Journées de l'Association Francophone d'Informatique Graphique, Bordeaux, 2006

Rendu réaliste de pluie en temps-réel

P. Rousseau & V. Jolivet & D. Ghazanfarpour

Institut XLIM (UMR CNRS n°6172), Université de Limoges

Abstract

Le rendu en temps réel de phénomènes météorologiques virtuels a été étudié dans de nombreux articles. L'ajout de brouillard ou de neige à une scène tridimensionnelle est relativement trivial. La pluie est l'un des phénomènes naturels les plus courants, mais son rendu manque généralement de réalisme.
Dans cet article, nous présentons des résultats antérieurs publiés dans [RJG06], et quelques améliorations, notamment la gestion du vent et des collisions. Pour simuler la réfraction observable à l'intérieur d'une goutte d'eau, la scène virtuelle est rendue dans une texture, qui est ensuite distordue selon les propriétés optiques des gouttes d'eau. Cette texture est plaquée sur chaque goutte. Notre méthode prend également en compte la persistance rétinienne.

Real-time rendering of virtual weather conditions has been investigated in many papers. Inserting fog or snow in a scene is rather straightforward. Rain is one of the most encountered natural phenomena, but its rendering often lacks realism.
In this paper, we present results previously published in [RJG06], and some improvements to this method, including wind and collisions handling. In order to simulate the refraction of the scene inside a raindrop, the scene is captured to a texture which is distorted according to optical properties of raindrops. This texture is mapped onto each raindrop. Our method also takes into account retinal persistence.

1. Introduction

Jusqu'à une époque récente, la rapidité d'exécution d'une application tridimensionnelle temps-réel était en général prioritaire par rapport au réalisme de l'application. Aujourd'hui, les possibilités des cartes graphiques rendent ces deux points de plus en plus compatibles. Les applications temps-réel se cherchent de nouveaux objectifs, pouvant être satisfaits sans sacrifier leurs performances : un rendu photo-réaliste, le respect des lois de la physique, la gestion d'un grand nombre de phénomènes naturels.

Un haut degré de réalisme est requis pour immerger l'utilisateur dans un environnement virtuel visuellement convaincant. C'est dans ce but que les développeurs ajoutent des phénomènes météorologiques à leurs applications. L'utilisation de brouillard réduit la profondeur observable dans une scène tridimensionnelle, et permet d'accélérer le processus de rendu. Elle est courante en informatique graphique depuis de nombreuses années, y compris par le biais d'une accélération graphique complète ([Bir03]). Les flocons de neige peuvent être rendus de manière satisfaisante par des matériaux simples, opaques et diffus. De ce fait, une chute de neige sera représentée réalistement par un système de particules simple ([LZK*04]). Cependant, la pluie manque encore de réalisme dans les applications temps-réel, bien que ce soit certainement le phénomène météorologique le plus courant.

Les méthodes de rendu de pluie se répartissent en deux catégories principales. La plupart des jeux vidéos utilisent des systèmes de particules et des textures statiques, ce qui conduit à un manque certain de réalisme. D'autres méthodes se basent sur des propriétés physiques ([SDY02], [KKY93], [KIY99]) dans le but de simuler le comportement et l'apparence de gouttes d'eau évoluant sur une surface. Elles produisent des résultats satisfaisants, au prix de temps de calculs importants. La technique que nous introduisons ici dispose des avantages de ces deux types de méthodes, sans en avoir les inconvénients.

Cet article décrit une méthode utilisant la carte graphique

pour permettre le rendu réaliste de pluie avec une vitesse d'exécution élevée. Cette méthode est basée sur les propriétés physiques (à la fois géométriques, dynamiques et optiques) des gouttes d'eau. Une image de la scène tridimensionnelle en arrière plan est capturée et stockée dans une texture. Cette texture est plaquée sur les gouttes suivant les lois de l'optique géométrique dans un *pixel shader*. Cette méthode est ensuite étendue pour prendre en compte la persistance rétinienne : des gouttes d'eau quasi-sphériques semblent prendre une forme allongée verticalement. Nous complétons également une méthode de simulation de la dynamique des particules sur la carte graphique, en lui ajoutant la gestion du vent et des collisions.

Après avoir présenté l'état de l'art du domaine et les caractéristiques physiques (géométriques, dynamiques et optiques) des gouttes d'eau, nous décrirons notre méthode permettant le rendu réaliste en temps-réel de gouttes de pluie, puis proposerons une extension pour prendre en charge la persistance rétinienne. Après cela, nous présenterons la technique de simulation dynamique utilisée, qui est une extension de méthodes existantes de simulation de particules sur la carte graphique, raffinée dans le contexte d'animation de gouttes de pluie, et intégrant la gestion des collisions et du vent. Enfin, nous présenterons nos résultats avant de conclure et d'introduire des pistes pour des travaux futurs.

2. Travaux antérieurs

Le rendu de pluie, déjà étudié dans quelques articles, tend à devenir courant dans les jeux vidéos. La plupart des techniques existantes visent selon le cas une vitesse d'exécution élevée, ou une exactitude physique poussée. Cet article cherche à rendre ces deux buts compatibles, en proposant une méthode basée sur les propriétés physiques des gouttes d'eau pour des résultats de qualité (proches de ceux obtenus par lancer de rayons), sans pour autant nécessiter des temps de calculs prohibitifs.

2.1. Rendu de pluie

Rendu temps-réel :

Dans de nombreux jeux vidéos (*Unreal Tournament 2004*, *Need For Speed Underground 2*, ...), la pluie est représentée par un système de particules basique, où les particules utilisent une texture blanche translucide. Cette méthode n'est pas très réaliste, mais permet à l'utilisateur d'avoir le sentiment général qu'il se trouve dans un environnement pluvieux.

Un travail intéressant dans le domaine des jeux vidéos a été mené par N. Wang et B. Wade pour *Microsoft Flight Simulator 2004* [WW04]. Une texture adaptée aux conditions souhaitées défile sur un cône englobant l'utilisateur. Cette méthode est plus rapide que les systèmes de particules, mais ne permet pas d'interactions entre la pluie et son environnement. De plus, une texture doit être créée pour chaque type de précipitation souhaité. La méthode que nous introduisons ici cherche à amener plus de réalisme et de flexibilité sans pour autant augmenter le coût.

Récemment, Tatarchuk ([TI06], [Tat06]) a proposé une technique de composition de couches de pluie en post traitement, utilisées dans la vidéo *Toy Shop Demo* de *ATI Research*. Enfin, nous avons introduit les bases de notre méthode dans [RJG06].

Méthodes basées sur les lois physiques :

[SDY02], [KKY93], [KIY99] ont introduit des méthodes permettant le rendu d'un nombre limité de gouttes évoluant à faible vitesse sur une surface (un pare-brise par exemple), par le biais de la génération dynamique d'une carte d'environnement cubique. Ces méthodes génèrent des résultats satisfaisants, mais impliquent des temps de calcul élevés (partiellement à cause d'un processus de simulation complexe). Notre technique réduit la génération de la carte d'environnement à une seule capture, suffisante pour simuler le phénomène de réfraction au travers des gouttes.

Plus récemment, [WMT05] a proposé une méthode pour animer des gouttes d'eau sur des surfaces quelconques, autorisant les gouttes à se scinder ou fusionner entre elles. Cette technique génère des résultats de très grande qualité, au prix d'un temps de calcul prohibitif.

[GN06] a introduit dernièrement une méthode utilisant une base de données conséquente (400 Mo) pour illuminer réalistement des gouttes étirées ("cordes").

Méthodes issues de la vision par ordinateur :

Dans le domaine de la vision par ordinateur, [Ros00] a étudié la forme et les propriétés optiques des gouttes d'eau, pour proposer une méthode de télédétection. [SW03], [GN03] et [GN04] ont décrit des techniques pour ajouter et retirer de la pluie à des vidéos. Pour valider l'approche proposée, [GN03] a besoin d'un modèle théorique précis pour comprendre l'influence de la pluie dans une vidéo, et décrit dans ce but une méthode de lancer de rayons générant des gouttes avec une précision élevée, et des temps de calcul dissuasifs.

Autres méthodes :

D'autres articles ne peuvent être rattachés à l'une ou l'autre des catégories présentées ci-dessus. Langer *et al.* [LZK*04] ont présenté une méthode de synthèse spectrale basée-image pour rendre de la pluie ou de la neige en temps interactif.

D'autres travaux, proposés par Yang *et al.* [YZZ04], ont introduit une méthode simple de distorsion d'une image de la scène observée, visant à donner l'impression de gouttes sur un pare-brise. Cette méthode utilise un algorithme phénoménologique au très faible coût, non relié aux

propriétés physiques de gouttes d'eau. Les résultats générés manquent de réalisme. La technique que nous proposons trouve sa source dans les propriétés physiques de gouttes d'eau, amenant ainsi un degré de réalisme plus important.

3. Propriétés physiques des gouttes d'eau

3.1. Forme, taille, et dynamique

L'idée largement répandue selon laquelle les gouttes de pluie ont une forme lacrymale ou allongée ("cordes") est inexacte. Cette impression est causée, comme nous le verrons dans la section 4.2, par le phénomène de persistance rétinienne. [Gre75], [BC87], [CB90] entre autres établissent que les gouttes en chute libre adoptent plutôt une forme ellipsoïdale, équilibre entre la tension de surface (qui cherche à minimiser la surface de contact entre eau et air), et la pression aérodynamique (qui tend à étirer la goutte horizontalement). Les gouttes de faible diamètre sont quasi sphériques, tandis que les gouttes plus larges arborent une base aplatie.

Beard et Chuang ([BC87], [CB90]) ont présenté un modèle basé sur une somme de cosinus pondérés, qui déforme une sphère suivant l'équation suivante :

$$r(\theta) = a\left(1 + \sum_{n=0}^{10} C_n cos(n\theta)\right) \quad (1)$$

où a est le rayon de la sphère avant distorsion, pris au centre de masse de la goutte. L'angle θ dénote l'élévation polaire ($\theta = 0^\circ$ étant la verticale descendante). A titre indicatif, quelques un des coefficients C_n sont indiqués dans la table 1.

	Coefficients ($c_n \cdot 10^4$) pour n =					
a (mm)	0	1	2	3	4	5
0.5	-28	-30	-83	-22	-3	2
1.0	-134	-118	-385	-100	-5	17
3.0	-843	-472	-2040	-240	299	168
4.5	-1328	-403	-2889	-106	662	153
a (mm)	6	7	8	9	10	
0.5	1	0	0	0	0	
1.0	6	-1	-3	-1	1	
3.0	-21	-73	-20	25	24	
4.5	-146	-111	18	81	31	

Table 1: *Coefficients C_n (équation 1) [CB90].*

La figure 1 illustre quelques formes de gouttes pour des rayons courants, calculées d'après l'équation 1.

3.2. Propriétés optiques

Il est physiquement correct de négliger les propriétés ondulatoires de la lumière dans le cas de gouttes de diamètre bien supérieur à la longueur d'onde de la lumière les traversant, ce qui est le cas dans notre étude. Nous pouvons donc nous concentrer sur les propriétés définies par les lois dites de l'optique géométrique qui considèrent la lumière comme un

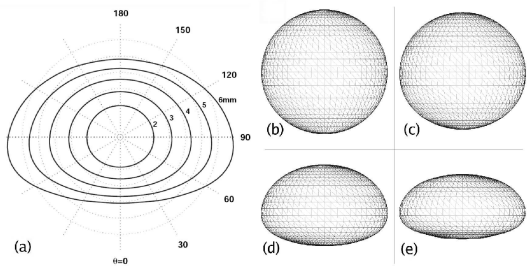

Figure 1: *Forme des gouttes. (a) Forme comparée de gouttes de rayons R = 1mm, 1.5mm, 2mm, 2.5mm et 3mm [Ros00]. (b) Forme d'une goutte de rayon R = 0.5mm. (c) R = 1.0mm. (d) R = 3.0mm. (e) R = 4.5mm.*

ensemble de rayons monochromatiques, qui se réfractent et se réfléchissent aux interfaces séparant les différents milieux de propagation.

À une interface, la loi de la réflexion décrit les directions du rayon réfléchi, et la loi de Snell-Descartes indique la direction du rayon réfracté. Pour un rayon donné, le *facteur de Fresnel* indique la proportion qui sera réfléchie ou réfractée, suivant l'angle d'incidence et la polarisation du rayon entrant. Plus de détails sur ces phénomènes peuvent être lus dans [Gla94].

Figure 2: *Photographie d'une goutte réelle réfractant la scène en arrière plan.*

Dans la figure 2, un exemple de réfraction au travers d'une goutte peut être observé. Cette photographie a été prise avec une haute vitesse d'obturateur (1/1000 s). Les points blancs sont dus au flash de l'appareil. Au moment de la photo, la goutte venait de se détacher du robinet, et n'a pas encore atteint sa forme stable.

4. Rendu en temps réel de gouttes d'eau

4.1. Description de la méthode

Le calcul du facteur de Fresnel démontre que la réflexion n'a une participation significative à l'apparence de la goutte que pour les angles d'observation rasants. Compte tenu de la

taille des gouttes dans notre application, la contribution de la réflexion sera inférieure à 10% pour tous les pixels sauf ceux situés sur le contour externe de la goutte. Il est donc raisonnable de négliger la participation de la réflexion dans l'apparence de la goutte, pour mieux se focaliser sur l'acuité de la réfraction.

Les gouttes sont rendues sous formes d'imposteurs [MS95], (quadrilatères faisant toujours face à la caméra) ; la forme externe de la goutte est donnée par un masque précalculé à partir de l'équation 1 pour le rayon désiré.

L'image que l'on perçoit au travers d'une goutte d'eau est une déformation de la scène en arrière plan, déformée et retournée, comme l'illustre la figure 2. Pour simuler cet effet, nous utilisons la fonctionnalité *rendu vers texture* de la carte graphique pour obtenir une texture qui sera plaquée sur chaque goutte dans un *pixel shader*.

Dans une phase de pré-calcul, un masque est généré pour le rayon désiré, puis stocké dans une texture. À l'exécution du programme, pour chaque image rendue, la scène est capturée vers une texture grand-angle. L'apparence de chaque pixel d'une goutte est déterminée de la manière suivante :

- À l'aide du masque, on détermine si le pixel est à l'intérieur ou à l'extérieur de la goutte.
- S'il est à l'intérieur, on utilise le masque pour déterminer la direction du vecteur réfracté.
- On cherche ensuite la position dans la texture capturée qui serait vue au travers de la goutte si elle ne déviait pas le rayon entrant.
- Dans l'espace image, on ajoute le vecteur de réfraction à cette position.
- Enfin, on extrait le pixel de cette nouvelle position, qui est celui qui sera vu au travers de la goutte.

Un programme annexe utilise l'équation 1 pour calculer en trois dimensions la forme d'une goutte de rayon fourni en paramètre. Pour chaque pixel de cette forme, le vecteur de réfraction est calculé et stocké dans une texture de résolution 512x512. Dans le *pixel shader*, le masque est utilisé simultanément pour donner sa forme à la goutte, et pour déterminer la direction de réfraction.

Une caméra est positionnée à l'emplacement de l'observateur, avec la même direction de visée, et un angle de vue très large (165 °, couvrant la déviation maximale des rayons dans une goutte). Elle capture une texture grand-angle de la scène (de résolution 512x512).

Pour chaque pixel P_i de la goutte, un *pixel shader* extrait de la texture capturée le pixel réfracté vers l'observateur au travers de la goutte (Figure 3). Le *pixel shader* localise dans un premier temps le pixel P_o dans la texture capturée qui est l'image de l'objet dans la scène vu par l'observateur suivant la direction de visée de P_i. Le vecteur de réfraction est ensuite extrait du masque, et combiné avec la position de P_o pour identifier le pixel P_c. C'est ce pixel qui donne la couleur de P_i, l'image réfractée par la goutte dans cette direction de visée.

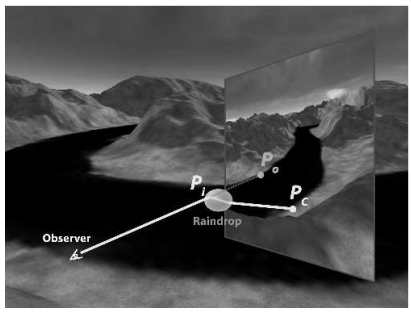

Figure 3: *Extraction du pixel à partir de la texture capturée.Les rayons émanant de l'observateur sont réfractés par les gouttes vers la texture capturée.*

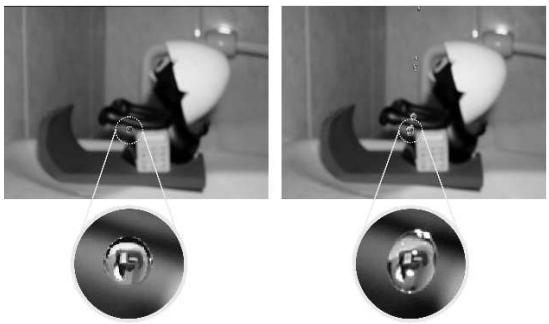

Figure 4: *Gauche : une image simulée par notre méthode. Droite : photographie d'une vraie goutte.*

La figure 4 propose une comparaison entre une goutte simulée grâce à notre méthode (à gauche) et une vraie goutte (droite). Plus de détails sur cette méthode peuvent être lus dans [RJG06].

4.2. Gestion de la persistance rétinienne

Nous avons défini un modèle général pour la simulation de pluie, qui ne prend pas en compte la perception de l'observateur. Du fait de la persistance rétinienne ou du temps d'exposition de la pellicule, un œil humain ou une caméra perçoivent généralement les gouttes de pluie comme des "cordes", allongées verticalement.

L'œil humain n'est pas habitué à observer des gouttes sphériques. Notre modèle, bien que basé sur des propriétés physiques, semble ainsi manquer de réalisme. Nous avons donc étendu ce modèle afin qu'il prenne en compte la persistance rétinienne, et permette le rendu de "cordes" en utilisant notre modèle physique de goutte.

Au cours de sa chute, une goutte d'eau subit des déformations périodiques, et deux mouvements oscillatoires se

combinent suivant des équations connues (voir [GN06] pour plus de détails). En complément de la technique introduite dans [RJG06], nous avons utilisé ces équations pour découper dynamiquement la forme de nos particules dans le *pixel shader* chargé du rendu. À partir du rayon de la goutte à l'état stable et du temps écoulé, ce dernier calcule la déformation du contour visible de la goutte (à gauche et à droite) et recale la goutte en mouvement dans ce contour.

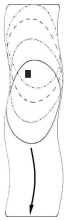

Figure 5: *Chaque pixel reçoit la contribution de plusieurs positions successives de la goutte.*

Pour simuler la persistance rétinienne, nous avons tout d'abord modifié la forme des particules rendues, que nous avons allongées verticalement. Chaque pixel à l'intérieur d'une corde ondulante reçoit la contribution des positions successives de la goutte, comme illustré sur la figure 5. Nous avons donc modifié le *pixel shader* utilisé de manière à ce que pour chaque pixel :

- Il détermine la position d'oscillation courante de la goutte, et établisse si le pixel à traiter appartient à la corde.
- Il calcule le pixel réfracté pour plusieurs positions successives de la goutte,
- Il calcule la moyenne de ces valeurs,
- Il diminue l'opacité de la particule, chaque corde étant obtenue par le déplacement d'une seule et unique goutte.

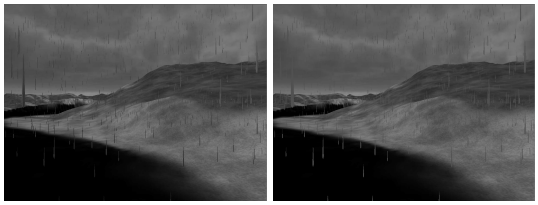

Figure 6: *Gauche : avec notre méthode. Droite : en utilisant une couleur fixe pour les cordes.*

La figure 6 souligne l'intérêt de cette extension, par rapport à des cordes de couleur fixe. L'image de gauche utilise notre méthode, tandis que l'image de droite a été générée en utilisant la même couleur pour toutes les particules. Cette couleur a été choisie de sorte à ce que les différences entre les deux images dans le coin haut droit soient minimales. On peut clairement observer des différences entre les images dans le coin bas gauche, qui justifient le léger sur-coût en temps de calcul lié à l'emploi de l'extension proposée.

5. Simulation dynamique des gouttes sur la carte graphique

5.1. Description

L'animation de pluie est de toute évidence un cas d'école pour les systèmes de particules. Dans notre application, il apparaît que 4000 gouttes au minimum sont requises pour obtenir une impression réaliste de pluie. La gestion du système de particule sur le processeur central de l'ordinateur implique le transfert de toutes les particules vers la carte graphique pour chaque image rendue. Ce transfert provoque un goulet d'étranglement, et réduit drastiquement les performances dès lors que l'on augmente le nombre de particules.

Pour outrepasser cette limite, nous avons utilisé l'approche décrite dans [KLRS04] et [KSW04]. Ces articles proposent des implémentations de systèmes de particules basées sur les possibilités des cartes graphiques actuelles. Les positions des particules sont stockées dans des textures de flottants, actualisées par l'intermédiaire d'un *pixel shader*. Cette technique permet d'éviter la quasi totalité des transferts entre la mémoire centrale et la carte graphique, toute la simulation dynamique se déroulant sur la carte graphique.

5.2. Implémentation

Notre implémentation de cette fonctionnalité diffère légèrement de celles décrites dans [KLRS04] et [KSW04]. Nous avons ciblé notre travail sur la simulation de pluie, et n'avons donc pas utilisé un système aussi général que celui proposé dans ces articles. Contrairement aux autres articles qui se contentent de le mentionner, nous avons fait usage de la fonctionnalité appelée *Vertex Texture Fetch* disponible avec les cartes graphiques compatibles avec le *Shader Model 3.0*. Cette fonctionnalité permet d'accéder à une texture à partir d'un *vertex shader*.

Nous considérons que les particules évoluent dans une "boîte à gouttes" qui se déplace avec l'utilisateur. Un positionnement judicieux de cette boîte permet à la plupart des gouttes d'être à tout instant dans le champ de vision de l'observateur, ce qui maximise l'efficacité du système. Dans notre implémentation, une "boîte à gouttes" typique s'étend sur 100m sur chaque axe.

Les positions des gouttes sont stockées dans une texture de flottants (dont la taille dépend du nombre de particules désiré). Chaque pixel de la texture indique la position d'une particule, exprimée dans un repère lié à la position et aux dimensions de la "boîte à gouttes".

Le processus de simulation se déroule intégralement dans un *pixel shader*, et peut être résumé sous la forme suivante :

- Au lancement de l'application, stockage des particules nécessaires dans la mémoire de la carte graphique.
- Mise à jour de la texture de position grâce à un *pixel shader*.

- Pour chaque particule, modification de sa position dans le *vertex shader*, en utilisant la texture de position.
- Rendu de chaque particule, en utilisant la technique de rendu décrite dans les sections précédentes.

Dans cette texture, les composantes rouge, verte et bleue de chacun des pixels correspondent aux coordonnées X, Y et Z d'une particule.

Lors de la mise à jour de cette texture, nous ajoutons simplement le déplacement effectué depuis l'image précédente (détaillé en partie 5.3) aux coordonnées de chaque pixel. Si la position résultante de cette addition est en dehors de la boîte à gouttes, on replace la particule du côté opposé à celui par lequel elle est sortie. Ainsi, lorsqu'une particule touche le bas de la boîte, elle est immédiatement replacée au sommet de celle ci, en modifiant légèrement ses coordonnées afin d'éviter un mouvement trop répétitif.

Le placement d'une particule doit prendre en compte son orientation par rapport à l'observateur. La texture de position ne comprend qu'un ensemble de coordonnées par particule, les coordonnées de son centre. Une consistance est donnée aux particules dans le *vertex shader*, suivant la nature des particules. Dans le cas des gouttes quasi-sphériques, une fois ce centre converti dans le système de coordonnées écran, la goutte est "étalée" sur l'écran ; chaque coin de la particule est décalé de sorte à ce que la particule soit en permanence dans le plan orthogonal à la direction de visée de l'observateur. Ainsi, les coordonnées du coin haut-gauche seront diminuées sur l'axe X et augmentées sur l'axe Y, d'une quantité dépendant de la taille souhaitée des gouttes.

Dans le cas des "cordes", les gouttes doivent sembler suivre un mouvement vertical. Les particules sont donc étirées verticalement avant leur transformation en coordonnées écran, et ensuite seulement étirées horizontalement. Elles restent donc dans un plan orthogonal à la direction de visée horizontale de l'observateur.

5.3. Gestion du vent et des collisions

Dans notre précédent article ([RJG06]), le déplacement effectué par une goutte entre deux images rendues était simplement basé sur le temps écoulé et une direction de chute commune à l'ensemble des particules (la vitesse étant de l'ordre de quelques mètres par secondes, selon la taille des particules). Nos travaux actuels ont pour objet de compléter cette technique en lui ajoutant la possibilité d'intégrer un vent variant dans le temps et l'espace, ainsi qu'une gestion des collisions.

Pour cela, nous avons créé une sœur jumelle à la texture de position, chargée de gérer la vélocité de chaque particule. La texture de position est mise à jour pour chaque image suivant les directions de chaque particule, lues dans la texture de vélocité. Cette dernière est elle même actualisée à partir d'informations tirées d'une texture chargée de décrire les collisions, et une autre chargée d'indiquer les forces et directions du vent en tout point de la boîte à gouttes.

Les gouttes suivent un mouvement à forte dominante verticale. Suivant cette considération, il nous est apparu que la plupart des collisions se produiraient avec des plans inclinés (toits par exemple) ou horizontaux (sol). L'approche que nous avons adoptée consiste donc à utiliser une carte de hauteur dynamique, générée par le biais d'une caméra surplombant la boîte à gouttes, et enregistrant la hauteur de chaque objet dans la composante alpha d'une texture, tandis que la normale à cet objet est stockée dans les composantes rouge, verte et bleue d'une texture. Grâce à cette texture de collisions, il devient simple de savoir lors de la mise à jour de la texture de vélocité si une particule entre en contact avec un objet. Lorsque cela ce produit, on utilise la normale de l'objet pour établir la direction dans laquelle la particule doit rebondir. Le rebond en lui même est géré par le *pixel shader* chargé de la mise à jour de la texture de vélocité. Dans le cas enfin où une particule entre en collision avec un objet bien en dessous de sa hauteur, nous considérons être dans le cas de l'absorption d'une goutte (par un mur, par exemple), et la réinitialisons (retour au sommet de la boîte).

Outre la texture de collisions, nous avons également défini une texture 3D chargée de représenter le vent présent en tout point de la boîte à gouttes. Chaque cellule de cette texture contient la direction du vent à un endroit donné. Pour éviter les sauts d'une case à l'autre qui rendraient l'animation des particules saccadée, nous utilisons une interpolation trilinéaire entre les cellules. L'avantage d'utiliser une texture pour la gestion du vent, est qu'elle peut être mise à jour par l'intermédiaire d'un *pixel shader*. Techniquement, le matériel actuel impose toutefois de travailler avec des textures bidimensionnelles, c'est pourquoi nous avons utilisé l'approche des textures 3D "plates" présentée dans [HBSL03]. Nous avons ainsi pu aisément faire évoluer nos gouttes dans une tornade animée entre autre type de flux.

Ces fonctionnalités supplémentaires donnent un comportement plus réaliste aux particules par rapport au précédent article, puisqu'elle ne partagent désormais pas toutes la même direction ni la même vitesse, et que l'on peut les voir rebondir sur les objets ou le sol de la scène. Lorsqu'une goutte entame un rebond, nous considérons qu'elle évolue à une vitesse suffisamment réduite pour ne pas être sujette à la persistance rétinienne, et effectuons donc le rendu grâce à l'algorithme décrit pour les gouttes quasi sphériques.

6. Résultats

Il ressort de nos tests que 4000 particules suffisent à offrir une impression de pluie réaliste en utilisant des gouttes larges ou des cordes. 10000 gouttes sont requises dans le cas de gouttes fines. Sur un ordinateur équipé d'un processeur AMD 3200+ et d'une carte graphique geforce 7800GT, notre méthode dépasse les 100 images par secondes pour une

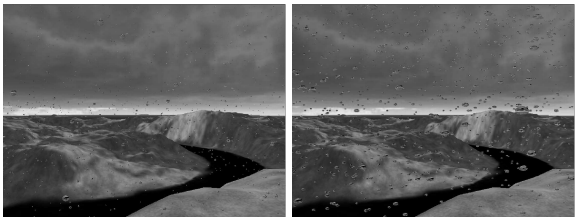

Figure 7: *De haut en bas : 16000 petites gouttes (R=1.0mm) ; 16000 gouttes moyennes (R=3.0mm).*

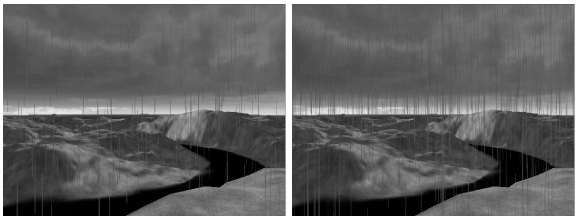

Figure 8: *Avec persistance rétinienne. Haut: 4000 cordes (R=1.0mm) ; bas : 16000 cordes (R=0.5mm).*

scène typique de 5000 particules. La technique de rendu de cordes utilise 5 positions de gouttes par pixel.

La figure 7 montre une scène pluvieuse typique rendue avec différentes tailles de particules. La figure 8 montre les résultats obtenus avec l'extension de persistance rétinienne, avec 4000 gouttes de rayon 1.0mm. Cette extension induit un sur-coût sur le temps d'exécution (dépendant du nombre de positions par gouttes utilisé), mais nécessite moins de particules pour obtenir un effet réaliste.

La pluie étant un phénomène animé, elle s'observe mieux sur des vidéos que sur des images statiques. Des vidéos générées grâce à notre méthode sont donc accessibles sur la page *http://msi.unilim.fr/∼rousseau/pluie.html*.

7. Conclusion et travaux futurs

Nous avons développé un modèle temps-réel basé sur des propriétés physiques pour le rendu de gouttes d'eau, auquel nous avons intégré la gestion de la persistance rétinienne. Notre modèle permet d'obtenir des résultats de meilleure qualité que ceux souvent utilisés dans les jeux vidéos, utilisant des textures fixes pour les gouttes. Notre modèle atteint également de bien meilleures performances que les modèles physiques existants, puisque notre méthode permet de maintenir une vitesse d'éxécution élevée sans pour autant sacrifier le réalisme visuel.

Nous sommes convaincus que notre technique pourrait être largement employée dans le domaine des jeux vidéos, du fait qu'elle permet l'obtention de résultats très satisfaisants sans trop compromettre les performances. Elle pourrait également être employée comme post-traitement, afin d'ajouter de la pluie à une image ou une vidéo existante (sous réserve que l'on dispose d'une vue grand angle de l'image à traiter).

Pour des résultats parfaitement exacts, les deux techniques employables sont un processus complexe de lancer de rayons, ou la génération d'une carte d'environnement pour chaque goutte. Aucune de ces méthodes ne peut être exécutée en temps-réel, du moins avec le matériel actuel. Notre modèle introduit des approximations à ses méthodes. Il n'est donc pas parfaitement conforme aux lois physiques, mais permet un rendu visuellement réaliste à haute vitesse d'exécution.

Lors de futurs travaux, nous poursuivrons notre étude de la simulation dynamique des gouttes sur la carte graphique. Nous ajouterons également la gestion de la réflexion à notre modèle afin d'obtenir des gouttes encore plus réalistes en vue de près. Enfin, nous envisageons d'étudier un modèle alternatif plus simple qui serait utilisé pour les gouttes éloignées de l'observateur, dont la taille à l'écran passe sous le seuil du pixel.

Remerciements

Ces travaux ont été partiellement financés par le projet GameTools de la Communauté Européenne (contrat numéro 004363).

References

[BC87] BEARD K. V., CHUANG C.: A new model for the equilibrium shape of raindrops. *J. Atmos. Sci. 44(11)* (1987), 1509–1524.

[Bir03] BIRI V.: *Techniques d'animation dans les méthodes globales d'illumination*. PhD thesis, Université de Marne La Vallée, 2003.

[CB90] CHUANG C., BEARD K.: A numerical model for the equilibrium shape of electrified raindrops. *J. Atmos. Sci. 47(11)* (1990), 1374–1389.

[Gla94] GLASSNER A. S.: *Principles of Digital Image Synthesis*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1994.

[GN03] GARG K., NAYAR S. K.: *Photometric Model of a Rain Drop*. Tech. rep., Columbia University, 2003.

[GN04] GARG K., NAYAR S. K.: Detection and removal of rain from videos. In *2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2004), with CD-ROM, 27 June - 2 July 2004, Washington, DC, USA* (2004), vol. 1, pp. 528–535.

[GN06] GARG K., NAYAR S.: Photorealistic Rendering of Rain Streaks. *ACM Trans. on Graphics (also Proc. of ACM SIGGRAPH)* (Jul 2006).

[Gre75] GREEN A. W.: An approximation for the shapes of large raindrops. *J. Appl. Meteor. 21* (1975), 1578–1583.

[HBSL03] HARRIS M. J., BAXTER W. V., SCHEUERMANN T., LASTRA A.: Simulation of cloud dynamics on graphics hardware. In *HWWS '03: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware* (Aire-la-Ville, Switzerland, Switzerland, 2003), Eurographics Association, pp. 92–101.

[KIY99] KANEDA K., IKEDA S., YAMASHITA H.: Animation of water droplets moving down a surface. *Journal of Visualization and Computer Animation 10*, 1 (1999), 15–26.

[KKY93] KANEDA K., KAGAWA T., YAMASHITA H.: Animation of water droplets on a glass plate. In *Computer Animation* (1993), pp. 177–189.

[KLRS04] KOLB A., LATTA L., REZK-SALAMA C.: Hardware-based simulation and collision detection for large particle systems. In *HWWS '04: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware* (New York, NY, USA, 2004), ACM Press, pp. 123–131.

[KSW04] KIPFER P., SEGAL M., WESTERMANN R.: Uberflow: A GPU-based particle engine. In *Eurographics Symposium Proceedings Graphics Hardware 2004* (2004), pp. 115–122.

[LZK*04] LANGER M. S., ZHANG L., KLEIN A. W., BHATIA A., PEREIRA J., REKHI D.: A spectral-particle hybrid method for rendering falling snow. In *Rendering Techniques 2004 (Eurographics Symposium on Rendering)* (june 2004), ACM Press.

[MS95] MACIEL P. W. C., SHIRLEY P.: Visual navigation of large environments using textured clusters. In *Symposium on Interactive 3D Graphics* (1995), pp. 95–102, 211.

[RJG06] ROUSSEAU P., JOLIVET V., GHAZANFARPOUR D.: Realistic real-time rain rendering. *Computers & Graphics 30*, 4 (2006), 507–518. special issue on Natural Phenomena Simulation.

[Ros00] ROSS O. N.: *Optical remote sensing of rainfall micro-structures*. Master's thesis, Freie Universität Berlin, 2000. in partnership with University of Auckland.

[SDY02] SATO T., DOBASHI Y., YAMAMOTO T.: A method for real-time rendering of water droplets taking into account interactive depth of field effects. In *Entertainment Computing: Technologies and Applications, IFIP First International Workshop on Entertainment Computing (IWEC 2002)* (2002), vol. 240 of *IFIP Conference Proceedings*, Kluwer, pp. 125–132.

[SW03] STARIK S., WERMAN M.: Simulation of rain in videos. In *3rd international workshop on texture analysis and synthesis (Texture03)* (Nice, France, 2003), pp. 95–100.

[Tat06] TATARCHUK N.: Artist-directable real-time rain rendering in city environments. In *SI3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games Poster* (New York, NY, USA, 2006), ACM Press, p. 30.

[TI06] TATARCHUK N., ISIDORO J.: Artist-directable real-time rain rendering in city environments. In *Eurographics Workshop on Natural Phenomena* (New York, NY, USA, 2006), ACM Press.

[WMT05] WANG H., MUCHA P. J., TURK G.: Water drops on surfaces. *ACM Trans. Graph. 24*, 3 (2005), 921–929.

[WW04] WANG N., WADE B.: Rendering falling rain and snow. In *ACM SIGGRAPH 2004 Technical Sketches Program* (2004).

[YZZ04] YANG Y., ZHU C., ZHANG H.: Real-time simulation: Water droplets on glass windows. *Computing in Science and Eng. 6*, 4 (2004), 69–73.