premier paramètre modifiable est plus évident. Augmenter le nombre d'intervalles sur la dimension s de la *patch* augmentera inévitablement le nombre de traits. Il y a tout de même une subtilité intéressante à ce sujet : augmenter le nombre de lignes diminue la taille des traits du même coup. Ceci est causé par le fait que la largeur des traits est dépendante de la distance du trait voisin (le vecteur $\vec{v}_{s,t}$ est calculé par rapport à la ligne voisine). Ceci permet également de conserver un ton constant sur la surface, indépendamment du nombre de lignes : propriété intéressante dans notre contexte. L'illustration suivante montre comment le nombre de lignes influence l'apparence des surfaces.



Le second paramètre modifiable est la vitesse à laquelle la largeur des traits augmente. Si la largeur augmente plus rapidement, les traits voisins se rejoindront plus rapidement et formeront des surfaces noires uniformes plus grandes. Nous nommerons ce paramètre ω, et il sera utilisé pour la nouvelle version du calcul de la largeur des traits :

$$\vec{m}_{s,t} = \frac{1}{2}\vec{v}_{s,t}w(s,t)\omega.$$

L'effet de la variation de ω est illustré ici :



Le dernier paramètre configurable est le décalage (*offset*) par rapport à la pseudo-lumière. Ce paramètre permet d'agrandir la région claire produite par la pseudo-lumière. Essentiellement, nous allons soustraire une valeur σ de $w(s,t)$ (tout en s'assurant que le résultat soit plus grand que zéro). Ainsi, l'effet du *feathering* sera retardé par cette soustraction. La nouvelle (et dernière) version de la formule de calcul de la largeur des traits devient donc :

$$\vec{m}_{s,t} = \frac{1}{2}\vec{v}_{s,t}\max(0, w(s,t) - \sigma)\omega.$$

L'effet de ce dernier paramètre est démontré sur les illustrations suivantes :

L'utilisation de ces trois paramètres simples donne une grande flexibilité au style de rendu des surfaces (Fig. 5.4). Cependant, nous supposons toujours que les surfaces sont planaires, créées à partir de traits 2D. Cette supposition est très contraingnante dans notre contexte puisque les paramètres proposés jusqu'ici ne permettent pas d'éliminer l'aspect planaire des surfaces. Cette apparence est plutôt gênante dans plusieurs cas, et nous aimerions limiter son effet. La prochaine section proposera une solution simple à ce problème.

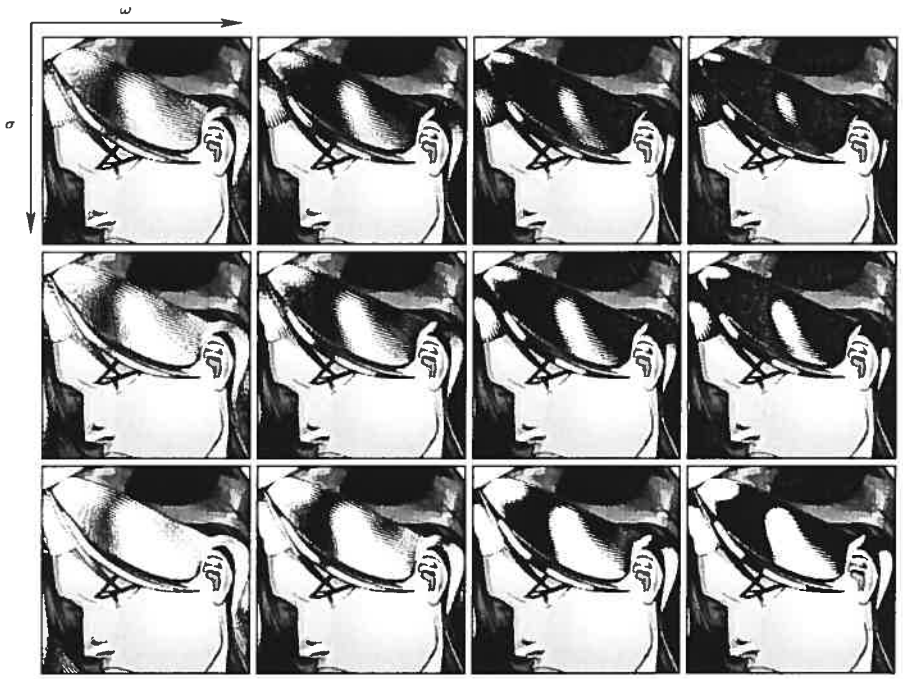


FIG. 5.4 – Effets des paramètres σ et ω. Image originale réalisée par Ed Benes et Vince Russell.

5.2 Placement des pseudo-lumières

Le placement des pseudo-lumières est un autre problème à résoudre. Effectivement, le placement de sources lumineuses est généralement considéré comme un problème complexe à résoudre en infographie photoréaliste. Cependant, le problème dans notre situation est considérablement réduit puisque les sources lumineuses (les pseudo-lumières) ont un impact très local sur les surfaces.

Ce que nous désirons faire, c'est permettre à l'usager de déterminer une région (toujours à l'aide de son *input device*) qu'il désire illuminer, et placer la pseudo-lumière automatiquement pour produire l'effet désiré. Une solution générale à ce problème serait évidemment très difficile à trouver puisqu'il existe plusieurs configurations différentes de pseudo-lumières qui permettent d'obtenir un résultat donné, mais nous allons simplifier le problème pour le rendre plus facilement réalisable.

Pour y arriver, nous allons restreindre la forme de la région spécifiée par l'usager à un simple cercle. Le rayon de ce cercle ainsi que la position de son centre nous seront suffisants pour déterminer (approximativement) la position d'une pseudo-lumière permettant de fournir un reflet dont la taille correspond à ce cercle.

Essentiellement, nous allons utiliser le rayon fourni par l'usager pour la distance (en z) de la pseudo-lumière. Cependant, nous devons tenir compte de la vitesse de croissance des traits (ω) puisqu'une croissance plus faible va générer un reflet plus large. Nous devons également tenir compte du décalage (σ) puisqu'un décalage plus grand créera un reflet plus large. Donc, la distance de la pseudo-lumière par rapport au centre du cercle variera en fonction de $r \cdot \frac{\sigma}{\omega}$, où r est le rayon du cercle fourni par l'usager. Ce processus est illustré sur la Fig. 5.5(a) - 5.5(b).

Un autre détail à considérer est le déplacement de multiples sources de pseudo-lumières. Ce détail est particulièrement important lorsque nous discuterons de l'animation des chevelures (Chapitre 6). Étant donné une configuration initiale de pseudo-lumières positionnées autour d'une surface, existe-t-il un moyen efficace de les déplacer simultanément de manière cohérente ? L'interface suggérée permet une telle manipulation par l'intermédiaire d'un contrôle maître. Essentiellement, le déplacement du contrôle maître déplacera les pseudo-lumières sous-jacentes relativement à leurs positions ini-

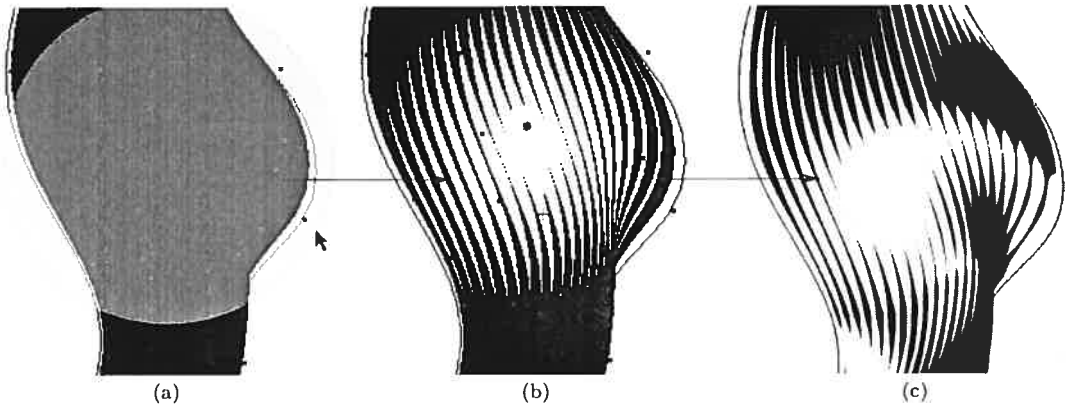


FIG. 5.5 – Processus de placement d'une pseudo-lumière et de perturbations.

tiales, tel qu'illustré sur la Fig. 5.6. Pour y arriver, il suffit de créer des vecteurs partant du contrôle maître vers les points de contrôle et s'arrêtant sur la surface. Les positions auxquelles se trouvent les points de contrôle sur les vecteurs sont sauvegardés. Lorsque le contrôle maître est déplacé, les vecteurs sont regénérés de la nouvelle position du contrôle maître vers les positions sur la surface déjà trouvées. Les points de contrôles sont alors repositionnés sur les nouveaux vecteurs aux mêmes positions (paramètres) que ceux sauvegardés précédemment. Cette technique permet d'introduire un mouvement cohérents aux multiples source de pseudo-lumières.

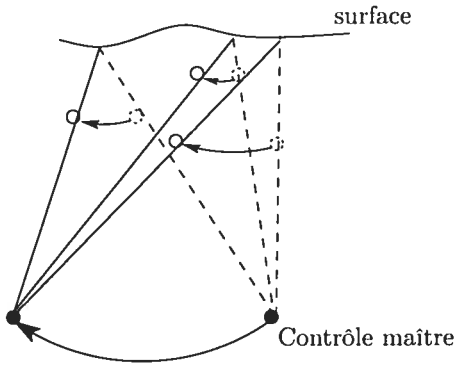


FIG. 5.6 – Déplacement de multiples sources de pseudo-lumières.

5.3 Amélioration de l'apparence des surfaces

Nous aimerions permettre à l'artiste de donner l'apparence voulue à la surface. Cependant, l'aspect planaire de celle-ci est limitative : les paramètres décrits à la section précédente ne sont pas suffisants pour représenter des styles arbitraires. Pour y arriver, nous allons permettre à l'usager de perturber la géométrie de la surface selon la profondeur, détruisant ainsi l'aspect planaire de ces dernières. Évidemment, le problème qui réside est comment effectuer ces perturbations simplement ?

La première solution qui nous vient à l'esprit est de permettre une rotation des surfaces pour pouvoir modifier la profondeur des points de contrôles. Cette approche est un échec à tous les points de vues : nous ne voulons pas exposer l'aspect 3D des surfaces à l'usager, mais toujours donner l'impression que le dessin est 2D, comme sur une table à dessin. De plus, l'effet visuel produit par les perturbations avec cette approche ne donne aucun *feedback* avant que l'usager annule la rotation. Il faut donc trouver une autre solution.

L'approche proposée dans ce travail règle les deux problèmes mentionnés ci-haut : elle n'expose pas la structure 3D des surfaces et permet un *feedback* immédiat des perturbations produites. Essentiellement, nous permettons à l'usager de perturber la surface aux endroits où il clique avec son périphérique d'entrée : plus il reste longtemps au même endroit, plus la perturbation sera grande. L'usager peut ainsi voir l'effet des perturbations en temps réel avec un simple clique, ce qui est souhaitable dans notre contexte.

Cependant, cette approche comporte également quelques problèmes. Tout d'abord, bien que la structure 3D reste cachée sous l'interface proposée, il n'en demeure pas moins que l'effet produit n'est pas toujours intuitif par rapport aux mouvements de l'usager. Par exemple, la perturbation peut enfoncer ou élever la surface (*i.e.*, perturber en $z+$ ou en $z-$), et nous devons laisser à l'usager le soin de décider dans quel sens il désire faire les modifications. Donc, l'usager doit être conscient que les perturbations seront effectuées "en profondeur". Néanmoins, puisque les changements effectués sont visibles immédiatement, l'expérience avec l'interface a montré que cette approche est suffisamment simple d'utilisation. La Fig. 5.5(c) montre un exemple de perturbation effectuée sur une surface ainsi que l'impact visuel engendré par cette technique.

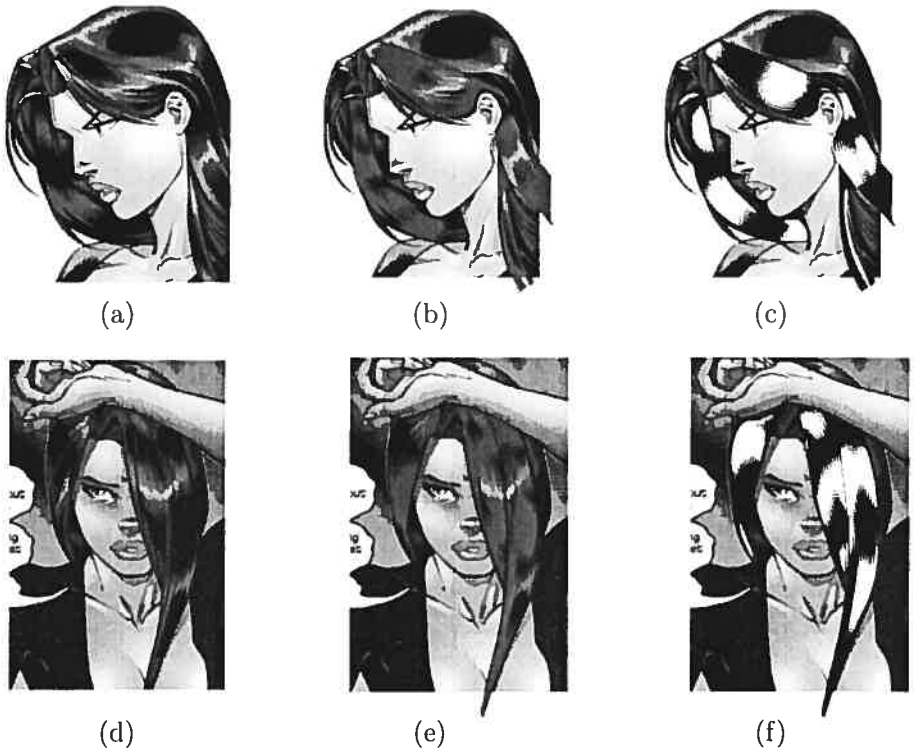


FIG. 5.7 – Chevelures produites sur le personnage de *Rainmaker*. Image originale réalisée par Ed Benes et Vince Russell.

5.4 Résultats

La première tentative de production de chevelures a été effectuée sur l'image de la Fig. 5.7(a). Cette image est la source même de l'inspiration de la technique proposée dans ce chapitre, les reflets et le *feathering* étant particulièrement visibles sur cette image. La Fig. 5.7(b) montre le placement des surfaces ainsi que les reflets produits par les pseudo-lumières. Finalement, la Fig. 5.7(c) montre le résultat final avec *shading*. Un résultat similaire fut obtenu sur le même personnage à la Fig. 5.7(d)-5.7(f).

Ces deux résultats avaient pour but de démontrer la validité de la technique de rendu. Cependant, le but est de placer les chevelures sur des dessis faits au crayon de plomb, question de faciliter l'encrage de ceux-ci. Le personnage de *Radella* offre une bonne image de base pour produire l'encrage. Les résultats obtenus avec notre approche, ainsi que

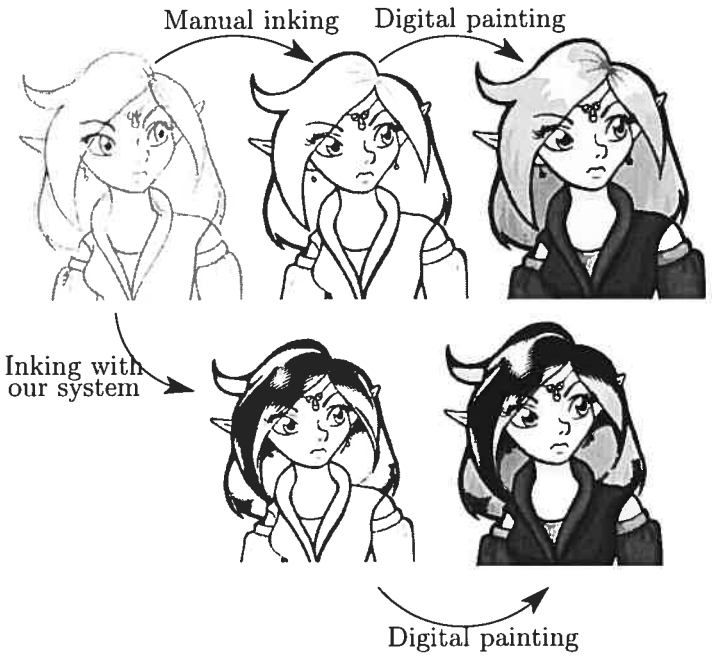


FIG. 5.8 – Chevelures et coloration produites sur le personnage de *Radella*. Image originale tirée de [Tae].

ceux produits manuellement par un artiste, sont illustrés à la Fig. 5.8. Cette illustration montre la différence entre une chevelure produite manuellement et une chevelure produite par notre système. Bien que le style soit différent, il n'en demeure pas moins que le résultat de notre technique est très expressif et donne un tout nouvel aspect à l'image. Des résultats supplémentaires sont également illustrés à la Fig. 5.9.

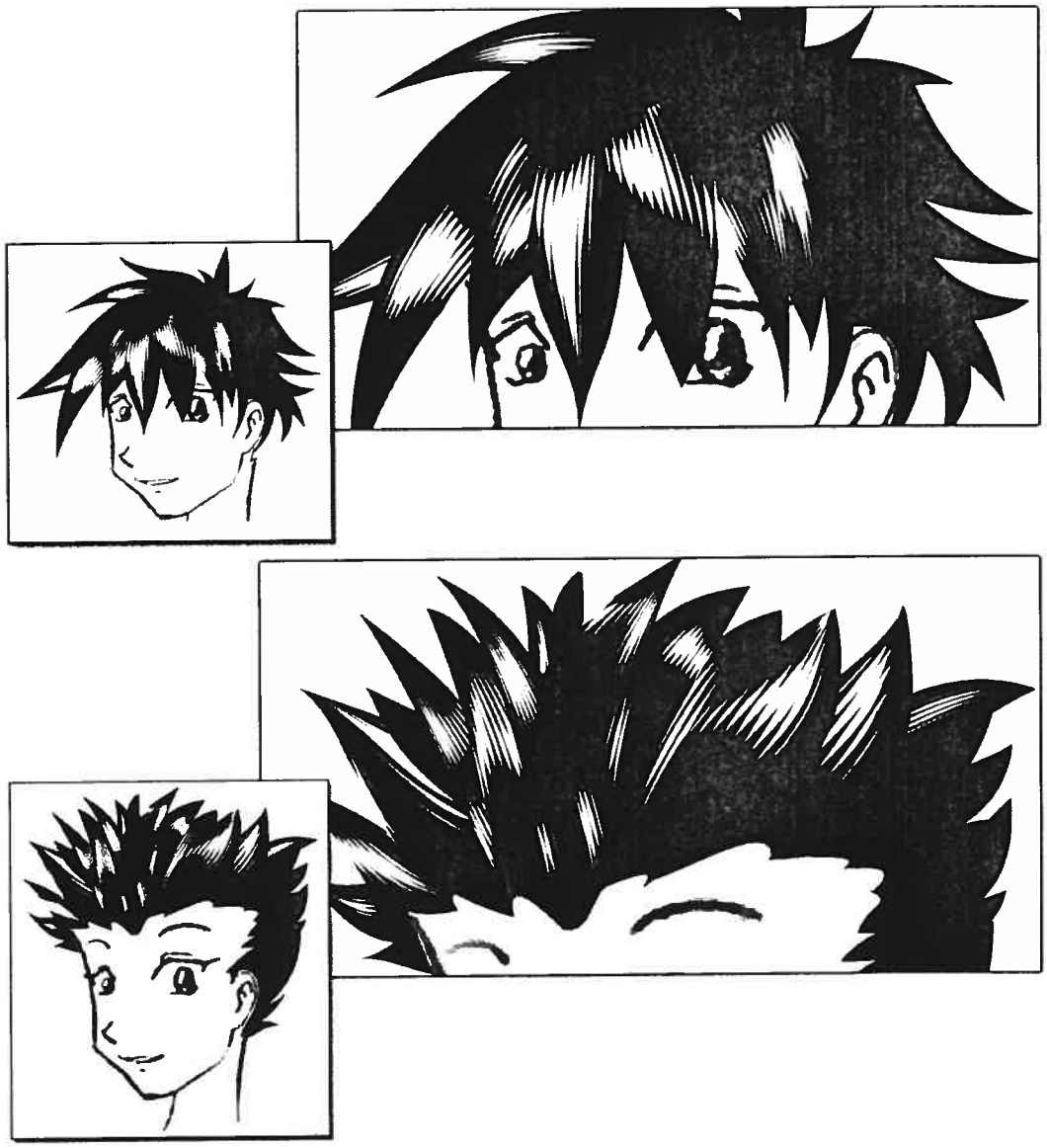


FIG. 5.9 – Autres résultats obtenus avec notre technique. Les images originales sont tirée de [bak].

Chapitre 6

Animation

Never confuse movement with action.

Ernest Hemingway

L'animation cohérente des chevelures produites lors du chapitre précédent est probablement le but ultime à atteindre dans notre contexte. Effectivement, produire des animations de bonne qualité est très difficile dans l'industrie, particulièrement à cause des chevelures (Chapitre 3). Ce chapitre propose une solution simple pour animer les mèches de cheveux. Tel qu'expliqué dans les chapitres précédents, le modèle a été conçu dans l'espoir d'y intégrer des mouvements facilement. Nous verrons si cet objectif est atteint. Essentiellement, nous allons tenter de reproduire automatiquement les images intermédiaires produites normalement à la main dans l'industrie.

6.1 Animation dans les logiciels professionnels

Les logiciels modernes d'animation (tels que *XSI, Maya* ou *3D Studio Max*) procurent de multiples outils d'animation. Le point commun entre tous ces logiciels est leur capacité d'interpoler des images clés (*keyframes*) de manière à produire un mouvement fluide et cohérent. L'interpolation est discutée avec plus de détails à la Section 6.2. Les logiciels

fournissent souvent d'autres modèles d'animation basés sur la physique. Par exemple, un objet se déplaçant rapidement pourra être déformé automatiquement par le logiciel puisque l'accélération produira une force à chaque sommet du modèle, produisant ainsi une déformation. Inversement, il est parfois possible de spécifier des forces sur le modèle, et le logiciel produira le mouvement causé par cette force.

Certains outils sont quelque fois fournis pour l'animation de chevelures. En fait, plusieurs logiciels professionnels permettent l'ajout de chevelures sur un modèle. Ces mêmes logiciels permettront également de produire des mouvements automatiquement sur ces cheveux, laissant relativement peu de flexibilité à l'utilisateur.

Notre but dans ce chapitre n'est pas de reproduire les fonctionnalités de ces logiciels, mais plutôt de démontrer qu'une approche simplifiée d'interpolation permet de fournir une approximation adéquate à partir de notre modèle de chevelures, et que cette même solution pourrait être intégrée facilement dans ces logiciels.

6.2 Interpolation des *keyframes*

La première solution proposée pour "automatiser" le processus d'animation est d'interpoler les images clés (*keyframes*), *i.e.* produire automatiquement des images intermédiaires entre ces *keyframes*. Ces images intermédiaires (*in-between images*) sont généralement produites manuellement dans l'industrie par des *in-betweeners*. Ce travail est long et fastidieux. Nous allons donc proposer une solution permettant de l'éliminer complètement, tout du moins en ce qui concerne l'animation des chevelures.

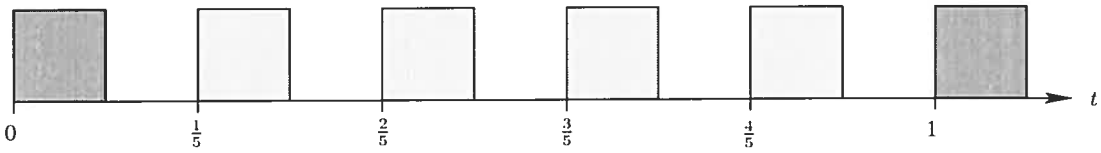
Le calcul des images intermédiaires se fait par interpolation. Nous désirons créer des images intermédiaires des surfaces *NURBS* ainsi que des pseudo-lumières. Pour simplifier, nous calculons uniquement l'interpolation des points de contrôles. Cette approche donne aussi de meilleurs résultats puisque le mouvement des *patches* paraîtra plus naturel qu'en interpolant la géométrie directement.

6.2.1 Interpolation linéaire

La manière la plus simple de calculer la position $\vec{p}$ d'un vecteur au temps t entre les vecteurs $\vec{p_0}$ et $\vec{p_1}$ est par interpolation linéaire. Nous supposons t compris entre 0 et 1 (inclusivement). Donc, $\vec{p} = \vec{p_0}$ lorsque $t = 0$, $\vec{p} = \vec{p_1}$ lorsque $t = 1$, $\vec{p}$ est à mi-chemin entre $\vec{p_0}$ et $\vec{p_1}$ lorsque $t = \frac{1}{2}$, ainsi de suite. Le calcul est effectué de la manière suivante :

$$\vec{p}(t) = \vec{p_0}(1 - t) + \vec{p_1}t.$$

La notation $\vec{p}(t)$ a été utilisée ici puisque la valeur de $\vec{p}$ est une fonction de t, évidemment. L'illustration suivante montre comment varie une surface simple entre $t = 0$ et $t = 1$ (les surfaces plus pâles sont les positions interpolées).

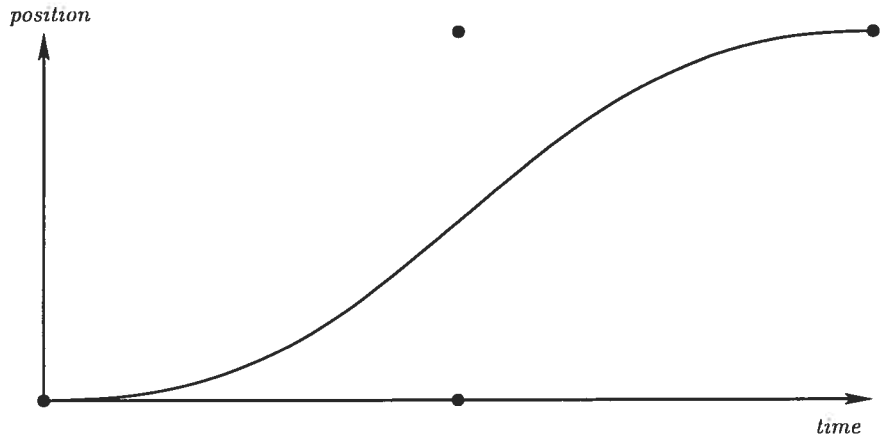


Il semble évident qu'une telle interpolation ne donnera pas un mouvement naturel aux mèches de cheveux puisque celles-ci subissent des accélérations et des décélérations plus subtiles. Pour arriver à des interpolations plus flexibles, nous allons plutôt utiliser une interpolation par *spline*.

6.2.2 Interpolation par *spline*

Une interpolation par *spline* utilise une courbe paramétrique (dans notre situation, une courbe de *Bézier*) pour déterminer la position par rapport au temps t. Il s'agit bêtement d'une fonction continue et monotone croissante*, tel qu'illustré sur la figure suivante.

*Il faut faire attention dans le choix du type de la courbe paramétrique pour que la fonction demeure toujours continue et monotone croissante en tout temps. Les courbes de *Bézier* satisfont ces contraintes pour autant que les deux points de contrôles intermédiaires soient compris dans l'enveloppe convexe décrite par les deux points de contrôles aux extrémités de la courbe.



La position et le temps sont toujours compris entre 0 et 1. Maintenant, au lieu d'utiliser le temps (qui varie linéairement) pour calculer l'interpolation, nous utiliserons la fonction décrite par la courbe. Évidemment, l'usager peut modifier la courbe (par les points de contrôles) afin d'utiliser une fonction d'interpolation différente. La fonction suivante fait le calcul voulu sur un vecteur $\vec{p}$:

$$\vec{p}(t) = \vec{p}_0(1 - \mathcal{F}(t)) + \vec{p}_1\mathcal{F}(t),$$

où $\mathcal{F}(t)$ est la fonction dépendante de t décrite par la courbe paramétrique. Le problème est maintenant de trouver quelles fonctions utiliser et dans quels contextes. Les illustrations de la Fig. 6.1 montrent des exemples de fonctions, ainsi que l'interpolation produite sur une surface simple par rapport au temps. Il y a quatre exemples : linéaire, accélération/décélération, accélération, décélération et décélération/accélération. Cette dernière illustration nous permet plus facilement de choisir des interpolations appropriées pour l'animation de chevelures. En particulier, la fonction accélération/décélération est efficace pour donner une impression naturelle de mouvements, représentant fidèlement les accélérations et décélérations subtiles des cheveux.

Cependant, les artistes produisant les *keyframes* de base devant être interpolés n'ont pas toujours l'intention d'accélérer et de décélérer entre chaque *keyframe*. Il ne sera pas rare de vouloir mettre un *keyframe* intermédiaire à l'animation tout en poursuivant une accélération. Les fonctions indépendantes d'accélération et de décélération permettront un tel effet.

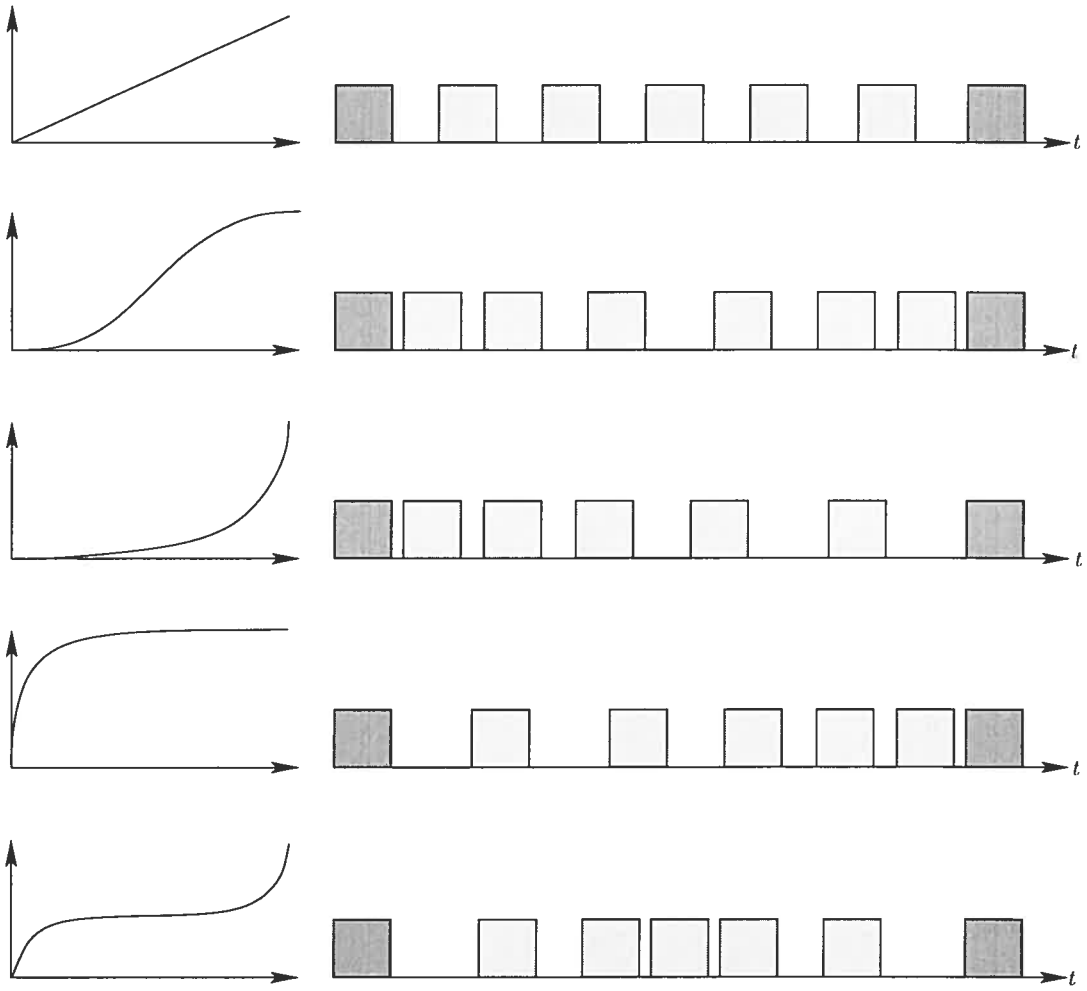


FIG. 6.1 – Différentes interpolations par *spline*.

6.2.3 Résultats

Les animations obtenues par interpolation sont difficiles à décrire textuellement. Quelques images représentant les surfaces interpolées sont donc affichées pour donner une idée du résultat de l'animation, mais des fichiers vidéos sont disponibles sur internet[†].

Le premier résultat obtenu est illustré à la Fig. 6.2. Il s'agit d'une simple mèche de cheveux qui tombe, réalisée à partir de quatre *keyframes*. L'interpolation utilisée entre chaque *keyframe* est du type accélération/décélération. Quelques positions interpolées sont illustrées sur la figure.

FIG. 6.2 – Mèches interpolées d'une animation simple.

Le deuxième groupe de résultats produits a été effectué sur des têtes entières, notre but ultime étant de permettre des animations sur des modèles complets, et non pas sur des mèches individuelles. Les illustrations de la Fig. 6.3 montrent quelques images interpolées de deux animations différentes. Le nombre de *keyframes* est plus élevé et des interpolations des types accélération et décélération ont été couramment utilisées.

6.3 Animations basées sur des modèles physiques

La plupart des contributions en animation de chevelures sont axées sur des modèles physiques, question de créer des mouvements sans interpolation de *keyframes*, mais plutôt à partir de forces externes, telles que le vent ou les mouvements de la tête.

L'utilisation de modèles physiques pour l'animation dépasse le cadre de ce travail. Cependant, certains modèles déjà existants pourraient être utilisés avec la technique proposée, particulièrement les modèles masses-ressorts. Le lecteur pourra être inspiré par

[†] *http ://www.iro.umontreal.ca/~cotema/hairnpr*

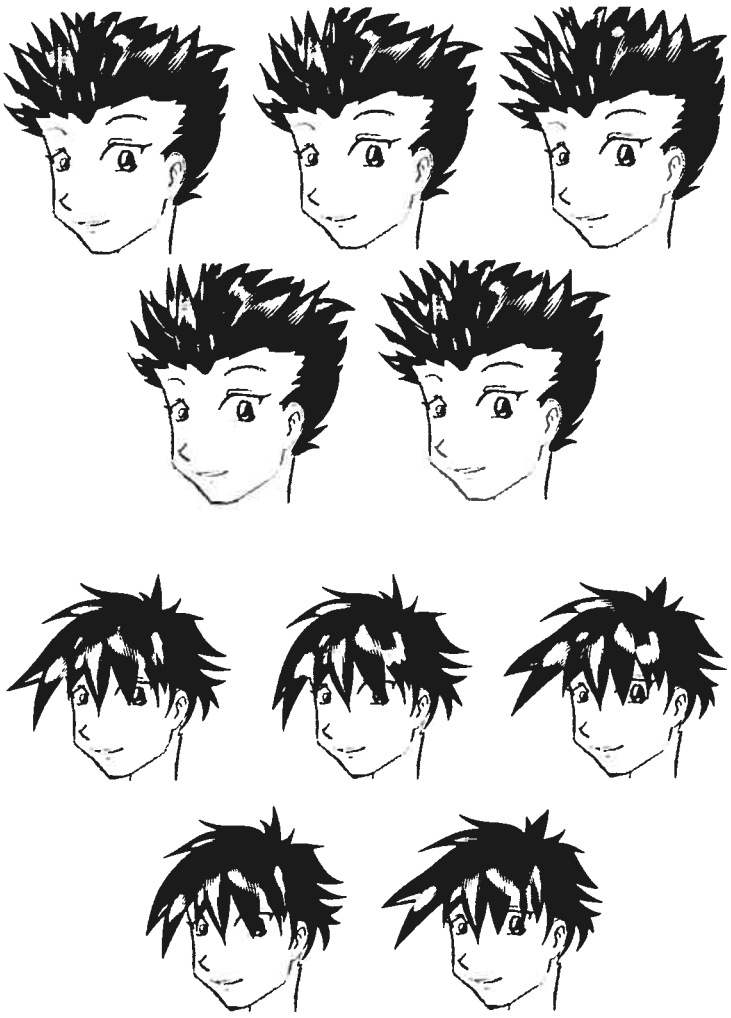


FIG. 6.3 – Mèches interpolées sur des têtes entières. Images originales tirées de [bak].

des lectures supplémentaires sur le sujet. La Section 2.1.2 propose plusieurs références concernant l'animation de chevelures, mais d'autres ouvrages concernant la simulation physique ainsi que l'intégration pourraient être intéressants. En particulier, l'ouvrage de Feynman [FLS89] est formidable pour expliciter les notions complexes de physique. D'autre part, les articles de Hecker [Hec96, Hec97a, Hec97b] expliquent à merveille par des algorithmes simples la simulation d'objets rigides appliquée à l'infographie moderne.

Chapitre 7

Conclusion

People do not like to think. If one thinks, one must reach conclusions. Conclusions are not always pleasant.

Helen Keller

Conclure sur un sujet de rendu non-photoréaliste est particulièrement difficile lorsque peu d'ouvrages sur la matière en question sont disponibles. Tel que mentionné dans les premiers chapitres de ce document, très peu d'auteurs ont attaqué le problème de rendu non-photoréaliste de chevelures ou encore de pilosité en général. De plus, ceux qui l'ont fait n'avaient pas l'idée de simplifier le travail des artistes, mais plutôt de produire un *pipeline* graphique destiné au rendu non-photoréaliste. Pour ces raisons, il est difficile de voir si les objectifs fixés au départ ont été atteints. En fait, le seul moyen vraisemblable d'y arriver serait de faire essayer les techniques développées dans cet ouvrage par un artiste ayant beaucoup d'expérience industrielle. Cependant, certaines contraintes rendent difficile l'accès à un tel artiste qui pourrait contribuer significativement à l'élaboration d'une interface "utilisable" dans un contexte industriel. Il nous est donc impossible de faire valider nos résultats par un artiste, tout du moins

en ce qui concerne l'interface, mais nous pouvons tout de même discuter des résultats réalisés par un étudiant *sans talent artistique* et *sans formation en dessin.* Nous verrons qu'à ce sujet, les techniques proposées permettront à quiconque possédant un intérêt en dessin de produire des chevelures de haute qualité avec un minimum d'effort.

D'un autre côté, il est plus facile de juger la qualité des rendus : il s'agit essentiellement de se fier à notre entourage et à leurs suggestions. Bien que certains puristes prétendent qu'il est préférable d'utiliser des méthodes empiriques (des métriques par exemple) afin de juger de la qualité d'un résultat, l'auteur croit que dans le contexte non-photoréaliste présenté dans ce travail, il est inapproprié d'utiliser de telles méthodes. Une simple comparaison visuelle aux résultats obtenus par de véritables artistes devrait suffire largement.

En ce qui concerne les animations, leur qualité correspond directement au talent de l'artiste. Néanmoins, la création d'outils puissants et faciles à utiliser pour l'usager pourra permettre à pratiquement tout le monde de produire des animations cohérentes dans un contexte de dessins animés, même si l'utilisateur ne possède pas, ou peu, de talent artistique, ce qui est précisément le cas avec les animations produites dans cet ouvrage. Encore une fois, nous n'utiliserons aucune métrique pour évaluer la qualité des animations : le jugement d'un être humain est, selon l'avis de l'auteur, un bien meilleur indicateur.

Ce chapitre aura donc la structure suivante : la Section 7.1 discutera des avantages et inconvénients des outils de modélisation proposés, la Section 7.2 discutera des forces et des améliorations possibles du système de rendu et la Section 7.3 discutera des problèmes avec les techniques d'animation du système.

7.1 Discussion de l'interface de modélisation

Tel que mentionné précédemment, il est difficile de juger de la qualité d'une interface sans que l'utilisateur destiné l'ait utilisé. Il s'agit précisément de notre situation ici, étant donné les difficultés rencontrées en essayant de trouver un artiste compétent qui accepte de nous donner des *feedbacks* significatifs. Néanmoins, l'utilisation abondante de l'auteur

(ainsi que de quelques collègues) pourra donner une indication générale non-négligeable. La lacune majeure dans cette étude est le manque de connaissances artistiques des utilisateurs : tenter de reproduire fidèlement les chevelures avec les mêmes mouvements que les artistes professionnels devient rigoureusement plus difficile.

Le Chapitre 3 a proposé un *framework* général de création et de rendu de chevelures non-photoréalistes. Le système présenté a illustré le désir de produire un encrage de chevelures au-dessus d'un dessin fait typiquement au plomb. Les illustrations présentées un peu partout dans cet ouvrage montrent que les artistes ont tendance à simplifier le dessin des chevelures en les regroupant par mèches. Nous avons donc tenté d'exploiter cette simplification par une interface de création similaire. Le Chapitre 4 a présenté en détail une solution de modélisation de surfaces représentant des mèches de cheveux. Essentiellement, nous avons créé des surfaces *NURBS* à partir d'une paire de traits générés par l'usager. Cette approche espère de permettre la reproduction fidèle des mouvements artistiques.

Le type d'outil utilisé par l'utilisateur lors de la création des traits importe pour beaucoup. Il est facilement imaginable que l'artiste voudra utiliser un *input device* similaire à un crayon afin de se sentir davantage en confiance avec ses mouvements. Heureusement, de tels *input devices* existent et sont couramment utilisés. Nous pouvons donc imaginer un artiste apposant l'image originale sur sa table, et dessinant les mèches par dessus. Le système proposé peut générer automatiquement un encrage de base des cheveux et il suffirait ensuite d'ajuster quelques paramètres (Chapitre 5) afin d'obtenir l'apparence voulue.

Une fois la surface générée, il est vraisemblable que l'utilisateur ne soit pas satisfait du résultat, ou plutôt, de la forme produite par le système. Pour cette raison, nous avons proposé une interface de modélisation permettant la modification d'une surface. Plus particulièrement, nous avons proposé d'utiliser la technique FFD (*free form deformation*) afin de produire des modifications au niveau désiré par l'utilisateur. Essentiellement, l'artiste peut sélectionner une région sur la surface où il désire effectuer des changements. La surface FFD (une *patch* de *Bézier*) entourera alors la région choisie par l'usager. Par la suite, la modification de cette surface permettra le changement de la région en

question. Cette approche possède un avantage majeur : la flexibilité du contrôle. En effet, l'usager peut décider de modifier l'apparence de la mèche entière (qui peut être constituée de plusieurs dizaines, voir plusieurs centaines de points de contrôles dans les cas extrêmes) avec une simple surface à seize points de contrôles. D'autre part, l'interface permet également d'autres transformations sur la surface FFD, telles que des translations, des mises à échelle ou des rotations, permettant un contrôle supplémentaire "gratuit". Le désavantage majeur de cette technique est l'exposition de la structure des surfaces. Effectivement, la région choisie par l'utilisateur pour la surface de déformation est nécessairement contrainte aux points de contrôles de la surface représentant la mèche, puisque cette dernière est modifiée par l'intermédiaire de ces points. Nous aimerions isoler la configuration des points de contrôles de l'utilisateur qui ne connaît probablement rien à la théorie des surfaces paramétriques.

Il existe d'autres facettes aux outils de modélisation, particulièrement en ce qui concerne le positionnement des pseudo-lumières sur les surfaces. Cependant, ces autres paramètres modifiables sont discutés à la section suivante concernant le rendu.

7.2 Discussion du rendu

Nous avons proposé une approche permettant de faire le rendu de surfaces avec un style *cartoon*. Il existe plusieurs styles produits par des centaines d'artistes différents, et trouver une solution générale permettant de reproduire n'importe quel de ces styles se ferait au détriment de la facilité d'utilisation par un artiste. Nous avons donc fait un choix : nous avons choisi un style très répandu, mais difficile à produire dans l'industrie. Il s'agit du *feathering*.

La technique de rendu de *feathering* est la contribution majeure de cet ouvrage. Évidemment, il existe plusieurs façons d'obtenir des résultats similaires, voir meilleurs. Cependant, nous avons proposé une solution flexible sur une approche polygonale permettant un rendu interactif avec OpenGL (de plus amples informations concernant les temps de calculs sont explicitées à l'Annexe B).

En résumé, le rendu de *feathering* requiert deux éléments : une surface *NURBS* ainsi

qu'une source permettant de générer des reflets que nous avons nommé *pseudo-lumière*. La création de la surface a été discutée à la section précédente, mais la création et le positionnement de la pseudo-lumière sont également problématiques. Nous avons donc proposé une technique simple permettant à l'usager de spécifier une région dans laquelle il désire un reflet et le système produira automatiquement une pseudo-lumière générant un tel reflet. Cette solution s'est avérée plus intéressante qu'espérée. Effectivement, certaines mèches de cheveux nécessiteront plusieurs pseudo-lumières pour provoquer un effet intéressant. Il est donc indispensable de fournir un outil puissant pour les produire efficacement.

Le rendu même du *feathering* est paramétrable avec un nombre restreint de variables : le nombre de traits (*hatches*), la vitesse de croissance des traits ainsi que le décalage des traits. Ces trois paramètres permettent de décrire simplement une grande variété de styles de *feathering*. Il s'agit donc d'un avantage majeur.

Le principal problème est l'amélioration de l'apparence des surfaces, c'est-à-dire les perturbations selon la profondeur. Bien que les résultats ainsi obtenus (Fig. 5.5(c)) soient visuellement intéressants, ils ne sont pas particulièrement intuitifs à obtenir. Au moins, des perturbations même très simples pourront généralement produire les résultats voulus et ceux-ci sont obtenus interactivement. Cependant, une approche plus puissante devrait être éventuellement développée. Idéalement, l'usager devrait pouvoir spécifier une région arbitraire dans laquelle il désire un reflet (pas nécessairement une région circulaire). Le système devrait pouvoir perturber la surface en conséquence pour produire les reflets voulus. Évidemment, une telle solution n'est pas triviale à réaliser ; il s'agit donc de travaux futurs au projet.

Un autre problème relié à notre problème est la présence d'aliasing causé par la présence de polygones de petite taille. Une solution potentielle à ce problème serait de faire le rendu du *feathering* dans une texture et d'appliquer cette texture sur les surfaces. La génération, l'affichage et le filtrage de textures est très efficace en OpenGL, ce qui pourrait améliorer l'apparence des chevelures. Il serait donc très intéressant d'explorer cette possibilité.

La technique de rendu proposée permet de créer facilement des chevelures non-

photoréalistes, mêmes par des utilisateurs sans entraînement et sans véritable talent artistique. Malgré les quelques désavantages énumérés dans cette section, nous pouvons considérer la solution choisie comme étant satisfaisante, malgré que nous n'ayons aucune garantie si un artiste professionel pourrait facilement adopter le système.

7.3 Discussion des animations

Les animations obtenues dans le cadre de ce travail sont relativement simplistes et ne servaient qu'à illustrer la capacité du système à produire des mouvements cohérents dans les chevelures. Néanmoins, les outils fournis sont facilement intégrables dans le processus d'animation industriel. Présentement, un artiste génère les *keyframes* de l'animation alors qu'une gamme de dessinateurs produisent les images intermédiaires : l'interpolation des images est donc produite manuellement. Nous avons fourni une alternative à cette approche pour l'animation de chevelures, qui sont d'autant plus complexes à animer.

Cependant, il aurait été très intéressant de produire des animations avec un modèle physique, même très simple. Même si les artistes n'ont pas l'habitude de travailler avec des systèmes de forces externes, une telle approche aurait pu produire des animations cohérentes sans l'intervention d'un expert en animation.

Donc, en résumé, nous avons développé un système de modélisation, de rendu et d'animation de chevelures non-photoréalistes dans le but de faciliter le travail des artistes traditionnels. Le système est inspiré d'une technique populaire et puissante nommée *feathering* pour produire des chevelures de qualité supérieure interactivement avec OpenGL. Les artistes bénéficieront d'une interface simple d'utilisation et fidèle aux techniques traditionnelles pour dessiner des mèches de cheveux, alors que le système de rendu utilise un nombre très réduit de paramètres. Le système propose également un interpolateur de *keyframes* permettant de simplifier, ou même d'annuler le travail des *in-betweeners*. Les résultats obtenus, malgré l'absence de métriques pour évaluer leur qualité, sont visuellement convainquants et plausibles dans un contexte de dessins animés. Nous croyons que le système serait non seulement utilisable dans l'industrie du *cartoon*, mais permettrait

également d'améliorer la qualité du produit final. Les rendus et animations obtenus sont très comparables à ceux produits par des professionnels (même s'ils ont été créés par un amateur utilisant notre système) et le temps de production est considérablement réduit. Il semble donc évident que les objectifs fixés au Chapitre 1 sont atteints. Les artistes ont maintenant la possibilité d'améliorer la qualité de leurs rendus et animations avec un système simple et intuitif d'utilisation.

Annexe A

HairNPR : le système

HairNPR est une implémentation des techniques proposées dans ce travail. Le but de cet annexe est de fournir un *framework* de base à quiconque désirant produire un système de rendu non-photoréaliste de chevelures. L'auteur invite donc toute personne intéressée à s'inspirer du travail déjà réalisé. Le langage *C++* [Str00] fut utilisé pour la réalisation du logiciel : la présence de compilateurs hautement optimisants, la grande variété de librairies disponibles, ainsi que la puissance des expressions de ce langage en font un choix évident.

L'implémentation utilise un certain nombre de classes. Leurs relations ainsi que leurs rôles dans le logiciel sont explicités à la Section A.1. Certains modules sont également décrits plus en détails dans les sections subséquentes, notamment le module de rendu (Section A.2) et le module d'animation (Section A.3).

La structure du logiciel a été inspirée d'un certain nombre d'ouvrages. Le lecteur pourra être intéressé à certaines références en matière de *design C++* ([ST01, Ale01, Dou02]), mais également en matière de concepts avancés en C++ ([Str00, Jos99]).

A.1 Classes

La Fig. A.1 montre la structure statique du logiciel. Elle est composée d'un certain nombre de classes*, lesquelles sont décrites avec plus de détails dans cette section.

- ***MainWindow***. Cette classe est une abstraction du contexte graphique du *window manager* du système d'exploitation. Cette classe fournit les éléments graphiques nécessaires à l'usager pour pouvoir fonctionner (*e.g.* fenêtre, boutons, etc.). Son rôle principal est de contenir un *Viewport*.
- ***Viewport***. Cette classe fait une abstraction du modèle d'affichage du système. Il s'agit en fait d'une sous-classe d'une fenêtre OpenGL, permettant ainsi les mêmes fonctionnalités. Elle est comprise dans un *MainWindow*, et possède une instance de *Renderer* pour l'affichage OpenGL.
- ***OpenGLViewport***. Cette classe est fournie par une librairie externe.† Son rôle est de permettre un affichage de primitives OpenGL. Elle n'est pas utilisée directement dans le système : elle est plutôt utilisée par une sous-classe (voir *Viewport*).
- ***Renderer***. Il s'agit de la classe produisant les primitives OpenGL. Elle utilise principalement les surfaces *NURBS* pour produire son affichage, mais également d'autres objets (*e.g.* sources pseudo-lumières).
- ***FeatheringPatch***. Cette classe encapsule tout le comportement d'une surface *NURBS* (dont elle dérive) pour permettre le rendu de *feathering* tel que décrit au Chapitre 5. Elle possède une instance de surface de *free form deformation* pour la modélisation.
- ***NURBSSurface***. Cette classe est fournie par une librairie externe.‡ Elle fait l'abstraction des comportements d'une surface *NURBS*.
- ***FFDPatch***. Cette classe permet de modéliser les surfaces de *feathering* par la technique de *free form deformation*. Il s'agit d'une surface de *Bézier*.
- ***EventHandler***. Cette classe filtre les événements de l'usager (reçus par le *MainWindow*) pour permettre la modélisation des surfaces par l'intermédiaire du *Mo-*

*Afin de simplifier la représentation, certaines classes secondaires ont été omises dans le diagramme.
†L'implémentation a été réalisée à l'aide de la librairie *Qt* [Tro].
‡*NURBS++ Library* [Lav].

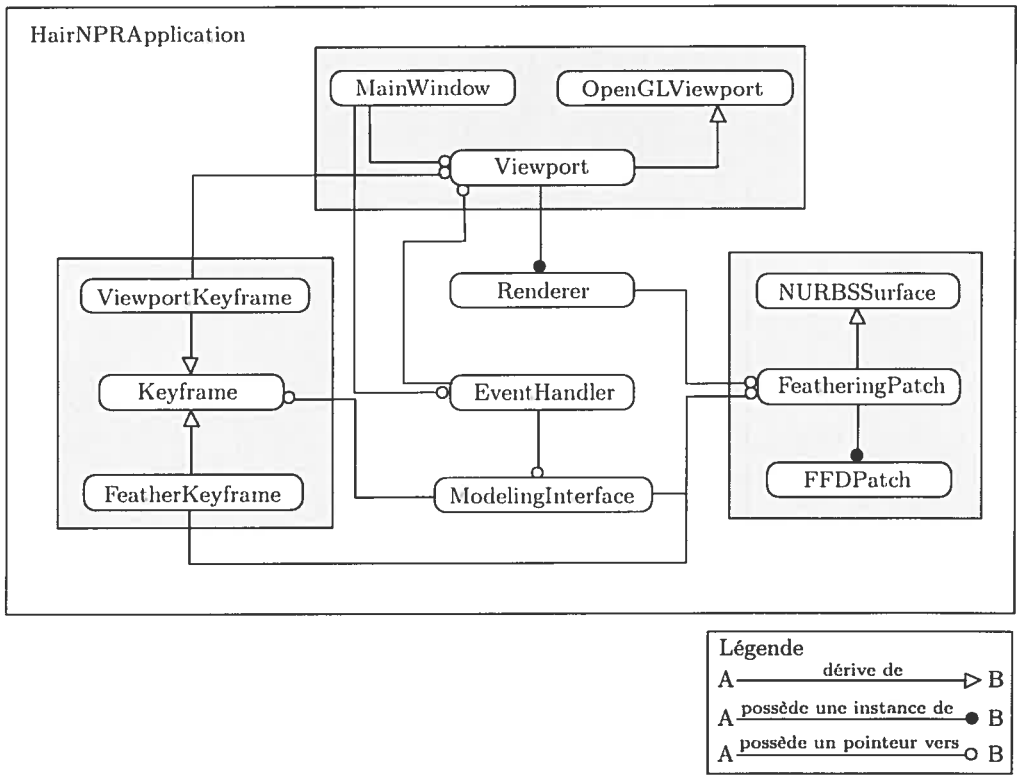


FIG. A.1 – Structure statique de *HairNPR*.

delingInterface. Elle permet également quelques transformations sur le *Viewport* (translation, rotation, etc.).

- ***ModelingInterface***. Classe permettant d'encapsuler les différents outils de modélisation décrits au Chapitre 4.
- ***Keyframe***. Super-classe de tous les *keyframes* disponibles, *i.e. ViewportKeyframe* et *Featherkeyframe*. Elle fournit les fonctionalités générales permettant d'ajouter, de remplacer et d'interpoler les *keyframes* avec les techniques décrites au Chapitre 6.
- ***ViewportKeyframe***. Sous-classe de *Keyframe* concernant toutes les transformations du *Viewport*, *i.e.* translations de la caméra, rotations, etc.
- ***FeatherKeyframe***. Sous-classe de *Keyframe* concernant toutes les transformations des surfaces *NURBS*. Celles-ci incluent les modifications sur les points de contrôles et les pseudo-lumières.

A.2 Module de rendu

Le module de rendu concerne les classes de *Renderer* et de *FeatheringPatch*. Le *Renderer* contient un vecteur de pointeurs vers toutes les surfaces de *feathering*, permettant ainsi la production de primitives OpenGL aisément. Les surfaces contiennent un certain nombre de paramètres, les plus importants étant la liste des points de contrôles, les dimensions de la surface (longueur/largeur), la liste des sources de pseudo-lumières et une instance de la surface FFD pour la modélisation.

Le *Renderer* fait très peu de travail en soit : il appelle plutôt les fonctions *render()* des différentes surfaces. Son rôle est alors de permettre les transformations de la caméra. La fonction *render()* des surfaces fait l'essentiel du rendu du *feathering*. Il utilise la position des lumières pour calculer la variation de la largeur des traits. L'algorithme de rendu des surfaces est explicité à l'Algorithme A.1. La fonction *feathering()* utilisée dans cet algorithme est une implémentation simple de la formule de calcul de largeur des traits du Chapitre 5. Notez que cette fonction utilise la contribution maximale des sources de pseudo-lumière sur la surface. Bien qu'il semblerait plus approprié d'effectuer

une somme sur les contributions, l'expérience à démontré que l'utilisation du maximum donne des résultats plus appropriés dans un contexte de dessin. Il s'agit tout de même d'un détail d'implémentation.

input : *Une patch* $\mathcal{P}$
input : *Un nombre de traits* n_t
input : *Une résolution de trait* n
input : *Un offset* σ
input : *Un coefficient de croissance* ω

---

for $l \leftarrow 0$ ***to*** *1* ***by*** $1/n_t$ **do**
  for $s \leftarrow 0$ ***to*** *1* ***by*** $1/n$ **do**
    $\vec{pp} \leftarrow \mathcal{P}_{l,s-1}$;
    $\vec{ps} \leftarrow \mathcal{P}_{l-1,s}$;
    $\vec{pls} \leftarrow \mathcal{P}_{l-1,s-1}$;
    $\vec{p} \leftarrow \mathcal{P}_{l,s}$;
    $\vec{n} \leftarrow$ normale à la position $\mathcal{P}_{l,s}$;
    $ill \leftarrow 0$;
    foreach *pseudo-lumière* $\vec{h}$ **do**
      $\vec{g} \leftarrow (\vec{h} - \vec{p})$;
      $t \leftarrow 1 - \vec{n} \cdot \vec{g} - \sigma$;
      if $t > ill$ **then** $ill \leftarrow t$;
    end
    glBegin(GL_QUADS) ;
    glVertex3fv($\vec{pp}$) ;
    glVertex3fv($\vec{p}$) ;
    glVertex3fv(feathering($\vec{p}, (\vec{ps} - \vec{p}), ill, \omega$)) ;
    glVertex3fv(feathering($\vec{p}, (\vec{pls} - \vec{pp}), ill, \omega$)) ;
    glEnd() ;
  end
end

Algorithme A.1: Procédure *render()* pour faire le rendu du *feathering* avec OpenGL.

Évidemment, une telle procédure est particulièrement exigeante à calculer. La bonne nouvelle est que son exécution n'est nécessaire que lorsque des changements bien précis se produisent : soit un changement au niveau de la surface (position des points de

contrôles) ou encore un changement de position d'une source de pseudo-lumière. Les autres transformations, particulièrement celles de la caméra, n'ont aucune conséquence sur l'apparence de la surface : la même géométrie peut donc être utilisée. Les temps de calcul requis dans les deux cas sont explicités à l'Annexe B.

A.3 Module d'animation

Le module d'animation est principalement centré sur la classe *Keyframe* qui gère l'ensemble des images clées ainsi que leurs interpolations (voir Chapitre 6). Essentiellement, cette classe fournit un certain nombre de services pour insérer, effacer ou modifier la position de *keyframes*. Elle fournit également des services pour interpoler ces mêmes *keyframes*. Pour ce faire, elle doit connaître les éléments à interpoler : ceci est réalisé par les sous-classes qui ajoutent les détails d'interpolation à la classe *Keyframe*. En ce moment, il existe deux sous-classes : *ViewportKeyframe* et *FeatherKeyframe*. Le *ViewportKeyframe* contient principalement les transformations de la caméra, permettant d'interpoler des effets de *zoom*, de *pan*, ainsi de suite. Le *FeatherKeyframe* s'occupe des transformations sur les points de contrôles des surfaces, ainsi que des sources de pseudo-lumières associées.

Une défi majeur dans le module d'animation était de permettre à chaque surface d'avoir son propre groupe de *keyframes* indépendant des autres surfaces. Cette approche est essentielle puisque nous ne désirons pas des *keyframes* globaux, mais plutôt des variations sur la durée des interpolations d'une mèche de cheveux à une autre. La sélection d'une nouvelle mèche a donc plusieurs conséquences dans le système comme le retirement des *keyframes* courants et la prise en charge des nouveaux *keyframes*. Également, le module d'interpolation doit changer de pointeur courant sur la liste des *keyframes* à traiter.

Une implémentation des techniques d'interpolation du Chapitre 6 a été réalisée. La classe *Keyframe* fournit une fonction *interpolateKeyframe()* qui prend en paramètre un temps t ainsi qu'une fonction d'interpolation et retourne un modèle interpolé à ce temps t (le modèle dépend de la sous-classe utilisée). Pour arriver à faire l'interpolation, la

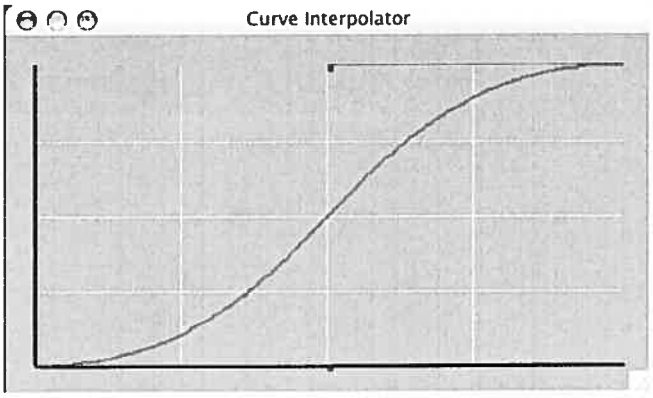


FIG. A.2 – Fenêtre pour décrire une fonction d'interpolation.

fonction détermine entre quels *keyframes* dans le temps se trouve t, nommons-les t_0 et t_1. La valeur de t est alors normalisée entre ces deux temps dans l'intervalle $[0, 1]$. Cette valeur normalisée peut être utilisée par la fonction d'interpolation pour évaluer le nouveau modèle interpolé.

La fonction d'interpolation est fournie par l'usager. Il s'agit d'une fenêtre (au sens du système d'exploitation) permettant d'éditer une courbe de *Bézier* destinée à représenter une fonction d'interpolation. Cette fenêtre est illustrée à la Fig. A.2. L'usager peut modifier les points de contrôles de la courbe (illustrés par des points bleus sur la figure) qui sont contraints de manière à ce que la courbe représente toujours une fonction, *i.e.* une fonction continue qui n'a qu'une seule valeur y pour chaque valeur x et vice-versa. Les courbes de *Bézier* sont facilement contraignables pour obtenir un tel résultat : il suffit de fixer les deux points de contrôles extrêmes (P_1 et P_4) et de contraindre les deux points de contrôles intermédiaires (P_2 et P_3) dans le rectangle défini par P_1 et P_4. Une preuve formelle de la validité de cette approche dépasse le contexte de ce travail, mais la pratique a démontré la fiabilité et la robustesse de cette approche.

Les points de contrôles normalisés entre $[0, 1]$ de la courbe de *Bézier* sont utilisés pour évaluer la position de la valeur interpolée. L'Algorithme A.2 décrit techniquement comment ce processus est réalisé. Il est difficile, étant donné une courbe paramétrique et une position en x, de trouver l'équivalent en y. Dans le cas général, la courbe ne représente pas nécessairement une fonction, alors plusieurs y pourraient exister pour

un seul x. Nos contraintes additionnelles nous assurent cette unicité, mais évaluer la fonction n'est pas plus simple pour autant. La solution utilisée est d'évaluer différents points sur la courbe et de vérifier si la valeur obtenue est suffisamment près de la valeur recherchée. Si c'est le cas, nous retournons cette valeur. Sinon, nous continuons à chercher sur la courbe. Afin d'éviter des recherches inutiles, quelques optimisations peuvent être effectuées pour nous permettre un changement de fonction en temps réel. L'algorithme commence par chercher à $t = \frac{1}{2}$. Si la valeur cherchée est plus petite que $\frac{1}{2}$, on cherche dans la portion $[0, \frac{1}{2}[$ de la courbe, sinon on recherche dans la portion $]\frac{1}{2}, 1]$ de la courbe. Le même processus peut être répété jusqu'à ce que la différence entre la valeur cherchée et la valeur trouvée soit plus petite qu'un epsilon ϵ donné.

input : *Une courbe de Bézier* $\mathcal{B}$
input : *Un temps recherché* t_r
output : *Un temps trouvé* t_t

$t \leftarrow 1/2$;
$inc \leftarrow t/2$;
while *true* **do**
  $\vec{p} \leftarrow \mathcal{B}_t$;
  if $|\vec{p}.x - t_r| \leq \epsilon$ **then**
    $t_t \leftarrow \vec{p}.y$;
    return ;
  end
  if $\vec{p}.x > t$ **then**
    $t \leftarrow t - inc$;
  end
  if $\vec{p}.x < t$ **then**
    $t \leftarrow t + inc$;
  end
  $inc = inc/2$;
end

Algorithme A.2: Procédure pour évaluer l'interpolation à partir d'une *Bézier*.

Annexe B

Temps de calculs

Tous les temps de calcul on été effectués sur un PowerPC G4, 1 GHz et 512 MB de RAM. La scène utilisée était toujours la même : une surface de 16 points de contrôles ainsi qu'une source de pseudo-lumière. La raison de ces choix réside dans le fait que le nombre de pseudo-lumières n'a aucune influence majeure sur le temps de calcul, de même pour le nombre de points de contrôles. Les Tables B.1 et B.2 (page suivante) illustrent le nombre d'images par seconde (*frames per second* — FPS) dans ces conditions avec ou sans le calcul complet de la géométrie (voir Annexe A).

Voici quelques notes concernant les temps de calcul. La Table B.1 montre les résultats sans calcul de la géométrie. En pratique, le rendu est effectué par les *display lists* d'OpenGL, ce qui explique l'incohérence des FPS lorsque le nombre de polygones est peu élevé. C'est uniquement lorsque le nombre de polygones devient considérablement plus important que les FPS diminuent. Le nombre de polygones peut être estimé assez fidèlement avec la formule suivante :

$$p = 2lr,$$

où p est le nombre de polygones, l est le nombre de lignes et r est la résolution.

Le nombre de lignes et les résolutions de la Table B.2 sont sur une échelle plus faible puisque l'évaluation de la géométrie est significativement plus lente que les *display lists*. Il est important d'illustrer des valeurs significatives pour les FPS.

# Lignes	Résolution	*FPS*
20	30	524.4
40	30	597.4
60	30	521.0
80	30	532.6
100	30	464.2
20	30	608.4
20	60	576.0
20	90	491.2
20	120	438.2
20	150	400.2
40	150	438.2
60	150	400.2
80	150	319.4
100	150	299.2

TAB. B.1 – Temps de calcul sans évaluation de la géométrie.

# Lignes	Résolution	*FPS*
10	10	36.4
20	10	19.6
30	10	13.0
40	10	10.0
10	20	20.4
10	30	12.6
10	40	10.0
20	40	5.0
30	40	3.6
40	40	3.0

TAB. B.2 – Temps de calcul avec évaluation de la géométrie.

Bibliographie

[Ala] Gerry Alanguilan. « Art Tips ». http://www.laguna.net/~timawa/tips2.htm.

[Ale01] Andrei Alexandrescu. *Modern C++ Design, Generic Programming and Design Patterns Applied.* Addison-Wesley, 2001.

[AUT92] K. Anjyo, Y. Usami et T.Kurihara. « A Simple Method for Extracting the Natural Beauty of Hair ». In *Proceedings of SIGGRAPH 92*, pages 26(2) :111–120, July 1992.

[bak] « Baka neko : Anime Drawing Tutorials ». http://www.cat-print.com/howto/.

[Bar84] Alan H. Barr. « Global and Local Deformations of Solid Primitives ». In *Computer Graphics (Proceedings of SIGGRAPH 84)*, volume 18, pages 21–30, juillet 1984.

[BBB87] Richard H. Bartels, John C. Beatty et Brian A. Barsky. *An introduction to splines for use in computer graphics & geometric modeling.* Morgan Kaufmann Publishers Inc., 1987.

[BC02] Seok-Hyung Bae et Byoung K. Choi. « NURBS surface fitting using orthogonal coordinate transform for rapid product development ». *Computer-Aided Design*, volume 34, numéro 10, pages 683–690, septembre 2002.

[BKCN03] Florence Bertails, Tae-Yong Kim, Marie-Paule Cani et Ulrich Neumann. « Adaptive Wisp Tree - a multiresolution control structure for simulating dynamic clustering in hair motion ». *Symposium on Computer Animation '03*, juillet 2003.

[Bli77] James F. Blinn. « Models of Light Reflection For Computer Synthesized Pictures ». In *Computer Graphics*, volume 11, pages 192–198, juillet 1977.

[CSDI99] Lieu-Hen Chen, Santi Saeyor, Hiroshi Dohi et Mitsuru Ishizuka. « A system of 3D hair style synthesis based on the wisp model ». *The Visual Computer*, volume 15, numéro 4, pages 159–170, 1999.

[Dou02] Bruce Powel Douglass. *Real-Time Design Patterns : Robust Scalable Architecture for Real-Time Systems*. Addison-Wesley, 2002.

[DS00] Oliver Deussen et Thomas Strothotte. « Computer-Generated Pen-and-Ink Illustration of Trees ». In *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, pages 13–18, juillet 2000.

[DTKT93] Agnes Daldegan, Nadia Magnenat Thalmann, Tsuneya Kurihara et Daniel Thalmann. « An Integrated System for Modeling, Animating and Rendering Hair ». volume 12, numéro 3, pages 211–221, 1993.

[FLS89] Feynman, Leighton et Sands. *The Feynman Lectures on Physics*, volume 1. Addison Wesley, 1989.

[FvDFH96] James D. Foley, Andries van Dam, Steven K. Feiner et John F. Hughes. *Computer Graphics — Principles and Practice*. The Systems Programming Series. Addison-Wesley, second edition in c édition, 1996.

[GCS02] Bruce Gooch, Greg Coombe et Peter Shirley. « Artistic Vision : Painterly Rendering Using Computer Vision Techniques ». In *NPAR 2002 : Second International Symposium on Non Photorealistic Rendering*, pages 83–90, juin 2002.

[GGSC98] Amy Gooch, Bruce Gooch, Peter S. Shirley et Elaine Cohen. « A Non-Photorealistic Lighting Model for Automatic Technical Illustration ». In *Proceedings of SIGGRAPH 98*, Computer Graphics Proceedings, Annual Conference Series, pages 447–452, juillet 1998.

[Hec96] Chris Hecker. « Physics, Part 1 ». *Game Developer Magazine*, novembre 1996.

[Hec97a] Chris Hecker. « Physics, Part 2 ». *Game Developer Magazine*, janvier 1997.

[Hec97b] Chris Hecker. « Physics, Part 3 ». *Game Developer Magazine*, mars 1997.

[Her98] Aaron Hertzmann. « Painterly Rendering with Curved Brush Strokes of Multiple Sizes ». In *Proceedings of SIGGRAPH 98*, Computer Graphics Proceedings, Annual Conference Series, pages 453–460, juillet 1998.

[HMT00] Sunil Hadap et Nadia Magnenat-Thalmann. « Interactive Hair Styler based on Fluid Flow ». In *Computer Animation and Simulation 2000*, pages 87–99, août 2000.

[HZ00] Aaron Hertzmann et Denis Zorin. « Illustrating Smooth Surfaces ». In *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, pages 517–526, juillet 2000.

[iAUK92] Ken ichi Anjyo, Yoshiaki Usami et Tsuneya Kurihara. « A simple method for extracting the natural beauty of hair ». In *Computer Graphics (Proceedings of SIGGRAPH 92)*, volume 26, pages 111–120, juillet 1992.

[ima] « Images Central ». http ://www.imagescentral.org.

[JO02] Pierre-Marc Jodoin et Victor Ostromoukhov. « Error-Diffusion with Blue Noise Properties for Midtones ». In *Proceedings SPIE Conf. Electronic Imaging*, pages 293–301, 2002.

[Jos99] Nicolai M. Josuttis. *The C++ Standard Library : A Tutorial and Reference.* Addison-Wesley, 1999.

[Kan99] Henry Kang. *Digital Color Halftoning.* SPIE Optical Engineering Press, 1999.

[KDMF03] Robert D. Kalnins, Philip L. Davidson, Lee Markosian et Adam Finkelstein. « Coherent Stylized Silhouettes ». *ACM Transactions on Graphics*, volume 22, numéro 3, pages 856–861, juillet 2003.

[KGC00] Matthew Kaplan, Bruce Gooch et Elaine Cohen. « Interactive Artistic Rendering ». In *NPAR 2000 : First International Symposium on Non Photorealistic Animation and Rendering*, pages 67–74, juin 2000.

[KH00] Chuan Koon Koh et Zhiyong Huang. « Real-Time Animation of Human Hair Modeled in Strips ». In *Computer Animation and Simulation 2000*, pages 101–110, août 2000.

[KH01] Chuan Koon Koh et Zhiyong Huang. « A simple physics model to animate human hair modeled in 2D strips in real time ». In *Computer Animation and Simulation 2001*, pages 101–110, août 2001.

[KiAT93] Tsuneya Kurihara, Ken ichi Anjyo et Daniel Thalmann. « Hair Animation with Collision Detection ». In *Proceedings of Computer Animation '93*, pages 128–138, 1993.

[KK89] James T. Kajiya et Timothy L. Kay. « Rendering Fur with Three Dimensional Textures ». In *Computer Graphics (Proceedings of SIGGRAPH 89)*, volume 23, pages 271–280, juillet 1989.

[KMM+02] Robert D. Kalnins, Lee Markosian, Barbara J. Meier, Michael A. Kowalski, Joseph C. Lee, Philip L. Davidson, Matthew Webb, John F. Hughes et Adam Finkelstein. « WYSIWYG NPR : Drawing Strokes Directly on 3D Models ». *ACM Transactions on Graphics*, volume 21, numéro 3, pages 755–762, juillet 2002.

[KMN+99] Michael A. Kowalski, Lee Markosian, J. D. Northrup, Lubomir Bourdev, Ronen Barzel, Loring S. Holden et John Hughes. « Art-Based Rendering of Fur, Grass, and Trees ». *Proceedings of SIGGRAPH 99*, pages 433–438, 1999.

[KN02] Tae-Yong Kim et Ulrich Neumann. « Interactive Multiresolution Hair Modeling and Editing ». *Proceedings of SIGGRAPH '02*, pages 620–629, 2002.

[LA01] Daniel Lau et Gonzalo Arce. *Modern Digital Halftoning*. Macel Dekker, 2001.

[Lav] Philippe Lavoie. « NURBS++ Library ». http ://libnurbs.sourceforge.net.

[LTT91] Andre M. LeBlanc, Russell Turner et Daniel Thalmann. « Rendering Hair using Pixel Blending and Shadow Buffers ». *Journal of Visualization and Computer Animation*, volume 2 (3), 1991.

[LV00] Tom Lokovic et Eric Veach. « Deep Shadow Maps ». In *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, pages 385–392, jul 2000.

[Mei96] Barbara J. Meier. « Painterly Rendering for Animation ». *Computer Graphics*, volume 30, numéro Annual Conference Series, pages 477–484, 1996.

[MJC+03] Stephen R. Marschner, Henrik Wann Jensen, Mike Cammarano, Steve Worley et Pat Hanrahan. « Light Scattering From Human Hair Fibers ». *ACM Transactions on Graphics*, volume 22, numéro 3, pages 780–791, juillet 2003.

[MKT+97] Lee Markosian, Michael A. Kowalski, Samuel J. Trychin, Lubomir D. Bourdev, Daniel Goldstein et John F. Hughes. « Real-Time Nonphotorealistic Rendering ». In *Proceedings of SIGGRAPH 97*, Computer Graphics Proceedings, Annual Conference Series, pages 415–420, août 1997.

[MMK+00] Lee Markosian, Barbara J. Meier, Michael A. Kowalski, Loring S. Holden, J. D. Northrup et John F. Hughes. « Art-based Rendering with Continuous Levels of Detail ». In *NPAR 2000 : First International Symposium on Non Photorealistic Animation and Rendering*, pages 59–66, juin 2000.

[MTHK02] Nadia Magnenat-Thalmann, Sunil Hadap et Prem Kalra. « State of the Art in Hair Simulation ». *International Workshop on Human Modeling and Animation*, pages 3–9, 2002.

[Ost99] Victor Ostromoukhov. « Digital facial engraving ». In *Proceedings of SIGGRAPH 99*, pages 417–424, 1999.

[PCP01] Eric Plante, Marie-Paule Cani et Pierre Poulin. « A Layered Wisps Model for Simulating Interactions inside Long Hair ». In *Eurographics Workshop on Computer Animation and Simulation 2001*, pages 139–148, septembre 2001.

[PCP02] Eric Plante, Marie-Paule Cani et Pierre Poulin. « Capturing the Complexity of Hair Motion ». In *Graphical Models*, volume 64, pages 40–58, janvier 2002.

[Pla99] Eric Plante. « Mouvements et interactions de la chevelure par mèches déformables ». M.sc. thesis, Département d'Informatique et Recherche Opérationnelle, Université de Montréal, décembre 1999.

[PT95] Les Piegl et Wayne Tiller. *The NURBS book.* Springer-Verlag, 1995.

[RCT91] R. Rosenblum, W. Carlson et E. Tripp. « Simulating the Structure and Dynamics of Human Hair : Modeling, rendering and animation ». *The Journal of Visualization and Computer Animation*, pages 2(4) :141–148, October – December 1991.

[RJB98] James Rumbaugh, Ivar Jacobson et Grady Booch. *The Unified Modeling Language Reference Manual.* Addison-Wesley, 1998.

[SMGG01] Peter-Pike Sloan, William Martin, Amy Gooch et Bruce Gooch. « The Lit Sphere : A Model for Capturing NPR Shading from Art ». In *Graphics Interface 2001*, pages 143–150, juin 2001.

[Sou03] Mario Costa Sousa. « Theory and Practice of Non-Photorealistic Graphics : Algorithms, Methods, and Production Systems », 2003. SIGGRAPH 03 course #10.

[ST90] Takafumi Saito et Tokiichiro Takahashi. « Comprehensible Rendering of 3-D Shapes ». In *Computer Graphics (Proceedings of SIGGRAPH 90)*, volume 24, pages 197–206, août 1990.

[ST01] Alan Shalloway et James R. Trott. *Design Patterns Explained, A New Perspective on Object Oriented Design.* Addison-Wesley, 2001.

[Str00] Bjarne Stroustrup. *The C++ programming language, 3^{rd} edition.* Addison Wesley, 2000.

[SWHS97] Michael P. Salisbury, Michael T. Wong, John F. Hughes et David H. Salesin. « Orientable Textures for Image-Based Pen-and-Ink Illustration ». In *Proceedings of SIGGRAPH 97*, Computer Graphics Proceedings, Annual Conference Series, pages 401–406, août 1997.

[Tae] Taeha. « Inking and Cleaning up Penciled Drawings ». http ://www.artlair.com/startart/tut_inking.html.

[Tro] Trolltech. « Qt, the C++ Cross-platform Library. ».

[Uli87] Robert Ulichney. *Digital Halftoning.* MIT Press, 1987.

[WNDS99] Mason Woo, Jackie Neider, Tom Davis et Dave Shreiner. *OpenGL Programming Guide.* Addison Wesley, 1999.

[XY01] Zhan Xu et Xue Dong Yang. « V-hairstudio : An Interactive Tool for Hair Design ». mai 2001.

[YO97] Tzong-Jer Yang et Ming Ouhyoung. « Rendering Hair with Back-lighting ». In *Proceedings of CAD/Graphics'97*, pages 291–196, décembre 1997.

[Yu01] Yizhou Yu. « Modeling Realistic Virtual Hairstyles ». In *9th Pacific Conference on Computer Graphics and Applications*, pages 295–304, octobre 2001.