Rendu réaliste de nuages en temps réel

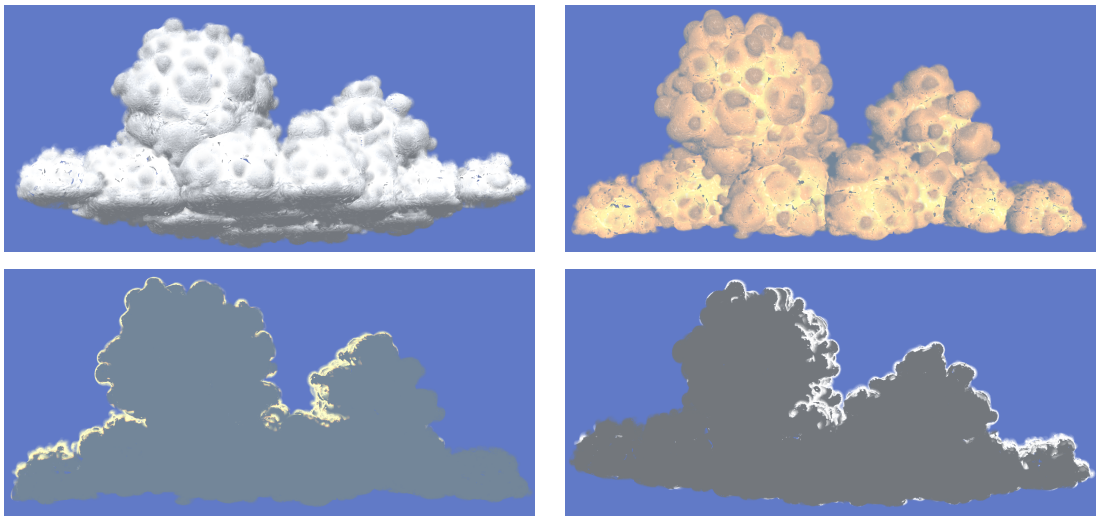
Antoine Bouthors

Rendu réaliste de nuages en temps réel

Antoine Bouthors

Mémoire de DEA et de fin d'études

9 septembre 2004



IMAG-INRIA
Laboratoire GRAVIR
Projet EVASION

ESIEE Paris
Majeure Informatique

Université de Marne-la-Vallée
DEA IFA
Filière Synthèse d'Images

Table des matières

Table des figures

Introduction

Avant-propos

J'ai effectué mon stage de DEA et de fin d'études d'ingénieur au sein de l'équipe EVASION (Environnements Virtuels pour l'Animation et la Synthèse d'Images d'Objets Naturels) du laboratoire GRAVIR, au sein de la fédération d'unités de recherche IMAG. Ce stage était encadré par Fabrice Neyret, chargé de recherche dans l'équipe EVASION, et s'est déroulé dans les locaux de l'INRIA Rhône-Alpes, du 1er mars au 9 juillet. J'ai également effectué pendant tout le mois d'août un CDD au sein de l'INRIA qui m'a permis de poursuivre mon stage de façon à me rapprocher des objectifs fixés, qui n'avaient pas été atteint à la fin du stage. D'autre part, je débute maintenant une thèse au sein de la même équipe et sous la direction de Fabrice Neyret, sur un sujet plus vaste dans lequel le stage s'inscrit, à savoir la simulation visuellement réaliste de nuages convectifs en temps-réel.

Introduction au domaine

La synthèse d'images est divisé en deux grands contextes applicatifs : la synthèse d'images *réaliste* et la synthèse d'images *temps-réel*. Le premier consiste à créer une image la plus fidèle possible à la réalité, peu importe (ou presque) le temps mis à la générer. Il trouve ses applications dans les domaines non interactifs, comme par exemple les effets spéciaux du cinéma. Le second consiste à essayer de dessiner des images dans un temps extrêmement court (de façon à pouvoir calculer ces images en quelques centièmes de seconde), afin d'obtenir un résultat permettant l'interactivité, tel que dans les jeux vidéos et les simulateurs.

L'équipe EVASION est spécialisée dans la synthèse d'images temps-réel d'objets naturels. Les scènes naturelles ont ceci de particulier qu'elles sont très vastes possèdent une très grande complexité visuelle, ce qui fait de leur synthèse un défi pour ceux qui s'y attellent, et pire encore dans le cas temps-réel. Une approche permettant de réduire cette complexité est d'utiliser des méthodes *phénoménologiques*, qui consistent à essayer de reproduire une série de caractéristiques macroscopiques du phénomène étudié plutôt que d'en simuler les causes microphysiques.

Introduction au sujet

Le sujet de mon stage concerne les nuages, élément important des scènes naturelles, et plus précisément les *cumulus*. La simulation de ces objets peut être divisée selon trois axes distinctes : la modélisation de la forme, la génération de l'aspect visuel (appelée *rendu*), et l'animation des nuages. Lors de ce stage, nous traiterons de la génération de la forme et du rendu.

L'approche phénoménologique a été choisie pour simuler ces objets. Nous avons donc commencé par étudier les propriétés visuelles caractéristiques des cumulus, puis nous avons tenté de les reproduire de la façon la plus efficace possible de manière a obtenir un rendu temps-réel ayant le plus de réalisme possible.

Nous présentons au chapitre 1 les caractéristiques visuelles des cumulus. Nous dressons ensuite au chapitre 2 un aperçu des différents travaux existants et qui nous ont inspirés. Le chapitre 3 présente brièvement l'idée générale qui nous guide dans notre travail. Nous exposons au chapitre 4 la méthode que nous avons choisie pour générer la forme du nuage, puis au chapitre 5 l'approche que nous avons utilisée pour effectuer le rendu. Nous envisagerons enfin au chapitre 6 de montrer les possibilités futures qu'ouvrent les résultats de ces travaux.

Le travail réalisé pendant la premières partie de ce stage a donné lieu à la publication d'un article (*short paper*, article court) à une conférence de synthèse d'images, *Eurographics*[1], qui sera présenté le 2 septembre à Grenoble lors de la tenue de cette conférence. L'annexe C contient une copie de cet article.

[1] ***Eurographics*** : association européenne d'informatique graphique, de renommée internationale, organisant chaque année la conférence du même nom. http://www.eg.org/

Chapitre 1

Caractéristiques des cumulus

Fig. 1.1: Un vrai cumulus

1.1 Forme

Les cumulus ont des caractéristiques très spécifiques, découlant d'un air instable. Ainsi, l'air sec et l'air humide ne sont quasiment pas mélangés, ce qui donne au nuage mûr un aspect compact et presque solide, avec une surface relativement bien définie. La forme de cette surface est formée par des turbulences et des courants ascendants, qui induisent ce motif caractéristique de 'chou-fleur'.

Un équilibre entre le cisaillement (induit par la croissance des volutes) et la diffusion (due aux courants) maintient une couche 'floue' d'épaisseur relativement constante entre l'air sec environnant et l'air humide au cœur du nuage [GC91, GC93]. Les principaux volutes (ou bulles) ont approximativement cent mètres de large. Un cumulus très jeune peut être constitué d'une simple bulle (un courant ascendant ayant monté jusqu'à l'altitude de condensation). Il n'a alors probablement pas les propriétés de netteté décrites plus haut. Nous considérons ici plus les cumulus mûrs ou les cumulonimbus, qui peuvent avoir des tailles de plusieurs kilomètres.

On peut également observer que les nuages possèdent une structure multi-échelle : les bulles se formant dans l'air ambiant grossissent, et des turbulences apparaissent alors sur les bords. Ces turbulences sont dues aux instabilités créées par le cisaillement, la poussée d'Archimède et les gradients de températures. Elles se transformant en nouvelles bulles lorsque les conditions atmosphériques nécessaires sont réunies. Ces nouvelles bulles ont notamment une taille caractéristique, déterminée par la lacunarité (rapport entre la taille d'une bulle et la taille des bulles se format à sa surface) du nuage. De même que leur bulle 'mère', ces nouvelles bulles vont ensuite grossir et s'étendre, et de nouveau à leur surface apparaîtront de nouvelles bulles, et ainsi de suite.

Enfin, la condensation de la vapeur d'eau en gouttelettes étant restreintes à certaines conditions de température et de pression bien précises, les nuages ne peuvent se former qu'à partir d'un certain seuil d'altitude (appelé point de rosée), l'eau présente en dessous de cette hauteur étant obligatoirement sous forme de vapeur. Il en résulte que la partie basse des cumulus apparaît comme 'tronquée' par une limite virtuelle qui est cette altitude de condensation. Un fond plat est donc caractéristique de la forme de ces nuages, lorsqu'ils sont suffisamment bas.

1.2 Aspect

Une caractéristique commune à tous les types de nuages est que l'albédo[1] est très proche de 1 (il n'y a quasiment aucune absorption), et que les gouttelettes d'eau induisent une dispersion principalement vers l'avant ('*forward scattering*'[2]), ce qui signifie que la lumière est très peu déviée par le nuage à l'échelle microscopique. Dans le cas particulier des cumulus, qui sont extrêmement denses et épais, ceci engendre une autre caractéristique unique et essentielle : la dispersion est quasiment isotrope (*i.e.,* la lumière est renvoyée dans toutes les directions) à une échelle macroscopique (l'explication est fournie dans l'annexe A), hormis dans la fine couche externe qui sépare le nuage de l'air ambiant, visible sur la silhouette.

Une conséquence est que la surface du nuage ré-émet quasiment la même quantité de lumière qu'elle a reçue au voisinage de chaque point, de façon isotrope, presque comme si elle reflétait simplement la lumière arrivant. On peut donc appliquer, en première approximation, le modèle de réflexion de Lambert[3].

Ceci induit une autre conséquence : la radiosité[4] joue un rôle important dans l'aspect d'un cumulus. En effet, la lumière ré-émise par un élément de surface a presque la même énergie que celle reçue (dû à l'albédo important). Si cette lumière ré-émise est reçue par un autre élément de surface, elle augmente de façon significative la quantité d'énergie que ce second élément de surface reçoit et ré-émet. De cette façon, les inter-réflexions ne sont quasiment pas absorbées, et les concavités sur la surface du nuage (*e.g.,* entre les bulles) agissent comme des pièges à lumière : toute l'énergie reçue est ré-émise dans un angle solide restreint (plutôt que 2π pour une surface plane), ce qui rend certaines concavités presque aussi lumineuses que le soleil. Ceci donne parfois l'impression que le nuage est illuminée depuis l'intérieur, puisque l'intérieur des sillons semble être bien plus lumineux que la surface autour.

Ce comportement peut aussi survenir, moins intensément cependant, dans les endroits qui ne sont pas éclairés directement par le soleil. D'autre part, ce phénomène se produit quelle que soit l'échelle des concavités, de la plus locale à la plus macroscopique.

Enfin, la corolle d'un cumulus a un aspect relativement différent de celui du cœur du nuage. En effet, étant donné la faible densité de particules d'eau au niveau de la surface du nuage, la dispersion cumulée reste principalement dans la même direction et vers l'avant, et non isotrope (notons que les nuages diffus tels que les stratus ou les jeunes cumulus possèdent cette propriété en tout point). Ceci rend les cumulus très contrastés : lorsqu'il sont illuminés par devant, le corps est intensément blanc et la corolle est relativement transparente, et de la couleur du ciel. Lorsqu'ils sont illuminés par derrière, le cœur du nuage est extrêmement sombre et homogène, alors que la corolle est lumineuse, grâce au 'forward scattering' (*cf.* figure 1.2).

Le peu de lumière ré-émise par le nuage dans les zones d'ombres ne provient en fait pas de la lumière traversant le nuage, mais principalement des réflexions du ciel et du sol, et encore plus probablement des réflexions de la lumière par les autres parties du nuage.

[1] **Albédo** : quantité de lumière renvoyée par la matière

[2] ***Forward scattering*** : terme anglo-saxon désignant le fait que la matière ne dévie presque pas la lumière, et la disperse donc 'vers l'avant' plutôt que dans toutes les directions

[3] **Modèle de réflexion de Lambert** : modèle de réflexion donnant à une surface un aspect mat, comme du plâtre. Le principe de ce modèle est que plus la surface fait face à une source de lumière, plus elle est illuminée. Ce modèle est très couramment utilisé en synthèse d'image pour les objets devant avoir un aspect solide.

[4] **Radiosité** : modèle de réflexion dans lequel on tient compte du fait que les surfaces recevant de la lumière en renvoient une partie dans toutes les directions.

FIG. 1.2: Un vrai cumulus illuminé par derrière. Notez la différence entre le corps du nuage, sombre, et la corolle, transmettant la lumière du soleil.

1.3 Evolution

Bien que l'évolution et l'animation des cumulus n'entre pas dans le cadre du sujet de ce stage, il est intéressant d'en connaître les propriétés afin de choisir des modèles de simulation ayant la possibilité d'être étendu à l'animation par la suite.

Les cumulus étant le produit de turbulences et de courants, leur mouvement est naturellement convectif. Les bulles formées ont les mêmes propriétés que celles d'un champignon atomique, exemple parfait d'objet convectif : celles-ci grossissent et leur matière subit un mouvement circulaire autour d'un anneau de vorticité. Cet anneau de vorticité est perturbé par les turbulences et par la vorticité créée par les autres bulles, voire par sa propre vorticité. De plus, les bulles nouvellement créées l'étant sur la surface de bulles existantes, elles sont également soumises à la vorticité de leur parentes. On obtient donc un schéma d'animation multi-échelle basé sur des mouvements convectifs. Cette évolution peut être simulée à l'échelle macroscopique, en simulant les vortex et les bulles en cause. En particulier, nous souhaiterions relier le résultat de ce stage à des travaux tels que [Ney97].

Chapitre 2

Etat de l'art

Il existe deux principaux moyens de simuler les nuages en synthèse d'images. La première méthode, volumique, a été étudiée par de nombreuses équipes et l'est encore. La seconde, surfacique, est sensiblement moins poursuivie malgré son intérêt, et nous souhaitons a présent la remettre au goût du jour. Cet état de l'art est divisé en deux sections, Modélisation de la forme et Rendu. Cependant, certains travaux portent sur les deux aspects à la fois.

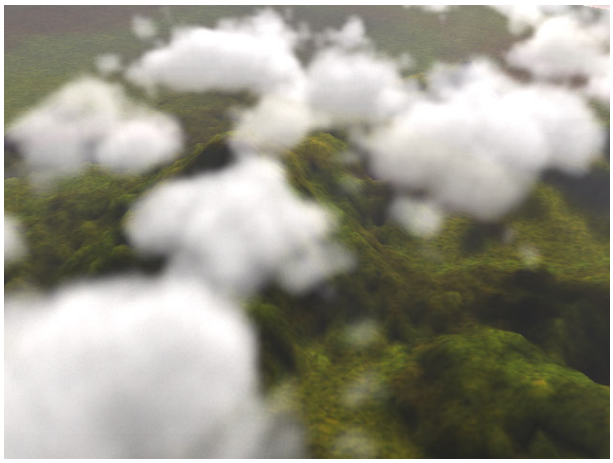


FIG. 2.1: Des nuages générés par Dobashi *et al.* . La faible résolution de la simulation empêche son utilisation pour des nuages détaillés du type cumulus – Rendu presque temps-réel

2.1 Modélisation de la forme

La forme naturelle des nuages étant un résultat de leur évolution, de nombreux travaux existant utilisent la simulation de la physique des nuages pour générer des formes réalistes. Cependant, cette simulation faisant intervenir de nombreuses et complexes équations de la physique des fluides, elle a été simplifiée pour satisfaire les contraintes de rapidité requises pour la synthèse d'images. Harris *et al.* [HISL03] (*cf.* figure 2.2) simulent de façon relativement précise l'évolution des nuages, comme Stam *et al.* [FSJ01], mais avec une résolution spatiale très frustre par rapport au pouvoir de discrimination de l'oeil humain. Dobashi *et al.* [DKY+00] (*cf.* figure 2.1) utilisent des automates cellulaires de façon à imiter la façon dont les nuages grossissent et se déplacent. Dans [MYDN01], Miyazaki *et al.* utilisent les *Coupled Map Lattice*, une technique similaire aux automates cellulaires mais permettant de simuler la physique des fluides, cependant toujours à une échelle relativement grossière.

D'autres travaux reposent sur une approche phénoménologique, de façon à obtenir une forme plausible à moindre coût. Ces approches utilisent parfois des éléments de simulation sous la forme de système de particules [SSEH03], tandis que d'autres préfèrent une approche purement phénoménologique : Gardner [Gar84, Gar85] (*cf.* figure 2.3) représente les nuages comme un amas d'ellipsoïdes. Perlin [Per85a] s'appuie sur un bruit fractal. D'autres auteurs [Ebe97] utilisent des fonctions de forme comme les surfaces implicites. En particulier, Nishita *et al.* [NND96] combinent des particules agencées de façon hiérarchiques

et des surfaces implicites : un niveau est construit sur le niveau précédent en discrétisant la surface implicite définie par la dernière couche de particules et en plaçant les particules du niveau suivant de façon égale sur cette surface.

Enfin, on peut citer les méthodes recréant des formes de nuages à partir de données 2D réelles, de type photographies ou images satellites [DNYO98], et celles demandant à un artiste de créer de toutes pièces des formes réalistes [Mic].

Les méthodes physiques donnent des résultats réalistes mais au prix d'un coût qui doit être compensé par une diminution de la résolution spatiale et/ou temporelle, ignorant ainsi les détails qui donnent aux nuages (et en particulier aux cumulus) leur richesse. A noter que les ces détails sont sans doute visuellement plus importants que les valeurs précises de l'intensité. Les méthodes phénoménologiques, quand à elles, permettent d'atteindre à un coup moindre des résultats plausibles et similaires aux méthodes physiques à faible résolution. Elles ne permettent donc pas non plus d'obtenir un résultat ayant la finesse suffisante.

FIG. 2.2: Des nuages générés par Harris *et al.* . Malgré le réalisme du rendu, le modèle n'est pas assez détaillé pour pouvoir représenter un cumulus – Rendu temps-réel

2.2 Rendu

Blinn [Bli82] et Max [Max86] ont proposé les premiers travaux pour représenter la réflexion et la diffusion de la lumière à travers les nuages. Kajiya [KH84] a introduit l'idée de rendu volumétrique. Max [Max94] a étendu le rendu pour prendre en compte la dispersion anisotrope de la lumière à l'intérieur du nuage, et ainsi obtenir une corolle illuminée lorsque la lumière se situe derrière le nuage. Cette méthode a été par la suite réutilisée, en utilisant des heuristiques basées sur le principe de '*forward scattering*' typique au nuages et en prenant en compte l'illumination du ciel, du sol et la radiosité [NND96], puis optimisée en utilisant les possibilités données par le matériel graphique [DKY+00, HISL03] (*cf.* figures 2.1 et 2.2).

Beaucoup de travail a été consacré à la radiosité volumique en général, en commençant par la méthode zonale par Rushmeier et Torrance [RT87]. Ces techniques et leur variantes ont parfois été utilisées pour rendre des nuages, comme dans [Ebe97] (*cf.* figure 2.4).

Dans les contexte temps-réel, ces volumes sont représentés par accumulation de tranches semi-transparentes (ou '*billboards*'[1]), mais cela reste relativement coûteux. En effet, si une image est débitée en cent tranches, chaque pixel de l'écran sera réécrit cent fois.

[1] ***Billboard*** : image représentant tout ou partie d'un objet et faisant toujours face à l'utilisateur

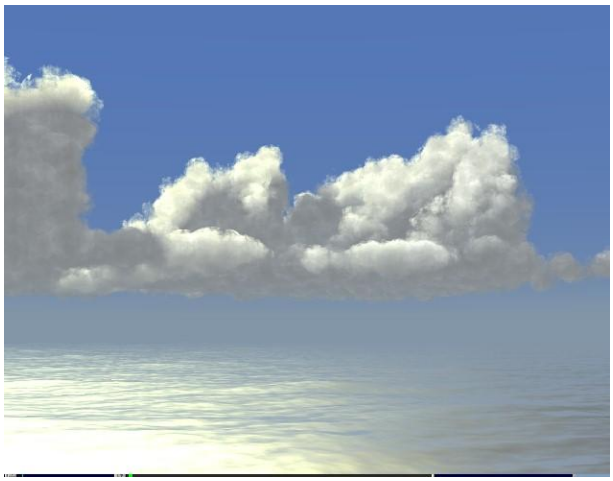


FIG. 2.3: Un cumulus généré par Gardner. On peut noter le détail apporté par l'utilisation des textures – Rendu non temps-réel

Un autre aspect est d'utiliser les textures, fractales et *shaders* pour représenter les détails des nuages. Musgrave [EMP+94] les a intensément utilisés pour afficher des ciels de nuages. Gardner [Gar84, Gar85] (*cf.* figure 2.3) a proposé un modèle particulièrement simple et donnant de bons résultats basés sur des ellipsoïdes texturées dont la transparence est amplifiée à proximité de la silhouette. Etant donné le matériel disponible à cette époque, cette méthode n'était réalisable que par *ray-tracing*[2], interdisant ainsi la possibilité d'un rendu temps-réel. Heureusement, le matériel graphique récent permet de définir des *pixel shaders*[3], permettant d'espérer adapter cette approche temps-réel. Une première tentative a été effectuée par Elinas [ES00]. Neyret [Ney00] a effectué une pré-étude pour améliorer la qualité du rendu en prenant en compte les observations phénoménologiques, mais en gardant un rendu par *ray-tracing*.

Les méthodes physiques ont été, pour le rendu, massivement utilisées et améliorées. Cependant, ces méthodes laissent de côté les détails, et les heuristiques utilisées ignorent la connaissance macroscopique disponible pour les cumulus, dont les détails sont importants et dont la dispersion est principalement isotrope. D'autre part, les méthodes utilisant les textures (idéales pour représenter les détails) ont été peu utilisées. Celles-ci manquent de réalisme, et leur application au rendu temps-réel n'a pas été profondément étudiée.

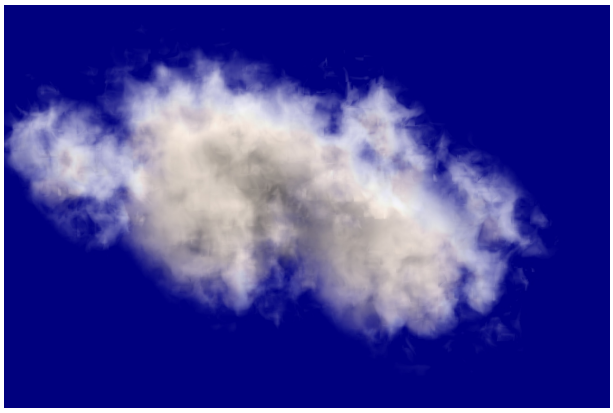


FIG. 2.4: Un nuage généré par Ebert. Notez l'utilisation des fractales pour ajouter du détail au nuage – Rendu non temps-réel

[2] ***Ray-tracing*** : technique de rendu non temps-réel permettant de donner un aspect unique à chaque pixel de l'image générée.

[3] ***Vertex Shaders* et *Pixel shaders*** : outil présent sur le matériel graphique récent permettant de donner un aspect unique à chaque pixel de l'image générée de façon temps-réel.

Chapitre 3

Notre approche

Notre idée est de représenter le nuage comme une hiérarchie de bulles quasi-sphériques ou particules. Cette idée pose deux problèmes à résoudre :

- Tout d'abord, nous avons besoin de fabriquer cette hiérarchie (qui pourrait un jour être animée dans l'esprit de [Ney97]), ainsi que la surface du nuage. Ce problème est résolu au chapitre 4, où nous présentons une méthode de création de la forme du nuage à partir d'une hiérarchie de particules placées sur des surfaces implicites.
- Deuxièmement, nous devons trouver un modèle d'illumination prenant en compte les caractéristiques d'aspect décrites au chapitre 1 pour reproduire les effets observés : aspect généralement réflectif, bords possédant une forte dispersion vers l'avant, sillons agissant comme des pièges à lumière, etc. Pour ceci, nous nous inspirons du modèle de rendu de Gardner [Gar84, Gar85], en appliquant sur nos formes quasi-sphériques un *shader* reproduisant ces effets, dont le rapport qualité/coût est bien plus important que pour le *ray-tracing* et qui devrait être réalisable en temps-réel. Cet modèle est décrit au chapitre 5.

Chapitre 4

Modélisation de la forme

4.1 Idée générale

Etant donné que le rendu implique une surface sur lequel l'appliquer, nous avons commencé par créer un outil permettant de créer une forme de nuages selon les critères observés en section 1.1. Notre modèle est basé sur des particules (ou "blobs") associées à un champ implicite qui permet de définir une surface, utilisée par la suite pour être rendue ou pour servir de surface de base à d'autres particules.

4.2 Notre représentation

Un nuage est composé d'un ensemble de niveaux hiérarchiques définis récursivement les uns sur les autres. Chaque niveau l est constitué d'un ensemble de particules p_i et définit une surface S_l. Chaque particule d'un niveau l est définie par des paramètres locaux et globaux : sa position P_i sur la surface S_{l-1}, son rayon r_i, son aplatissement e_i, et des paramètres de mélange définis globalement pour chaque niveau. Certains de ces paramètres sont ajustés automatiquement selon le niveau ou selon les particules avoisinantes, tandis que d'autres peuvent être réglés directement ou indirectement par l'utilisateur.

Chaque particule définit sa propre surface S_i, qui consiste en une forme "pure" (une sphère aplatie) qui sera par la suite altérée pour répondre aux critères régis par les paramètres cités plus haut. Ceci est réalisé en utilisant une formulation implicite inspirée par [Gas93]. Cette surface implicite est définie par :

$$S_i = \{\mathbf{P} \in \mathbf{R}^3 / f_i(\mathbf{P}) = 1\}$$

avec

$$f_i(\mathbf{P}) = g_i(\mathbf{P}) + h_i(\mathbf{P})$$

$g_i(\mathbf{P})$ est une fonction de champ basique représentant une sphère aplatie, ce qui donne la forme "pure" :

$$g_i(\mathbf{P}) = \exp\left(1 - \frac{\mathrm{d}_i}{r_i(1 - e_i\,\mathrm{d}_{l-1})}\right)$$

avec d_i et d_l la distance de $\mathbf{P}$ à $\mathbf{P}_i$ et à S_l, respectivement. L'aplatissement (définit aléatoirement) représente les différentes étapes de développement d'une bulle de nuage et permet de donner un aspect moins uniforme à la surface du nuage. $h_i(\mathbf{P})$ est une fonction de champ qui altère cette forme sphérique pour lui faire respecter nos contraintes. Pour la définir, nous nous inspirons de [Gas93] : leur but était de définir un contact exact entre deux formes implicites. Au contraire, ce que nous désirons ne sont pas des blobs se collant entre eux (les bulles des nuages réelles ne le font pas, *cf.* Figure 1.1), mais des *sillons* entre les blobs, qui doivent néanmoins se recoller de façon continue. Nous définissons $h_i(\mathbf{P})$ comme une combinaison de plusieurs fonctions de champ contrôlées par les paramètres des particules :

$$h_i(\mathbf{P}) = m_i(\mathbf{P}) + n_i(\mathbf{P}) + o_i(\mathbf{P})$$

• $m_i(\mathbf{P})$ est une fonction de champ qui empêche les blobs de s'interpénétrer, et est donc dans son esprit très proche de celle de [Gas93]. Cependant, nous avons ajouté un paramètre d'offset ϵ pour repousser les surfaces plus loin que pour le contact de façon à créer des sillons entre les blobs. Nous avons également

utilisé une équation différente pour la génération des bombements de façon à produire une transition plus lisse entre la zone de répulsion et la forme pure éloignée de la zone d'interaction. Cela donne :

$$m_i(\mathbf{P}) = \sum_j m_i^j(\mathbf{P})$$

$$\text{avec} \quad m_i^j(\mathbf{P}) = (1 - \epsilon - g_j(\mathbf{P})) \min\left(1, g_j^2(\mathbf{P})\right)$$

• $n_i(\mathbf{P})$ est une fonction de champ qui élargit la surface du blob au fur et à mesure qu'elle est plus proche de la surface de base (*i.e.,* la surface du niveau précédent S_{l-1}), de telle sorte que la particule n'apparaît pas comme "surgissant" hors du nuage. $n_i(\mathbf{P})$ est contrôlée par deux paramètres : b, qui définit la proportion d'élargissement, et I, qui ajuste la fraction de la bulle qui sera élargie pour créer le mélange. Nous avons choisi :

$$n_i(\mathbf{P}) = b \ \min\left(1, e^{-\frac{I\, \mathrm{d}_{l-1}}{r_i}}\right) e^{1 - \frac{\mathrm{d}_i}{r_i}}$$

I contrôle la portion de la bulle qui sera élargie pour créer le mélange entre le blob et sa surface de base.

• $o_i(\mathbf{P})$ est une fonction de champ donnant un fond plat au nuage, en contraignant la surface des blobs au-dessus d'une hauteur absolue donnée h_0 avec une raideur (*i.e.,* une force de répulsion) α_h :

$$o_i(\mathbf{P}) = g_i(\mathbf{P}) \min\left(0, \frac{\mathrm{height}(\mathbf{P}) - h_0}{\alpha_h}\right)$$

Au final, S_l est définie par le potentiel :

$$f_l(P) = \max\left(f_{l-1}(\mathbf{P}), \max_i f_i(\mathbf{P})\right)$$

4.3 Création de la surface du nuage

Nous avons besoin de la surface S_{l-1} pour placer les particules du niveau suivant, et pour effectuer le rendu du dernier niveau de a hiérarchie. Nous pouvons obtenir les surface discrétisées de tous les blobs appartenant à un niveau de hiérarchie $l-1$ en utilisant la méthode décrite dans [Gas93]. Pour placer les particules p_i du niveau l, nous avons préféré extraire une isosurface de $l-1$ en utilisant un système de particules comme décrit dans [WH94] et [CA97], plutôt que de discrétiser explicitement la surface S_{l-1} à la manière de [NND96]. Ce système de particules est ensuite utilisé directement comme l'ensemble des particules du niveau l, en utilisant le rayon de répulsion des particules comme le rayon des blobs.

Cette approche simule des particules qui se repoussent les unes les autres et grossissent pour peupler toute la surface disponible. Lorsqu'une particule a trop grossi, ou lorsqu'une zone de la surface n'est pas assez peuplée, les particules concernées se divisent pour créer de nouvelles particules. En revanche, si une zone de la surface est surpeuplée, alors certaines particules sont supprimées. L'utilisateur fourni à cette méthode une distance inter-particules désirée (*i.e.,* le double du rayon des particules désiré), que toutes les particules auront après avoir atteint l'état d'équilibre.

Nous voulons reproduire une distribution naturelle, cette distribution régulière n'est donc pas adéquate. Nous prenons donc un rayon de répulsion différent pour chaque particule, de façon à ce que toutes les particules d'un niveau n'aient pas le même aspect et que la surface soit inégale. Pour prendre en compte ce nouveau rayon d'influence indépendant, nous avons modifié l'algorithme de [WH94] de la façon suivante : si l'énergie d'une particule est trop basse, le rayon de répulsion de la particule évolue de façon à atteindre l'énergie désirée, comme dans [WH94]. En revanche, il évolue de façon à atteindre le rayon désiré. De plus, l'énergie est aussi différente pour chaque particule : elle est modulée de la même façon que le rayon désiré.

Une fois que l'ensemble des particules du niveau l est créé sur la surface implicite S_{l-1}, les surfaces des blobs sont définies comme décrit dans la section 4.2. Ainsi, nous pouvons générer la surface implicite S_l à partir de ces blobs, que nous utiliseront pour placer les particules du niveau $l+1$, et ainsi de suite. Le dernier niveau définit la forme du nuage.

Les particules du niveau 0 (*i.e.,* la racine) n'ont pas de surface implicite sur laquelle s'appuyer et doivent donc être créées d'une autre manière. Suivant le but de l'application, ceci peut se faire de manière procédurale (*e.g.,* pour créer un ciel de nuages), ou contrôlé par l'utilisateur. Dans le premier cas, nous créons un petit nombre de particules placées aléatoirement sur un plan horizontal ou dans une partie restreinte de l'espace. Dans le second cas, une interface laisse l'utilisateur former et placer les particules racines à la main, lui permettant de créer des formes réalistes telles que des tourelles de nuage.

4.4 Résultats

Les images présentées sur les figures 4.1 à 4.5 montrent un nuage avec un niveau racine constitué de six particules placées manuellement. C'est un très faible travail de spécification pour l'utilisateur, mais permet de contrôler la forme générale du nuage. La figure 4.1 montre ce niveau racine. Nous avons généré sur celui-ci deux niveaux supplémentaires, résultant en un cumulus montré en figure 4.2 (le maillage est exhaustif, nous travaillons actuellement à le rendre adaptatif). L'effet d'un shader du type Gardner est montré sur les figures 4.6 et 4.5. Notez que ce rendu est de basse qualité et peut réaliste, car ceci constitue le rôle de la suite du stage et correspond au chapitre 5.

Les paramètres de l'application ont été choisis comme suit. Le rayon des particules racines (*i.e.,* les particules du niveau 0) manuellement positionnées est compris entre 200 et 450 mètres. Le rayon des particules pour le niveau l est un tiers de celui des particules du niveau $l-1$ (imitant ainsi la lacunarité estimée sur la figure 1.1), modulé par un facteur aléatoire de $\pm$ 30%. Les particules ont été aplaties avec un facteur e_i aléatoirement choisi entre 0.1 et 1. Le paramètre contrôlant la taille des sillons ϵ est fixé à 0.5, ce qui signifie que chaque blob est séparé des autres d'au moins la moitié de son rayon.

Le coefficient d'élargissement b a été mis à zéro pour les niveaux 0 et 1, et 1.0 pour le niveau 2. Le paramètre I est fixé à 10.

La hauteur de référence h_0 correspondant au fond du nuage est de 100 mètres, ce qui est approximativement la hauteur de la particule la plus basse du niveau 0. La rigidité de la répulsion α_h est également de 100 mètres.

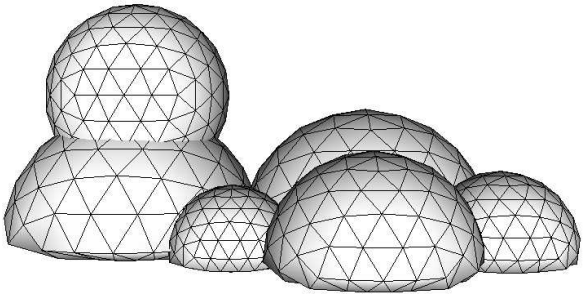


FIG. 4.1: Six blobs définis par l'utilisateur.

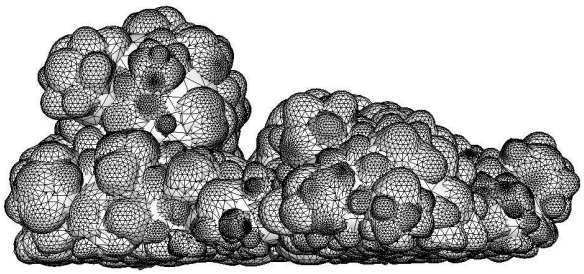


FIG. 4.2: Deux niveaux de hiérarchie générés sur le niveau racine de la figure 4.1 forment un nuage.

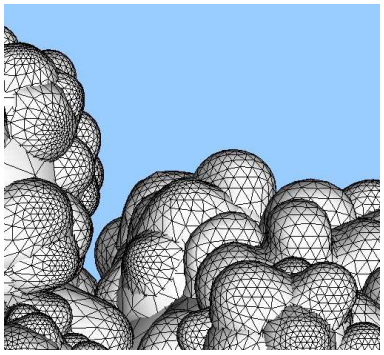


FIG. 4.3: Zoom sur le maillage des blobs générés : notez la déformation des sphères pour créer des sillons tout en assurant une continuité.

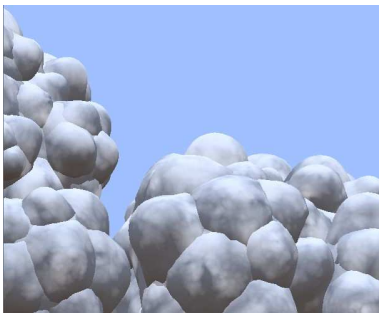


FIG. 4.4: Zoom sur le maillage des blobs générés avec une texture de Perlin. Le maillage a été légèrement extrudé au moment du rendu, ce qui explique les discontinuités.

FIG. 4.5: Zoom sur le maillage des blobs générés avec un *shader* dans le style de Gardner.

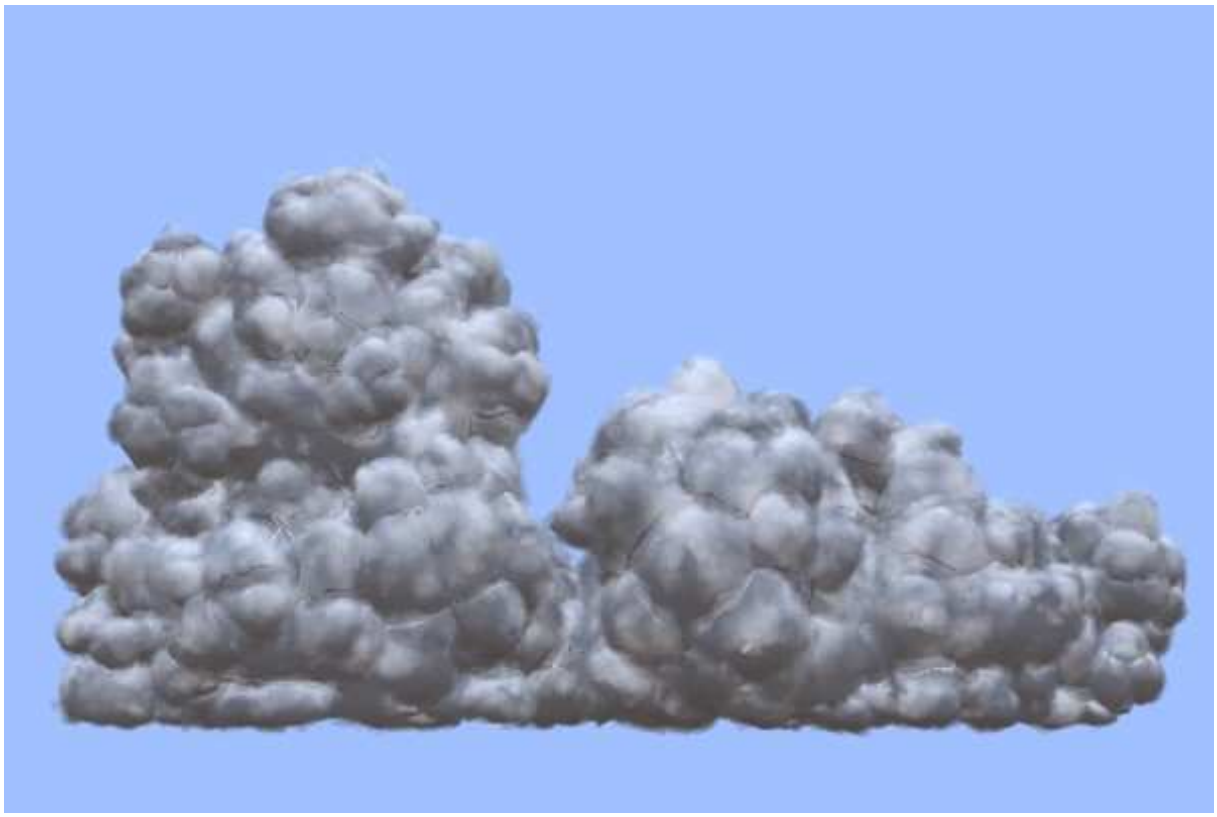


FIG. 4.6: Le nuage affiché avec un *shader* dans le style de Gardner.

Le résultat est conforme à nos attentes, et nous fourni une surface réaliste afin d'y appliquer un rendu réaliste décrit dans le chapitre 5. La simulation de niveaux hiérarchiques supplémentaires sera notamment effectuée lors de ce rendu (*cf.* section 5.4), plutôt que dans le modèle de forme, évitant ainsi une surcharge de la géométrie.

Le résultat de ce travail (qui a occupé environ les deux premiers mois de mon stage) a donné lieu à la publications d'un papier court pour la conférence *Eurographics*. Cet article est reproduit à l'annexe C.

Chapitre 5

Rendu

5.1 Idée générale

Nous utilisons les observations rapportées en section 1.2 pour créer une fonction de rendu inspirée de celle de Gardner [Gar84, Gar85] que nous étendrons en s'appuyant sur la pré-étude effectuée par Neyret [Ney00] pour donner un aspect réaliste et crédible au nuage. De plus, nous implémentons la fonction de rendu en utilisant les fonctionnalités du matériel graphique récent (*vertex* et *pixel shaders*) de façon à obtenir un rendu temps réel.

En plus de reprendre les éléments du chapitre 4, nous utiliserons les notations mathématiques suivantes pour le reste de ce chapitre : C_s représente la couleur du soleil et C_c la couleur du ciel. Pour un point $\mathbf{P}$ donné, $\vec{N}$ représente sa normale[1], $\vec{L}$ le vecteur unitaire partant de $\mathbf{P}$ et se dirigeant vers le soleil, et $\vec{O}$ le vecteur unitaire partant de $\mathbf{P}$ et se dirigeant vers l'observateur, comme indiqué sur la figure 5.1. Notez que bien que $\vec{N}$, $\vec{L}$ et $\vec{O}$ dépendent de $\mathbf{P}$, ce point n'apparaît pas dans leur notation pour plus de lisibilité. Nous définissons également la fonction saturate telle que

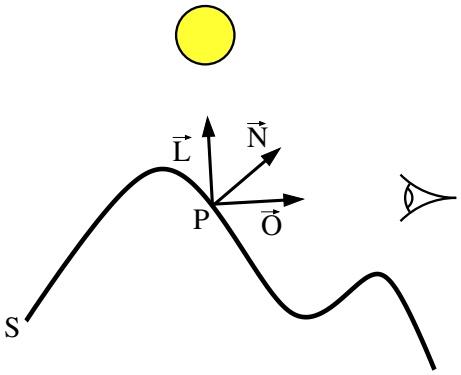


FIG. 5.1: Eléments utilisés dans le calcul de l'illumination d'un point.

$$\text{saturate}(x) = \min(1, \max(0, x))$$

et la fonction normalize telle que

$$\text{normalize}(\vec{V}) = \frac{\vec{V}}{\|\vec{V}\|}$$

5.2 Modèle basique de Lambert

Un cumulus, comme indiqué en section 1.2, a ceci de particulier que sa surface en général correspond en première approximation au modèle de réflectivité de Lambert [Lam60, Bli77] (*cf.* figure 1.1). Nous partons donc d'un *shader* simulant ce modèle, que l'on applique sur la surface du nuage. Ce modèle sera petit à petit modifié et étendu au long des sections qui suivent de façon à simuler les particularités des cumulus.

Ce modèle décrit comment une surface non brillante est illuminée. La luminosité d'une surface en un point, d'après la loi de Lambert, est directement proportionelle au cosinus de l'angle entre la normale à cette surface et la direction de la lumière. Sous forme mathématique, la couleur $I(\mathbf{P})$ au point $\mathbf{P}$ est donnée par :

$$I(\mathbf{P}) = C_s \,\text{saturate}(\vec{N}.\vec{L}) \tag{5.1}$$

L'équation 5.1 constitue donc notre modèle de départ pour le calcul de la couleur du nuage.

[1] **Normale** à une surface S en un point $\mathbf{P}$: vecteur unitaire, partant de $\mathbf{P}$ et orthogonal à S

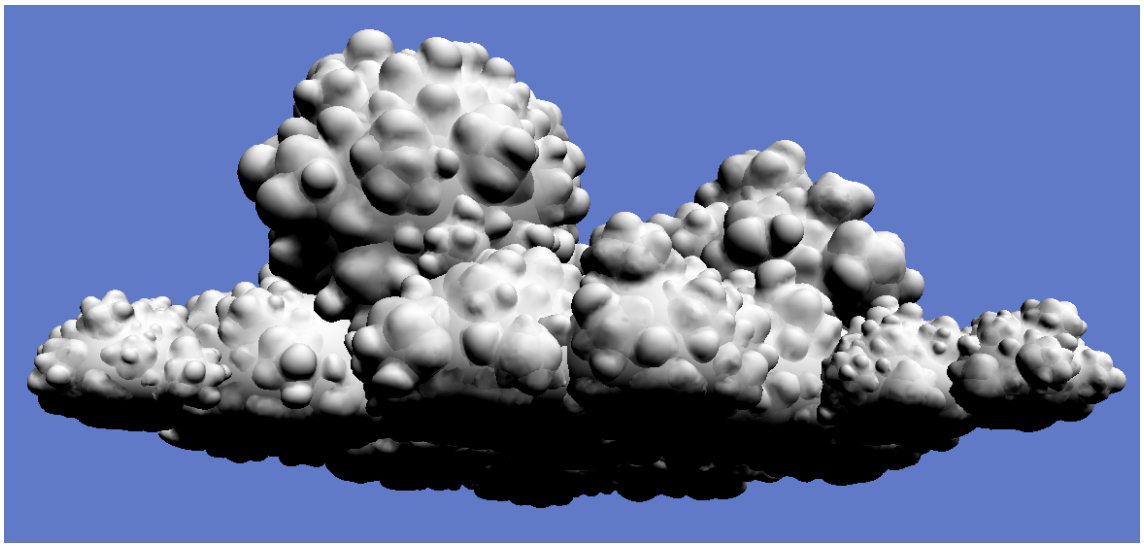


FIG. 5.2: Nuage rendu avec le modèle de Lambert. L'aspect est pour le moment très peu réaliste, et les parties non éclairées sont notamment très sombres.

5.3 Eclairage ambiant

Une des premières choses que nous pouvons remarquer sur la figure 5.2 est que les parties non éclairées du nuage sont totalement noires, ce qui est loin d'être le cas dans la réalité. Nous ajoutons donc à notre équation d'illumination un terme dit *ambiant*, qui représente la lumière ambiante, c'est-à-dire la lumière provenant de toutes les directions. Cette lumière étant réfléchie également dans toutes les directions, ce terme est une simple constante. On considère que dans le ciel, la lumière ambiante a quasiment la couleur du ciel :

$$L(\mathbf{P}) = \text{saturate}(\vec{N}.\vec{L}) \quad (5.2)$$

$$I(\mathbf{P}) = \alpha_a C_c + \alpha_l C_s L(\mathbf{P}) \quad (5.3)$$

Les termes α_a et α_l permettent à la fois à l'équation 5.3 de ne pas saturer (*i.e.,* dépasser la valeur de 1.0, ce qui aurait pour effet d'afficher un blanc uni à l'écran), et à contrôler l'importance de chaque terme de l'équation. Nous aurons par la suite, pour chaque terme ajouté à $I(\mathbf{P})$, un coefficient α du même type.

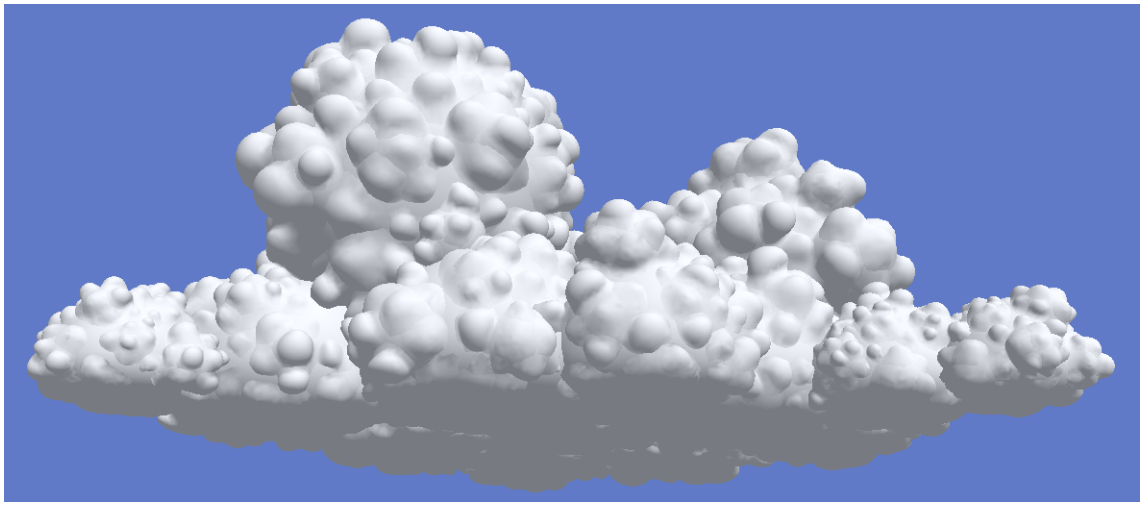


FIG. 5.3: Nuage rendu avec le modèle de Lambert et une lumière ambiante. Bien que le résultat ne soit pas encore à la hauteur de nos attentes, le fait que les parties non éclairées ne soient pas complètement noires ajoute un peu de réalisme.

5.4 Simulation des niveaux supérieurs

Les niveaux contenant les bulles les plus petites (niveau 4 et plus dans les résultats présentés au chapitre 4) comportent des détails fins qui nécessiteraient une géométrie lourde pour être représentés, ce qui réduirait la rapidité de rendu du nuage.

Nous préférons donc les simuler lors du rendu, en perturbant la surface lors de l'exécution des autres *shaders*, par *bump mapping*[2] [Bli78]. Pour cela, nous utilisons le bruit de Perlin [Per85b] pour générer une texture de hauteur, qui représente la différence entre la surface simulée et la surface originale, plane. A partir de cette texture de hauteur, une carte de normales (*normal map*) est générée, représentant la différence d'orientation entre la surface simulée et la surface originale, pour chaque point de la texture. La texture de hauteur (ou *height map*) est générée de façon à ce que la génération de la *normal map* soit aisée : c'est une image en niveau de gris, dont chaque pixel représente une élévation. Plus le pixel est blanc, plus sa hauteur par rapport à la surface sur laquelle il reposera sera grande. La *normal map* est donc simplement générée en effectuant un gradient de la *height map*

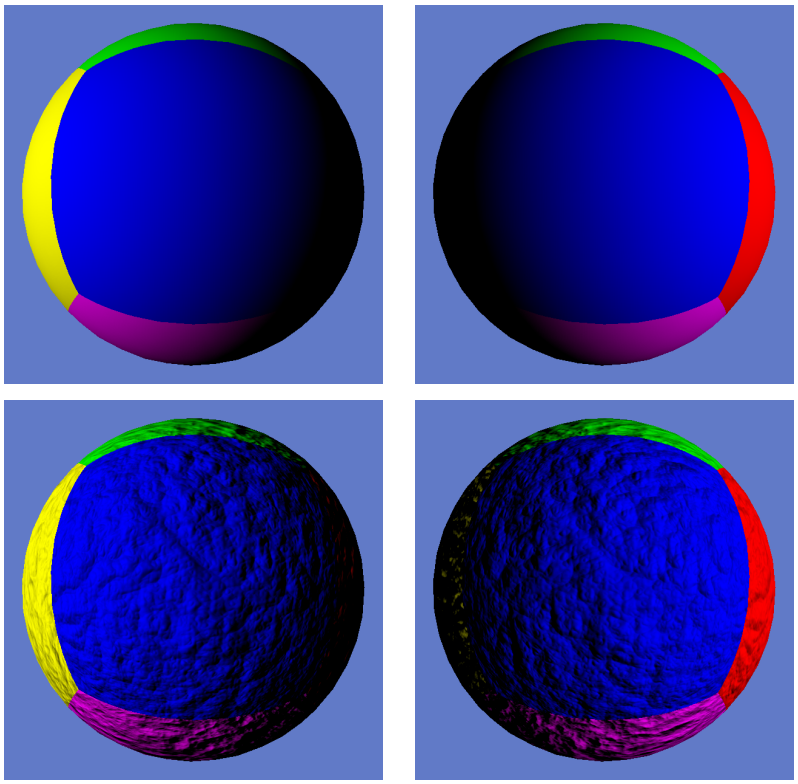


FIG. 5.4: La même sphère, éclairée depuis deux origines différentes, avec (bas) et sans (haut) *bump mapping*. Notez comme l'éclairage de chaque point change suivant l'orientation de la lumière, contrairement au plaquage de texture classique. Remarquez également que, bien que cet effet donne un aspect granuleux à la sphère, cet impression disparaît sur la silhouette, qui reste lisse.

Au moment du rendu de la surface du nuage, le vecteur enregistré dans la *normal map* est utilisé pour perturber la normale à cette surface en chaque point. Cette perturbation peut être modulée par un coefficient α_b représentant l'amplitude de la perturbation : plus α_b est élevé, et plus la perturbation sera importante.

Nous modifions donc l'équation 5.3 pour y introduire la perturbation :

$$\vec{N_b} = \vec{N} + \alpha_b \vec{B}(\mathbf{P}) \tag{5.4}$$

$$L(\mathbf{P}) = \text{saturate}(\vec{N_b}.\vec{L}) \tag{5.5}$$

$$I(\mathbf{P}) = \alpha_a C_c + \alpha_l C_s L(\mathbf{P}) \tag{5.6}$$

[2] ***Bump Mapping*** : technique de rendu consistant à modifier l'aspect d'une surface lisse en fonction de son éclairage de façon à lui ajouter des détails sans alourdir la géométrie. Cette technique peut être comparée au "trompe-l'oeil" utilisé par les peintres pour faire croire à un relief sur un mur plan.

Dans l'équation 5.4, $\vec{B}(\mathbf{P})$ représente le vecteur de perturbation lu à partir de la *normal map* et correspondant au point $\mathbf{P}$.

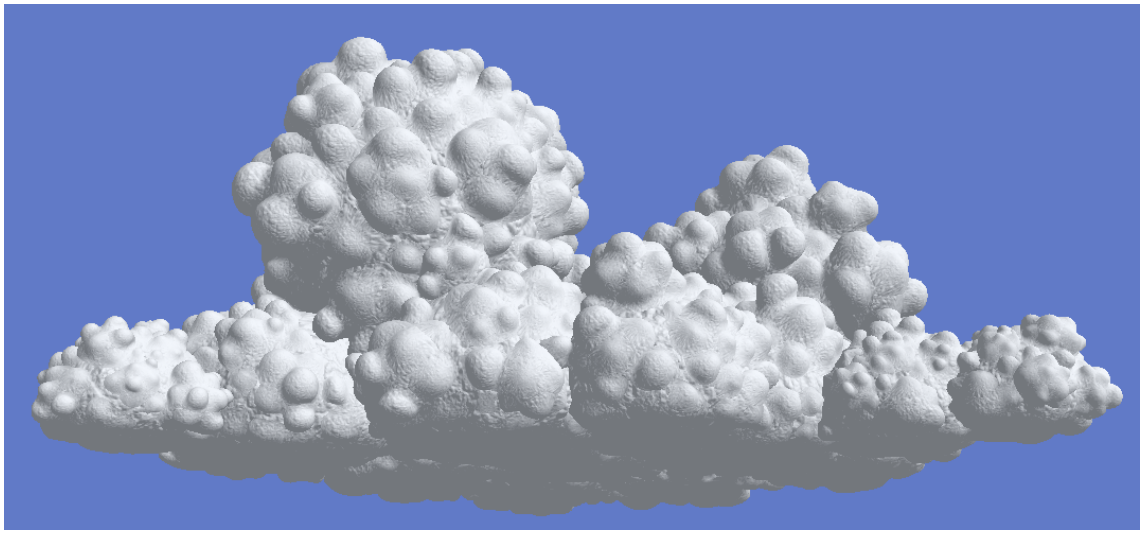


FIG. 5.5: Nuage rendu en ajoutant du *bump mapping*.

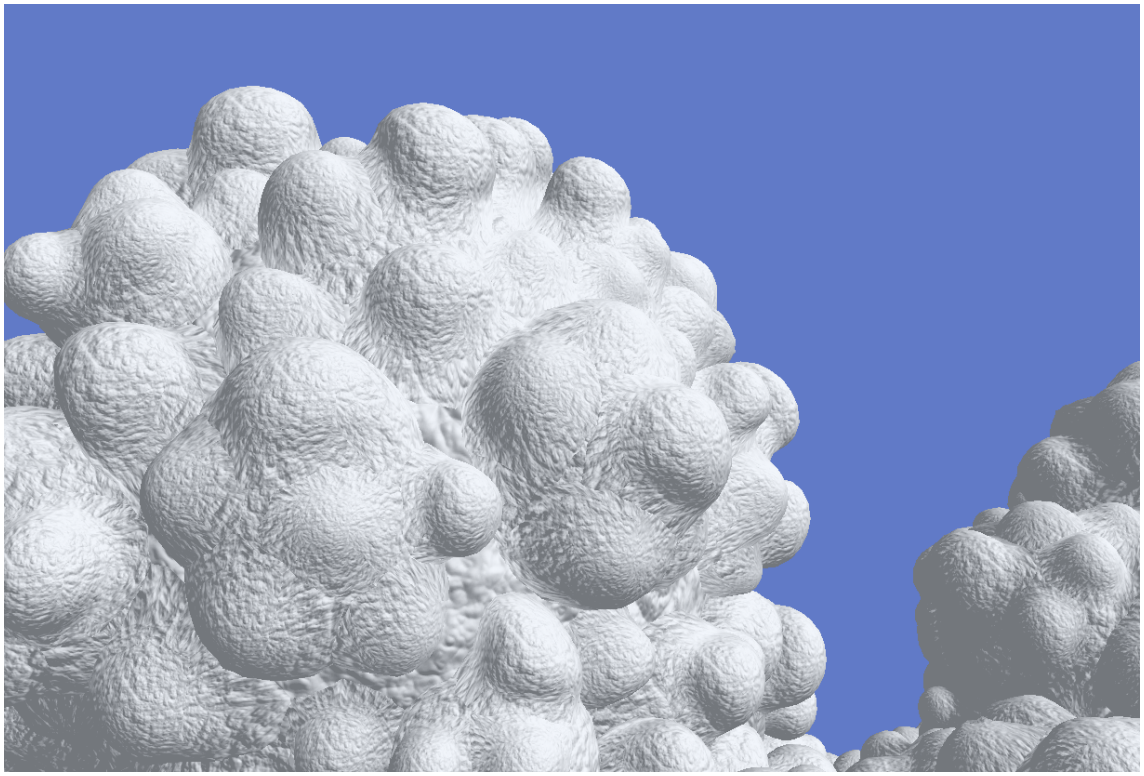


FIG. 5.6: Zoom sur le nuage rendu en ajoutant du *bump mapping*.

5.5 Les pièges à lumière

Nous modifions ensuite notre modèle de façon à simuler l'effet des réflexions multiples se produisant dans les sillons du nuage éclairés par le soleil, comme selon les observations décrites en section 1.2. Comme il est impossible de calculer en temps réel le résultat final correct d'un tel phénomène, nous essayons d'obtenir un résultat similaire avec une technique beaucoup plus rapide, tirant parti de la façon dont nous avons créé la forme du nuage.

En effet, au moment de la génération de la forme du nuage, nous connaissons, pour chaque point $\mathbf{P}$ sur la surface S_l, la distance d_{l-1} de $\mathbf{P}$ à la surface S_{l-1}. Cette information est très précieuse, car nous savons que les sillons correspondent exactement aux minima locaux de d_{l-1}. Nous enregistrons donc cette information avec la géométrie du nuage, et nous en servons ensuite dans notre *shader* en augmentant la luminosité des points possédant un d_{l-1} faible.

$$F^-(\mathbf{P}) \quad = \quad \text{saturate}(1 - \alpha_d \, \mathrm{d}_{l-1}) \tag{5.7}$$

Le paramètre α_d apparaissant dans l'équation 5.7 est un paramètre visant à contrôler la largeur des pièges à lumière : plus α_d sera important et plus l'intensité ajoutée par $F^-(\mathbf{P})$ diminuera vite avec d_{l-1}, et donc plus la lumière apportée par les réflexions multiples sera limitée aux creux des sillons. A l'inverse, si α_d est faible, alors les réflexions multiples auront une aire d'effet plus grande.

Cependant, nous avons besoin de plus d'informations que d_{l-1}. En effet, augmenter la luminosité dans *tous* les sillons aurait un effet irréaliste, car seuls les sillons faisant face au soleil sont sur-illuminés. Nous avons donc besoin de cette information d'orientation. Nous disposons déjà de la normale locale de la surface au point $\mathbf{P}$, mais cette normale ne peut pas nous aider à connaître l'orientation globale du sillon, car elle varie beaucoup trop le long de la surface. En revanche, lors de la génération de la forme, nous pouvons avoir accès, en même temps que d_{l-1}, à la normale $\vec{N}_{l-1}(\mathbf{P})$ du point de S_{l-1} le plus proche de $\mathbf{P}$. Nous savons que cette normale a une fréquence de variation beaucoup plus faible, étant situé sur une surface générée à partir de blobs de plus grande taille. Elle est donc parfaitement adaptée pour déterminer si le sillon fait face au soleil.

$$F^-(\mathbf{P}) \quad = \quad \text{saturate}(1 - \alpha_d \, \mathrm{d}_{l-1}) \vec{N}_{l-1}(\mathbf{P}).\vec{L} \tag{5.8}$$

Mais nous pouvons faire encore mieux. En effet, nous avons remarqué dans nos observations que les réflections multiples se produisent à toutes les échelles. Or, comme notre modèle de forme est également basé sur plusieurs niveaux de taille, nous pouvons nous en servir pour prendre en compte ce phénomène multi-échelle. En effet, si d_{l-1} nous indique si nous sommes dans un sillon à un certain niveau, alors, de la même façon, d_{l-2} nous indiquera si nous somme dans un sillon à l'échelle supérieure, et ainsi de suite. Nous n'enregistrons donc pas, pour chaque point $\mathbf{P}$, la paire $\left(\mathrm{d}_{l-1}, \vec{N}_{l-1}(\mathbf{P})\right)$ mais toutes les paires $\left(\mathrm{d}_{l-1}, \vec{N}_{l-i}(\mathbf{P})\right), i = 1..n$.

$$F^-(\mathbf{P}) \quad = \quad \sum_{j=1}^{n} \text{saturate}(1 - \alpha_d \, \mathrm{d}_{l-j}) \vec{N}_{l-j}(\mathbf{P}).\vec{L} \tag{5.9}$$

Enfin, puisque nous simulons également les niveaux supérieurs non représentés par la géométrie, nous devrions également simuler les pièges à lumière à ces niveaux-là. Etant donné que ces niveaux sont générés à partir d'une carte de hauteur (*cf.* section 5.4), il est très facile d'utiliser la méthode décrite précédemment pour simuler l'effet des réflexions multiples : en passant la *height map* originale au *shader*, celui-ci peut s'en servir pour augmenter la luminosité là où la *height map* contient des valeurs faibles :

$$F^+(\mathbf{P}) \quad = \quad \text{saturate}\left(\alpha_h - \mathrm{h}(\mathbf{P})\right) \vec{N}.\vec{L} \tag{5.10}$$

Dans l'équation 5.10, $\mathrm{h}(\mathbf{P})$ est la hauteur de la *height map* au point $\mathbf{P}$, et α_h est l'équivalent de α_d pour $F^+(\mathbf{P})$: les points pour lesquels $\mathrm{h}(\mathbf{P}) < \alpha_h$ seront considérés comme étant au creux d'un sillon, et recevront donc un supplément d'intensité.

Finalement, nous modifions l'équation 5.6 de façon à prendre en compte les pièges à lumière, en lui ajoutant le résultat des équations 5.9 et 5.10 :

$$F(\mathbf{P}) \quad = \quad F^-(\mathbf{P}) + F^+(\mathbf{P}) \tag{5.11}$$

$$I(\mathbf{P}) \quad = \quad \alpha_a C_c + C_s \left(\alpha_f F(\mathbf{P}) + \alpha_l L(\mathbf{P})\right) \tag{5.12}$$

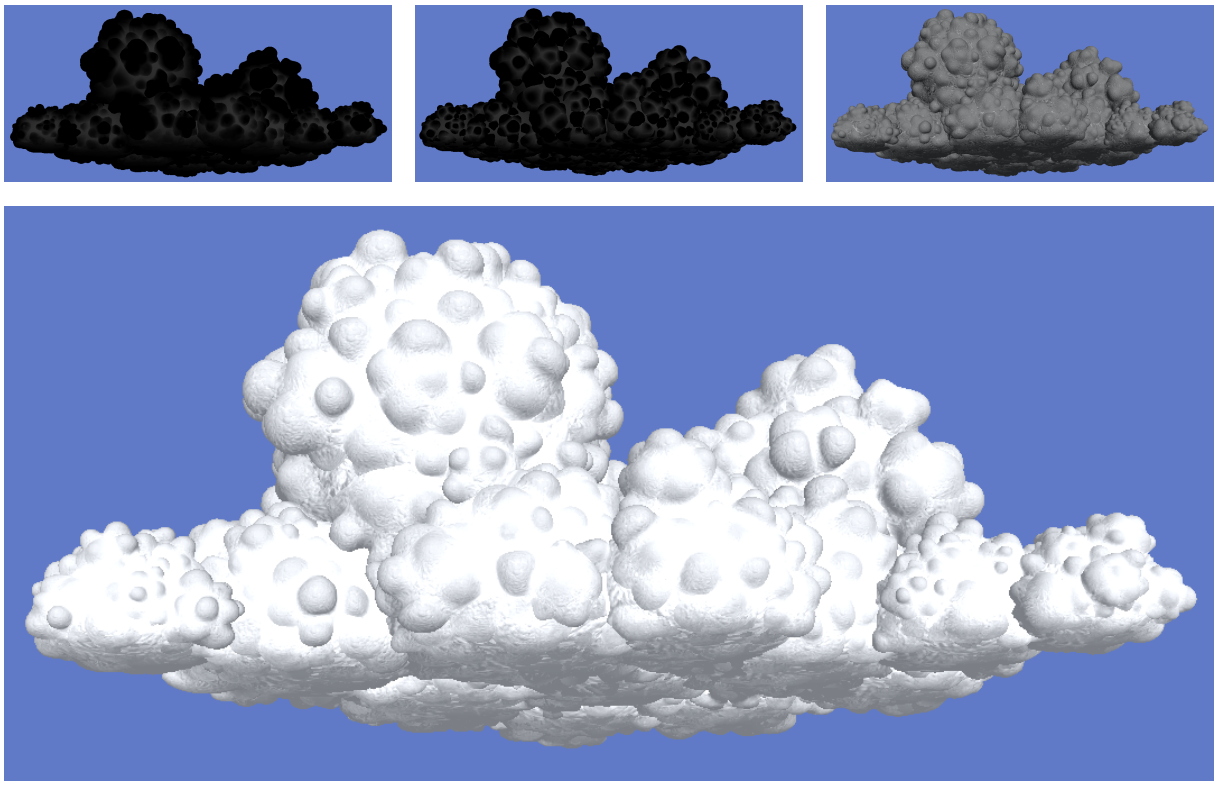


FIG. 5.7: *En bas :* Nuage avec sur-illumination des sillons. *En haut, de gauche à droite :* sur-illumination seule, pour les sillins du niveau 1, du niveau 2 et du niveau 3

5.6 L'effet volumique de Gardner

Malgré le réalisme apporté, notre nuage a un aspect encore trop plâtreux et net : on distingue une surface qui a un air "solide", et non un nuage formé de gouttelettes d'eau. Ceci est bien sûr dû au fait que nous faisons le rendu d'une surface et non d'un volume. Cependant, Gardner [Gar84, Gar85] a montré qu'avec une fonction de rendu adéquate, il est possible de donner à objet simplement rendu par une surface une impression de volume saisissante (*cf.* figure 2.3). Bien que la fonction de rendu de Gardner nécessite à l'époque la technique du ray-tracing[2] pour être appliquée, le matériel graphique actuel, grâce aux *shaders*, nous permet d'appliquer cette méthode en temps réel sur notre nuage. Nous utilisons ici une technique déjà étudiée par Fabrice Neyret [Ney00], et ce qui suit diffère peu de l'original.

Cette méthode consiste à appliquer une transparence à la surface en fonction de la densité du nuage se trouvant derrière le point considéré. Jusqu'à présent, lorsque nous faisions le rendu d'un point à l'écran, nous ne considérions que la surface sur laquelle se trouve se point, et non ce qui se trouve derrière. Nous modifions désormais notre algorithme de façon à, lorsque l'on dessine un point $\mathbf{P}$ à l'écran, on prenne en compte toute l'étendue de matière qui se trouve derrière ce point, le long d'une droite partant de l'observateur et passant par $\mathbf{P}$.

Bien évidemment, il ne suffit pas seulement de dire que l'on prend en compte cette matière pour que tout soit réglé. Il faut, pour prendre en compte cette matière, disposer d'informations sur cette matière, telles que : quelle est la densité du nuage le long de cette droite, et quelles en seront les conséquences en matière d'absorption, de réflexion, de dispersion de la lumière ? Si nous avions un modèle de nuage sous forme de volume, contenant par exemple la densité de gouttelettes d'eau pour chaque élément de volume, nous pourrions calculer ces informations, comme nombre de chercheurs l'ont fait, de Kajiya *et al.* [KH84] à Harris *et al.* [HL01] en passant par Max [Max94], de façon plus ou moins efficaces. Cependant, étant donné que nous ne disposons que d'une surface, nous ne possédons aucune de ces informations. Heureusement, Gardner a montré qu'en extrapolant ces informations à partir du peu de données que l'on a, il est possible d'obtenir des résultats très réalistes (*cf.* figure 2.3).

Pour cela, nous estimons qu'en première approximations, nos blobs ont une forme sphérique. Grâce à cette approximation, il est très facile de déterminer localement quelle est la position du point $\mathbf{P}$ par rapport au blob sur lequel il se trouve. En effet, nous avons accès à la normale $\vec{N}$ de ce point $\mathbf{P}$, et nous

savons que pour une sphère, il y a une bijection entre la normale d'un point et sa position sur la sphère, car la normale est le vecteur unitaire partant du centre de la sphère et dirigé vers $\mathbf{P}$. La normale $\vec{N}$ dans le repère de la caméra nous informe donc directement sur la position de $\mathbf{P}$ sur la sphère : si cette normale est dirigée vers la caméra, $\mathbf{P}$ se trouve au milieu du cercle représentant la sphère à l'écran, ce qui veut dire que la droite D partant de la caméra et passant par $\mathbf{P}$ traverse la sphère de part en part, par le plus long chemin possible à l'intérieur de la sphère. Si $\vec{N}$ est au contraire orthogonal à la direction de la caméra, ceci signifie que $\mathbf{P}$ est sur le contour du cercle représentant la sphère à l'écran, et donc que D ne traverse pas la sphère, mais est tangente à elle.

Nous pouvons donc déduire, grâce à $\vec{N}$, la longueur traversée par D à l'intérieur de notre sphère de rayon R_i :

$$l_i(\mathbf{P}) \quad = \quad 2R_i\vec{N}.\vec{O} \tag{5.13}$$

Cette longueur va nous permettre de calculer analytiquement l'intégrale de la densité du nuage le long de ce rayon. Mais pour cela, nous avons besoin de savoir comment est organisée la densité à l'intérieur d'un nuage. Comme nous l'avons vu au chapitre 1, les cumulus ont ceci de particulier qu'ils ont une densité très importante et quasiment constante. En définissant 1.0 comme étant la densité maximale pouvant être atteinte par un nuage, et 0.0 une densité nulle, nous simulons la répartition de la densité à l'intérieur de notre sphère de nuage comme suit : tous les points situés à une distance de moins de $R_i - r$ du centre sont considérés comme ayant une densité de 1.0. Entre $R_i - r$ et R_i, la densité décroît linéairement jusqu'à 0.0. Ceci nous donne donc la densité $\rho_i(\mathbf{P})$ d'un point $\mathbf{P}$:

$$\rho_i(\mathbf{P}) = \text{saturate}\left(\frac{R_i - \mathrm{d}_i}{r}\right) \tag{5.14}$$

Nous avons maintenant, simplement à partir des équations 5.13 et 5.14, tous les éléments pour calculer une intégrale analytique de la densité traversée par D. Cependant, pour simplifier encore plus nos calculs, nous allons alléger cette intégrale en assumant que la densité est constante tout au long de D à l'intérieur de la sphère. L'intégration se résumera alors à une simple multiplication, au prix d'un réalisme diminué.

Pour ce faire, de la même façon que nous avons calculé $l_i(\mathbf{P})$, nous pouvons calculer la distance minimale entre la droite D et le centre de la sphère :

$$d_i^{min}(\mathbf{P}) \quad = \quad R_i\sqrt{1 - (\vec{N}.\vec{O})^2} \tag{5.15}$$

A cette distance minimale $d_i^{min}(\mathbf{P})$ est associée une densité : c'est la densité maximale $\rho_i^{max}(\mathbf{P})$ traversée par D dans la sphère :

$$\rho_i^{max}(\mathbf{P}) \quad = \quad \text{saturate}\left(\frac{R_i - \mathrm{d}_i^{min}(\mathbf{P})}{r}\right) \tag{5.16}$$

Nous considérons que tous les points traversés par D dans la sphère possèdent cette densité maximale $\rho_i^{max}(\mathbf{P})$. Ainsi, la quantité $\hat{\rho}_i(\mathbf{P})$ de gouttelettes d'eau traversées par D dans la sphère se résume à :

$$\hat{\rho}_i(\mathbf{P}) \quad \approx \quad \rho_i^{max}(\mathbf{P})l_i(\mathbf{P}) \tag{5.17}$$

$$\approx \quad l_i(\mathbf{P})\,\text{saturate}\left(\frac{R_i}{r}\left(1 - \sqrt{1 - (\vec{N}.\vec{O})^2}\right)\right) \tag{5.18}$$

Nous multiplions finalement $\hat{\rho}_i(\mathbf{P})$ par un facteur α_ρ nous permettant d'ajuster l'importance de la variation de densité entre le coeur du nuage et sa corolle. Enfin, et c'est probablement l'un des points les plus important, nous ajoutons un bruit à $\hat{\rho}_i(\mathbf{P})$ de façon à simuler également dans le calcul de cette densité les niveaux supérieurs du nuage non modélisés. Pour être cohérent avec ce qui a déjà été fait dans les sections 5.4 et 5.5, nous utilisons la même carte de hauteur. Nous obtenons finalement un terme que l'on appellera simplement ρ :

$$\rho \quad = \quad \alpha_\rho\Big(1 - \alpha_b\,\mathrm{h}_i(\mathbf{P})\Big)l_i(\mathbf{P})\,\text{saturate}\left(\frac{R_i}{r}\left(1 - \sqrt{1 - (\vec{N}.\vec{O})^2}\right)\right) \tag{5.19}$$

Nous avons au final, grâce à ρ, un moyen de savoir où se trouve un point par rapport à la sphère à laquelle il appartient, et surtout quelle est la densité du nuage se trouvant derrière ce point. Vous aurez remarqué que dans cette section, nous n'avons pas du tout modifié le calcul de la couleur $I(\mathbf{P})$ de $\mathbf{P}$

proprement dite. Cependant, le calcul de ρ décrit ici va beaucoup nous servir dans la section suivante, dédiée à la corolle du nuage.

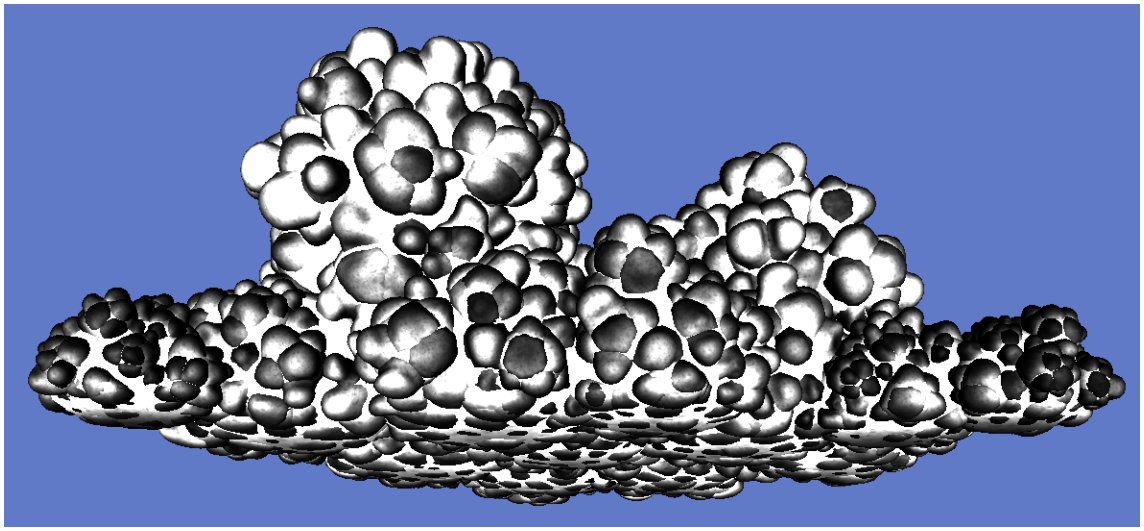


FIG. 5.8: Densité ρ pour chaque point du nuage : l'intensité d'un pixel représente ici ρ.

5.7 La corolle du nuage

Au niveau de la corolle, la surface du nuage possède un comportement visuel spécifique (*cf.* figure 1.2). Grâce au ρ calculé en 5.6, nous sommes capables de savoir si le point que nous dessinons se trouve sur la corolle, et même bien plus puisque ρ nous donne la densité de nuage que l'on traverserait dans le blob sur lequel se trouve **P** si l'on traçait un rayon partant de la caméra et passant par **P**.

Cette corolle est particulière car elle est la seule partie du nuage où la dispersion en avant (ou *forward scattering*) de la lumière a lieu, au contraire des nuages à faible densité où cette dispersion a lieu partout. La conséquence majeure de ce phénomène est que lorsque le soleil se trouve derrière le nuage, on peut voir la corolle s'illuminer, car elle transmet la lumière vers l'observateur, alors que le coeur est sombre, puisqu'il renvoie tous les rayons du soleil.

Une autre conséquence est que les bords du nuage ne se découpent pas avec une netteté parfaite sur le ciel, mais sont en général transparents, laissant passer la lumière provenant de derrière. Un effet intéressant peut également être noté : sur cette corolle, dans la partie la plus dense, le nuage est suffisamment dense pour ne pas être transparent, mais pas assez dense pour diffuser la lumière de la même manière que le coeur. Il en résulte que le nuage, lorsqu'il est éclairé de face, présente une corolle intérieure sombre et peu transparente. Ceci est quasiment invisible sur la silhouette du nuage, ca la corolle est quasiment de la même couleur que le ciel. Cependant, s'il y a un autre nuage derrière une corolle, on distingue très nettement cet effet. Les calculs ci-dessous ne prennent néanmoins pas en compte cet effet et se contentent d'ajouter de la transparence à la corolle et de simuler le *forward scattering*.

De même que pour la section 5.6, nous nous basons sur le travail effectué par Fabrice Neyret[Ney00]. A partir la densité ρ, nous pouvons calculer la quantité de lumière diffusée par le soleil dans la direction de l'observateur à travers le nuage au point **P**. En effet, nous assumons (*cf.* annexe A) que la fonction de phase du nuage est une Gaussienne dont la dérivation standard dépend de la densité traversée. Nous approximons cette Gaussienne par une puissance de cosinus :

$$T_i(\mathbf{P}) = e^{-\frac{(-\vec{O}.\vec{L})^2}{2\rho^2}} \tag{5.20}$$

$$e^{-\frac{x^2}{2\sigma^2}} \approx \cos(x)^{\frac{1}{\sigma^2}} \tag{5.21}$$

$$(5.20) \text{ et } (5.21) \Rightarrow T_i(\mathbf{P}) \approx \cos(-\vec{O}.\vec{L})^{\frac{1}{\rho^2}} \tag{5.22}$$

Nous avons maintenant tous les éléments nécessaires pour calculer la quantité de lumière transmise par

la corolle :

$$T(\mathbf{P}) = \cos(-\vec{O}.\vec{L})^{\frac{\alpha_g}{\rho^2}} \quad (5.23)$$

$$I(\mathbf{P}) = \alpha_a C_c + C_s\left(\alpha_t T(\mathbf{P}) + \alpha_f F(\mathbf{P}) + \alpha_l L(\mathbf{P})\right) \quad (5.24)$$

Le terme α_g introduit dans l'équation 5.23 nous permet d'ajuster la largeur de la Gaussienne utilisée. Enfin, α_t permet d'ajuster l'intensité du terme de transmission.

Parallèlement, nous calculons l'opacité $A(\mathbf{P})$ du nuage au point $\mathbf{P}$. Nous la définissons très simplement :

$$A(\mathbf{P}) = \rho \quad (5.25)$$

Cette opacité est cependant peu réaliste. Son inconvénient est de décroître trop rapidement le long de la corolle, ce qui diminue fortement l'effet du *forward scattering* et empêche la simulation de la corolle sombre. Le temps a cependant manqué pour déterminer une bonne approximation analytique de $A(\mathbf{P})$. Afin d'augmenter l'impression de *forward scattering* , nous avons utilisé une solution d'appoint consistant à augmenter l'opacité lorsque le *forward scattering* est présent :

$$A(\mathbf{P}) = \rho \max(T(\mathbf{P}), 1) \quad (5.26)$$

Cette solution n'a rien de physique et devrait dans le futur être supprimée et remplacée par un calcul correct de $A(\mathbf{P})$.

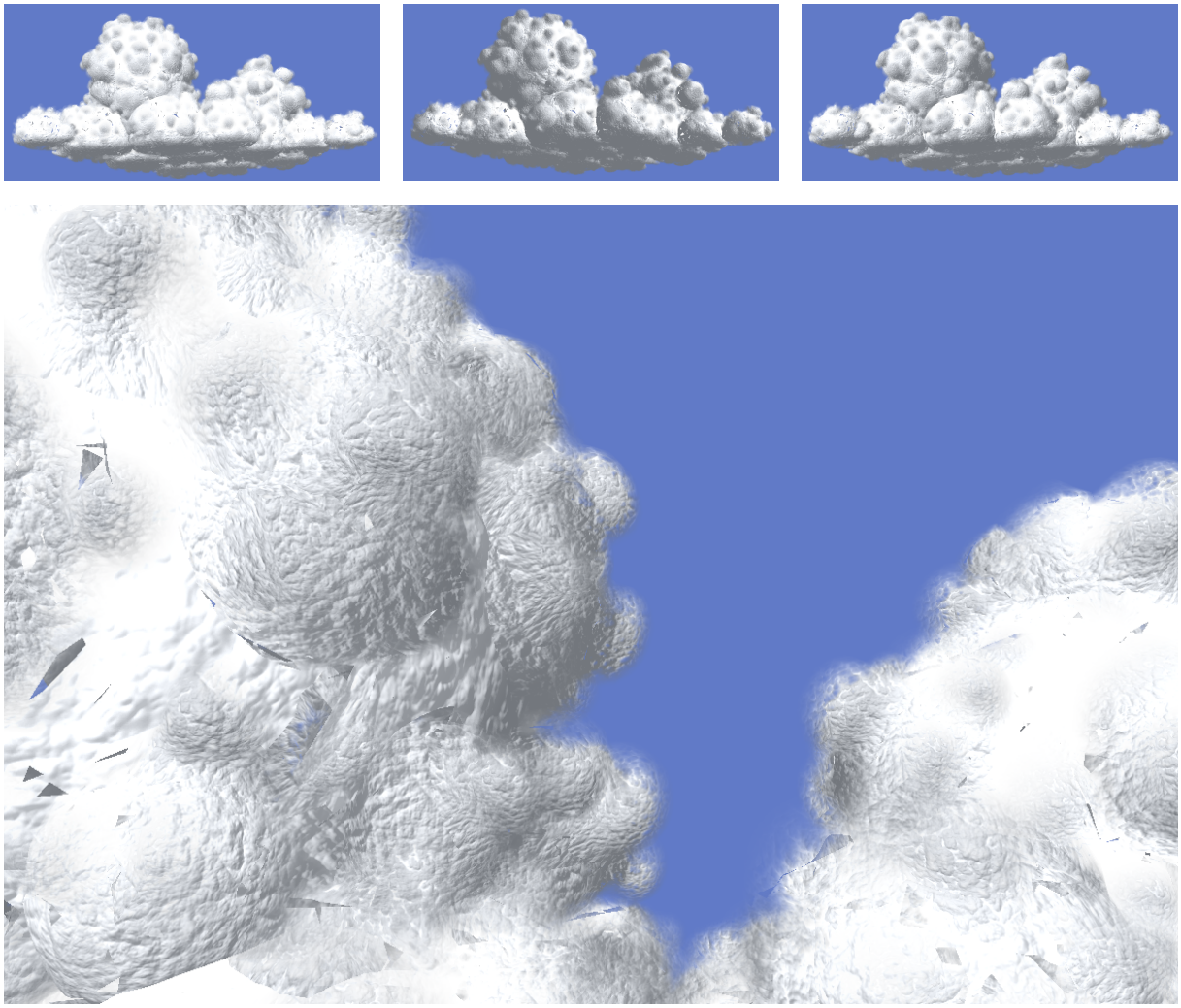


FIG. 5.9: Rendus d'un nuage avec transparence dans la corolle. Notez les "trous" dans la surface.

L'introduction de la transparence amène le problème de l'intersection des faces décrit en section 5.9, comme on peut le voir sur la figure 5.9. Ce problème est inhérent à l'utilisation de la transparence en

conjonction avec le *z-test*, et la seule solution pour éviter les "trous" observés dans le nuage est d'utiliser une géométrie correcte. Nous reviendrons sur ce sujet dans la suite de ce rapport.

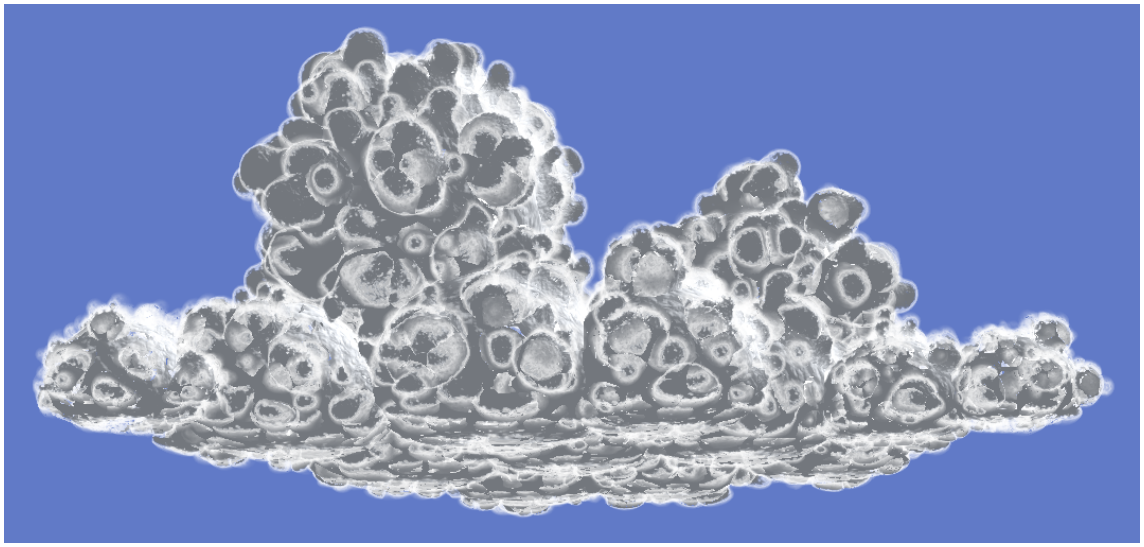


FIG. 5.10: Rendu d'un nuage avec *forward scattering* dans la corolle : le *forward scattering* apparait y compris dans les parties qui de recoivent pas la lumière du soleil !

5.8 Ombrage

Comme vous pouvez le voir sur la figure 5.10, notre modèle présente un défaut majeur : étant local, il ne tient absolument pas compte des objets environnants, et notamment de ceux qui pourraient être amenés à cacher la lumière du soleil. Du coup, les bords de tous les blobs diffusent la lumière, y compris celles qui ne le devraient pas, étant cachées du soleil par le reste du nuage. Nous devons donc trouver un moyen de ne prendre en compte la diffusion que sur la silhouette.

Nous avons pour cela décidé d'utiliser une technique d'ombrage, appelée *shadow mapping* [Wil78, SKvW+92]. Cette technique nous permet à la fois de ne pas avoir de diffusion sur les corolles ne se situant pas sur la silhouette, mais aussi de réduire l'intensité de la réflexion directe du nuage pour les parties ne recevant pas la lumière du soleil.

Cette technique consiste à effectuer dans un premier temps un rendu du nuage depuis le point de vue du soleil. Lors de ce premier rendu, c'est moins la couleur que l'information de profondeur qui nous intéresse. En effet, lors d'un rendu, on calcule non seulement la couleur de chaque pixel affiché, mais aussi la distance de ce pixel par rapport à la caméra. Cette distance, communément appelée z (car elle correspond à la coordonnée du point sur l'axe Z du repère de la caméra), est enregistrée pour chaque pixel dans un tampon nommé *z-buffer*. A chaque fois qu'un autre pixel est dessiné au même endroit à l'écran, la valeur de profondeur correspondant à ce point de l'écran est lue depuis le *zbuffer*, et comparée avec la profondeur du point nouvellement calculé. Si ce point se trouve avoir une profondeur supérieure à celle lue dans le *z-buffer*, ceci veut dire qu'un point se situant plus près de la caméra a déjà été affiché. Le nouveau point ne doit alors pas être affiché, de façon à ne pas cacher un élément qui se trouverait devant lui. Dans le cas contraire, le nouveau point est dessiné par-dessus l'ancien. Ceci constitue la méthode utilisée par toutes les cartes d'accélération 3D pour déterminer quelles sont les surfaces visibles, et se nomme *z-test*.

Ce qui nous intéresse dans le rendu fait depuis le point de vue du soleil est l'état final du *z-buffer*, que nous enregistrons dans une texture, que l'on appelle communément *shadow map*. En effet, ce tampon contient la distance de tous les points visibles depuis le soleil, c'est-à-dire *tous les points qui reçoivent la lumière du soleil*. Lorsque nous effectuons, dans un second temps, le rendu de la scène depuis le point de vue de l'observateur, nous calculons alors pour chaque point, non seulement sa couleur et sa profondeur par rapport à la caméra, mais aussi sa profondeur z_l par rapport au soleil. Nous lisons également la valeur de profondeur z_s de la *shadow map* telle qu'on la lirait si l'on dessinait ce point vu du soleil. Exactement de la même manière que pour le *z-test*, nous comparons z_l et z_s : si $z_l = z_s$, alors ce point est visible depuis le soleil, donc éclairé. Au contraire, si $z_l < z_s$, ce point n'est pas visible depuis le soleil car il est caché par un autre point plus proche du soleil. Il se trouve donc dans l'ombre.

Le *shadow mapping* est tellement bien adapté au matériel graphique qu'il est maintenant supporté par les cartes d'accélération 3D [ERC02] : il est en effet possible à l'intérieur d'un *shader*, d'avoir directement accès aux informations nécessaires au calcul du *shadow mapping*, à sa voir la distance d'un point à une source de lumière z_l, et la valeur z_s de la *shadow map* correspondant à ce point, pour cette lumière.

Discuter les détails, avantages, inconvénients et limites du *shadow mapping* sortirait du cadre de ce rapport. Nous pouvons cependant citer que, au contraire des autres techniques d'ombrages, les performances du *shadow mapping* sont très peu dépendante de la complexité géométrique à la fois des objets créant des ombres et de ceux recevant les ombres, et c'est la raison qui m'a amené à utiliser cette méthode, les nuages générés étant très complexes.

Nous modifions à présent notre *shader* de façon à tenir compte de l'ombre :

$$E(\mathbf{P}) = \begin{cases} 1 \text{ si } z_l = z_s \\ 0 \text{ si } z_l < z_s \end{cases} \tag{5.27}$$

$$I(\mathbf{P}) = \alpha_a C_c + E(\mathbf{P}) C_s \left(\alpha_t T(\mathbf{P}) + \alpha_f F(\mathbf{P}) + \alpha_l L(\mathbf{P})\right) \tag{5.28}$$

Le nuage rendu dans la *shadow map* est dessiné de façon à ce que seulement les points ayant $\rho > 1.0$ (*i.e.,* les points situés au coeur du nuage) soient utilisés pour l'ombre. Ainsi, la corolle, étant partiellement transparente, ne projette pas d'ombre sur le nuage. Ceci pour deux raisons. La première est que dans la réalité, la corolle projette effectivement moins d'ombre que le coeur, étant donné qu'elle est moins dense. Il est vrai qu'elle ne laisse pas totalement passer la lumière, mais un des inconvénients majeurs du *shadow mapping* est qu'il ne permet pas de projeter des ombres "douces" ou partielles : un point est toujours soit totalement dans l'ombre, soit totalement dans la lumière. Nous sommes donc obligé de choisir entre l'un ou l'autre, et nous avons donc décidé de ne pas projeter d'ombre avec la corolle, trouvant cela plus réaliste.

La seconde raison à la non-projection d'ombre par la corolle est que ceci nous permet de conserver le *forward scattering*. En effet, si nous prenions la totalité de la surface du nuage comme occulteur, alors le *forward scattering* n'aurait jamais lieu, puisque la corolle serait considérée comme bloquant totalement la lumière ! En n'utilisant pas la corolle pour la projection de l'ombre, nous permettons donc au *forward scattering* d'être présent, tout en le supprimant aux endroits où il est réellement inexistant, c'est-à-dire là où même le coeur du nuage stoppe le soleil.

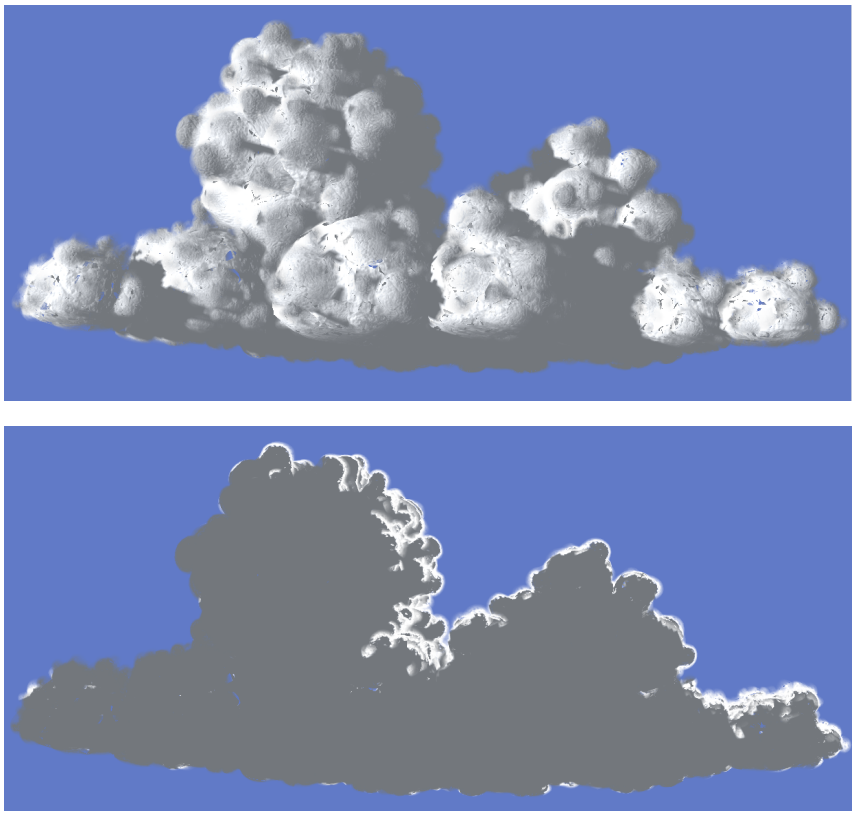


FIG. 5.11: Rendu d'un nuage avec prise en compte de l'ombre.

Enfin, bien que la raison première d'introduire le *shadow mapping* était d'obtenir une corolle illuminée correctement, nous obtenons automatiquement un effet secondaire non des moindres : nous pouvons simuler l'ombre que le nuage porte sur lui-même. Il est également possible, sans rien changer à notre modèle, de prendre en compte les ombres que le nuage porte sur d'autres objets (sol, autres nuages, etc.), et les ombres que d'autres objets projettent sur notre nuage.

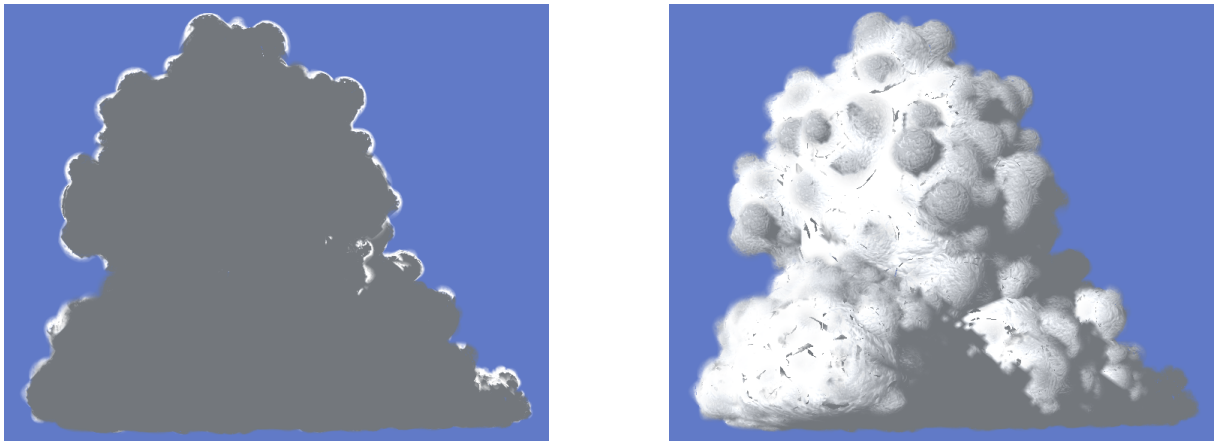


FIG. 5.12: Rendu du même nuage, d'un autre point de vue, avec prise en compte de l'ombre.

5.9 Lissage des normales

Dans notre implémentation du générateur de nuages, la surface finale n'est en fait pas une seule surface continue, mais simplement l'ensemble des surfaces de tous les blobs. Bien que cela reste très proche de l'idée de base, il en résulte deux principaux problèmes :

Premièrement, il arrive que certaines surfaces s'intersectent. Ceci ne pose pas de problèmes pour une surface opaque, mais dès qu'on applique de la transparence à la surface, on verra apparaître des artefacts lors du rendu, dûs à l'utilisation du *z-test* (*cf.* section 5.8) en même temps que la transparence. La seule solution à ce problème est d'éviter les intersections, soit en découpant les faces en sous-faces le long des intersections, soit en ne générant pas de géométrie possédant des intersections. Nous reviendrons sur ce problème en section 6.1

Deuxièmement, il y a une transition nette entre les normales lorsque l'on passe de la surface d'un blob à la surface d'un autre blob. Ceci produit des "cassures" inesthétiques dans le rendu de notre nuage, montrant que sa géométrie possède des angles, ce qui ne devrait pas être le cas. Pour pallier à ce problème, nous lissons les normales entre les surfaces des blobs adjacentes afin de masquer ces angles et d'obtenir un rendu tel qu'il le serait si la surface du nuage n'était faite que d'un seul tenant. La normale $\vec{N}$ utilisée dans tout ce chapitre n'est donc pas la normale originale, mais la normale lissée. Ce lissage peut s'effectuer une fois pour toute après la génération de la géométrie.

Il consiste, pour une normale $\vec{N}$, lui ajouter une fraction des normales $\vec{N}_{l-j}(\mathbf{P})$ des surfaces des niveaux supérieurs, proportionnellement à d_{l-j} :

$$w_j(\mathbf{P}) = \mathrm{saturate}(\alpha_w \, \mathrm{d}_{l-j}) \tag{5.29}$$

$$\vec{N}' = \mathrm{normalize}\left(\sum_{j=1}^{n} w_j(\mathbf{P})\vec{N} + (1 - w_j(\mathbf{P}))\,\vec{N}_{l-j}\right) \tag{5.30}$$

C'est le $\vec{N}'$ de l'équation 5.30 qui a été utilisé dans tout ce chapitre, excepté pour le calcul de ρ (équation 5.19, et toute la section 5.6 en général), où l'on a besoin de la normale exacte correspondant au blob.

5.10 Formule finale

Nous regroupons ici toutes les formules intervenant finalement dans le calcul du rendu de notre nuage :

$$l_i(\mathbf{P}) = 2R_i\vec{N}.\vec{O} \tag{5.31}$$

$$\rho = \alpha_\rho\Big(1-\alpha_b\,\mathrm{h}_i(\mathbf{P})\Big)l_i(\mathbf{P})\,\mathrm{saturate}\left(\frac{R_i}{r}\left(1-\sqrt{1-(\vec{N}.\vec{O})^2}\right)\right) \tag{5.32}$$

$$w_j(\mathbf{P}) = \mathrm{saturate}(\alpha_w\,\mathrm{d}_{l-j}) \tag{5.33}$$

$$\text{Normale lissée de } \mathbf{P} : \vec{N'} = \mathrm{normalize}\left(\sum_{j=1}^{n} w_j(\mathbf{P})\vec{N} + (1-w_j(\mathbf{P}))\,\vec{N}_{l-j}\right) \tag{5.34}$$

$$\text{Normale perturbée de } \mathbf{P} : \vec{N}_b = \vec{N'} + \alpha_b\vec{B}(\mathbf{P}) \tag{5.35}$$

$$\text{Eclairage de } \mathbf{P} : E(\mathbf{P}) = \begin{cases} 1 \text{ si } z_l = z_s \\ 0 \text{ si } z_l < z_s \end{cases} \tag{5.36}$$

$$\text{Surillum. (niv. inférieurs) en } \mathbf{P} : F^-(\mathbf{P}) = \sum_{j=1}^{n} \mathrm{saturate}(1-\alpha_d\,\mathrm{d}_{l-j})\vec{N}_{l-j}(\mathbf{P}).\vec{L} \tag{5.37}$$

$$\text{Surillum. (niv. supérieur) en } \mathbf{P} : F^+(\mathbf{P}) = \mathrm{saturate}\Big(\alpha_h-\mathrm{h}(\mathbf{P})\Big)\vec{N'}.\vec{L} \tag{5.38}$$

$$\text{Surillum. en } \mathbf{P} : F(\mathbf{P}) = F^-(\mathbf{P}) + F^+(\mathbf{P}) \tag{5.39}$$

$$\textit{Forward scattering} \text{ en } \mathbf{P} : T(\mathbf{P}) = \cos(-\vec{O}.\vec{L})\frac{\alpha_g}{\rho^2} \tag{5.40}$$

$$\text{Terme de Lambert en } \mathbf{P} : L(\mathbf{P}) = \mathrm{saturate}(\vec{N}_b.\vec{L}) \tag{5.41}$$

$$\textbf{Couleur de } \mathbf{P} : I(\mathbf{P}) = \alpha_a C_c + E(\mathbf{P})C_s\Big(\alpha_t T(\mathbf{P}) + \alpha_f F(\mathbf{P}) + \alpha_l L(\mathbf{P})\Big) \tag{5.42}$$

$$\textbf{Opacité de } \mathbf{P} : A(\mathbf{P}) = \rho\max\left(T(\mathbf{P}),1\right) \tag{5.43}$$

Le code des shaders correspondant à ces formules est disponible en annexe B.

5.11 Résultats

On est en droit de se demander, au vu des formules de la section 5.10, si un *shader* est capable de calculer de telles opérations en temps réel. Et en effet, sur notre implémentation, la vitesse de rendu des images présentées dans ce rapport est descendu jusqu'à 0.3 images par seconde, ce qui signifie que trois secondes ont été nécessaires pour calculer chaque image. Ceci est loin du temps réel. Cependant, nous pensons qu'il est tout à fait possible d'obtenir un rendu temps réel sans même simplifier les équations utilisée. Nous discuterons de ces possibilités au chapitre 6.

Le réalisme est une notions subjective. Nous pensons cependant que les résultats obtenus sont plus réalistes que les modèles précédents, notamment au niveau de la richesse du détail. Nous laissons au lecteur le soin d'apprécier les quelques exemples de résultats qui suivent.

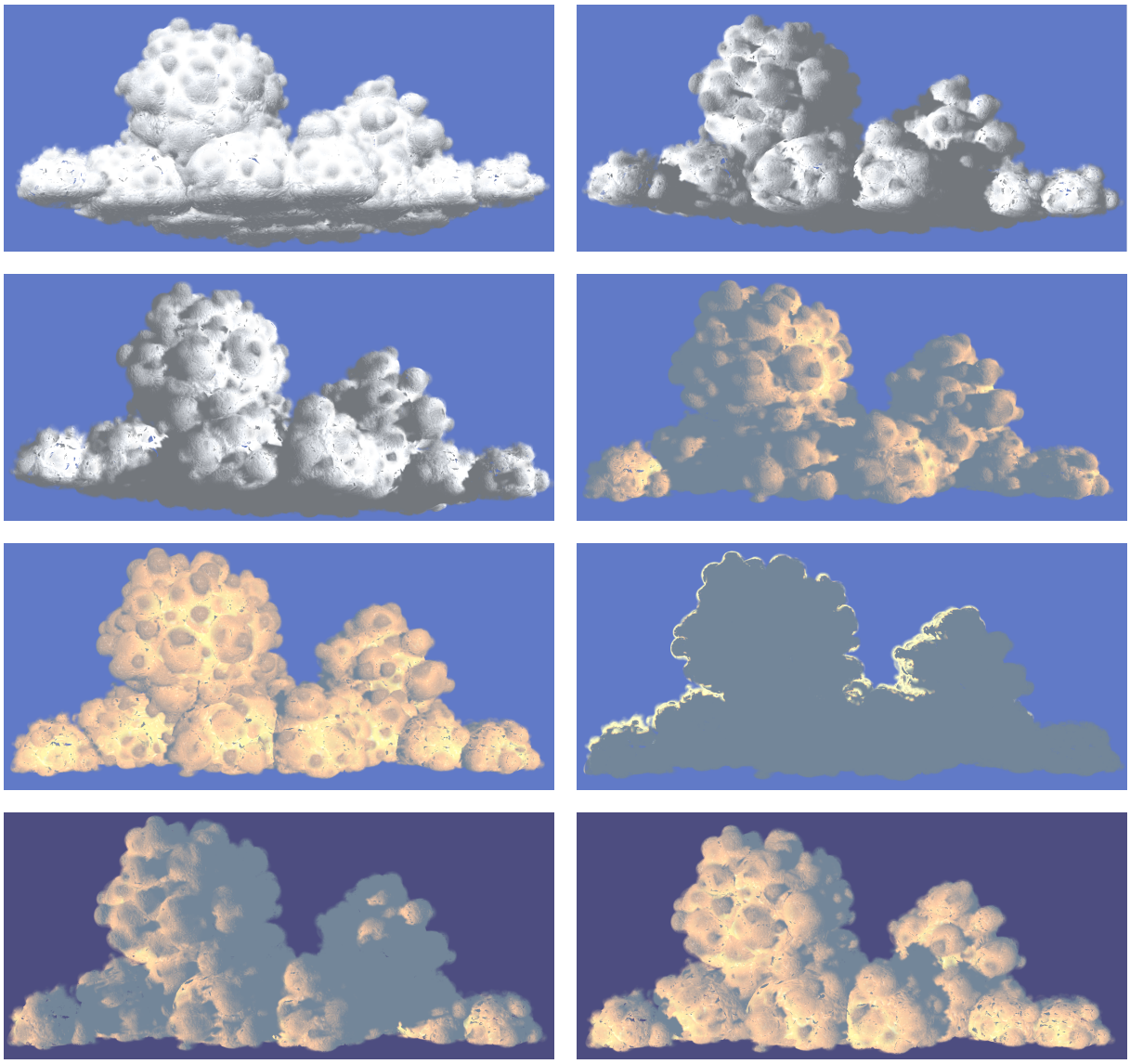


FIG. 5.13: Quelques nuages rendus avec notre modèle final

Chapitre 6

Perspectives futures

L'objectif à long terme dans lequel est inscrit ce stage est l'obtention d'un ciel nuageux évoluant de façon plausible et avec un rendu également plausible, avec une vitesse d'affichage interactive. On peut donc dégager à partir de nos résultats partiels un certain nombre d'idées de travaux permettant d'avancer vers cet objectif.

6.1 Modélisation de la forme

Notre forme de nuage a deux problèmes majeurs. Le premier, nous l'avons vu dans les résultats et notamment sur la figures 5.9, est le fait que la géométrie contient des faces se rentrant les unes dans les autres, ce qui provoque des sortes de "trous" dans le nuage à l'affichage. Ceci est dû à la façon dont nous avons implémenté notre théorie et non à la théorie elle-même. Nous devrions donc dans un futur proche modifier notre programme de façon à ce que la surface finale générée soit d'un seul tenant, et non un ensemble de petites surfaces comme c'est le cas actuellement.

Le deuxième problème est que la géométrie que nous créons à l'heure actuelle est excessivement lourde : beaucoup de parties pourraient être simplifiées en réduisant le nombre de triangles représentant la surface. D'autant plus que, comme notre nuage n'est pas une surface unique mais l'ensemble des blobs, une grande partie de la géométrie (la partie des blobs se trouvant à l'intérieur du nuage, et donc non visible) est totalement inutile ! De plus, si le nuage se trouve loin de l'observateur (ce qui est destiné à arriver à long terme, lorsque l'on affichera un ciel chargé de nuages), le nombre de triangles à afficher devrait être encore plus réduit, certains détails géométriques devenant indistinguables. Cette orgie de triangles est en grande partie responsable de la lenteur du rendu, d'autant plus que, puisque nous avons des faces transparentes, nous devons les trier dans un ordre particulier à chaque image.

Notre prochain modèle de forme de nuage devra donc être une surface unique, avec un nombre de triangle réduit et adapté à la taille apparente du nuage (modèle *adaptatif*.

Une perspective intéressante pourrait être de pouvoir générer une forme de nuage à partir de données existantes. En particulier, il devrait être possible d'adapter la méthode proposée par Dobashi *et al.* [DNYO98] pour créer les particules des premiers niveaux, pour ensuite générer les particules des niveaux suivants selon notre méthode. Ceci permettrait d'obtenir des nuages réalistes sans intervention aucune de l'utilisateur.

6.2 Rendu

6.2.1 Implémentation

Notre modèle de rendu souffre de quelques défauts, et son implémentation a également besoin d'être améliorée. En effet, du fait du peu de temps disponibles, les *shaders* écrits sont loin d'être optimisés. En particulier, quasiment aucun calcul n'est fait dans le *vertex shader*. A priori, tout calcul ne nécessitant pas l'accès à une texture pourrait être fait dans ce *vertex shader*, et son résultat serait disponible sous forme d'interpolation dans le *pixel shader*. Ceci accélérerait énormément le temps de calcul, car le nombre de calculs ne serait alors pas proportionnel au nombre de pixels dessinés à l'écran, mais au nombre de points de la géométrie, qui lui est fixe et approximativement cent fois plus petit que le nombre de pixels dessinés pour nos exemples !

D'autre part, beaucoup d'améliorations peuvent être possibles concernant la simulation des niveaux inférieurs. En premier lieu, la texture utilisée pour ces niveau a actuellement la mauvaise lacunarité, ce qui rend le résultat peu réaliste. En utilisant simplement une texture ayant la même lacunarité que la géométrie, on obtiendrait un résultat bien plus crédible. Le plaquage de la *height map* et de la *normal map* pourrait également être améliorer. Ces textures sont en effet a l'heure actuelle légèrement étirées à certains endroit, ce qui nuit aussi au réalisme.

Ensuite, la technique d'ombrage que nous utilisons actuellement n'applique que des ombres "dures", ce qui n'est pas très plausible pour un nuage. Nous étudions actuellement la possibilité de rendre ces ombres plus douces, bien que ce problème en général reste ouvert. Une meilleure technique d'ombrage aurait également pour effet de rendre la transition entre la corolle illuminée et le coeur sombre moins marquée, ce qui serait une grande amélioration.

6.2.2 Modèle

Nous l'avons vu en 5.7, le calcul de l'opacité est pour le moment mauvais, et l'une de nos première priorité est de trouver un modèle de cette opacité plus correct. Une approximation qui aurait également besoin d'être améliorée est le calcul de ρ, qui considère pour le moment que la densité sur le chemin de D est la densité maximum rencontrée, ce qui est loin d'être vrai. D'autre part, si nous considérons que nos blobs sont sphériques, ils ne le sont en réalité pas et il serait bon d'améliorer cette approximation, en enregistrant par exemple le rayon de courbure de la surface avec $\mathbf{P}$, ou en utilisant d'autres méthodes pour extrapoler $l_i(\mathbf{P})$ et $d_i^{min}(\mathbf{P})$.

Au niveau de la sur-illumination des sillons, nous obtiendrions de bien meilleurs résultats si nous prenions en compte les vrais sillons plutôt que d'utiliser une astuce telle que la nôtre. D'autant plus que notre technique ne permet pas de simuler les sillons à la plus grande échelle, c'est-à-dire ceux qui se trouvent entre deux blobs du niveau racine, étant donné que ce niveau ne repose sur aucune surface.

En ce qui concerne la corolle, sa densité est en réalité loin d'être la même dans tous les endroits d'un même nuage : il est très courant d'avoir dans une partie du nuage (typiquement, la partie éclairée par le soleil, où les échanges thermiques ont lieu) un volute très marqué et en pleine formation, où la corolle sera quasiment inexistante, alors que dans une autre partie (à l'ombre), ce même nuage perd de sa densité et présente une corolle très large. Tenir compte de ce phénomène apporterait aussi sa part de réalisme.

Le modèle dépendant fortement de seulement deux couleurs, la couleur du ciel C_c et la couleur du soleil C_s, il est très important de les choisir avec soin. Ayant un piètre niveau en graphisme, j'ai choisi des couleurs qui ne sont pas toujours les meilleures, et ceci se ressent énormément dans les résultats, où peut voir un nuage éclairé par un coucher de soleil devant un ciel de midi. Il convient donc de choisir de meilleures couleurs, l'idéal étant d'utiliser un modèle de ciel tels que [PSS99] ou [NDKY96] et un environnement autre qu'un fond vide.

D'autre part, le nombre de paramètres à régler dans notre modèle (10) est relativement élevé et devrait être réduit, en les fixant ou en les restreignant à des valeurs mesurées dans la réalité.

Enfin, on pourrait envisager intégrer des effets de réalisme additionnels tels que l'influence du sol, du ciel, et des autres nuages sur l'aspect de celui-ci. Il sera également intéressant d'étudier dans quelle mesure ce modèle de rendu peut s'appliquer sur d'autres objets naturels convectifs tels que la fumée ou certains types d'avalanches.

6.3 Animation

Le modèle de nuage que nous proposons étant basé sur des particules, il est possible de l'animer. Nous envisageons l'adaptation d'un modèle d'animation tel que celui décrit en [Ney03] pour notre cas. Il devrait également être possible d'utiliser le modèle d'animation phénoménologique présenté en [Ney97] pour animer nos particules de façon plausible.

Conclusion

Nous avons décrit dans ce mémoire une méthode phénoménologique permettant de réaliser un modèle (forme et aspect visuel) de cumulus à partir d'observations de scènes réelles. Nous avons également présenté une approche permettant le rendu rapide réaliste de ce modèle, et nous pensons que cette approche peut aisément être amenée au temps-réel.

Le résultat de ce stage a permis une certaine avancée dans le domaine de la simulation de nuages en fournissant une modélisation phénoménologique de forme de cumulus possédant une richesse de détails plus importante que les modèles précédents. Notre modèle est également propice à l'animation, qui fera l'objet de futures études. Nous pensons que le rendu de cette surface donne des résultats au moins aussi réalistes que ceux de Gardner, à une vitesse interactive.

Les nuages sont un objet passionnant mais complexe, et le travail de manquera pas pendant la thèse pour pousser ces pistes jusqu'au bout. A noter que des sociétés comme *Buf compagnie* (effets spéciaux) et *Nadéo* (jeux vidéos) sont intéressées par des modèles de nuages réalistes et efficaces, les modèles de la littérature étant considérés comme insuffisants et trop coûteux.

Bibliographie

[Bli77] James F. Blinn. Models of light reflection for computer synthesized pictures. volume 11, pages 192–198, July 1977. 5.2

[Bli78] James F. Blinn. Simulation of wrinkled surfaces. In *Computer Graphics (Proceedings of SIGGRAPH 78)*, volume 12, pages 286–292, August 1978. 10

[Bli82] James F. Blinn. Light reflection functions for simulation of clouds and dusty surfaces. In *Proceedings of the 9th annual conference on Computer graphics and interactive techniques*, volume 16, pages 21–29. ACM Press, July 1982. 2.2

[CA97] Patricia Crossno and Edward Angel. Isosurface extraction using particle systems. In *IEEE Visualization '97*, pages 495–498, November 1997. 4.3

[DKY+00] Yoshinori Dobashi, Kazufumi Kaneda, Hideo Yamashita, Tsuyoshi Okita, and Tomoyuki Nishita. A simple, efficient method for realistic animation of clouds. In *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, pages 19–28, July 2000. 2.1, 2.2

[DNYO98] Y. Dobashi, T. Nishita, H. Yamashita, and T. Okita. Modeling of clouds from satellite images using metaballs. In *Pacific Graphics '98*, October 1998. 2.1, 6.1

[Ebe97] David S. Ebert. Volumetric procedural implicit functions : A cloud is born. In Turner Whitted, editor, *SIGGRAPH 97 Technical Sketches Program*. ACM SIGGRAPH, Addison Wesley, August 1997. ISBN 0-89791-896-7. 2.1, 2.2

[EMP+94] David Ebert, Kent Musgrave, Darwyn Peachey, Ken Perlin, and Worley. *Texturing and Modeling : A Procedural Approach*. Academic Press, October 1994. ISBN 0-12-228760-6. 6

[ERC02] Cass Everitt, A. Rege, and C. Cebenoyan. Hardware shadow mapping, 2002. 5.8

[ES00] Pantelis Elinas and Wolfgang Stürzlinger. Real-time rendering of 3d clouds. *Journal of Graphics Tools*, 5(4) :33–45, 2000. 8

[FSJ01] Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen. Visual simulation of smoke. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, pages 15–22, August 2001. 2.1

[Gar84] Geoffrey Y. Gardner. Simulation of natural scenes using textured quadric surfaces. In Hank Christiansen, editor, *Computer Graphics (SIGGRAPH '84 Proceedings)*, volume 18, pages 11–20, July 1984. 2.1, 6, 3, 5.1, 5.6

[Gar85] Geoffrey Y. Gardner. Visual simulation of clouds. In B. A. Barsky, editor, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pages 297–303, July 1985. 2.1, 6, 3, 5.1, 5.6

[Gas93] Marie-Paule Gascuel. An implicit formulation for precise contact modeling between flexible solids. In *Proceedings of SIGGRAPH 93*, Computer Graphics Proceedings, Annual Conference Series, pages 313–320, August 1993. 4.2, 4.3

[GC91] W. W. Grabowski and T. L. Clark. Cloud-environment interface instability : Rising thermal calculations in two spatial dimensions. *Journal of the Atmospheric Sciences*, 48 :527–546, 1991. 1.1

[GC93] W. W. Grabowski and T. L. Clark. Cloud-environment interface instability, part II : Extension to three spatial dimensions. *Journal of the Atmospheric Sciences*, 50 :555–573, 1993. 1.1

[HISL03] Mark J. Harris, William Baxter III, Thorsten Scheuermann, and Anselmo Lastra. Simulation of cloud dynamics on graphics hardware. In *Graphics Hardware 2003*, pages 92–101, July 2003. 2.1, 2.2

[HL01] Mark J. Harris and Anselmo Lastra. Real-time cloud rendering. *Computer Graphics Forum*, 20(3) :76–84, 2001. 5.6

[KH84] James T. Kajiya and Brian P. Von Herzen. Ray tracing volume densities. In *Computer Graphics (Proceedings of SIGGRAPH 84)*, volume 18, pages 165–174, July 1984. 2.2, 5.6

[Lam60] T. H. Lambert. *Photometria Sive de Mensura et Gradibus Luminus, Colorum et Umbrae*. Eberhard Klett, 1760. 5.2

[Max86] Nelson Max. Light diffusion through clouds and haze. *Comput. Vision Graph. Image Process.*, 33(3) :280–292, 1986. 2.2

[Max94] Nelson L. Max. Efficient light propagation for multiple anisotropic volume scattering. In *Fifth Eurographics Workshop on Rendering*, pages 87–104, June 1994. 2.2, 5.6

[Mic] Microsoft. Microsoft flight simulator 2004. (ï¿½alement prï¿½entï¿½comme *sketch* ï¿½Siggraph 2003) http://www.ofb.net/~eggplant/clouds/. 2.1

[MYDN01] R. Miyazaki, S. Yoshida, Y. Dobashi, and T. Nishita. A method for modeling clouds based on atmospheric fluid dynamics. In *9th Pacific Conference on Computer Graphics and Applications*, pages 363–372, October 2001. 2.1

[NDKY96] Tomoyuki Nishita, Yoshinori Dobashi, Kazufumi Kaneda, and Hideo Yamashita. Display method of the sky color taking into account multiple scattering. In *Proceedings of Pacific Graphics '96*, pages 117–132, 1996. 6.2.2

[Ney97] Fabrice Neyret. Qualitative simulation of cloud formation and evolution. In D. Thalmann and M. Van de Panne, editors, *8th Eurographics Workshop on Computer Animation and Simulation (EGCAS'97)*, pages 113–124. Eurographics, Springer Wein, September 1997. 1.3, 3, 6.3

[Ney00] Fabrice Neyret. A phenomenological shader for the rendering of cumulus clouds. Technical Report RR-3947, INRIA, May 2000. 8, 5.1, 5.6, 5.7

[Ney03] Fabrice Neyret. Advected textures. *Symposium on Computer Animation'03*, july 2003. 6.3

[NND96] Tomoyuki Nishita, Eihachiro Nakamae, and Yoshinori Dobashi. Display of clouds taking into account multiple anisotropic scattering and sky light. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, pages 379–386. ACM SIGGRAPH, Addison Wesley, August 1996. 2.1, 2.2, 4.3

[Per85a] Ken Perlin. An image synthesizer. In B. A. Barsky, editor, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19(3), pages 287–296, July 1985. 2.1

[Per85b] Ken Perlin. An image synthesizer. In *Computer Graphics (Proceedings of SIGGRAPH 85)*, volume 19, pages 287–296, July 1985. 10

[PSS99] A. J. Preetham, Peter S. Shirley, and Brian E. Smits. A practical analytic model for daylight. In *Proceedings of SIGGRAPH 99*, Computer Graphics Proceedings, Annual Conference Series, pages 91–100, August 1999. 6.2.2

[RT87] Holly E. Rushmeier and Kenneth E. Torrance. The zonal method for calculating light intensities in the presence of a participating medium. In *Computer Graphics (Proceedings of SIGGRAPH 87)*, volume 21, pages 293–302, July 1987. 2.2

[SKvW+92] Mark Segal, Carl Korobkin, Rolf van Widenfelt, Jim Foran, and Paul E. Haeberli. Fast shadows and lighting effects using texture mapping. In *Computer Graphics (Proceedings of SIGGRAPH 92)*, volume 26, pages 249–252, July 1992. 5.8

[SSEH03] Joshua Schpok, Joseph Simons, David S. Ebert, and Charles Hansen. A real-time cloud modeling, rendering, and animation system. *Symposium on Computer Animation'03*, pages 160–166, july 2003. 2.1

[WH94] Andrew P. Witkin and Paul S. Heckbert. Using particles to sample and control implicit surfaces. In *Proceedings of SIGGRAPH 94*, Computer Graphics Proceedings, Annual Conference Series, pages 269–278, July 1994. 4.3

[Wil78] Lance Williams. Casting curved shadows on curved surfaces. In *Computer Graphics (Proceedings of SIGGRAPH 78)*, volume 12, pages 270–274, August 1978. 5.8

Annexe A

Dispersion à grande échelle

Une fonction de phase[1] vers l'avant peut être représentée par une Gaussienne en coordonnées polaires avec une déviation standard σ^2. Cette valeur correspond à une quantité unitaire de gouttelettes traversées. Lorsqu'un volume deux fois plus large des traversé, chaque rayon dévié sortant de la section unitaire est dévié une deuxième fois dans la section suivante. Ceci correspond exactement à la convolution de la Gaussienne par elle-même. Plus généralement, si on considère un volume n fois plus grand que le volume unitaire, on obtient une dispersion cumulée dont la déviation standard est $n.\sigma^2$. Comme les coordonnées polaires sont cycliques, le diagramme devient finalement isotrope au dessus d'une échelle donnée, qui est probablement environ une douzaine de mètres. Les observations faites, telles que le fait qu'un cumulus est hautement réflectif et que le cœur d'un cumulus mûr illuminé par derrière est uniformément sombre, appuient cette théorie. En revanche, les autres types de nuages ou les jeunes cumulus sont très transparents et relativement anisotropes, car ils sont à la fois fins et peu denses.

[1] **Fonction de phase** : fonction décrivant comment est redistribuée de la lumière incidente dans les diverses directions

Annexe B

Code source des *shaders*

Les deux pages suivantes contiennent le code source des *shaders* créés pendant ce stage.

```
#include "basics.cg"


/*****************************************************/

/*
    Entrees :
        IN.HPos.xyzw   : P
        IN.PCol.xyz    : N
        IN.SCol.xyz    : N'
        IN.Tex0.xyzw   : TexCoord ShadowMap
        IN.Tex1.xyz    : N_0
        IN.Tex1.w      : d_0
        IN.Tex2.xyz    : N_1
        IN.Tex2.w      : d_1
        IN.Tex3.xyzw   : TexCoord
        IN.Tex3.w      : R
*/
pixout main(    vertout IN,
        uniform float3 L,
        uniform float r = 1500,
        uniform float alpha_b = .8,
        uniform float alpha_h = .3,
        uniform float alpha_w = 5,
        uniform float alpha_a = .75,
        uniform float alpha_s = 0.3,
        uniform float alpha_l = .35,
        uniform float alpha_t = 2,
        uniform float alpha_g = 1,
        uniform float alpha_rho = .0005,
        uniform samplerCUBE Texture,
        uniform samplerCUBE NormalMap,
        uniform float Trans = 8,
        uniform float4x4 ModelViewIT,
        uniform sampler2D ShadowTexture
     )
{

    float3 C_c = float3( .60, .70, .80 );
    float3 C_s = float3( 1, .5, .1 );
    pixout OUT;

//Ombre
    float4 ShadowTexCoord = IN.Tex0;
    float E = tex2Dproj( ShadowTexture, ShadowTexCoord ).x;

//Bump
    float3 NprimeWorld = normalize( IN.SCol.xyz*2-1 );
    float3 TexCoord = IN.Tex3.xyz;
    float2 B = 2 * texCUBE( NormalMap, TexCoord ).xy - float2( 1, 1 );
    float3 Tan2 = normalize( cross( NprimeWorld, float3( 0, 1, 0 ) ) );
    float3 Tan1 = cross( NprimeWorld, -Tan2 );

    float3 N_b = mul( (float3x3)ModelViewIT,
                          normalize( ( 1-alpha_b) * NprimeWorld
                                     + alpha_b * (  B.x * Tan2
                                                  + B.y * Tan1
                                                  )
                                   )
                        );

//Lambert
    float Lambert = saturate( dot( N_b, L ) );

//Sillons
    float h = texCUBE( Texture, TexCoord ).x;
    float d_0 = IN.Tex1.w;
    float3 N_0 = IN.Tex1.xyz;
    float d_1 = IN.Tex2.w;
    float3 N_1 = IN.Tex2.xyz;
    float3 Nprime = mul( (float3x3)ModelViewIT, NprimeWorld );
    float F_minus = saturate( dot( N_1, L ) ) * saturate( (1-d_1*alpha_w) );
    F_minus += saturate( dot( N_0, L ) ) * saturate( (1-d_0*alpha_w) );
    float F_plus = saturate( alpha_h - h ) * saturate( dot( Nprime, L ) ) ;
    float F = F_minus + F_plus;

//Rho
    float3 N = mul( (float3x3)ModelViewIT, normalize( IN.PCol.xyz*2-1 ) );
    float R = IN.Tex3.w;
    float NO = N.z;
    float l_i = 2*R*NO;
    float rho = saturate( alpha_rho*( 1-alpha_b*h )*l_i)*saturate( R/r*( 1-sqrt( 1-NO*NO) ) );

//Corolle
    float a = (1-rho);
    float OL = ( max( saturate( -L.z ), 0 ) );
```



```
    float c = (1-rho) * 1 * pow( OL, alpha_g/a );
    float T = saturate( (1-rho)*c*Trans );


//Couleur finale
    OUT.Col.rgb = C_c*alpha_a + E*C_s*( alpha_l*Lambert + alpha_s*F + alpha_t*T );

//Opacité
    OUT.Col.a = rho * max( T, 1 );

    return OUT;
}


/*
    Entrees :
        IN.HPos.xyzw   : P
        IN.PCol.xyz    : N
        IN.Tex3.xyz    : TexCoord
        IN.Tex3.w      : R
*/
pixout shadowfunc(  vertout IN,
        uniform float4x4 ModelViewIT,
        uniform samplerCUBE Texture,
        uniform float r = 1500,
        uniform float alpha_b = .8,
        uniform float alpha_rho = .0005
 )
{
    pixout OUT;

//Rho
    float3 TexCoord = IN.Tex3.xyz;
    float h = texCUBE( Texture, TexCoord ).x;
    float3 N = mul( (float3x3)ModelViewIT, normalize( IN.PCol.xyz*2-1 ) );
    float R = IN.Tex3.w;
    float NO = N.z;
    float l_i = 2*R*NO;
    float rho = saturate( alpha_rho*( 1-alpha_b*h )*l_i)*saturate( R/r*( 1-sqrt( 1-NO*NO) ) );

    OUT.Col.rgb = 1;
    OUT.Col.a = rho+.00001;
    return OUT;
}
```

Aug 31, 04 1:23 **vertex_shader.cg** Page 1/1

```
#include "basics.cg"

/*
    Entrees :
        IN.VPos.xyzw : P
        IN.Norm.xyz  : N'
        IN.PCol.xyz  : N
        IN.Tex0.xyz  : TexCoord
        IN.Tex0.w    : R
        IN.Tex1.xyz  : N_0
        IN.Tex1.w    : d_0
        IN.Tex2.xyz  : N_1
        IN.Tex2.w    : d_1
    Sorties :
        OUT.HPos.xyzw   : P
        OUT.PCol.xyz    : N
        OUT.SCol.xyz    : N'
        OUT.Tex0.xyzw   : TexCoord ShadowMap
        OUT.Tex1.xyz    : N_0
        OUT.Tex1.w      : d_0
        OUT.Tex2.xyz    : N_1
        OUT.Tex2.w      : d_1
        OUT.Tex3.xyzw   : TexCoord
        OUT.Tex3.w      : R

*/
vertout main(   appin IN,
        uniform float4x4 ShadowMatrix )
{
    vertout OUT;
    float4x4 ModelViewProj = glstate.matrix.mvp;
    float3x3 ModelViewIT = glstate.matrix.invtrans.modelview[0];

//CoordonnÃ©es homogÃ¨nes
    OUT.HPos = mul( ModelViewProj, IN.VPos );

//GÃ©nÃ©ration de coordonnÃ©es de texture
    OUT.Tex0.xyzw = mul( ShadowMatrix, IN.VPos );

//Divers paramÃ¨tres
    OUT.Tex1.xyz = mul( ModelViewIT, IN.Tex1.xyz );
    OUT.Tex1.w = IN.Tex1.w;
    OUT.Tex2.xyz = mul( ModelViewIT, IN.Tex2.xyz );
    OUT.Tex2.w = IN.Tex2.w;
    OUT.Tex3.xyzw = IN.Tex0.xyzw;
    OUT.PCol.xyz = .5*IN.PCol.xyz+.5;
    OUT.SCol.xyz = .5*IN.Norm.xyz+.5;

    return OUT;
}



/*
    Entrees :
        IN.VPos.xyzw : P
        IN.PCol.xyz  : N
        IN.Tex0.xyz  : TexCoord
        IN.Tex0.w    : R
    Sorties :
        OUT.HPos.xyzw   : P
        OUT.PCol.xyz    : N
        OUT.Tex3.xyz    : TexCoord
        OUT.Tex3.w      : R

*/
vertout shadowfunc(     appin IN )
{
    vertout OUT;
    float4x4 ModelViewProj = glstate.matrix.mvp;

//CoordonnÃ©es homogÃ¨nes
    OUT.HPos = mul( ModelViewProj, IN.VPos );

//Divers paramÃ¨tres
    OUT.Tex3.xyzw = IN.Tex0.xyzw;
    OUT.PCol.xyz = .5*IN.PCol.xyz+.5;


    return OUT;
}
```

Aug 31, 04 1:38 **basics.cg** Page 1/1

```
struct appin
{
    float4 VPos    : POSITION;       // model space vertex position
    float4 VWeight : BLENDWEIGHT;    // weights for skinning (u,1-u)
    float4 Norm    : NORMAL;         // model space vertex normal
    float4 PCol    : COLOR0;         // primary color (diffuse)
    float4 SCol    : COLOR1;         // seconary color (specular)
    float4 PSize   : PSIZE0;         // point size
    float4 Tex0    : TEXCOORD0;      // texture coordinate set 0
    float4 Tex1    : TEXCOORD1;      // texture coordinate set 1
    float4 Tex2    : TEXCOORD2;      // texture coordinate set 2
    float4 Tex3    : TEXCOORD3;      // texture coordinate set 3
};

//=====================================================================
//                                                    Vertex -> Fragment
// Define outputs from vertex shader
//=====================================================================

struct vertout
{
    float4 HPos    : POSITION;      // homogeneous clip space postion
    float4 PCol    : COLOR0;        // primary (diffuse) color
    float4 SCol    : COLOR1;        // secondary (specular) color
    float  Fog     : FOG;           // fog factor
    float  PSize   : PSIZE;         // point size
    float4 Tex0    : TEXCOORD0;     // texture coordinate set 0
    float4 Tex1    : TEXCOORD1;     // texture coordinate set 1
    float4 Tex2    : TEXCOORD2;     // texture coordinate set 2
    float4 Tex3    : TEXCOORD3;     // texture coordinate set 3
};

struct pixout
{
    float4 Col  : COLOR;
    float  Depth : DEPTH;
};
```

Annexe C

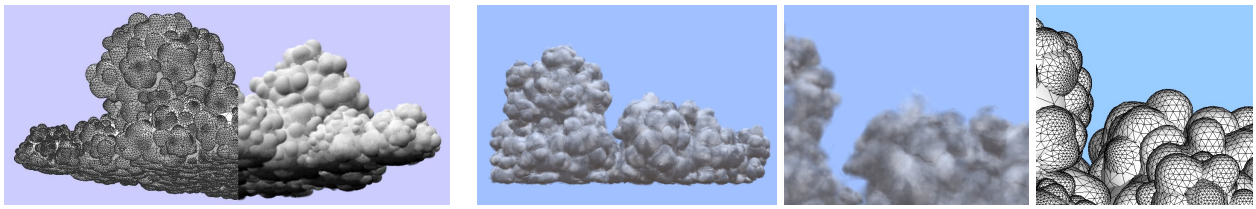
Short paper accepté à *Eurographics*

Les quatre pages qui suivent sont la copie de l'article publié aux *Eurographics Short Presentations*. Celui-ci est basé sur le travail présenté au chapitre 4, et sera présenté lors de la conférence le 2 septembre 2004 à Grenoble.

Modeling clouds shape

Antoine Bouthors† Fabrice Neyret†

EVASION-GRAVIR/ IMAG-INRIA
http://www-imagis.imag.fr/Publications/2004/BN04/



Abstract

We propose a model for representing the shape of cumulus clouds. We draw on several approaches that we combine and extend: We store a hierarchy of quasi-spherical particles (or blobs) living on top of each other. The shape of these particles is defined by an implicit field which deforms under the influence of neighbor particles. We define a set of shaders to simulate a volumetric appearance in the spirit of Gardner's textured ellipsoids*, to make the shape* appear *continuous, and to account for dedicated shading effects.*
In this paper we deal only with the definition of the shape. However, we believe that this model is well suited to be integrated with particle animation and advanced rendering.

Keywords: *natural phenomena, clouds, particles, implicit surfaces, shaders, procedural modeling.*

1. Introduction and Previous Work

Clouds are an important element of natural scene in quality rendering as well as real-time applications (e.g. flight simulators). Billowing clouds (and smoke) are complex matter in many ways: the shape has a fractal nature, so does the animation, and the rendering should account for local variations as well as global volumetric illumination. A wide variety of clouds exists, requiring different approaches. In this paper we only deal with *cumulus clouds*, which are well-formed and very contrasted clouds showing a quasi-surface.

Three ways have been followed in the representation of cloud shapes: Volumetric clouds (either explicit [NND96] or procedural [Ebe97, SSEH03]), billboards [HL01, DKY*00], and surfaces [Gar84, Gar85]. (Note that the dense billboarding of [DKY*00] may be considered as a volume discretization). Volume approaches are still not amenable to real-time rendering of clouds: volume rendering techniques are getting very fast, but the resolution required by a cloud sky – which is both large and very detailed – will long be too much for graphics hardwares. Billboards and impostors are the current solution used in games [HL01, Mic], but still a small number of slices must be used to keep an interactive pace (the problem lies in the huge number of pixels drawn rather than in the number of polygons). Surfaces permit very efficient rendering, but they look too crude to represent a volumetric shape. However, Gardner [Gar84, Gar85] was able to make ellipsoid surfaces look volumetric in the scope of ray-tracing thanks to view-dependent transparency shaders evaluated at each ray. Fortunately, recent hardware allows to define *pixels shaders*, making this approach amenable to real-time rendering. An early attempt in that direction was done in [ES00].

Concerning the specification of clouds shape, three approaches have been followed: using simulation data as input (or some extrapolation of acquired data such as satellite views), relying on procedural fractal noise [Per85] and a shaping function such as implicit surfaces like in [Ebe97, SSEH03], or relying on particles (manually or procedural placed). In particular, [NND96] combined hierarchical particles and implicit surfaces: a level is built upon the previous one by tessellating the implicit surface defined by the last layer of particles and placing next level blobs evenly on this surface.

† [Antoine.Bouthors | Fabrice.Neyret]@imag.fr

Considering that cloud shape is simply the result of the air movement, other possible approaches consist in simulating the fluid dynamics [FSJ01, HBSL03] or an animated system of particles [Ney03].

Our model is based on particles (or blobs) associated to an implicit field which allows to define a surface (to be rendered or to sustain other particles). This primitive is associated to a shader in order to make it appear volumetric. Our hierarchical model relies on subdivision and repulsion to populate each level evenly. The implicit field of particles is responsible for deforming them so to avoid particle intersections in the spirit of [Gas93].

Figure 1: A real cloud.

2. Our representation

A cloud is composed of a set of hierarchical levels recursively defined on top of each other. Each level l consists of a set of particles p_i and defines a surface S_l. Each particle of level l is defined by local and global parameters: its location $\mathbf{P}_i$ on the surface S_{l-1}, its radius r_i, its flattening e_i, and blending parameters defined globally for each level. Some of these parameters are automatically adjusted depending on the level or on the surrounding particles whereas the others can be directly or indirectly tweaked by the user.

Each particle defines its own surface S_i, which consists of a 'pure' shape (a flattened sphere) altered to fit the surrounding particles (taking the particle's blending parameters into account). This is done using an implicit formulation inspired by [Gas93]. This implicit surface is defined by:

$$S_i = \{\mathbf{P} \in \mathbf{R}^3 / f_i(\mathbf{P}) = 1\}$$

where $f_i(\mathbf{P}) = g_i(\mathbf{P}) + h_i(\mathbf{P})$

$g_i(\mathbf{P})$ is a basic flattened spherical field function which yields the 'pure' shape:

$$g_i(\mathbf{P}) = \exp\left(1 - \frac{\mathrm{d}_i}{r_i(1 - e_i \mathrm{d}_{l-1})}\right)$$

with d_i and d_l the distance from $\mathbf{P}$ to $\mathbf{P}_i$ and S_l, respectively. The flattening (set randomly) accounts for the various stages of development of cloud bubbles and provides a less uniform aspect of the cloud surface. $h_i(\mathbf{P})$ is a field function which alters this spherical shape to meet our fitting constraints, as in [Gas93]. Their purpose was to define exact contact between two shapes. Contrary to them, we do not want blobs to stick to each other (real blobs on clouds do not, see Figure 1): we want *furrows* between blobs, which should nevertheless be continuous. We define $h_i(\mathbf{P})$ as a combination of several field functions controlled by the particle parameters: $h_i(\mathbf{P}) = m_i(\mathbf{P}) + n_i(\mathbf{P}) + o_i(\mathbf{P})$

• $m_i(\mathbf{P})$ is a field function that prevents the blobs from overlapping, and is thus in its spirit very close to that of [Gas93]. However, we added an offset parameter ε to repulse the surfaces more than for contact in order to create furrows between the blobs. Also, we used a different equation for the bulge generation to produce a smoother transition between the repulsion area and the pure shape far from the interaction area. This gives $m_i(\mathbf{P}) = \sum_j m_i^j(\mathbf{P})$

$$\text{with} \quad m_i^j(\mathbf{P}) = (1 - \varepsilon - g_j(\mathbf{P}))\min(1, g_j^2(\mathbf{P}))$$

• $n_i(\mathbf{P})$ is a field function that enlarges the blob surface as it is closer to the base surface (*i.e.,* the previous level surface S_{l-1}) so that the particle does not appear as "popping" out of the cloud. $n_i(\mathbf{P})$ is controlled by two parameters : b, that defines the amount of blending, and I, that adjusts the fraction of the bubble that will be enlarged to create the blending. We choose:

$$n_i(\mathbf{P}) = b \ \min\left(1, e^{-\frac{I d_{l-1}}{r_i}}\right) e^{1 - \frac{d_i}{r_i}}$$

• $o_i(\mathbf{P})$ is a field function responsible for the flat cloud bottom, by restricting the blob surfaces above a given absolute height h_0 with a stiffness (*i.e.,* repulsion force) α_h.

$$o_i(\mathbf{P}) = g_i(\mathbf{P}) \min\left(0, \frac{\text{height}(\mathbf{P}) - h_0}{\alpha_h}\right)$$

Finally, S_l is defined from the potential $f_l(\mathbf{P}) = \max\left(f_{l-1}(\mathbf{P}), \max_i f_i(\mathbf{P})\right)$

3. Making a cloud shape

We need the surface S_{l-1} in order to place the particles of the next level, and to render the last level of the hierarchy. We can obtain the discretized surfaces of all the blobs belonging to one hierarchy level $l-1$ by using the method described in [Gas93]. To place the particles p_i of level l, we preferred extracting an isosurface of $l-1$ using a particle system as described in [WH94] and [CA97], instead of discretizing explicitly the surface S_{l-1} as done in [NND96]. This particle system is then used directly as the particle set of level l, using the repulsion radius of the particles as the blob radius.

This approach simulates particles that repulse each other and grow to populate all the available surface. When a particle has grown too much or when an area is not crowded enough, the concerned particles split to create new particles. On the other hand, if a part of the surface is overcrowded some particles are removed. A target inter-particle distance is provided (*i.e.,* twice the target particle radius), that all particles will have after reaching the equilibrium state.

We want to reproduce a natural distribution, so regular distributions are not desired. We take different radius of influence for each particle, so that all the particles of a level do not have the same aspect and the layer look uneven. To take in account this new independent radius of influence, we modified the algorithm [WH94] as follows : if the energy of a particle is too low, the repulsion radius of the particle evolves so as to reach the desired energy, as in [WH94]. Otherwise, it evolves so as to reach the desired radius. In addition, the desired energy is also different for each particle, as it is modulated as much as the desired radius.

Once the particle set of level l is created on the implicit surface S_{l-1}, the blobs surfaces are defined as described in section 2. Thus we can generate the implicit surface S_l from these blobs that we will use to place the particles of level $l+1$, and so on. The last one defines the cloud shape.

Particles of level 0 (*i.e.,* root) have no implicit surface to rely on and have to be created another way. Depending on the application purpose, this can be done procedurally (*e.g.,* to create a cloud sky) or this can be controlled by the user. In the first case we create a small number of particles randomly positioned on a horizontal plane or in a small subspace. In the second case, an interface lets the user shape and place them by hand, allowing him to create realistic features such as cloud turrets.

4. Rendering a cloud shape

The scope of this paper is the shape, not the rendering. However, local details are not always best represented by a geometric surface, especially if they are complex or fuzzy. Kajiya suggests in [Kaj85] to switch from geometry to texture to shaders with distance (or size). This is precisely what the Gardner's *textured ellipsoid* model [Gar85] does. We define the local aspect of the cloud using textures and shaders in the same spirit.

First, we simulate blobs at lower scales relying on a texture mapped on the blobs surface (we used a classical Perlin solid texture).

Then we define a Gardner-like shader simulating a fuzzy layer of material by increasing the transparency t near the silhouette (it is thus a view-dependent shader). The silhouette proximity is detected using $\vec{N}.\vec{C}$ which values 0 on the silhouette, where $\vec{N}$ is the local normal (*i.e.,* the field gradient) and $\vec{C}$ is the camera direction. Then we can set the transparency t as $(1-\alpha\vec{N}.\vec{C})^{\alpha_s}$, where α and α_s control the thickness and the sharpness of the transparent margin, respectively.

Similarly, continuity between blobs can be eased using a "normal blending" shader: assuming the final mesh corresponds to level l, we store at each mesh vertex the potential p and the normalized gradient $\vec{g}$ of S_{l-1}. This provides us with a normalized distance $d = 1-p$ to the base surface of the blobs and a normal of this base surface. Thus we can interpolate the normals close to the base surface so that the furrows looks continuous even if the blob surfaces do not connect perfectly smooth: $\vec{N'} = (1-p^{\alpha_N})\vec{N} + p^{\alpha_N}\vec{g}$ where α_N controls the size of the smoothing area.

5. Results

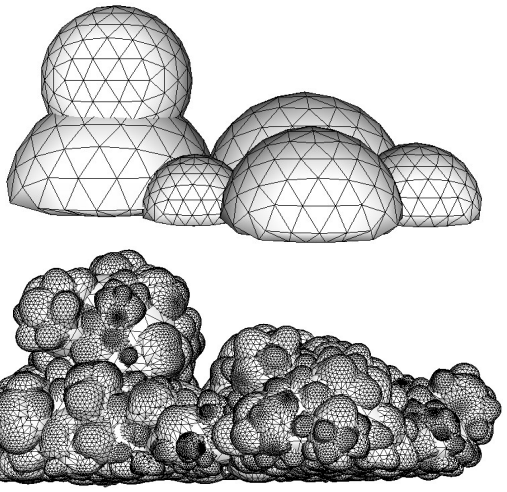


Figure 2: *Top:* 6 user-defined blobs. *Bottom:* with 2 levels generated.

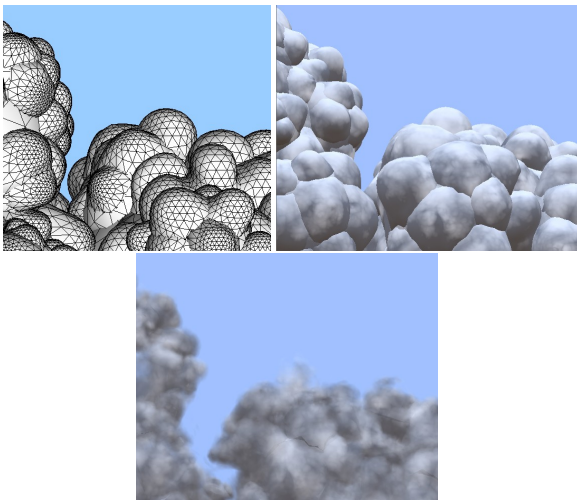


Figure 3: *Top left:* a zoom on the mesh of the generated blobs. *Top right:* with the Perlin texture. *Bottom:* with the Gardner-like shader.

The pictures presented in Figure 2,3 and the teaser show a cloud with a root level made of six hand-placed particles. It is a very small work for the user, but this controls the overall shape of the cloud. Figure 2.top shows this root level. We generated on top of it two extra levels resulting in the picture shown in Figure 2.bottom. The effect of the shaders is shown on Figure 3 and on the teaser.

Parameters have been chosen as follow. The radius of the hand-placed root particles (*i.e.,* level 0) is comprised between 200m and 450m. The radius of the particles for level l is one third of the radius of particles of level $l-1$ (thus mimicking the lacunarity estimated on Figure 1), modulated by a random factor of $\pm$ 30%. The particles were flattened with a factor e_i randomly chosen between 0.1 and 1. The parameter controlling the furrow size ε was set to 0.5, meaning that each blobs is separated at least half of its radius from the others.

We set the amount of blending b as null for levels 0 and 1, and 1.0 for level 2. The parameter I was set to 10.

The reference height h_0 corresponding to the floor of the cloud was 100m, which was approximately the height of the lowest particles of the root level. The stiffness of the repulsion α_h was 100m as well.

6. Conclusion and Future Work

In this paper, we proposed a model to define and build the *shape* of cumulus clouds, resulting in a surface mesh and shaders. Since we do not rely on a volumetric representation contrary to [NND96], the rendering (and therefore the animation) of our model is really fast. It is also more realistic, as the blobs are deformed to mimic the shape of real clouds. Moreover, we do not suffer from voxel discretization.

The realistic *rendering* of this shape taking into account the various global illumination effects is yet to be done. We believe that it is feasible to reproduce most of these effects using shaders and precomputations, which is our main goal as future work. Moreover our model is based on particles and thus should be animatable. We would be interested in studying how to adapt an animation model like [Ney03] to our case. Our long term goal is to obtain a plausible evolving cloud sky with plausible rendering at interactive rate.

References

[CA97] CROSSNO P., ANGEL E.: Isosurface extraction using particle systems. In *IEEE Visualization '97* (Nov. 1997), pp. 495–498. 2

[DKY*00] DOBASHI Y., KANEDA K., YAMASHITA H., OKITA T., NISHITA T.: A simple, efficient method for realistic animation of clouds. In *Proceedings of ACM SIGGRAPH 2000* (July 2000), pp. 19–28. 1

[Ebe97] EBERT D. S.: Volumetric procedural implicit functions: A cloud is born. In *SIGGRAPH 97 Technical Sketches Program* (Aug. 1997), Whitted T., (Ed.), ACM SIGGRAPH, Addison Wesley. ISBN 0-89791-896-7. 1

[ES00] ELINAS P., STÜRZLINGER W.: Real-time rendering of 3D clouds. *Journal of Graphics Tools 5*, 4 (2000), 33–45. 1

[FSJ01] FEDKIW R., STAM J., JENSEN H. W.: Visual simulation of smoke. In *Proceedings of ACM SIGGRAPH 2001* (Aug. 2001), Computer Graphics Proceedings, Annual Conference Series, pp. 15–22. 2

[Gar84] GARDNER G. Y.: Simulation of natural scenes using textured quadric surfaces. In *Computer Graphics (SIGGRAPH '84 Proceedings)* (July 1984), Christiansen H., (Ed.), vol. 18, pp. 11–20. 1

[Gar85] GARDNER G. Y.: Visual simulation of clouds. In *Computer Graphics (SIGGRAPH '85 Proceedings)* (July 1985), Barsky B. A., (Ed.), vol. 19, pp. 297–303. 1, 3

[Gas93] GASCUEL M.-P.: An implicit formulation for precise contact modeling between flexible solids. In *Proceedings of SIGGRAPH 93* (Aug. 1993), pp. 313–320. 2

[HBSL03] HARRIS M. J., BAXTER W. V., SCHEUERMANN T., LASTRA A.: Simulation of cloud dynamics on graphics hardware. In *Graphics Hardware 2003* (July 2003), pp. 92–101. 2

[HL01] HARRIS M. J., LASTRA A.: Real-time cloud rendering. *Computer Graphics Forum 20*, 3 (2001), 76–84. 1

[Kaj85] KAJIYA J. T.: Anisotropic reflection models. In *Computer Graphics (SIGGRAPH '85 Proceedings)* (July 1985), Barsky B. A., (Ed.), vol. 19(3), pp. 15–21. 3

[Mic] MICROSOFT: Microsoft flight simulator 2004. (also presented as a Siggraph Sketch 2003). `http://www.ofb.net/~eggplant/clouds/`. 1

[Ney03] NEYRET F.: Advected textures. *Symposium on Computer Animation'03* (july 2003). 2, 4

[NND96] NISHITA T., NAKAMAE E., DOBASHI Y.: Display of clouds taking into account multiple anisotropic scattering and sky light. In *SIGGRAPH 96 Conference Proceedings* (Aug. 1996), Rushmeier H., (Ed.), ACM SIGGRAPH, Addison Wesley, pp. 379–386. 1, 2, 4

[Per85] PERLIN K.: An image synthesizer. In *Computer Graphics (SIGGRAPH '85 Proceedings)* (July 1985), Barsky B. A., (Ed.), vol. 19(3), pp. 287–296. 1

[SSEH03] SCHPOK J., SIMONS J., EBERT D. S., HANSEN C.: A real-time cloud modeling, rendering, and animation system. *Symposium on Computer Animation'03* (july 2003), 160–166. 1

[WH94] WITKIN A. P., HECKBERT P. S.: Using particles to sample and control implicit surfaces. In *Proceedings of SIGGRAPH 94* (July 1994), Computer Graphics Proceedings, Annual Conference Series, pp. 269–278. 2, 3