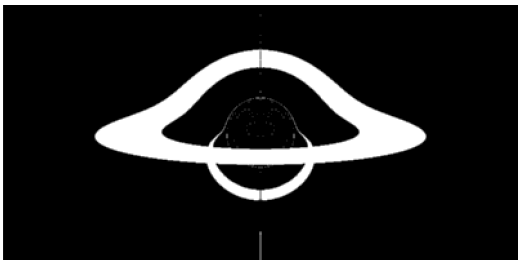
RayTracer en Relativité Générale



Objectif

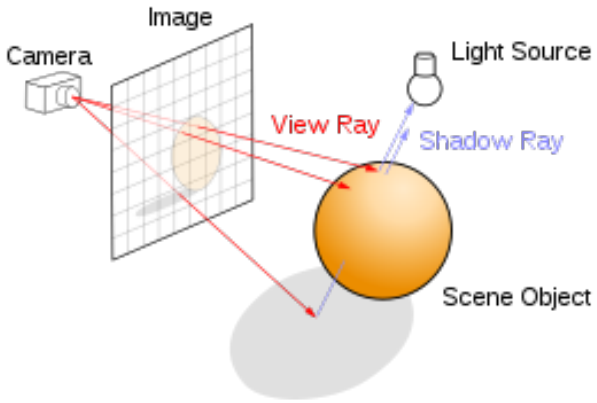
Créer un programme en Rust capable de réaliser un rendu (une image 2D) d'une scène simple dans un espace 3D non-euclidien. La technique adoptée est le *Ray Tracing*, qui s'appuie sur une simulation des trajets de chaque rayon de lumière qui relie un objet lumineux à un pixel de la caméra. La particularité est que dans notre situation, ces rayons ne sont pas rectilignes mais sont courbés par la courbure locale de l'espace-temps. L'intérêt du projet est de générer des images stylées de trous noirs.

On doit pour cela être capable de modéliser :

- La caméra (position, orientation, nombre de pixels x*y, fov)
- L'espace (entièrement défini par les coefficients de Christoffel $\left(\Gamma^i_{k,l}\right) \in \mathcal{M}_{4,4,4}(\mathbb{R})$
- Un photon, repéré par sa position $X = (t, x, y, z)\ ou\ (t, \rho, \theta, \phi) \in \mathbb{R}^4$, et son déplacement par rapport à un paramètre λ qui sert à paramétrer le rayon de lumière.
- Des objets simples placés dans la scène. Sur l'image ci-dessus, le seul objet présent est un anneau lumineux d'épaisseur nulle.
- Une image 360° du fond de ciel. On peut récupérer des images sur internet. Lorsqu'un rayon de lumière part vers l'infini sans intercepter d'objet, une luminosité lui est associée à partir de cette image 360°.

I – Principes de base du Ray Tracing

Au lieu de simuler tous les rayons de lumières qui émanent de chaque objet de la scène et détecter s'ils finissent par arriver sur le capteur, on fait le résonnement inverse en partant de la caméra : chaque pixel de la caméra « envoie » un ou plusieurs rayons de lumière qui se propagent dans l'espace 3D. Si ces rayons touchent un objet, c'est qu'il existe un trajet inverse (égal à celui qu'on vient de tracer) qui permet de remonter à la caméra. Ainsi le pixel « voit » la partie de l'objet qui a été interceptée par le rayon. Si cette partie est lumineuse, le pixel doit être éclairé.



Concrètement, on a le centre (x,y,z) de la caméra, et l'ensemble des positions (x_i, y_i, z_i) des pixels i de l'image. Pour chaque pixel i, on peut donc calculer le vecteur $\overrightarrow{d_i} = (x_i - x,\ y_i - y,\ z_i - z)$ qui représente la direction de base du rayon de lumière lorsqu'il part du pixel i. Dans un espace euclidien cette direction d est constante, mais en relativité générale son évolution est décrite par une équation différentielle (voir prochaine partie).

En espace euclidien, il suffit alors d'itérer la mise à jour de la position du « photon » :

$$\vec{X}_{i,k+1} = \vec{X}_{i,k} + \Delta l \times \vec{d}$$

jusqu'à intercepter un objet. l est le paramètre de la courbe (on ne le nomme pas t pour ne pas le confondre avec le temps propre du photon).

Pour la détection de la collision avec un objet, voir la partie à ce sujet.

II – Géodésiques en espace non-euclidien

En espace-temps non-euclidien, la seule différence est que le vecteur directeur d change en chaque position. Un objet quelconque (un photon dans notre cas) est repéré par sa position $s = (t, x, y, z)$ (ou coordonnées sphériques, etc). La métrique dépend du tenseur métrique d'ordre 2 g :

$$ds^2 = g_{i,j} ds_i ds_j \qquad (1)$$

(On somme sur les éléments i et j). Le champ tensoriel g peut être trouvé sur internet pour toute configuration de l'espace-temps que l'on veut simuler. Pour la solution de Schwartschild (trou noir immobile sans rotation), on a en coordonnées sphériques :

$$g_{0,0} = -\left(1 - \frac{r_s}{r}\right)$$
$$g_{1,1} = \frac{1}{1 - \frac{r_s}{r}}$$
$$g_{2,2} = r^2$$
$$g_{3,3} = r^2 sin(\theta)^2$$

On commence généralement l'indexation à 0 pour la coordonnée temporelle. On peut noter que la métrique est un tenseur diagonal dans ce cas, et que $g_{2,2}$ et $g_{3,3}$ sont identiques à un espace euclidien en coordonnées sphériques. À noter que g est exprimé dans le référentiel presque euclidien situé en $r = +\infty$ (on peut vérifier la valeur des coefficients dans ce cas).

Géodésiques

Une seule équation est nécessaire pour simuler le déplacement d'une particule dans l'espace-temps : l'équation des géodésiques. Toute particule avance à une vitesse constante c, la vitesse de la lumière :

$$\left(\frac{ds}{d\lambda}\right)^2 = c^2 \qquad (2)$$

Donc la norme de notre vecteur déplacement est toujours fixe ! En gros, si ma particule est immobile dans l'espace, c'est que sa direction de déplacement est colinéaire à la coordonnée t. Pour un photon, sa vitesse dans l'espace est c, donc sa coordonnée dt est toujours nulle (on en reparlera lol). Le trajet de l'objet est décrit par l'équation différentielle d'ordre 2 :

$$\frac{\mathrm{d}^2 s_i}{d\lambda^2} = \Gamma^i_{k,l} \frac{ds_k}{d\lambda} \frac{\mathrm{ds_l}}{\mathrm{d}\lambda} \qquad (3)$$

(On somme sur k et l encore une fois). λ est le paramètre de la courbe (trajectoire) que l'on trace, et donc le pas Δλ de l'intégration numérique peut être choisi arbitrairement. $\Gamma^i_{k,l}$ est le coefficient de Christoffel d'indice (i,k,l), qui dépend de la métrique g, et qui peut être trouvé dans la littérature.

Pour l'intégration numérique de l'équation différentielle, on utilisera une méthode du type RK4. Il est important de rappeler que Γ dépend de la position (t, r, θ, ϕ) dans l'espace ! C'est ce qui ralenti en partie cette intégration numérique. Heureusement, seuls quelques coefficients sont non-nuls pour les cas simples (comme pour le trou noir par exemple).

II – Collision avec objets

III – Fond de ciel 360°

IV – Pseudocodes très très simplifié

Fonctionnement général

N_MAX ← …
nombre_pixels ← …
Pixels ← Tableau(nombre_pixels, 0)
POUR p **DANS** [0, nombre_pixels [
- (X, dX) ← Initialisation_du_photon_partant_du_pixel(p)
- N ← 0
- **TANT QUE** non(A) **ET** N < N_MAX
  - Calcul_un_pas_d_integration(X, dX, pas)
  - A ← detecte_collision(X, dX)
  - N ← N+1
- **FIN TANT QUE**
- **SI** A=VRAI **ALORS**
  - Pixels[p] ← Pixels[p] + 1

FIN POUR

On y reviendra mdr.

Pour l'intégration :

La méthode de Runge-Kutta d'ordre quatre avec dérivée seconde [modifier | modifier le code]

Dans le cas $y'' = f(t, y, y'), \quad y(t_0) = y_0, \quad y'(t_0) = y'_0$, nous pouvons décomposer le problème en un système d'équations :

$$\begin{cases} y' = g(t, y, z) \\ z' = s(t, y, z) \end{cases}$$

avec ici $y' = g(t, y, z) = z$ et $s = f$.

En appliquant la méthode RK4 à chacune de ces équations, puis en simplifiant nous obtenons[1] :

$$k_1 = f\left(t_n, y_n, y'_n\right)$$

$$k_2 = f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}y'_n, y'_n + \frac{h}{2}k_1\right)$$

$$k_3 = f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}y'_n + \frac{h^2}{4}k_1, y'_n + \frac{h}{2}k_2\right)$$

$$k_4 = f\left(t_n + h, y_n + hy'_n + \frac{h^2}{2}k_2, y'_n + hk_3\right)$$

On en déduit y_{n+1} et y'_{n+1} grâce à :

$$y_{n+1} = y_n + hy'_n + \frac{h^2}{6}\left(k_1 + k_2 + k_3\right)$$

$$y'_{n+1} = y'_n + \frac{h}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right)$$

Dans notre cas, on estime la dérivée seconde qui est le vecteur $\frac{d^2s}{d\lambda^2}$ donné par l'équation (3).