

IFT6095 - Sujets en infographie

Travail pratique numéro 3 - SH et Path Tracing.
17/03/2014

Anis Benyoub

1 Introduction :

Comme sources externes, j'ai utilisé la librairie webgl-utils pour la boucle de rendu pour la partie 2. J'ai aussi récupéré certaines parties du TP1 et TP2. J'ai implémenté toutes les parties du TP (bonus également) à l'exception de l'utilisation des coefficients SH fournis et leur comparaison. J'ai aussi rajouté quelques contributions personnelles supplémentaires que j'ai trouvé intéressantes. Le rapport est un peu long mais comporte beaucoup d'images et d'équations.

2 Spherical Harmonics and Phong Reflection :

Nous partons de l'expression des harmoniques pour le cosinus clampé donnée dans le sujet.

$$f^m = \int_l \max(\mathbf{n} \cdot \omega, 0) * y^m(\omega) d\omega$$

Etant donné qu'en coordonnées sphériques normalisées, nous savons que :

$$\omega(\theta, \phi) = (\sin(\theta) * \cos(\phi), \sin(\theta) * \sin(\phi), \cos(\theta))$$

Comme $n = z$, nous avons alors que :

$$\mathbf{n} \cdot \omega = \cos(\theta)$$

Nous effectuons ensuite un changement de variables dans l'intégrale :

$$\int_{S^2} d\omega = \int_0^\pi \int_0^{2*\pi} \left| \frac{d\omega}{d\theta} \times \frac{d\omega}{d\phi} \right| d\theta d\phi$$

Après développement on trouve :

$$\frac{d\omega}{d\theta} \times \frac{d\omega}{d\phi} = \begin{pmatrix} \cos(\theta) * \cos(\phi) \\ \cos(\theta) * \sin(\phi) \\ -\sin(\theta) \end{pmatrix} \times \begin{pmatrix} -\sin(\theta) * \sin(\phi) \\ \sin(\theta) * \cos(\phi) \\ 0 \end{pmatrix} = \begin{pmatrix} -\sin(\theta)^2 * \cos(\phi) \\ \sin(\theta)^2 * \sin(\phi) \\ \cos(\theta) \sin(\theta) \end{pmatrix}$$

$$\begin{aligned}
\left| \frac{d\omega}{d\theta} \times \frac{d\omega}{d\phi} \right| &= \sqrt{(\sin(\theta)^4 * \cos(\phi)^2 + \sin(\theta)^4 * \sin(\phi)^2 + \sin(\theta)^2 * \cos(\theta)^2)} \\
&= \sqrt{(\sin(\theta)^4 * (\cos(\phi)^2 + \sin(\phi)^2) + \sin(\theta)^2 * \cos(\theta)^2)} \\
&= \sqrt{(\sin(\theta)^4 + \sin(\theta)^2 * \cos(\theta)^2)} \\
&= \sqrt{(\sin(\theta)^2 * (\sin(\theta)^2 + \cos(\theta)^2)} \\
&= \sqrt{(\sin(\theta)^2)} \\
&= |\sin(\theta)|
\end{aligned}$$

Comme θ est entre 0 et π donc $\sin(\theta)$ est toujours positif. Alors on a :

$$\frac{d\omega}{d\theta} \times \frac{d\omega}{d\phi} = \sin(\theta)$$

L'équation finale devient donc : $f_l^m(\theta, \phi) = \int_0^\pi \int_0^{2*\pi} \max(\cos(\theta), 0) * \sin(\theta) * y_l^m(\theta, \phi) d\theta d\phi$

Si on discrétise notre intégrale à l'aide du théorème de monte-carlo avec une distribution uniforme avec N points : $f_l^m(\theta, \phi) = \frac{(2*\pi-0)*(0-\pi)}{N} \sum \sum \max(\cos(\theta), 0) * \sin(\theta) * y_l^m(\theta, \phi)$

J'ai ensuite codé l'estimateur montecarlo (tp3p1a.html), voici les résultats numériques obtenus (Tous les autres facteurs ont donné des valeurs nulles) :

```

Factor L=0, M= 0, VAL = 2.4987924944310436
Factor L=2, M= 0, VAL = 1.3803589958677969
Factor L=4, M= 0, VAL = -0.29992783197988493

```

Pour valider mes résultats, j'ai effectué le calcul sur Maple :

FIGURE 1 – Screenshot des calculs maple.

```

> evalf(f(0, 0));
2.506628274
> evalf(f(2, 0));
1.401247804
> evalf(f(4, 0));
-0.3133285342

```

Dans cette deuxième partie j'ai fait une évaluation numérique des coefficient des harmoniques sphériques en javascript. Le code source est dans le fichier (tp3p1b.html). Voici le résultat numérique de ces valeurs (de la même manière, les coefficients nuls ne sont pas affichés).

```

POW 0
Factor L=0, M= 0, VAL = 2.4756159450847597
Factor L=2, M= 0, VAL = 1.3767345593614992
Factor L=4, M= 0, VAL = -0.28411281924042253
POW 1
Factor L=0, M= 0, VAL = 2.494064980142964
Factor L=2, M= 0, VAL = 3.584045033781966
Factor L=3, M= 0, VAL = 0.12493407875654114
Factor L=4, M= 0, VAL = 1.4645887928749453
POW 2
Factor L=0, M= 0, VAL = 2.5119348175482865
Factor L=1, M= 0, VAL = 0.13738217582817872
Factor L=2, M= 0, VAL = 4.298680906543962
Factor L=3, M= 0, VAL = -0.3831145860474819
Factor L=4, M= 0, VAL = 3.1162073572816205
POW 3
Factor L=0, M= 0, VAL = 2.5073073578266403

```

```

Factor L=1, M= 0, VAL = 0.13130146961612968
Factor L=2, M= 0, VAL = 5.57585063584913
Factor L=3, M= 0, VAL = 0.11917510806978339
Factor L=4, M= 0, VAL = 6.16933987163121
Factor L=5, M= 0, VAL = 0.5290615255307245

```

Je n'ai pas eu le temps de finir cette partie du TP. L'idée était de le faire en utilisant la propriété de la double somme des harmoniques sphériques.

3 2 - Path Tracing :

J'ai fait deux implémentations pour le premier cas (CPU), j'ai fait un échantillonage uniforme naif hémisphérique et un échantillonage autour du cosinus clampé. Ces implémentations correspondent respectivement aux fichiers tp3p2aa.html et tp3p2ab.html. Voici un rendu visuel de ces implémentations pour des chemins de taille 2 pour 10 échantillons par pixel :

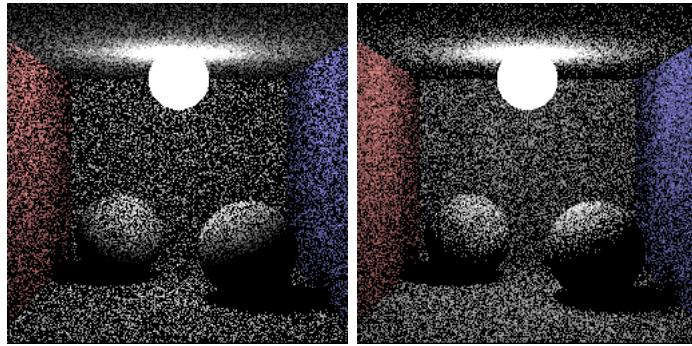


FIGURE 2 – 10 échantillons par pixel. A gauche hemisphérique, à droite cosinus clampé

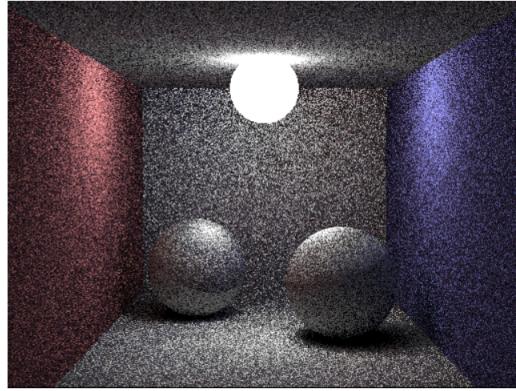


FIGURE 3 – *CPU : Chemins de taille 3, PDF : Hemispherique, 1000 échantillons par pixel.*

Suite à cette implémentation, j'ai implémenté l'acceleration GPGPU de l'uniform sampling grace aux fragment shaders ([tp3p2ba.html](#)). Pour éviter les limitations du glsl quand à la récursivité ainsi que pour éviter un débordement quant au nombre d'instruction, j'ai passé l'algorithme de path tracing en itératif. La première passe calcule une image et fait la moyenne avec les passes précédentes, la deuxième passe affiche cette moyenne et la tone-map.

Le résultat visuel est le suivant :

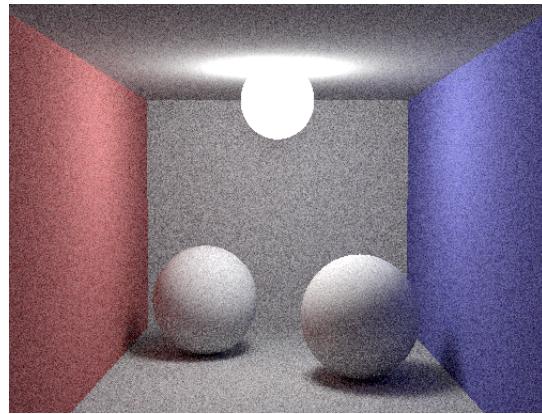


FIGURE 4 – *GPU : Chemins de taille 5, uniform sampling, quelques secondes.*

J'ai aussi fait l'importance sampling selon le cosinus clampé, le convergence vers la solution est beaucoup plus rapide([tp3p2bb.html](#)). Le résultat visuel au bout de quelques minutes est le suivant :

Voici une comparaison entre l'échantillonage naif hémisphérique et celui autour du cosinus clampé, on remarque que la convergence cosinus clampé est plus rapide :

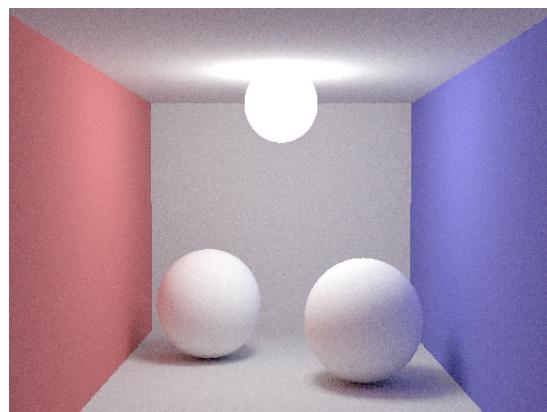


FIGURE 5 – GPU : Chemins de taille 5, clamped cosine sampling, quelques minutes.

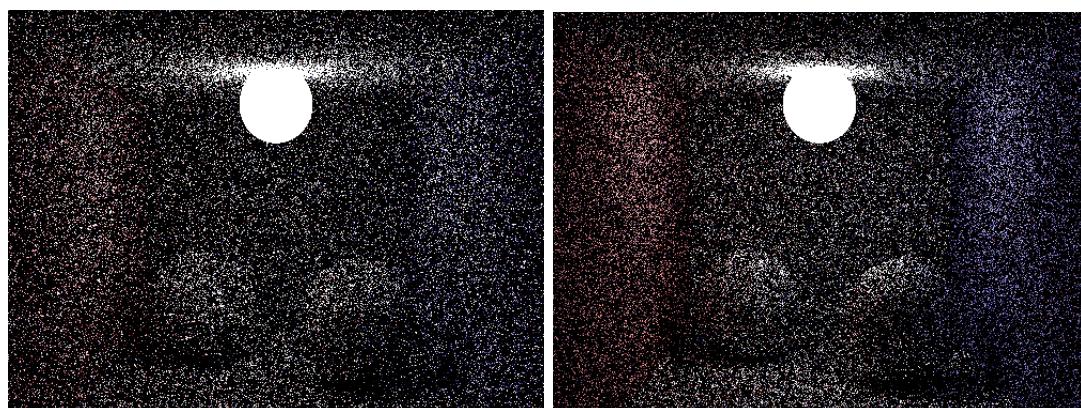


FIGURE 6 – GPU, 5spp, Gauche : Naif Hemisphérique, Droite : Cosinus clampé

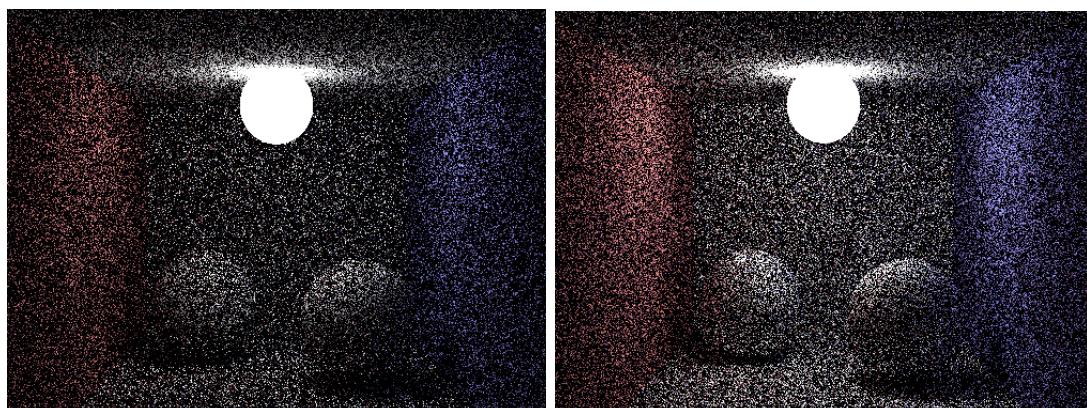


FIGURE 7 – GPU, 10spp, Gauche : Naif Hemisphérique, Droite : Cosinus clampé

J'ai aussi implémenté l'explicit direct illumination ([tp3p2bc.html](#)). Je me suis aidé des équations mathématiques proposées par "Réalistic Ray tracing - Second edition - Peter

Shirley, R. Keith Morle”, section ”sampling a spherical luminaire”. Cette approche propose une implémentation qui réduit beaucoup le bruit du sampling en choisissant une pdf adaptée à l’échantillonage sur une sphère lumineuse.

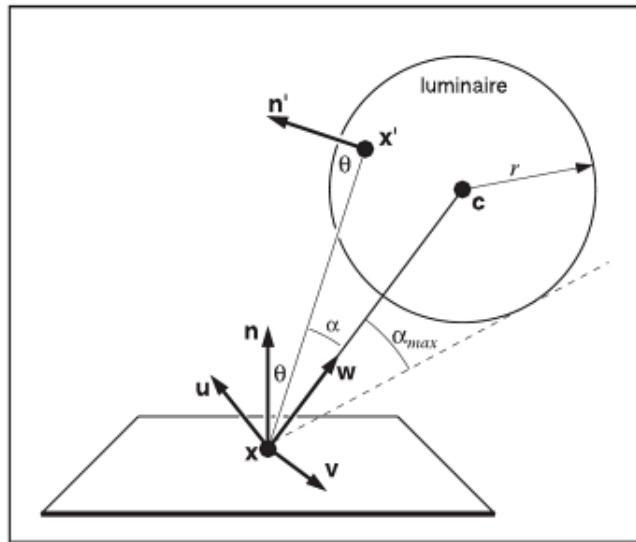


FIGURE 8 – Cas de l’échantillonage hémisphérique

$$L_s(\mathbf{x}, \mathbf{k}_o) \approx \frac{\rho(\mathbf{k}_i, \mathbf{k}_o) L_e(\mathbf{x}', \mathbf{x} - \mathbf{x}') v(\mathbf{x}, \mathbf{x}') \cos \theta_i \cos \theta'}{p(\mathbf{x}') \|\mathbf{x} - \mathbf{x}'\|^2}.$$

FIGURE 9 – Approximation de l'équation de rendu

$$\alpha_{\max} = \arcsin \left(\frac{r}{\|\mathbf{x} - \mathbf{c}\|} \right) = \arccos \sqrt{1 - \left(\frac{r}{\|\mathbf{x} - \mathbf{c}\|} \right)^2}.$$

FIGURE 10 – Angle maximal

the sphere

$$q(\mathbf{k}_i) = \frac{1}{2\pi(1 - \cos \alpha_{\max})},$$

and we get

$$\begin{bmatrix} \cos \alpha \\ \phi \end{bmatrix} = \begin{bmatrix} 1 + \xi_1(\cos \alpha_{\max} - 1) \\ 2\pi\xi_2 \end{bmatrix}.$$

FIGURE 11 – Echantillonage aléatoire

$$p(\mathbf{x}') = \frac{q(\mathbf{k}_i) \cos \theta'}{\|\mathbf{x} - \mathbf{x}'\|^2}.$$

FIGURE 12 – Valeur de la pdf

Le résultat visuel est le suivant au bout de deux secondes est très bluffant :

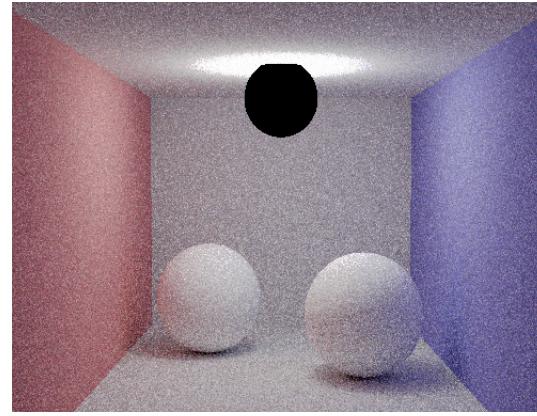


FIGURE 13 – GPU : Explicit path tracing.

Voici une comparaison entre l'échantillonage naïf hémisphérique, celui autour du cosinus clampé et l'explicit direct illumination. On remarque que la convergence direct converge

très vite :

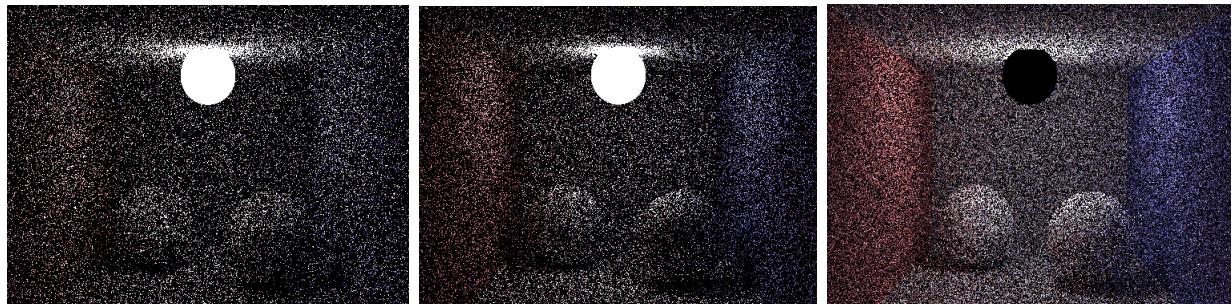


FIGURE 14 – GPU, 5spp, Gauche : Naif Hemisphérique, Millieu : Cosinus clampé, Droite : Explicit direct illumination

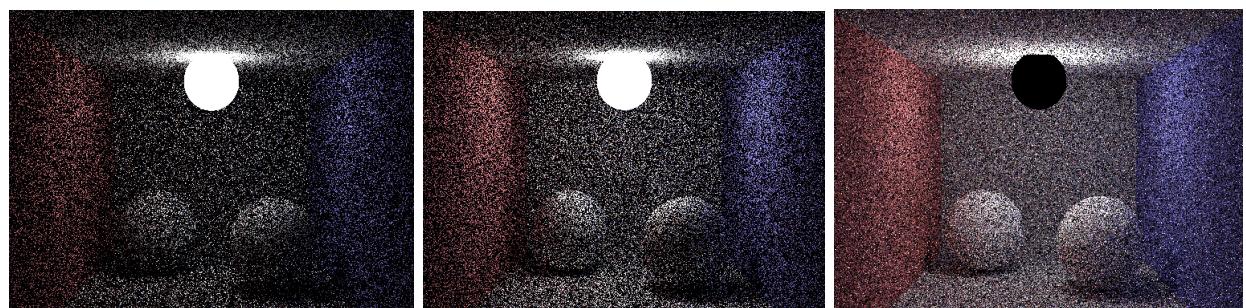


FIGURE 15 – GPU, 10spp, Gauche : Naif Hemisphérique, Millieu : Cosinus clampé, Droite : Explicit direct illumination

4 Bonus :

J'ai codé le flou, la réflexion, la réfraction et le brillant pour la cornell box ([tp3p2bd.html](#)). Il suffit de changer les propriétés dans le fragment shader pour obtenir chacun de ces résultats suivants :

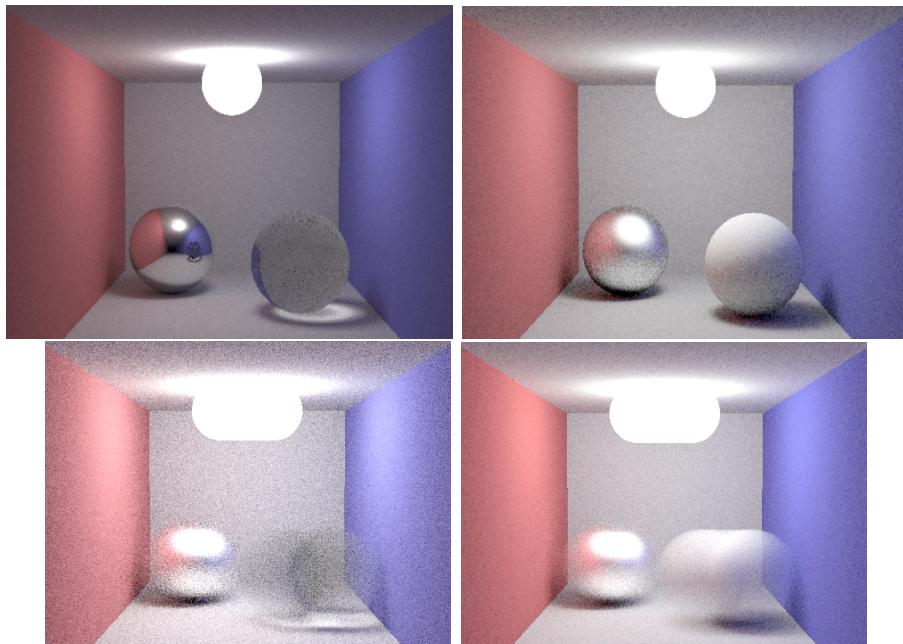


FIGURE 16 – GPU : Flou, réflexion, réfraction et glossy dans la cornell box.