PHAM Vincent p1610072
ORAN Mikail p1608112

TP Synthèse d'image

Raytracing et Monte Carlo

Lancer de rayons et visibilité

Visualisation par lancer de rayons

Le repère dans lequel les rayons sont le plus facile à calculer est celui de l'image. En effet, pour chaque coordonnées (x, y) dans l'image, nous lui associons pour troisième coordonnée 0 ou 1 si nous voulons qu'il soit l'origine ou l'extrémité du rayon.
Avec ces rayons, nous pouvons construire l'image suivante dans laquelle un pixel blanc correspond à la collision entre un rayon et un triangle, et sinon un pixel noir

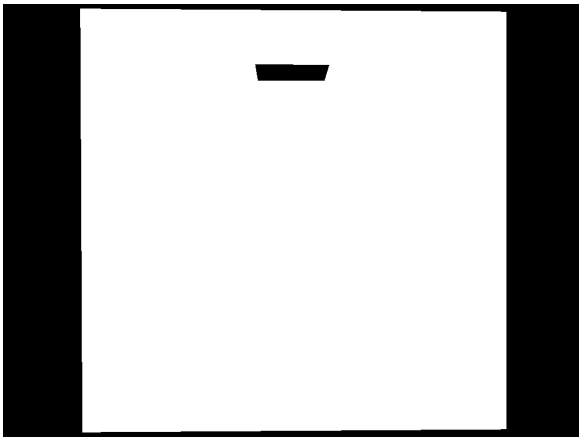

Figure 1 - CornellBox 1024x768, intersections

Matière

Nous pouvons aussi récupérer la couleur du triangle touché par nos rayons. En effet, lors de l'intersection de nos rayons avec un triangle, l'id du triangle ainsi que sa matière sont retournés nous permettant de créer l'image suivante:

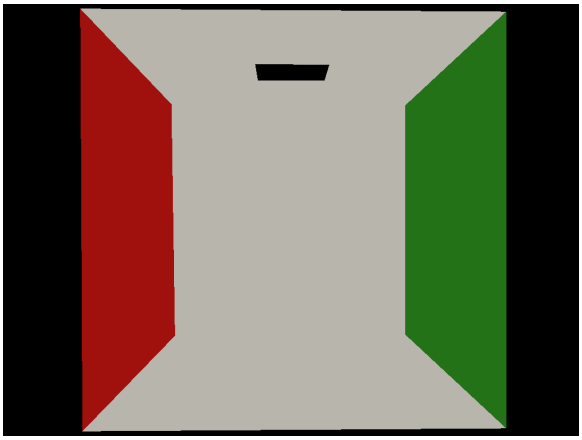

Figure 2 - CornellBox 1024x768, matières

Ombres et éclairage direct

Pour obtenir des ombres simples, nous vérifions pour chaque point dans notre image qu'il soit visible par au moins une source de lumière. Pour cela, nous allons chercher si un obstacle se trouve entre nos points et les sources de lumières. Il faut donc tester l'intersection entre un rayon qui a pour origine notre point (décalé avec la normale du triangle pour éviter les erreurs dans les calculs de flottants) et pour direction, celle vers la lumière. Si le triangle intersecté est celui de la lumière testée, nous dessinons la couleur du triangle, sinon du noir.

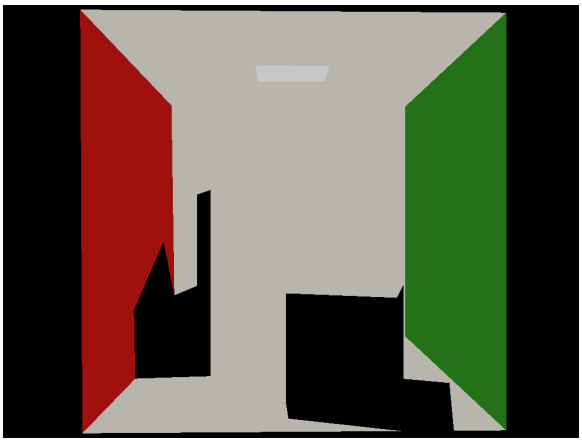

Figure 3 - CornellBox 1024x768, ombres / éclairage direct, 545ms

Nous remarquons déjà un temps de synthèse d'image un peu long expliqué par les nombreux rayons à testés.

Pénombre et éclairage direct

Pour obtenir un ombrage plus réaliste, nous intégrons l'éclairage direct reçu par N points sur nos sources de lumière, l'échantillonnage des points a été fait avec la solution GI compendium eq 18.

$$L_r(p,o) = L_e(p,o) + \int L_e(s,p) f_r(s,p,o) V(s,p) \frac{\cos\theta \cos\theta_s}{d^2(\mathrm{s}\ ,\ \mathrm{p}\ )} ds$$

$$L_r(p,o) \approx L_e(p,o) + \frac{1}{N} \sum_{k=1}^{k=N} L_e(s_k,p) f_r(s_k,p,o) V(s_k,p) \frac{\cos\theta \cos\theta_{sk}}{d^2(s_k,p)} \frac{1}{pdf(s_k)}$$

Pour chaque point visible dans la scène, nous testerons N rayons pour chacune de nos lumières pour obtenir la lumière reçue. La couleur et l'intensité perçue dépendra également de l'angle de la normale avec la lumière et la caméra.

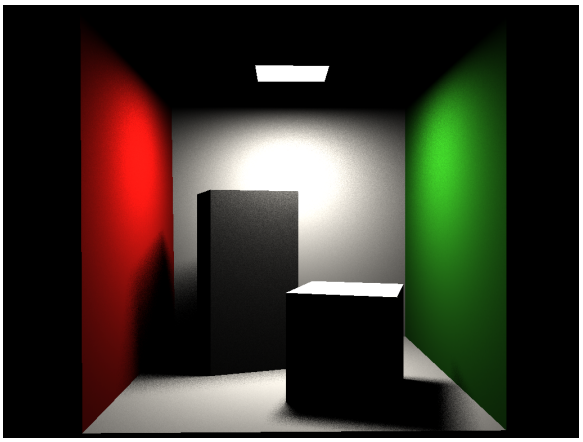

Figure 4 - CornellBox 1024x768, pénombre N = 16, 5902ms

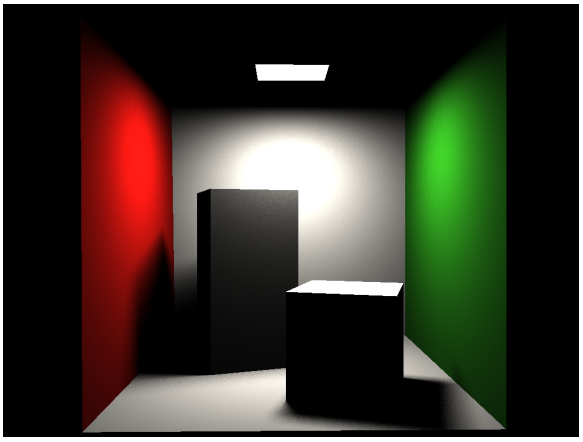

Figure 5 - CornellBox 1024x768, pénombre N = 64, 22 697ms

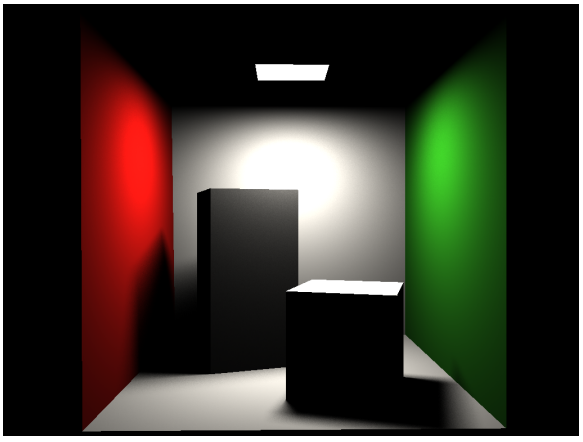

Figure 6 - CornellBox 1024x768, pénombre N = 256, 89 203ms

Nous pouvons remarquer que nos temps de synthèse explosent puisque nous multiplions par N, le nombre de tests par rapport à l'ombrage précédent.

Occultation ambiante

Cette fois nous utilisons la solution GI compendium eq 35 pour générer N directions pour chaque point visible.

$$L_r(p,o) = \int \frac{1}{\pi} V(p,\vec{v}) \cos\theta dv$$

$$L_r(p,o) \approx \frac{1}{N} \sum_{k=1}^{k=N} \frac{1}{\pi} V(p,\vec{v_k}) \cos\theta_k \frac{1}{pdf(\vec{v_k})}$$

Le but sera cette fois de chercher si nos points peuvent "voir" le ciel avec ces rayons et ainsi intégré la lumière reçue du ciel.

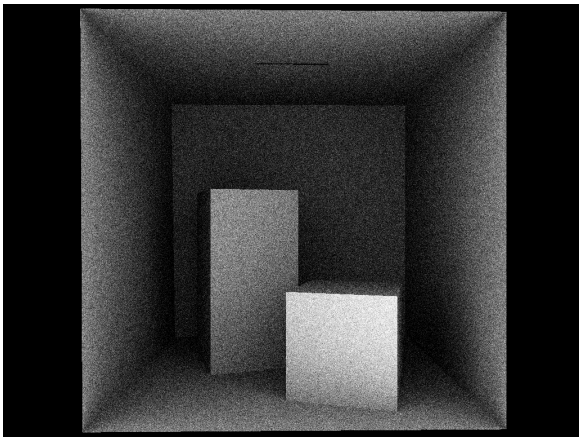

Figure 7 - CornellBox 1024x768, occultation ambiante N = 16, 5295ms

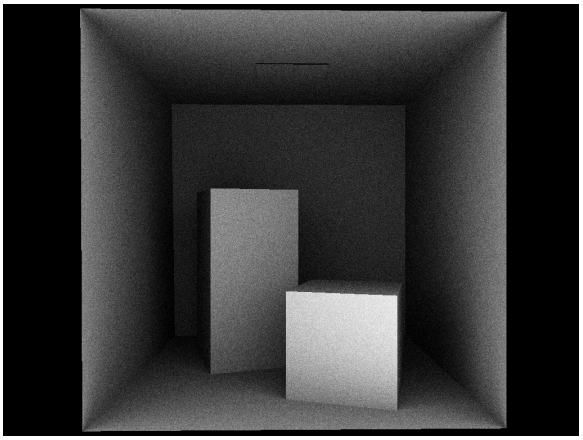

Figure 8 - CornellBox 1024x768, occultation ambiante N = 64, 19188ms

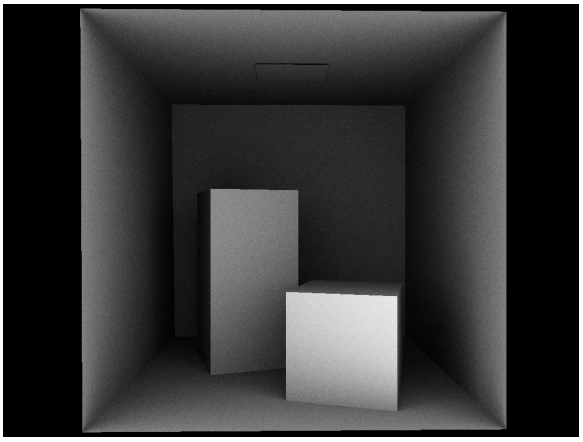

Figure 9 - CornellBox 1024x768, occultation ambiante N = 256, 71 066ms

Les performances sont similaires à celles des calculs de pénombre puisqu'ils sont aussi proportionnels à N.

Optimisations

Parallelisation

Nous avons pu grandement optimiser notre algorithme en parallélisant nos boucles d'itérations. En ajoutant simplement `#pragma omp parallel for schedule(dynamic, 1)` à notre boucle principale, le programme peut désormais partager dans plusieurs thread le lancer de rayons pour chaque ligne au lieu de le faire pour chaque pixel itérativement.

	Itératif	Parallèle
Ombre simple	545 ms	272 ms
Pénombre N = 16	5902 ms	2891 ms
Pénombre N = 64	22 697 ms	11 506 ms

Pénombre N = 256	89 203 ms	46 847 ms
Occultation Ambiante N = 16	5295 ms	1817 ms
Occultation Ambiante N = 64	19 188 ms	6954 ms
Occultation Ambiante N = 256	71 066 ms	26 914 ms

Rien qu'avec ce changement, nous divisons par 2 voir 3 la synthèse de nos images, mais ce n'est pas encore suffisant.

BVH

Nous avons pu jusque là construire des scènes très simples, mais avec des scènes composées de millions de triangles et de centaines de lumières notre algorithme ne pourrait pas les dessiner. Il y aurait beaucoup trop de rayons à tester, c'est pourquoi il faudrait implémenter une structure telle qu'un BVH pour accélérer nos tests. Il nous permettrait de grouper des triangles dans des boîtes englobantes afin d'ignorer un nombre important de tests inutiles.

Cependant, même en suivant le tutoriel présent dans gkit pour les bvh, nous n'avons pas réussi à l'implémenter.