Réflexions spéculaires en temps réel sur des surfaces lisses

David Roger, Nicolas Holzschuch

Réflexions spéculaires en temps réel sur des surfaces lisses

D. Roger, N. Holzschuch

Équipe ARTIS / Laboratoire GRAVIR

{david.roger,nicolas.holzschuch}@imag.fr

Résumé : *Le calcul des réflexions spéculaires est nécessaire pour atteindre un haut degré de réalisme dans une image de synthèse. De nombreuses méthodes permettent de le faire de façon approximative en temps réel, notamment par l'utilisation de textures type* environment map *calculées à partir du réflecteur. Nous proposons ici une approche différente qui calcule le reflet de chaque sommet de la scène, puis utilise le* pipeline *graphique classique pour interpoler entre les sommets. La méthode proposée cherche un extremum de la distance parcourue par la lumière pour calculer le reflet de chaque sommet de la scène, selon le principe de Fermat. Le matériel graphique réalise ensuite l'interpolation entre les reflets des sommets.*

Mots-clés : Réflexions spéculaires, rendu réaliste, temps réel

1 Introduction

Une branche de l'informatique graphique s'intéresse à la représentation en deux dimensions de scènes 3D. Ceci peut être fait de plusieurs façons, par exemple en mettant en valeurs certaines informations ou bien en tentant de générer une image semblable à une photographie de la scène. Ainsi, on se place dans ce dernier cas et on souhaite faire un du rendu photo-réaliste de scènes 3D : on cherche alors à représenter de façon réaliste une scène du point de vue d'un observateur virtuel.

Les reflets spéculaires sont causés par le rebond des rayons lumineux suivant la loi de Descartes. Ils contribuent fortement au réalisme de la scène car ils permettent de percevoir matériaux des objets et donnent des informations sur toute la scène (par exemple il se peut que certains objets de la scène ne soient visibles que par l'intermédiaire de leur reflet sur un autre objet). Les reflets spéculaires apparaissent par exemple sur des miroirs, des objets métalliques (comme une carrosserie de voiture par exemple) ou en verre. Ce sont des effets très complexes car globaux (le reflet sur un objet dépend d'un grand nombre d'objets de la scène) et dépendants du point de vue.

Il existe de nombreuses méthodes qui parviennent à représenter les reflets spéculaires, mais au prix de calculs longs pouvant atteindre plusieurs heures par image. Elles sont trop coûteuses pour leur permettre un calcul en temps réel, ce qui leur interdit certaines applications comme le jeu vidéo ou la visualisation interactive, car on a alors besoin de calculer les images au fur et a mesure de leur affichage. On propose ici un algorithme permettant le rendu en temps réel des reflets spéculaires au prix d'approximations.

Le plan de cet article est le suivant : la section 2 présente l'état de l'art et confronte notamment les deux approches principales existantes pour le calcul de réflexions en temps réel. L'algorithme est ensuite présenté en section 3. La section 4 présente les résultats obtenus par cette méthode, et enfin la section 5 conclut et donne les pistes envisagées pour la poursuite des travaux.

2 État de l'art

La méthode du lancer de rayons est la méthode de référence pour le rendu des réflexions spéculaires, et permet d'obtenir des images photo-réalistes de grande qualité. Cependant, elle est très coûteuse et ne peut pas être effectuée en temps réel sur un ordinateur classique : les temps de calcul s'étendent de plusieurs secondes à plusieurs heures par image selon la méthode utilisée.

De nombreuses méthodes approximatives ont donc été mises au point pour obtenir un rendu en temps réel. On distingue deux types d'approches qui coexistent, toutes deux tirant partie des possibilités offertes par le matériel graphique : l'approche par pixel, utilisant une texture (*environment map*), et l'approche par sommets.

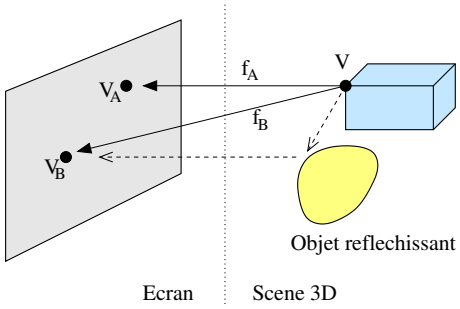

FIG. 1 – Approche par sommets : on utilise le *pipeline* graphique classique en remplaçant la fonction de projection sur l'écran f_A par une nouvelle fonction f_B

2.1 Approche par pixel

Les approches par pixel font suite à la méthode de Blinn et Newell [BN76]. Cette méthode consiste à créer une image sphérique de la scène en plaçant le point de vue au centre du réflecteur, et à la stocker dans une texture appelée *environment map*. Ensuite le réflecteur est rendu et en chaque pixel, la texture est consultée en fonction de la direction du rayon réfléchi.

Cette technique, très rapide, constitue une approximation : elle n'est exacte que lorsque les objets réfléchis sont à une distance infinie du réflecteur. Elle ne permet pas de rendre les effets de parallaxe ni des matériaux dont l'aspect dépend du point de vue sur les objets réfléchis. Enfin cette méthode est limitée aux uniquement les objets visibles depuis le centre du réflecteur.

Des améliorations apportées par Patow [Pat95] et Szirmay-Kalos *et al.* [SKALP05] consistent à stocker la profondeur des objets rendus dans la texture et permettent d'améliorer la qualité des résultats pour des objets situés à distance finie du réflecteur. Certains objets non visibles depuis le centre du réflecteur créent alors des trous dans l'image résultat.

Ces méthodes peuvent également présenter de problèmes d'aliassage dus à la résolution de la texture.

2.2 Approche par sommets

L'approche par sommets consiste à calculer uniquement le reflet des sommets de la scène et à utiliser le matériel graphique pour interpoler automatiquement les triangles. Ce genre de méthodes tire profit du pipeline graphique classique, en modifiant simplement la façon dont les sommets sont projetés : au lieu de les projeter sur l'écran par une multiplication matricielle f_A, on rajoute une transformation qui commence par les projeter sur le réflecteur pour obtenir la projection f_B (voir figure 1). Dans le cas où le réflecteur est plan, la transformation f_B est linéaire, on peut donc l'implémenter dans le *pipeline* graphique et obtenir des résultats exacts de manière très efficace [Rey96]. Cependant, si le réflecteur n'est pas plan, f_B n'est pas linéaire, et la difficulté de l'approche par sommet est le calcul de cette transformation.

Ces méthodes permettent de lever certaines limitations de l'approche par pixel : pas de restriction à des objets à l'infini, pas de problèmes de trous dans les images, pas d'aliassage, possibilité de rendre des matériaux dont l'aspect dépend du point de vue.

Les approches par sommets ont également leurs faiblesses :

– elles associent à chaque sommet de la scène un reflet unique sur le réflecteur ; cependant un point peut avoir un nombre fini de reflets (dans ce cas il faut les rendre en plusieurs fois, par exemple en subdivisant le réflecteur), ou même une infinité de reflets (cas extrême : si l'œil est au foyer d'un ellipsoïde, un objet situé à l'autre foyer se reflète sur toute la surface de l'ellipsoïde) et on ne pourra pas les représenter par ces méthodes.
– même si les reflets des sommets sont calculés de manière exacte, l'interpolation est effectuée de manière linéaire entre ces sommets : en conséquence le reflet d'un triangle sera toujours un triangle. Ceci est une approximation lorsque le réflecteur est courbé (le reflet d'un triangle doit alors avoir des arêtes courbes). Ce défaut peut être atténué en tesselant la scène finement.

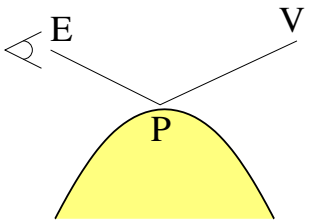

FIG. 2 – Principe de Fermat : on cherche P tels que $d_1 + d_2$ soit extrêmale

Différentes méthodes sont basées sur cette approche. Mitchell et Hanrahan [MH92] l'ont utilisé comme une alternative au lancer de rayons, notamment pour le calcul de caustiques. Leur méthode n'est pas temps réel.

Une technique consiste à lancer quelques rayons par une méthode classique puis les utiliser comme base pour calculer le reflet de chaque sommet. Ainsi, Chen et Arvo [CA00b, CA00a], en utilisant une équation implicite du réflecteur, cherchent le rayon le plus proche du sommet à réfléchir et le perturbent afin de trouver le point exact du reflet. Martin et Popescu [MP04] stockent dans un octree la réflexion d'un très grand nombre de rayons pour chaque position échantillonnée de la caméra. Au moment du rendu, pour chacune des 8 positions voisines de la caméra, ils cherchent le rayon qui passe le plus près du sommet à réfléchir. Ils obtiennent ainsi 8 positions sur le réflecteur qui sont ensuite interpolées tri-linéairement. Cette méthode produit de beaux résultats mais reste assez lente et demande un pré-calcul lourd. De plus, le réflecteur ne peut pas bouger.

Ofek et Rappoport [OR98, Ofe98] utilisent quant à eux une structure d'accélération appelée *explosion map* pour calculer la position du reflet rapidement, mais cela introduit beaucoup d'approximations.

3 Algorithme

La méthode proposée dans cet article est une approche par sommets : elle consiste à calculer le reflet de chaque sommet de la scène sur le réflecteur puis à utiliser le matériel graphique pour réaliser l'interpolation entre les sommets.

On ne connaît pas de formule analytique rapide à évaluer pour trouver la position du reflet d'un sommet, même pour des formes très simples de réflecteur comme des sphères ou des cylindres. Pour calculer la position du reflet d'un sommet, on va donc utiliser le principe de Fermat : la lumière emprunte toujours des chemins de longueur extrêmale, comme l'illustre la figure 2. Pour tout point V de la scène, on cherche le point P sur le réflecteur qui réalise un extremum de la somme des distances $d = EP + PV$.

Plusieurs points du réflecteur peuvent se projeter sur le même pixel à l'écran. Cependant seul un de ces points doit être visible, et les autres, qui sont situés derrière, doivent être masqués. Le même phénomène se produit sur le réflecteur : plusieurs objets de la scène peuvent se refléter au même point du réflecteur, et il faut éliminer ceux qui sont masqués. Il est donc nécessaire de mettre en place une élimination des parties cachées à deux niveaux, comme l'illustre la figure 4. Ces mécanismes seront détaillés en section 3.4

3.1 Données nécessaires

L'objet réfléchissant est un des objets de la scène, désigné par l'utilisateur. Le réflecteur doit être un objet lisse (en particulier, pas de surface granuleuse ou d'arêtes vives).

Pour les besoins de l'algorithme nous avons besoin d'une représentation paramétrée $f(u,v)$ du réflecteur. Il n'est pas nécessaire de connaître explicitement la fonction $f(u,v)$. On peut en effet se contenter d'un échantillonnage $f(u_i, v_i)_{i,j}$ à intervalles réguliers que l'on stocke dans une texture à 2 dimensions. Chaque élément de texture contient la position en 3 dimensions du réflecteur au paramètre correspondant. Nous aurons également besoin de connaître les dérivées partielles $\frac{\partial f(u,v)}{\partial u}$ et $\frac{\partial f(u,v)}{\partial v}$, sous forme analytique ou échantillonnées sous forme de texture.

3.2 Principales étapes de l'algorithme

On commence par dessiner la scène sans le réflecteur avec une méthode de rendu classique utilisant la carte graphique. Ensuite, on rend une première fois le réflecteur dans une texture de profondeur, sans calculer les reflets. Ceci sera utile pour l'élimination des parties cachées expliquée à la section 3.4.

Puis on dessine les reflets. Pour cela on modifie le *vertex shader* pour modifier la façon dont les sommets sont projetés, afin de les projeter sur le réflecteur avant de les afficher à l'écran. Cette modification est expliquée à la section 3.3. On modifie aussi le *pixel shader* afin de déplacer le point de vue dans le calcul de l'éclairage (voir section 3.5).

Enfin on calcule l'aspect du réflecteur, qu'on superpose aux reflets à l'aide d'un *blending*.

1. rendu de la scène sans le réflecteur ;

2. rendu du réflecteur dans une texture de profondeur. Cette texture de profondeur sera utilisée pour l'élimination des parties cachées ;

3. nouveau rendu de la scène sans le réflecteur en modifiant le pipeline graphique de la façon suivante :
   - au niveau du traitement des sommets : projeter les sommets selon f_B au lieu f_A de (voir figure1)
   - au niveau du traitement des pixels : corriger la position de l'œil pour le calcul d'éclairage, élimination des parties cachées ;

4. si le matériau du réflecteur a une composante non spéculaire, on la superpose au résultat avec un *blending*

3.3 Calcul de la position des reflets des sommets

Cette étape remplace la projection habituelle des sommets sur l'écran. Elle se déroule donc dans le *vertex shader* de la carte graphique. Pour chaque sommets de la scène on va calculer la position de son reflet sur le réflecteur.

Pour calculer ce reflet, on cherche un extremum de la distance parcourue par la lumière selon le principe de Fermat (voir figure 2). Ainsi, pour tout point V de la scène, on cherche le point P sur le réflecteur qui réalise un extremum de la somme des distances $d = EP + PV$. Pour cela, on utilise un algorithme de minimisation 2D dans l'espace (u,v) des paramètres.

Le calcul d'un extremum se déroule comme suit :

1. choix de 3 points initiaux A_0, B_0 et C_0 dans l'espace (u,v) ;
2. tant que l'on a pas atteint le nombre maximum d'itérations et que la taille du triangle ABC est suffisante :
   - détermination d'un nouveau point (u,v) à l'aide des gradients $\mathbf{G_A}$, $\mathbf{G_B}$, $\mathbf{G_C}$ de $d(u,v)$ en chacun des trois points A, B et$C$;
   - le point le plus éloigné du nouveau point est éliminé et est remplacé par (u,v) ;
3. le reflet est placé à la position (u,v).

Le calcule de la position des reflets est effectué dans le *vertex shader* de la carte graphique.

3.3.1 Choix des points initiaux

La position du reflet converge très vite : en pratique quelques itérations suffisent pour pratiquement n'importe quels les points de départ de l'algorithme. Cependant, si le triangle initial est petit et contient le reflet, la convergence sera plus rapide.

Pour initialiser l'algorithme on cherche donc des points caractérisant la zone dans laquelle se trouve le reflet. Idéalement, le reflet doit se trouver à l'intérieur du triangle initial. Cependant, pour un réflecteur quelconque ce ne sera pas toujours le cas, mais l'algorithme étant capable d'extrapoler, le reflet pourra quand même être calculé.

Deux des points initiaux nous sont inspirés par le cas où le réflecteur est une sphère. Dans ce cas le calcul du reflet perd une dimension pour devenir 1D : le reflet se trouve sur l'intersection de la sphère et du plan passant par E, V

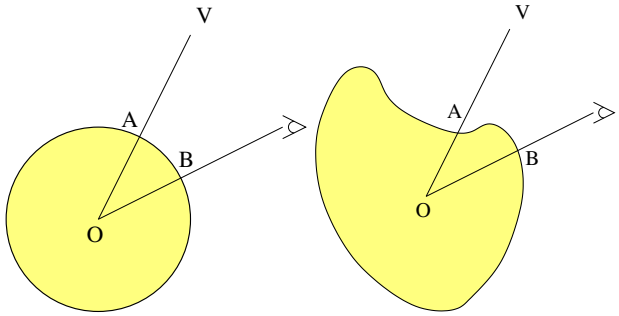

FIG. 3 – Initialisation des points pour l'étape de recherche de l'extremum. Pour un réflecteur sphérique, A et B délimitent la position du reflet. Ce n'est pas forcément le cas pour un réflecteur quelconque, mais A et B sont néanmoins utilisés pour caractériser la zone du reflet

et le centre de la sphère. Ofek a montré dans [Ofe98] que le reflet du sommet se situe entre A et B, où A et B sont les projections respectives du sommet et de l'œil sur la sphère (voir figure 3). Les cas limites sont atteints lorsque le sommet (pour A) ou l'œil (pour B) est contre la sphère. A et B délimitent donc la position du reflet sur une sphère.

Dans le cas où le réflecteur n'est pas une sphère, on choisit un point O à l'intérieur du réflecteur et on prend comme points de départ les intersections des segments [œil,O] et [sommet,O] avec le réflecteur (voir figure 3). Le reflet n'est alors pas forcément situé entre ces deux points, néanmoins ces points sont souvent à proximité du reflet et fournissent un bon point de départ pour l'algorithme de minimisation.

Le troisième point C est tout simplement choisi de manière a former un triangle pas trop plat dans l'espace des paramètres.

3.3.2 Calcul du nouveau point

On va effectuer une descente de gradient dans l'espace (u,v) des paramètres. On maintient un triangle ABC dont les sommets se rapprochent de la position du reflet à chaque étape par la procédure suivante : on calcule une nouvelle approximation P du reflet en utilisant les gradients $\mathbf{G_A}$, $\mathbf{G_B}$, $\mathbf{G_C}$ de $d(u,v)$ en A, B et C, puis P remplace le sommet du triangle qui est le plus éloigné de lui.

On commence par calculer les gradients $\mathbf{G_A}$, $\mathbf{G_B}$, $\mathbf{G_C}$ de $d(u,v)$ en chacun des trois points A, B et C.

En notant $|\vec{A}| = \frac{\vec{A}}{\|\vec{A}\|}$, on obtient une expression du gradient de $d(u,v)$ pour tout (u,v) :

$$\begin{aligned}\frac{\partial d}{\partial u}(u,v) &= -\frac{\partial f(u,v)}{\partial u}.(|E-f(u,v)|+|V-f(u,v)|)\\ \frac{\partial d}{\partial v}(u,v) &= -\frac{\partial f(u,v)}{\partial v}.(|E-f(u,v)|+|V-f(u,v)|)\end{aligned}$$

Dans le cas où on a pré-calculé $f(u,v)$, $\frac{\partial f(u,v)}{\partial u}$ et $\frac{\partial f(u,v)}{\partial v}$, l'évaluation du gradient ne demande que très peu de calculs.

En interpolant linéairement les trois gradients $\mathbf{G_A}$, $\mathbf{G_B}$ et $\mathbf{G_C}$, on cherche une approximation P du point où le gradient s'annule. P vérifie alors :

$$P = (1-\alpha-\beta)A+\alpha B+\beta C \quad (3.1)$$

$$0 = (1-\alpha-\beta)\mathbf{G_A}+\alpha\mathbf{G_B}+\beta\mathbf{G_C} \quad (3.2)$$

Les coordonnées barycentriques α et β s'obtiennent en résolvant l'équation 3.2. Les coefficients α et β ne sont pas forcément compris entre 0 et 1, et dans cas l'algorithme a alors effectué une extrapolation qui lui permet de trouver le reflet même lorsque celui-ci n'est pas situé dans le triangle ABC. Le point P est ensuite calculé grâce à l'équation 3.1.

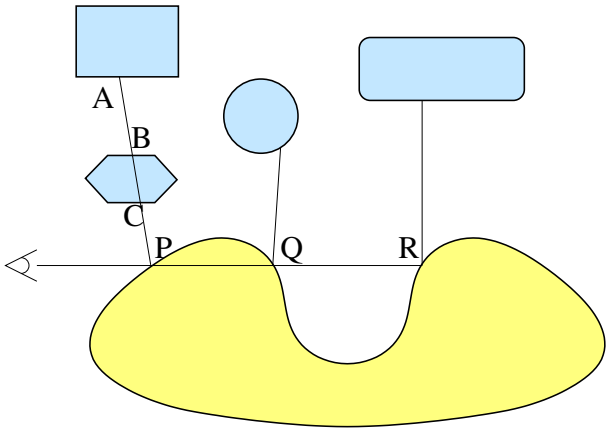

FIG. 4 – Le réflecteur apparaît en jaune et les objets réfléchis en bleu. Q et R sont éliminés par un premier niveau d'élimination des parties cachées qui agit sur le réflecteur. A et B sont éliminés par un deuxième niveau qui agit sur les objets réfléchis. Le reflet visible sera finalement celui de C en P.

Le point le plus éloigné de P parmi les points A, B et C est remplacé par P pour l'étape suivant. On réduit ainsi la taille du triangle à l'étape suivante en éliminant le point le plus éloigné de la nouvelle approximation P de la position du reflet.

3.3.3 Arrêt

Lorsque le triangle ABC devient petit, on est proche du reflet exact du sommet. De plus, si le triangle devient vraiment trop petit ou trop plat des erreurs numériques peuvent apparaître dans la résolutions de l'équation 3.2. On arête donc les itérations lorsque un nombre de pas maximal est atteint (en pratique 4 ou 5 pas suffisent) ou que le triangle est assez petit.

3.4 Élimination des parties cachées

Plusieurs points du réflecteur peuvent se projeter sur le même pixel à l'écran. Cependant seul un de ces points doit être visible, et les autres, qui sont situés derrière, doivent être masqués. Le même phénomène se produit sur le réflecteur : plusieurs objets de la scène peuvent se refléter au même point du réflecteur, et il faut éliminer ceux qui sont masqués. Il est donc nécessaire de mettre en place une élimination des parties cachées à deux niveaux, comme l'illustre la figure 4.

Cette élimination est faite pixel par pixel et est effectuée dans le *pixel shader* de la carte graphique.

– Étape appliquée à la scène : plusieurs points de la scène peuvent se refléter au même point du réflecteur, mais seul celui qui est le plus proche du réflecteur doit être visible au final car il masque les autres. Par exemple, sur la figure4, le point C masque les points A et B donc ceux-ci doivent être éliminés. Pour cela on utilise le *Z-buffer* matériel de la carte graphique : après avoir calculé la position sur l'écran du reflet d'un sommet, on donne comme profondeur au sommet $z = PV$ (voir figure 2). Ensuite la profondeur est automatiquement interpolée entre les sommets et l'algorithme du *Z-buffer* permet de garder en chaque pixel le reflet ayant une distance PV minimale.
– Étape appliquée au réflecteur : plusieurs points du réflecteur peuvent se projeter au même point de l'écran, mais seul celui qui est le plus proche du point de vue doit être visible au finale car il masque les autres. Par exemple, sur la figure 4, le point P masque les points Q et R donc les points qui s'y reflètent doivent être éliminés. À l'étape 2 de l'algorithme (voir section 3.2), on a stocké la profondeur du réflecteur en chaque pixel de l'écran. De plus, on connaît la position dans l'espace de chaque point du reflet et on en déduit facilement sa profondeur par rapport à l'oeil. Il suffit donc de comparer la profondeur des points du reflet avec la profondeur du réflecteur au pixel correspondant. Si le reflet est derrière le réflecteur alors il est éliminé.

3.5 Éclairage

Pour le calcul de l'éclairage, il suffit de faire comme si l'œil était en P lorsqu'on calcule l'éclairage du sommet V (notations de la fig.2). On peut ainsi avoir un modèle d'éclairement du reflet cohérent, avec calcul sur la carte graphique et possibilité de rendre les mêmes effets dans le reflet que dans le reste de la scène. Il est ainsi possible de rendre de façon correcte des matériaux dont le comportement dépend du point de vue.

4 Résultats

Des exemples de résultats obtenus sont présentés à la figure 5. À gauche on peut voir les résultats obtenus par notre algorithme, et à droite les résultats obtenus par la méthode de référence (lancer de rayons). Notre algorithme fournit donc de résultats de très bonne qualité et permet d'obtenir un reflet convaincant quelle que soit la distance de l'objet au réflecteur, avec un éclairage identique à celui de la méthode de référence. On peut également observer les effets de parallaxe et rapprocher à volonté le point de vue du réflecteur sans apparition d'aliassage.

Cependant, en observant attentivement l'image du salon rendue avec notre algorithme, on peut noter quelques imperfections dues aux longs triangles, comme par exemple le bord de la table rectangulaire ou les pieds de la chaise la plus proche, qui apparaissent courbés sur l'image de référence et droits sur l'image obtenue en temps réel. Ceci est dû à l'interpolation entre les sommets, interpolation linéaire, approximation qui impose que les bords des triangles soient rectilignes sur le reflet Le sol, initialement représenté par un seul rectangle, a été subdivisé en 16×16 rectangles en pré-traitement de la scène ; et si cela n'avait pas été fait, ses bords seraient apparus droits et pas arrondis.

Des problèmes numériques peuvent également surgir lorsqu'un objet se reflète au voisinage du pôle de la paramétrisation, causant une déformation du reflet ou des artefacts visuels.

Les calculs (position du reflet et éclairage) sont effectués sur la carte graphique. Pour nos tests nous avons utilisé un ordinateur équipé d'un processeur *Pentium 4* à 2.4 GHz et d'une carte graphique *Nvidia 6800 Ultra*. La scène de Yoda comporte environ 2000 polygones et le reflet est calculé à 100 Hz, pour celle du salon (environ 20 000 polygones) on est à 20 Hz environ. À titre de comparaison, les images de référence ont nécessité respectivement plus de 5 et 30 minutes de calcul.

5 Conclusion et travaux envisagés

Nous avons présenté une méthode permettant de calculer rapidement les reflets spéculaires de façon réaliste. Elle calcule précisément le reflet de chaque sommet de la scène et permet de rendre les effets de parallaxe et des matériaux dont la couleur dépend de la position du point de vue pour les objets réfléchis.

Cependant la méthode présente une approximation : l'interpolation entre les reflets des sommets est réalisée de façon linéaire, ce qui entraîne que certains objets qui devraient apparaître courbés deviennent droits. Ce problème peut être résolu par une tesselation fine de la scène, et donc un surcoût de calcul.

L'algorithme reste dans l'ensemble beaucoup plus coûteux que les méthodes utilisant l'approche par pixel type *environment map*. Il est donc plus judicieux de l'utiliser en combinaison avec d'autres méthodes, moins précises mais plus rapides. L'algorithme que nous avons présenté est particulièrement intéressant lorsqu'il est important que le reflet soit physiquement exact. Par exemple, sur une surface très lisse, le cerveau humain est capable de prévoir la forme du reflet. Par conséquent un reflet faux serait choquant et nuirait au réalisme de la scène. Ainsi il est important de représenter ce type reflet avec précision, par exemple avec l'algorithme présenté dans cet article. À l'inverse, si le réflecteur est très irrégulier, voire bosselé, l'observateur n'a pas d'intuition de la forme que doit prendre le reflet. On peut alors se contenter d'une méthode moins précise mais plus rapide que celle présentée ici.

Des évolutions de la méthode sont envisagées. Elles permettraient de :
- représenter des réflecteurs possédant des arêtes vives. Les fortes discontinuités que sont les arêtes vives posent problème lorsque notre algorithme a besoin de réaliser une extrapolation (cas où α et β ne sont pas compris entre 0 et 1, voir section 3.3.2), et l'image obtenue présente alors des artefacts visuels.
- étendre les résultats sur la réflexion à la réfraction. En effet la réfraction est un phénomène proche de celui de la

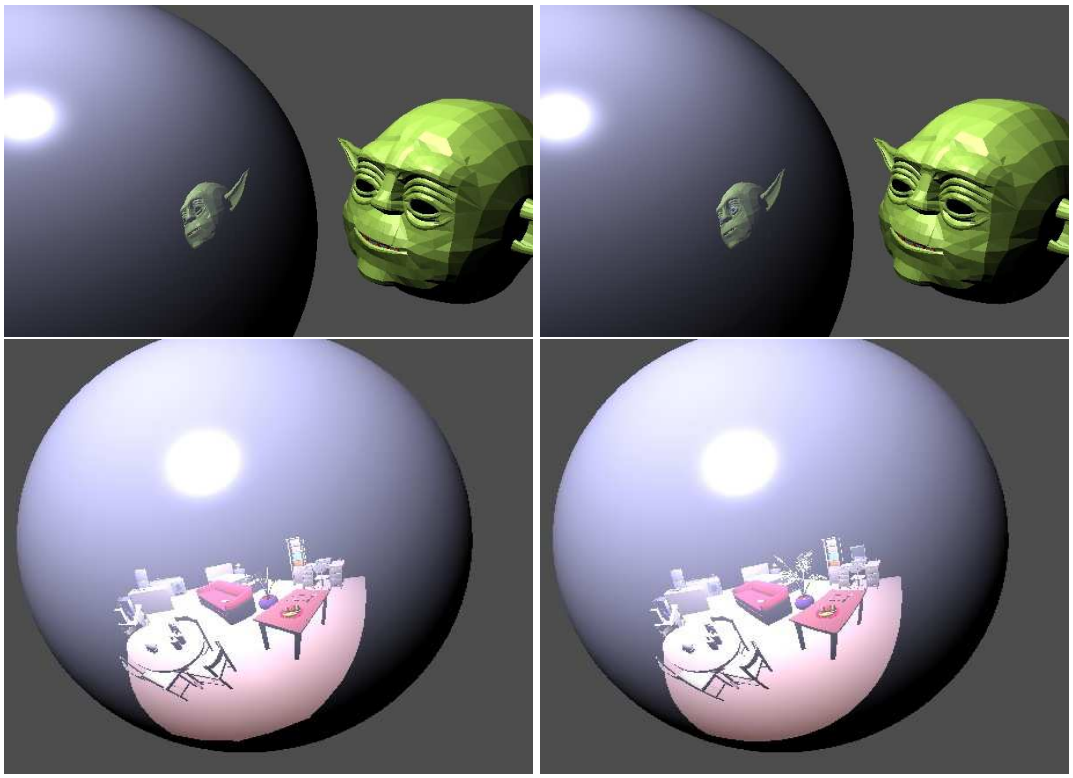

FIG. 5 – À gauche les images obtenues par notre algorithme, et à droite les images de référence obtenues par lancer de rayons

réflexion, les calculs de réfractions sont donc très similaires à ceux que nous avons réalisé et peuvent être résolus par notre méthode.

Références

[BN76] James F. Blinn et Martin E. Newell. Texture and reflection in computer generated images. *Commun. ACM*, 19(10) :542–547, 1976.

[CA00a] Min Chen et James Arvo. Perturbation methods for interactive specular reflections. *IEEE Transactions on Visualization and Computer Graphics*, 6(3) :253–264, 2000.

[CA00b] Min Chen et James Arvo. Theory and application of specular path perturbation. *ACM Trans. Graph.*, 19(4) :246–278, 2000.

[MH92] Don Mitchell et Pat Hanrahan. Illumination from curved reflectors. In *SIGGRAPH '92 : Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 283–291. ACM Press, 1992.

[MP04] Andrew Martin et Voicu Popescu. Reflection morphing. Technical Report CSD TR#04-015, Purdue University, 2004.

[Ofe98] Eyal Ofek. *Modeling and Rendering 3-D Objects*. PhD thesis, Institute of Computer Science, The Hebrew University, 1998.

[OR98] Eyal Ofek et Ari Rappoport. Interactive reflections on curved objects. In *SIGGRAPH '98 : Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 333–342. ACM Press, 1998.

[Pat95] Gustavo A. Patow. Accurate reflections through a Z-buffered environment map. In *Proceedings of Sociedad Chilena de Ciencias de la Computación*, 1995.

[Rey96] Tom Mc Reynolds. Programming with OpenGL : Advanced rendering. SIGGRAPH'96 Course, 1996.

[SKALP05] László Szirmay-Kalos, Barnabás Aszódi, István Lazányi, et Mátyás Premecz. Approximate ray-tracing on the GPU with distance impostors. *Computer Graphics Forum(Proceedings of Eurographics '05)*, 24(3), 2005.