Projet de rendu

Explorateur de l'espace

Présenté à

M. Adrien Gruson

Dans le cadre du cours

MTI882

Sujets spéciaux II : Rendu d'images de synthèse photoréalistes

Groupe 01

Par l'équipe A composée de :

Vincent Gingras – GINV31079700

William Lebel – LEBW22029900

École de technologie supérieure

Le 12 décembre 2022

Table des matières

1 Introduction 1

1.1 Context 1
1.2 Images choisies 1
1.3 Image obtenue 2
1.4 Liste des fonctionnalités mises en œuvre 2

2 Rendu volumétrique homogène 4

2.1 Détails techniques 4
2.2 Validation 8
2.3 Difficultés rencontrées 17

3 Modèle de matériaux par micro-facettes 19

3.1 Détails techniques 19
3.2 Validation 20
3.3 Difficultés rencontrées 22

4 Amélioration légère du *plugin* Blender 23

4.1 Détails techniques 23
4.2 Validation 23
4.3 Difficultés rencontrées 24

5 Carte d'environnement 25

5.1 Détails techniques 25
5.2 Validation 27
5.3 Difficultés rencontrées 32

6 Meilleur échantillonnage des lumières 35

6.1 Détails techniques 36
6.2 Validation 36
6.3 Difficultés rencontrées 38

7 Gestion des masques 39

7.1 Détails techniques 39
7.2 Validation 39
7.3 Difficultés rencontrées 41

8 Autres difficultés rencontrées 43

8.1 Trou noir 43

8.2 Texture du Soleil invisible *43*

9 Répartition du travail 45

10 Conclusion 46

Références pour l'image 47

1 Introduction

1.1 Context

Le projet de rendu présenté dans ce rapport a été réalisé dans le cadre du cours MTI822 à l'ÉTS. À l'occasion d'une compétition de rendu entre les étudiants du groupe, nous devions générer une image de synthèse présentée à un jury. Cette image devait être créée à partir du moteur de rendu que nous avions développé tout au long de la session, et ce, en y ajoutant quelques fonctionnalités supplémentaires. De plus, l'image devait respecter le thème de la compétition qui était *Les technologies d'exploration*.

1.2 Images choisies

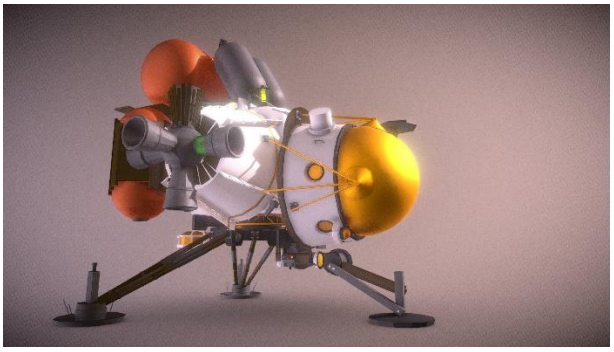
L'image dont nous nous sommes inspirés pour concevoir notre image est la suivante.

Source : https://www.reddit.com/r/outerwilds/comments/hnrqtk/recently_listened_to_andrew_prahlows_new_outer/

Il s'agit d'une image digitale inspirée du jeu vidéo Outer Wilds. Cette image nous a particulièrement inspirées avec le thème, car elle représente le côté grandiose, presque invraisemblable, mais parfois pavé de solitude de l'exploration et de ses technologies.

Cette image n'est pas photoréaliste. Cependant, nous avions le désir de reproduire quelque chose qui s'apparente à celle-ci avec notre moteur de rendu.

Une autre image utilisée est celle du vaisseau spatial, qui correspond à l'aspect technologique de notre idée et qui accompagnera l'astronaute dans notre image finale.



Source : https://sketchfab.com/3d-models/outer-wilds-ship-d72c6f84533c432ba068d08786b89cb0

1.3 Image obtenue

1.4 Liste des fonctionnalités mises en œuvre

Dans le cadre du travail, nous devions mettre en œuvre des fonctionnalités supplémentaires à notre moteur de rendu afin de générer notre image finale. Chaque fonctionnalité était notée sur un nombre de points dépendamment de sa difficulté :

- Fonctionnalité difficile : 8 points;
- Fonctionnalité moyenne : 4 points;
- Fonctionnalité facile : 1 point.

Le total des points à mettre en œuvre était 16. Voici les fonctionnalités que nous avons choisies pour répondre à cette contrainte :

- Rendu volumétrique homogène (8 points);
- Modèle de matériaux par micro-facettes (4 points);
- Amélioration légère du *plugin* Blender (1 point);
- Carte d'environnement (1 point);
- Meilleur échantillonnage des lumières (1 point);
- Gestion des masques (1 point).

2 Rendu volumétrique homogène

Pour notre rendu final, nous avions comme objectif d'ajouter une atmosphère autour de la planète principale aussi qu'autour de notre soleil. Malheureusement, nous n'avons pas réussi notre objectif. Nous avons tout de même réussi à obtenir quelque chose de fonctionnel en utilisant un intégrateur *path* normal, mais nous n'avons pas réussi à l'implémenter avec MIS. De ce fait, pour tous nos tests, nous nous sommes limités à la fonction de phase de Henyey-Greenstein et n'avons pas implémenté le support des couleurs différentes en RGB avec nos valeurs sigma.

À noter aussi qu'avec notre implémentation il n'est pas possible d'ajouter plus d'un volume à la fois dans la scène. Cette condition n'a pas été incluse pour faciliter le développement de cette fonctionnalité.

Malgré cela, voici un résultat obtenu en utilisant un intégrateur *path*. L'image de gauche ne comporte pas de volume. L'image de gauche utilise un volume sur l'ensemble de la scène.

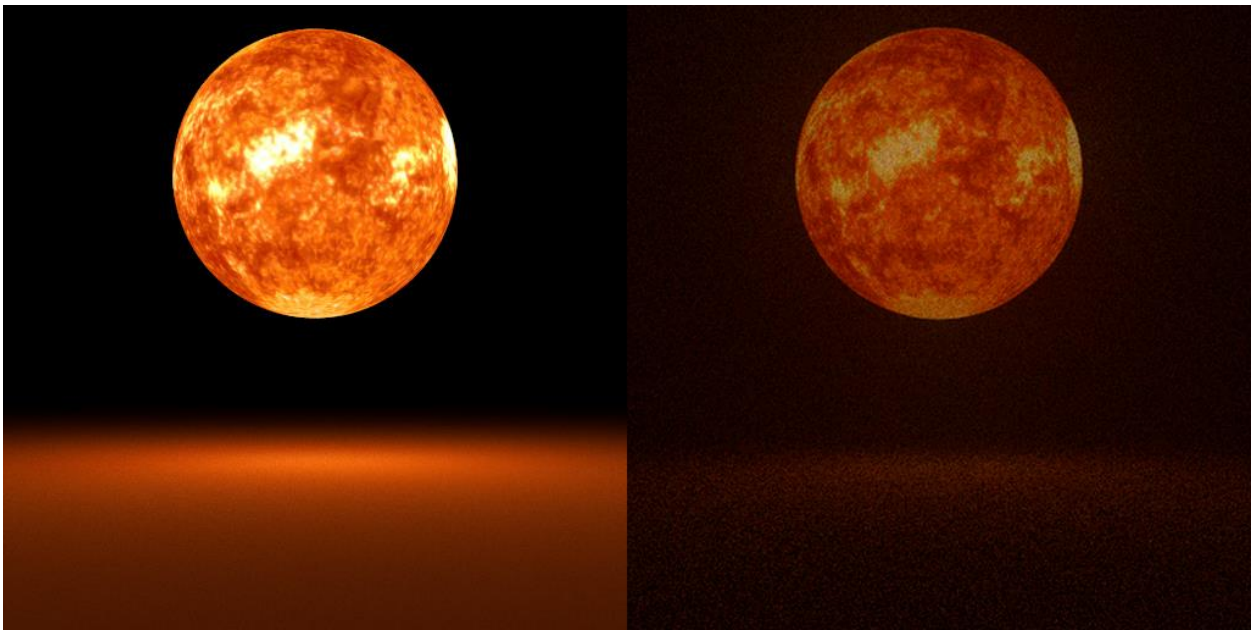


Figure 2.1 Comparaison entre une scène avec et sans volume

2.1 Détails techniques

2.1.1 Classe Medium et chargement des volumes

Pour implémenter un système de volume, la première étape a été d'implémenter une nouvelle classe nommée *Medium.h*. Cette classe se comporte similairement à un matériau. En effet, chaque forme (*shape*) créée comporte un nouveau pointeur vers un médium. Si le pointeur pointe vers un médium, alors la forme est considérée comme un volume. À l'inverse, s'il pointe vers un *nullptr*, alors la forme n'est pas un volume et est considérée comme une forme normale.

Cette nouvelle classe définit les paramètres associés à un volume : sa valeur σ_a, σ_s, σ_t, la fonction de phase, l'équation du *pdf* associé au volume, sa valeur alpha, etc.

Pour considérer une forme comme étant un volume, dans les fichiers de configurations de scène, un objet « medium » doit être appliqué à une forme. Nous avons modifié le processus de chargements des fichiers de configuration de scène pour gérer ce nouveau tag. Voici les étapes pour transformer une forme en volume.

Premièrement, comme pour les matériaux, une balise « mediums » qui va inclure tous les types de médiums possibles doit être créée :

```
"mediums": [
  {
    "name": "atmosphere",
    "type": "homogeneous"
  }
]
```

Le nom associé au médium permet à celui-ci d'être réutilisé sur plusieurs objets. Le type permet de définir le type de volume. Dans notre cas, comme nous n'avons implémenté que des volumes de type homogène, le seul choix possible est « homogeneous ».

Deuxièmement, une fois fait, on doit définir le médium à appliquer sur les formes voulues :

```
{
    "type": "mesh",
    "filename": "../../assets/cube.obj",
    "name": "monolith",
    "transform": [
        {
            "translate": [0, 0.5, 0]
        }, {
            "scale": [1, 9, 4]
        }
    ],
    "medium": "atmosphere"
}
```

En spécifiant un médium à notre forme, lors du chargement de la scène, un médium sera attribué à la forme.

À la base, lorsqu'une forme est créée, celle-ci est ajoutée dans une liste. Cette liste est alors utilisée pour déterminer s'il y a une intersection entre un rayon et un objet sur la scène. Toutefois, avec notre méthode, nous ne souhaitons pas considérer les rebonds sur les volumes. Nous voulons seulement savoir si un volume est traversé entre la position de départ du rayon et l'objet touché. Sachant cela, lorsque nous chargeons les différents volumes et formes, ces derniers sont placés dans des listes séparées : une pour les formes considérées comme « normal », une autre pour les formes considérées comme « volume ».

2.1.2 Intégrateur *path_vol*

Avant nos tentatives avec l'intégrateur MIS, nous avons commencé par concevoir un intégrateur simple réutilisant le code de *path.h*. Normalement, l'intégrateur *path* détermine si un rayon intersecte un objet sur la scène. Lorsque c'est le cas, on ajoute au *throughput* le poids du matériau, puis on régénère un rayon. Si on touche une lumière, on retourne son émission. Si aucun objet n'est touché, on multiplie le *throughput* par la valeur d'arrière-plan. Notre intégrateur *path_vol* réutilise cette logique.

Désormais, chaque fois qu'on lance un rayon, on détermine s'il va intersecter un volume ou non. Si aucun volume n'est intersecté, on procède de la même façon qu'avec l'intégrateur *path* de base. Si un volume est intersecté, il y a trois nouveaux cas possibles : le rayon intersecte une particule, le rayon traverse le volume et touche un objet ou le rayon traverse le volume et ne touche aucun objet. Si un volume est intersecté, de nouvelles choses doivent être prises en compte.

Note : avant toute chose, il est important de préciser qu'avec les volumes, un rayon peut débuter à l'intérieur d'un volume. Lorsque c'est le cas, le calcul de l'intersection avec les volumes est différent. Pour régler ce problème, nous avons décidé d'ajouter une valeur booléenne à la classe *Ray.h* qui indique si oui ou non le rayon débute dans un volume.

Revenons à l'intégrateur « path_vol ». Si l'on détecte qu'un rayon intersecte une particule à l'intérieur d'un volume, on applique le pseudo-code vu à la page 73 du PDF du cours 9.

```
z = near_surface(x, ω)
t = -log(1-rand()) / σt
if t < z:
    x += t * ω
    pdf_t = σt * exp(- σt * t)
    [ω', pdf_w'] = sample_fp(ω)
    return ft(t)/pdf_t * ((1-alpha) * Le(x, ω)
                          + alpha * fs(ω, ω') * trace_vol(x, ω') / pdf_w')
```

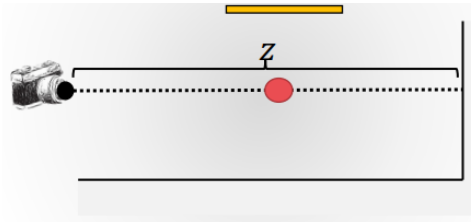


Figure 2.2 Pseudo-code utilisé pour le cas où un rayon intersecte une particule d'un volume

À cette étape, on commence par échantillonner la nouvelle direction du prochain rebond. Celle-ci peut être déterminée grâce à la fonction `sample()` qui est présente dans la classe *Medium.h*. On effectue ensuite les différents calculs nécessaires pour calculer le nouveau *throughput*. Finalement, on met à jour le rayon pour que celui-ci débute à la position de la particule touchée et se dirige vers la direction échantillonnée. On spécifie aussi que le rayon débute dans un volume. Ce code est implémenté dans la fonction `compute_li_inside_volume()` de notre classe « path_vol ».

Si on est au deuxième cas, c'est-à-dire si le rayon réussit à traverser un volume et touche un objet, on applique le pseudo-code suivant, provenant de la page 74 du PDF du cours 9 :

```
x += z * ω
P_z = exp(- σt * t)
[ω', pdf_w'] = sample_fs(x,ω)
return Tr(z)/P_z * (Le(x,ω)+trace_vol(x, ω')fs(x,ω,ω')cos(theta)/pdf_w')
```

Figure 2.3 Pseudo-code utilisé pour le cas où un rayon traverse un volume

Il est important de noter que si la surface touchée possède un matériau spéculaire, on applique le pseudo-code suivant :

```
Value = Tr(z) / P_z * sample_material->weight
```

Après avoir calculé le nouveau *throughput*, on met à jour le rayon. On procède alors comme à l'habitude : on définit la position de départ comme étant le point d'intersection sur la surface touchée et l'on définit la direction comme étant la direction de sortie du matériau sur la surface. Cependant, comme le rayon peut débuter à l'intérieur d'un volume (si la surface touchée est dans un volume), on doit alors indiquer si le rayon débute dans un volume ou non. Ce code est implémenté dans la fonction `compute_li_outside_volume()` du même intégrateur.

Finalement, si l'on est au dernier cas, c'est-à-dire si le rayon traverse un volume et ne touche aucun objet, on calcule la valeur de la transmittance dans le volume pour ensuite la multipliée par la couleur en arrière-plan. Ce code est implémenté dans la fonction `compute_li_background()` de l'intégrateur *path_vol*.

Toute cette logique nous permet d'obtenir l'image calculée au début de la section 2.

2.1.3 Intégrateur volumétrique avec MIS

Même si nous n'avons pas réussi à intégrer le volumétrique avec MIS, nous avons tout de même essayé certaines choses et avons obtenus des résultats proches. Malgré cela, voici les détails quant à notre implémentation dans MIS.

Dans *path_vol_mis*, nous débutons de la même façon qu'avec notre intégrateur *path_vol* : nous commençons par déterminer s'il y a une intersection avec un volume, et si c'est le cas nous séparons la suite des étapes en trois cas possibles : le rayon intersecte une particule, le rayon traverse le volume et touche un objet ou le rayon traverse un volume et ne touche rien.

Si l'on est au premier cas (une particule d'un volume est intersectée), on commence par calculer la nouvelle valeur du *throughput* en réutilisant la fonction `compute_li_inside_volume()`. Ensuite, on calcule la contribution avec deux nouvelles méthodes : `volume_sample_emitter()` et `volume_sample_bsdf()`. Ces deux méthodes réutilisent le même processus que les méthodes traditionnelles d'échantillonnage par émetteur et par BSDF, mais sont un peu modifiées. Elles seront présentées plus tard dans cette section.

Si l'on est au deuxième cas (le rayon traverse un volume et touche un objet), le processus est le même que dans le cas précédent, sauf que le nouveau *throughput* est obtenu par la fonction `compute_li_outside_volume()`. Cependant, dans ce cas-ci, comme on touche un objet, il faut déterminer si l'objet intersecté est un émetteur. Si c'est le cas, on ne met pas à jour le *throughput* et on ne lance pas un autre rebond.

Si on est au troisième cas possible, c'est-à-dire que le rayon traverse un volume, mais ne touche aucun objet, on retourne tout simplement la valeur de transmittance multipliée par la couleur de l'arrière-plan si l'on est à la profondeur 1 (s'il n'y a pas déjà eu de rebonds).

Comme nous l'avons précisé plus tôt, deux nouvelles méthodes ont été créées pour calculer le MIS lorsqu'il y a une interaction avec un volume : `volume_sample_emitter()` et `volume_sample_bsdf()`.

Dans `volume_sample_emitter()`, le processus est le même qu'avec `mis_sample_emitter()` : on échantillonne un point sur un émetteur pour ensuite calculer la contribution. Ce qui change avec la version volumétrique est que le point de départ *x* à partir duquel on fait l'échantillonnage change selon si une particule est touchée dans un volume ou si l'on touche une surface. Similairement, la valeur de G(*x*,*y*) change selon si *x* est sur une surface ou une particule, *f*(*x*,*y*,*z*) change si *x* est dans un volume et Le(*x*,*y*) change si *y* est dans un volume. Nous calculons aussi la valeur Tr(*x*,*y*) auquel nous multiplions à la contribution.

Dans `volume_sample_bsdf()`, les mêmes valeurs changent selon les mêmes cas (sauf pour G(x,y) qui n'est pas inclus). Nous calculons aussi Tr(*x*,*y*) auquel nous multiplions à la contribution.

2.2 Validation

2.2.1 Validation de *path_vol*

Il est très difficile de prouver que notre implémentation du rendu volumétrique fonctionne correctement avec notre intégrateur normal. Déjà, nous avons eu beaucoup de difficulté à trouver des valeurs σ_a et σ_s sur lesquels nous baser. De même, nous n'avons pas d'images de références pour ce qui est de la comparaison des images. Toutefois, nous avons utilisé notre bon sens et avons conçu de nombreuses scènes pour valider l'effet du volumétrique dans différentes conditions.

Premièrement, nous avons conçu une scène dans laquelle une sphère était intégrée dans un volume sphérique. Les valeurs utilisées sont les suivantes : $\sigma_a = 1$ et $\sigma_s = 0.1$.

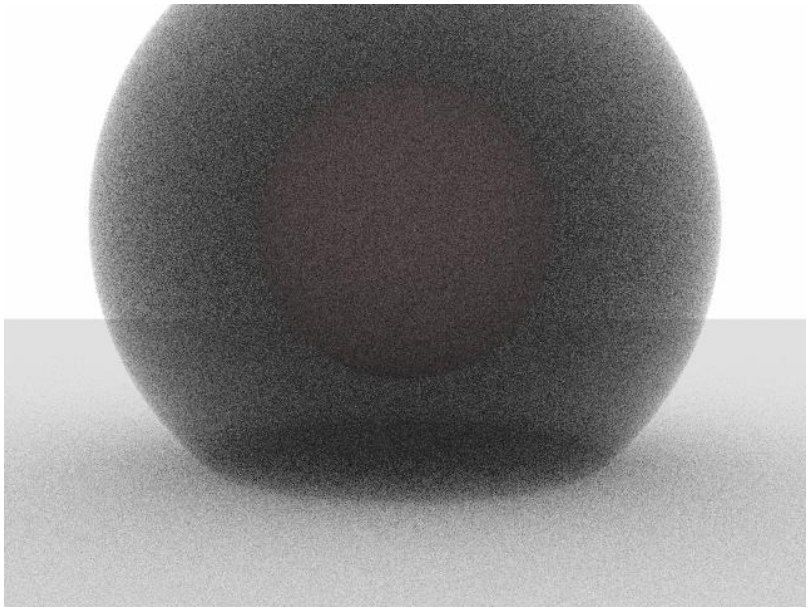


Figure 2.4 Scène de test volumétrique 1 - Démonstration 1

Voici la même scène, mais avec les valeurs suivantes : $\sigma_a = 0.1$ et $\sigma_s = 1.0$.

Figure 2.5 Scène de test volumétrique 1 - Démonstration 2

Comme on peut le voir, changer les valeurs absorption et de diffusion semble affecter la scène de la façon attendue : augmenter la valeur d'absorption assombrit la zone, car celle-ci est absorbée par les particules, augmenter la diffusion rend plus difficile la vision au travers la zone. Nous avons ensuite changé les valeurs de σ_a et σ_s pour des valeurs plus mélangées pour vérifier le comportement du volume sur cette même scène. L'image de gauche utilise les valeurs $\sigma_a = 0.1$ et $\sigma_s = 0.4$ alors que l'image de droite utilise $\sigma_a = 0.34$ et $\sigma_s = 0.28$.

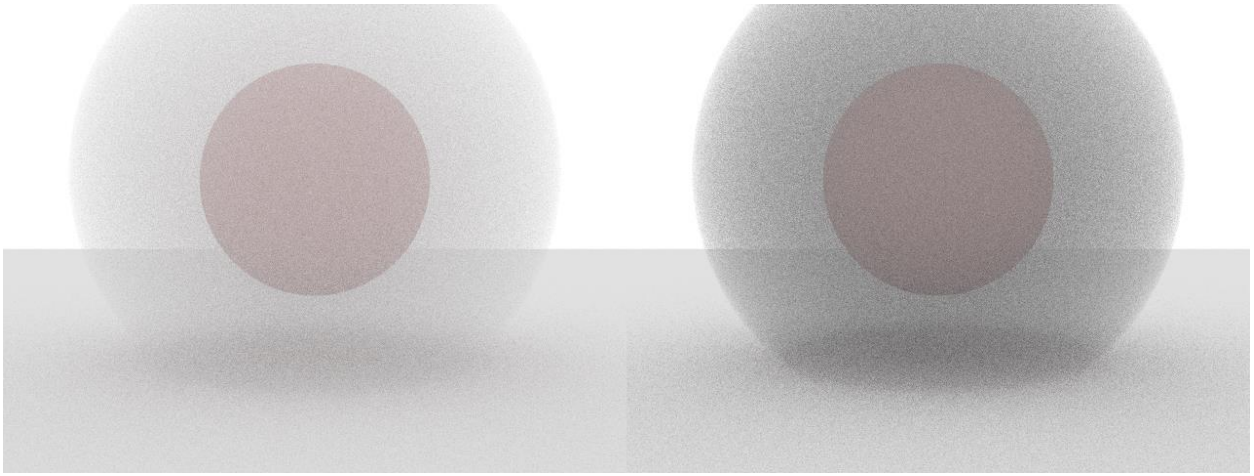


Figure 2.6 Exemples supplémentaires de la scène de test volumétrique 1

Mixer les valeurs semble ainsi donner des résultats concluants. Nous avons ensuite généré une autre scène simple avec comme valeur $\sigma_a = 0.02$ et $\sigma_s = 0.7$. Le résultat obtenu est le suivant.

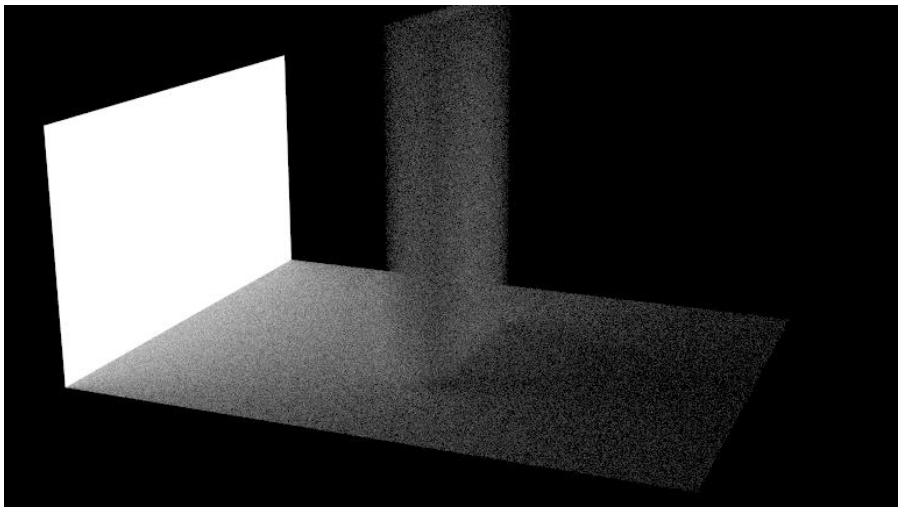


Figure 2.7 Scène de test volumétrique 2 avec un volume

Comparons le résultat obtenu avec l'image originale.

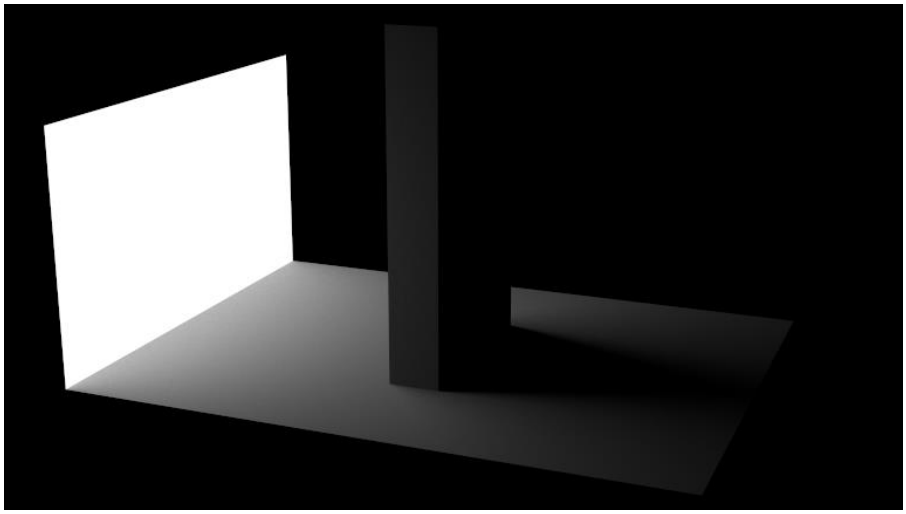


Figure 2.8 Scène de test volumétrique 2 sans volume (image originale)

En comparant les deux images, on peut remarquer deux choses : premièrement, l'ombre projetée par le prisme est atténuée par le volume au lieu d'être totalement noir. Deuxièmement, on peut remarquer qu'au bas du prisme, la lumière est beaucoup plus présente, indiquant ainsi qu'une certaine partie de la lumière a réussi à traverser le volume et à se refléter sur celui-ci. On peut alors affirmer que ce test est en quelque sorte concluant.

Comme troisième scène de test, nous avons déplacé et agrandi le prisme pour que l'émetteur intègre celui-ci. Le but de cette scène est de vérifier comment interagit la lumière si celle-ci est située à l'intérieur d'un volume. Avec $\sigma_a = 0.02$ et $\sigma_s = 0.2$, voici le résultat obtenu.

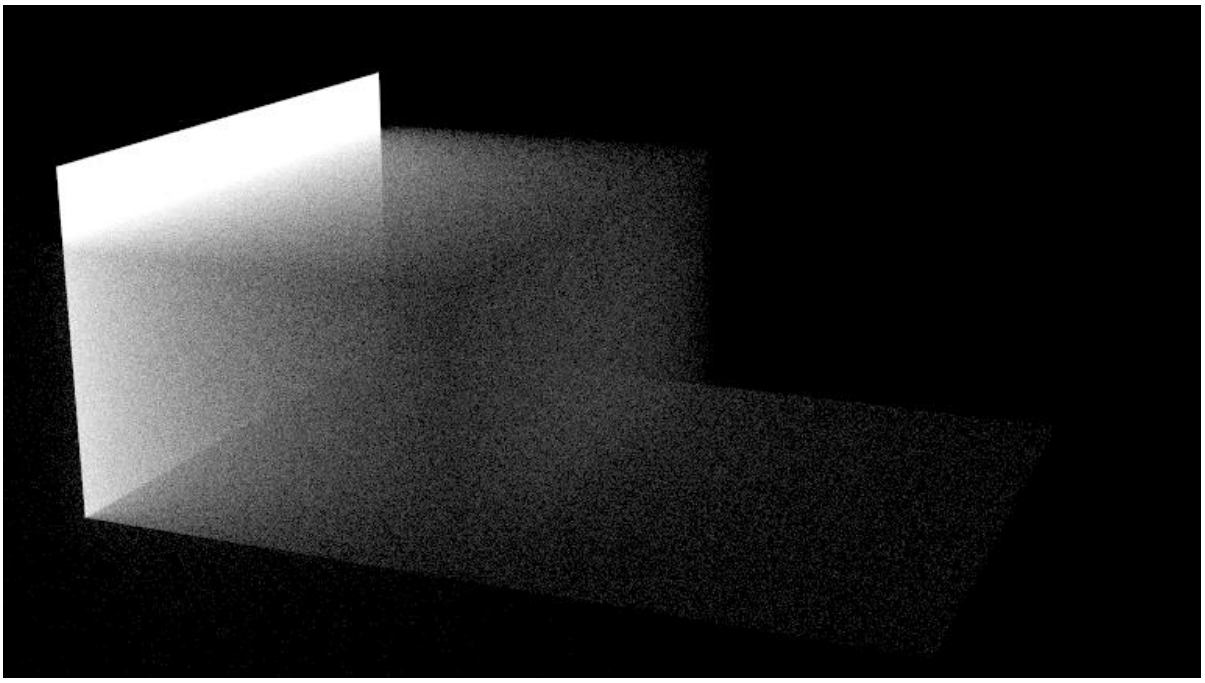


Figure 2.9 Scène de test volumétrique 3

Comme on peut le voir, la lumière a beaucoup plus de difficulté à se rendre sur le côté droit du plan. On peut donc émettre l'idée que la lumière traverse correctement un volume, même si celle-ci est située dans un volume. Si on regarde directement l'émetteur, on peut remarquer le volume atténue aussi la lumière sortante. Plus la distance que doit traverser le rayon à travers le volume avant d'atteindre l'émetteur est grande, plus la lumière visible s'atténue. On peut alors estimer que l'interaction d'un volume avec un émetteur est valide.

Comme quatrième test, nous avons essayé d'englober une partie scène reconnue dans un volume. Cette scène a pour but de valider deux choses : l'interaction d'un volume avec un ensemble d'objets et l'interaction d'un volume avec des matériaux spéculaires. Avec les valeurs $\sigma_a = 0.1$ et $\sigma_s = 0.7$, voici le résultat obtenu :

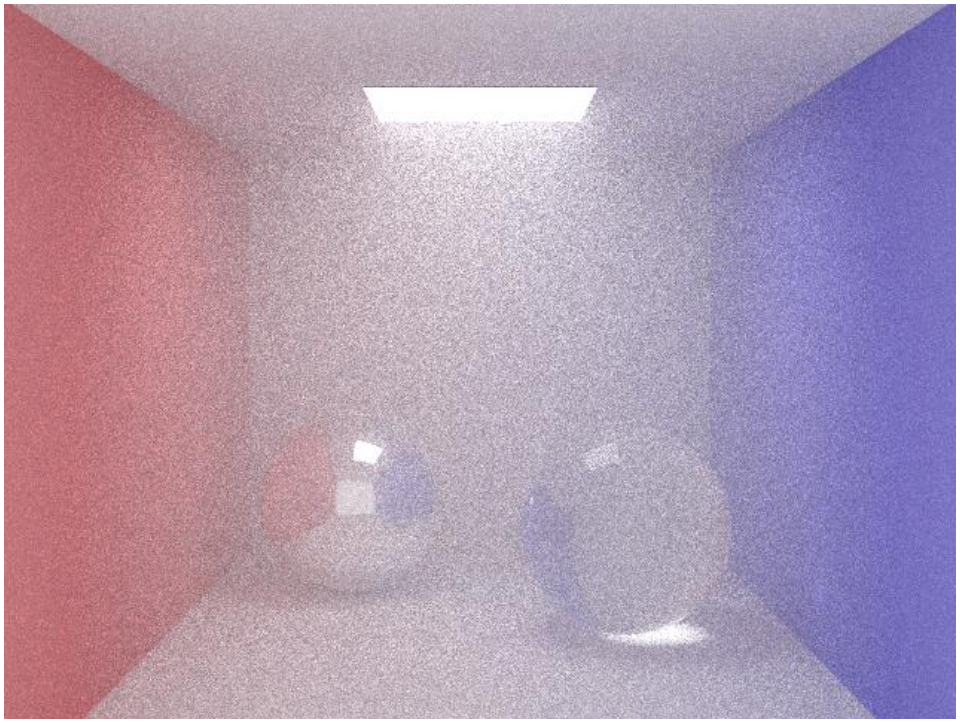
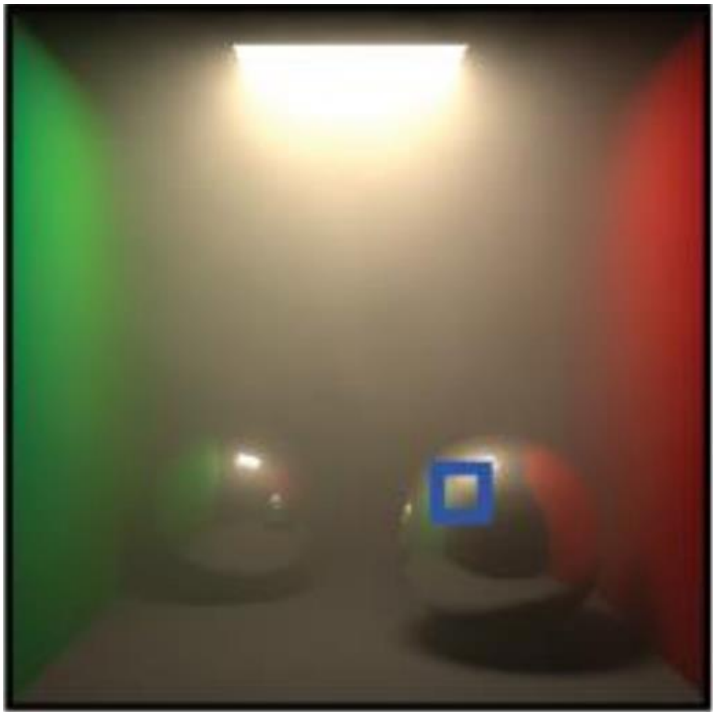


Figure 2.10 Scène de test volumétrique 4

Pour valider cette scène, nous nous sommes basés sur une image présente les PDF du cours 9 à la page 28.



Source : https://cs.dartmouth.edu/wjarosz/publications/jarosz08radiance.pdf

Cette image applique un volume sur la même scène que nous utilisons. Même si nous n'utilisons pas les mêmes valeurs pour le volume, il reste possible de comparer les résultats pour estimer si oui ou non notre image est valide. Déjà, en regardant attentivement notre image, on peut remarquer que le faisceau de lumière est quelque peu présent dans le volume. De plus, outre le fait que notre bille de droite utilise du verre, la réflexion semble agir de la même façon que dans l'image de référence (outre le fait que nous utilisons un arrière-plan blanc lieu de noir). La réfraction du verre semble quant à elle aussi inchangée. On peut donc affirmer que ce test est concluant.

Comme dernier test, nous avons réutilisé cette scène, mais avons ajouté une sphère devant les autres sphères, et nous avons augmenté énormément la valeur de diffusion. Voici les données utilisées : σ_a = 1 et σ_s = 10. Le résultat obtenu est le suivant :

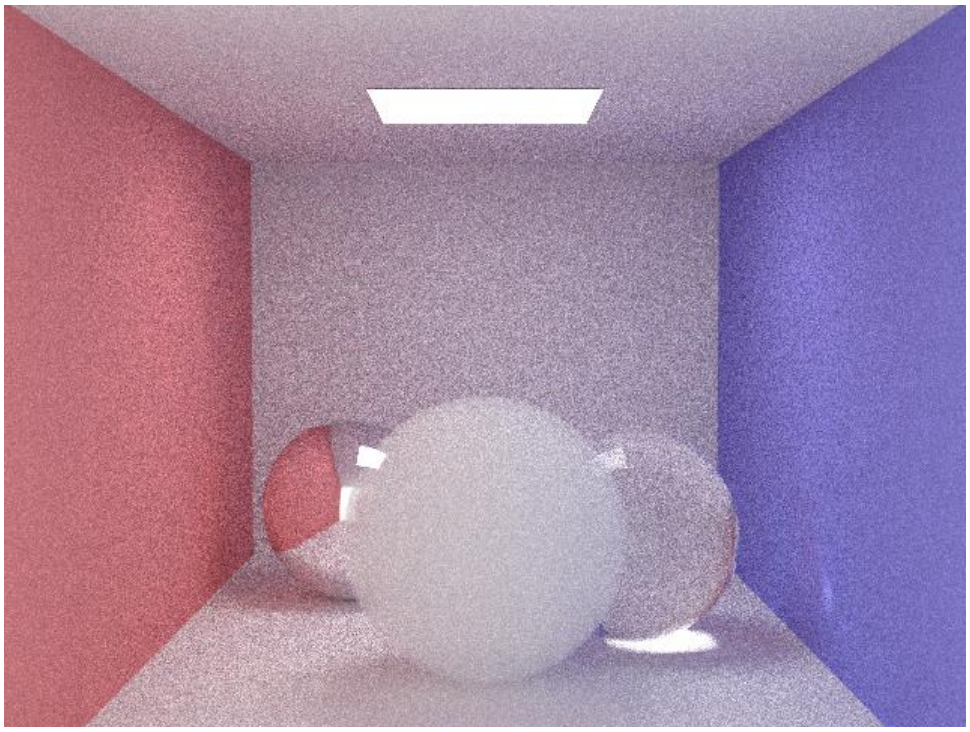


Figure 2.11 Scène de test volumétrique 5

Comme attendu, le volume est beaucoup plus dense. La lumière a beaucoup de difficulté à traverser le volume. Point intéressant, l'ombre sous la sphère vaporeuse est moins sombre que celles des deux billes. Cela peut être expliqué par le fait qu'une certaine quantité de lumière réussit tout de même à traverser la sphère.

Comme mentionné au début de cette sous-section, on ne peut affirmer si oui ou non notre implémentation dans *path_vol* fonctionne correctement. Toutefois, comme les nombreux tests que nous avons effectués semblent valides, on peut estimer que notre implémentation fonctionne et que même si celle-ci est faussée, elle aurait tout de même pu nous générer un rendu de notre scène proche du résultat attendu si nous avions fait un rendu avec un très grand taux d'échantillonnage et une profondeur élevée.

2.2.2 Validation de *path_vol_mis*

Malgré le fait que notre intégrateur *path_vol* semble fonctionnel, nous n'avons malheureusement pas réussi à implémenter le volumétrique avec MIS. Cependant, nous avons tout de même réussi à obtenir des résultats. Cette section ne servira donc pas à valider notre intégrateur, mais elle servira plutôt à présenter l'état actuel de notre système et à théoriser quelles sont les erreurs potentielles qui nous ont empêchés d'obtenir un résultat concluant.

Si l'on génère notre deuxième scène de test du volumétrique, voici le résultat que nous obtenons :

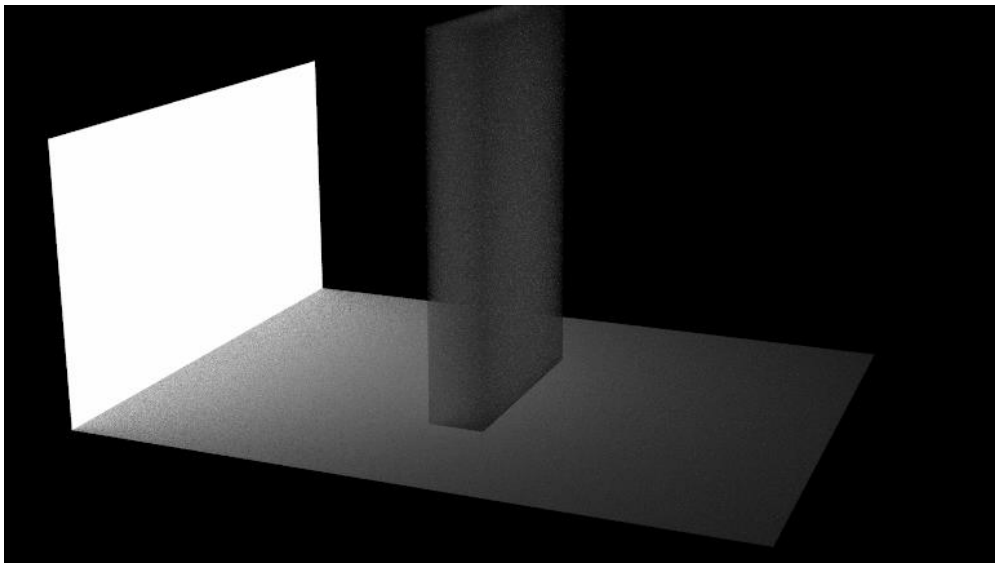


Figure 2.12 Scène de test volumétrique 2 avec notre intégrateur *path_vol_mis*

À première vue, l'image générée semble concluante. Cependant, si on la compare avec notre image générée par *path_vol* (voir section 2.2.1), on peut remarquer plusieurs choses : le prisme est plus opaque, l'ombre projetée au sol n'est pas présente et la couleur du prisme semble être plus sombre en général. Aussi, en augmentant la luminosité de l'arrière-plan, deux nouveaux problèmes nous sautent aux yeux.

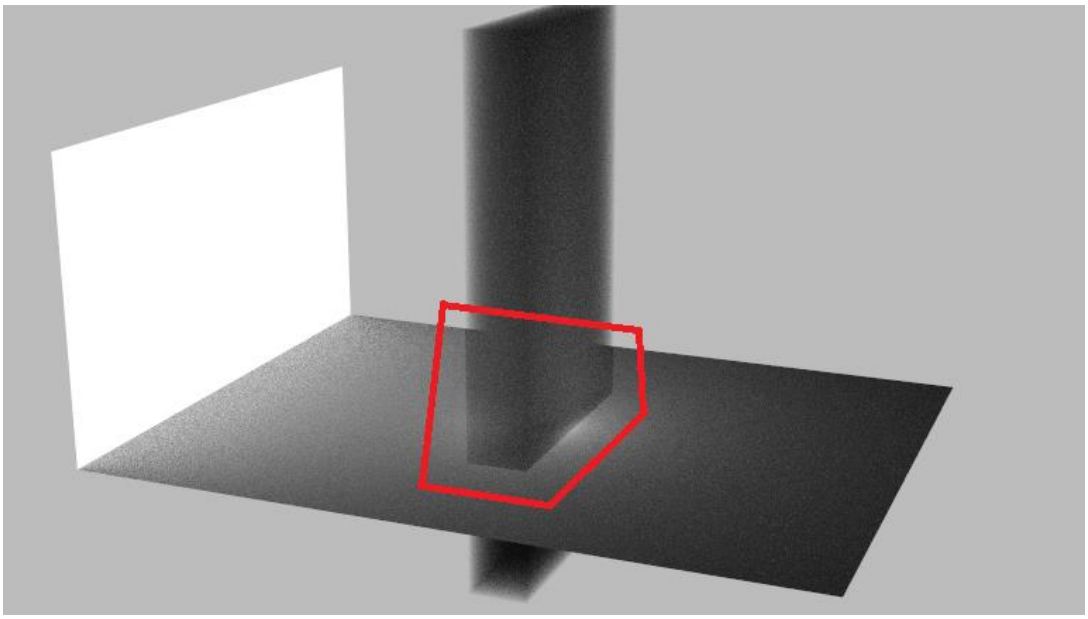


Figure 2.13 Scène de test volumétrique 2 avec notre intégrateur *path_vol_mis* en utilisant un arrière-plan gris

Si on regarde bien la base du prisme, on remarque le sol semble beaucoup plus éclairé qu'il ne le devrait. De plus, la partie du prisme qui couvre le sol ne s'atténue pas autant sur les côtés que lorsqu'on

le compare avec les parties qui couvrent l'arrière-plan. Il est difficile d'expliquer la cause de ce problème, mais elle semble être créée lorsqu'un rayon réussit à traverser un volume et à toucher une surface. Potentiellement que régler ce problème permettrait aussi de corriger le problème où la lumière ne réussit pas à traverser correctement le volume. Pour le sol qui semble plus éclairé, cela peut être dû à une mauvaise interaction avec l'arrière-plan.

En générant notre troisième scène de test, un autre problème assez apparent survient :

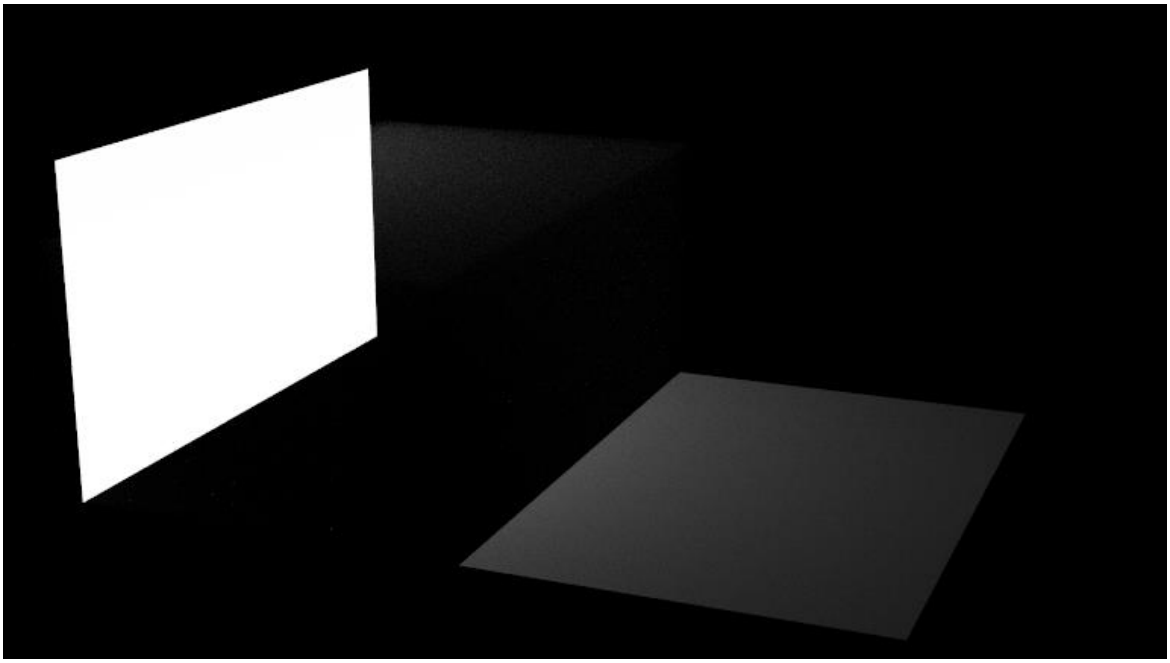


Figure 2.14 Scène de test volumétrique 3 avec notre intégrateur *path_vol_mis*

Le volume semble très peu visible, et la lumière ne semble pas se propager correctement à travers le volume (le côté droit du plan est assez clair). De même, la lumière directe de l'émetteur n'est pas atténuée par le volume. Encore une fois, ce problème est probablement créé dû à une erreur d'interaction entre un rayon traversant un volume et touchant une surface. En effet, si l'on regarde le haut de l'image, on peut remarquer qu'une partie du volume est visible et que la lumière semble se propager en partie.

Finalement, si nous analysons notre cinquième scène, un autre problème survient.

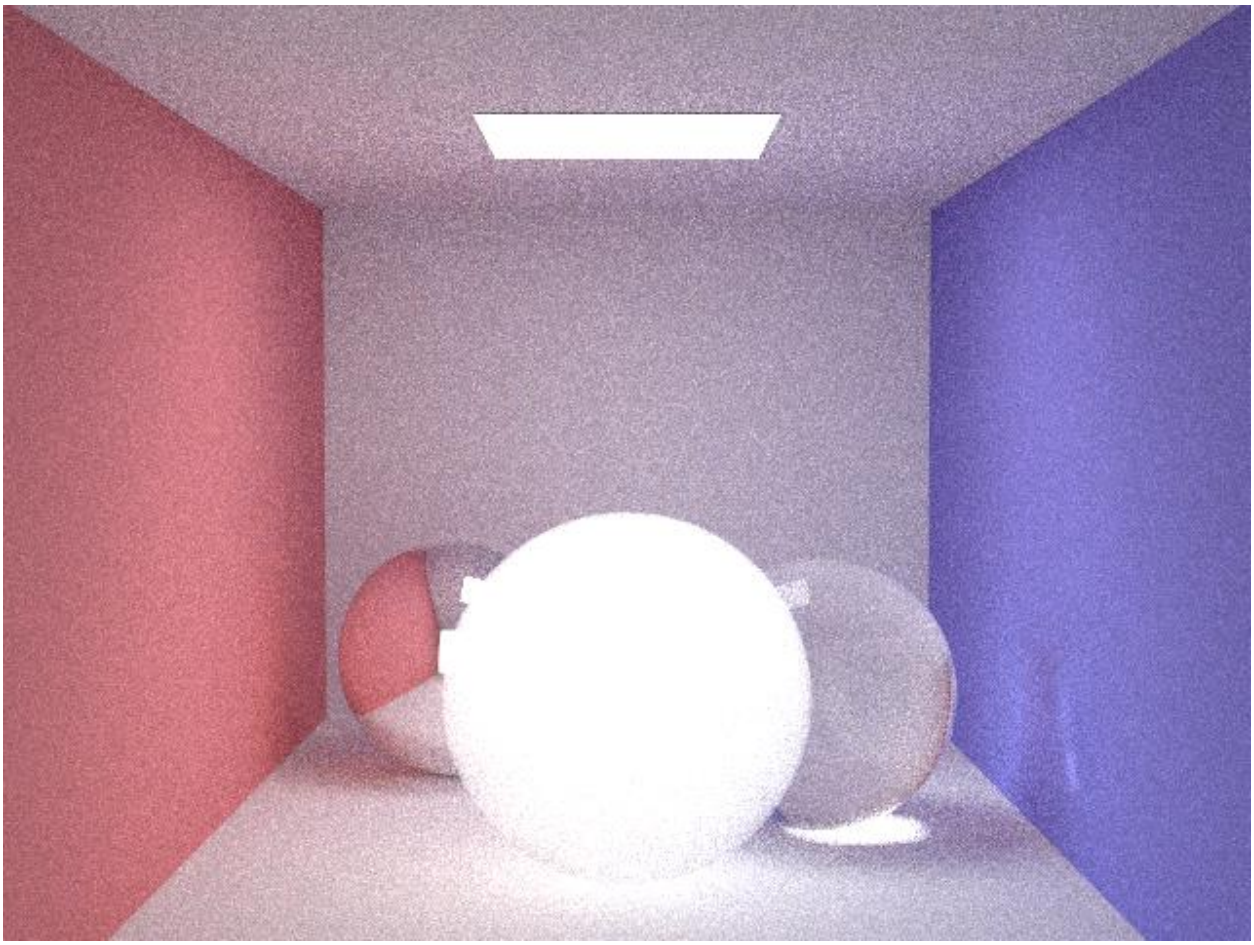


Figure 2.15 Scène de test volumétrique 5 avec notre intégrateur *path_vol_mis*

Dans ce cas-ci, notre volume est beaucoup plus clair qu'il ne le devrait. Considérant que l'arrière-plan est blanc, et comme indiqué plus tôt, cela peut être dû à une mauvaise gestion des calculs avec l'arrière-plan. Une autre raison potentielle, dans cette scène existe des matériaux spéculaires.

Comme on l'a vu, notre intégrateur *path_vol_mis* n'est pas tout à fait au point. Il est difficile d'expliquer pourquoi ces erreurs surviennent. Cependant, au vu des résultats obtenus, une mauvaise interaction avec l'arrière-plan ou une erreur de calcul dans le cas où un rayon traverse un volume et touche une surface semble être les sources d'erreur les plus plausibles.

2.3 Difficultés rencontrées

Lors de l'implémentation de cette fonctionnalité, nous avons rencontré un très grand nombre de problèmes nous empêchant de continuer. La plupart étaient dus à des erreurs plutôt simples à résoudre. Toutefois, notre plus gros problème a été d'essayer de comprendre comment implémenter le volumétrique avec le MIS. Concevoir la classe *medium.h*, générer les volumes et faire la détection des intersections avec ces mêmes objets était plutôt simple. Nous avons seulement commencé à avoir de nombreux problèmes lorsque nous avons tenté d'implémenter le tout avec MIS.

Une des raisons rendant la tâche difficile était d'essayer de comprendre comment fonctionnait le volumétrique en même temps d'essayer de comprendre comment implémenter le volumétrique avec un algorithme de tracé de chemin et le MIS.

Pour réussir à résoudre ce problème, nous avons commencé par concevoir un intégrateur simple réutilisant le code provenant de notre algorithme *path.h*. Il était alors plus facile de comprendre comment fonctionnait le volumétrique étant donné que nous n'avons qu'à nous soucier de problèmes plus simples, c'est-à-dire comment un rayon doit interagir avec un volume. Comme on peut le voir dans la validation de notre intégrateur *path_vol*, nous avons réussi à obtenir des résultats qui semblent concluants.

Une fois cela fait, le nouveau défi était d'implémenter ce que nous avions dans *path_vol* avec MIS. Comme nous avions déjà réussi à concevoir un intégrateur sans MIS, il était alors possible de réutiliser ce que nous avions fait pour préparer notre nouvel intégrateur. Ce faisant, il ne nous restait plus qu'à mettre à jour l'intégrateur *path_mis* pour supporter nos modifications.

Malgré cela, nous n'avons tout de même pas réussi à terminer notre implémentation du volumétrique avec MIS dû à la complexité et par manque de temps. Toutefois, par notre expérience, on peut tout de même affirmer qu'implémenter un intégrateur de base est primordial pour la compréhension.

3 Modèle de matériaux par micro-facettes

Dans le cadre du projet, nous voulions implémenter un modèle de matériaux plus réaliste afin de donner de la fidélité à notre image. Nous avons choisi de procéder avec les micro-facettes, qui donnent une apparence plus réelle aux objets de la scène qu'avec le modèle *Phong* utilisé dans le modèle de rendu.

Par manque de temps avec le projet, nous n'avons pas pu finaliser l'implémentation du modèle de matériau micro-facette. Celui-ci ne se retrouve donc pas dans l'image finale. Les sections suivantes présentent tout de même notre avancement sur cette fonctionnalité ainsi que la validation des éléments qui fonctionnent et les difficultés que nous avons rencontrées.

3.1 Détails techniques

Nous avons choisi d'implémenter le modèle de micro-facettes standard, dont la forme de la BRDF est :

$$f_r = \frac{F_r * D * G}{4|(\overrightarrow{w_i} * \boldsymbol{n}) * (\overrightarrow{w_o} * \boldsymbol{n})|}$$

Dans cette formule, f_r représente la BRDF que nous cherchons, F_r le coefficient de Fresnel du matériau, D la distribution de facettes, G l'assombrissement du matériau dû aux ombrages des facettes, $\boldsymbol{n}$ la normale de la surface, $\overrightarrow{w_i}$ le rayon incident, dans l'espace local, et $\overrightarrow{w_o}$, le rayon sortant, dans l'espace local. Il est également pertinent de mentionner que le dénominateur représente le Jacobien du modèle.

À l'intérieur des équations, nous retrouvons souvent le vecteur *h*, qui représente la mi-chemin entre $\overrightarrow{w_i}$ et $\overrightarrow{w_o}$. D'autres propriétés du matériau micro-facettes sont la rugosité $\boldsymbol{\alpha}$ et le coefficient de réfraction η.

Pour implémenter ce nouveau matériel, nous avons ajouté la classe *Microfacet* au programme.

Le modèle que nous voulions utiliser pour le coefficient de Fresnel F_r est l'approximation de Schlick, que nous avions déjà utilisé pour nos matériaux diélectriques. Cette approximation offre une manière peu coûteuse et assez réaliste pour approximer la valeur du coefficient de Fresnel.

Pour la distribution des facettes D, nous voulions utiliser le modèle GGX/Trowbridge-Reitz. Ce modèle offre une forme anisotrope, avec une valeur de $\boldsymbol{\alpha}$ sur deux axes. Cependant, nous ne voulions qu'implémenter la partie isotopique du modèle.

Pour le terme de masquage et d'ombrage G, nous souhaitions utiliser l'approximation de Torrance-Sparrow qui limite la complexité du matériau.

Pour échantillonner une direction par rapport au matériau, nous avons décidé d'y aller avec un échantillon par rapport à la distribution des facettes D. C'est en effet le terme qui a le plus d'impact sur l'image finale, et donc qui réduira le plus la variance s'il contrôle l'échantillonnage. Ainsi, nous avons ajouté les méthodes *sample_ggx* et *pdf_ggx* dans le fichier *vec.h* (où les autres méthodes

d'échantillonnage se retrouvent). Celles-ci permettent d'échantillonner une direction par rapport à notre fonction de distribution de facettes, c'est-à-dire GGX.

3.2 Validation

Bien que ce ne soit pas tout le modèle qui fonctionne, nous avons des éléments à présenter dans cette section du rapport.

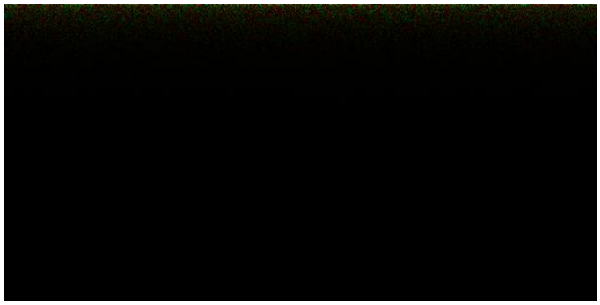
3.2.1 Échantillonnage sur GGX

Tout d'abord, l'échantillonnage par rapport à GGX peut être validé. Pour tester cet échantillonnage, nous avons utilisé le même code qui nous permettait, dans le cadre du laboratoire 3 de ce cours, de tester l'échantillonnage sur les certaines formes. Nous avons testé l'échantillonnage par rapport à GGX pour des valeurs de $\boldsymbol{\alpha}$ de 0.2 et de 0.8. Voici les résultats en ligne de commande.

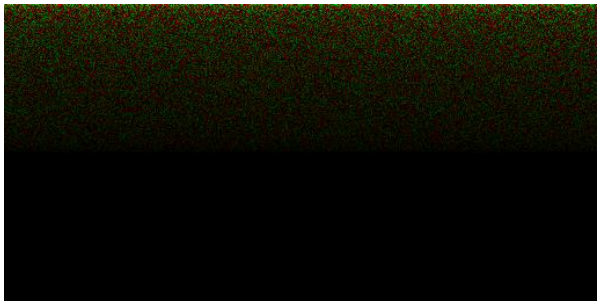
```
=====================================
Tache 2: Distributions simples
=====================================
Echantillonnage distribution GGX (roughness=0.2)
 100% completed: [####################] (520ms/520ms)
 100% completed: [####################] (1.844s/1.844s)
[2022-12-12 14:18:14.538] [info] Integrale de la pdf (devrais etre proche de 1): 1.000043483335922
[2022-12-12 14:18:14.542] [info] 95% percentile de la pdf: 7.940880026270961
[2022-12-12 14:18:14.542] [info] La difference entre l'histogramme et la pdf (devrais etre proche de 0): -6.021459880735385e-05
Echantillonnage distribution GGX (roughness=0.8)
 100% completed: [####################] (906ms/906ms)
 100% completed: [####################] (1.716s/1.716s)
[2022-12-12 14:18:17.422] [info] Integrale de la pdf (devrais etre proche de 1): 0.9999968098406116
[2022-12-12 14:18:17.423] [info] 95% percentile de la pdf: 0.49732415012596165
[2022-12-12 14:18:17.424] [info] La difference entre l'histogramme et la pdf (devrais etre proche de 0): -0.00021529323258446536
```

On remarque dans l'image précédente que, dans les deux cas, la *pdf* s'intègre à près de 1, ce que l'on cherche. De plus, on peut voir que la différence entre la *pdf* et l'histogramme, qui représente les échantillons réels, est très faible dans les deux cas, ce qui également un bon signe.

Si on compare les deux images (la *pdf* et l'histogramme), on observe les résultats suivants :



α = 0.2



α = 0.8

Dans les deux cas, on peut constater que la concentration de points rouges et de points verts est à peu près la même. La distribution est donc suivie.

3.2.2 Échantillonnage par rapport au matériau

Malheureusement, c'est durant l'échantillonnage par rapport aux matériaux que nous avons été obligés de prendre la décision de ne plus travailler sur ce modèle. Plus de détails seront expliqués à la section 3.3.

3.2.3 Rendu final

Pour comparer l'image finale avec ce qui devrait être attendu, nous avons créé une scène dans Blender en utilisant le moteur de rendu Cycles. Voici le résultat :

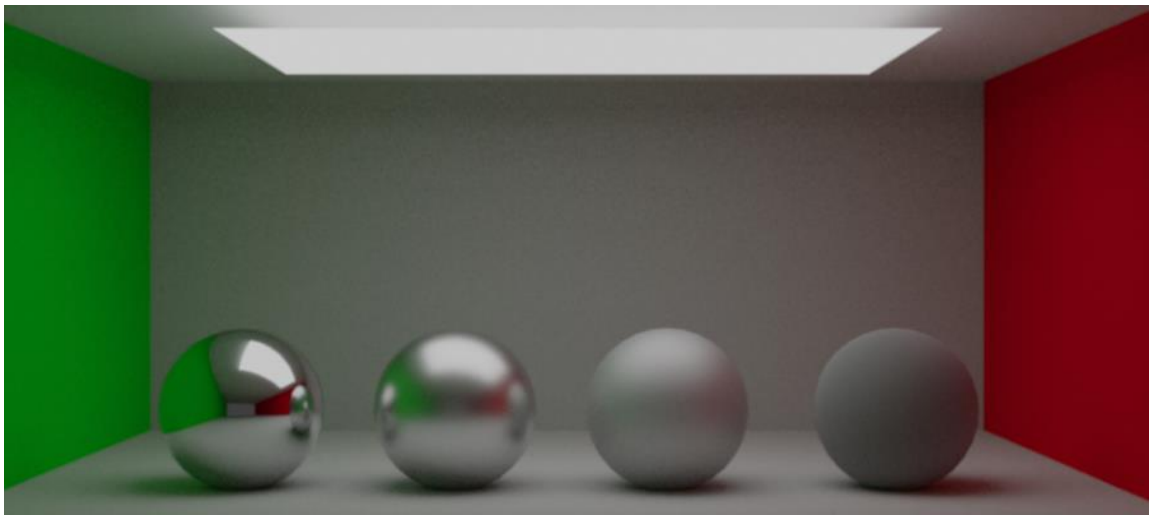


Image de notre scène générée à l'aide du moteur de rendu Cycles de Blender

Dans la scène précédente, les sphères ont une valeur α de gauche à droite de 0.00, 0.33, 0.66 et 1.00.

Lorsque nous avons arrêté de travailler sur le modèle de micro-facettes, l'image rendue était la suivante :

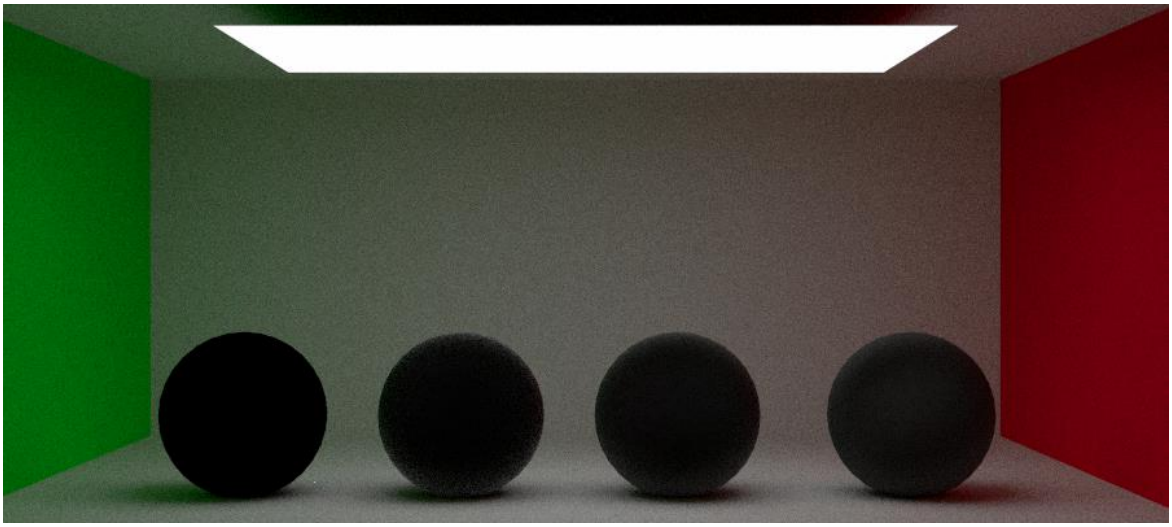


Image de notre scène générée à l'aide de notre moteur de rendu

Il est possible de remarquer, dans l'image précédente, que plus la valeur de α est faible, plus la sphère absorbe la lumière au lieu de la réfléchir. Notre hypothèse est une erreur avec la fonction d'ombrage ou avec l'implémentation de GGX dans notre évaluation du matériau.

3.3 Difficultés rencontrées

3.3.1 Difficulté à trouver un modèle micro-facettes

Étant donné que seulement quelques parties éparses du modèle de micro-facettes étaient données dans la documentation du cours, nous ne savions pas par où commencer pour ce qui est de l'implémentation. D'autant plus, beaucoup de modèles trouvés en ligne ne suivent pas les mêmes conventions et les mêmes algorithmes que ceux suivis dans le cours.

Nous sommes donc parties dans la mauvaise direction en élaborant le modèle et nous n'avons pas été en mesure de corriger le tir pour la date limite de la remise du projet.

Cependant, l'hypothèse est que ce problème vient de l'évaluation du matériau. Il ce peut que le terme d'ombrage ou le terme de distribution des facettes GGX soient erronés dans notre code.

4 Amélioration légère du *plugin* Blender

Dans le cadre du travail, nous avions accès à un *plugin* Blender pour exporter les scènes créées à partir du logiciel *Blender* dans un format lisible par notre moteur de rendu. Dans le cadre du projet, nous avons donc implémenté deux changements au *plugin* pour subvenir à nos besoins de projets.

1. Ajout d'un nom de scène;
2. Exportation des matériaux micro-facettes.

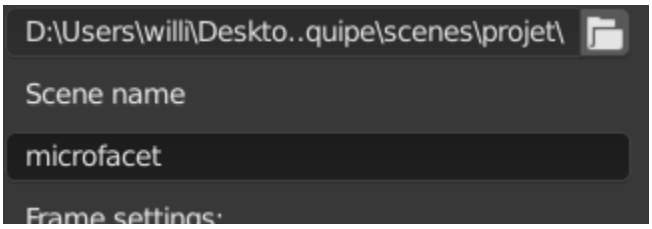
Pour cette première fonctionnalité, l'idée est venue du fait qu'à la base, le *plugin* exportait les scènes avec un nom qui était difficile de suivre (ex. : 00001.json). Nous souhaitions donc exporter nos scènes avec des noms déjà représentatifs pour que la traçabilité se fasse mieux.

Pour cette deuxième fonctionnalité, nous voulions être capables d'exporter des matériaux micro-facettes directement avec le *plugin* pour notre scène de test des micro-facettes.

4.1 Détails techniques

4.1.1 Ajout d'un nom de scène

Pour l'implémentation de la première solution, nous avons utilisé l'API de Blender pour créer un nouveau champ « Scene name » dans le dialogue d'exportation de la scène. Pour ce faire, nous nous sommes basés sur les champs déjà créés dans le cadre de ce *plugin*. Voici le champ dans la fenêtre :



Puis, lorsque l'on crée la scène, le nom donné est passé en paramètre au flux d'exécution qui va créer le bon nom de fichier ainsi que la bonne hiérarchie de répertoire pour les dossiers de *mesh* et de *textures*.

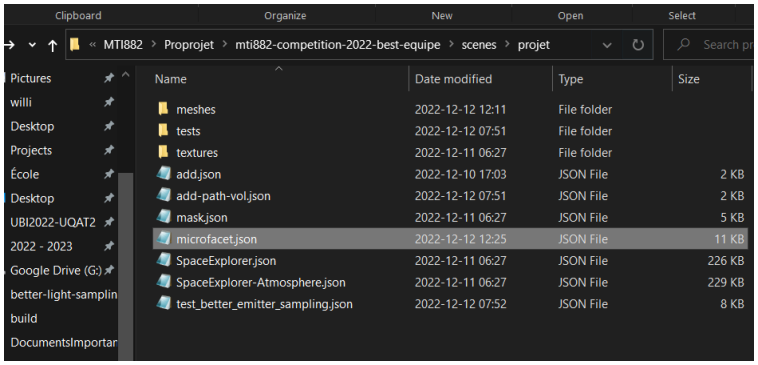
4.1.2 Exportation des matériaux micro-facettes

Pour l'exportation des matériaux micro-facettes, nous nous sommes basés sur ce qui était déjà fait pour l'exportation des autres matériaux. Lorsque l'utilisateur sélectionne le nœud *GlossyBSDF* dans Blender, cela indique à notre *plugin* qu'il doit créer un matériau micro-facettes.

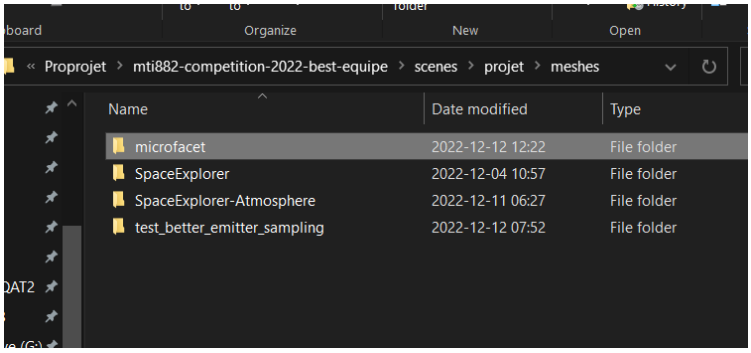
4.2 Validation

4.2.1 Aout d'un nom de scène

En exécutant l'exportation de la scène *microfacet* présentée à la section suivante, on remarque que le fichier de scène a été créé à la bonne place.

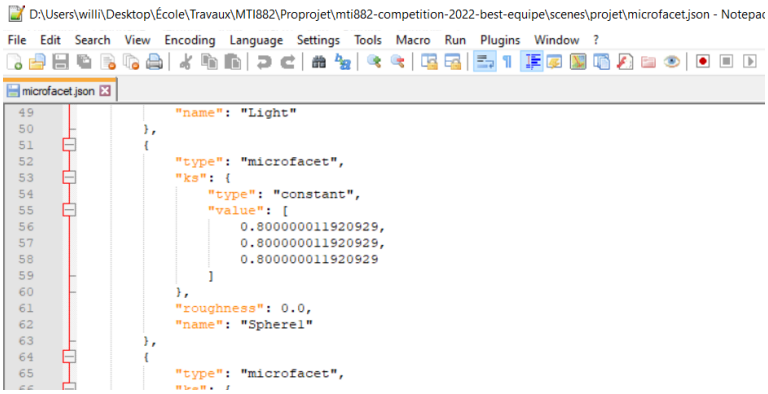


De plus, on remarque qu'il a créé un sous-dossier avec le bon nom dans *meshes* pour contenir les modèles utilisés par la scène.



4.2.2 Exportation des matériaux micro-facettes

En ouvrant le fichier *microfacet.json* généré avec le *plugin*, on peut voir que le matériau micro-facette a bien été mis dans la scène.



4.3 Difficultés rencontrées

4.3.1 Apprendre à travailler avec l'API de Blender

Travailler avec ce *plugin* correspondait à notre première interaction avec l'API de Blender. La documentation en ligne sur cet API ainsi que les forums de développement nous ont beaucoup aidé.

5 Carte d'environnement

Dans notre projet, nous avons décidé d'ajouter une carte d'environnement. Dans notre scène, l'espace est un environnement vide et gigantesque. Peu importe l'orientation de la caméra, s'il n'y a pas d'objets qui cachent la vue, tout ce que l'on devrait voir est un ciel étoilé.

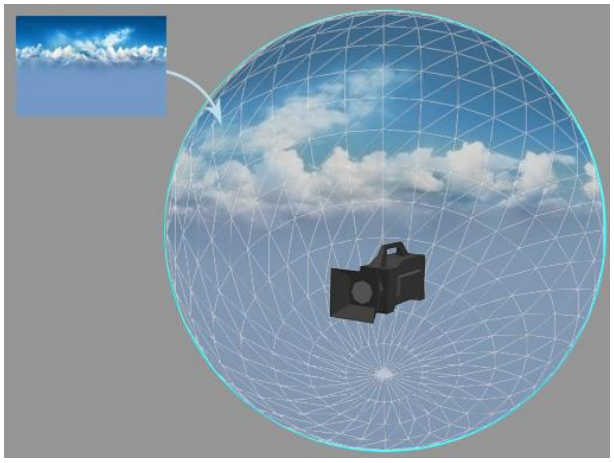
Pour reproduire cela, le skydome est une solution. Cette technique nous permet d'ajouter une image en tant que « couleur » d'arrière-plan. Elle nous permet aussi d'éviter toute réflexion sur le ciel, augmentant ainsi le réalisme.

Voici un exemple où on peut voir l'impact d'un arrière-plan imagé sur une scène.

Figure 5.1 Démonstration d'une scène comportant un arrière-plan

5.1 Détails techniques

Pour concevoir cela, nous pouvons représenter le ciel comme étant l'intérieur d'une sphère. Ainsi, si une personne se situe au centre d'une sphère, peu importe la direction dans laquelle elle regarde, tout ce qu'elle verra est la surface intérieure de cette sphère.



Source:
https://manual.reallusion.com/iClone_7/ENU/Content/iClone_7/Pro_7.4/10_Scene/Atmosphere/Utilizing_Sky.htm

Nous avons réappliqué ce concept pour créer notre skydome. Lors d'un rendu, nous commençons par créer une sphère de rayon 1 auquel nous appliquons une texture. Cette forme n'est cependant pas ajoutée à la liste des objets de la scène étant donné que nous ne voulons pas qu'il y ait une intersection avec le ciel.

Ensuite, lorsque nous envoyons un rayon pour récupérer la lumière, si jamais ce rayon ne touche aucun objet et qu'il va vers l'infini, nous récupérons sa direction. Avec cette direction, on peut alors calculer l'intersection avec la sphère du skydome. Pour ce faire, nous créons un nouveau rayon que l'on positionne au centre de la sphère [0,0], puis nous lui appliquons la direction précédemment calculée. À partir de là, il est désormais possible de calculer l'intersection sur le skydome.

Une fois l'intersection obtenue, on calcule la valeur de la texture sur la sphère grâce aux coordonnées UV. Avec la couleur de la texture obtenue, il est alors possible de calculer l'impact de la couleur de l'arrière-plan sur la scène.

Les images suivantes démontrent à quoi ressemble une scène avec et sans skydome.

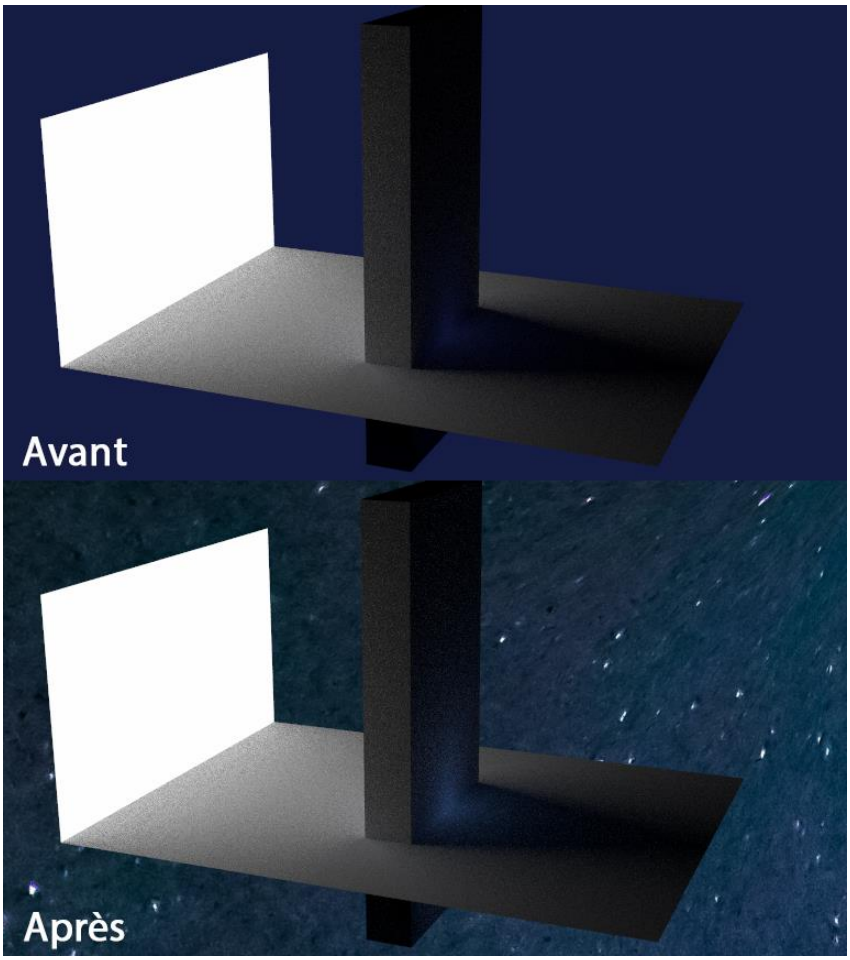


Figure 5.2 Différence entre une scène avec et sans un arrière-plan imagé

Pour ajouter une image en arrière-plan, l'objet « background » dans les fichiers de configurations de la scène a été modifié.

```
"background": {
  "type": "background",
  "default_color": [0.008, 0.012, 0.059],
  "texture": {
    "type": "texture",
    "filename": "textures/background.jpg"
  }
}
```

Désormais, il faut lui spécifier un « type » car l'arrière-plan est considéré comme un matériel. On peut aussi spécifier une couleur par défaut dans le cas où une texture n'a pas été trouvée. Finalement, la balise « texture » doit être ajoutée si l'on veut charger une image pour l'arrière-plan.

5.2 Validation

Pour valider cette fonctionnalité, nous avons eu à valider deux choses : valider l'impact de la couleur de l'image d'arrière-plan sur la scène, et valider qu'on obtient les bonnes positions des textures selon la direction des rayons. Valider l'impact de la couleur nous permet de vérifier que nous calculons correctement les effets de lumière, et valider le positionnement nous permet de valider que nous obtenons les bonnes valeurs de la texture.

5.2.1 Impact de la lumière sur la scène

Pour ce qui est de valider l'impact de l'arrière-plan sur la scène, nous n'avions qu'à comparer le résultat obtenu en utilisant une couleur unie et en utilisant une image. En utilisant la couleur [0.008, 0.012, 0.059], nous avons obtenu le résultat suivant :

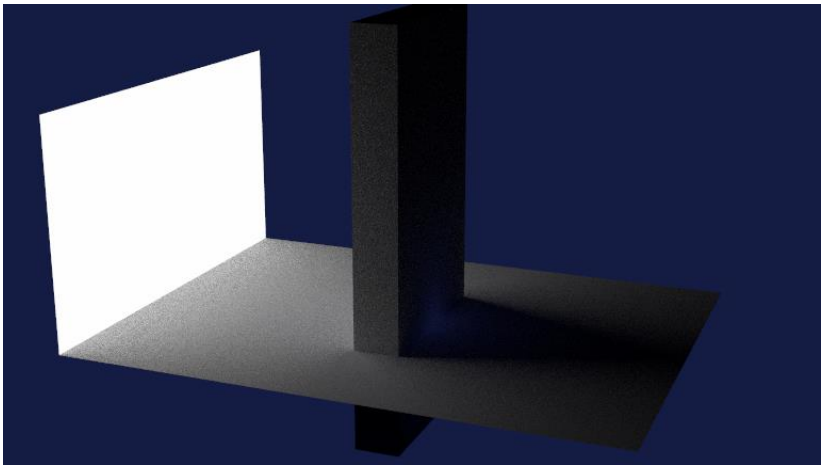


Figure 5.3 Scène de test avec un arrière-plan utilisant une couleur unie

Ensuite, nous avons généré la même scène, mais avec une image comme arrière-plan. Le résultat obtenu est le suivant:

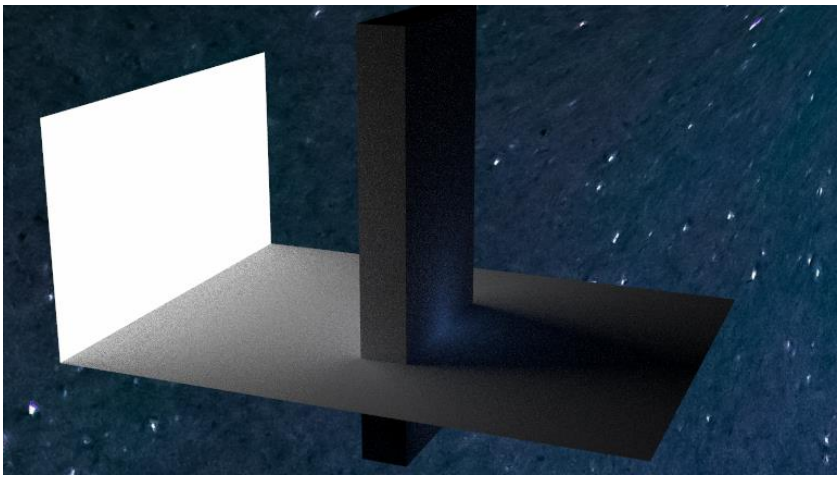


Figure 5.4 Scène de test avec un arrière-plan texturé

Si l'on compare les deux images entre elles tout en augmentant l'exposition à 2.7, on remarque qu'utiliser une image comme arrière-plan impacte la scène au même endroit, c'est-à-dire la base du prisme. Évidemment, comme la couleur de l'image n'est pas la même, la lumière résultante est différente.

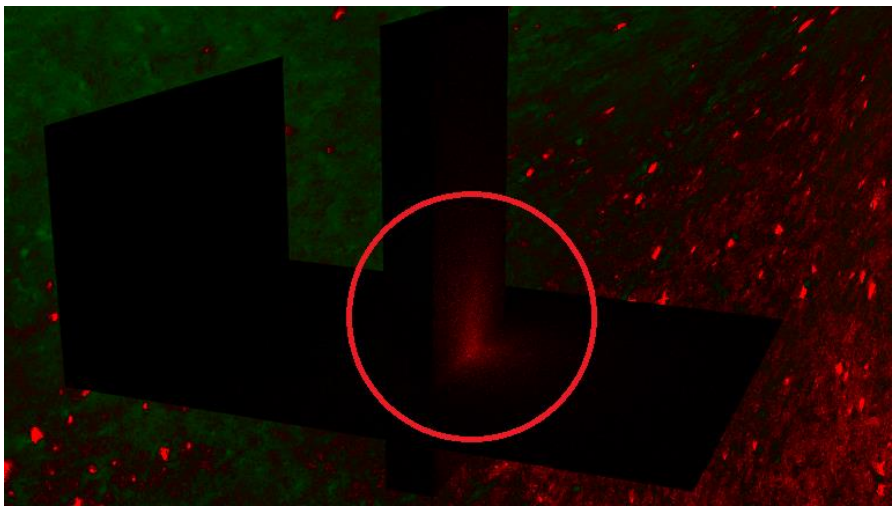


Figure 5.5 Différence entre les scènes avec et sans arrière-plan texturé

5.2.2 Positionnement et déformation de la texture

Pour valider le positionnement et la déformation de l'image d'arrière-plan, nous avons comparé les images générées par notre moteur rendu avec d'autres générées depuis Blender en utilisant les mêmes configurations. Avant toute chose, il est important de noter que pour tous nos exemples, nous avons utilisé un FOV de 30 ainsi qu'une distance focale de 20.

Comme premier exemple, nous avons orienté la caméra selon l'angle [0,0,0] dans Blender. Dans notre fichier de configuration de la scène, cela équivaut à appliquer les transformations suivantes sur notre caméra :

```
"transform": {
    "from": [
        0.0,
        0.0,
        0.0
    ],
    "at": [
        0.0,
        0.0,
        -1.0
    ],
    "up": [
        0.0,
        1.0,
        0.0
    ]
}
```

Le résultat obtenu pour une orientation de [0,0,0] est le suivant:

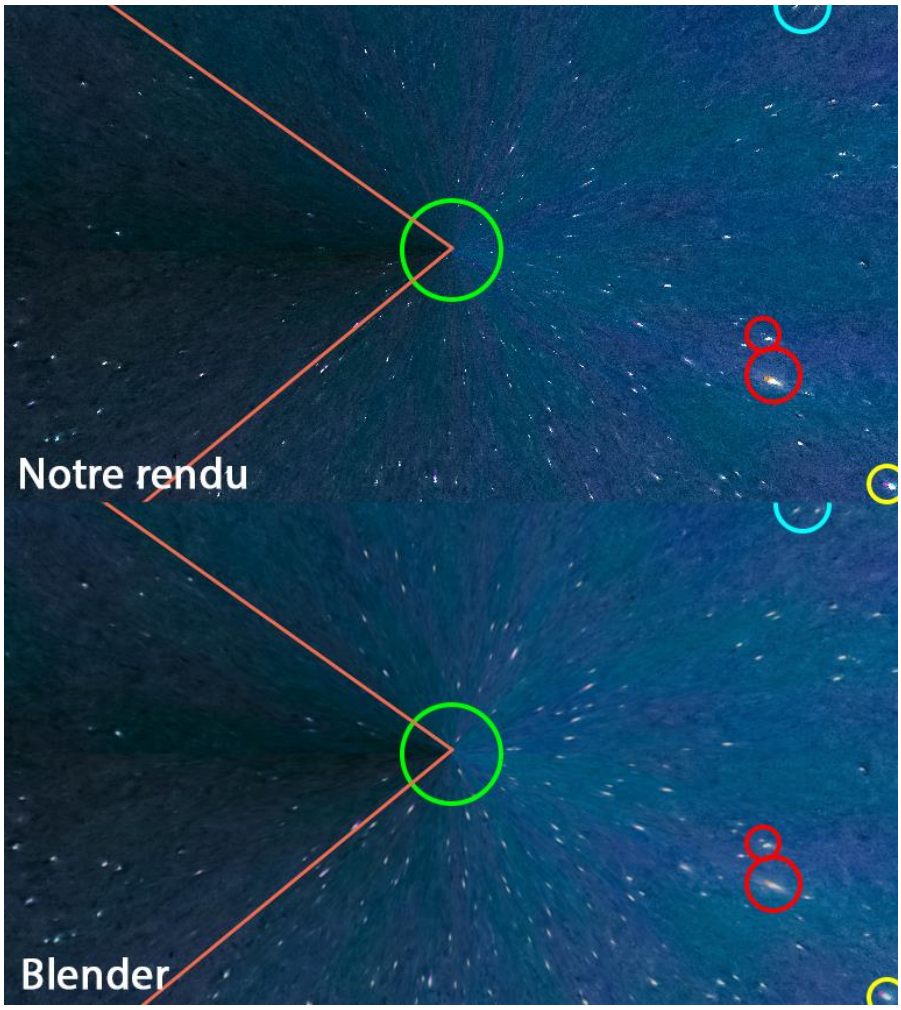


Figure 5.6 Test 1 validant notre implémentation d'une carte d'environnement

Comme on peut le voir, notre moteur de rendu positionne et déforme correctement la texture sur la sphère. En effet, si l'on regarde bien, le côté gauche des deux images est plus sombre au même endroit, et de nombreuses étoiles peuvent être retrouvées aux mêmes positions.

Le même genre de résultat peut être obtenu si nous changeons l'orientation de la caméra vers [0,180,0]. Dans notre fichier de configuration, les valeurs de transformation pour la caméra changent alors pour les suivantes:

```
"transform": {
    "from": [
        0.0,
        0.0,
        0.0
    ],
    "at": [
        8.742277657347586e-08,
        0.0,
        1.0
    ],
    "up": [
        0.0,
        1.0,
        0.0
    ]
}
```

Avec une orientation de [0,180,0] pour la caméra, nous obtenons ainsi le résultat suivant:

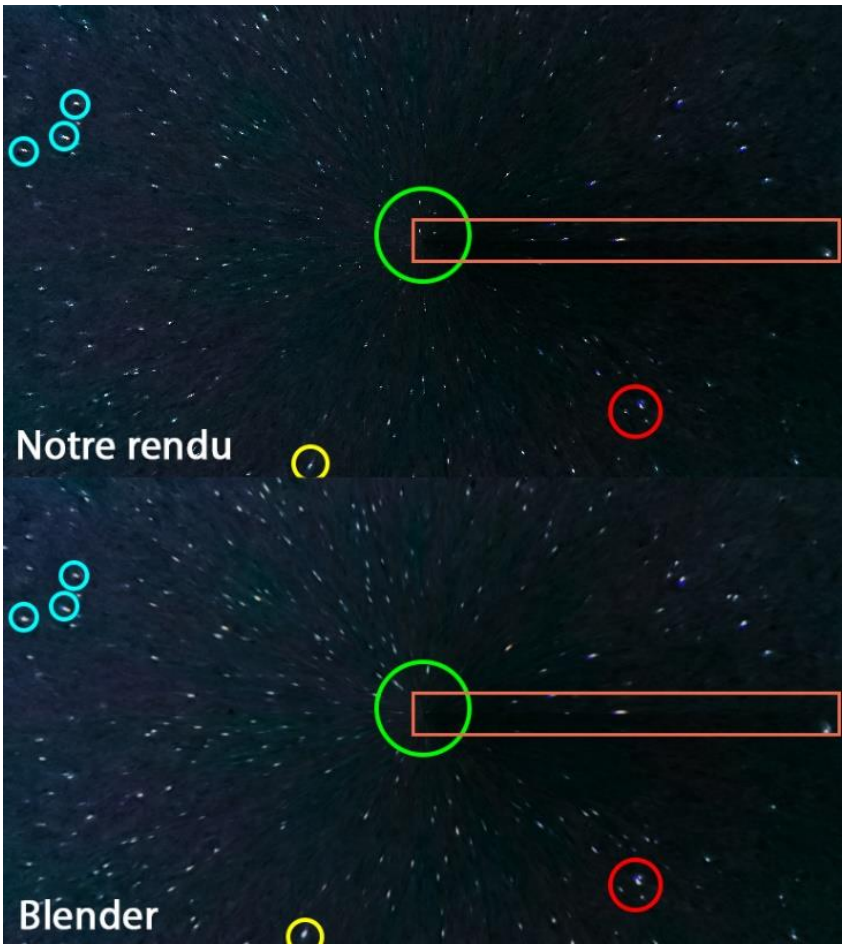


Figure 5.7 Test 2 validant notre implémentation d'une carte d'environnement

Pour valider encore plus notre implémentation, nous avons aussi effectué ce test en orientant la caméra selon des angles plus aléatoires, soit [30, 45, 72]. Dans notre fichier de configuration de la scène, cela équivaut à appliquer les transformations suivantes sur notre caméra:

```
"transform": {
    "from": [
        0.0,
        0.0,
        0.0
    ],
    "at": [
        -0.6647617220878601,
        -0.4278923273086548,
        -0.6123723983764648
    ],
    "up": [
```

```
            -0.7143850922584534,
            0.6038658022880554,
            0.3535533845424652
        ]
    }
```

Avec les angles [30, 45, 72], le résultat obtenu est alors le suivant:

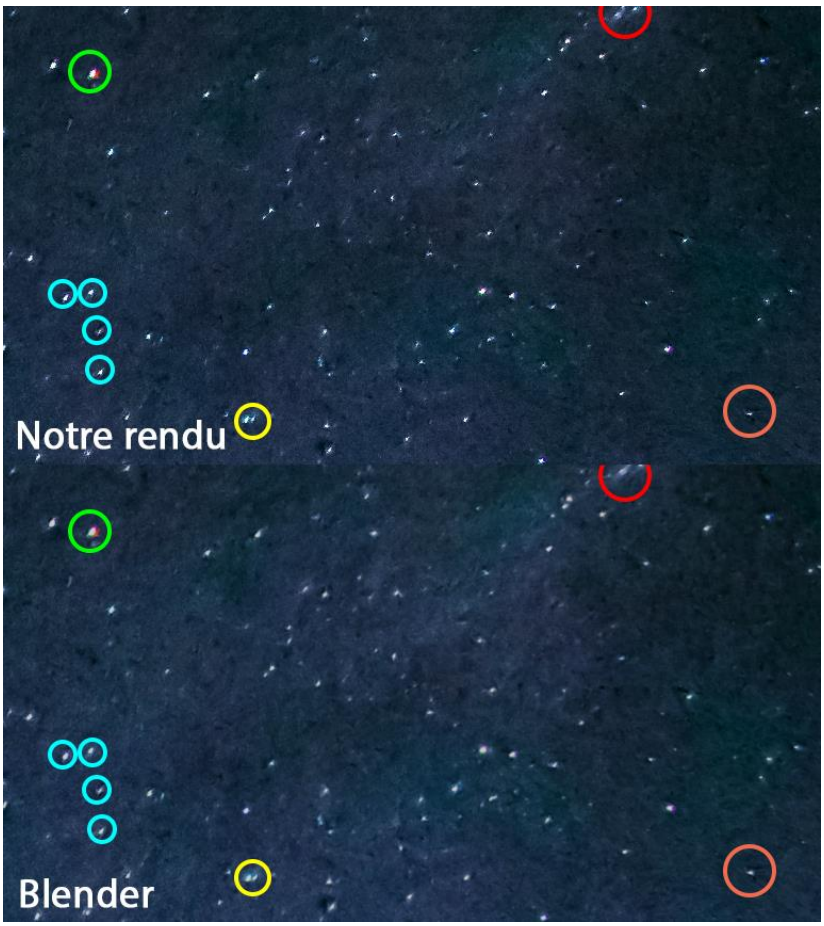


Figure 5.8 Test 3 validant notre implémentation d'une carte d'environnement

Encore une fois, l'image générée par Blender est la même que celle générée par notre moteur de rendu. C'est pourquoi on peut alors valider que le positionnement et la déformation de la texture dans notre skydome sont valides.

5.3 Difficultés rencontrées

Lors de l'implémentation de notre skydome, nous avons rencontré au moins un problème pas évident à résoudre. Durant nos premiers tests, la nouvelle texture appliquée en arrière-plan impactait trop la lumière sur la scène.

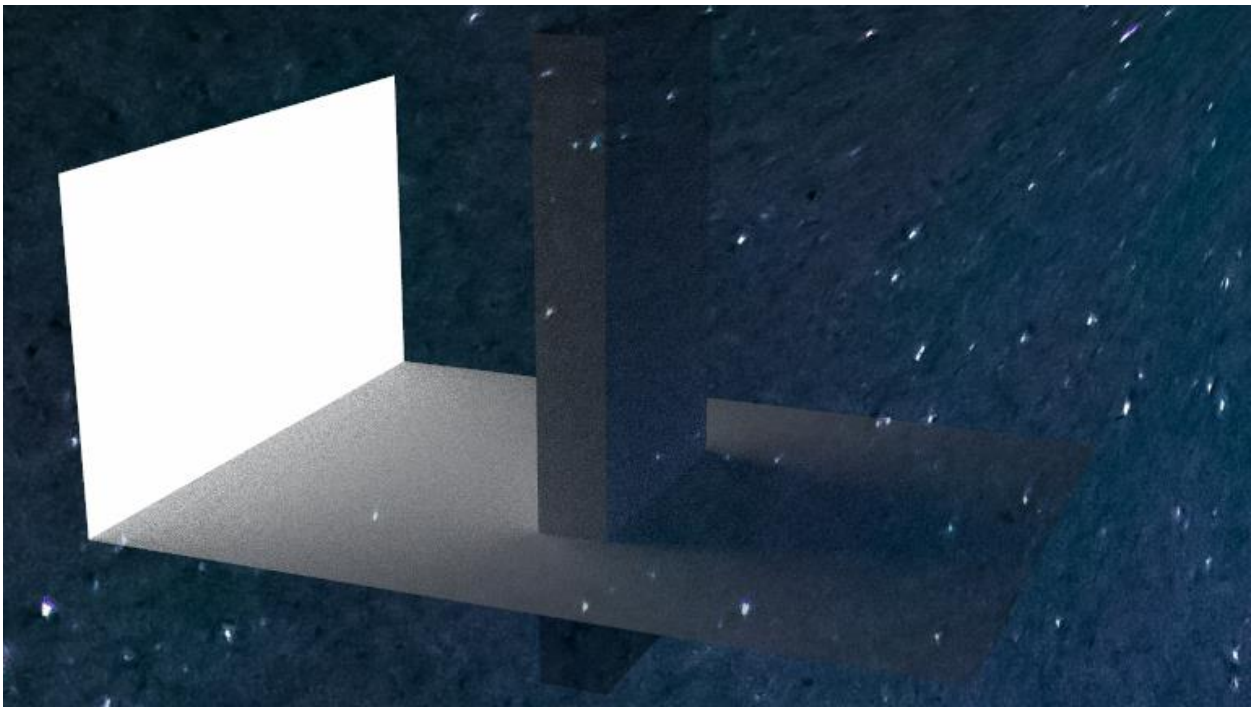


Figure 5.9 Carte d'environnement - Problème 1

Après avoir analysé et testé certaines choses, nous avons fini par comprendre ce qui causait l'erreur: à chaque fois qu'aucun objet n'était touché par un rayon, nous allions chercher la valeur en arrière-plan. Cela causait le problème précédemment montré où l'arrière-plan semble passer au-dessus des objets. Pour résoudre ce problème, il fallait en fait faire deux choses.

Premièrement, nous avons ajouté une nouvelle condition : si aucun objet n'est touché et qu'on est au premier rebond, on retourne directement la valeur de l'arrière-plan.

Deuxièmement, nous avons considéré la valeur de l'arrière-plan dans le calcul du MIS. En effet, au départ, lors du calcul de la BSDF, si aucun objet n'était touché après un rebond sur une surface, nous retournions la couleur noire. Nous avons changé cela pour que le calcul de la BSDF continue et utilise plutôt la couleur de l'arrière-plan. Grâce à cela, nos résultats ont alors changé pour le mieux : l'arrière-plan ne traversait plus les objets. Toutefois, un problème persistait toujours, car l'arrière-plan semblait se refléter sur les objets même si ceux-ci utilisaient un matériau diffus :

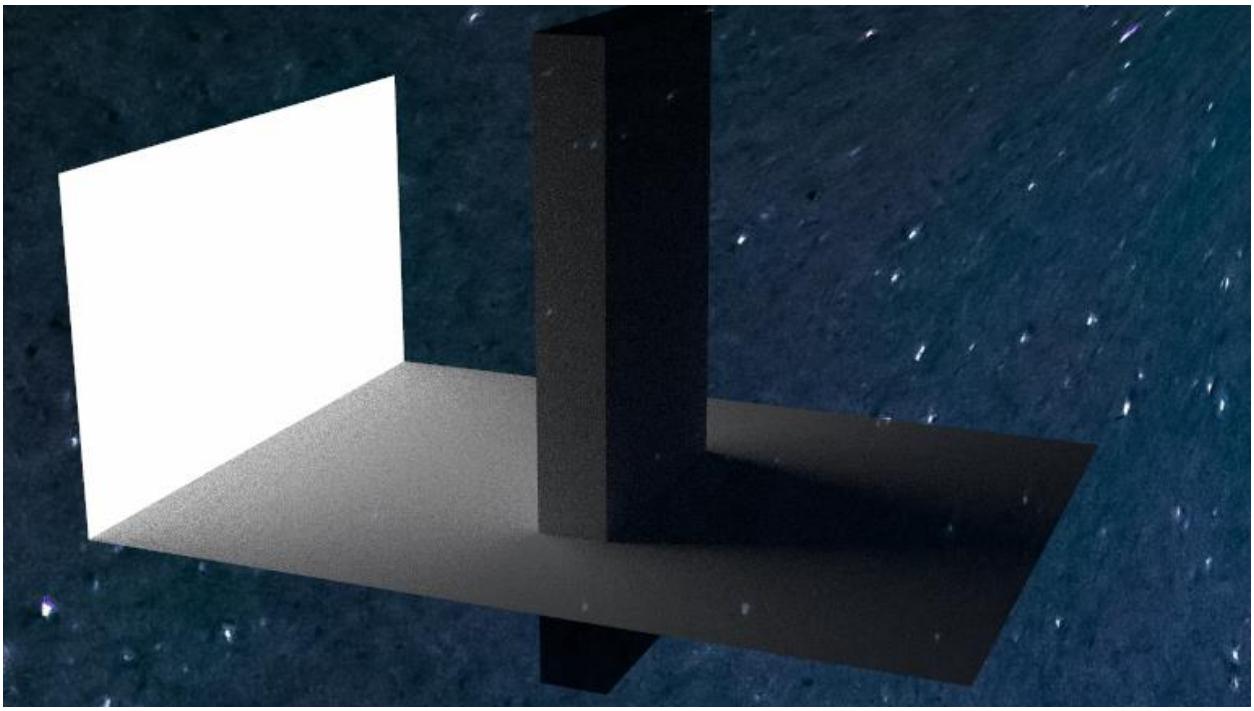


Figure 5.10 Carte d'environnement - Problème 2

Pour résoudre ce nouveau problème, nous avons considéré l'arrière-plan comme étant un matériau émissif. Ainsi, nous avons créé un nouveau matériau nommé *skydome.h*. Faire cela a ainsi réglé le dernier problème occasionné par cette nouvelle fonctionnalité.

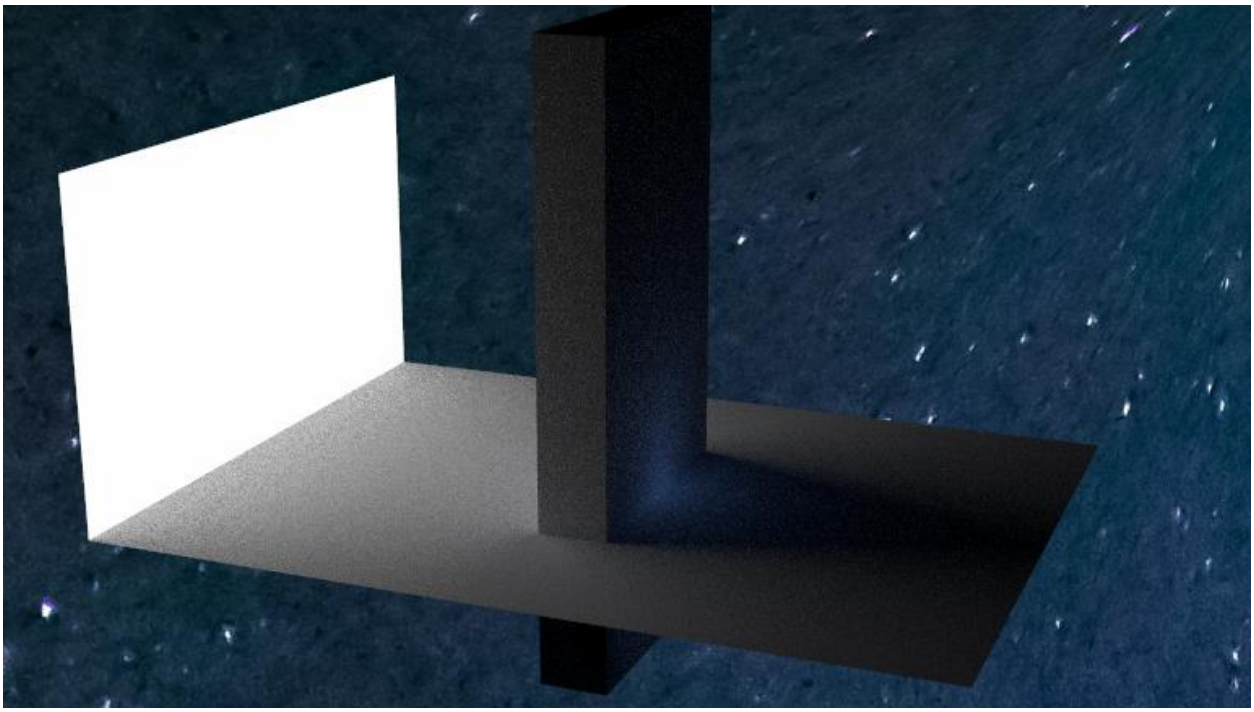


Figure 5.11 Scène de test finale de notre carte d'environnement

6 Meilleur échantillonnage des lumières

Dans notre moteur de rendu, pour l'échantillonnage des lumières qui se fait dans plusieurs intégrateurs, la source de lumière est échantillonnée de façon aléatoire avec une probabilité égale pour chaque source de lumière. Ceci fonctionne bien lorsque nous n'avons qu'une seule source de lumière. L'image suivante, à 20 *spp* et dont l'intégration n'est faite qu'avec la lumière directe, montre cette situation.

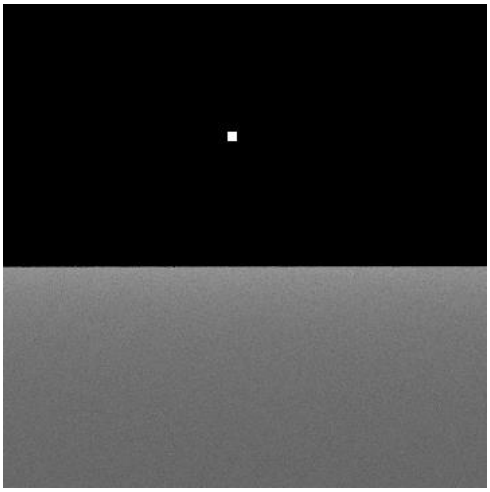


Figure 6.1 : 20 spp, une seule source de lumière

Nous pouvons voir dans l'image précédente que, même avec un nombre d'échantillons limité, la variance de la lumière au sol n'est pas haute.

Cependant, lorsque nous avons plusieurs sources de lumière qui varient grandement en intensité et en distance, comme c'est le cas dans notre image finale, la variance de tous les éléments est augmentée. L'image suivante représente cette situation.

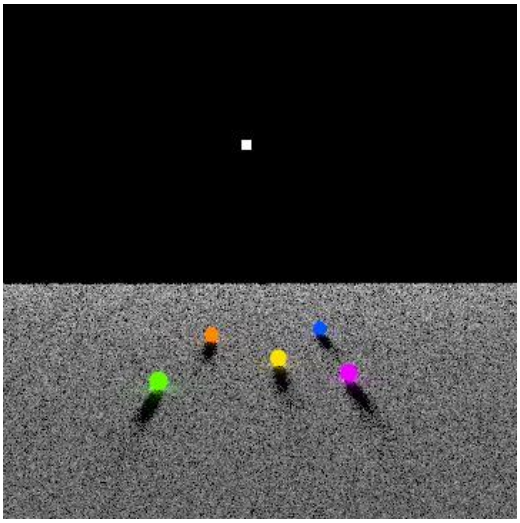


Figure 6.2 : 20 spp, plusieurs sources de lumière, méthode originale

L'image précédente montre que l'ajout de plusieurs sources de lumière vient faire augmenter grandement la variance sur toute l'image, ce qui n'est pas optimal.

Nous nous sommes donc penchés sur une méthode qui échantillonnerait une lumière selon sa distance et son intensité.

6.1 Détails techniques

Pour mettre en place cette technique, nous avons changé les façons dont nous échantillonnions une source de lumière et dont nous calculions sa *pdf* dans le fichier *scene.h*.

En effet, cette nouvelle technique consiste à calculer un poids pour chacune des sources lumineuses avec la formule suivante :

$$w_i = \frac{L_i}{|x - y_i|^2}$$

Dans cette formule, w_i est le poids calculé de la source lumineuse, L_i est l'intensité lumineuse de la source, x est le point sur la surface à partir duquel nous échantillonnons une source de lumière et y_i est le centre de la source de lumière. Ici, nous prenons le centre de la source, puisqu'il nous est impossible de savoir à ce moment quel point sur la surface sera échantillonné. Notre hypothèse est que le centre est une bonne estimation.

Le poids d'une source plus intense sera plus fort. Elle aura donc plus de chance d'être prise. Au contraire, le poids d'une source plus éloignée sera moins fort, ce qui lui laissera moins de chance d'être prise.

Ensuite, nous itérons sur toutes les sources lumineuses de la scène, calculant la *cdf*. Lorsque celle-ci est plus grande que notre variable aléatoire, alors nous savons que nous avons trouvé la bonne source de lumière à échantillonner.

Nous avons ajouté un attribut booléen, *use_better_light_sample*, dans le fichier de configuration de la scène qui, lorsque mis à *true*, active cette nouvelle méthode.

6.2 Validation

Pour valider nos résultats, nous avons utilisé les mêmes conditions que l'image précédemment générée, mais en utilisant notre nouvelle méthode d'échantillonnage de lumière. L'image suivante représente le résultat obtenu.

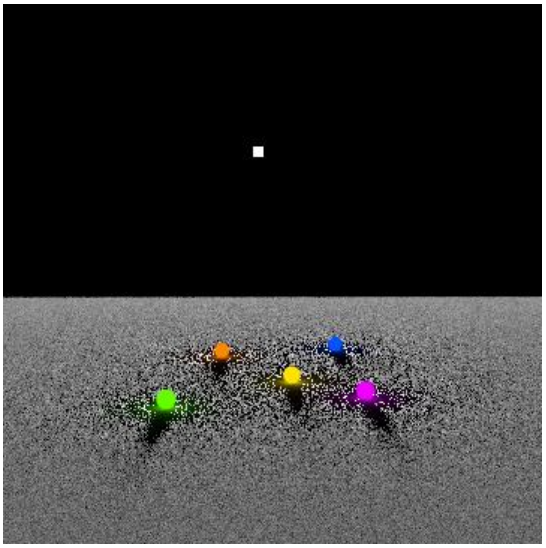


Figure 6.3 : 20 spp, plusieurs sources de lumière, nouvelle méthode

Nous remarquons, dans l'image précédente, que la variance proche des sources de lumière est plus élevée que pour la figure 6.2. Ceci pourrait être dû au fait que la lumière intense sera moins souvent échantillonnée près des lumières moins intenses.

Cependant, nous pouvons voir que le sol qui est plus éloigné des lumières a une variance moins élevée que celle de la figure 6.2. La présence de plusieurs lumières n'a donc plus autant d'influence qu'avec la méthode originale. C'est le résultat que nous espérions obtenir.

Si nous faisons une analyse différentielle sur les figures 6.2 et 6.3, nous obtenons l'image suivante.

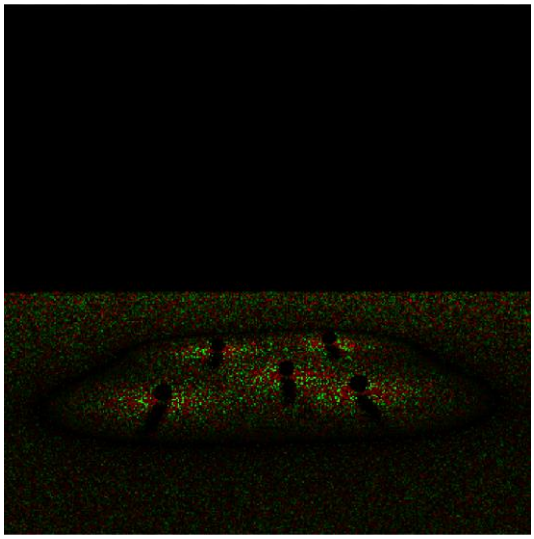


Figure 6.4 : Analyse différentiel de l'erreur entre les images 6.2 et 6.3

L'image précédente présente une quantité similaire de points verts (intensité plus élevée) et de points rouges (intensité plus faible). Nous pouvons donc estimer que la méthode offre des résultats qui sont valides.

6.3 Difficultés rencontrées

6.3.1 Calculer la distance

Originalement, les méthodes *sample_emitter* et *pdf_emitter* de la classe *Scene* n'avaient pas les moyens de calculer la distance entre la source de lumière et le point d'origine.

Pour régler ce problème, nous avons changé les définitions des fonctions pour prendre en entrée ce point d'origine qui nous permet de calculer les distances. Il a également fallu mettre à jour tous les appels de la méthode.

6.3.2 Calculer l'intensité

Dans notre moteur de rendu, l'intensité d'une source de lumière est habituellement calculée selon une direction et les coordonnées *uv* sur une surface. Cependant, nous n'avons pas accès à ces informations lorsque vient le temps d'échantillonner une source. Dans le cas parfait, il nous faudrait donc intégrer l'intensité sur l'ensemble de la surface pour avoir une estimation correcte. Cependant, cela serait coûteux en termes de performance, ce qui viendrait à l'encontre de notre raison pour implémenter cette nouvelle méthode.

Puisque, dans toutes nos scènes, l'intensité des sources lumineuses est la même indépendamment de la direction donnée et des coordonnées *uv*, nous avons simplement choisi d'ignorer ces informations lors du calcul d'intensité.

7 Gestion des masques

Pour notre image finale, nous voulions placer une certaine végétation dans la scène. Cependant, faire ou trouver un modèle 3D où tous les brins d'herbe et toutes les feuilles sont modélisés n'était pas possible. Nous avons donc décidé de supporter la transparence à l'aide de masques pour le modèle.

7.1 Détails techniques

Le masquage, dans notre moteur de rendu, fonctionne de la même manière que dans une application d'édition d'images. En effet, le masque est une image en teinte de gris que nous fournissons au moteur. Cette image représente la transparence de l'objet, où noir (0) est complètement transparent et blanc (1) est complètement opaque.

Dans le code, nous avons ajouté le nouveau matériau *Mask* qui est défini de cette façon :

```
{
  "type": "mask",
  "name": "nomDuMatériau",
  "mask": "cheminVersTextureMasque",
  "material": {
    ...
  }
}
```

Niveau technique, nous utilisons Monte Carlo pour approximer la transparence du matériau selon la texture. La valeur du texel représente la chance que le rayon a de passer au travers du matériau.

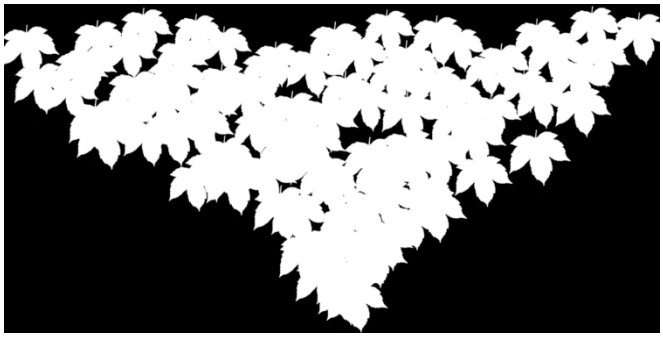
Par manque de temps, cette fonctionnalité n'a pas pu être incluse dans l'image finale. Cependant, celle-ci reste fonctionnelle dans notre moteur de rendu.

7.2 Validation

Pour valider cette fonctionnalité, nous avons utilisé une texture transparente avec laquelle nous connaissions les résultats attendus. Voici la texture en question :

Source : https://blender.stackexchange.com/questions/92613/baking-materials-with-alpha-blended-textures

Pour valider notre implémentation, nous avons utilisé le masque suivant :



En utilisant le logiciel Photoshop, nous avons pu déterminer que le résultat attendu dans le moteur de rendu pour un *quad* de même dimension serait le suivant :

Voici l'image générée :

Dans l'image précédente, nous pouvons constater que le l'image sur le mur arrière correspond en effet à l'image attendu produite par Photoshop. De plus, on peut voir que l'ombre projetée par le matériel sur le mur semble bonne.

Dans cette image, nous avons également testé notre implémentation sur une sphère et sur un modèle de lapin. Nous constatons que la texture suit bien les valeurs d'UV du matériau et que le masque fonctionne sur des modèles plus complexes.

7.3 Difficultés rencontrées

7.3.1 Problème d'échantillonnage

Étant donné que le matériau de masque laisse parfois passer les rayons tout droits lors d'une intersection, il est impossible de le considérer comme diffus. Nous n'avons donc pas été en mesure d'élaborer le code pour évaluer le matériau et son algorithme.

La solution trouvée à ce problème est de considérer le matériau masque comme étant spéculaire. Les matériaux spéculaires, dans notre moteur de rendu, sont effectivement traités différemment que les matériaux diffus puisque qu'ils contiennent une distribution de Dirac dans leur équation.

7.3.2 Matériaux diffus unidirectionnels

Initialement, nos matériaux diffus ne supportaient pas les rayons qui intersectaient avec le matériau à partir de l'intérieur de celui-ci. Voici le résultat donné par cette situation :

Nous pouvons constater que les feuilles qui sont de l'autre côté du modèle de la sphère et du lapin sont complètement noires. De plus, les ombres en arrière des feuilles sur le mur sont plus noires que

ce qu'elles devraient être. Ceci est dû au fait que le matériau diffus considérait tous les rayons intersectant avec son intérieur comme étant invalides.

Pour régler ce problème, nous avons ajouté l'attribut *render_both_sides* dans le matériau diffus qui active l'affichage de l'intérieur de celui-ci.

8 Autres difficultés rencontrées

8.1 Trou noir

Dans notre proposition de départ, nous avions comme objectif de concevoir un trou noir pour notre rendu final. Après au moins deux jours de lectures et de tentatives, nous avons fini par abandonner l'idée.

En effet, bien que mentionné par certains de nos camarades de classe, créer un trou noir est beaucoup plus complexe que ça en a l'air. Les papiers sur lesquels nous nous basions utilisaient des notions dont nous n'avions jamais vu, et il y avait, selon nous, trop d'équations à implémenter (et donc à comprendre) pour que nous puissions enfin obtenir un résultat concluant. De ce fait, nous avons laissé tomber cette idée pour aller vers de quoi de beaucoup plus simple.

8.2 Texture du Soleil invisible

Un autre problème mineur rencontré lors du développement de notre projet est que la texture des objets émissifs à forte intensité devient presque invisible. Voici un exemple du problème mentionné. Dans l'image ci-dessous, on applique une texture de soleil à l'émetteur. En revanche, comme l'émetteur a une intensité de 500, la texture n'est presque plus visible.

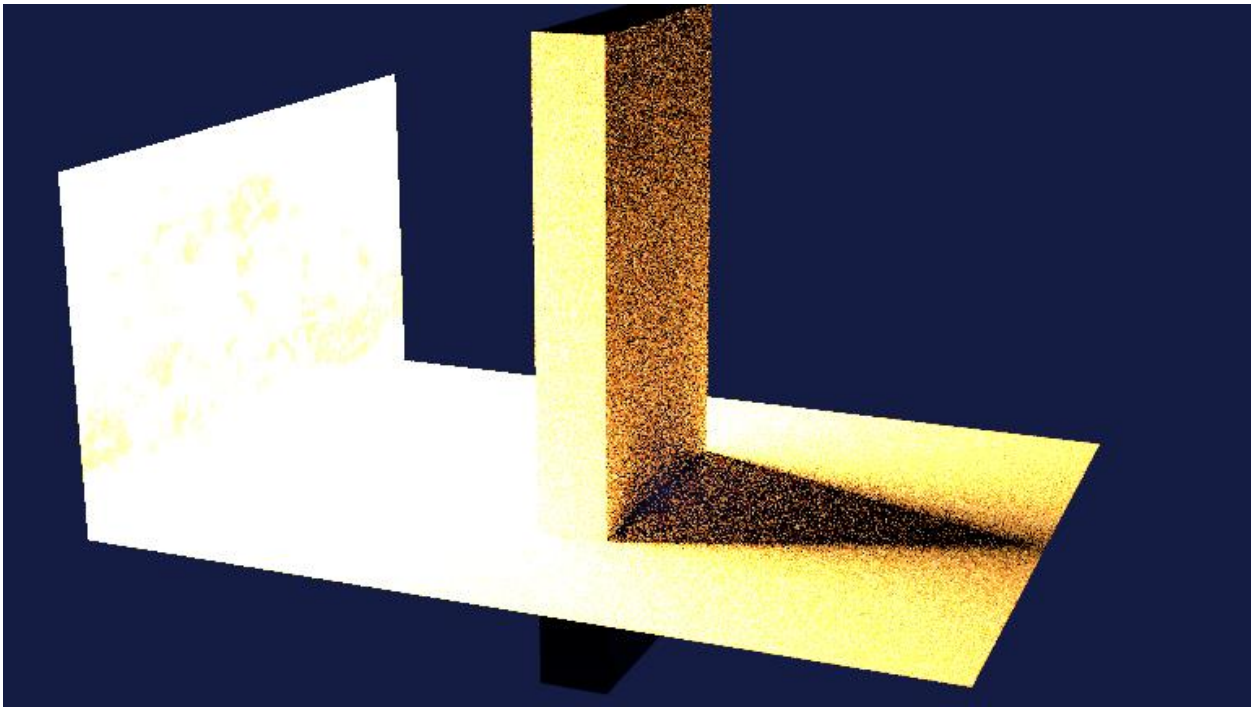


Figure 8.1 Problème d'un matériau émissif à haute intensité avec une texture

Pour résoudre ce problème, nous avons opté pour une solution irréaliste, mais simple : si on échantillonne directement un émetteur (c'est-à-dire qu'aucun rebond n'a encore été effectué), on retourne tout simplement la valeur de la texture sans l'intensité de la lumière. La texture de l'émetteur

reste alors visible, mais la forme émet toujours autant de lumière. Voici la même image obtenue à la suite de ce changement.

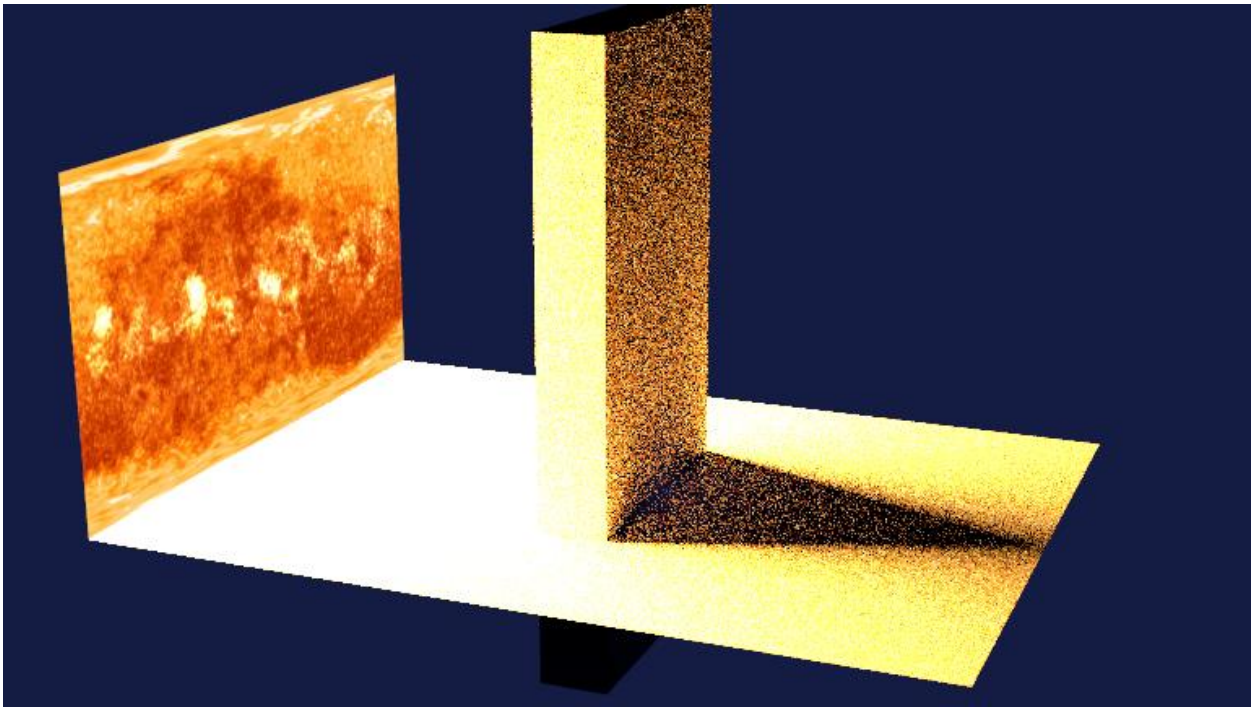


Figure 8.2 Démonstration d'un matériau émissif à haute intensité ajusté

9 Répartition du travail

Notre travail a été réparti équitablement selon le nombre de points que valait chacune des tâches.

William :

- Modèle de matériaux par micro-facettes (4 points);
- Amélioration légère du *plugin* Blender (1 point);
- Meilleur échantillonnage des lumières (1 point);
- Gestion des masques (1 point);
- Création de la scène dans Blender.

Vincent :

- Rendu volumétrique (8 points);
- Carte d'environnement (1 point);
- Tentative d'implémentation d'un trou noir;
- Correction de l'intensité des matériaux émissifs.

10 Conclusion

Pour conclure ce projet, nous pouvons affirmer que celui-ci nous a été très formateur. En effet, implémenter les différentes fonctionnalités nous a permis de mieux comprendre certains concepts que nous avions vu en classe. Par exemple, implémenter un système de rendu volumétrique a été très utile pour la compréhension de cette matière. Implémenter les modèles de micro-facettes nous a procuré le même effet.

Malgré cela, il est malheureux d'affirmer que nous n'avons pas réussi à accomplir tous nos objectifs et à obtenir un résultat final satisfaisant. Nous pourrions blâmer le fait que nous nous sommes pris à la dernière minute, par moment. Toutefois, cette raison n'est pas applicable pour l'ensemble du projet. En effet, comme la quantité de travail demandée était plutôt grande, comme nous ne maîtrisions pas la matière au moment de concevoir les fonctionnalités au vu de leur complexité, et comme nous avions aussi d'autres cours à nous préoccuper, ce sont plutôt ces raisons qui nous ont malheureusement empêchés de terminer toutes nos tâches.

Références pour l'image

Modèles utilisés pour notre scène Blender

Vaisseau: https://sketchfab.com/3d-models/outer-wilds-theship-f6797d8650794c8387708f7ef78ee0d5

Tente: https://www.cgtrader.com/free-3dmodels/furniture/outdoor-furniture/outdoor-camp-or-tent

Feu de camp: https://www.cgtrader.com/3dmodels/various/various-models/campfire-0b106c12-2262-4701-96fa-076e9fd794d5

Arbre: https://www.cgtrader.com/3dmodels/plant/conifer/ue4-fir-trees

Astronaute: https://www.cgtrader.com/3dmodels/character/sci-fi-character/astronaut--2

Textures utilisées pour notre scène Blender

Terrain: https://quixel.com/megascans/home?category=surface&search=grass&assetId=sgmkajak

Soleil: https://www.solarsystemscope.com/textures/

Ciel étoilé: https://pixabay.com/fr/illustrations/contexteespace-univers-galaxie-3883181/