POV-ray: comment booster les tracés

Merci à Diamond Editions pour son aimable autorisation pour la mise en ligne de cet article, initialement publié dans Linux Magazine N°70

Olivier Saraja – olivier.saraja@linuxgraphic.org

Après une désormais longue série d'articles sur KpovModeler et POV-ray, nous avons vu comment mettre en oeuvre des fonctionnalités plutôt avancées de POV-ray, pour obtenir des images (et même des animations!) toujours plus réalistes et plus riches. Jusqu'à présent, nous ne nous sommes que rarement arrêté sur la longueur des temps de calcul. Mais tout le monde n'a pas la chance de posséder un ordinateur dernier-cri surpuissant, gavé de MégaHertz et de MégaOctets. Cet article est donc là pour aider les moins nantis d'entre nous à tirer le meilleur de leur matériel, quel qu'il soit, sans sacrifier (autant que faire se peut) à la sacro-sainte qualité des images tracées.

Figure 1: l'image dont le calcul sert de benchmark à POV-ray

Cette étude va se découper selon trois axes d'optimisation: l'utilisateur (ne riez pas, je vous prie, ce sera certainement plus intéressant que ce que vous pensez), le matériel, et les scènes.

1. Côté utilisateur

Et oui, c'est souvent par là qu'il faut commencer lorsque l'on veut optimiser les choses: il va falloir se débarrasser des mauvaises habitudes (celles que je qualifierai, principalement, d'habitudes de confort) qui prennent inutilement sur les temps de calcul, ou mettre explicitement en oeuvre les petites astuces qui vont pouvoir nous aider.

1.1 Les programmes inutiles

Souvent votre ordinateur a, presque en permanence, plusieurs applications en cours de fonctionnement. Pour peu que vous ayez une connexion permanente à l'internet, je suis à peu près persuadé que vous avez à tout moment une session de client de courriel et votre client de mssagerie instantanée branchés en permanence. Sans compter que la navigation par onglets de **Firefox** ou **Konqueror** est si tentante que vous êtes sûrement en train d'écumer l'internet pendant que **QtPovEditor** a lancé en tâche de fond le calcul d'une image. Je ne parle pas non plus du cédérom qui joue dans le lecteur de l'ordi, ou peut-être des morceaux en Ogg Vorbis crachés par **XMMS**. Chaque programme monopolise des ressources du processeur, et donc, du temps de calcul, alors que celui-ci aurait pu être employé au tracé de votre image.

> **Attention**
>
> Pour accélérer un rendu, fermez toujours les applications qui ne vous sont pas rigoureusement utiles: gestionnaire de tâches, moniteurs divers de système, connexions internet, gestionnaire de son, ainsi que tous ces petits icones qui figurent généralement dans voter barre des tâches, mais qui ne vous sont pas indispensables.

Par exemple, j'ai effectué le tracé de `benchmark.pov` avec les applications suivantes actives (sous KDE): **Kontact** (toujours ouvert), **Open Office** (pour la rédaction de cet article), trois sessions de **Konqueror** (une ouverte sur la documentation HTML de **POV-ray**, une ouverte en tant que gestionnaire de fichier, avec tracé de `benchmark.pov` dans l'émulateur de console, et la dernière en tant que navigateur internet, avec trois onglets de navigation), **Totem** (avec une play-list constituée de 13 morceaux Ogg Vorbis libres tirées du cédérom de LinuxPratique N°27) et quelques instances lancées puis tuées dans l'intervalle de rendu.

```
Total Scene Processing Times
Parse Time:    0 hours  0 minutes  1 seconds (1 seconds)
Photon Time:   0 hours  0 minutes 52 seconds (52 seconds)
Render Time:   0 hours 38 minutes  7 seconds (2287 seconds)
Total Time:    0 hours 39 minutes  0 seconds (2340 seconds)
```

Voici les mêmes résultats avec la même machine, bureau vide (écran de veille également désactivé) avec pour seule session ouverte, une session de **Konqueror** (gestionnaire de fichier, avec émulateur de console pour le tracé de `benchmark.pov`).

```
Total Scene Processing Times
Parse Time:    0 hours  0 minutes  2 seconds (2 seconds)
Photon Time:   0 hours  0 minutes 42 seconds (42 seconds)
Render Time:   0 hours 32 minutes 35 seconds (1955 seconds)
Total Time:    0 hours 33 minutes 19 seconds (1999 seconds)
```

La machine: un Athlon XP 3000+ avec 512 Mo de mémoire RAM. Je ne m'appesantirai pas sur l'étrangeté du temps de *parsing*, peut-être du au fait qu'entre ces deux tests et la fermeture de toutes les applications, certaines n'ont peut-être pas libéré la mémoire RAM aussi vite que souhaité. On constate toutefois que l'on passe de 2340 secondes de tracé à 1999 secondes de tracé, ce qui représente pas loin de 15% de gain de temps. Bref, dans le cadre du rendu d'images complexes et/ou d'animations, vous économiserez pas mal de temps à laisser votre machine tranquille pendant ses calculs.

1.2 Prioriser les applications

Malheureusement, il ne vous sera pas toujours possible de fermer toutes les applications de votre bureau pendant tout le temps du tracé de l'image, aussi apprécierez-vous peut-être de pouvoir donner à certaines applications une plus grande priorité que d'autre.

Cela se fait aisément à l'aide de la commande `nice`. Par exemple:

```
nice -n -5 povray benchmark
```

vous permet de lancer le tracé de l'image benchmark en attribuant à **Pov-ray** une priorité de -5. La priorité va de -20 (priorité la plus élevée) à +19 (priorité la plus faible), mais il est à noter que seul un super-utilisateur (root) peut spécifier des priorités négatives, et que la plupart des

applications, lancées normalement, ont une priorité de 0.

1.3 Le tracé en temps réel de l'image

Vous pouvez également accélérer les calculs en effectuant le tracé sans l'afficher à l'écran. Pour ce faire, cachez la fenêtre de l'image ou, mieux, minimisez-là. Toutefois, vous n'économisez là que du temps de calcul issu du non traçage à l'écran des pixels calculés, ce qui est totalement négligeable si vous tracez une image unique, grand format, très complexe; en revanche, les gains deviennent intéressants dans le cadre du rendu d'une animation constituées d'images plutôt simples à calculer et de faible dimension.

> **Attention**
>
> Dès que possible, abaissez la fenêtre dans laquelle apparaît l'image en cours de calcul ou, mieux, ajoutez `Display = off` dans le fichier `ini` de la scène.

1.4 La mémoire

Plus vous aurez de mémoire RAM, moins vous aurez de chance de voir GNU/Linux se mettre à swaper sur ses disques; par conséquent, vous aurez plus de ressources processeur disponibles pour le tracé de l'image. Avoir une grande quantité e mémoire RAM peut donc accélérer les choses, dans une certaine mesure: si avec 512 Mo de RAM, pour une scène donnée, votre ordinateur ne swape déjà plus, il ne vous servira à rien d'avoir, par exemple 1024 Mo de RAM.

1.5 Les effets spéciaux

De nombreux effets spéciaux, pourtant autorisés par **POV-ray**, consomment plus de ressources de calcul que d'autres. Par conséquent, ils vont gréver les temps de tracé de façon plus ou moins conséquente, et il est souvent utile de connaître ces ennemis sournois que vous appréciez tant, en temps normal:

- L'anti-aliasing: l'anti-crénelage qui donne aux objets de votre image des contours doux et légèrement « floutés » nécessite également plus de temps de calcul, puisqu'il s'agit d'échantillonner et de moyenner les valeurs des pixels voisins selon des règles pré-établies par **POV-ray**.
- Les sources lumineuses: le temps de tracé est souvent fortement proportionnel au nombre de lampes dans votre scène; en effet, au plus vous aurez de lampes, au plus **POV-ray** va être obligé de calculer d'ombres, de transitions ombres/lumières, d'ajuster les intensités des halos, etc. En ce domaine, votre pire ennemi est souvent `l'area_light`, particulièrement gourmande en temps de calcul; en effet, une `area_light` de résolution 3x3 équivaut à peu près à une scène éclairée par 9 lampes à la fois!
- Les médias: qu'il s'agisse de l'usage de la fonction fog ou de celui des medias, **POV-ray** va échantillonner des portions d'images le long des rayons qu'il lance, ce qui aura tendance à alourdir ses calculs et donc à pénaliser les temps de tracé.
- Le flou focal: et oui, cette fonction si facile à mettre en oeuvre et qui booste tellement le réalisme de vos oeuvre va, en fonction du nombre d'échantillons de floutage que vous allez spécifier, dramatiquement allonger les temps de tracé de votre scène.

> **Attention**
>
> Surveillez particulièrement le nombre de sources lumineuses présentes dans votre scène, et réservez l'usage des area lights aux cas particuliers ou vous voulez obtenir des ombres vraiment douces.

1.6 La qualité du tracé

Vous serez généralement (et logiquement) sans concession sur le sujet. Pourtant, au cours de l'élaboration d'une scène, vous aurez de nombreuses occasions d'effectuer des tracés intermédiaire pour voir « la gueule du résultat ». Et c'est là une astucieuse façon d'économiser le temps passé sur le projet en question; diminuez la qualité du rendu. Voici quelques « seuils » qui pourraient vous intéresser:

n=5 et plus: prise en compte des ombres

n=6 et plus: prise en compte des motifs et des photons

n=8 et plus: prise en compte des reflets, de la transparence et de la réfraction

n=11: prise en compte des médias et de la radiosité

> **Remarque**
>
> Pour une prévisualisation très rapide, la commande `povray +Q3` semble donc utile. Pour juger de votre éclairage, `povray +Q5` semble plus appropriée. S'il est important de constater le rendu de la scène avec reflets, réfraction et transparence, `povray +Q8` est un minimum.

2. Côté matériel

Vous songez peut-être à changer de matériel, de sorte à vous faciliter la vie avec **POV-ray**. C'est une bonne idée en soi (s'acheter du nouveau matériel fait toujours plaisir - sauf à nos compagnes!) mais les composants les plus intéressants ne sont pas les mêmes selon l'usage que vous souhaitez faire de votre ordinateur: jeux vidéos, multimédia, musique, échanges de fichiers et... images de synthèse!

2.1 La carte vidéo

La carte vidéo, en particulier, est un lieu commun qu'il convient d'effacer immédiatement. Elle n'apporte aucun avantage lors du calcul en *raytracing*. Tout au plus vous sera-t-elle (marginalement!) utile lors de l'affichage de l'image, ou (certainement plus) lors de la prévisualisation Open-GL de la scène si vous utilisez un modeleur s'en servant (c'est le cas de **Giram** ou de **KpovModeler**) .

2.2 Le processeur et la carte mère

A investir dans du matériel, investissez plutôt dans un processeur rapide, capable d'ingurgiter un grand nombre d'opérations à la seconde. Mais c'est surtout le couple processeur + carte mère qu'il faut bichonner. La vitesse du bus devrait être la plus grande possible, ainsi que les tailles des caches mémoire de niveau 1 et 2. En effet, la vitesse de bus permet de transférer rapidement les opérations depuis les caches jusqu'au processeur, ainsi que d'y stocker les résultats, sans temps mort. La taille de ces cache est également importantes, car elles dialoguent beaucoup plus rapidement avec le processeur que ne le ferait de la mémoire RAM, beaucoup plus lente. Donc, au plus vous aurez de mémoire cache, moins vous solliciterez la mémoire RAM avec une vitesse de transmission des opérations et résultats moindre.

2.3 La mémoire RAM

Malgré tout, avoir une bonne quantité de mémoire RAM est toujours une bonne idée, car lorsque le processeur ne peut se débrouiller avec ses seules mémoires caches, il déborde (presque toujours, d'ailleurs) sur la mémoire RAM, bien moins rapide. Et lorsqu'il a besoin de déborder encore, c'est sur le disque dur, encore moins rapide que la mémoire RAM, il faut le souligner. Pour éviter cela, investir dans une quantité raisonnable de mémoire RAM est toujours une bonne idée

(voir *La mémoire* dans la partie précédente).

3. Côté codage de vos scènes

C'est là la partie la plus passionnante. C'est en effet dans votre façon de coder vos scènes **POV-ray** que vous allez pouvoir optimiser les temps de calcul, en guidant le plus possible le moteur de rendu vers les solutions optimales et, donc, les plus rapides. Les astuces qui existent sont limitées, et découlent souvent de l'approche mathématique qui est caractéristique de **POV-ray**.

3.1 Les isosurfaces

Il y a quelques variables sur lesquelles vous pouvez jouer afin d'accélérer la vitesse de tracé d'une isosurface. Malheureusement, cela se traduira par une perte de netteté, précision, résolution, de l'isosurface, qui sera alors approximée.

accuracy: cette variable détermine avec quelle précision la surface géométrique de l'isosurface est tracée. Une faible valeur correspond à une résolution très fine, et donc à une grande précision et par conséquent des temps de tracé plus longs. Si vous en regardez pas votre isosurface de trop près, essayez des valeurs à chaque fois un peu plus grande jusqu'à obtenir un bon compromis aspect/temps de calcul.

max_gradient: pour accélérer le tracé d'une isosurface, vous pouvez diminuer la valeur de `max_gradient`, au risque toutefois que **POV-ray** ne trace pas la surface avec une précision suffisante pour que toutes ses convolutions naturelles apparaissent correctement: artefacts et tâches noires risquent alors d'apparaître. Notez toutefois qu'en consultant le log du tracé, **POV-ray** vous indique une valeur optimale de `max_gradient` qui aurait des chances de bien fonctionner. En utilisant celle-ci, vous avez alors de bonne chance d'avoir une isosurface décrite avec la précision optimale, sans forcer **POV-ray** à faire des calculs excédentaires et inutiles. Par exemple, vous pourriez obtenir, lors du tracé, l'avertissement suivant:

```
Warning: The maximum gradient found was 16.284, but max_gradient of
the isosurface was set to 25.000. Adjust max_gradient to
get a faster rendering of the isosurface.
```

contained_by : une isosurface cherchant à décrire mathématiquement une portion finie d'un espace infini, en positionnant tous les points de cet espace infini par rapport à la fonction de l'isosurface; on obtient alors trois types de résultats: à l'intérieur, à l'extérieur, à la frontière. Pour éviter que **POV-ray** ne tourne indéfiniment en essayant de positionner tous les points de l'espace infini par rapport à la fonction et ainsi déterminer, pour chacun, son résultat, il est de bon ton de spécifier à l'isosurface des limites « physiques ». Par exemple, si vous spécifiez une isosurface devant décrire une sphère de rayon 1 unité (**`function {x*x + y*y + z*z - 1}`**), limiter l'espace de calcul de l'isosurface à une boîte (**`contained_by {box {-1.1,1.1}}`**), un tout petit peu plus grande serait une excellent idée. Sans compter que le **`max_gradient`** optimal peut augmenter dramatiquement si l'objet contenant est trop large/vaste/surdimensionné par rapport au volume final de l'isosurface.

3.2 La radiosité

Cette méthode de rendu peut être très longue. Il est souvent intéressant de chercher à l'optimiser un minimum. Voici quelques astuces et informations pour vous y aider, ainsi que des valeurs moyennes qui vous aideront à converger vers une solution « économique ». Et oui, gagner du temps lors d'un tracé, c'est souvent, aussi, éviter d'en perdre en essayant des valeurs aberrantes.

normal: si vous cherchez à obtenir une solution de radiosité pure, vous aurez sans doute remarqué que par défaut, **POV-ray** ne tient pas compte du **`normal`** d'un objet, et trace, en radiosité, des surfaces parfaitement lisses. Vous savez certainement qu'en spécifiant l'option **`normal on`** dans le bloc de radiosité, **POV-ray** affichera alors une surface granuleuse conforme au `normal` de l'objet. Cette option, toutefois, est très gourmande en

temps de calcul, et vous gagnerez, la plupart du temps, à employer une méthode mixte: radiosité + éclairage conventionnel, en cherchant un compromis acceptable entre les deux solutions, l'éclairage pouvant vite échapper à votre contrôle.

recursion_limit: essayez de maintenir ce paramètre aussi bas que possible, en faisant des essais prélimianires avec une valeur de 1, puis en l'augmentant jusqu'à deux ou trois. Vous ne devriez, habituellement, pas avoir besoin de l'augmenter plus.

Mémoire RAM: encore un cas où le matériel peut venir à votre rescousse. Le calcul d'une solution de radiosité peut faire une grande consommation de mémoire, pouvant amener vos disques à swapper, et donc à gréver les temps de calcul. Une bonne quantité de mémoire peut donc contribuer à accélérer les calculs les plus complexes.

Objets cachés: ils n'ont que peu ou pas d'importance lors d'un tracé classique, car ils ne vous volent qu'un peu de mémoire et de temps de *parsing*. Lors de la recherche d'une solution de radiosité, toutefois, même un objet caché pourra contribuer à l'illumination globale de par le jeu de l'éclairage indirect. Mais plus vous aurez d'objets dans votre scène, et plus le calcul de la solution de radiosité sera complexe et nécessitera du temps. Supprimer de votre scène les objets cachés qui ne contribuent pas ou peu à la solution de radiosité finale pourra sensiblement diminuer les temps de calcul. Par objets cachés, nous n'entendons pas seulement les objets masqués par d'autres, plus gros. Pensez également aux objets situés *derrière* la caméra.

Bien sûr, vous ne trouverez jamais la solution de radiosité optimale du premier coup, et y parvenir vous demandera sans doute de nombreux rendus, sur lesquels vous souhaiterez vraisemblablement passer le moins de temps possible. Une bonne façon de commencer consiste à spécifier les valeurs de départ suivantes:

```
global_settings {
   radiosity {
      count 20
      nearest_count 1
      gray_threshold 0
      }
   }
```

Vous obtiendrez alors un résultat particulièrement grossier, mais qui sera très visuel et vous permettra tout de suite de voir les parties de votre scène que solution de radiosité va le plus influencer. Vous pouvez alors commencer à jouer sur les valeurs **error_bound** et **minimum_reuse** pour voir comment évolue l'image avec ces deux paramètres.

Bien sûr, vous ne souhaiterez pas en rester là, et chercherez certainement à obtenir une solution de radiosité plus fine. Cela se fera en augmentant le paramètre **count**, par incrément de 50 ou 100, par exemple. Vous pourrez également diminuer le **low_error_factor** vers une valeur comme 0.5 et 0.6, et augmenter le **nearest_count** vers une valeur comme 7 ou 8. La solution de radiosité devrait alors gagner en finesse et présenter une transition plus douce.

Si votre solution de radiosité reste grossière dans les angles de voter scène, ou à proximité de certains objets, en ce cas essayer de diminuer petit à petit la valeur **error_bound**; allez-y prudemment, car les temps de calcul auront tendance à s'allonger considérablement.

Parfois, aussi fines soient les valeurs que vous emploierez, vous obtiendrez des artefacts dans la solution de radiosité, car le moteur ajoutera au dernier moment des échantillons supplémentaires, qui ne seront pas forcément très heureux. Pour les diminuer, vous pouvez faire usage de la variable **pretrace_end**, et lui donner une valeur proche de 0.05. Vous pouvez carrément interdire **POV-ray** d'utiliser des échantillons supplémentaires (et donc s'en tenir aux échantillons déterminés lors de la première passe de calcul) en employant les mots clés **always_sample no**.

Enfin, il se peut que vous ayez spécifié des objets avec des valeurs **ambient** pour les forcer à éclairer votre scène. Pour que le résultat soit satisfaisant, vous viserez certainement ces valeurs types:

```
global_settings {
   radiosity {
      count 500 // et plus! en fonction du nombre d'objets ambiants et la complexité
de la solution
      error_bound 1
```

```
    adc_bailout 0.01
    }
}
```

3.3 Le raytracing

Par définition, le *raytracing* va consister au lancement d'une certaine quantité de rayons, depuis la caméra. Si un rayon touche une surface réfléchissante un nouveau rayon est alors lancé depuis celle-ci pour déterminer la couleur de l'objet au point considéré, celui-ci prenant alors la couleur de son environnement, déterminée par le deuxième rayon. Si l'environnement est lui-même un objet réfléchissant, un troisième rayon lancé, ainsi de suite, peut-être jusqu'à l'infini... Chaque rayon nécessitant du temps de calcul, plus vous demanderez de précision au tracé de votre image au niveau du jeu de réflexion et de réfraction de la lumière, plus vous grèverez vos temps de tracé. Il y a heureusement quelques mécanismes qui permettent de limiter la casse.

ADC_Bailout

Celui-ci est particulièrement utile... ADC_Bailout vous permet de déterminer un seuil au-dessous duquel vous considérez que la couleur résultant de multiples réflexions ou réfractions ne varie plus suffisamment pour justifier qu'un rayon supplémentaire soit lancé. La valeur par défaut est 1/255 (0.0039), mais si les objets présentant les surfaces réflectives ne sont pas au premier plan de votre image, vous pouvez essayer des valeurs de l'ordre de 0.01 car vous ne devriez pas pouvoir observer de différence significative.

```
global_settings {
    ...
    adc_bailout 0.01
    }
```

Max_Trace_Level

Ainsi que nous venons de le voir, à chaque fois qu'un rayon atteint une surface transparente ou réflective, un nouveau rayon est lancé pour déterminer la couleur de l'environnement, et ainsi de suite, de proche en proche, jusqu'à ce qu'un objet non réflectif et/ou non transparent soit atteint. Dans une pièce faite de miroir, le processus de tracé n'aura théoriquement, donc, pas de fin. Nous avons déjà vu **adc_bailout**, qui permet de mettre fin au lancé des rayons supplémentaires lorsque la contribution du dernier rayon à la couleur du pixel n'est plus assez significative. Mais nous pouvons aussi déterminer de façon plus franche (et brutale!) le nombre maximum de rayons consécutifs à lancer. Par défaut, celui-ci est fixé à 5, mais vous gagnerez souvent en temps de calcul à le diminuer à 3, par exemple, si l'objet est situé en arrière-plan de la scène, par exemple, sans que la qualité globale du tracé paraisse altérée.

```
global_settings {
    ...
    max_trace_level 3
    }
```

PS

L'usage conjoint de **max_trace_level** et **adc_bailout** peut s'avérer très efficace si, par exemple, vous avez des objets réflectifs multiples au premier plan de votre image. En fixant, une valeur plus élevée de **max_trace_level** (comme 8 ou 10!), vous garantissez un niveau de récursivité plus que satisfaisant, et des reflets particulièrement réalistes pour les objets de premier plan. En parallèle, en augmentant **adc_bailout** vers une valeur comme 0.05, les objets réflectifs en arrière-plan s'arrêteront rapidement de lancer des rayons supplémentaires. Il en résultera une très grande précision de tracé des reflets pour les objets de premier-plan, et des reflets plus approximatifs pour les objets d'arrière-plan. Le compromis temps de calcul par rapport à la qualité de l'image est alors clairement à votre avantage.

Cas particulier des objets transparents

Imaginons une scène décrivant des verres alignés dans un bar, et l'angle de la caméra

place pratiquement tous les verres les uns derrière les autres. Par défaut, les cinq premiers verres seront visibles les uns derrière les autres, conformément au taux de transparence et à l'indice de réfraction des matériaux mis en jeu. Mais au-delà, les autres verres apparaîtront totalement noirs! C'est à nouveau dû à **max_trace_level**, qui ne lancera plus de rayon au-delà du cinquième objet transparent rencontré. Il vous faudra donc trouver le compromis entre un faible **max_trace_level** permettant d'obtenir une qualité satisfaisante des reflets et un **max_trace_level** plus élevé vous permettant de ne pas obtenir ces disgracieux artefacts noirs au milieu de vos objets transparents. A nouveau, la réponse à ce problème peut résider dans une rigoureuse gestion de l'**adc_bailout**: utilisez une valeur élevée de **max_trace_level**, et « réglez » le temps de rendu de votre scène grâce à **adc_bailout**.

```
global_settings {
   ...
   max_trace_level 25
   adc_bailout 0.1
   }
```

4. Astuces en vrac pour la prévisualisation de vos scènes

Tous les conseils que nous avons vu jusqu'à présent, *a fortiori* utiles pour le tracé final de vos scènes, sont également valables *a priori* pour le tracé de vos prévisualisations. Nous allons donc nous concentrer sur quelques astuces qui devraient vous permettre de simplifier vos scènes, et donc d'accélérer leur tracé, alors que vous ne serez encore qu'en phase de design de celles-ci.

4.1 les sources de lumière

Mauvaise surprise; lorsque vous mettez en place une source de lumière, un rayon est lancé pour chaque pixel de la scène pour savoir s'il est éclairé ou placé dans l'ombre d'un autre objet de la scène. Il en résulte que si vous multipliez les sources de lumière, vous allez également multiplier les temps de tracé. Il n'y a là aucun honnête compromis, si vous voulez obtenir un éclairage et des ombres réalistes. Il n'y a donc que deux façons d'accélérer les choses lors de vos tracés de prévisualisation:

- mettez en commentaire les lampes inutiles ou dont l'éclairage est secondaire ou peu significatif;
- remplacez les **light_source** ponctuelles en ajoutant le mot-clé **spotlight** (un rayon déterminant si le pixel est éclairé ou ombré ne sera lancé que si le pixel en question se trouve dans le « volume d'action » du **spotlight**).

4.2 Les aires lumineuses

Remplacez les aires lumineuses par des lumières classiques en commentant le mot-clé **area_light** ou, mieux, en le remplaçant par le mot-clé **spotlight**, comme vu précédemment. Si vous ne pouvez pas faire autrement que de conserver l'option **area_light**, utilisez alors l'option **adaptative 1**.

4.3 Les textures

Groupez tous les objets possédant les mêmes textures au sein d'une déclaration **union { ... }** afin de diminuer les temps de *parsing* et l'usage de la mémoire (ce qui vous évitera peut-être de swaper ultérieurement). Cette astuce fonctionne particulièrement bien pour les images statiques, mais dans le cadre d'animations, les résultats risquent d'être surprenant dans la mesure où la texture ne bougera plus conformément à l'objet mobile qui la reçoit...

De même, pour les prévisualisations, préférez utiliser de simples couleurs **pigment { rgb ... }** à la place des textures. Vous pouvez obtenir un résultat similaire en utilisant l'option +Q3, par exemple, lorsque vous lancez le tracé de l'image (voir *La qualité du tracé* dans la partie *Côté utilisateur*).

5. Conclusions

Cet article avait un objectif à la fois modeste et ambitieux: vous montrer comment optimiser vos scènes, pour gagner un temps qui peut être précieux. Ambitieux car ce n'est pas en quelques lignes d'un trop court article qu'il est possible d'expliquer, illustrer et démontrer par des exemples concrets et solides les bienfaits de l'optimisation. Modeste, car cet article se contente de survoler de nombreuses notions, admettant à la fois les limites de la place limitée du format papier et les compétences de l'auteur.

Si l'objectif a été atteint, vous êtes désormais sensibilisé à l'optimisation de vos scènes, et vous regarderez dorénavant d'un oeil neuf certains mots-clés obscurs rendus disponibles par **POV-ray**. Plus que jamais, vous pressentez que la documentation, foisonnante et intimidante, de **POV-ray** est et restera longtemps votre meilleure amie, à condition que vous daigniez la feuilleter. Il y a tellement de mots-clés, de paramètres et de situations différentes qu'il faudrait bien l'expérience de toute une vie pour en faire le tour. C'est pourquoi, bénéficier du retour d'expérience d'une communauté entière, qui se retrouve ensuite capturée dans les pages de documentation, peut pour vous être le moyen le plus rapide pour... gagner du temps!

Liens

La homepage de POV-ray (version courante: v3.6): **http://www.povray.org**

La documentation officielle en français de POV-ray:
http://users.skynet.be/bs936509/povfr/index.htm

La section POV-Ray de linuxgraphic.org:
http://www.linuxgraphic.org/section3d/povray/index.html