Université de Montréal

Simulation de l'écoulement et de la forme de gouttes sur des surfaces

par

Patrick Fournier

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

Décembre 1997

Université de Montréal
Faculté des études supérieures

Ce mémoire intitulé

Simulation de l'écoulement et de la forme de gouttes sur des surfaces

présenté par

Patrick Fournier

a été évalué par un jury composé des personnes suivantes :

Jean Meunier, président du jury
Pierre Poulin, directeur de recherche
Yoshua Bengio, membre du jury

Mémoire accepté le

Sommaire

En animation assistée par ordinateur, on cherche à automatiser l'animation des actions secondaires d'une scène. Ces actions ne sont habituellement que des conséquences d'autres actions et leur déroulement suit la plupart du temps un ensemble de lois bien déterminées.

Les phénomènes naturels forment une classe d'actions secondaires dont le comportement est dicté par les lois de la physique. Ils sont difficiles à animer manuellement, car leur comportement doit correspondre à ce que le spectateur est habitué de voir. L'écoulement de gouttes de liquide sur une surface est un de ces phénomènes naturels.

Jusqu'à maintenant, aucun modèle rapide et complet d'animation, simulant à la fois le déplacement et la forme des gouttes, n'avait été proposé. La simulation de fluides à l'aide des équations de Navier-Stokes est précise mais coûteuse ; la simulation à l'aide de système à particules est généralement rapide, mais la forme des gouttes et leurs intersections ne sont pas traitées. Ce mémoire propose un modèle rapide qui simule de façon réaliste l'écoulement et la forme de gouttes sur des surfaces. Nous préconisons une approche séparant le phénomène en deux modèles : un modèle d'écoulement et un modèle de forme. Cette séparation permet de raffiner chacun des modèles au niveau de détail approprié.

Le modèle simulant l'écoulement des gouttes utilise une approche analytique pour le calcul des trajectoires, basée sur les caractéristiques du comportement des gouttes. Il simule plusieurs phénomènes connexes, tels les traînées laissées par les gouttes, l'adhérence des gouttes à la surface et la fusion des gouttes lors de collisions. Pour ce dernier phénomène, nous avons développé un algorithme rapide de détection de collisions, qui

utilise les caractéristiques de la surface de support pour réduire le nombre de calculs.

Le modèle simulant la forme des gouttes utilise un système de masses-ressorts pour le calcul de la surface enveloppant la goutte. Ce système est régi par un ensemble de contraintes que nous avons dérivées d'observations sur le comportement des gouttes ; de plus, il permet de déformer légèrement les gouttes de façon aléatoire.

Notre modèle pourrait servir de base à la modélisation d'autres phénomènes liés au déplacement de gouttes, tels la simulation d'accumulations et de dépôts ou la modélisation des textures créées par des liquides (peintures, etc.) qui coulent en s'asséchant. Avec l'ajout de fonctions de rendu sophistiquées, notamment pour faire le rendu de liquides transparents, notre système pourrait être utilisé pour simuler la sueur sur la peau, les larmes ou des gouttes de pluie sur une vitre. Notre modèle pourrait aussi s'insérer dans un système général d'animation de l'eau, qui permettrait de simuler le comportement de différentes « formes » d'eau, comme l'océan, les flaques, la pluie et le brouillard.

Table des matières

Sommaire **iv**

Remerciements **xii**

Conventions de notation **xiii**

1 Introduction **1**

2 Travaux antérieurs **4**

2.1 Animation d'objets rigides 5

2.2 Modélisation de masses d'eau 7

2.2.1 Modèle de Gerstner–Rankine simplifié 7

2.2.2 Modèle hydrodynamique 10

2.3 Systèmes à particules 12

2.3.1 Le système à particules de base 13

2.3.2 Structure, comportements et messages 15

2.3.3 Modélisation d'objets mous 17

2.4 Modélisation de gouttes d'eau 19

2.4.1 Dynamique des fluides 19

2.4.2 Écoulement de gouttes 20

2.5 Rendu 21

2.5.1 Rendu d'objets transparents 22

2.5.2 Rendu des vagues 23

2.6 Le système d'animation *Taarna* 23

3 Animation, modélisation et rendu des gouttes 25

3.1 Mouvement des gouttes 26

3.1.1 Équation analytique de la trajectoire des gouttes 29

3.1.2 Fusion des gouttes 34

3.1.3 Traînées 35

3.1.4 Adhérence à la surface et force de cohésion 40

3.2 Forme des gouttes 42

3.2.1 Modèle masse-ressort 44

3.2.2 Déformations aléatoires 47

3.3 Réalisation 47

3.3.1 Déplacement à l'intérieur d'un triangle 48

3.3.2 Passage d'un triangle à l'autre 49

3.3.3 Mouvement dans l'espace 50

3.3.4 Détection des collisions entre les gouttes 50

3.4 Rendu 52

4 Résultats et analyse 54

4.1 Vitesse d'exécution 54

4.1.1 Temps global 55

4.1.2 Détection des collisions 58

4.1.3 Calcul des trajectoires 59

4.1.4 Optimisation de la subdivision 60

4.1.5 Remarques générales 61

4.2 Animations 62

4.2.1 Gouttes de masse variable 63

4.2.2 Gouttes de viscosité variable 63

4.2.3 Variation de l'adhérence à la surface 63

4.2.4 Variation de la rugosité de la surface 65

4.2.5 Une larme coule sur une joue 66

4.2.6 L'effet d'un champ de force sur une goutte 67

5 Améliorations et travaux futurs **69**

5.1 Comportement 70

5.1.1 Détection de collisions dans l'espace 70

5.1.2 Évaporation et dépôts 71

5.1.3 Ajouts de niveaux de détails 72

5.2 Traînées 73

5.3 Forme 74

5.4 Rendu 75

6 Conclusion **78**

Bibliographie **81**

Liste des tableaux

4.1 Temps de calcul (en secondes) pour un pas de simulation complète, en excluant le temps de rendu. 56
4.2 Temps de calcul (en secondes) pour un pas de simulation, en excluant la génération de la forme des gouttes. 57
4.3 Temps de calcul (en secondes) pour la détection des collisions entre les gouttes 58
4.4 Temps requis (en secondes) pour le calcul des trajectoires. Nombre de triangles traversés par intervalle : 0 ou 1. 60
4.5 Temps de calcul (en secondes) des trajectoires en fonction du nombre de polygones traversés. 60
4.6 Temps de calcul (en secondes) pour différentes subdivisions, $n = 4000$. . 62

Table des figures

2.1 Une trochoïde 8
2.2 Mouvement orbital de l'eau 9
2.3 Solide de dispersion 23

3.1 Décomposition de la force de gravité s'exerçant sur une particule glissant sur un plan. 29
3.2 L'effet de la rugosité de la surface sur les gouttes : (a) une petite goutte est complètement arrêtée par une aspérité ; (b) une grosse goutte est freinée par une aspérité ; (c) une traînée (représentée par une goutte sur le rebord de l'aspérité) facilite le passage d'une goutte. 30
3.3 Traînée laissée par une goutte 37
3.4 Construction d'une traînée à l'aide de polygones. (a) La traînée est découpée suivant le triangle de support. (b) Les parties à l'extérieur du triangle de support sont projetées sur les triangles voisins. 38
3.5 Modification des coefficients de rugosité. 40
3.6 L'adhérence (a) est plus forte que la force normale, et la goutte reste sur la surface ; (b) est moins forte que la force normale, et la goutte tombera. 41
3.7 Comparaison entre (a) la forme désirée et (b) la forme obtenue en utilisant des transformations affines. 42
3.8 Fusion de deux objets mous (vue en coupe). Des protubérances se forment le long de l'axe reliant les deux gouttes. 43
3.9 Structure du modèle de goutte. 45
3.10 Amplitude de la déformation en fonction de la vitesse de la goutte. . . . 48

3.11 Pseudo-code de l'algorithme de détection de collisions entre les gouttes . 51

4.1 Gouttes de masse variable. 63
4.2 Gouttes de viscosité variable. 64
4.3 Adhérence = 10 N. 64
4.4 Adhérence = 5 N. 64
4.5 Adhérence = 0 N. 65
4.6 Variations de la rugosité de la surface. 65
4.7 Une larme coule sur une joue. 66
4.8 Des gouttes coulent sur un visage. 66
4.9 L'effet d'un champ de force sur une goutte. Le champ de force est dans le plan XZ. Pour chaque valeur de l'angle du champ de force (mesuré par rapport à l'axe X), la goutte est représentée vue de $Y+$ (fond noir) et vue de $Z+$ (fond blanc). 68

5.1 Comparaison d'une demi-sphère déformée par cisaillement et de la forme générée par l'algorithme de Habibi. 76

Remerciements

J'aimerais tout d'abord remercier mon directeur de recherche, Pierre Poulin, de son aide, de ses idées et de son support continu au cours des deux dernières années. Sans lui, je serais probablement encore en train de faire le design d'un système d'animation universel. J'aimerais aussi remercier Arash Habibi et Fabrice Neyret pour leur contribution à mes travaux, ainsi que Cyriaque Kouadio, dont l'aide technique m'a été bien précieuse lorsqu'est venu le temps d'intégrer mon projet dans le système d'animation *Taarna*.

J'aimerais remercier de leurs encouragements et de leur support moral Frances de Verteuil, Véronik de la Chenelière et Myrian Colombo ; sans elles, ce projet serait sans doute resté inachevé.

Finalement, je remercie la société *Taarna* et le CRSNG du support financier qu'ils m'ont apporté. Le laboratoire d'infographie est financé par le CRSNG, la société *Taarna*, l'Université de Montréal et le FCAR.

Conventions de notation

Les vecteurs seront notés à l'aide de lettres grasses. Exemple : une goutte subit une force $\mathbf{F}$.

Les composantes des vecteurs et les scalaires seront notés par des lettres italiques. Exemples :

- F_i est la i-ème composante du vecteur $\mathbf{F}$, $i \geq 1$.
- F est une composante quelconque du vecteur $\mathbf{F}$.
- Au temps t, la goutte est immobile.

L'accélération gravitationnelle sera notée $\mathbf{g}$, et elle vaut $(0; -9{,}8; 0)\ ms^{-2}$.

Le produit vectoriel des vecteurs $\mathbf{u}$ et $\mathbf{v}$, $\begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ u_1 & u_2 & u_3 \\ v_1 & v_2 & v_3 \end{vmatrix}$, sera noté $\mathbf{u} \times \mathbf{v}$.

Le produit scalaire des vecteurs $\mathbf{u}$ et $\mathbf{v}$, $u_1v_1 + u_2v_2 + u_3v_3$, sera noté $\mathbf{u} \bullet \mathbf{v}$.

Chapitre 1

Introduction

Alors Scheherazade, adressant la parole à Schahriar, commença de cette sorte : [...]

Les Mille et Une Nuits

Au cinéma, on fait souvent la distinction entre l'action principale et les actions secondaires. L'action principale est l'événement principal d'une scène, sur laquelle est concentrée l'attention du spectateur. C'est l'ensemble des actions principales qui racontent l'histoire du film. Les actions secondaires, quant à elles, sont là pour supporter l'action principale, la mettre en contexte. Le spectateur les voit mais ne leur porte généralement pas toute son attention.

Les actions secondaires sont habituellement des conséquences d'une autre action. Elles ne doivent pas devenir le centre d'intérêt du spectateur, ni détourner son attention de l'action principale ; en conséquence, elles doivent se dérouler normalement, c'est-à-dire de façon prévisible, en suivant les lois physiques auxquelles le spectateur est habitué. En général, elles ne laissent que peu de place à la créativité du réalisateur.

Dans les films d'animation traditionnels, le traitement réservé aux actions secondaires varie selon le budget alloué à la réalisation du film. Pour un court métrage destiné à la télévision, les actions secondaires seront souvent réalisées à l'aide de techniques demandant peu de travail, parfois même simplistes. D'un autre côté, pour les films d'animation destinés au grand écran, comme par exemple ceux produits par Walt Disney,

une plus grande attention sera portée aux actions secondaires, afin de situer l'action principale dans un contexte plus élaboré. La réalisation de ces actions secondaires est toutefois un travail long ... et coûteux.

Les caractéristiques des actions secondaires nous permettent, dans certains cas, de confier à l'ordinateur le soin de les réaliser. En effet, lorsque les acteurs[1] impliqués dans les actions secondaires suivent un ensemble de lois bien déterminées, nous pouvons programmer ces lois et demander à l'ordinateur de simuler l'évolution des objets. Les dessinateurs peuvent ainsi se consacrer à des tâches plus créatives et moins répétitives. La qualité de l'animation augmente car le réalisateur peut se permettre, pour un même budget, des actions secondaires plus détaillées.

Certaines actions secondaires sont particulièrement difficiles à réaliser à la main. Ce sont habituellement des actions dans lesquelles le nombre d'acteurs est grand et les déplacements complexes, comme les foules ou l'Hydre du film *Hercules*, de Walt Disney. Dans ces cas, l'utilisation de l'ordinateur s'avère encore plus intéressante, car en plus de gagner du temps, on peut gagner en qualité.

Les phénomènes naturels forment une classe d'actions secondaires à laquelle les chercheurs en infographie se sont beaucoup intéressés au cours des quinze dernières années. Parmi les modèles proposés, on compte ceux sur la représentation et la modélisation d'arbre, la modélisation de l'effet du vent sur les champs d'herbe et la modélisation des vagues.

L'écoulement de gouttes de liquide, comme l'eau déposée sur une vitre par la pluie, la sueur sur la peau ou des gouttes de peinture sur un mur, est un phénomène naturel qui a reçu peu d'attention de la part des chercheurs. Pourtant, c'est un phénomène difficile à animer à la main. Le déplacement des gouttes suivant les lois de la physique, ce phénomène est un bon candidat pour l'animation assistée par ordinateur.

Simuler l'écoulement de gouttes sur des surfaces n'est pas une tâche facile, si l'on veut que cela soit fait de façon efficace et réaliste. Dans l'espace, nous n'aurions qu'à tenir compte de la gravité et des détections de collisions avec l'environnement ; sur une surface, cependant, nous devons aussi tenir compte de la rugosité de la surface et de la viscosité

[1] Par acteurs, nous entendons aussi bien les personnages que les objets inanimés.

du liquide. Nous devons simuler l'adhérence des gouttes à la surface et les traînées de liquide que les gouttes laissent derrière elles. La simulation est compliquée par le fait que souvent, pour faire une simulation réaliste, nous aurons besoin de milliers de gouttes. Si nous détectons les collisions entre les gouttes et les objets de façon traditionnelle, c'est-à-dire en déplaçant les gouttes et en intersectant leur trajectoire avec les autres objets, nous passerons un temps incroyable à cette tâche. Nous devons donc utiliser un algorithme efficace de détection de collision, pouvant détecter des milliers de collisions sans une forte pénalité.

La simulation de la forme des gouttes de liquide est elle aussi compliquée par le fait que les gouttes se trouvent sur une surface. Dans l'espace, nous pourrions très bien utiliser des sphères légèrement étirées. Sur une surface, cependant, les gouttes peuvent être déformées par la capillarité et par la courbure de la surface (entre autres). Nous devons donc trouver un modèle qui rend bien ces déformations.

Jusqu'à maintenant, aucun modèle rapide et complet d'animation, simulant à la fois le déplacement et la forme des gouttes, n'avait été proposé. Plusieurs chercheurs se sont cependant penchés sur des sujets connexes, tels la modélisation de vagues [Pea86] [FR86] [TB87] [KM90], la modélisation de phénomènes naturels à l'aide de systèmes à particules [Ree83] [RB85] [YUM86] [Rey87] [Sim90] [WH91], la modélisation d'objets mous [WMW86a] [WMW86b] [MP89] et les changements d'apparence des objets causés par un écoulement répété de liquide [DPH96]. Nous verrons en détail au chapitre 2 ce que ces modèles ont apporté à la modélisation des liquides.

Nous présenterons ensuite au chapitre 3 l'approche que nous avons adoptée pour réaliser un système de simulation qui soit à la fois rapide et réaliste. Nous discuterons des résultats que nous avons obtenus au chapitre 4. Puis, nous terminerons en suggérant un certain nombre de pistes à suivre pour améliorer et augmenter les fonctionnalités de notre modèle.

Chapitre 2

Travaux antérieurs

[. . .] les nez ont été faits pour porter des lunettes, aussi avons-nous des lunettes.

Voltaire, *Candide ou l'optimisme*

Peu de chercheurs se sont penchés sur le problème de la modélisation et de l'animation de gouttes de liquide à la surface des objets. Seuls Dorsey *et al.* [DPH96] ont publié les résultats de leurs travaux, qui se rapprochent en partie des nôtres. Cependant, plusieurs chercheurs se sont intéressés à des sujets connexes, qui peuvent nous apporter des éléments intéressants de solution. Pensons à la modélisation de masses d'eau, la modélisation à l'aide de systèmes à particules et ses dérivés (appliqués avec un certain succès à la modélisation de plusieurs phénomènes naturels) et la modélisation d'objets mous.

Dans ce chapitre, nous ferons un bref survol des problèmes qui se posent lorsqu'on veut animer des objets rigides et des solutions qui ont été proposées. Nous verrons ensuite quelles approches ont été suggérées pour l'animation d'objets non rigides. Nous nous intéresserons particulièrement à la modélisation des vagues et à la modélisation à l'aide de systèmes à particules. Nous verrons les possibilités et avantages que présente un tel modèle ; nous verrons aussi quelques dérivés des systèmes à particules, comme les systèmes de modélisation par comportement et par messages. Ensuite, nous présenterons différentes approches de la modélisation de gouttes d'eau, entre autres celle de Dorsey

et al. [DPH96].

Nous consacrerons l'avant-dernière section au rendu de l'eau. Nous verrons les approches adoptées par différents chercheurs pour rendre les résultats des techniques d'animation visuellement intéressants. Finalement, nous présenterons le système d'animation de la société *Taarna*, système dans lequel s'insère notre programme.

2.1 Animation d'objets rigides

Traditionnellement, les films d'animation sont réalisés image par image ; pour chaque image, on place les modèles (personnages, décors, etc.) et la caméra, puis on prend une photo. La juxtaposition dans le temps de toutes les photos donnera une illusion de mouvement. C'est un processus long qui demande une bonne expérience de la part des animateurs afin que le mouvement obtenu soit fluide et réaliste. Depuis quelques années, on utilise de plus en plus l'ordinateur pour réaliser des animations. L'ordinateur offre de nombreux avantages par rapport aux techniques traditionnelles. D'abord, les modèles n'ont pas à être construits physiquement ; aucune contrainte ne s'applique donc sur leur forme ou leur position. Il en est de même pour la caméra, qui peut être placée et déplacée selon le bon vouloir de l'animateur. L'utilisation de l'infographie en animation offre un autre avantage, plus important encore que les précédents : l'ordinateur peut assister l'animateur dans le positionnement des modèles et de la caméra, afin de créer un mouvement fluide ; on peut ainsi « filmer » une scène plus rapidement qu'en utilisant les techniques traditionnelles.

Les premiers logiciels d'animation à être apparus sont les systèmes à scripts, dans lesquels les mouvements des objets et de la caméra sont programmés à l'aide d'un langage de haut niveau par un animateur-programmeur. L'ordinateur se charge d'interpréter les commandes du script pour trouver la position des objets de la scène. La réalisation d'animations réalistes (*i.e.* dont les acteurs se comportent en conformité avec les lois physiques de notre monde), n'est pas beaucoup plus facile qu'avec les techniques traditionnelles. De plus, cette méthode ne s'applique bien que lorsque l'on a un nombre restreint d'objets à animer ; il serait très laborieux d'écrire un script décrivant le mouvement de centaines

ou de milliers d'objets qui interagissent entre eux.

Après les systèmes à scripts sont apparues les techniques d'interpolation d'images clefs. Ces techniques s'inspirent de la façon dont les dessins animés traditionnels sont réalisés. Un animateur conçoit une scène et dessine une série d'images clefs représentant les étapes importantes de l'action. Ensuite, d'autres artistes se chargent d'interpoler ces images pour former un mouvement continu. Grâce à des techniques développées en infographie, l'ordinateur peut maintenant faire ces interpolations. Un animateur spécifie la valeur de différents paramètres (position, vitesse, couleur, etc.) à des moments clefs et l'ordinateur, le plus souvent à l'aide de splines cubiques, se charge de générer les positions intermédiaires. Encore une fois, cette technique s'applique difficilement lorsqu'on veut contrôler un grand nombre d'objets. De plus, on doit utiliser différentes techniques d'interpolation pour animer différent types d'objets.

Une autre technique est apparue avec l'utilisation de l'ordinateur en animation. Elle utilise la physique dynamique (ou tout autre ensemble de lois) pour déterminer la position des objets. Au début de l'animation, l'animateur spécifie les conditions initiales du système (positions, vitesses, forces), puis l'ordinateur s'occupe de calculer la position des objets pour chacune des images de l'animation. Cette technique s'applique bien lorsque le nombre d'objets est peu élevé ; cependant, l'animateur perd presque tout contrôle sur le comportement des objets durant l'animation, leurs positions étant uniquement déterminées par les forces appliquées sur les objets et l'ensemble des lois du système. Cette technique est habituellement utilisée pour modéliser des actions secondaires d'une scène, actions dont le déroulement exact n'est pas aussi important que celui de l'action principale.

Ces techniques et leurs variations (voir [WW92], chapitres 15 et 16, et [FvDFH90], chapitre 21, pour une discussion plus en profondeur des techniques d'animation d'objets rigides) forment la base de l'animation d'objets rigides par ordinateur. Les deux premières techniques, les scripts et l'interpolation d'images clefs, s'appliquent difficilement à la modélisation de gouttes d'eau si celles-ci sont présentes en grand nombre ; le travail de l'animateur serait beaucoup trop fastidieux. Quant à la technique d'animation dynamique, qui automatise totalement le déplacement des objets, elle offre des carac-

téristiques intéressantes pour résoudre le problème qui nous préoccupe. Cependant, il y a une différence importante entre les objets rigides animés et nos gouttes d'eau. En effet, les objets rigides ne changent pas de forme au cours de l'animation, alors que les gouttes d'eau, en glissant sur une surface, laissent des traînées, changent de forme et se fusionnent avec d'autres gouttes. Si nous utilisons une technique d'animation dynamique, elle devra être modifiée afin de tenir compte de ces comportements.

Plusieurs chercheurs se sont intéressés à la modélisation de fluides. Parmi les premiers travaux réalisés, nombreux sont ceux qui traitent de la simulation des vagues de grandes masses d'eau. C'est ce dont nous discuterons à la prochaine section.

2.2 Modélisation de masses d'eau

Durant les années 1980, la communauté infographique a été prise d'un grand engouement pour la modélisation des phénomènes naturels, et l'océan n'y a pas échappé. Plusieurs chercheurs se sont intéressés à la modélisation de vagues ; parmi les modèles proposés, on note ceux de Peachey [Pea86], de Fournier et Reeves [FR86], de Ts'o et Barsky [TB87] et de Kass et Miller [KM90]. En ce qui a trait à la modélisation de gouttes d'eau, les modèles de vagues sont intéressants pour les équations qui gouvernent le déplacement de l'eau, leurs techniques de rendu (dont nous parlerons à la section 2.5.2) et de modélisation des embruns (lorsque le modèle en tient compte). On peut diviser les modèles de vagues en deux groupes. Le premier, que nous détaillerons dans la section qui suit, utilise les équations d'onde pour construire les vagues, alors que le second, que nous verrons à la section 2.2.2, se base sur la dynamique des fluides.

2.2.1 Modèle de Gerstner–Rankine simplifié

Peachey [Pea86], Fournier et Reeves [FR86] et Ts'o et Barsky [TB87] ont tous présenté un modèle s'appuyant sur le modèle de vague de Gerstner–Rankine [Kin65, Ach90], c'est-à-dire une description sinusoïdale de vagues de petites amplitudes pour un fluide irrotationnel, incompressible et non visqueux. À la base de ce modèle, on retrouve l'équa-

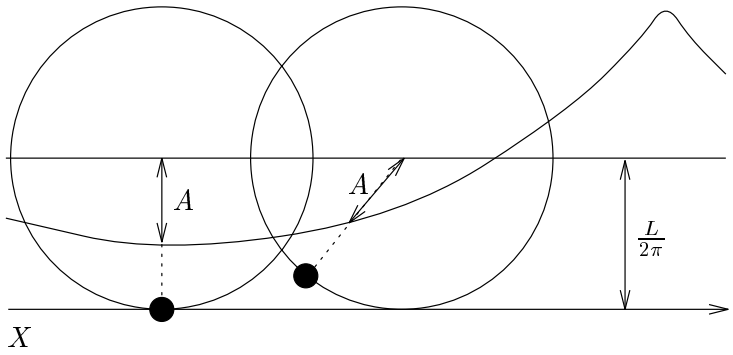


FIG. 2.1: Une trochoïde

tion

$$f(x,t) = A\cos\left(\frac{2\pi(x-Ct)}{L}\right) \tag{2.1}$$

où x est la distance du point d'origine, A est l'amplitude, L est la longueur d'onde et C la vitesse de propagation du train de vagues. Ts'o et Barsky déterminent la trajectoire des vagues en faisant un tracé des vagues. Tout d'abord, ils cartographient le fond marin en traçant ses courbes de niveau. Puis, ils lancent des rayons perpendiculaires au front de vagues ; lorsque ces rayons intersectent une des courbes de niveau du fond marin, il sont réfractés (ainsi que la vague) suivant la loi de Snell-Descartes, de la même façon qu'un rayon de lumière est réfracté lorsqu'il change de milieu. La profondeur du fond joue le rôle d'indice de réfraction dans les calculs. Une fois toutes les réfractions calculées, on connaît la position des vagues et on peut calculer la hauteur de l'eau en différents points de la surface. Puisque la surface de l'eau est modélisée par un champ $f : \mathbb{R}^2 \mapsto \mathbb{R}$, Ts'o et Barsky ne peuvent pas créer de vagues qui se brisent ; il n'y a donc pas, contrairement à ce qui se produit dans les modèles de Peachey et Fournier-Reeves, de production d'embruns.

Peachey [Pea86] et Fournier et Reeves [FR86] considèrent l'équation 2.1 comme la description d'une trochoïde, c'est-à-dire une courbe générée par un point situé à une distance A du centre d'un cercle de rayon $L/2\pi$ roulant sur une ligne, considérée comme l'axe X (figure 2.1).

Suivant ce modèle, le déplacement net d'eau causé par une vague est nul. Lorsqu'une vague passe en un point, l'eau se déplace en suivant un mouvement circulaire, et l'eau

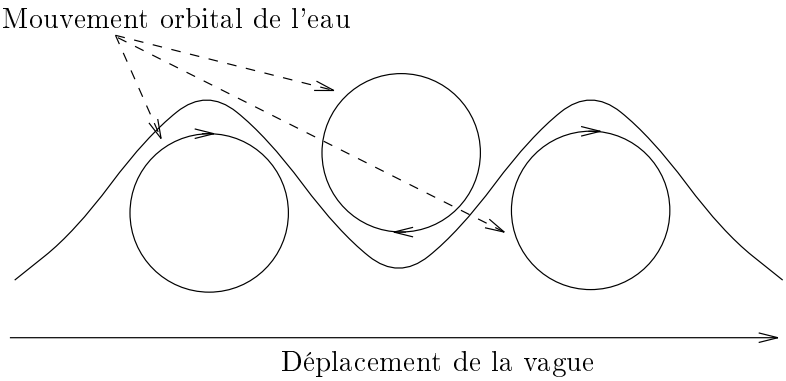


FIG. 2.2: Mouvement orbital de l'eau

doit parcourir une orbite complète entre deux sommets de vague (figure 2.2). La vitesse orbitale moyenne est donc

$$\bar{v} = \frac{\pi H}{T} = \pi SC \tag{2.2}$$

où H est le diamètre de l'orbite, T la période de la vague, C, sa vitesse et $S = H/L$, sa pente. Si la vague se déplace de gauche à droite, le mouvement circulaire s'effectue dans le sens des aiguilles d'une montre. Une vague se brise lorsque la vitesse orbitale instantanée de l'eau au sommet de la vague est plus grande que la vitesse de propagation des vagues (ou, ce qui est une condition équivalente d'après l'équation 2.2, si l'inclinaison de la vague est trop abrupte).

À partir d'ici, les modèles de Peachey et de Fournier-Reeves diffèrent. Peachey se sert assez directement du modèle, tandis que Fournier et Reeves le modifient en lui ajoutant quelques paramètres dans le but de réaliser certains effets intéressants, mais qui dépassent le cadre de la présente discussion.

Dans les modèles de Peachey et de Fournier et Reeves, il y a création d'embruns. Dans le modèle de Peachey, ceux-ci se forment lorsque les vagues se brisent. Dans le modèle de Fournier-Reeves, il y a une condition supplémentaire : il faut que la vitesse orbitale de l'eau au sommet des vagues dépasse un certain seuil. Si ce n'est pas le cas, la vague se brise et seule l'écume apparaît. Dans les deux modèles, les embruns sont modélisés à l'aide d'un système à particules. Les particules sont générées au sommet de la vague avec une vitesse initiale proche ou égale à la vitesse orbitale de la vague,

puis animées par les lois de la physique (gravitation, frottement de l'air, etc.). Nous reviendrons plus en détail sur les systèmes à particules à la section 2.3.

2.2.2 Modèle hydrodynamique

Kass et Miller [KM90] proposent de leur côté un modèle de vague s'inspirant des équations de base de la dynamique des fluides, les équations de Navier-Stokes. Ces équations décrivent le mouvement d'un fluide incompressible de densité ρ et de viscosité μ constantes. Ce sont :

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \bullet \nabla)\,\mathbf{u} = -\frac{1}{\rho}\nabla p + \frac{\mu}{\rho}\nabla^2 \mathbf{u} + \mathbf{g}$$

$$\nabla \bullet \mathbf{u} = 0$$

où $\mathbf{u}$ est la vitesse d'un élément de liquide, p est la pression et ∇ est l'opérateur de gradient $\left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}\right)$. La première équation est simplement l'expression de la troisième loi de Newton, *pour toute action il y a une réaction*, et elle met en relation toutes les forces qui s'exercent sur un morceau de liquide. La seconde équation exprime l'incompressibilité du liquide.

Kass et Miller simplifient les équations de Navier-Stokes afin d'obtenir l'équation différentielle linéaire décrivant une vague unidimensionnelle (*i.e.* une corde ondulée) :

$$\frac{\partial^2 h}{\partial t^2} = \frac{\partial^2 h}{\partial x^2} g d \tag{2.3}$$

où $d(x) = h(x) - b(x)$, $h(x)$ étant la hauteur de la surface de l'eau, $b(x)$ la hauteur du fond marin et $d(x)$ la profondeur de l'eau. Ils discrétisent ensuite cette équation par la méthode des différences finies. Ils se retrouvent ainsi avec une matrice tridiagonale leur donnant la hauteur de l'eau au temps t_n à partir de la hauteur de l'eau aux temps t_{n-1} et t_{n-2}. Leur méthode est stable, rapide et permet de modéliser des effets dont les modèles de Peachey, Fournier et Reeves et Ts'o et Barsky ne tenaient pas compte, comme la réflexion des vagues sur des rebords verticaux, les déplacements d'eau et les variations des conditions aux bornes dues à un changement de la topologie. Pour modéliser une vague bidimensionnelle (une surface ondulée), ils remplacent l'équation 2.3 par

$$\frac{\partial^2 h}{\partial t^2} = g d \nabla^2 h = g d \left(\frac{\partial^2 h}{\partial x^2} + \frac{\partial^2 h}{\partial y^2}\right) \tag{2.4}$$

qu'ils résolvent en la séparant en deux équations, une en x et l'autre en y, et en itérant alternativement en x et en y.

Cependant, il leur est impossible de modéliser des rouleaux et des vagues qui se brisent, car une des simplifications à la base de leur modèle est que la surface de l'eau doit être décrite par un champ $f : \mathbb{R}^2 \mapsto \mathbb{R}$.

Foster et Metaxas [FM96] ont eux aussi présenté un modèle basé sur la résolution des équations de Navier-Stokes. Leur modèle permet de modéliser une grande variété de phénomènes, comme la chute d'une goutte dans l'eau, des vagues se brisant sur la rive et le remplissage d'un bassin par un jet d'eau. Afin que leur méthode soit rapide, Foster et Metaxas résolvent les équations de Navier-Stokes en utilisant une discrétisation grossière. Puis, pour obtenir une bonne qualité d'image, ils génèrent une représentation de la surface du liquide à partir des résultats obtenus sur la discrétisation grossière. Ils proposent trois méthodes pour définir cette surface.

La première, une méthode volumétrique semblable à celle utilisée par Nishikawa et Abe [NA91] (voir section 2.4.1), consiste à introduire dans le liquide des particules sans masse, appelées marqueurs. Ainsi, une cellule de la grille de discrétisation ne contenant aucun marqueur est une cellule vide ; une cellule contenant au moins un marqueur et qui est adjacente à une cellule vide est une cellule contenant la surface du liquide ; une cellule contenant au moins un marqueur sans être adjacente à une cellule vide est une cellule pleine.

La deuxième méthode, surfacique, utilise aussi des marqueurs, mais sous forme de grille représentant la surface du liquide. Les marqueurs sont déplacés sous l'effet de la vitesse du liquide et de la pression. Si deux marqueurs se rapprochent trop, l'un d'eux est éliminé ; si deux marqueurs s'éloignent trop l'un de l'autre, un nouveau marqueur est inséré entre les deux.

Les deux méthodes précédentes sont appropriées même lorsque le liquide fait des éclaboussures et retombe sur lui-même. Cependant, lorsque le liquide est plus calme, on peut utiliser un champ $f : \mathbb{R}^2 \mapsto \mathbb{R}$ pour déterminer la position de la surface. C'est la troisième méthode proposée. La hauteur de l'eau est déterminée en résolvant une équation différentielle mettant en jeu la vitesse du liquide.

Il est intéressant de noter que l'animateur n'a pas besoin de décrire l'environnement (les obstacles) à l'algorithme de simulation. Celui-ci est capable de les inférer à partir de la géométrie de l'environnement et de les insérer dans le système d'équations différentielles sous forme de contraintes. L'animateur a aussi pleins pouvoirs pour décider de la vitesse et de la pression initiale du liquide, de même qu'il peut décider d'ajouter ou d'enlever du liquide, ou d'appliquer une pression à la surface du liquide. Le modèle de Foster et Metaxas permet aussi de faire flotter des objets dans le liquide, en utilisant les résultats obtenus de la résolution des équations de Navier-Stokes pour déterminer la force agissant sur un objet. Cependant, les objets flottant dans le liquide n'étant pas pris en compte dans leur modèle de flot, ceux-ci n'influencent pas le mouvement du liquide.

Les méthodes de modélisation de masse d'eau ont donné des résultats intéressants, mais nous croyons qu'il serait soit difficile, soit beaucoup trop coûteux de les appliquer à la modélisation de plusieurs petites gouttes d'eau. Nous avons cependant vu une piste intéressante dans la technique retenue par Peachey [Pea86] et Fournier et Reeves [FR86] pour modéliser les embruns : la technique du système à particules.

2.3 Systèmes à particules

La modélisation à l'aide de systèmes à particules a été introduite en infographie par Reeves en 1983 [Ree83] dans le but de modéliser des objets flous, mal définis ou irréguliers, et d'animer ces objets de façon plus complexe que par une simple transformation affine. Ces systèmes ont eu un grand succès dans la modélisation de phénomènes naturels et de phénomènes complexes, tels que le feu, les chutes d'eau et les feux d'artifice. Plus tard, d'autres chercheurs ont ajouté des éléments au système de Reeves afin de rendre l'animation des particules plus sophistiquée. Nous verrons dans cette section le modèle à particules tel que présenté par Reeves, puis les extensions proposées subséquemment, comme les comportements et les messages. Nous présenterons de plus diverses caractéristiques mathématiques, physiques et structurelles associées aux particules.

2.3.1 Le système à particules de base

Un système à particules est une méthode de modélisation qui utilise des points de matière — les particules — comme primitive, tout comme les polygones servent de primitive pour modéliser les objets plus réguliers (meubles, édifices, voitures). L'intérêt principal du système à particules est sa grande flexibilité et la facilité avec laquelle on peut modéliser et animer des phénomènes complexes. En effet, la simplicité d'une particule nous permet de l'utiliser en grande quantité, ce qui donne une impression visuelle de complexité. On peut donc utiliser un système à particules pour visualiser des phénomènes comme le déplacement de l'eau dans une rivière, un feu ou une explosion. L'algorithme à la base d'un système à particules, tel que présenté par Reeves, effectue plusieurs itérations des quatre étapes suivantes :

1. Générer les nouvelles particules et fixer leurs attributs ;
2. Enlever les vieilles particules du système ;
3. Animer les particules durant un temps Δt ;
4. Faire le rendu du système.

Nous allons détailler quelque peu les trois premières étapes. La dernière étape sera traitée à la section 2.5, consacrée au rendu.

Génération des particules. L'étape de la génération des particules est celle de la modélisation. Elle peut être faite une seule fois, au début de la simulation [YUM86], ou elle peut être répétée à chaque Δt [Ree83], si l'on désire que des éléments nouveaux s'ajoutent continuellement. La plupart du temps, la génération des particules se fait de façon stochastique, pour éliminer toute régularité et maximiser l'aspect complexe et « naturel » du phénomène modélisé.

Lors de la génération des particules, on initialise les attributs de ces dernières. L'attribut de base d'une particule est sa position, mais un ensemble d'autres propriétés s'ajoutent habituellement. Les attributs que l'on choisira pour nos particules dépendent beaucoup du phénomène que l'on désire modéliser. Reeves, qui modélise une explosion, leur donne en plus une masse, une vitesse, une couleur, un âge, une durée de vie, une forme et une transparence.

Élimination de particules. Il peut être souhaitable, durant une simulation, d'éliminer certaines des particules. Le critère d'élimination variera selon le phénomène modélisé par la disparition des particules. Reeves se base sur leur âge ; dès que l'âge d'une particule dépasse sa durée de vie, elle est éliminée du système. On peut aussi vouloir éliminer des particules si celles-ci sortent du champ de la caméra, si elles sortent des limites du monde défini, ou si elles entrent en collision et se regroupent (on remplacera alors un groupe de particules par une seule particule). Il va sans dire que l'élimination des particules est une étape cruciale si l'on veut que le système reste rapide à long terme.

Animation. La phase d'animation consiste à mettre à jour les caractéristiques des particules qui évoluent avec le temps (position, couleur, etc.). L'algorithme d'animation est donc très général ; le plus souvent, une fonction $f(t)$ est définie pour chacune des caractéristiques et est appliquée par l'algorithme d'animation. Le choix de $f(t)$ est totalement arbitraire. Il dépend uniquement du type d'animation que l'on veut réaliser. Si on veut une animation réaliste, on dérivera la fonction des lois physiques. Sinon, on invente une fonction en espérant que le résultat sera intéressant.

Dans le but de faire des animations réalistes, donc basées sur les lois de la physique, tout en gardant un certain contrôle sur le comportement des particules, Sims [Sim90] a proposé un système d'animation dans lequel sont définies un ensemble de primitives de vitesse et d'accélération, comme la rotation, le vortex, l'amortissement et le rebond, pour n'en nommer que quelques unes. Ces primitives sont disposées dans l'espace, de façon à former des champs de force qui peuvent être appliqués à un groupe de particules. On obtient ainsi un contrôle général du déplacement des particules sans avoir à spécifier leur mouvement individuel et sans trop diminuer l'aspect complexe de l'animation. Wejchert et Haumann [WH91] utilisent un système semblable pour animer des feuilles mortes dans le vent.

Le système à particules de base a été utilisé pour modéliser plusieurs phénomènes et objets complexes, comme les feux d'artifice, le *Genesis Effect* du film *Star Trek II : The Wrath of Khan* [Ree83], de la neige, des chutes d'eau, du feu [Sim90], les embruns produits

par les vagues lorsqu'elles se brisent [FR86] (cf. section 2.2.1) et même une simulation de la planète Jupiter pour le film *2010* [YUM86]. Dans ces systèmes, chaque particule est indépendante des autres. Il est possible de construire des systèmes plus complexes, dans lesquels les particules sont interdépendantes. C'est ce dont nous traiterons à la section suivante.

2.3.2 Structure, comportements et messages

Les systèmes d'animation présentés jusqu'à maintenant utilisaient un grand nombre de particules indépendantes animées à l'aide de règles générales comparables aux lois physiques de notre monde. Il est cependant possible de simuler des échanges d'information entre les entités et, à un plus haut niveau, des comportements en faisant tout simplement interagir les particules entre elles.

Reeves et Blau [RB85] présentent les premiers un système à particules structuré, construit à partir du système présenté par Reeves [Ree83]. Ce nouveau système sert à générer des brins d'herbes et des arbres et à modéliser leur mouvement dans le vent. La génération des particules se fait en deux étapes. La première consiste à placer les plantes sur le terrain, ce qui se fait en plaçant des « graines » de façon aléatoire ou ordonnée, selon que l'on désire une forêt ou un verger. Ces graines sont ensuite développées en plantes. Pour faire pousser un arbre, Reeves et Blau génèrent un arbre de particules, en partant du tronc et en allant jusqu'aux feuilles ; la taille, la couleur et l'emplacement d'une particule enfant sont déterminés de façon stochastique à partir de la taille, la couleur et l'emplacement de la particule parent. Dans le cas de l'herbe, chaque brin est modélisé par un groupe de particules, dans lequel chaque particule représente une section du brin.

Pour modéliser l'effet du vent sur les brins d'herbe, un autre système à particules est utilisé, où chaque particule représente un souffle de vent. Avec cela, Reeves et Blau construisent pour chaque image de l'animation une carte de vent bidimensionnelle, qui contient l'intensité du vent à chaque endroit du terrain. Le vent est ensuite appliqué à chaque brin d'herbe, déplaçant les particules qui le composent de façon à le courber. L'effet du vent sur une particule du brin d'herbe est proportionnel à l'intensité du vent

et à la distance particule – base du brin d'herbe. Lorsque l'intensité du vent diminue, la force sur les particules du brin d'herbe diminue aussi et ce dernier revient à sa position initiale.

Reynolds [Rey87] modélise le vol d'oiseaux, les troupeaux d'animaux et les bancs de poissons en utilisant un système à particules dans lequel les particules (les animaux) peuvent connaître la position des particules voisines et modifier leur mouvement afin de rester près d'elles. Une particule a une perception de l'environnement limitée à son entourage immédiat. Elle ne perçoit donc pas tous les membres du troupeau, mais seulement ceux qui se trouvent près d'elle. Les règles d'attroupement, qui peuvent être considérées comme une simulation du comportement animal, sont : éviter une collision avec les voisins ; aller à la même vitesse que les voisins ; se maintenir au centre des particules voisines. Les particules peuvent être groupées en hiérarchie, de sorte qu'il est possible de diriger le troupeau en entier en modifiant manuellement les paramètres du « chef » du troupeau.

On voit que plus les systèmes à particules deviennent sophistiqués, plus les particules ressemblent à des objets, au sens informatique du terme. D'ailleurs, plusieurs des systèmes présentés jusqu'à maintenant ont été développés à l'aide d'un langage supportant la notion de classe et d'objet. Il en est ainsi pour le système que présentent Haumann et Parent [HP88], dans lequel ils définissent un ensemble de comportements pour chacun des éléments d'une scène, de façon à ce que le mouvement des éléments émerge des règles comportementales. Les éléments d'une scène ainsi dotés de comportements se nomment « acteurs ». Les acteurs sont hiérarchisés, et chaque super-acteur est responsable d'animer ses sous-acteurs. Un super-acteur, à la racine de l'arbre contenant tous les acteurs, est le grand coordonnateur de la scène. Dans le système de Haumann et Parent, cet acteur s'appelle l'*acteur simulation*. La boucle d'animation de chaque acteur se résume à :

```
pour chaque pas de temps
    agir
    répondre
    injection de données
fin
```

Par exemple, supposons que l'on ait dans une scène trois acteurs `force` reliés à un

acteur `masse`, et que ces quatre acteurs soient les sous-acteurs de l'acteur `simulation`. À l'étape `agir`, le super-acteur appellera la fonction `agir` de chacun des acteurs `force`, qui communiqueront leur force à l'acteur `masse` ; à l'étape `répondre`, l'acteur `masse` modifiera sa vitesse et sa position en réponse aux trois acteurs `force` qui ont agi sur elle. L'étape `injection de données` est une étape de script, permettant à l'animateur de modifier les valeurs des acteurs de façon totalement arbitraire.

Haumann et Parent ont utilisé leur système pour créer et animer un modèle d'objet flexible (*e.g.* une balle, du papier, du tissu). L'objet est composé d'acteurs `masse` interconnectés par des acteurs `ressort` et des acteurs `charnière`. L'environnement est composé des acteurs `gravité`, `sol` et `contact`, qui agissent sur les acteurs `masse` pour les faire tomber et rebondir.

Signalons enfin que Breen et Kühn [BK89] présentent un système très semblable à celui de Haumann et Parent, et qu'ils l'utilisent pour modéliser le comportement d'objets géométriques interconnectés par des ressorts.

Les systèmes à particules structurés sont bien adaptés pour la modélisation d'objets flexibles. Les techniques que nous avons vues dans cette section représentent un objet flexible à l'aide d'un ensemble de particules reliées entre elles. Les forces agissant sur les particules et les communications inter-particules déterminent la forme de l'objet flexible.

2.3.3 Modélisation d'objets mous

Une autre approche a été suggérée par Wyvill *et al.* pour la modélisation d'objets flexibles [WMW86a] [WMW86b]. Cette approche est plus appropriée pour les objets faits de matériaux pâteux ou visqueux (que nous appellerons *objets mous*). Ils s'inspirent des travaux de Blinn [Bli82] et de Nishimura *et al.* [NHK$^+$85] sur les surfaces implicites [Blo97] pour construire un modèle utilisant des champs pour déterminer la position de l'objet dans l'espace.

Comme dans les systèmes à particules habituels, les objets sont définis à l'aide d'un groupe de particules, appelés points de référence. À ces points de référence sont associés des champs. N'importe quelle fonction $f : \mathbb{R}^3 \mapsto \mathbb{R}$ peut faire office de champ ; cependant, les résultats sont plus intéressants lorsque la fonction choisie décroît à mesure que l'on

s'éloigne de la particule. La fonction utilisée par Wyvill *et al.* est

$$C(r) = 1 - \frac{4r^6}{9R^6} + \frac{17r^4}{9R^4} - \frac{25r^2}{9R^2}$$

où r est la distance entre le point pour lequel on calcule le champ et le point de référence et R est le rayon d'influence du point de référence, au-delà duquel le champ devient nul. Lorsque tous les points définissant un objet sont placés, leurs champs sont additionnés et la surface de l'objet est donnée par une isosurface de la somme des champs. Le rendu de cette surface peut être fait par tracé de rayon [KH84] ou en polygonisant l'isosurface à l'aide de l'algorithme du *marching cube* [UK88].

Pour animer ces objets, Wyvill *et al.* proposent plusieurs techniques. L'une d'elles consiste à interpoler la position des points de référence, de la même façon que les animateurs traditionnels interpolent les images clefs. Cette technique est particulièrement appropriée pour changer la forme d'un objet de façon arbitraire. Une variation de cette technique consiste à spécifier des chemins pour guider l'interpolation des points de référence. Une autre technique d'animation consiste à déplacer les points de référence à l'aide d'une simulation physique ou mathématique, de la même façon que le font les systèmes à particules (voir section 2.3).

Miller et Pearce [MP89] ont développé un système versatile pour la modélisation de liquides visqueux et de solides fusibles. À la base de leur système se trouve le globule, une particule qui interagit avec les particules voisines, avec laquelle ils modélisent les objets fluides ou fusibles de la scène. L'interaction entre les globules se fait sous la forme de force de répulsion/attraction et de force de traînée (cette force simule la viscosité du fluide). La formule exprimant la force qu'exerce une particule sur une autre est assez complexe, et nous ne nous y attarderons pas ; mentionnons simplement qu'elle s'inspire des lois physiques. À ces forces inter-globules s'ajoutent les contraintes externes, qui permettent aux globules d'interagir avec leur environnement, notamment de rebondir sur les obstacles. Pour trouver la position d'un globule, la somme des forces agissant sur un globule est intégrée deux fois par rapport au temps. Les méthodes d'intégration utilisées sont celles d'Euler et de Runge-Kutta. Pour que ces méthodes soient stables, les globules ne doivent se déplacer que d'une fraction de leur rayon à chaque itération ; il est donc nécessaire de faire plusieurs pas de simulation pour passer d'une image à une

autre (jusqu'à trente pas par image, dans les exemples présentés par Miller et Pearce). De plus, si l'on veut augmenter la qualité des images en augmentant la résolution des objets, on doit réduire le rayon des globules, ce qui se traduit par une augmentation du nombre de pas de simulation.

2.4 Modélisation de gouttes d'eau

Nous avons constaté que très peu de résultats ont été publiés sur la modélisation de gouttes d'eau. Pourtant, celles-ci sont présentes dans une variété de phénomènes de la vie de tous les jours. Peu d'équipes ont étudié le phénomène ; Nishikawa et Abe [NA91] se sont intéressés au comportement d'une goutte en chute libre faisant contact avec une surface, Curtis *et al.* [CAS+97] modélisent les effets de l'aquarelle à l'aide d'équations décrivant le comportement de liquides peu profonds, alors que Dorsey *et al.* [DPH96] ont modélisé la patine créée par la pluie.

2.4.1 Dynamique des fluides

Nishikawa et Abe [NA91] emploient la méthode de la force brute pour modéliser et visualiser la chute de gouttes sur une surface solide ou liquide. Ils utilisent les équations de Navier-Stokes, qu'ils résolvent par la méthode des différences finies sur de puissants ordinateurs. À l'aide de marqueurs (voir Foster et Metaxas, section 2.2.2), ils construisent la surface de la goutte et en font le rendu. Cette méthode a l'avantage d'être très réaliste au point de vue physique. Cependant, elle devient extrêmement coûteuse si on l'applique à des centaines ou des milliers de gouttes, chaque goutte devant être subdivisée en environ 2500 cellules.

Curtis *et al.* [CAS+97] ont présenté un modèle pour simuler les effets graphiques créés par l'aquarelle. Leur modèle est basé sur les équations de flot des liquides peu profonds. Dans leur modèle, l'eau se déplace à la surface du papier en y déposant les pigments de couleur. Les effets de diffusion des pigments sont simulés par une modélisation de l'absorption de l'eau par le papier.

2.4.2 Écoulement de gouttes

Les pierres des vieux édifices prennent avec le temps un aspect usé et sale. La saleté et les sédiments causant cette patine proviennent de plusieurs sources : pollution atmosphérique, poussière au sol, oxydation, dissolution et dépôt de matériau par la pluie. Sans la pluie, qui glisse sur la surface en petits torrents, les sédiments se déposeraient uniformément sur la surface de l'objet. Cependant, sous l'effet de la pluie, les sédiments sont dissouts en certains endroits et sont déposés ailleurs.

Dorsey *et al.* [DPH96] se sont penchés sur la simulation de la patine que prennent les objets sous l'effet des phénomènes météorologiques, patine que l'on peut voir surtout sur les vieux édifices en pierre ou les constructions en métal oxydé. Pour modéliser l'écoulement de la pluie sur les édifices, ils utilisent un système à particules. Chaque particule représente une goutte de pluie ; lorsque la goutte touche la surface, elle glisse sur celle-ci en transportant une certaine quantité de sédiments dissout. Les particules sont générées à l'aide d'une fonction de distribution modélisant la pluie incidente, et s'écoulent sur la surface de l'objet sous l'effet des forces de gravité, de friction et d'autorépulsion. Ces forces sont additionnées et projetées sur le polygone où se trouve la goutte, pour ensuite être appliquées. Dorsey *et al.* ne donnent pas plus de précisions sur la façon dont leur algorithme fonctionne. En plus des forces que nous avons mentionnées, Dorsey *et al.* tiennent compte de la rugosité de la surface et proposent deux façons de la modéliser : la première consiste à soumettre les particules à une force aléatoire dans le plan de la surface ; la seconde consiste à appliquer un *bump map* [Bli78] sur l'objet pour modifier la normale de sa surface. L'effet du *bump map* semble assez intéressant : les gouttes d'eau évitent les bosses et s'accumulent dans les creux, comme si la surface était vraiment irrégulière.

Si l'angle entre le vecteur vitesse d'une goutte et la tangente à la surface dépasse une valeur donnée (seuil d'adhérence), la goutte quittera la surface ; l'adhérence n'est donc pas directement fonction de la force appliquée sur la goutte. Lorsque les gouttes quittent la surface, elles peuvent, en retombant, créer un flot secondaire sur une autre partie de la surface de l'objet.

L'interaction de l'eau avec la surface comporte deux facettes. Il faut d'abord simuler

l'absorption de l'eau par le matériau formant la surface. Le taux d'absorption est déterminé par l'absorptivité du matériau, qui est elle-même fonction de la saturation du matériau. L'absorption ainsi que l'évaporation ont pour effet de diminuer la masse de la goutte d'eau. Si la masse d'une goutte est inférieure à un certain seuil, la goutte est retirée de la simulation.

La deuxième facette de l'interaction eau – surface est la modélisation de la dissolution, du transport et du dépôt des matériaux qui formeront la patine. La dissolution et le dépôt de sédiments sont modélisés par un ensemble de deux équations différentielles, fonctions de la concentration de sédiments sur la surface et dans la goutte d'eau

$$\frac{\partial S}{\partial t} = -k_S S + k_D D \frac{m}{A}$$

$$\frac{\partial D}{\partial t} = k_S S \frac{A}{m} + k_D D + I_D$$

où S est la concentration de sédiments dissouts dans une goutte d'eau, D est la concentration de sédiments sur la surface, m est la masse de la goutte, A est l'aire de la surface de contact avec la surface, I_D les dépôts dus à des facteurs environnementaux (pollution, etc.) et k_S et k_D les constantes de taux de solution et d'adhésion, respectivement. On voit donc que le dépôt de sédiments est causé seulement par la diminution de masse de la goutte d'eau, diminution qui entraîne une augmentation de la concentration de sédiments dans la goutte.

2.5 Rendu

Le rendu est l'étape qui s'efforce de donner un aspect réaliste aux objets. Le rendu des surfaces opaques est un domaine assez bien développé et pose en général peu de problèmes. Les objets transparents, cependant, sont plus difficiles et plus coûteux à traiter. Nous verrons dans cette section quelles solutions ont été proposées pour faire le rendu de tels objets. Nous verrons en particulier les techniques proposées pour faire le rendu de vagues. Il semble malheureusement qu'aucune technique de rendu de gouttes d'eau n'ait été proposée.

2.5.1 Rendu d'objets transparents

Les objets transparents sont la cause de plusieurs effets optiques (réfraction, caustiques, dispersion de la lumière, réflexion spéculaire) qu'il est essentiel de reproduire si l'on veut en faire un rendu réaliste. Les réflexions spéculaires et les réfractions peuvent être traitées facilement en utilisant un algorithme de tracé de rayon standard. Les caustiques et la dispersion de la lumière requièrent cependant de nouvelles techniques.

Pour les caustiques, Shinya *et al.* [SST89] proposent d'utiliser une combinaison de tracé de rayon inverse [Arv86] et de tracé de faisceau (*pencil tracing*) [STN87] [SST89]. Un faisceau est fait d'un rayon dans lequel est encodé l'étalement du faisceau. Pour calculer la radiance à un point d'une surface, on trace le faisceau en partant de la lumière. Lorsque le faisceau frappe des surfaces réfléchissantes ou transparentes, on change sa direction, la courbure de la surface modifiant l'étalement du faisceau. Lorsque le faisceau atteint une surface, son intersection est approximée par un polygone de lumière et un faisceau réfracté.

Shinya *et al.* proposent aussi un modèle de dispersion où chaque polygone est transformé en un solide de dispersion, représentant le polygone d'intersection sous différentes longueurs d'onde (figure 2.3). Le triplet de couleur $(R(x,y), G(x,y), B(x,y))$ en un point du polygone est donné par les trois intégrales

$$R(x,y) = \int_{\lambda_{min}(x,y)}^{\lambda_{max}(x,y)} t(\lambda) r(\lambda) L(\lambda) d\lambda$$

$$G(x,y) = \int_{\lambda_{min}(x,y)}^{\lambda_{max}(x,y)} t(\lambda) g(\lambda) L(\lambda) d\lambda$$

$$B(x,y) = \int_{\lambda_{min}(x,y)}^{\lambda_{max}(x,y)} t(\lambda) b(\lambda) L(\lambda) d\lambda$$

où $L(\lambda)$ est le spectre de la lumière, $(r,g,b)(\lambda)$ est une fonction donnant la valeur des composantes R, G et B pour reproduire la couleur λ et $t(\lambda)$ est la transparence spectrale.

Takagi *et al.* [TYT+84] ont proposé, pour traiter la dispersion, de faire du tracé de rayon en remplaçant chaque rayon de lumière polychromatique par plusieurs rayons monochromatiques. Ainsi, chaque rayon peut être réfracté en fonction de sa longueur

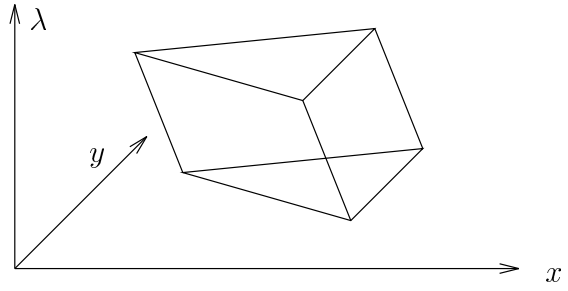


FIG. 2.3: Solide de dispersion

d'onde, au prix toutefois d'une multiplication du temps de calcul. Dans d'autres approches [Tho86] [YKIS88], le rayon réfracté devient un faisceau, qui est à son tour tracé dans la scène. Ces approches sont toutefois coûteuses, car le nombre de rayons ou de faisceaux à tracer croît très rapidement.

2.5.2 Rendu des vagues

Pour faire le rendu des vagues, Peachey [Pea86] propose deux méthodes. D'abord, le tracé de rayon, qui permet de calculer les réflexions du ciel et de l'environnement à la surface de l'eau. Étant donnée le coût presque prohibitif, à l'époque, de calculer des animations avec cette méthode, il propose comme alternative une version modifiée du *A-buffer* (qui est elle-même, rappelons-le, une version modifiée du *Z-buffer*). La version du *A-buffer* proposée est moins gourmande en mémoire que la version originale ; elle effectue la conversion des vagues ligne par ligne en considérant la surface de l'océan comme une surface paramétrique. Pour traiter la réflexion des objets environnants, Peachey propose d'utiliser la technique de l'*environment map* [BN76]. Fournier et Reeves [FR86] utilisent une méthode semblable, c'est-à-dire le balayage ligne par ligne et l'*environment map* ; ils ajoutent un *bump map* pour modéliser les ondelettes à la surface de l'eau.

2.6 Le système d'animation *Taarna*

Étant donné que nous avons réalisé notre programme à l'intérieur du système d'animation *Taarna*, nous en glisserons ici quelques mots. Le système d'animation *Taarna*

est un système d'animation extensible. Il est composé d'un module de gestion de l'animation, appelé `cameraman`, qui instancie et gère des *animateurs*. Un animateur n'est pas un programme exécutable par lui-même ; c'est un fichier contenant du code compilé. Lorsque `cameraman` charge un animateur, il en crée une instance. Cette instance peut être configurée, en attribuant des valeurs à certains paramètres, au moyen d'une interface graphique. Chaque instance d'animateur est chargée d'animer un objet. Cet objet est décrit par une forme, généralement un maillage de polygones ou de surfaces paramétriques, et une surface, définie à l'aide d'un programme *RenderMan* [Ups89]. *RenderMan* utilise une variante du *A-buffer* pour faire le rendu ; il est donc difficile de tenir compte de la réfraction de la lumière.

L'algorithme de production d'une animation est le suivant :

```
Pour chaque image
    Pour chaque animateur
        Exécuter l'animateur pour un pas de temps
    Pour chaque objet
        Faire son rendu
```

Le programme `cameraman` possède une interface graphique permettant de contrôler divers aspects du rendu de l'animation, telle la distance focale et la position de la caméra. On peut aussi visualiser l'animation au fur et à mesure qu'elle est calculée, et ce en trois modes (fil de fer, rendu par *OpenGL*, rendu par *RenderMan*). Ceci permet de juger rapidement de la qualité de l'animation et d'y apporter les corrections nécessaires.

Chapitre 3

Animation, modélisation et rendu des gouttes

On s'obstine toujours à attacher les gens avec des cordes, alors que c'est tellement plus facile de les clouer.

Boris Vian, *L'Équarissage pour tous*

Dans les films, on établit souvent une distinction entre l'action principale et les actions secondaires d'une scène. Par exemple, on pourrait avoir une scène dans laquelle l'action principale serait un bébé lançant ses jouets un peu partout ; les actions secondaires seraient alors les jouets rebondissant sur les murs, frappant et entraînant avec eux d'autres objets. L'action principale est l'action sur laquelle l'attention du spectateur est concentrée, alors que les actions secondaires ne servent que de contexte à l'action principale.

Dans les films d'animation traditionnels, chaque objet doit être dessiné par un artiste, qu'il fasse partie de l'action principale ou d'une action secondaire. Souvent, on pourrait automatiser le comportement des objets faisant partie des actions secondaires, car ils suivent la plupart du temps des lois bien établies (comme celles de la physique). L'animation par ordinateur offre cette possibilité, et nous avons voulu l'exploiter. Dans le cadre de notre projet, nous avons décidé d'écrire un logiciel qui permette d'automatiser

l'évolution de gouttes de liquide sur la surface d'objets.

Les principaux critères qui ont guidé la conception de notre logiciel sont :

La rapidité : nous voulions que le logiciel puisse calculer des séquences d'animation dans lesquelles des centaines, voire des milliers, de gouttes glissent sur une surface composée de quelques milliers de polygones.

Le réalisme : nous voulions que les gouttes des animations calculées par notre logiciel aient l'air de se comporter comme de vraies gouttes. Il est important de souligner que nous ne cherchions pas à réaliser une simulation physique exacte de l'écoulement de gouttes sur des surfaces. Nous voulions plutôt une simulation visuellement réaliste.

L'intégration : nous avons essayé de concevoir le logiciel de façon à ce qu'il puisse être facilement intégré comme animateur dans l'architecture du système d'animation *Taarna*.

Dans le reste de ce chapitre, nous décrirons l'approche que nous avons adoptée pour résoudre le problème, ainsi que quelques éléments particuliers à la réalisation de notre logiciel. Nous terminerons en glissant quelques mots sur la façon dont nous avons fait le rendu de notre modèle.

3.1 Mouvement des gouttes

Nous avons décidé d'utiliser un système à particules comme modèle guidant la conception de notre logiciel. Deux facteurs ont motivé ce choix :

- d'abord, la nature même des gouttes d'eau. En faisant abstraction de certains comportements, on peut concevoir une goutte de liquide sur un objet comme une particule se déplaçant sur une surface dans un espace à trois dimensions ;
- ensuite, la généralité et la flexibilité du système à particules nous permettent de commencer par construire un système relativement simple et d'y ajouter des fonctionnalités plus complexes par la suite.

Dès le début, nous avons décidé d'utiliser les lois de la physique mécanique classique pour guider les particules du système. Ainsi, le déplacement des particules est déterminé

par des forces, qui créent des accélérations, qui modifient les vitesses. Pour effectuer les calculs appropriés et décrire l'état d'une goutte, nous aurons donc besoin de quatre variables :

- $\mathbf{p}$, le vecteur position, exprimé en mètres ;
- $\mathbf{v}$, le vecteur vitesse, exprimé en mètres par seconde ;
- $\mathbf{a}$, le vecteur accélération, exprimé en mètres par seconde carrée ;
- m, la masse, exprimée en kilogrammes.

Lorsqu'on fait une simulation de l'interaction d'un grand nombre d'objets rigides qui se déplacent de façon arbitraire, la détection des collisions peut devenir très coûteuse. Dans les premiers prototypes de notre logiciel, les gouttes étaient des sphères se déplaçant dans un environnement où « vivaient » des objets composés de polygones ; le déplacement des gouttes était gouverné par la loi $\mathbf{F} = m\mathbf{a}$. Lorsqu'une collision se produisait entre une goutte et un objet, la vitesse $\mathbf{v}$ de la sphère devenait $-k_e\mathbf{v}$, où $0 \leq k_e \leq 1$ était le coefficient d'élasticité de la collision. Pour détecter une collision entre une goutte et son environnement, nous considérions le segment de droite (dans $\mathbb{R}^3$) délimité par les positions initiale et finale de la goutte, segment que nous intersections avec chacun des éléments de l'environnement.

Il y a des désavantages évidents à utiliser cette méthode. D'abord, le nombre d'intersections à vérifier est $O(n(n+m))$, n étant le nombre de gouttes et m le nombre de polygones dans l'environnement. De plus, les gouttes qui roulent sur une surface sont en collision permanente avec celle-ci. Pour éviter qu'une goutte ne passe à travers la surface à cause des erreurs dues à la précision limitée de l'ordinateur, on doit la placer légèrement au-dessus de la surface. Il en résulte, et nous avons pu l'observer dans nos premières animations, un sautillement agaçant des gouttes.

Étant donné le nombre de collisions à détecter dans nos premiers prototypes, la majeure partie du temps de calcul était employée à la détection de collisions. Pour atteindre notre objectif de rapidité, nous devions réduire drastiquement le temps consacré à cette activité. Pour ce faire, nous pouvions :

- réduire le nombre de détections de collisions ;

– simplifier le calcul des détections de collisions.

Le calcul des détections de collisions étant déjà assez simple (il ne s'agit que de calculer l'intersection entre une droite et un plan, puis de vérifier si l'intersection se trouve à l'intérieur du polygone), nous avons décidé de réduire le nombre de détections de collisions.

Pour accomplir cela, nous aurions pu utiliser un algorithme de subdivision hiérarchique de l'espace comme ceux utilisés dans le tracé de rayon [DS84] [Gla84], ou utiliser une hiérarchie de volumes englobants [RW80] [Weg84] [KK86]. Cependant, cela n'aurait pas résolu le problème de sautillement des gouttes. Nous avons cru plus profitable d'exploiter le comportement de la goutte. Lorsqu'une goutte se déplace sur une surface, elle reste (en général) sur cette surface. Donç plutôt que de déplacer nos gouttes dans l'espace, nous les ferons se déplacer (ou glisser) sur les surfaces. Ainsi, tant que la goutte ne sort pas des limites de la surface, nous n'avons pas besoin de faire de calcul de détection de collisions avec les objets de la scène. Cependant, il est difficile de faire rouler des particules sur des surfaces quelconques. Hégron [Heg91] a proposé une solution dans le cas d'une sphère qui roule sur une surface biparamétrique. Elle consiste à déplacer la sphère en suivant la tangente de la surface, puis à projeter sa nouvelle position sur la surface. Nous avons adopté une solution similaire. Nous ferons « rouler » nos particules sur une surface ; à la différence de Hégron, cependant, nos surfaces seront constituées de polygones. On pourrait penser que, pour le rendu, l'utilisation de polygones constitue un désavantage. Ce n'est en général pas le cas, car un maillage suffisamment fin de polygones approxime bien n'importe quelle surface. Dans les rares cas où l'utilisation d'un maillage ne donne pas de bons résultats, rien ne nous empêche d'utiliser une représentation différente de l'objet pour en faire le rendu. Il faut cependant s'assurer que les gouttes reposent bien sur cette surface, et non sur le maillage utilisé dans la simulation ; pour ce faire, nous pouvons par exemple projeter la position des gouttes sur cette surface.

L'utilisation d'un maillage de polygone est très importante pour le problème qui nous préoccupe, car elle permet de simplifier nos algorithmes. Nous verrons dans la section qui suit que, grâce à l'utilisation des polygones, il est possible de dériver l'équation donnant la position d'une particule sur une surface à n'importe quel instant de la simulation.

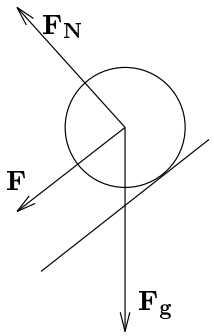


FIG. 3.1: Décomposition de la force de gravité s'exerçant sur une particule glissant sur un plan.

3.1.1 Équation analytique de la trajectoire des gouttes

Nous avons établi jusqu'à maintenant que les gouttes sont représentées par des particules et qu'elles glissent sur une surface composée de sections de plan (polygones planaires). Dans le but de simplifier les calculs et de garder la paramétrisation de notre modèle simple, nous considérerons que les particules ne sont soumises qu'à deux forces : la somme des forces exercées par la gravité ($\mathbf{F_g}$) et la surface ($\mathbf{F_N}$) (figure 3.1), que nous noterons $\mathbf{F}$, et la force de friction, $\mathbf{f}$. La première est constante tant que la particule se déplace sur un même plan (nous supposons que la surface ne bouge pas durant la traversée d'un polygone), alors que la seconde est variable.

Pour plus de commodité, nous séparerons la force de friction en deux composantes : la friction $\mathbf{f_r}$ causée par la rugosité de la surface et la friction $\mathbf{f_v}$ causée par la viscosité du liquide.

La rugosité

La rugosité d'une surface a pour effet de freiner les gouttes qui y glissent. Il est clair que la masse de la goutte est un facteur important pour déterminer l'effet de la rugosité sur le mouvement des goutte. En effet, plus la goutte est grosse, plus les aspérités sur la surface doivent être importantes pour freiner la goutte (figure 3.2). Nous avons conçu un modèle pour simuler l'effet de la rugosité de la surface sur les gouttes. Tout d'abord,

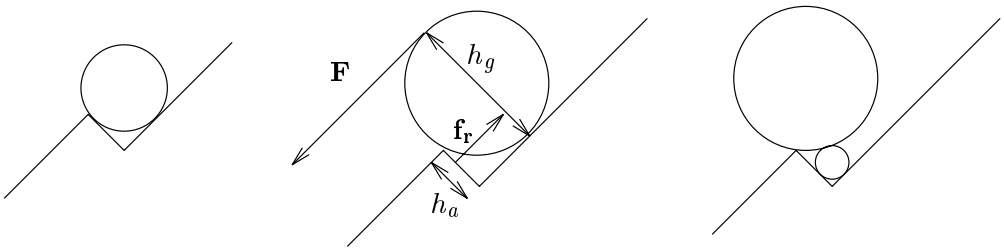


FIG. 3.2: L'effet de la rugosité de la surface sur les gouttes : (a) une petite goutte est complètement arrêtée par une aspérité ; (b) une grosse goutte est freinée par une aspérité ; (c) une traînée (représentée par une goutte sur le rebord de l'aspérité) facilite le passage d'une goutte.

nous définissons la force de freinage due à la rugosité :

$$\mathbf{f_r} = k_r \mathbf{F}$$

où $0 \leq k_r \leq 1$ est le coefficient de rugosité de la surface. L'effet de la rugosité est de diminuer la force résultante subie par la goutte :

$$\mathbf{F} \leftarrow \mathbf{F} - \mathbf{f_r}.$$

Maintenant, nous pouvons approximer l'effet d'une aspérité sur une goutte à l'aide du rapport entre la hauteur de l'aspérité et la hauteur de la goutte (figure 3.2 (b)). Nous définissons le coefficient de rugosité de la surface comme suit :

$$k_r(h_g) = \begin{cases} C\frac{h_a}{h_g} & \text{si } C\frac{h_a}{h_g} \leq 1 \\ 1 & \text{sinon.} \end{cases} \tag{3.1}$$

La constante $1/C$ représente la hauteur d'aspérité qui stoppe une goutte. Cette constante est exprimée comme une fraction de la hauteur de la goutte. Par exemple, si $C = 2$, alors toute goutte plus petite que deux fois la hauteur des aspérités sera arrêtée.

La hauteur d'une goutte est fonction de son volume. Puisque nous approximons les gouttes par des sphères, la hauteur d'une goutte est donnée en mètres par :

$$h_g = 2r = 2\sqrt[3]{\frac{3V}{4\pi}} \tag{3.2}$$

où V représente le volume de la goutte, en mètres cube. Nous savons que la masse volumique de l'eau est de 1000 kg/m^3 (c'est une bonne approximation pour la plupart des liquides). Nous pouvons donc exprimer la hauteur d'une goutte en fonction de sa masse comme suit :

$$h_g = 2r = 2\sqrt[3]{\frac{3000m}{4\pi}}. \tag{3.3}$$

En insérant cette équation dans l'équation 3.1, nous obtenons

$$k_r(m) = \begin{cases} C\frac{h_a}{2\sqrt[3]{\frac{3000m}{4\pi}}} & \text{si } C\frac{h_a}{2\sqrt[3]{\frac{3000m}{4\pi}}} \leq 1 \\ 1 & \text{sinon} \end{cases} \tag{3.4}$$

Nous pouvons vectoriser cette équation afin de simuler le comportement d'une surface anisotropique. Définissons tout d'abord un système de coordonnées locales pour chaque polygone de la surface. Soient $\mathbf{A}$ et $\mathbf{B}$ deux sommets quelconques du polygone, et soit $\mathbf{N}$ la normale du polygone (rappelons que les polygones doivent être planaires). Le système de coordonnées locales du polygone est défini par

$$\mathcal{L}_p = \{l_1, l_2, l_3\} = \left\{\mathbf{N}, \frac{\mathbf{B}-\mathbf{A}}{\|\mathbf{B}-\mathbf{A}\|}, \mathbf{N} \times \frac{\mathbf{B}-\mathbf{A}}{\|\mathbf{B}-\mathbf{A}\|}\right\} \tag{3.5}$$

Ainsi, nous obtenons, pour $i = 2, 3$[1] :

$$\mathbf{k_r}_i(m) = \begin{cases} \mathbf{C}_i\frac{h_a}{2\sqrt[3]{\frac{3000m}{4\pi}}} & \text{si } \mathbf{C}_i\frac{h_a}{2\sqrt[3]{\frac{3000m}{4\pi}}} \leq 1 \\ 1 & \text{sinon.} \end{cases} \tag{3.6}$$

La viscosité

La force de frottement causée par la viscosité du liquide est proportionnelle à la vitesse de la goutte. Nous la définissons donc comme :

$$\mathbf{f_v}(t) = k_v\mathbf{v}(t)$$

où $\mathbf{v}(t)$ est la vitesse de la particule et $k_v < 0$ le coefficient de viscosité. On remarque que contrairement à la force de frottement rugueuse $\mathbf{f_r}$, la force de frottement visqueuse $\mathbf{f_v}$ varie en fonction du temps.

[1] Pour $i = 1$ (la composante du coefficient de rugosité normale à la surface), on définit $k_{r1} \equiv 0$

L'équation du mouvement

Maintenant que nous avons détaillé les forces s'exerçant sur une goutte de masse m, nous pouvons dériver l'équation donnant la position d'une goutte à un instant t. Nous poserons comme hypothèse que la masse d'une goutte est constante durant son passage sur un polygone.

En partant de la seconde loi de Newton, nous avons

$$\mathbf{v}(t) = \mathbf{v}(0) + \int_0^t \frac{\mathbf{F}(u)}{m} du$$

ce qui se traduit dans notre cas

$$\mathbf{v}(t) = \mathbf{v}(0) + \int_0^t \mathbf{a} + \frac{k_v}{m}\mathbf{v}(u) du$$

où

$$\mathbf{a} = \frac{\mathbf{F} - \mathbf{f_r}}{m}.$$

En dérivant de chaque côté par rapport à t, on obtient, pour une composante quelconque du vecteur vitesse $\mathbf{v}$

$$\frac{dv}{dt} = a + \frac{k_v}{m} v. \tag{3.7}$$

Si on fait abstraction de l'accélération a, on a

$$\frac{dv}{v} = \frac{k_v}{m} dt$$

que l'on intègre, ce qui donne

$$v(t) = c_1 e^{\frac{k_v t}{m}}. \tag{3.8}$$

En substituant l'équation 3.8 dans l'équation 3.7, on obtient :

$$c_1' e^{\frac{k_v t}{m}} + \frac{k_v}{m} c_1 e^{\frac{k_v t}{m}} = a + \frac{k_v}{m} c_1 e^{\frac{k_v t}{m}}$$

$$c_1' = a e^{-\frac{k_v t}{m}}$$

$$c_1 = -\frac{ma}{k_v} e^{-\frac{k_v t}{m}} + c_2. \tag{3.9}$$

En substituant l'équation 3.9 dans l'équation 3.8, on obtient :

$$v(t) = -\frac{ma}{k_v} + c_2 e^{\frac{k_v t}{m}}.$$

La constante c_2 représente la condition initiale :

$$v(0) = -\frac{ma}{k_v} + c_2$$

$$c_2 = v(0) + \frac{ma}{k_v}$$

donc

$$v(t) = -\frac{ma}{k_v} + \left(v(0) + \frac{ma}{k_v}\right) e^{\frac{k_v t}{m}}$$

ou, sous forme vectorielle,

$$\mathbf{v}(t) = -\frac{m\mathbf{a}}{k_v} + \left(\mathbf{v}(0) + \frac{m\mathbf{a}}{k_v}\right) e^{\frac{k_v t}{m}}.$$

Nous avons trouvé une équation pour la vitesse de la goutte. Trouvons maintenant l'équation de sa position. On sait que

$$\mathbf{p}(t) = \mathbf{p}(0) + \int_0^t \mathbf{v}(u)du$$

ce qui se traduit dans notre cas par

$$\mathbf{p}(t) = \mathbf{p}(0) + \int_0^t -\frac{m\mathbf{a}}{k_v} + \left(\mathbf{v}(0) + \frac{m\mathbf{a}}{k_v}\right) e^{\frac{k_v u}{m}} du$$

$$\mathbf{p}(t) = \mathbf{p}(0) - \frac{m\mathbf{a}t}{k_v} + m\left(\mathbf{v}(0) + \frac{m\mathbf{a}}{k_v}\right)\left(\frac{e^{k_v t} - 1}{k_v}\right). \tag{3.10}$$

Nous venons de dériver une équation nous donnant la position d'une particule à n'importe quel moment t de la simulation. Cependant, cette équation n'est valable que si la pente de la surface où se trouve la particule ne change pas. Donc, la surface sur laquelle nos gouttes glissent étant polygonisée, notre équation n'est valable qu'à l'intérieur d'un polygone.

Il est important de noter que notre équation ne s'applique que si notre modèle est un maillage de polygones. Il est fort probable, mais nous ne l'avons pas vérifié, qu'une équation analytique donnant la position des gouttes pour d'autres types de surfaces (*e.g.* splines) soit beaucoup plus difficile à dériver, peut-être même impossible.

3.1.2 Fusion des gouttes

Sur une fenêtre, par jour de pluie, il est courant d'observer la fusion de deux gouttes d'eau qui se rencontrent. Notre modèle approxime ce phénomène en détectant les collisions entre les gouttes. De la même façon que nous avons voulu réduire le coût de la détection des collisions goutte-surface, nous verrons dans cette section comment nous avons réduit le coût de la détection des collisions entre les gouttes.

Tout d'abord, nous avons simplifié le calcul à effectuer pour chaque détection de collision. Il eût été tentant de considérer pour ce calcul la trajectoire d'une goutte et de l'intersecter avec la trajectoire de chacune des autres gouttes et ce, en fonction du temps. C'eût d'ailleurs été la solution correcte. Cependant, étant donnée la complexité de la trajectoire de la goutte (équation 3.10), il eût fallu utiliser une méthode numérique (par exemple, l'algorithme de Newton-Raphson) plus coûteuse qu'une solution analytique. Pour réduire le nombre de calculs, nous aurions pu envelopper les trajectoires dans une boîte englobante et vérifier d'abord les intersections entre les boîtes. Cependant, étant donnée l'équation d'une trajectoire, il n'est pas facile de trouver sa boîte englobante.

Nous avons donc décidé de détecter les collisions seulement à l'instant pour lequel nous calculons la position des gouttes. Cette approche se justifie visuellement par le fait qu'un observateur peut interpoler visuellement ce qui se passe entre deux images d'une animation, parce que la position des gouttes doit présenter une certaine continuité à ses yeux. Cependant l'observateur ne peut pas déterminer, surtout lorsque l'animation comporte beaucoup de gouttes, si deux gouttes auraient dû se fusionner durant le dernier trentième de seconde (pour une animation à trente images par seconde).

Ainsi, à l'instant t_i, pour chaque goutte j, nous avons une position $\mathbf{p}_j(t_i)$. Nous approximons l'espace occupé par une goutte à l'aide d'une sphère centrée à la position de la goutte et dont le rayon r est donné par l'équation 3.3, ci-haut. Pour détecter si une collision se produit entre les gouttes a et b, il suffit de calculer la distance entre les centres de celles-ci ; si elle est inférieure à $r_a + r_b$, une collision s'est produite.

Il est aussi possible de réduire le nombre de calculs de détection de collisions en regroupant les gouttes en voisinages. On peut choisir plusieurs critères pour déterminer le voisinage d'une goutte. Dans le cadre de notre simulation, nous avons la chance de pou-

voir utiliser un critère « naturel » : le triangle sur lequel se trouve la goutte. Ainsi, plutôt que de vérifier si une collision se produit entre la goutte a et chacune des autres gouttes du système, nous ne considérons comme candidates que les gouttes qui se trouvent sur le même triangle que la goutte a. Puisque, en général, les triangles d'un modèle sont petits, le nombre de gouttes se trouvant sur un même triangle est beaucoup plus petit que le nombre total de gouttes du système, d'où une réduction importante du nombre de calculs de détection de collision. Il est évident, cependant, que si les gouttes coulent sur un immense plan modélisé par deux grands triangles, alors notre méthode est beaucoup moins avantageuse (voir à cet effet le tableau 4.2, à la section 4.1.1). Signalons aussi que notre méthode ne fonctionne pas toujours ; en effet, si deux gouttes se touchent mais qu'elles sont situées sur deux triangles différents, par exemple sur deux triangles adjacents, notre algorithme ne détectera pas de collision.

La fusion des gouttes ajoute au réalisme ; cependant, elle invalide l'hypothèse utilisée pour dériver l'équation du mouvement des gouttes (section 3.1.1), *i.e.* que la masse d'une goutte doit rester constante. Mais, si nous supposons que les variations de masse se produisent instantanément, nous pouvons simplement, lorsqu'il y a fusion de gouttes, recalculer les paramètres de l'équation du mouvement, afin qu'elle demeure valide pour les calculs ultérieurs.

3.1.3 Traînées

Lorsqu'une goutte coule sur une surface, elle peut laisser une traînée derrière elle. L'effet de cette traînée est double : non seulement modifie-t-elle l'aspect visuel de la surface, mais elle diminue aussi le coefficient de rugosité de la surface. Ainsi, si une goutte croise une traînée, elle aura tendance à modifier sa trajectoire pour suivre la traînée.

Dans les deux sections qui suivent, nous décrirons l'approche que nous avons adoptée pour traiter les effets des traînées.

Modification de l'aspect visuel

Le fait de voir par où passe une goutte en modifiant l'aspect visuel de la surface ajoute beaucoup au réalisme de la simulation. Grâce aux traînées, le spectateur a moins l'impression que la goutte « saute » d'un endroit à l'autre, surtout lorsqu'il y a beaucoup de gouttes ou que leur mouvement est rapide ou complexe. Les traînées aident à renforcer l'impression de continuité.

Lorsqu'on veut modifier les caractéristiques visuelles d'une surface, on pense tout de suite à utiliser les textures, car c'est une méthode à la fois flexible et efficace. Elle est flexible car, suivant les propriétés du liquide coulant sur la surface, la texture peut modifier le coefficient de réflexion spéculaire (par exemple, si nous avons des gouttes d'eau), la couleur (si nous avons des gouttes de peinture), les normales de la surface ou n'importe quel autre paramètre de notre choix. C'est une méthode efficace car elle ne requiert pas l'insertion de nouvelles surfaces dans la scène. Nous épargnons donc sur le calcul de visibilité. Cependant, il est très difficile de construire une paramétrisation isométrique[2] ; nous devons donc utiliser une résolution qui soit satisfaisante pour chaque section du modèle. Cela peut se traduire par des textures qui prennent un espace mémoire assez important. De plus, chaque polygone de la traînée doit être transformé en texels. Puisque cette transformation n'est pas isométrique, il s'agit là d'une opération coûteuse et approximative. Tout ceci, conjugué à des contraintes de temps, nous a fait opter pour une modélisation de traînées à l'aide de polygones appliqués sur la surface.

Pour placer la traînée, nous devons tout d'abord déterminer la trajectoire de la goutte sur la surface. Celle-ci peut être assez complexe (voir équation 3.10). Pour éviter de passer trop de temps à ce calcul, nous utilisons des segments de droite pour approximer la trajectoire. Ces segments sont formés par la position de la goutte au début et à la fin d'un intervalle[3] (figure 3.3). La largeur de la traînée est déterminée par le diamètre de

[2] Une paramétrisation isométrique a l'avantage de toujours faire correspondre à toute aire de grandeur A de la surface une aire de grandeur $f(A)$ sur la texture. Ainsi, le niveau de détail est le même pour chaque région de la surface. Le système *Taarna* fournit une paramétrisation de textures, mais elle n'est pas isométrique.

[3] Un intervalle est la plus petite des deux valeurs suivantes : la durée entre deux *frames* d'animation consécutifs, ou la durée pendant laquelle la goutte reste sur le triangle où elle se trouve.

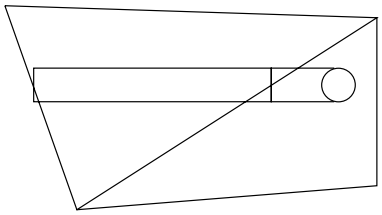
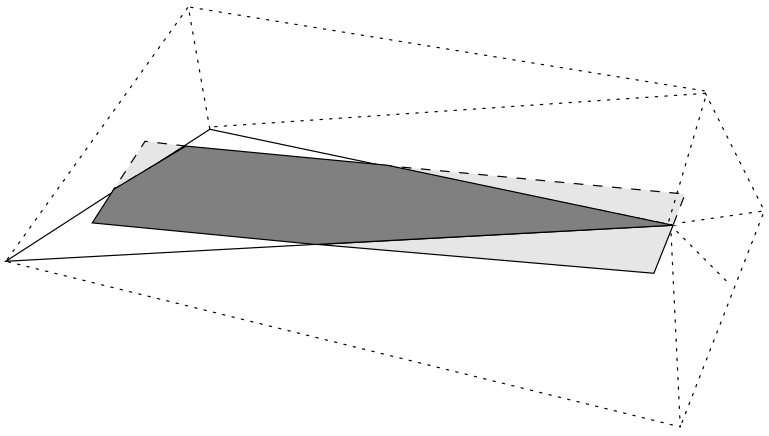


FIG. 3.3: Traînée laissée par une goutte

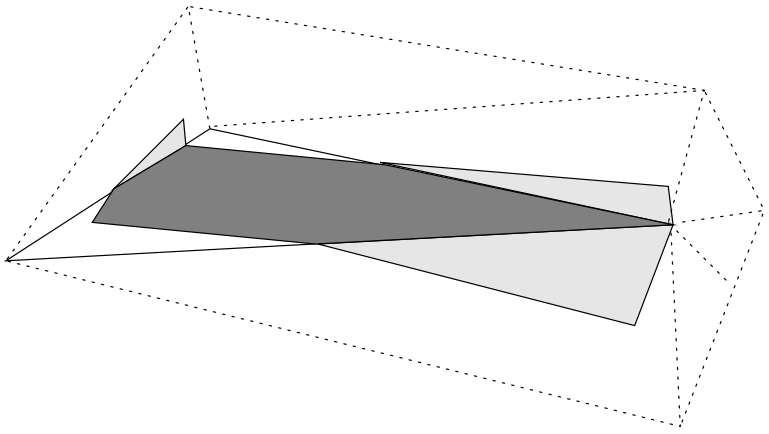
la goutte.

La traînée est donc composée d'une série de rectangles orientés de façon parallèle au triangle où coule la goutte (triangle de support). Cependant, il arrive souvent que ces rectangles débordent des triangles de support, surtout si la dimension de la goutte est semblable à celle du triangle. C'est pourquoi nous devons les découper. Une fois découpées, les parties du rectangle à l'extérieur du triangle de support sont projetées sur les triangles voisins, afin qu'elles soient toujours parallèles à la surface sous-jacente (figure 3.4). Cette opération est facilitée par l'utilisation du graphe des voisins du triangle, qui permet de trouver rapidement le triangle sur lequel effectuer la projection. (Nous parlerons du graphe de voisinage plus en détail à la section 3.3.2.) Il y a deux inconvénients à la projection. Le premier est que la traînée peut, dans certains cas, être considérablement déformée par la projection. Cependant, pour la plupart des modèles, qui sont relativement lisses, ces déformations sont peu apparentes. Si ces déformations deviennent trop gênantes, nous pourrions faire pivoter les parties de rectangles, plutôt que de les projeter. Le deuxième inconvénient est le grand nombre de polygones nécessaires pour modéliser une traînée. Dans une animation contenant plusieurs milliers de gouttes, le rendu des traînées peut alors consommer un temps de calcul non négligeable. Plusieurs améliorations bien connues sont envisageables, mais les résultats obtenus nous satisfaisaient déjà.

Nous avons une méthode qui permet d'enregistrer de façon satisfaisante les modifications visuelles causées par les traînées. Voyons maintenant comment nous procédons pour modifier les caractéristiques physiques de la surface.



(a)



(b)

FIG. 3.4: Construction d'une traînée à l'aide de polygones. (a) La traînée est découpée suivant le triangle de support. (b) Les parties à l'extérieur du triangle de support sont projetées sur les triangles voisins.

Modification du coefficient de rugosité

La traînée introduit une variation de la rugosité de la surface ; en effet, on peut imaginer que la traînée remplit les aspérités de la surface, rendant celle-ci plus lisse. Ce phénomène entraîne une variation de la force de friction rugueuse entre les gouttes et la surface. Or, pour trouver l'équation de la trajectoire des gouttes (équation 3.10), nous avons posé comme hypothèse une force de friction rugueuse qui ne varie pas dans le temps ; nous ne pouvons donc pas, sous peine de ne plus pouvoir utiliser une solution analytique, attribuer une valeur de rugosité différente à chaque point de la surface (en utilisant une texture, par exemple).

En fait, à cause de cette hypothèse, nous devons nous contenter d'une approximation du coefficient de rugosité de la surface. La solution que nous avons retenue est la suivante : pour chaque triangle, nous gardons deux coefficients, R_2 et R_3 ($R_i \geq 0$), pour paramétriser la rugosité de la surface. Nous définissons ces coefficients en fonction du système de coordonnées locales du triangle, $\mathcal{L}_p$, donné par l'équation 3.5 de la section 3.1.1 [4]. Initialement, nous posons $R_2 = R_3 = h_a/h_g$ pour tous les triangles. Lorsqu'une goutte passe sur un triangle, nous modifions ses coefficients de rugosité en tenant compte de la direction du parcours de la goutte (figure 3.5). Ainsi, si une goutte de rayon r part de (a_2, a_3) et qu'elle se rend à (b_2, b_3), nous modifierons les coefficients de rugosité R_2 et R_3 du triangle comme suit :

$$\Theta = \arctan\left(\frac{b_3 - a_3}{b_2 - a_2}\right)$$

$$A_2 = T_2\left[|2r\sin(\Theta)| + \left|\cos(\Theta)\sqrt{(b_2 - a_2)^2 + (b_3 - a_3)^2}\right|\right]$$

$$A_3 = T_3\left[|2r\cos(\Theta)| + \left|\sin(\Theta)\sqrt{(b_2 - a_2)^2 + (b_3 - a_3)^2}\right|\right]$$

$$R_2 \leftarrow \begin{cases} R_2\left(1 - \frac{A_2}{A}\right) & \text{si } \frac{A_2}{A} \leq 1 \\ 0 & \text{sinon.} \end{cases}$$

[4] Rappelons que l_1 est perpendiculaire au triangle ; il n'y a donc pas de rugosité dans cette direction.

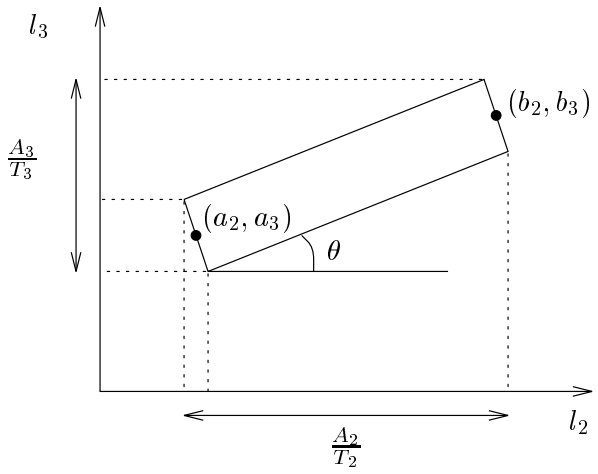


FIG. 3.5: Modification des coefficients de rugosité.

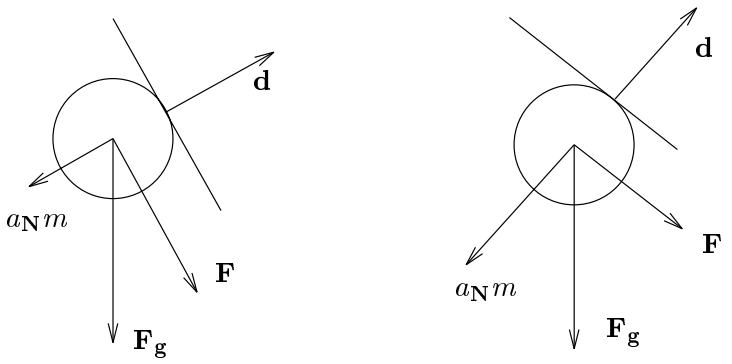
$$R_3 \leftarrow \begin{cases} R_3 \left(1 - \frac{A_3}{A}\right) & \text{si } \frac{A_3}{A} \leq 1 \\ 0 & \text{sinon.} \end{cases}$$

où A est l'aire du triangle, T_2 et T_3 étant des paramètres ajustables indiquant le « taux » de remplissage des aspérités lors du passage d'une goutte dans les directions l_2 et l_3, respectivement. Ainsi, si plusieurs gouttes passent dans la même direction sur un triangle donné, le coefficient de rugosité de celui-ci pour cette direction diminuera, facilitant le passage des autres gouttes dans cette direction.

Nous aurions pu simuler plus directement la traînée, en laissant de minuscules gouttes tout le long de la trajectoire empruntée par une goutte. Visuellement, cela se justifie pour les surfaces hydrophobes. Cette méthode aurait automatisé la modification du coefficient de rugosité due au passage d'une goutte, mais elle aurait fait augmenter de façon drastique le nombre de gouttes à traiter dans la simulation. Puisque le temps consacré aux opérations sur les gouttes (déplacement, fusion, rendu) est *grosso modo* proportionnel au nombre de gouttes, la simulation aurait été beaucoup plus coûteuse qu'avec la méthode retenue.

3.1.4 Adhérence à la surface et force de cohésion

Certaines des gouttes qui glissent sur une surface peuvent tomber de celle-ci à un moment ou à un autre de leur parcours. Ceci se produit lorsque l'accélération subie par



(a) (b)

FIG. 3.6: L'adhérence (a) est plus forte que la force normale, et la goutte reste sur la surface ; (b) est moins forte que la force normale, et la goutte tombera.

la goutte est plus forte que la force qui la retient à la surface. Notre modèle tient compte de ce phénomène ; chaque fois que l'accélération d'une goutte change (cela se produit lorsqu'une goutte passe d'un triangle à un autre), nous comparons la force qui s'exerce sur la goutte à une valeur d'adhérence d, fonction de la taille de la goutte (figure 3.6) :

$$d = D_{\text{triangle}}\pi r^2$$

où D_{triangle} est le coefficient d'adhérence du liquide sur la surface du triangle et r est le rayon de la goutte. Si l'on note $a_{\mathbf{N}}$ la composante de l'accélération de la goutte normale à la surface et m la masse de la goutte, la goutte quitte la surface et tombe en chute libre dans l'espace dès que $a_{\mathbf{N}}m > d$.

Nous avons déjà mentionné que

$$r = \sqrt[3]{\frac{3000m}{4\pi}};$$

nous pouvons donc en déduire que la goutte décolle de la surface si

$$a_{\mathbf{N}} > D_{\text{triangle}}\sqrt[3]{\left(\frac{3000}{4}\right)^2\frac{\pi}{m}}.$$

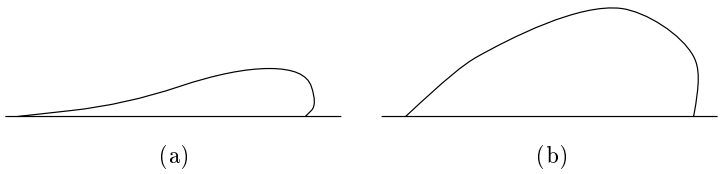


FIG. 3.7: Comparaison entre (a) la forme désirée et (b) la forme obtenue en utilisant des transformations affines.

3.2 Forme des gouttes

La trajectoire des gouttes est un aspect important de la simulation de l'évolution de gouttes sur des surfaces. Jusqu'à maintenant, nous avons fait l'hypothèse que les gouttes pouvaient être modélisées avec des sphères. Cependant, une goutte sur une surface n'a jamais exactement la forme d'une sphère, ni même d'une demi-sphère. Le plus souvent, elle aura une forme allongée dans la direction de la gravité, avec un fond relativement plat, qui repose sur la surface. Donc, si nous voulons faire une simulation plus réaliste, nous devons nous attarder sur la forme des gouttes.

Plusieurs options s'offrent à nous. La plus évidente consiste à simplement appliquer une déformation sur une sphère, de manière à modéliser l'action de la gravité sur les gouttes. Pour que cette approche soit intéressante, il faudrait que nous puissions exprimer la déformation à l'aide de transformations affines, afin de pouvoir la garder sous forme matricielle et d'utiliser les propriétés de ces transformations. Ce n'est pas un problème facile si nous désirons que nos gouttes aient l'air physiquement correctes (figure 3.7).

Nous pourrions nous servir d'objets mou [Blo97][WMW86a] [WMW86b]. Nous devrions alors trouver une fonction de champ qui simule bien la forme d'une goutte. Nous pourrions par exemple additionner un champ sphérique, associé à la position de la goutte, avec un champ de potentiel gravitationnel inversé qui ferait augmenter le volume des parties inférieures de la goutte. Un champ répulsif associé à la surface assurerait que la goutte ne pénètre pas dans cette dernière.

L'utilisation d'objets mous présente à première vue un avantage au niveau de la fusion. En effet, lorsque deux objets mous s'approchent suffisamment l'un de l'autre, leur fonction de champ s'additionnent, modélisant ainsi la fusion des deux objets. Ceci

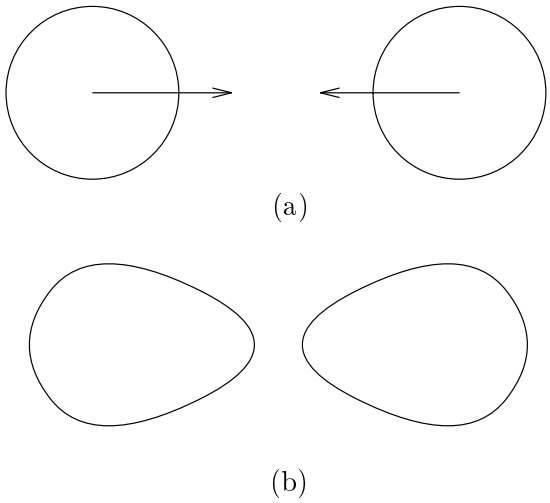


FIG. 3.8: Fusion de deux objets mous (vue en coupe). Des protubérances se forment le long de l'axe reliant les deux gouttes.

pourrait sembler intéressant, mais en y regardant de plus près, on s'aperçoit que cette fusion est peu réaliste. Dans les moments qui précèdent immédiatement la fusion, des protubérances se forment dans l'axe reliant les deux objets (figure 3.8). Plus les objets sont rapprochés, plus les protubérances sont importantes. Cet effet peut devenir agaçant, car il donne l'impression que les gouttes augmentent temporairement de volume. De plus, il donne des formes erronées aux gouttes. En effet, lorsqu'une goutte avance dans une direction, nous nous attendons à ce qu'elle ait une « queue » dans la direction opposée. Or, à la figure 3.8, la queue se forme dans la direction du mouvement, c'est-à-dire dans la direction contraire à ce que nous nous serions attendus.

Une autre approche consisterait à discrétiser le volume d'une goutte en utilisant des particules reliées par des ressorts. Nous pourrions ensuite soumettre ce système masses-ressorts à un algorithme de simulation physique qui déterminerait la forme de la goutte. En principe, cela devrait nous donner une forme de goutte assez réaliste ; cependant, les temps de calculs pourraient être assez longs. Pour réduire ces temps, nous pourrions discrétiser la surface de la goutte, plutôt que son volume. C'est l'approche qu'a développée Habibi [Hab97b] et que nous avons utilisée.

3.2.1 Modèle masse-ressort

Avant d'aller plus loin dans l'explication du modèle, nous adopterons deux hypothèses qui simplifieront grandement le travail de simulation. La première est que la forme d'une goutte n'influence pas son mouvement. Ainsi, pour tous les calculs de trajectoire, nous continuerons d'utiliser l'hypothèse qu'une goutte est sphérique. Cette hypothèse est vérifiée lorsque l'énergie cinétique de la goutte est beaucoup plus grande que son énergie de déformation [Hab97a]. La seconde hypothèse est que la forme d'une goutte atteint l'équilibre instantanément.

Dans notre système de simulation de trajectoire, nous utilisons deux algorithmes distincts : l'un pour calculer la trajectoire des gouttes lorsqu'elles évoluent sur une surface, et l'autre pour calculer leurs trajectoires lorsqu'elles évoluent dans l'espace. Il en sera de même pour la forme des gouttes : lorsqu'elles seront sur une surface, nous utiliserons le modèle développé par Habibi, alors que lorsqu'elles seront dans l'espace, nous leur donnerons simplement une forme sphérique. Dans le reste de cette section, nous décrirons le modèle proposé par Habibi. Pour plus de détails sur ce modèle, nous invitons le lecteur à consulter ses travaux [Hab97a] [Hab97b].

Il est difficile de décrire la forme d'une goutte. Celle-ci change constamment sous l'effet des champs de force et des interactions avec la surface. Le modèle de forme proposé par Habibi simule la surface de la goutte en satisfaisant un certain nombre de contraintes. Ces contraintes expriment un comportement causé par différents phénomènes physiques. Elles sont :

1. La tendance à garder un volume constant (phénomène associé à l'incompressibilité des liquides).
2. La tendance à minimiser sa surface de contact avec l'air (phénomène associé à la capillarité).
3. La tendance à maximiser la surface de contact avec l'objet sur lequel elle repose (phénomène associé à la capillarité des surfaces hydrophiles).
4. La tendance à être déformée par les forces externes, telle que la gravité (phénomène associé à l'équilibre entre les forces externes et la force de cohésion).

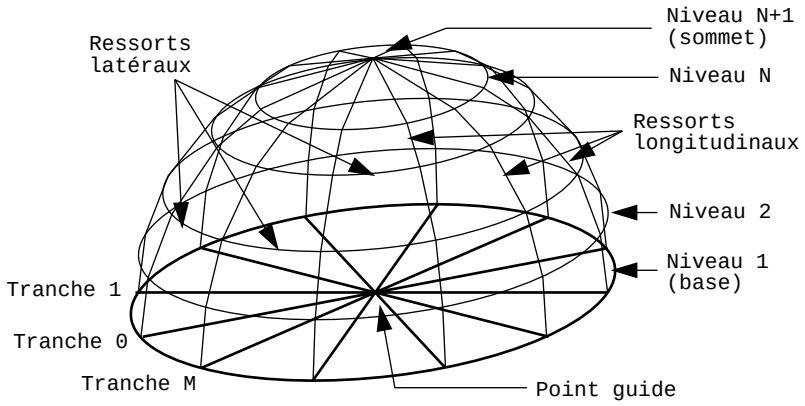


FIG. 3.9: Structure du modèle de goutte.

Ces contraintes peuvent s'opposer ou se compléter, et elles forment un système dynamique. Nous choisirons comme forme de nos gouttes la forme qui satisfait le mieux toutes ces contraintes.

Pour trouver la forme optimale, Habibi utilise un système de masses reliées entre elles par des ressorts. Ce système est gouverné par l'équation

$$m\mathbf{a} = \left(k(l_0 - l) - z\frac{dl}{dt}\right)\mathbf{u}, \tag{3.11}$$

où $\mathbf{a}$ est l'accélération, m la masse, k la raideur du ressort, z sa viscosité, l sa longueur, l_0 sa longueur au repos et $\mathbf{u}$ le vecteur unitaire indiquant la direction du ressort. Le système masse-ressort évolue pendant quelque temps, puis atteint un état d'équilibre. Cet état ne dépend que de la raideur des ressorts et de la force de gravité par rapport à la normale de la surface. Ainsi, si ces paramètres ne changent pas, par exemple lorsque nous simulons la forme de plusieurs gouttes de même masse sur un même polygone, nous n'avons pas besoin de recalculer un état d'équilibre, ce qui peut être avantageux au point de vue temps de calcul.

En supposant que la courbure de la surface sous la goutte est petite, nous pouvons représenter cette surface à l'aide d'un vecteur normal et d'un point guide, qui correspond dans notre système à la position de la goutte. Nous plaçons les masses dans l'espace en discrétisant la surface de la goutte en M tranches radiales et $N + 1$ niveaux (le dernier niveau est le sommet), ce qui requiert $NM + 1$ masses (figure 3.9).

Les masses à la base de la goutte sont attachées à la surface à l'aide de ressorts très

raides, perpendiculaires à la surface. Pour satisfaire la troisième contrainte (la maximisation de la surface de contact avec l'objet), ces points sont attachés au point guide avec des ressorts ayant une grande longueur au repos. Pour satisfaire la deuxième contrainte (la minimisation de la surface de contact avec l'air), chaque masse est attachée par des ressorts de raideur k_n et de longueur l_{0n} aux masses voisines latérales et par des ressorts de raideur k_l et de longueur l_{0l} aux masses voisines longitudinales.

Si nous satisfaisons les conditions

1. la longueur au repos des ressorts longitudinaux est nulle ;
2. les ressorts latéraux de la base sont presque rigides ;
3. la longueur au repos des ressorts latéraux est définie par ${l_0}_t^i = {l_0}_t^1 \frac{N+1-i}{N}$, où ${l_0}_t^i$ est la longueur au repos des ressorts latéraux du niveau i ;
4. toutes les autres forces appliquées sur les masses d'un même niveau sont égales ;

alors la position des masses du niveau i peut être obtenue par une translation et une homothétie des masses du premier niveau. De plus, dans ce cas, le volume de la représentation polyédrique de la goutte est donné par

$$V = \frac{S}{N^2}\left(\frac{h_{N+1}}{2} + 2\sum_{i=2}^{N} h_i\,(N+1-i)\right) \tag{3.12}$$

où S est l'aire du premier niveau et h_i la distance séparant le niveau i de la base. Étant donné que nous avons observé que S reste relativement constant[5], le volume de la goutte, défini par le polyèdre, reste constant si et seulement si le terme $\sum_{i=2}^{N} h_i\,(N+1-i)$ ne varie pas.

Pour satisfaire la première contrainte (volume constant), Habibi introduit un lien, qu'il nomme *lien solco*. Comme les ressorts, ce lien est caractérisé par une raideur, une viscosité et une longueur au repos. En connectant toutes les masses de chacun des niveaux à la surface supportant la goutte à l'aide d'un lien solco, nous nous assurons que le terme $\sum_{i=2}^{N} h_i\,(N+1-i)$, donc le volume de la goutte, demeure constant.

Une fois que la position d'équilibre des masses a été trouvée, nous nous en servons pour définir des courbes splines. Nous définissons d'abord une série d'arceaux passant

[5]Habibi [Hab97b] donne une méthode pour s'assurer que le volume de la goutte reste réellement constant ; cependant, nous n'avons pas eu besoin de l'utiliser pour nos simultaions.

par le sommet de la goutte. Ces arceaux servent ensuite à définir un ensemble de courbes splines parallèles. Ces courbes sont discrétisées en segments de ligne reliés en polygones, qui formeront notre surface. Il est intéressant de noter que nous calculons la forme de la goutte sans nous préoccuper de savoir si la surface résultante sera lisse ou non. Ce n'est qu'à la toute fin que nous habillons la forme, et cet habillage est nécessairement lisse (C^2).

Ce modèle n'est, comme tous les modèles, qu'une approximation d'un phénomène physique ; il est basé sur des observations du comportement des gouttes et tente de reproduire le mieux possible les aspects que nous jugeons importants pour notre application. Ce modèle ne conserve pas le volume des gouttes, mais, contrairement aux objets mous, les variations ne sont pas assez importantes pour être observables. En général, ce modèle semble bien décrire le comportement d'une goutte sur une surface lisse.

3.2.2 Déformations aléatoires

Il est rare, cependant, que des gouttes se déplacent sur des surfaces parfaitement lisses. La plupart du temps, celles-ci sont assez rugueuses pour induire des déformations chez la goutte. Habibi suggère de modéliser ces déformations en déplaçant de façon aléatoire les points de contrôle des splines. Ces déplacements sont fonction de la vitesse de la goutte. Dans nos simulations, nous avons utilisé une fonction qui a plus ou moins la forme d'une tente (figure 3.10). Lorsqu'une goutte ne bouge pas elle n'est pas déformée. Lorsqu'elle se met à bouger, l'amplitude de la déformation croît jusqu'à un maximum, puis décroît jusqu'à ce que la vitesse atteigne un certain seuil. Pour toutes les vitesses au-delà de ce seuil, la déformation est nulle.

3.3 Réalisation

Jusqu'ici, nous avons beaucoup parlé des équations régissant le mouvement des gouttes, mais assez peu de l'algorithme qui gère les déplacements. Nous allons combler cette lacune immédiatement en décrivant dans cette section le procédé par lequel les gouttes se déplacent sur une surface triangularisée.

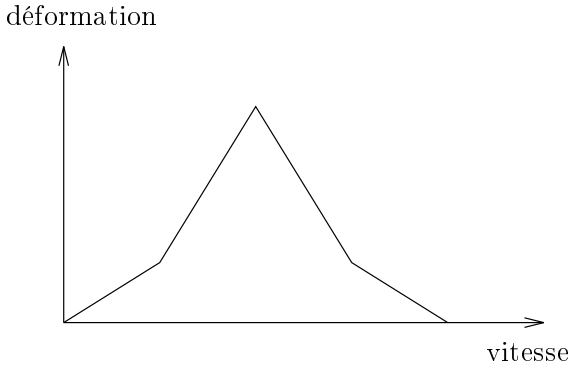


FIG. 3.10: Amplitude de la déformation en fonction de la vitesse de la goutte.

3.3.1 Déplacement à l'intérieur d'un triangle

Nous avons déjà mentionné, à la section 3.1, que quatre variables étaient nécessaires pour décrire l'état d'une goutte : sa position $\mathbf{p}$, sa vitesse $\mathbf{v}$, son accélération $\mathbf{a}$ et sa masse m. Pour nous servir de l'équation 3.10 dans un triangle donné, nous devons savoir à quel instant la goutte entre dans le triangle (instant que nous appellerons t_0), la position de celle-ci à son entrée dans le triangle (que nous appellerons $\mathbf{p}_0$), ainsi que sa vitesse à l'entrée ($\mathbf{v}_0$). Ainsi, tant que la goutte reste dans le triangle, nous pouvons trouver sa position à n'importe quel moment $t \geq t_0$ grâce à l'équation

$$\mathbf{p}(t) = \mathbf{p}_0 - \frac{m\mathbf{a}\,(t - t_0)}{k_v} + m\left(\mathbf{v}_0 + \frac{m\mathbf{a}}{k_v}\right)\left(\frac{e^{\frac{k_v}{m}(t-t_0)} - 1}{k_v}\right). \tag{3.13}$$

Lorsque la goutte sort d'un triangle pour entrer dans un autre, nous mémorisons les nouveaux t_0, $\mathbf{p}_0$ et $\mathbf{v}_0$ et les substituons dans l'équation ci-dessus. La façon évidente de déterminer si une goutte sort d'un triangle est de vérifier, chaque fois qu'on calcule sa position, si celle-ci est à l'extérieur du triangle. Nous pouvons cependant faire cela de façon beaucoup moins coûteuse. Lorsqu'une goutte entre dans un triangle, nous calculons, à l'aide de l'algorithme de Newton-Raphson, l'intersection de sa trajectoire, donnée par l'équation ci-dessus, avec les droites passant par chacun des trois côtés du triangle (en effet, la goutte peut ressortir par le côté par lequel elle est entrée). L'intersection la plus rapprochée dans le temps nous donne l'instant t_s auquel la goutte sort du triangle. Nous gardons en mémoire ce temps, ainsi que le côté du triangle par lequel la goutte sort et sa position de sortie. Donc, lorsqu'on doit calculer la position de la goutte à un instant

t, nous vérifions que $t \leq t_s$. Si c'est le cas, nous pouvons nous servir de l'équation 3.13. Sinon, nous devons transférer la goutte sur son nouveau triangle avant de poursuivre.

3.3.2 Passage d'un triangle à l'autre

Pour transférer une goutte sur son nouveau triangle, nous devons tout d'abord trouver quel est ce nouveau triangle. Durant la simulation, nous parvenons à faire ceci rapidement en nous servant d'un graphe de voisinage que nous construisons avant de commencer la simulation.

Avant de commencer à faire glisser les gouttes sur une surface, notre logiciel doit lire cette surface. Lors de cette lecture, il en profite pour construire un graphe de voisinage des triangles qui composent la surface. Ce graphe est fait de nœuds appelés `MeshNode`. La structure d'un `MeshNode` est la suivante :

```
class MeshNode
{
private:
    Triangle     mTriangle;     // Triangle pour lequel ce MeshNode
                                // contient les informations
                                // d'adjacence
    List<Voisin> mVoisinCote1;  // Liste des MeshNodes touchant le côté
                                // 1 du triangle mTriangle
    List<Voisin> mVoisinCote2;  // Liste des MeshNodes touchant le côté
                                // 2 du triangle mTriangle
    List<Voisin> mVoisinCote3;  // Liste des MeshNodes touchant le côté
                                // 3 du triangle mTriangle

    // ...
};
```

Deux triangles sont considérés voisins dès qu'ils se touchent. S'ils se touchent par leur sommet, nous considérons qu'ils se touchent par deux côtés à la fois. La classe `Voisin`, qui contient les informations sur les adjacences, est :

```
class Voisin
{
private:
    MeshNode     *mVoisin;      // Pointeur sur le voisin
    double       mDebut;        // Coordonnée du début de l'intersection
    double       mFin;          // Coordonnée de la fin de l'intersection
```

```
    // ...
};
```

Dans cette structure, nous paramétrisons le côté formé par les sommets **A** et **B** comme suit : un point **p** est représenté par la valeur du paramètre s tel que $\mathbf{p} = \mathbf{A} + s(\mathbf{B} - \mathbf{A})$, $0 \leq s \leq 1$. Les membres `mDebut` et `mFin` contiennent donc les valeurs de s délimitant l'intervalle du côté du triangle touchant au `mVoisin`. Ainsi, connaissant le côté et l'endroit où sort une goutte, nous pouvons rapidement trouver sur quel triangle elle entre en cherchant dans la liste de voisins appropriée.

3.3.3 Mouvement dans l'espace

Nous avons vu comment nous traitons le mouvement d'une goutte sur une surface. Or, d'après notre modèle (section 3.1.4), il est toujours possible qu'une goutte quitte la surface sur laquelle elle glisse et qu'elle commence à évoluer dans l'espace. On dit alors que la goutte est en chute libre.

Dans ce cas, les équations et l'algorithme régissant le déplacement de la goutte se simplifient énormément. L'équation régissant son mouvement devient

$$\mathbf{p}(t) = \mathbf{p}_0 + \mathbf{v}_0 (t - t_0) + \frac{\mathbf{g} (t - t_0)^2}{2} \tag{3.14}$$

où $\mathbf{p}_0$, $\mathbf{v}_0$ et t_0 sont respectivement la position, la vitesse et le temps au moment où la goutte quitte la surface. L'algorithme déterminant la position d'une goutte en chute libre applique tout simplement cette équation, quelle que soit la valeur de $tgeqt_0$. Il est à noter que nous n'avons pas implanté d'algorithme de détection de collisions entre une goutte en chute libre et les surfaces de la scène. Cela ne serait pas difficile, mais consommerait un temps de calcul non négligeable sans apporter beaucoup à la solution du problème qui nous préoccupe. Il serait possible de réduire ce temps, mais au prix d'un algorithme plus complexe, utilisant par exemple une subdivision hiérarchique de l'espace.

3.3.4 Détection des collisions entre les gouttes

Une fois que nous avons déplacé les gouttes sur la surface, nous vérifions si celles-ci s'intersectent et, si c'est le cas, nous les fusionnons. Comme nous l'avons expliqué plus

```
DetectionCollision(Modele m)
{
    Pour chaque triangle t du modèle m
    {
        Pour chaque goutte g0 qui se trouve sur le triangle t
        {
            Pour chaque goutte g1 qui se trouve sur le triangle t
            {
                Si (g0 et g1 ne sont pas la même goutte)
                {
                    Si (Norme(g1.position - g0.position) <=
                        (g1.rayon + g2.rayon))
                    {
                        Fusionner(g0, g1);
                    }
                }
            }
        }
    }
}
```

FIG. 3.11: Pseudo-code de l'algorithme de détection de collisions entre les gouttes

haut, notre algorithme de détection de collisions repose sur le découpage de la surface en triangles. Dans cette section, nous expliquerons brièvement le fonctionnement de cet algorithme.

Afin de faciliter le déroulement du programme, chaque goutte contient un pointeur vers le triangle sur lequel elle se trouve. De même, chaque triangle contient une liste de pointeurs vers les gouttes qui se trouvent sur lui. Notre algorithme de détection de collision se sert de cette information, comme on peut le voir dans le pseudo-code donné à la figure 3.11. Notre algorithme parcourt la liste des triangles du modèle. Pour chacun de ces triangles, il consulte la liste des gouttes qui se trouvent sur ce triangle et applique un algorithme simple qui détecte les collisions entre ces gouttes. L'algorithme considère que les gouttes sont des sphères parfaites. Étant donné que la forme des gouttes générées par l'algorithme de Habibi n'est pas sphérique, on doit ajuster empiriquement le rayon des gouttes pour que, lors du rendu, nous ne voyions pas deux gouttes qui se touchent et qui ne se fusionnent pas.

Notre algorithme est plus rapide qu'un algorithme de détection de collision sans partition des gouttes. Cependant, il se peut qu'il ne détecte pas certaines collisions. En effet, si deux gouttes sont suffisamment près l'une de l'autre pour se fusionner mais qu'elles se trouvent sur deux triangles différents, il n'y aura pas de fusion. Toutefois, de telles situations ne se produisent pas souvent, et elles durent rarement assez longtemps pour que nous les remarquions. Si jamais il était nécessaire de détecter toutes les collisions, nous pourrions modifier notre algorithme pour qu'il considère les gouttes se trouvant sur les triangles situés dans un certain rayon de la goutte `g0`.

3.4 Rendu

Puisque notre programme constitue une partie du système d'animation *Taarna*, nous devons utiliser les mécanismes fournis par ce dernier pour faire le rendu des gouttes et des traînées, c'est-à-dire le logiciel *RenderMan*, de *Pixar* [Ups89]. Avant de poursuivre, nous dirons quelques mots sur ce logiciel.

RenderMan est un logiciel de rendu. Un utilisateur peut, par des appels de fonction, placer les objets dans la scène, définir leur couleur et placer la caméra. De plus, et c'est ce qui rend le logiciel très flexible, il permet de définir des programmes, appelés *shaders*, qui encodent l'apparence des objets de la scène (surfaces, lumières, atmosphère) à l'aide de son langage de rendu. On peut associer ces programmes à chaque objet de la scène ; lors du rendu, *RenderMan* appelera ces programmes pour déterminer l'apparence des objets.

Pour faire le rendu d'une scène, *RenderMan* projette les objets de la scène sur la fenêtre de vue. Les objets sont ensuite convertis en pixels, la couleur de chaque pixel étant déterminée à l'aide des *shaders*. L'algorithme de rendu de *RenderMan* est basé sur un algorithme de *A-buffer*, par opposition à un algorithme de tracé de rayon.

Le système *Taarna* utilise *RenderMan* par l'intermédiaire des *shaders*. Il a donc toutes les capacités de *RenderMan*, mais aussi les mêmes limites. Ces limites se sont faites sentir lorsqu'est venu le temps de faire le rendu de certaines de nos gouttes.

La façon dont nous devons faire le rendu des gouttes et des traînées dépend évidem-

ment des caractéristiques du liquide qui compose les gouttes. Les gouttes étant le plus souvent un aspect secondaire des animations (elles ne constituent pas le centre d'intérêt principal du spectateur), nous ne voulons pas passer trop de temps à en faire le rendu. D'un autre côté, nos gouttes doivent présenter certaines caractéristiques importantes, de façon à ce que nous les reconnaissions lorsque nous les voyons.

Dans le cas de liquides opaques, cela ne pose généralement pas de problème. Nous pouvons définir une surface assez simple qui ressemble raisonnablement à la surface du liquide que nous modélisons, et le temps de rendu reste acceptable. Dans le cas des liquides transparents, comme l'eau, le problème est moins facile. Une des caractéristiques principales des gouttes d'eau est la caustique qu'elles créent. La seule façon de recréer cette caustique est d'utiliser un algorithme de tracé de rayon modifié qui dépose la lumière sur les surfaces en partant des lumières de la scène. Ceci crée une distribution indirecte d'énergie, à partir de laquelle le rendu de la scène est fait. Or, comme nous l'avons mentionné plus haut, *RenderMan* ne fait pas son rendu avec un tel algorithme. Nous pourrions essayer d'approximer les caustiques avec les *shaders* de volume de *RenderMan*, mais dans ce cas, le coût du rendu pourrait grimper de beaucoup, et il n'est pas garanti que le résultat serait pleinement satisfaisant. Nous étudions présentement une technique qui nous permettrait de simuler les caustiques des gouttes sans utiliser directement un algorithme aussi complexe. Nous en discutons à la section 5.4. Dans l'attente d'une solution satisfaisante, nous avons décidé de produire des animations ne contenant que des gouttes de liquides opaques.

Chapitre 4

Résultats et analyse

Forty-two.

Douglas Adams,
The Hitch Hicker's Guide to the Galaxy

Dans le chapitre précédent, nous avons discuté des objectifs que nous nous étions fixés au début du projet, ainsi que de l'approche que nous avons adoptée pour les atteindre. Nous avons aussi discuté de quelques détails de la réalisation de notre programme. Dans ce chapitre, nous verrons dans quelle mesure notre approche et nos techniques nous ont permis d'atteindre nos objectifs. Nous présenterons tout d'abord un compte rendu et une analyse du temps que requiert notre programme pour produire une animation. Ensuite, nous présenterons différentes animations, afin de montrer l'effet des paramètres de notre modèle sur le comportement des gouttes.

4.1 Vitesse d'exécution

Au début du projet, rappelons-le, nous nous étions fixés comme objectif d'obtenir une bonne rapidité d'exécution de la part de notre programme. En effet, la plupart du temps, nous aurons besoin de centaines, voire de milliers, de gouttes pour simuler de façon réaliste un phénomène. Nous voulons que les utilisateurs du système puissent obtenir des résultats rapidement ; nous n'avons jamais visé une simulation en temps réel, mais nous avions *grosso modo* comme objectif de passer, sans tenir compte du temps de

rendu, moins d'une minute par image en temps de calcul pour une simulation comprenant plusieurs milliers de gouttes évoluant sur un modèle composé de milliers de polygones.

Les temps de production de nos premières animations nous ont grandement encouragés. Notre système était très rapide. Nous n'avons pas chronométré le temps passé à calculer ces animations, mais nous avons pu constater que le temps requis pour calculer une image, en incluant le rendu, était inférieur à une minute. Pour étayer et détailler ces observations, nous avons inséré dans notre programme des fonctions de chronométrage. Nous avons utilisé la fonction UNIX `times`, qui retourne le temps consommé par un processus depuis sa naissance.

Nous avons vu au chapitre précédent que le fait d'utiliser une triangularisation du modèle facilitait le calcul de la position des gouttes ; nous avons aussi vu qu'une triangularisation fine du modèle réduit le temps passé à la détection des collisions goutte–goutte. Nous pouvons toutefois nous demander si le fait d'utiliser une triangularisation trop fine n'augmenterait pas le temps de calcul, à cause des intersections trajectoires–côté-des-triangles plus nombreuses, au point de rendre négligeables les gains apportés du côté de la détection des collisions goutte–goutte.

Dans le reste de cette section, nous discuterons des temps de calculs pour diverses parties de notre programme. Ensuite nous verrons comment il est possible d'optimiser notre scène pour réduire le temps de calcul. Tous les résultats présentés ici ont été obtenus lors de simulations effectuées sur un ordinateur Silicon Graphics Indigo^2, équipé d'une carte graphique Solid Impact, d'un microprocesseur MIPS R10000 tournant à 195 MHz et d'une mémoire vive de 128 mégaoctets.

4.1.1 Temps global

Nous avons tout d'abord mesuré le temps de simulation requis pour faire descendre des gouttes sur un plan, en tenant compte de toutes les étapes de la simulation : calcul des trajectoires, détections des collisions goutte–goutte et construction de la forme des gouttes. Nous avons fait des simulations comprenant de $n = 10$ à $n = 100$ gouttes disposées aléatoirement en suivant une distribution uniforme sur un plan subdivisé en $m = 2$ à $m = 512$ triangles. (Dans le reste de ce chapitre, nous continuerons d'employer

Nombre de triangles	Nombre de gouttes									
	10	20	30	40	50	60	70	80	90	100
2	1,0	2,0	3,2	4,1	5,2	6,1	7,3	8,0	9,4	10,6
8	0,9	1,7	3,0	3,8	5,0	6,2	7,2	7,9	9,4	10,1
32	0,9	1,6	3,2	4,0	5,2	6,2	7,2	8,1	8,9	10,2
128	1,0	2,2	3,0	4,1	5,3	6,5	7,3	8,5	9,1	10,2
512	1,0	2,1	3,2	4,2	5,4	6,4	7,4	8,3	9,5	10,1

TAB. 4.1: Temps de calcul (en secondes) pour un pas de simulation complète, en excluant le temps de rendu.

n pour désigner le nombre de gouttes et m pour désigner le nombre de triangles.) Les temps de calcul sont présentés au tableau 4.1.

De ces résultats nous pouvons déduire, et nous nous y attendions, que le temps de simulation augmente avec le nombre de gouttes. Il augmente même *grosso modo* linéairement, ce qui est assez surprenant, étant donné que la détection de collisions goutte–goutte est un algorithme $O(n^2)$. Cela indique que ce n'est pas cette partie de la simulation qui consomme la majeure partie du temps de calcul. Cela est corroboré par le fait que le temps de simulation est à peu près semblable, que le plan soit divisé en 2 triangles ou qu'il soit divisé en 512. En effet, notre algorithme de détection de collisions goutte–goutte détecte les collisions entre les gouttes situées sur un même triangle. Les gouttes étant réparties uniformément sur tout le plan, le nombre moyen de gouttes sur un triangle est inversement proportionnel au nombre de triangles. Le temps de calcul pour cette partie du programme devrait donc diminuer lorsque le nombre de triangles augmente. Cela n'apparaissant pas dans nos statistiques sur le temps total de simulation, nous pouvons inférer que le temps passé à la détection de collisions est petit par rapport au temps total.

Nous avons donc essayé de déterminer quelle opération consommait la majeure partie du temps de calcul. Nous avons mesuré le temps pris par l'algorithme de génération de forme pour calculer la forme d'une goutte. Nous avons trouvé un temps moyen d'environ 0,10 seconde. En mettant cette donnée en relation avec les résultats du tableau 4.1, nous

Nombre de triangles	Nombre de gouttes							
	1000	2000	3000	4000	5000	6000	7000	8000
2	0,6	2,6	6,0	12,8	21,2	31,8	45,8	59,4
8	0,2	0,8	1,8	3,1	4,9	7,9	11,2	16,7
32	0,1	0,3	0,7	1,3	1,9	2,9	3,8	5,8
128	0,1	0,2	0,4	0,8	1,3	2,1	2,9	4,2
512	$< 0,1$	0,1	0,3	0,6	1,0	1,4	2,1	3,1

TAB. 4.2: Temps de calcul (en secondes) pour un pas de simulation, en excluant la génération de la forme des gouttes.

voyons que cette étape de la simulation requiert toujours plus de 90 % du temps.

Pour une analyse plus fine de notre programme, nous avons désactivé la fonction de génération de forme et refait les mêmes simulations ; nous avons toutefois utilisé un nombre de gouttes plus grand afin que les erreurs de chronométrage ne soient pas trop importantes. Les temps obtenus sont présentés dans le tableau 4.2.

La première chose que nous remarquons est que même si le nombre de gouttes a été multiplié par au moins 10, les temps de simulation ont diminué de façon importante. Nous avons estimé que la génération de la forme des gouttes prend, lorsqu'on a entre 1000 et 8000 gouttes, entre 93,1 % et 99,9 % du temps de simulation. C'est considérable. Nous voyons aussi que le temps de simulation augmente encore avec le nombre de gouttes. Cette fois, cependant, la progression n'est plus $O(n)$, mais plutôt $O(n^2)$. Puisque, *a priori*, la seule opération de notre algorithme qui soit d'ordre $O(n^2)$ est la détection de collisions goutte–goutte, nous pouvons émettre comme hypothèse que c'est cette opération qui prend le plus de temps, après la génération des formes. Pour nous en assurer, nous avons chronométré séparément chacune des opérations effectuées par notre algorithme de simulation. Les résultats et la discussion sont présentés dans les sections suivantes. La section 4.1.2 traite des résultats obtenus pour la détection des collisions ; la section 4.1.3 discute des temps de calcul des trajectoires.

Nombre de triangles	Nombre de gouttes							
	1000	2000	3000	4000	5000	6000	7000	8000
2	0,6	2,5	6,0	12,7	21,1	31,6	45,7	59,2
8	0,2	0,8	1,7	3,0	4,8	7,8	11,1	15,5
32	0,1	0,3	0,6	1,2	1,8	2,8	3,7	5,6
128	0,1	0,3	0,4	0,7	1,2	2,0	2,8	4,0
512	$< 0,1$	0,1	0,3	0,5	0,9	1,3	2,0	2,9

TAB. 4.3: Temps de calcul (en secondes) pour la détection des collisions entre les gouttes

4.1.2 Détection des collisions

Si l'on se fie à la description de l'algorithme de détection de collisions goutte–goutte faite à la section 3.3.4, le temps de calcul de cette opération devrait être

$$t_c = c_0 + m\left(c_1 + \frac{n^2}{m^2}c_2\right)$$

où c_0, c_1 et c_2 sont des constantes de temps représentant le coût de différentes parties de l'algorithme. Nos résultats devraient donc montrer que le temps de calcul augmente comme le carré du nombre de gouttes et qu'il est soit proportionnel, soit inversement proportionnel, au nombre de triangles qui composent la surface, selon les valeurs de n, c_1 et c_2.

Pour vérifier cela, nous avons mesuré les temps requis des calculs de collisions entre les gouttes pour des simulations contenant de 1000 à 8000 gouttes placées aléatoirement sur un plan subdivisé en 2 à 512 triangles. Les résultats sont présentés dans le tableau 4.3. Il apparaît clairement que, quelle que soit la subdivision du plan utilisée, l'ordre du temps requis pour effectuer cette opération est légèrement supérieur à $O(n^2)$, ce qui n'est pas très éloigné de nos attentes.

Lorsqu'on regarde les temps de calcul en fonction du nombre de triangles qui composent le plan, la relation est beaucoup moins claire. Nous voyons très bien qu'elle diminue avec le nombre de triangles, mais la relation n'est pas inversement proportionnelle. Dans tous les cas, lorsqu'on passe d'une subdivision de 128 triangles à une subdivision de 512 triangles, le gain est minime, alors qu'il est appréciable lorsqu'on passe d'une

subdivision de deux triangles à une subdivision de huit triangles. On remarque aussi que, quel que soit le nombre de gouttes, la diminution du temps de calcul est semblable lorsqu'on passe d'un niveau de subdivision à l'autre. Nous pouvons émettre comme hypothèse que la constante c_1 est assez élevée, si on la compare à c_2. Ceci impliquerait que si le nombre de gouttes est très petit ou si le nombre de triangles approche le nombre de gouttes, notre algorithme est $O(m)$. Inversement, si le nombre de gouttes est très grand, notre algorithme devient $O(1/m)$.

Nous avons donc montré que, en ce qui a trait à la détection de collision, notre programme respecte assez bien les temps de calcul que nous avions prévu. Nous allons maintenant chronométrer et analyser les temps de calcul des trajectoires des gouttes.

4.1.3 Calcul des trajectoires

En théorie, le temps de calcul des trajectoires des gouttes devrait augmenter linéairement avec le nombre de gouttes. La progression en rapport avec le nombre de triangles subdivisant la surface où coulent les gouttes est moins claire. En effet, le temps de calcul devrait augmenter proportionnellement avec le nombre de triangles traversés durant un intervalle de temps[1]. Cependant, ce nombre n'est pas directement fonction du nombre de polygones ; il est plutôt fonction de la taille de ces polygones et de la vitesse atteinte par les gouttes au cours de la simulation. Nous pouvons cependant dire que, pour un modèle donné, si on augmente le niveau de subdivision du modèle, le nombre de triangles traversés par une goutte augmentera d'autant.

Pour avoir une idée du temps de base consacré au calcul de la trajectoire des gouttes, nous avons désactivé la détection de collisions goutte–goutte et nous avons placé de façon aléatoire de 1000 à 8000 gouttes, d'abord sur un plan subdivisé en deux triangles, ensuite sur un plan subdivisé en 512 triangles. Dans chacun des cas, nous avons utilisé un intervalle de temps tel que les gouttes ne traversent pas plus d'un triangle par intervalle. Les résultats de ces simulations sont présentés dans le tableau 4.4. On peut voir que l'effet du niveau de subdivision du plan sur les temps de calcul est presque nul.

Ensuite, pour voir l'influence du nombre de triangles traversés par une goutte lors

[1] L'intervalle de temps a été défini à la section 3.1.3.

Nombre de triangles	Nombre de gouttes							
	1000	2000	3000	4000	5000	6000	7000	8000
2	$< 0,1$	$< 0,1$	0,1	0,1	0,1	0,1	0,2	0,2
512	$< 0,1$	$< 0,1$	0,1	0,1	0,1	0,2	0,2	0,2

TAB. 4.4: Temps requis (en secondes) pour le calcul des trajectoires. Nombre de triangles traversés par intervalle : 0 ou 1.

Nombre de polygones traversés	Temps (*sec.*)
0	0,1
1	0,5
2	0,8
3	1,2
4	1,4
5	1,7
6	2,1

TAB. 4.5: Temps de calcul (en secondes) des trajectoires en fonction du nombre de polygones traversés.

d'un intervalle de temps, nous avons généré 4000 gouttes, toutes placées au même endroit, au haut d'un plan divisé en 512 triangles. Nous avons ajusté les intervalles afin que ces gouttes traversent en moyenne de zéro à six triangles par intervalle de temps. Les résultats sont présentés dans le tableau 4.5. Nous pouvons constater que la progression du temps en fonction du nombre de polygones traversés est presque linéaire. Si nous notons p le nombre de polygones traversés par une goutte durant un intervalle de temps, nous pouvons dire que notre algorithme semble $O(p)$.

4.1.4 Optimisation de la subdivision

Nous avons pu voir, dans les sections précédentes, que le niveau de subdivision de la surface influence le temps de calcul de deux façons opposées. Plus le niveau de subdivision

est élevé, plus le temps passé à la détection de collisions goutte–goutte est petit, et plus le temps passé au calcul des trajectoires risque d'être grand.

Cela signifie qu'en principe, il est possible de trouver un niveau de subdivision optimal, qui minimise le temps de calcul total pour un nombre de gouttes donné. Pour cela, il suffit de faire quelques simulations de test avant de commencer la simulation finale. Ces tests servent à déterminer les statistiques suivantes :

1. le nombre moyen de polygones traversés par une goutte durant un intervalle de temps ;
2. le temps requis pour faire la détection de collisions goutte–goutte sur une surface ayant différents niveaux de subdivision ; les gouttes doivent traverser un nombre fixe de triangles par intervalle de temps ;
3. le temps requis pour traverser différents nombres de triangles durant un intervalle de temps donné ; les gouttes coulent sur une surface subdivisée en un nombre fixe de polygones.

Par exemple, supposons que nous voulions faire une longue simulation dans laquelle 4000 gouttes doivent évoluer sur un plan. Nous voulons déterminer le niveau optimal de subdivision de ce plan. Nous avons effectué des tests avec un plan subdivisé en 512 triangles et nous avons déterminé qu'en moyenne, durant un intervalle, les gouttes traversent quatre triangles. Nous avons aussi déterminé (tableau 4.3, quatrième colonne) les temps requis pour différents niveaux de subdivision lorsque les gouttes traversent moins de deux (*i.e.* zéro ou un) triangles. Finalement, nous avons déterminé (tableau 4.5) les temps requis pour traverser différents nombres de triangles durant un intervalle. Nous pouvons maintenant construire un tableau donnant une estimation des temps de calcul pour différents niveaux de subdivision de la surface. Ces temps sont donnés dans le tableau 4.6. Nous pouvons constater que le temps de simulation optimal est de 1,5 seconde, avec un niveau de subdivision de 128 triangles.

4.1.5 Remarques générales

Nous avons pu voir que les temps de calcul des trajectoires et des détections de collisions sont relativement courts, et qu'il est possible de les optimiser en subdivisant

Niveau de subdivision	Nombre de triangles traversés durant un intervalle (p)	Temps de base	Temps pour traverser p triangles	Temps total
2	0	12,7	0,1	12,8
8	0	3,0	0,1	3,1
32	1	1,2	0,4	1,6
128	2	0,7	0,8	1,5
512	4	0,5	1,4	1,9

TAB. 4.6: Temps de calcul (en secondes) pour différentes subdivisions, $n = 4000$.

adéquatement la surface sur laquelle coulent les gouttes. Cependant, tous ces temps restent largement inférieurs aux temps requis pour la génération de forme. Il serait intéressant de voir dans quelle mesure l'algorithme de génération de forme pourrait être optimisé. Nous reviendrons plus en détail sur ce sujet au chapitre 5.

Nous pouvons dire que nos objectifs de rapidité ont été atteints. Même dans les simulations les plus complexes, il est rare que nous aurons à utiliser plus de quelques milliers de gouttes ; nous avons vu que, dans la plupart de ces cas, il est possible d'obtenir des temps de simulation (pour les trajectoires et les détections de collisions) inférieurs à deux secondes.

4.2 Animations

Nous avons discuté jusqu'ici de la rapidité et de l'efficacité de nos algorithmes. Or, en infographie, ce ne sont pas les seuls aspects importants d'un logiciel. La qualité des résultats visuels comptent. C'est cependant un résultat difficile à quantifier, car sujet à la subjectivité de chacun. Puisqu'il est difficile de juger de la qualité d'une animation en se basant uniquement sur des images extraites de ces animations, nous avons produit un court vidéo présentant les principales caractéristiques de notre programme. Pour le bénifice des lecteurs n'ayant pas accès à ce vidéo, nous reproduirons les images les plus significatives de ces animations. Ces animations sont aussi disponibles en format *QuickTime* sur le site WWW du laboratoire d'infographie, à l'adresse `http://www.iro.umontreal.ca/labs/infographie/theses/fourniep`.

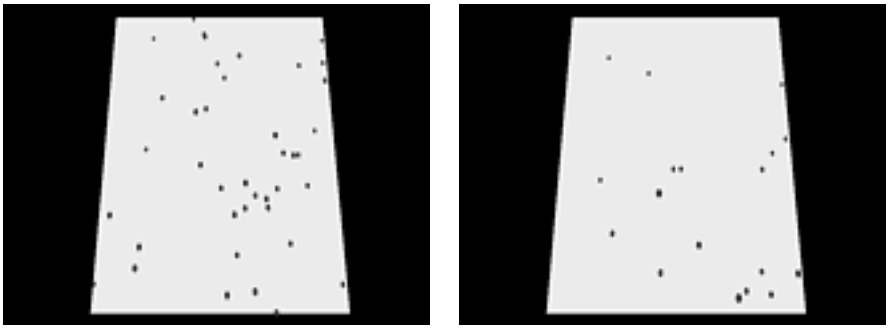


FIG. 4.1: Gouttes de masse variable.

Dans les sections qui suivent, nous discuterons brièvement de chacune des animations présentées sur la cassette fournie en annexe.

4.2.1 Gouttes de masse variable

Nous avons généré, sur un plan incliné, des gouttes de masse aléatoire. Le ratio entre la plus petite masse et la plus grande masse est de 1:4. Les gouttes ne laissent pas de traînées, mais elle modifient la rugosité de la surface lors de leur passage. On peut voir dans l'animation que les grosses gouttes descendent plus rapidement que les petites gouttes (figure 4.1), à cause du frottement proportionnellement plus élevé pour ces dernières. On peut aussi voir dans l'animation des gouttes qui entrent en collision et qui s'unissent.

4.2.2 Gouttes de viscosité variable

Sur chacun des trois plans (figure 4.2) coulent des gouttes de viscosité différente. Sur le plan de gauche, la viscosité est de 1 kg/s, sur celui du centre, elle est de 10 kg/s, alors que sur celui de droite, elle est de 100 kg/s. On peut voir clairement dans l'animation que la viscosité influence la vitesse que peuvent atteindre les gouttes.

4.2.3 Variation de l'adhérence à la surface

Pour ces trois animations, nous avons généré une série de gouttes ayant toutes la même masse sur une sphère légèrement tronquée aux deux pôles. Dans la première animation (figure 4.3), nous avons utilisé une adhérence de 10 N ($D_{\text{triangle}} = 8,3$ g/ms^2),

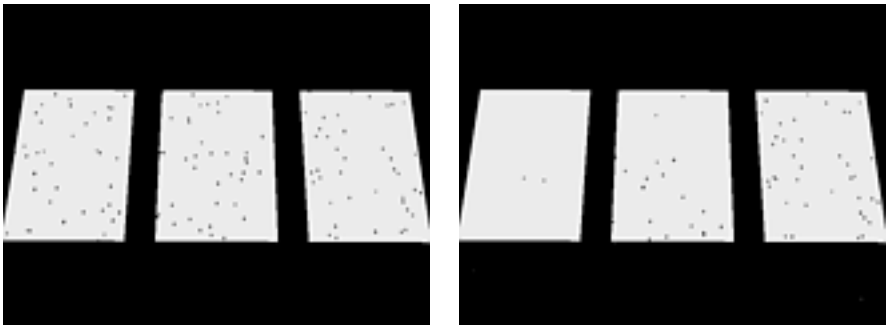


FIG. 4.2: Gouttes de viscosité variable.

FIG. 4.3: Adhérence = 10 N.

qui est suffisante pour garder les gouttes attachées à la surface quel que soit l'angle de la surface. Dans la seconde animation (figure 4.4), nous avons utilisé une adhérence de 5 N ($D_{\text{triangle}} = 4,1$ g/ms^2). On peut voir les gouttes quitter la surface environ au trois quarts de la sphère, alors que l'angle de la surface avec l'horizontal forme un angle de 45 degrés. Dans la dernière animation (figure 4.5), l'adhérence est nulle, et les gouttes quittent la sphère à mi-parcours, dès que l'angle de la surface atteint 90 degrés.

FIG. 4.4: Adhérence = 5 N.

FIG. 4.5: Adhérence = 0 N.

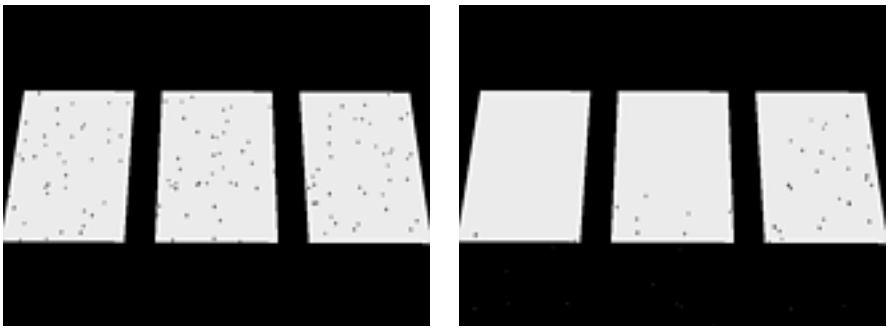


FIG. 4.6: Variations de la rugosité de la surface.

4.2.4 Variation de la rugosité de la surface

Nous avons généré aléatoirement des gouttes sur trois plans identiques, excepté pour la rugosité. Dans l'animation, le plan de gauche est le plus lisse, et celui de droite, le plus rugueux. On peut observer (figure 4.6) que plus la rugosité est grande, plus les gouttes ont de la difficulté à descendre.

Dans l'animation, on peut observer un phénomène intéressant à l'intérieur du rectangle rouge, que nous avons superposé au plan de droite[2]. Lorsque la goutte du haut commence à suivre la traînée (invisible dans l'animation) laissée par la goutte qui la précède, elle accélère subitement. Cela est dû à la baisse de rugosité causée par le passage de la goutte précédente. À un certain moment, la goutte du haut rejoint la goutte qui la précède et s'unit avec elle. La vitesse de la nouvelle goutte baisse brusquement (si on la compare avec la vitesse de la goutte du haut) car il n'y a plus de traînée pour faciliter son passage.

[2]Ce rectangle n'est pas présent dans l'animation disponible sur le site WWW du laboratoire.

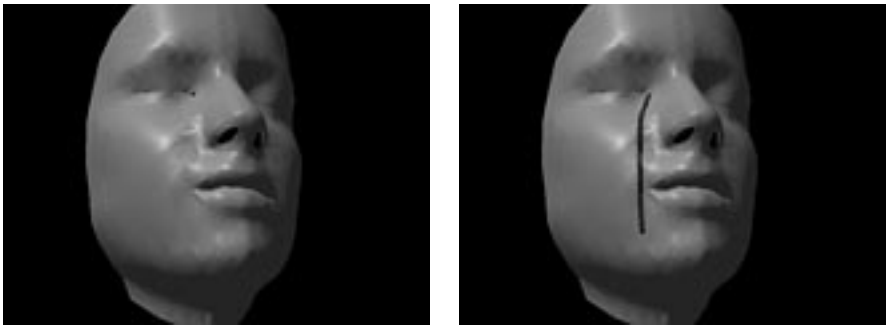


FIG. 4.7: Une larme coule sur une joue.

FIG. 4.8: Des gouttes coulent sur un visage.

4.2.5 Une larme coule sur une joue

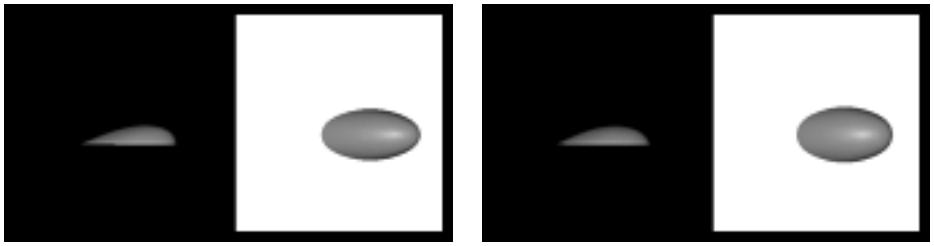
Afin de montrer le réalisme des animations produites par notre système, nous avons fait couler une larme sur un visage (figure 4.7). Le visage est composé de plus de 2000 triangles, et la goutte laisse une traînée derrière elle.

Ensuite, nous avons généré de façon aléatoire des gouttes sur le visage, et nous les avons laissées couler (figure 4.8). Quand les gouttes atteignent des discontinuités du visage, c'est-à-dire les fentes des yeux, les narines et la fente entre les deux lèvres, elles se détachent de la surface et se déplacent librement dans l'espace. Nous ne faisons pas de détection de collisions entre les gouttes dans l'espace et les objets qui composent la scène.

4.2.6 L'effet d'un champ de force sur une goutte

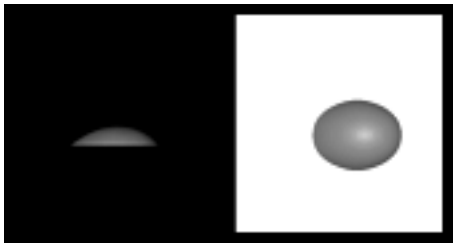
Finalement, nous montrons l'effet d'un champ de force (par exemple, la gravité) sur la forme des gouttes. On voit (figure 4.9) que la goutte est déformée en fonction du champ de force. La déformation n'est pas qu'un simple cisaillement ; on peut observer une migration de la masse de la goutte dans la direction du champ de force.

À la vue des animations que nous avons réalisées, nous pouvons dire que notre programme produit des simulations raisonnablement réalistes. Cependant, il y a toujours place à l'amélioration, tant du côté du réalisme que du côté des temps de simulation. C'est de ce que nous discuterons au chapitre suivant.

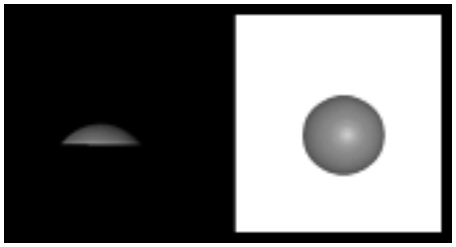


(a) $\phi = 0$

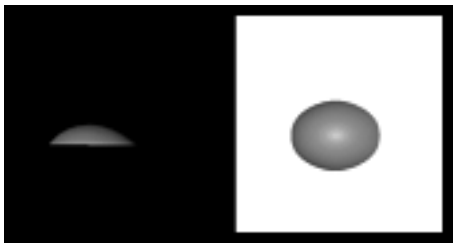
(b) $\phi = \frac{\pi}{5}$



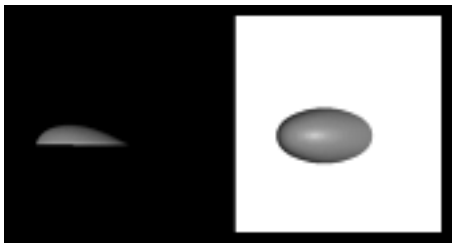
(c) $\phi = \frac{2\pi}{5}$



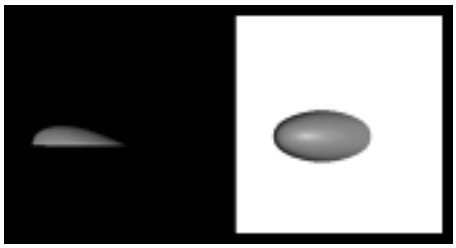
(d) $\phi = \frac{\pi}{2}$



(e) $\phi = \frac{3\pi}{5}$



(f) $\phi = \frac{4\pi}{5}$



(g) $\phi = \pi$

FIG. 4.9: L'effet d'un champ de force sur une goutte. Le champ de force est dans le plan XZ. Pour chaque valeur de l'angle du champ de force (mesuré par rapport à l'axe X), la goutte est représentée vue de $Y+$ (fond noir) et vue de $Z+$ (fond blanc).

Chapitre 5

Améliorations et travaux futurs

Sa chevelure était longue et soyeuse, ses dents d'une blancheur nacrée, mais cela ne faisait que mieux ressortir l'horreur de ses yeux vitreux [...]

Mary W. Shelley,
Frankenstein ou le Prométhée moderne

Nous avons fait état, dans le chapitre précédent, des résultats que nous avons obtenus avec notre programme. Bien qu'ils soient à notre avis suffisamment bons, il y a toujours place à amélioration. De plus, nous avons réalisé que notre programme pourrait servir de base pour la réalisation d'autres types de simulations. Dans les sections qui suivent, nous traiterons des différentes améliorations et fonctionnalités supplémentaires que nous pourrions incorporer à notre programme. Nous traiterons tout d'abord des améliorations que nous pourrions apporter au comportement des gouttes, ainsi que des applications auxquelles notre système pourrait servir, au prix de quelques modifications. Ensuite, nous discuterons des approches que nous aurions pu adopter pour modéliser les traînées laissées par les gouttes, puis nous verrons comment nous pourrions améliorer, en termes de rapidité et de réalisme, l'algorithme de génération de forme. Finalement, nous glisserons quelques mots sur la façon dont nous pourrions améliorer le rendu des gouttes, en particulier celui des gouttes d'eau.

5.1 Comportement

Comme nous l'avons vu au chapitre précédent, l'algorithme qui gère le comportement des gouttes est rapide et donne de bons résultats. Nous nous sommes cependant concentrés sur l'essentiel ; plusieurs aspects du comportement des gouttes ne sont donc pas simulés. Souvent, ces aspects ne sont pas capitaux pour obtenir une simulation correcte, mais il serait tout de même intéressant de pouvoir les intégrer afin d'obtenir un niveau de réalisme plus élevé. Dans les sections qui suivent, nous discuterons des améliorations que nous pourrions apporter à notre algorithme de déplacement de gouttes.

5.1.1 Détection de collisions dans l'espace

Nous avons mentionné précédemment que lorsque les gouttes se déplacent dans l'espace (par opposition à lorsqu'elles se déplacent sur une surface), nous ne détectons ni les collisions entre les gouttes, ni les collisions entre les gouttes et les autres objets de la scène. Nous avions justifié cela par le fait que nous voulions seulement simuler le comportement de gouttes se déplaçant sur une surface. Cependant, pour plus de réalisme, nous pourrions ajouter un algorithme qui ferait la détection des collisions dans l'espace.

Pour que les collisions soient détectées correctement, il faudrait utiliser une formule analytique donnant l'endroit et l'instant où une goutte entre en collision avec un objet. Dans l'espace, la trajectoire des gouttes est parabolique (voir l'équation 3.14) ; la détection de collisions revient donc à intersecter cette parabole avec chacun des objets de la scène. Cependant, les objets peuvent avoir une forme quelconque, et cette opération pourrait se révéler extrêmement coûteuse pour certains d'entre eux. Pour réduire ce coût, nous pourrions envelopper chaque objet dans un volume englobant, comme cela se fait dans l'algorithme de tracé de rayon [Whi80] [RW80] [Tot85] [KK86]. Nous pourrions même utiliser une hiérarchie de volumes englobants lorsque la scène ou les objets sont particulièrement complexes [RW80] [Weg84] [KK86] et ainsi obtenir une détection de collisions dans l'espace en $O(\log(n))$, où n est le nombre d'objets dans la scène.

Une autre façon de rendre la détection de collision dans l'espace plus efficace est d'utiliser la subdivision de l'espace. Cette technique a été proposée dans le but d'accélérer

les algorithmes de tracé de rayon. La technique la plus simple consiste à diviser tout l'espace régulièrement en cubes de mêmes dimensions. Ensuite, on attache à chacun des cubes la liste des objets ou des parties d'objets qu'il contient. Le calcul d'intersection s'en trouve accéléré car, puisque l'on connaît l'équation de la trajectoire d'une goutte, on peut déterminer quels cubes de la subdivision sont traversés par la goutte. Sachant cela, nous pouvons restreindre la liste des objets potentiellement intersectés par la goutte aux objets contenus dans les cubes. Il est possible d'améliorer cette technique en utilisant une subdivision adaptative de l'espace [DS84] [Gla84] ou en hiérarchisant la subdivision [SB87]. La technique de subdivision de l'espace est cependant moins appropriée que l'utilisation de volumes englobants, à cause de la nécessité de mettre à jour les listes d'objets contenus dans les subdivisions lorsque les objets se déplacent. Il est toujours possible d'ajouter le temps comme facteur de subdivision, mais la quantité de données à conserver et à traiter augmente alors considérablement [Gla88].

5.1.2 Évaporation et dépôts

Un phénomène bien connu de tous ceux qui font la vaisselle et qui ne l'essuient pas est que le volume des gouttes diminue avec le temps à cause de l'évaporation. Notre système ne tient pas compte de ce phénomène. En fait, nous posons même comme hypothèse (section 3.1.1) que la masse des gouttes est constante. Il serait cependant intéressant de voir comment nous pourrions incorporer ce phénomène dans notre système d'une façon efficace. Peut-être pourrions-nous, si nous exprimons la masse d'une goutte comme une fonction continue du temps, dériver une équation de position similaire à l'équation 3.10.

L'évaporation pourrait s'appliquer non seulement aux gouttes, mais aussi aux traînées. Il faudrait se pencher sur un modèle qui simule bien l'assèchement d'une traînée, afin que celle-ci ne disparaissent pas d'un seul coup, mais de manière à ce qu'elle s'amincisse d'une façon graduelle et inégale. Si nous gardons nos traînées sous forme de textures, nous pourrions nous inspirer des algorithmes d'amincissement de lignes utilisés pour la reconnaissance de motifs ou pour la numérisation d'images tracées au crayon [Cat78] [Cyc94] [SDPO81]. Le taux d'évaporation pourrait être proportionnel à la chaleur de la surface, ou encore à l'intensité de la lumière reçue.

Nous pourrions, comme Dorsey *et al.* [DPH96] et Curtis *et al.* [CAS+97] l'ont fait, transporter de la matière dans nos gouttes. Cette matière pourrait être présente au moment de la création des gouttes ou être dissoute de la surface lors du passage d'une goutte. La matière dissoute se déposerait lors de l'évaporation des gouttes et des traînées. La simulation du transport de matière nous permettrait de reproduire, tout comme Dorsey *et al.*, les marques laissées par le passage répété de l'eau sur une surface. Ceci pourrait être accompli à l'aide d'une texture qui enregistrerait les dépôts et prélèvements faits sur la surface par les gouttes. De plus, si le liquide qui s'évapore a un haut taux de matière dissoute, comme par exemple la peinture et le sang, nous pourrions simuler le changement d'aspect visuel des traînées qui sèchent. Nous pourrions simuler l'apparition de craquelure, ou encore nous pourrions modéliser la formation de stalagmites et de stalagtites par accumulation de matière. La façon la plus facile de simuler ces phénomènes serait d'utiliser (comme nous le faisons actuellement) des polygones pour modéliser les traînées. Nous pourrions adjoindre à ces polygones des textures ou des *shaders* dont l'un des paramètres serait le degré d'assèchement du liquide. Pour simuler la formation de stalagtites et stalagmites, nous pourrions remplacer les polygones de traînées qui se superposent par un modèle tridimensionnel représentant la colonne de matière en formation.

Nous pourrions aussi simuler l'augmentation de la viscosité du liquide qui s'évapore et qui laisse des traînées plus visqueuses sur la surface. Dans ce cas, il faudrait modifier notre équation du mouvement (car nous considérons que le coefficient de friction visqueuse est constant), ou alors ajouter un nouveau paramètre à notre modèle et modifier notre algorithme de simulation afin d'approximer le mieux possible cet effet.

5.1.3 Ajouts de niveaux de détails

Nous avons conçu notre programme afin qu'il simule bien les grosses gouttes, qui coulent assez rapidement sur la surface. Il serait intéressant d'intégrer à notre programme d'autres algorithmes qui simuleraient plus adéquatement le comportement des petites gouttes, ainsi que les phénomènes de fusion et de séparation. Aussi, il serait intéressant d'avoir un algorithme de modélisation de masses de liquide tel que celui proposé par

Foster et Metaxas [FM96]. Ainsi, nous pourrions modéliser le comportement d'accumulations de liquide formées par le ruissellement de gouttes. Nous pourrions aussi utiliser un algorithme de propagation de vagues, par exemple celui décrit par Ts'o et Barsky [TB87], pour simuler les vagues causées par la chute d'une goutte dans une flaque.

Avec tous ces algorithmes, notre système pourrait simuler le comportement de « morceaux » de liquide à de multiples niveaux, que ce soit des gouttes ou des flaques. Nous pourrions, en fonction de la masse du « morceau », choisir l'algorithme le mieux adapté à la simulation.

5.2 Traînées

Comme nous l'avons mentionné à la section 3.1.3, les traînées auraient pu être réalisées à l'aide de textures. Nous croyons que cette approche, bien que présentant plus de difficultés d'implantation, est plus flexible et plus efficace que l'utilisation de polygones, la technique que nous avons retenue. En effet, avec l'utilisation de textures, la complexité de la scène n'augmente pas avec l'ajout des traînées. Nous épargnons donc sur le temps consacré au calcul de visibilité et au rendu. De plus, les textures permettent de modifier n'importe quel paramètre de la surface (coefficients de réflexion, couleur, orientation, etc.), pas seulement la couleur. Il vaudrait donc la peine de prendre le temps de les incorporer dans notre programme. Pour cela, il faudrait écrire un algorithme qui convertit les informations que nous avons sur la traînée (son point de départ, son point d'arrivée et sa largeur) en texels. Nous pourrions utiliser un algorithme de rendu par balayage (*scan conversion* [FvDFH90], section 3.6). Cette opération n'est pas du tout évidente, à cause des débordements des traînées sur les triangles voisins (voir la figure 3.4 du chapitre 3). Il faudrait sans doute utiliser l'algorithme de découpage des traînées que nous utilisons actuellement (section 3.1.3), puis convertir les polygones obtenus à l'aide de l'algorithme de rendu par balayage.

Nous avons aussi mentionné, à la section 3.1.3, que nous aurions pu modéliser les traînées à l'aide de gouttes laissées tout le long du chemin des gouttes. Dans le cas des surfaces hydrophiles, l'apport de ce modèle au plan visuel est limité, car les traînées res-

tent rarement sous formes de gouttes. Cependant, dans le cas des surfaces hydrophobes, comme le teflon, l'utilisation des gouttes pour modéliser la traînée serait beaucoup plus réaliste. Lors de la réalisation, nous avions rejeté cette approche car nous craignions que son utilisation diminue de façon importante l'efficacité de notre programme. La perte d'efficacité serait principalement due à l'augmentation du temps passé dans l'algorithme de génération de forme, la partie la plus coûteuse de notre programme (section 4.1.1). Cependant, puisque les petites gouttes ne se déplacent presque pas, nous pourrions, une fois leur forme calculée, la garder en mémoire et l'utiliser jusqu'à ce que le champ de force appliqué sur la goutte change significativement.

Nous pourrions incorporer à notre programme un modèle de traînée utilisant des gouttes, en plus du modèle polygonal ou à textures. Notre programme pourrait générer les traînées en se servant des deux modèles, privilégiant l'un plutôt que l'autre en fonction du caractère hydrophile de la surface.

5.3 Forme

Du côté de la génération de la forme des gouttes, la principale amélioration à apporter consisterait à réduire le temps de calcul. En effet, nous avons vu au chapitre précédent que cette partie de la simulation prend souvent plus de 90 % du temps de calcul. Une optimisation de cet algorithme améliorerait beaucoup les temps de simulations. Déjà, en expérimentant avec les paramètres qui gouvernent la simulation de la forme de la goutte, nous avons pu réduire le temps consacré à cet algorithme de près de 50 %, et cela sans modifier la forme résultante.

Nous pourrions donner le choix à l'utilisateur de visualiser les gouttes soit à l'aide de la forme que nous calculons, soit à l'aide de demi-sphères. Ainsi, un utilisateur pourrait avoir un aperçu de la position des gouttes et apporter rapidement des correctifs aux paramètres de la simulation du mouvement.

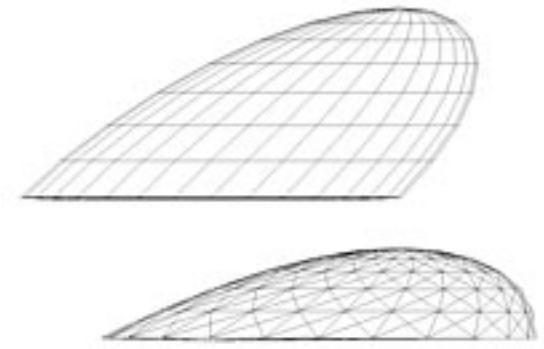
Nous pourrions aussi utiliser différents algorithmes de génération de forme en fonction de la position de la caméra. Ainsi, si la caméra est assez éloignée, nous pourrions utiliser comme représentation une demi-sphère déformée par un cisaillement. Nous pou-

vons voir, à la figure 5.1, que la différence entre la forme obtenue par l'algorithme de Habibi et la demi-sphère déformée n'est pas énorme, surtout lorsqu'on la regarde du dessus. Nous croyons que, lorsque placé assez loin, le spectateur ne pourrait discerner entre les deux. Nous pouvons pousser cette simplification plus loin : à la limite, nous pourrions représenter les gouttes à l'aide d'un seul polygone faisant face à la caméra. Ces représentations plus simples nous éviteraient une simulation coûteuse de l'action d'un champ de force sur un ensemble de masses reliées par des ressorts tout en donnant un résultat visuel satisfaisant. Toutefois, comparativement à l'algorithme de Habibi, il est moins facile de paramétriser la déformation de la sphère en fonction du champ de force dans lequel se trouve la goutte.

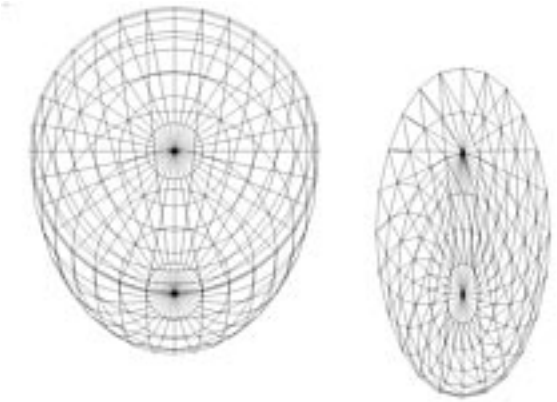
Nous pourrions aussi modifier l'algorithme de génération de forme afin qu'il puisse modéliser des déformations plus complexes, comme celles qui se produisent lorsque deux gouttes se fusionnent, lorsqu'une goutte (ou une partie d'une goutte) quitte la surface, ou lorsqu'une goutte entre en collision avec une surface ou une flaque. Nous pourrions, comme nous l'avons suggéré pour la simulation des comportments (section 5.1.3), utiliser un ensemble d'algorithmes adaptés pour bien simuler chaque phénomème. Lorsqu'un certain phénomène se produit, notre programme appellerait l'algorithme approprié, à condition que le spectateur soit placé de façon à pouvoir en apprécier tous les détails. Ainsi, nous obtiendrions un bon niveau de détail sans que les coûts de la simulation augmentent en flèche. Il faudrait cependant s'assurer que la transition entre les modèles soit douce, afin qu'il n'y ait pas de « discontinuité » lors du passage d'une représentation des gouttes à une autre.

5.4 Rendu

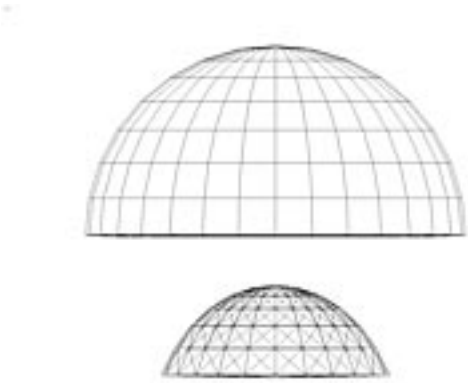
Il n'est pas facile de faire le rendu de l'eau ; c'est un liquide sans couleur, présentant à la fois des propriétés réfléchissantes et réfractives importantes. Les chercheurs qui se sont penchés sur la modélisation des vagues ont tous proposé un modèle d'illumination pour faire le rendu de l'eau [FR86] [Pea86] [TB87]. Cependant, leurs modèles sont orientés vers le rendu de grandes masses d'eau, avec une surface assez planaire ; ils s'adaptent



(a) Vue de X+



(b) Vue de Y+



(c) Vue de Z+

FIG. 5.1: Comparaison d'une demi-sphère déformée par cisaillement et de la forme générée par l'algorithme de Habibi.

mal au rendu de formes d'eau placées dans un environnement complexe.

Lorsque nous avons fait nos animations, nous avons tenté de créer un modèle d'illumination pour nos gouttes. Cependant, nous avons consaté qu'il était difficile de simuler de façon réaliste l'aspect de l'eau avec *RenderMan*. C'est pourquoi, dans toutes nos animations, nous utilisons des liquides opaques. Il serait toutefois bien pratique d'avoir un modèle d'illumination qui puisse faire un rendu réaliste de gouttes d'eau. Nous avons donc essayé de cerner ce qui fait qu'une goutte d'eau ressemble à une goutte d'eau. À nos yeux, la principale caractéristique de l'eau est sa capacité de réfracter la lumière. La réfraction occasionne deux phénomènes bien visibles. Le premier est le changement de brillance de la surface sous la goutte. Les surfaces imperméables apparaissent plus claires ou plus foncées sous la goutte, selon la position de la lumière. Le second phénomène, encore plus frappant que le premier, est la présence d'une caustique. Le plus souvent, cette caustique se produit sur la surface directement sous la goutte, mais elle peut aussi se produire sur la surface à l'extérieur de la goutte.

Pour avoir des caustiques, il est nécessaire de faire le rendu des gouttes avec un algorithme de tracé de rayon. Or, le logiciel *RenderMan* n'utilise pas un tel algorithme. Étant d'avis qu'une représentation approximative mais cohérente des caustiques est préférable à une absence de caustique, nous avons cherché un moyen de contourner cette limitation. Pour cela, nous avons pensé à utiliser un ensemble de textures. Cet ensemble contiendrait la caustique produite par une lumière frappant une goutte hémisphérique à différents angles d'incidence. Ainsi, lorsque nous devons produire la caustique d'une goutte pour un angle d'incidence donné, nous irions chercher la texture correspondant à l'angle d'incidence de la lumière. Puisque les gouttes ne sont jamais parfaitement hémisphériques, nous déformerions la texture afin qu'elle corresponde le mieux possible à la base de la goutte. Cette approche est actuellement à l'essai. Si les résultats s'avèrent concluants, nous pourrions l'intégrer dans notre logiciel.

Chapitre 6

Conclusion

Le rap américain, ou la techno européenne, [...] indiquent une voie claire : plus de rapidité, moins de mythologie de la création, davantage d'amateurisme musical (souvent même revendiqué), mais une qualité de son et de production « professionnelle ».

Alain Le Diberder, *L'État du monde 1995, p. 33*

Nous avons présenté une méthode de modélisation d'un phénomène naturel, l'écoulement de gouttes de liquide sur des surfaces. Notre modèle permet de simuler la trajectoire et la forme de gouttes de liquide. Notre approche préconise une séparation du phénomène en deux modèles : un modèle d'écoulement et un modèle de forme. Cette séparation permet de raffiner chacun des modèles au niveau de détail approprié.

Le modèle simulant l'écoulement des gouttes utilise une approche analytique pour le calcul des trajectoires. Grâce à cette approche, la simulation de l'écoulement est très rapide, même lorsque le nombre de gouttes est très grand et que la surface de support est finement divisée. De plus, le modèle d'écoulement simule plusieurs phénomènes connexes, tels que les traînées laissées par les gouttes, l'adhérence des gouttes à la surface et la fusion des gouttes lors de collisions. Pour ce dernier phénomène, nous avons développé un algorithme rapide de détection de collisions, qui utilise les caractéristiques de la surface de support pour réduire le nombre de calculs.

Le modèle simulant la forme des gouttes utilise un système de masses-ressorts pour le calcul de la surface enveloppant la goutte. Ce système est régi par un ensemble de contraintes que nous avons dérivé d'observations sur le comportement des gouttes. Ce modèle maximise la surface de contact de la goutte avec la surface de support, minimise l'interface air–goutte, conserve le volume de la goutte et la déforme sous l'effet de champs de force. Nous avons de plus étendu ce modèle pour déformer légèrement les gouttes, de façon aléatoire, afin de reproduire les déformations que subissent les gouttes qui s'écoulent lentement sur des surfaces rugueuses.

La majorité du temps, l'écoulement et la forme des gouttes ne demande pas un effort créatif de l'animateur. C'est une action secondaire de la scène. Notre modèle pourrait être utilisé pour simuler ce phénomène et libérer les animateurs de cette tâche d'animation.

Nous avons obtenu avec ce modèle des animations qui correspondaient à nos attentes du côté réalisme. Les gouttes suivent la surface, et leur trajectoire est modifiée par les caractéristiques (forme, rugosité, etc.) de celle-ci. Les traînées laissées par les gouttes ajoutent au réalisme et aident le spectateur à mieux visualiser la trajectoire empruntée par la goutte.

Nous sommes très satisfaits des temps de calcul de la partie trajectoire de notre modèle. Cependant, les temps de calcul du modèle de forme auraient avantage à être améliorés. Nous avons suggéré plusieurs modifications pour réduire ce temps, mais au prix d'une perte de qualité de la forme.

Notre modèle est assez général ; il pourrait servir de base à la modélisation d'autres phénomènes, tels que la simulation d'accumulations et de dépôts ou la modélisation des textures créées par des liquides (peintures, etc.) qui coulent en séchant. Avec l'ajout de fonctions de rendu sophistiquées, notamment pour faire le rendu de liquides transparents, notre système pourrait être utilisé dans la simulation de la sueur sur la peau, de larmes ou de gouttes de pluies sur une vitre.

Notre modèle se prête bien à la simulation de gouttes indépendantes. Il serait intéressant de l'intégrer dans un système de simulation multi-échelle des liquides, où plusieurs algorithmes différents pourraient simuler diverses « formes » d'un liquide. Ainsi, nous pourrions modéliser des scènes complexes comme une plage pluvieuse, où nous re-

trouverions une grande masse d'eau, l'océan, dans lequel se formeraient des vagues qui viendraient se jeter sur la plage ; sur cette plage se trouveraient plusieurs petites flaques, alimentées à la fois par l'océan et par la pluie qui tombe. Le tout pourrait être complété par le ruissellement de l'eau sur les roches, la chute de milliers de gouttes de pluies et un brouillard d'embruns.

Bibliographie

[Ach90] D. J. Acheson. *Elementary Fluid Dynamics*. Clarendon Press, Oxford, 1990.

[Arv86] James Arvo. « Backward Ray Tracing ». In *SIGGRAPH '86 Developments in Ray Tracing seminar notes*, volume 12, août 1986.

[BK89] D. E. Breen et V. Kühn. « Message-Based Object-Oriented Interaction Modeling ». In *Proceedings of Eurographics '89*, pages 489–503. Elsevier Science Publishers, septembre 1989.

[Bli78] James F. Blinn. « Simulation of Wrinkled Surfaces ». In *Computer Graphics (SIGGRAPH '78 Proceedings)*, volume 12, pages 286–292, août 1978.

[Bli82] James F. Blinn. « A Generalization of Algebraic Surface Drawing ». *ACM Transactions on Graphics*, volume 1, numéro 3, pages 235–256, juillet 1982.

[Blo97] Jules Bloomenthal, éditeur. *Introduction to Implicit Surfaces*. Computer Graphics and Geometric Modeling. Morgan Kaufmann Publisher, San Francisco, 1997.

[BN76] J.F. Blinn et M.E. Newell. « Texture and Reflection in Computer Generated Images ». *Communications of the ACM*, volume 19, numéro 10, pages 542–547, octobre 1976.

[CAS+97] Cassidy J. Curtis, Sean E. Anderson, Joshua E. Seims, Kurt W. Fleisher et David H. Salesin. « Computer-Generated Watercolor ». In *Computer Graphics (SIGGRAPH '97 Proceedings)*, volume 31, pages 421–430, août 1997.

[Cat78] Edwin E. Catmull. « Line Thinning ». TM 4, NYIT Computer Graphics Lab, mars 1978.

[Cyc94] Joseph M. Cychosz. « Efficient Binary Image Thinning Using Neighborhood Maps ». In Paul Heckbert, éditeur. *Graphics Gems IV*, pages 465–473. Academic Press, Boston, 1994.

[DPH96] Julie Dorsey, Hans Køhling Pedersen et Pat Hanrahan. « Flow and Changes in Appearance ». In *Computer Graphics (SIGGRAPH '96 Proceedings)*, volume 30, pages 411–420, août 1996.

[DS84] Mark A. Z. Dippé et John Swensen. « An Adaptive Subdivision Algorithm and Parallel Architecture for Realistic Image Synthesis ». In Hank Christiansen, éditeur. *Computer Graphics (SIGGRAPH '84 Proceedings)*, volume 18, pages 149–158, juillet 1984.

[FM96] Nick Foster et Dimitri Metaxas. « Realistic Animation of Liquids ». In *Proceedings of Graphics Interface '96*, pages 204–212, mai 1996.

[FR86] Alain Fournier et William T. Reeves. « A Simple Model of Ocean Waves ». In David C. Evans et Russell J. Athay, éditeurs. *Computer Graphics (SIGGRAPH '86 Proceedings)*, volume 20, pages 75–84, août 1986.

[FvDFH90] James D. Foley, Andries van Dam, Steven K. Feiner et John F. Hughes. *Computer Graphics : Principles and Practice.* Addison-Wesley Publishing Company, Reading, Massachusetts, deuxième édition, 1990.

[Gla84] Andrew S. Glassner. « Space Subdivision For Fast Ray Tracing ». *IEEE Computer Graphics and Applications*, volume 4, numéro 10, pages 15–22, octobre 1984.

[Gla88] Andrew S. Glassner. « Spacetime Ray Tracing for Animation ». *IEEE Computer Graphics and Applications*, volume 8, numéro 2, pages 60–70, mars 1988.

[Gla95] Andrew S. Glassner. *Principles of Digital Image Synthesis.* Morgan Kaufmann Publishers, Inc., San Francisco, California, 1995.

[Hab97a] Arash Habibi. *Modèles physiques supports de la relation mouvement-forme-image.* Thèse de doctorat, ACROE/CLIPS — Institut National Polytechnique de Grenoble, janvier 1997.

[Hab97b] Arash Habibi. « Volume Preserving Shapes ». Rapport technique, Laboratoire d'infographie, D.I.R.O., Université de Montréal, décembre 1997.

[Heg91] G. Hegron. « Rolling on a Smooth Bi-parametric Surface ». In *Eurographics Workshop on Animation and Simulation*, pages 205–213, 1991.

[HP88] David R. Haumann et Richard E. Parent. « The Behavioral Test-bed : Obtaining Complex Behavior from Simple Rules ». *The Visual Computer*, volume 4, numéro 6, pages 332–347, décembre 1988.

[KH84] James T. Kajiya et B. Von Herzen. « Ray Tracing Volume Densities ». In Hans Christiansen, éditeur. *Computer Graphics (SIGGRAPH '84 Proceedings)*, volume 18, pages 165–173, juillet 1984.

[Kin65] Blair Kinsman. *Wind Waves*. Prentice-Hall, 1965.

[KK86] Timothy L. Kay et James T. Kajiya. « Ray Tracing Complex Scenes ». In David C. Evans et Russell J. Athay, éditeurs. *Computer Graphics (SIGGRAPH '86 Proceedings)*, volume 20, pages 269–278, août 1986.

[KM90] Michael Kass et Gavin Miller. « Rapid, Stable Fluid Dynamics for Computer Graphics ». In Forest Baskett, éditeur. *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 49–57, août 1990.

[MP89] Gavin Miller et Andrew Pearce. « Globular Dynamics : A Connected Particle System for Animating Viscous Fluids ». *Computers and Graphics*, volume 13, numéro 3, pages 305–309, 1989.

[NA91] Nobu Nishikawa et Toshio Abe. « Artificial Nature in Splash of Droplets ». In *COMPUGRAPHICS '91*, volume I, pages 457–466, 1991.

[NHK+85] H. Nishimura, M. Hirai, T. Kawai, T. Kawata, I. Shirakawa et K. Omura. « Object Modeling by Distribution Function and a Method of Image Generation ». *Trans. IECE Japan, Part D*, volume J68-D, numéro 4, pages 718–725, 1985.

[Pea86] Darwyn R. Peachey. « Modeling Waves and Surf ». In David C. Evans et Russell J. Athay, éditeurs. *Computer Graphics (SIGGRAPH '86 Proceedings)*, volume 20, pages 65–74, août 1986.

[RB85] William T. Reeves et Ricki Blau. « Approximate and Probabilistic Algorithms for Shading and Rendering Structured Particle Systems ». In B. A. Barsky, éditeur. *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pages 313–322, juillet 1985.

[Ree83] W. T. Reeves. « Particle Systems – a Technique for Modeling a Class of Fuzzy Objects ». *ACM Trans. Graphics*, volume 2, pages 91–108, avril 1983.

[Rey87] Craig W. Reynolds. « Flocks, Herds, and Schools : A Distributed Behavioral Model ». In Maureen C. Stone, éditeur. *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 25–34, juillet 1987.

[RW80] S. M. Rubin et T. Whitted. « A 3-Dimensional Representation for Fast Rendering of Complex Scenes ». *Computer Graphics*, volume 14, numéro 3, pages 110–116, juillet 1980.

[SB87] John M. Snyder et Alan H. Barr. « Ray Tracing Complex Models Containing Surface Tessellations ». In Maureen C. Stone, éditeur. *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 119–128, juillet 1987.

[SDPO81] P. Suetens, P. Dierckx, R. Piessens et A. Oosterlinck. « A Semiautomatic Digitization Method and the Use of Spline Functions in Processing Line Drawings ». *Computer Graphics and Image Processing*, volume 15, pages 390–400, avril 1981.

[Sim90] Karl Sims. « Particle Animation and Rendering Using Data Parallel Computation ». In Forest Baskett, éditeur. *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 405–413, août 1990.

[SST89] Mikio Shinya, Takafumi Saito et Tokiichiro Takahashi. « Rendering Techniques for Transparent Objects ». In *Proceedings of Graphics Interface '89*, pages 173–182, Toronto, Ontario, juin 1989. Canadian Information Processing Society.

[STN87] Mikio Shinya, Tokiichiro Takahashi et Seiichiro Naito. « Principles and Applications of Pencil Tracing ». In Maureen C. Stone, éditeur. *Computer*

Graphics (SIGGRAPH '87 Proceedings), volume 21, pages 45–54, juillet 1987.

[TB87] Pauline Y. Ts'o et Brian A. Barsky. « Modeling and Rendering Waves : Wave-Tracing Using Beta-Splines and Reflective and Refractive Texture Mapping ». *ACM Transactions on Graphics*, volume 6, numéro 3, pages 191–214, 1987.

[Tho86] S. W. Thomas. « Dispersive refraction in ray tracing ». *The Visual Computer*, volume 2, numéro 1, pages 3–8, janvier 1986.

[Tot85] Daniel L. Toth. « On Ray Tracing Parametric Surfaces ». In B. A. Barsky, éditeur. *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pages 171–179, juillet 1985.

[TYT+84] Atsushi Takagi, Shigeki Yokoi, Shinji Tsuruoka, Fumitaka Kimura et Yasuji Miyake. « A Ray Tracing Model Considering the Color Dispersion of Light ». *Proc. Graphics and CAD Symposium*, pages 81–87, 1984.

[UK88] Craig Upson et Michael Keeler. « VBUFFER : Visible Volume Rendering ». In John Dill, éditeur. *Computer Graphics (SIGGRAPH '88 Proceedings)*, volume 22, pages 59–64, août 1988.

[Ups89] Steve Upstill. *The RenderMan Companion : A Programmer's Guide to Realistic Computer Graphics*. Addison-Wesley Publishing Company, Reading, MA, 1989.

[Weg84] Hank Weghorst. « An Image Synthesis System with Emphasis on Ray Tracing Techniques ». Mémoire de maîtrise, Program of Computer Graphics, Cornell University, 1984.

[WH91] Jakub Wejchert et David Haumann. « Animation Aerodynamics ». In Thomas W. Sederberg, éditeur. *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 19–22, juillet 1991.

[Whi80] Turner Whitted. « An Improved Illumination Model for Shaded Display ». *Communications of the ACM*, volume 23, numéro 6, pages 343–349, juin 1980.

[WMW86a] Brian Wyvill, Craig McPheeters et Geoff Wyvill. « Animating Soft Objects ». *The Visual Computer*, volume 2, numéro 4, pages 235–242, 1986.

[WMW86b] Brian Wyvill, Craig McPheeters et Geoff Wyvill. « Data Structure for Soft Objects ». *The Visual Computer*, volume 2, numéro 4, pages 227–234, 1986.

[WW92] Alan Watt et Mark Watt. *Advanced Animation and Rendering Techniques : Theory and Practice*. Addison-Wesley Publishing Company, New York, New York, 1992.

[YKIS88] Ying Yuan, Tosiyasu L. Kunii, Naota Inamoto et Lining Sun. « Gemstone-Fire : adaptive dispersive ray tracing of polygons ». *The Visual Computer*, volume 4, numéro 5, pages 259–270, novembre 1988.

[YUM86] Larry Yaeger, Craig Upson et Robert Myers. « Combining Physical and Visual Simulation — Creation of the Planet Jupiter for the Film "2010" ». In David C. Evans et Russell J. Athay, éditeurs. *Computer Graphics (SIGGRAPH '86 Proceedings)*, volume 20, pages 85–93, août 1986.