Projet : path tracer

L'objectif de ce projet est d'implémenter un path tracer en GLSL, comme vu en cours. L'equation de réflectance sera approximée avec l'estimateur de Monte-Carlo, comme vu en cours :

$$L_o(\mathbf{v}) \approx \frac{1}{N} \sum \frac{f(\mathbf{l}, \mathbf{v}) L_i(\mathbf{l})(\mathbf{n}.\mathbf{l})}{pdf(\mathbf{l})}$$

Le principe est de lancer un rayon par pixel et, pour chaque intersection, lancer un rayon aléatoire sur l'hémisphère pour déterminer une nouvelle direction l. Cette nouvelle direction sera à son tour utilisée comme point de départ du nouveau rayon à l'itération suivante. Si rien n'est touché, la couleur de l'environnement sera considérée la source de lumière (et donc multipliée au masque précédemment calculée lors des anciens rebonds pour être accumulée dans un buffer).

L'algorithme est très similaire à celui du ray-tracing (la fonction directIllumination a disparue et l'on n'utilise pas exclusiement le vecteur reflété, mais une direction aléatoire). Vous implémenterez votre path tracer dans un noeud “genericPingpong” de Gratin. Ce type de noeud fonctionne de la même manière qu'un genericImage, excepté que la texture dans laquelle on écrit est réinjectée en entrée à ce même shader (elle s'appelle pingpong0). Cela permet donc d'accumuler les contributions de couleurs pour chacun des rayons. Il faut aussi stocker la nombre de rayons lancés dans cette texture (dans le canal alpha) pour pouvoir effectuer la division par N dans un noeud genericImage connecté à la sortie.

Votre path tracer devra contenir les features suivantes par défaut :

- Une scène implicite ou analytique contenant plusieurs objets (vous avez le choix d'utiliser la méthode analytique ou le ray-marching pour les tests d'intersection) – Vu au cours et tps 2 et 3.
- Des matériaux différents (diffus / glossy / spéculaire) gérés avec la BRDF de Phong – Vu au cours / tp 4.
- Des textures procédurales simples (checkerboard ou autre) et fractales (marbre, bois, etc...) et des textures bitmaps (la carte d'environnement en est une) qui modifient les matériaux, formes, éclairement dans une scène – Vu au cours / tp 5.
- De l'antialiasing : chaque rayon lancé doit échantilloner l'espace du pixel en générant des rayons aléatoires à l'intérieur du pixel – Vu au cours 6.
- Un échantillonnage aléatoire (uniforme) de l'hémisphère autour des normales aux points considérés. **La pdf est une constante dans ce cas et est égale à 1/(2*pi)** – Vu au cours 6.

Quelques infos dont vous aurez besoins pour implémenter votre path tracer :

La BRDF utilisée sera la combinaison d'un terme diffus lambertien et d'un lobe de Phong pour les rayons reflétés. Dans l'équation ci dessous, kd est la couleur diffuse du matériau, ks est sa couleur spéculaire et alpha contrôle la rugosité (plus alpha est grand, plus le matériau est spéculaire). Enfin, r est le vecteur reflété de l par rapport à n : max(reflect(l,n),0.0).

$$f(\mathbf{l}, \mathbf{v}) = \frac{kd}{\pi} + ks\frac{\alpha + 1}{2\pi}(\mathbf{r}.\mathbf{v})^{\alpha}$$

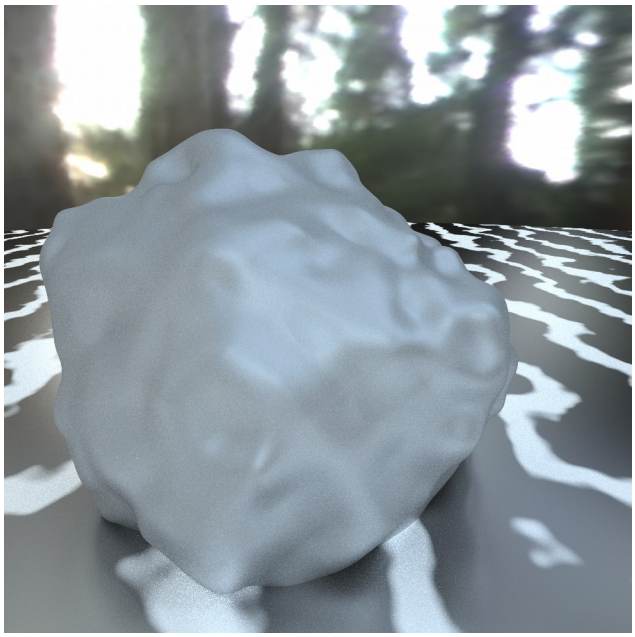
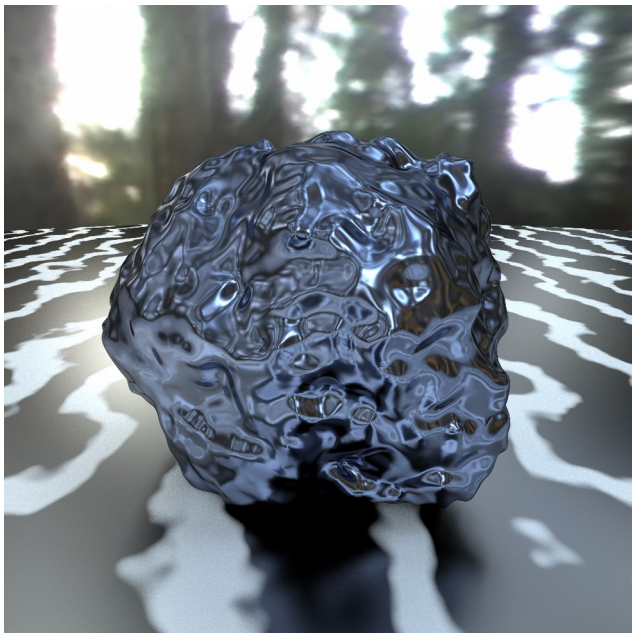
Vous aurez besoins d'une fonction qui génère des valeurs aléatoires dans votre shader. Vous pourrez utiliser la fonction rand() suivante qui génère une valeur aléatoire entre 0 et 1. Le seed est une variable globale initialisée en fonction de la position du pixel dans l'image, ainsi que du numéro de la passe qui est exécutée (le nombre de fois que vous exécutez votre shader pour obtenir l'image finale. La variable numpass est accessible par défaut dans un noeud pingpong) :

```
float seed = (float(numpass)*0.9610356 + 1024. * texcoord.x + texcoord.y)*1.51269341231;

float rand() { return fract(sin(seed++)*43758.5453123); }
```

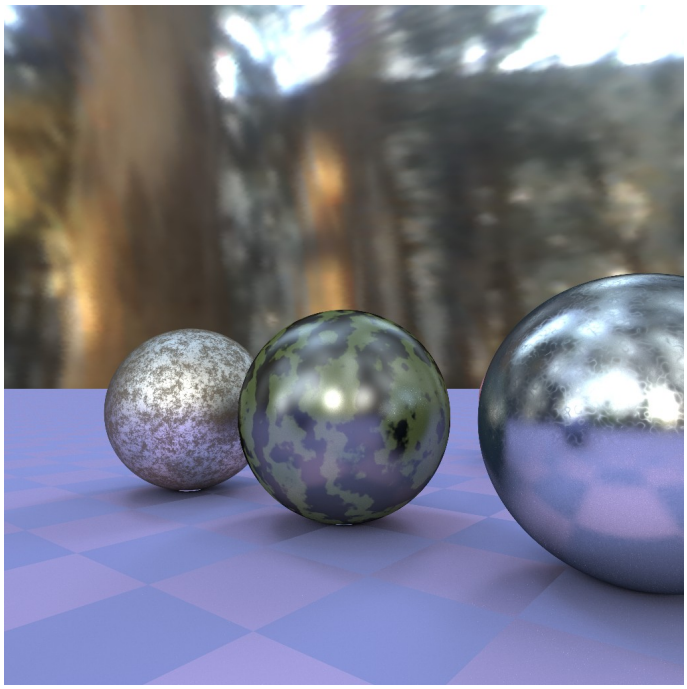
Enfin, la fonction suivante pourrait vous être utilise pour générer du bruit. Elle calcule 1 octave d'une texture “value noise” (qui ressemble à perlin). Il vous faudra bien sur combiner plusieurs textures à plusieurs octaves / fréquences pour générer des textures fractales.

```
float hash( float n ) { return fract(sin(n)*43758.5453123); }

float valueNoise( in vec3 x ) {
vec3 p = floor(x);
vec3 f = fract(x);
f = f*f*(3.0-2.0*f);
float n = p.x + p.y*157.0 + 113.0*p.z;
return mix(mix(mix( hash(n+ 0.0), hash(n+ 1.0),f.x),
mix( hash(n+157.0), hash(n+158.0),f.x),f.y),
mix(mix( hash(n+113.0), hash(n+114.0),f.x),
mix( hash(n+270.0), hash(n+271.0),f.x),f.y),f.z);
}
```

Quelques exemples qui peuvent inspirer vos propres scènes :

Ces exemples utilisent des surfaces implicites sphériques déformées par des textures fractales avec plusieurs types de matériaux. 2000 rayons / pixel. 1024x1024 pixels.



La scène du TP précédent (analytique) rendue avec path tracing : 20 000 rayons / pixel. 1024x1024 pixels.

Extensions

Vous ajouterez par la suite des extensions à votre path tracer que vous choisirez dans la liste ci-dessous.

Importance sampling

Les 2 functions suivantes permettent d'échantilloner l'hémisphère en prenant en compte le cosinus de l'angle entre le rayon lumineux et la normale d'une part et le lobe de de la BRDF de Phong d'autre part. Leur utilisation augmentera fortement le temps de convergence de l'image générée. Vous pourrez utiliser l'une, l'autre ou les deux en fonction des propriétés de matériaux (la première pour un matériau diffus, la seconde pour un matériau glossy ou aléatoirement entre les 2 pour un matériau à la fois diffus et glossy).

```
vec3 cosWeightedRandomHemisphereDirection(in vec3 n) {

// n = normal

// http://people.cs.kuleuven.be/~philip.dutre/GI/TotalCompendium.pdf (page 20)
float r1 = rand();
float r2 = rand();
float tmp1 = PI*2*r1;
float tmp2 = sqrt(r2);
float x = cos(tmp1)*tmp2;
float y = sin(tmp1)*tmp2;
float z = sqrt(1.-r2);
vec3 w = normalize(n);
vec3 u = normalize(cross(w.yzx,w));
vec3 v = cross(w,u);


return u*x+v*y+w*z;
}


vec3 reflectWeightedRandomHemisphereDirection(in vec3 r,in float alpha) {
// r = reflected vector
// alpha = material rugosity
// http://people.cs.kuleuven.be/~philip.dutre/GI/TotalCompendium.pdf (page 20)
float r1 = rand();
float r2 = rand();


float tmp1 = PI*2*r1;
float tmp2 = 1.-r2;
float tmp3 = sqrt(1.-pow(tmp2,2./(alpha+1.)));
float x = cos(tmp1)*tmp3;
float y = sin(tmp1)*tmp3;
float z = pow(tmp2,1./(alpha+1.));
vec3 w = normalize(r);
vec3 u = normalize(cross(w.yzx,w));
vec3 v = cross(w,u);


return u*x+v*y+w*z;
}
```

ATTENTION : la pdf ne sera plus une constante dans ce cas. Elle sera égale à

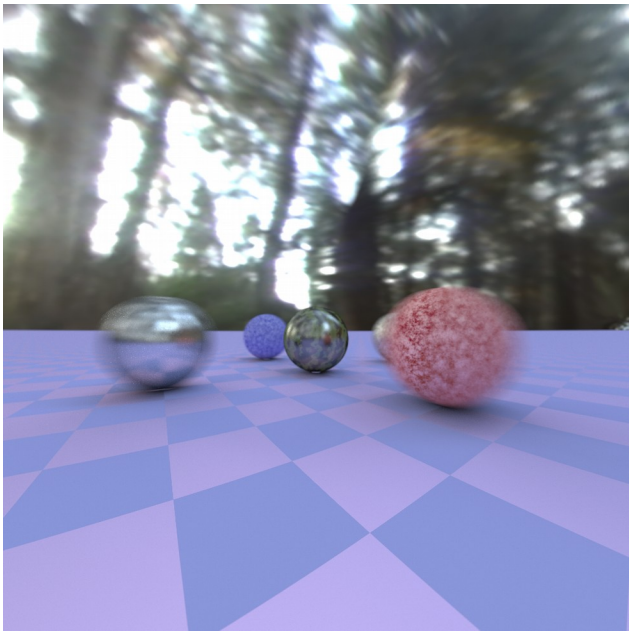
pdf = clamp(dot(l,n),0.,1.)/PI; // pour la version cosWeighted...

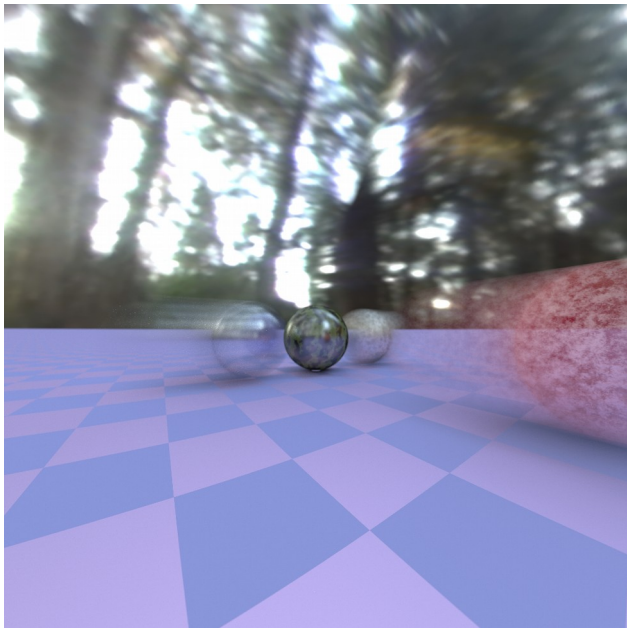
*pdf = ((alpha+1.)*pow(clamp(dot(r,l),0.,1.),alpha))/PI*2; // pour la version reflectWeighted...*

Notez que les pdf annulent parfois une partie de la BRDF dans l'équation du rendu. Vous pouvez donc aussi optimiser algorithme en le prenant en compte.

Motion blur

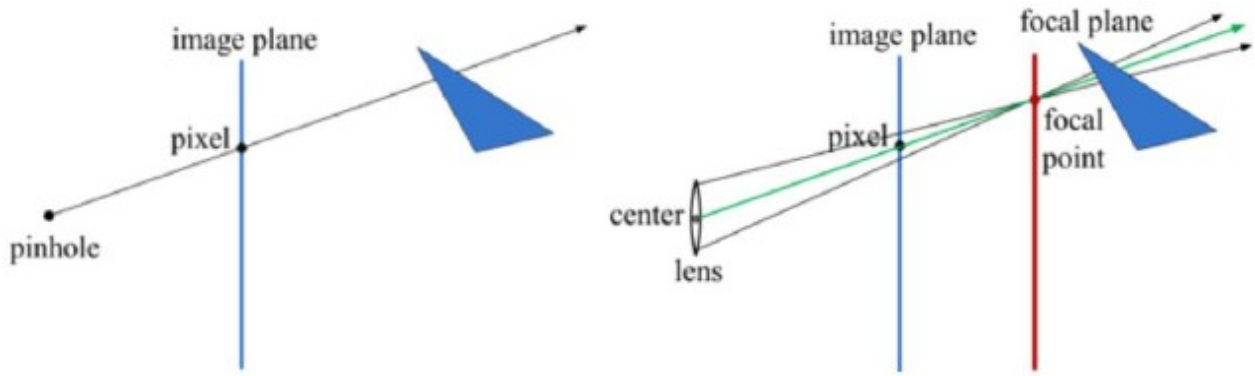
Le motion blur peut s'obtenir très simplement en échantillonant aléatoirement la scène en mouvement dans un intervalle de temps prédéfinis. Vous pourrez obtenir des scènes comme ci-dessous :



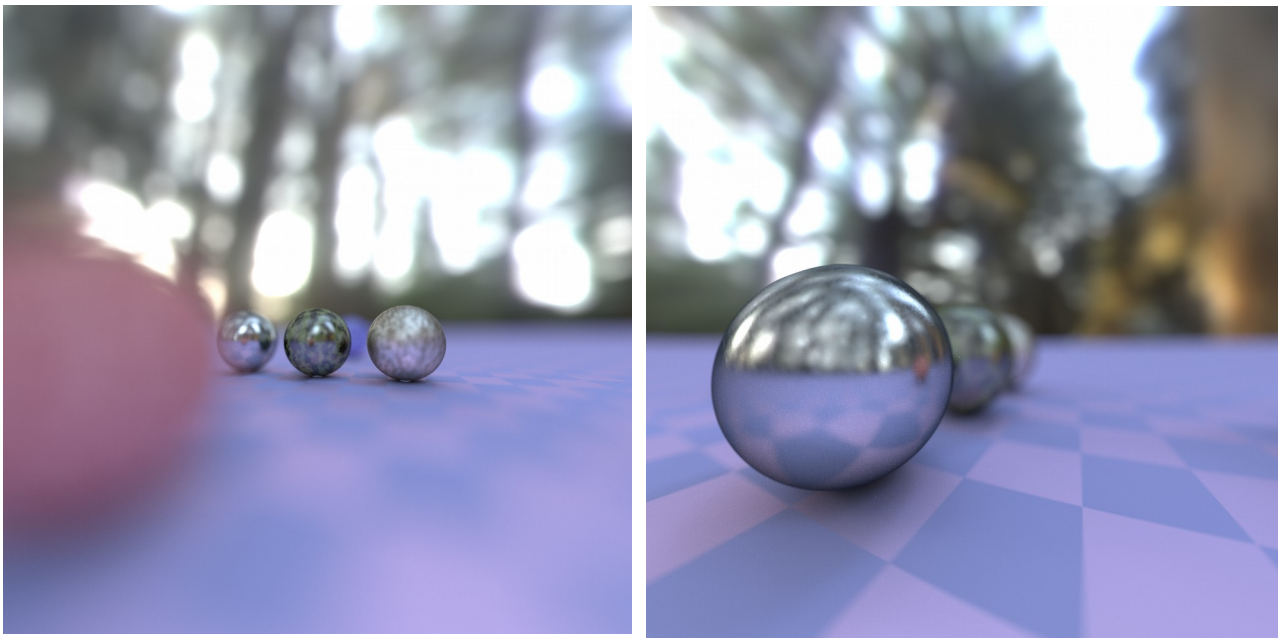


Depth of field

Le flou de profondeur s'obtient en simulant et en échantillonant aléatoirement une lentille. Comme vu en cours, cet effet nécessite de contrôler 2 valeurs : la distance au plan focal (le plan qui sera net) et l'ouverture (à quel point les rayons vont diverger si on s'éloigne du plan focal). Voir shéma ci-dessous.



A gauche, la génération des rayons classique (caméra pinhole). A droite : simulation de la lentille. L'idée est, pour un pixel donné, de trouver le point d'intersection avec le plan focal (le point situé à une certaine distance de la caméra en utilisant le rayon de base au centre du pixel). Ensuite, on échantillonne aléatoirement le disque autours de la position de la caméra (le rayon de ce disque est l'ouverture et détermine le taux de flou obtenu). La nouvelle position de la caméra sera le point du disque obtenu et le vecteur vue sera le vecteur de ce point vers le point d'intersection avec le plan focal. Note : la formule d'échantillonnage d'un disque avec 2 valeurs aléatoires peut se trouver ici : http://mathworld.wolfram.com/DiskPointPicking.html. Des exemples ci-dessous :



A rendre (deadline = vendredi 6 Mars)

Enregistrer et nommer votre pipeline prenom-nom-projet.gra. Dans un document prenom-nom-projet.pdf, vous ajouterez les rendus obtenus ainsi que les explications des différentes étapes / difficultés / choix que vous aurez rencontrés.

Envoyer votre pipeline ainsi que le rapport à benoit.arbelot@gmail.com (titre du mail : [GICAO SIA] Projet)