24-11-2024

Samuel Fournier

November 2024

Dans le dernier mois (oui, ça fait déjà un mois depuis le dernier rapport). J'ai continuer de travailler sur le *Ray Marching*. Malheureusement, lors de mon implémentation initiale de l'algorithme, j'ai commis certaines erreurs. La première et la plus importante selon moi, était le calcul de la couleur de scatter. Dans la première itération du code, la couleur du scatter finissait toujours par être **(0,0,0)**, ce qui faisait en sorte que la couleur finale n'était que la couleur de l'objet multiplié par le facteur d'opacité **hit.transmittance**. De se fait, la fumée résultante était très sombre (presque noir) dans les régions où la densité était élevé. La raison pour le problème était à cause de cette ligne de code dans la fonction **lightMediumIntersection**:

```
    if ((tmin > tymax) || (tymin > tmax)) return false;
```

Lorsque j'ai écrit cette fonction, je n'ai que copié la fonction d'intersection avec le médium **localIntersect**, cependant cette fonction ne gère pas le cas où l'origine du rayon est à l'intérieur du médium. Lorsque l'on tire le **lightRay**, son origine est à l'intérieur du médium, ce qui fait en sorte que l'on entre dans ce **if** et on retourne **false**. Ensuite, on utilise cette valeur de retour pour tester:

```
    // Use epsilon to avoid self intersection;
    if(!lightMediumIntersection(lightRay, EPSILON, &t_light)) {
        continue;
    }
```

Comme on obtenait toujours **false**, on entrait dans le **if** et on passait à la prochaine itération de la boucle. En passant à la prochaine itération, on saute complètement le calcul du scatter, ce qui fait en sorte que **hit.scatter** reste à sa valeur initiale de **(0,0,0)**. Pourquoi est-ce que la fonction retournait toujours **false**? Voici l'explication mathématique:
On rappelle l'équation d'un rayon:

$$r(t) = \begin{cases} x(t) = d_x t + o_x \\ y(t) = d_y t + o_y \\ z(t) = d_z t + o_z \end{cases}$$

Le médium est défini par les plans suivants:

$$x = 0$$
$$x = 1$$
$$y = 0$$
$$y = 1$$
$$z = 0$$
$$z = 1$$

Maintenant, supposons que notre rayon à comme origine:

$$(o_x, o_y, o_z) = (0.5, 0.5, 0.5)$$

Voici les calculs pour les valeurs de t:

---

x = 0

$$\begin{aligned} 0 &= d_x t_{minx} + 0.5 \\ -0.5 &= d_x t_{minx} \\ \frac{-0.5}{d_x} &= t_{minx} \end{aligned}$$

---

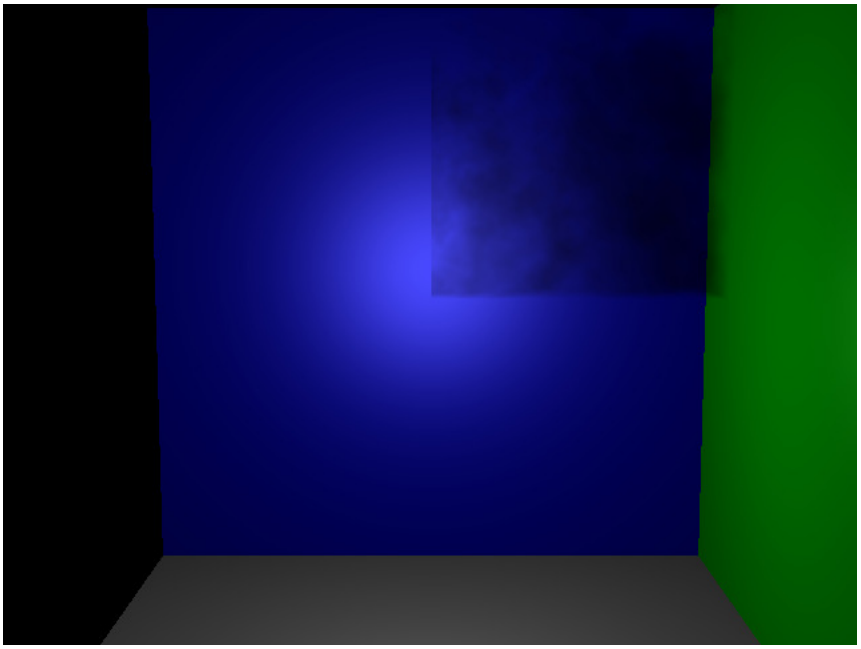
x = 1

$$\begin{aligned} 1 &= d_x t_{maxx} + 0.5 \\ 0.5 &= d_x t_{maxx} \\ \frac{0.5}{d_x} &= t_{maxx} \end{aligned}$$

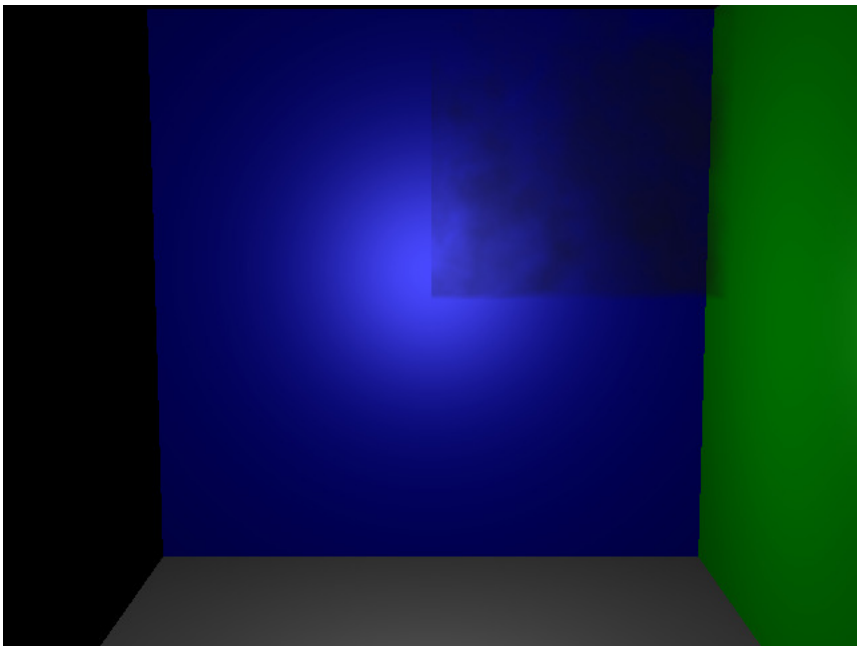
On utilise des calculs similaires pour obtenir t_{miny}, t_{maxy}, t_{minz} et t_{maxz}. Bref, on remarque qu'en fonction du signe de **d**, soit t_{min} ou t_{max} sera négatif. Si **d** est négatif, alors t_{max} sera négatif et si **d** est positif, alors t_{min} sera négatif. Par conséquent, peut importe le signe de **d**, les conditions suivantes:

```
1    if ((tmin > tymax) || (tymin > tmax)) return false;
2    if ((tmin > tzmax) || (tzmin > tmax)) return false;
```

étaient toujours vraie. Par conséquent, on sortait de la boucle trop d'avance, ce qui faisait en sorte que **colorResult** (la valeur que l'on attribue à **hit.scatter**) valait toujours **(0,0,0)**. Le fix était très simple, mais je réalise que je vais devoir gérer ce cas dans la fonction d'intersection principale aussi. Bref, voici des images qui démontrent la différence:



hit.scatter = (0,0,0) et $\sigma_a = 0.2$ et $\sigma_s = 0.9$



hit.scatter $\neq$ (0,0,0) et $\sigma_a = 0.2$ et $\sigma_s = 0.9$

À première vue, la différence est difficile à remarquer, cependant elle est tout de même présente. La plus grande différence est la couleur de la fumée. Dans le coin en haut à droite, dans la première

image, la fumée est considérablement plus foncée que dans la seconde.

De plus, j'ai finalement implémenté du bruit de perlin pour générer les densités dans mes voxels. Le résultat est une fumée plus réaliste qu'avant.

Ensuite, j'ai implémenter les autres types de **Ray Marcher** (Pas régulier, Pas régulier avec jitter, Centre du voxel, Centre du voxel avec jitter).

En ce moment, je travaille sur une nouvelle fonction de **Shading** pour pouvoir faire en sorte que le volume de fumée produise des ombres sur les objets de la scène. En ce moment, la fonction de **shading** est binaire. Soit que l'objet reçoit de la lumière, soit il n'en reçoit pas. La différence avec la fumée est que l'objet peut recevoir 50% (par exemple) de la lumière plutôt que juste 0% ou 100%. Malgré le fait que le changement ne semble pas si compliqué que ça, je n'ai pas encore été en mesure d'obtenir des résultats convaincants, alors mon objectif pour le moment est d'avoir un bon **Shadinfg**.