

# Synthèse d'images 3D

## TIPE

Oscar Buon

7 juin 2021

# Objectifs

- Concevoir un programme reposant sur l'algorithme du tracé de rayon afin de créer des images de synthèse.
- Optimiser le programme et augmenter la qualité des images.
- Utiliser la synthèse d'images 3D pour prévisualiser une pièce de bâtiment.

# Table des matières

## 1 Tracé de rayon

- Problématique
- Modélisation
- Description de l'algorithme

## 2 Implémentation et amélioration

- Parallélisation
- Priorisation
- Traitement d'image
- Tests d'intersection

## 3 Résultat final

- Application
- Conclusion
- Annexe

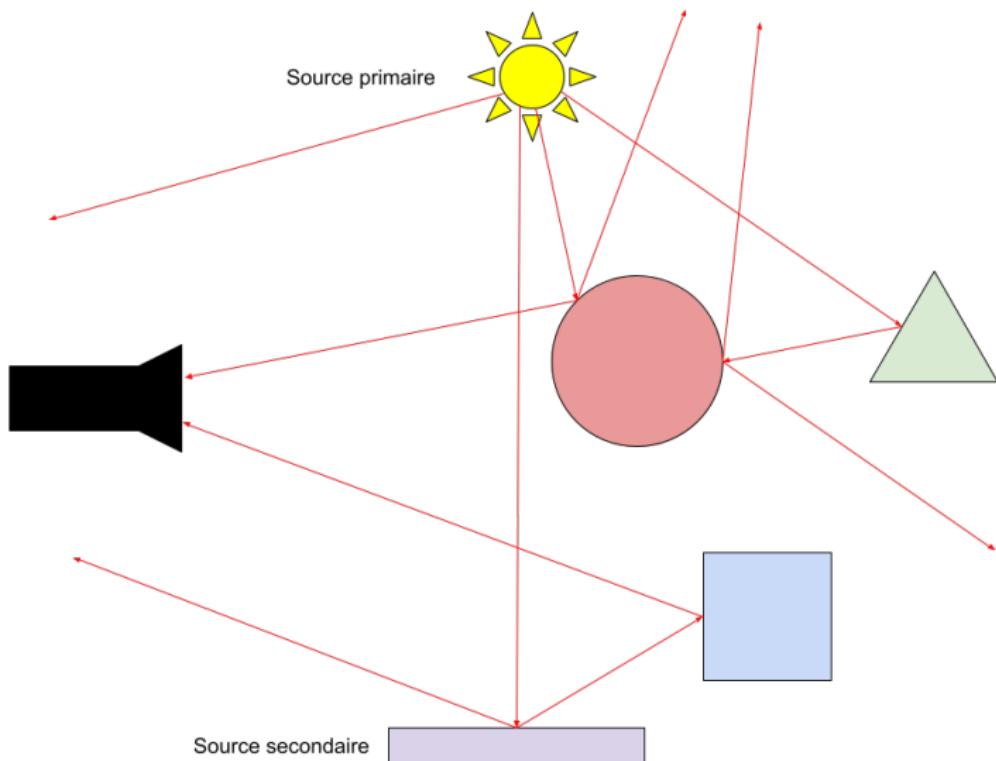
Scène 3D : ensemble d'objets définis via des points, vecteurs et autres propriétés stockés dans la mémoire de l'ordinateur.

Infographie 3D : créer une image matricielle 2D à partir d'une scène numérique 3D.

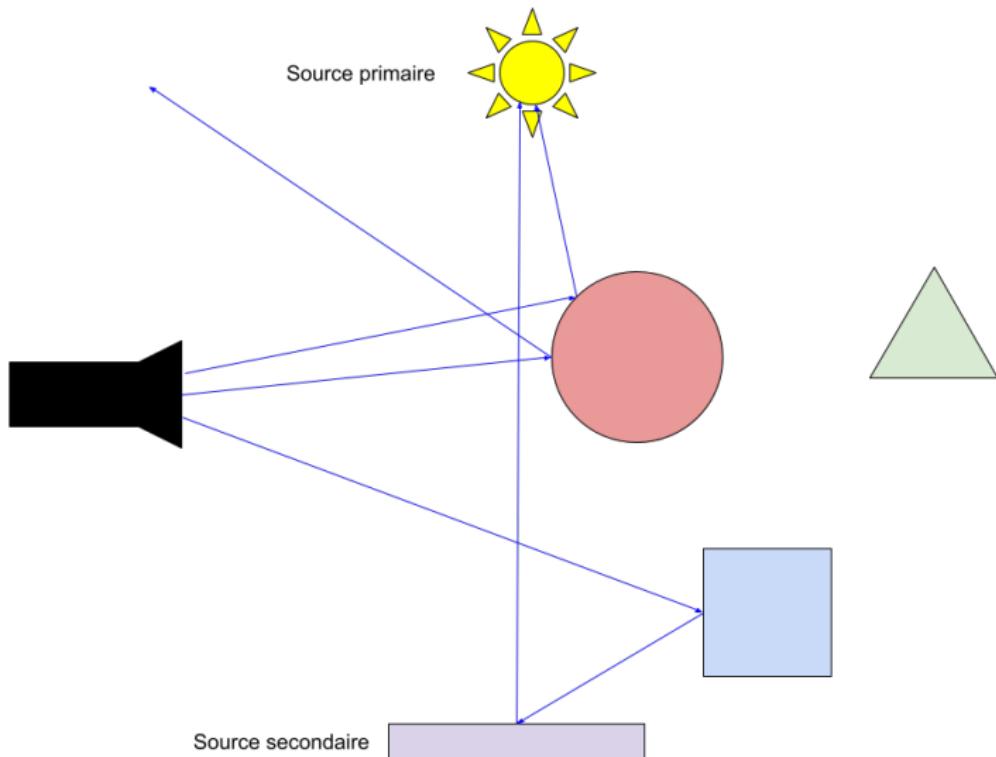
Tracé de rayon : un algorithme permettant de faire de l'infographie 3D.

- On modélise la lumière comme des rayons lumineux.
- Les objets sont délimités par des surfaces géométriques.
- Lorsqu'un rayon intersecte une surface, plusieurs choses peuvent se passer (absorption, réflexion, transmission...) en fonction des propriétés de l'objet.

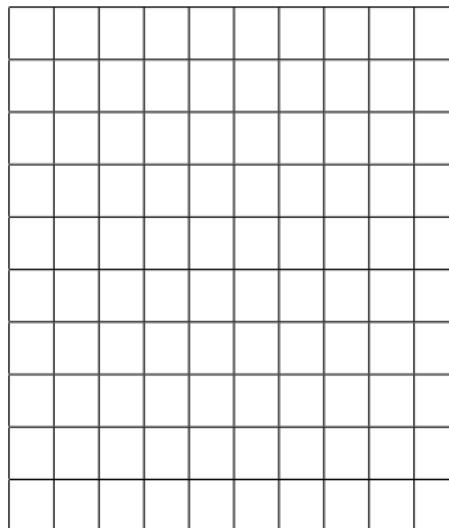
# Modélisation de la lumière



# Principe de Fermat



# Image matricielle



On définit une fonction de lancer de rayon qui à un point d'origine et une direction :

- Calcul quel est l'objet intersecté le plus proche.
- Récupère les propriétés optiques de l'objet.
- En fonction de celles-ci peut récursivement lancer de nouveaux rayons : rayon réfléchi, rayon transmis...
- Renvoie une information de couleur dépendant des propriétés de l'objet et des couleurs renvoyées par les nouveaux rayons.

Pour chaque pixel, on lance un rayon depuis la position de la caméra vers la direction du pixel. On obtient ainsi une image matricielle.

## Exemple

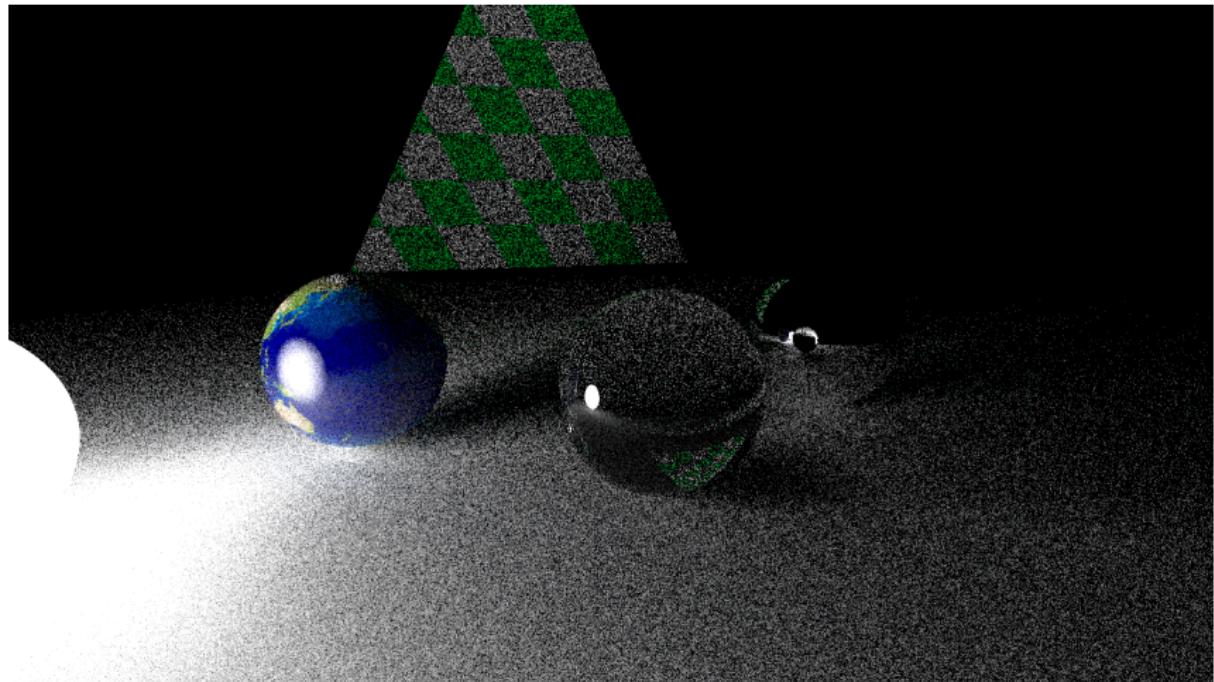


FIGURE – 100 samples, 105 seconds

En faisant tourner 4 instances de l'algorithme en parallèle fonctionnant sur 4 coeurs différents, on passe de 105 à 51 secondes. L'image reste la même.

# Remarque

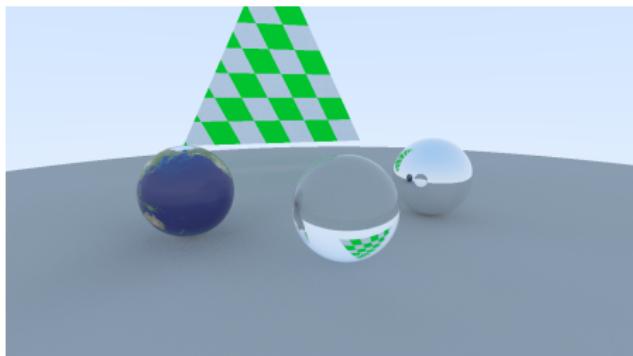
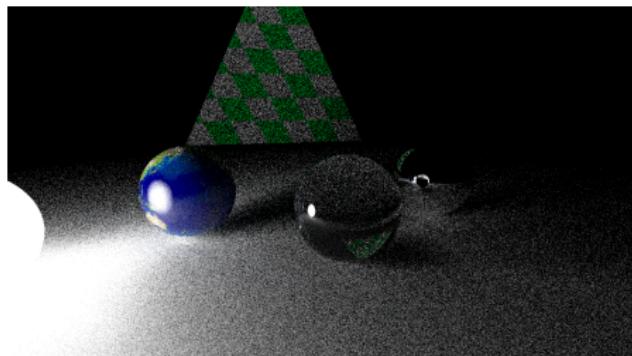
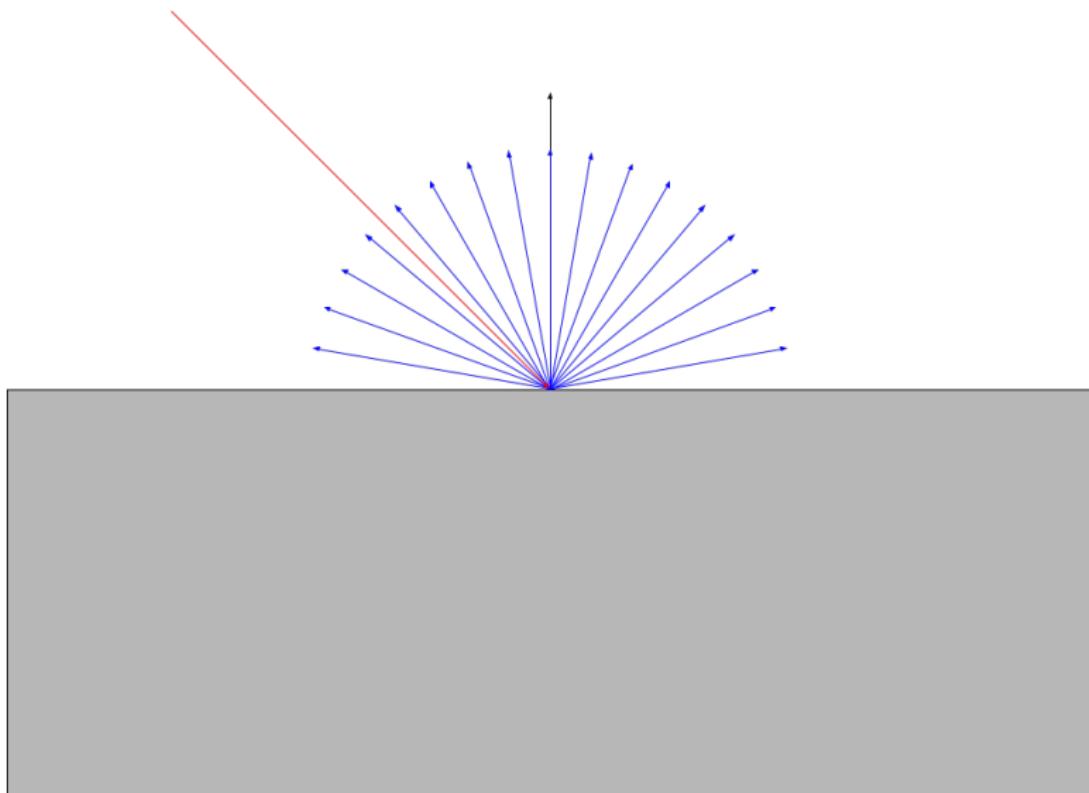
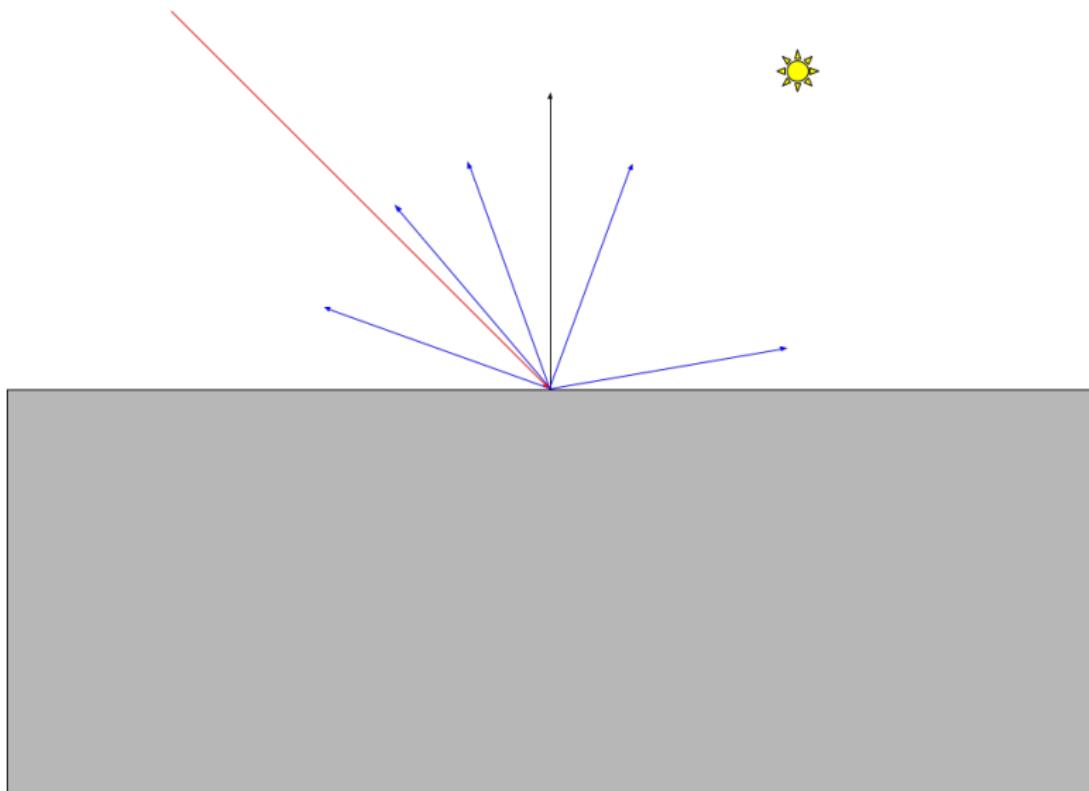


FIGURE – 100 samples, 50 seconds

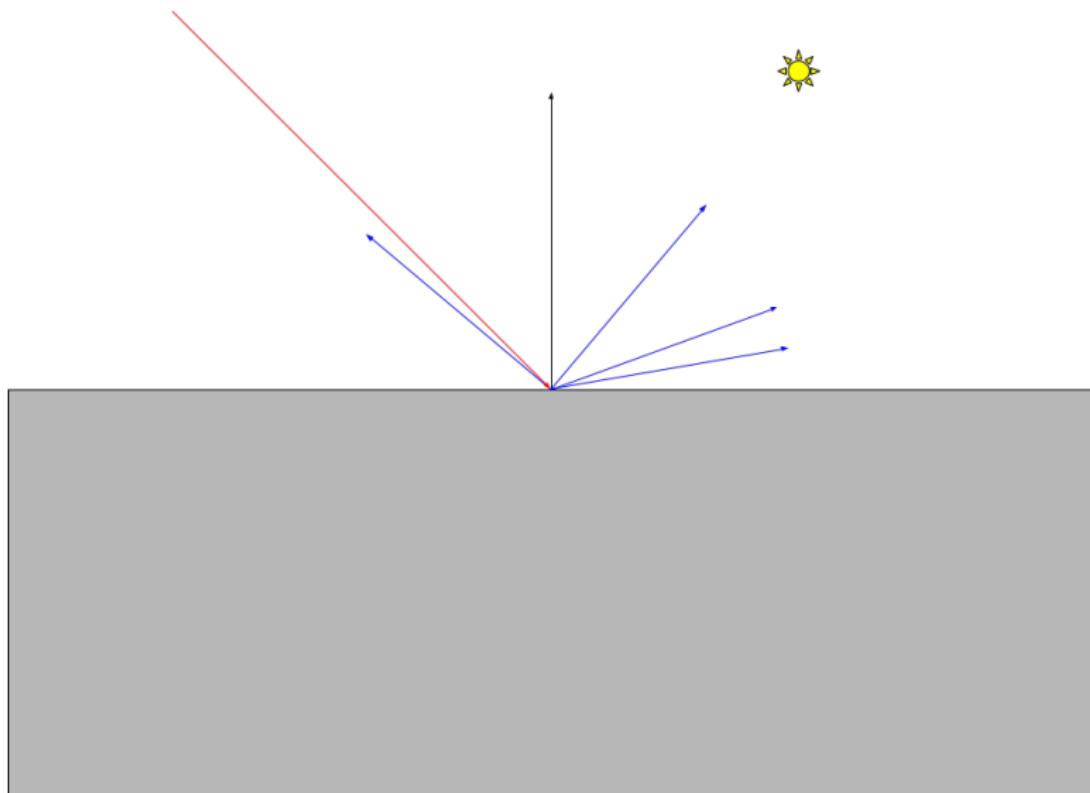
# Objets diffus



# N'échantillonne pas la source lumineuse



# Échantillonne la source lumineuse



## Sans priorisation

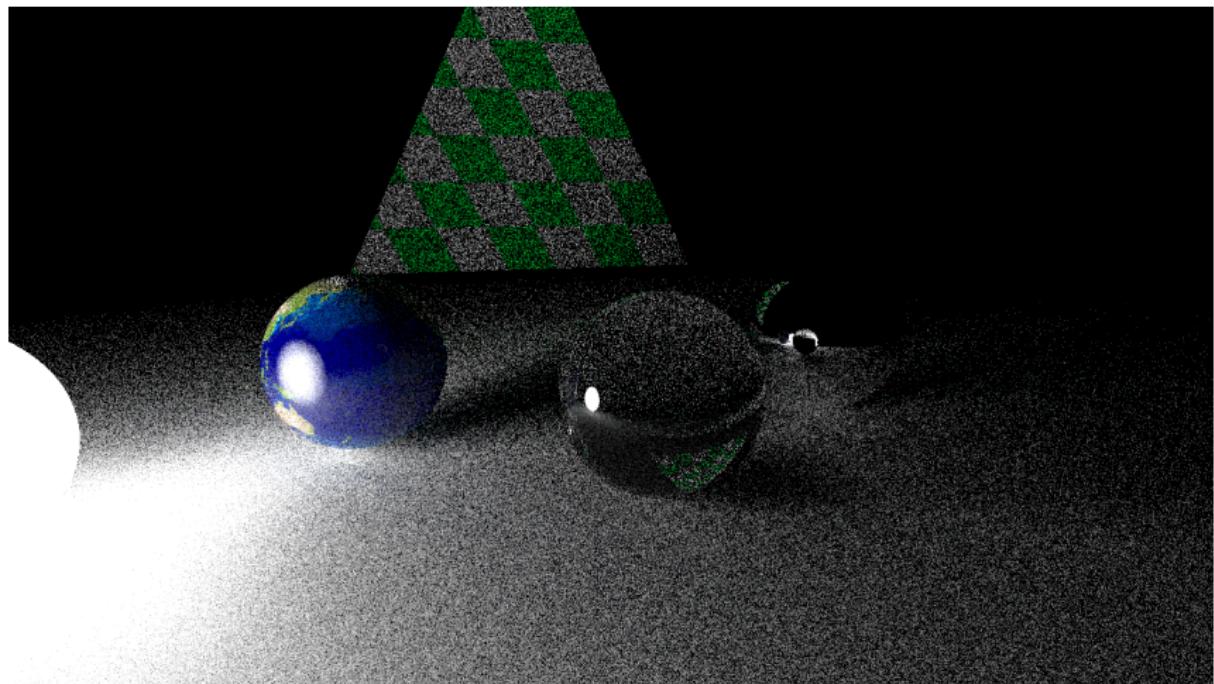


FIGURE – 100 samples, 50 seconds

## Avec priorisation

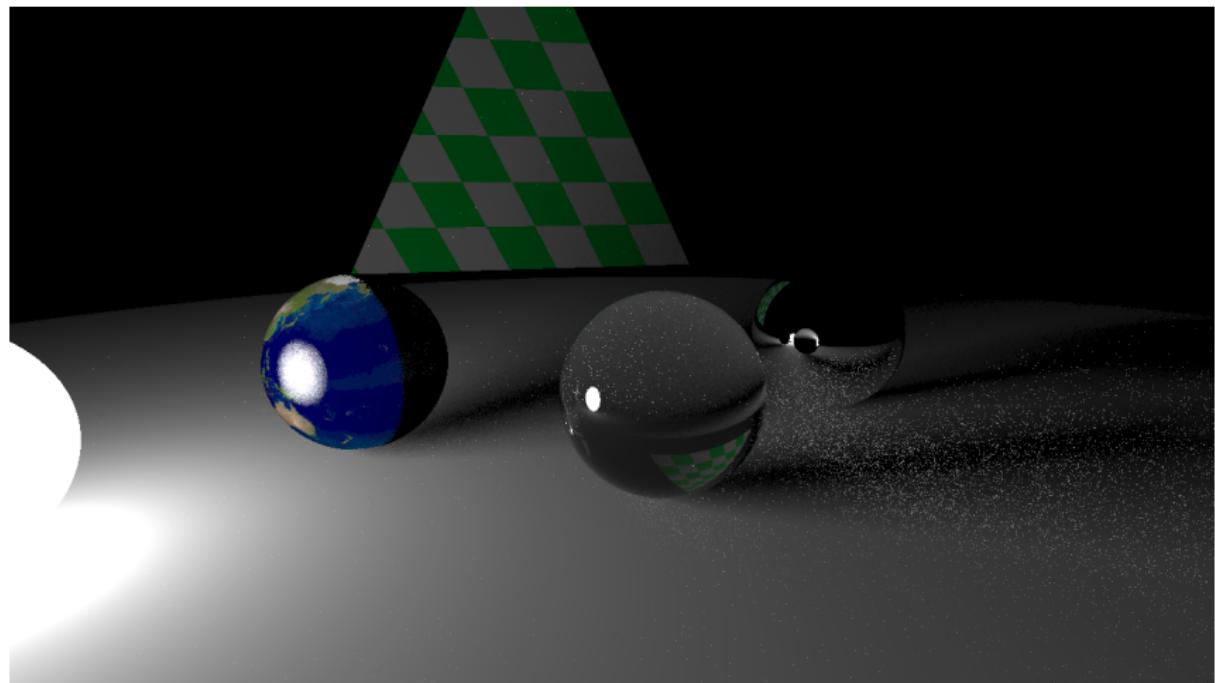


FIGURE – 100 samples, priority 0.5, 46 seconds

## Idée

Produire une image plus rapidement mais de moins bonne qualité, pour ensuite l'améliorer avec des algorithmes de traitement d'images.

## Sans traitement

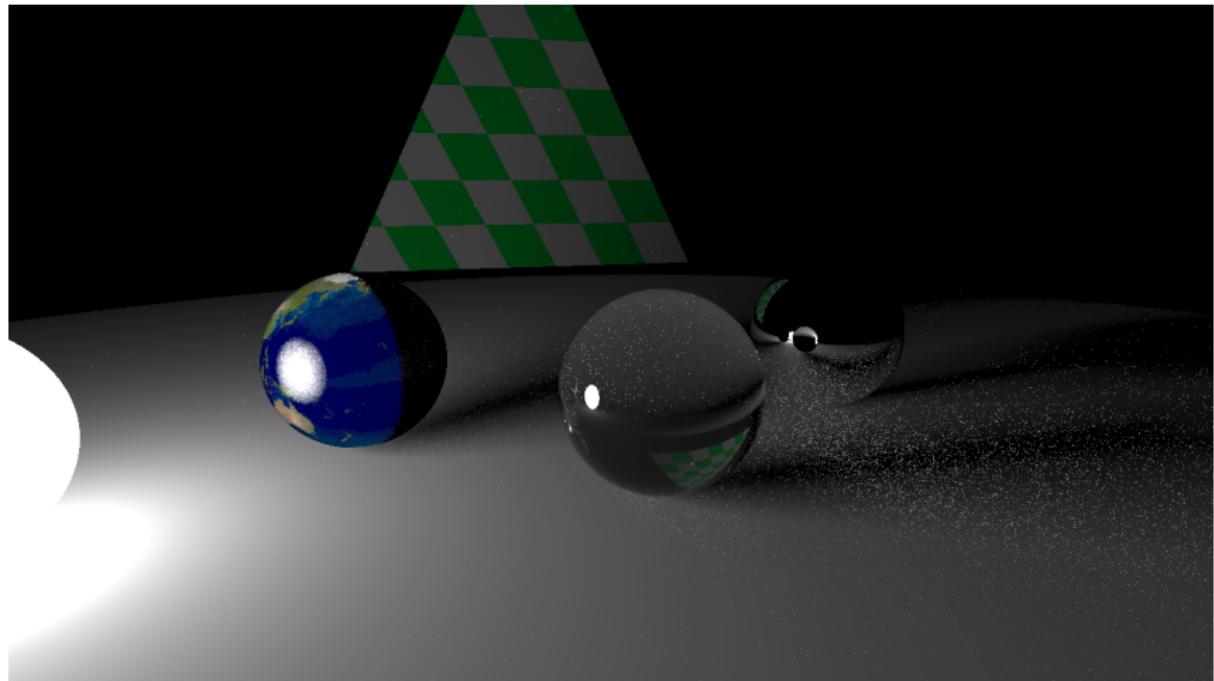


FIGURE – 100 samples, priority 0.5, 46 seconds

# Avec filtre gaussien

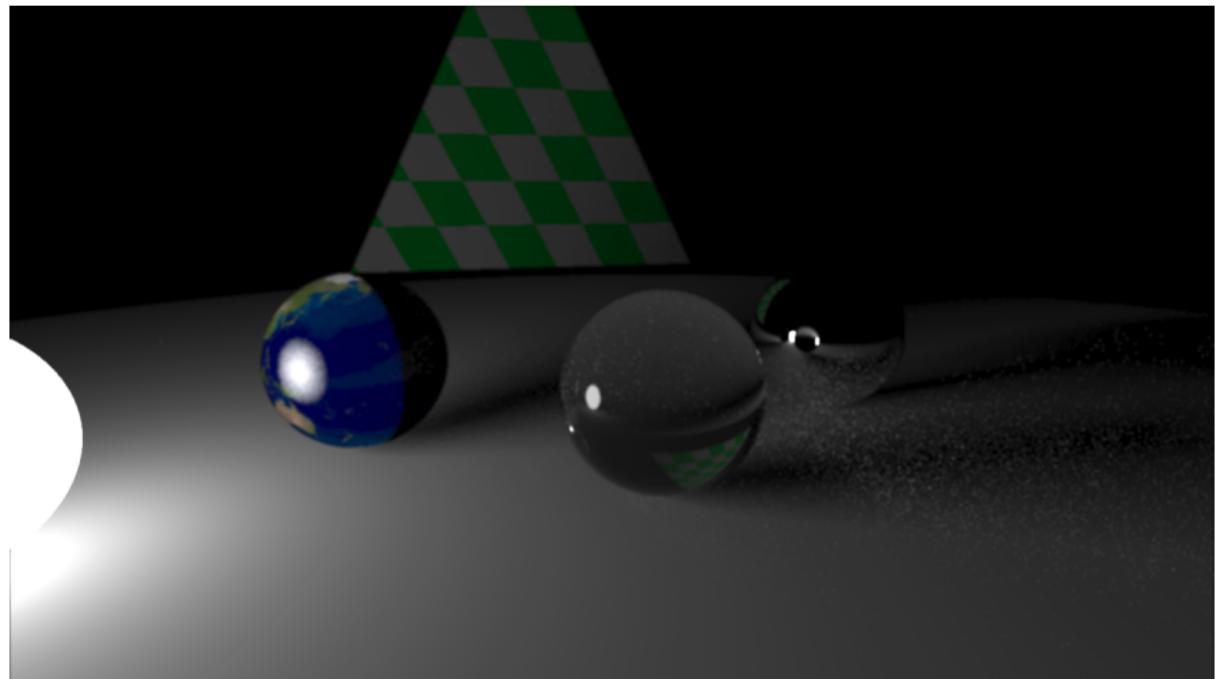


FIGURE – 100 samples, priority 0.5, gaussian 5/1.6, 49 seconds

# Avec filtre gaussien

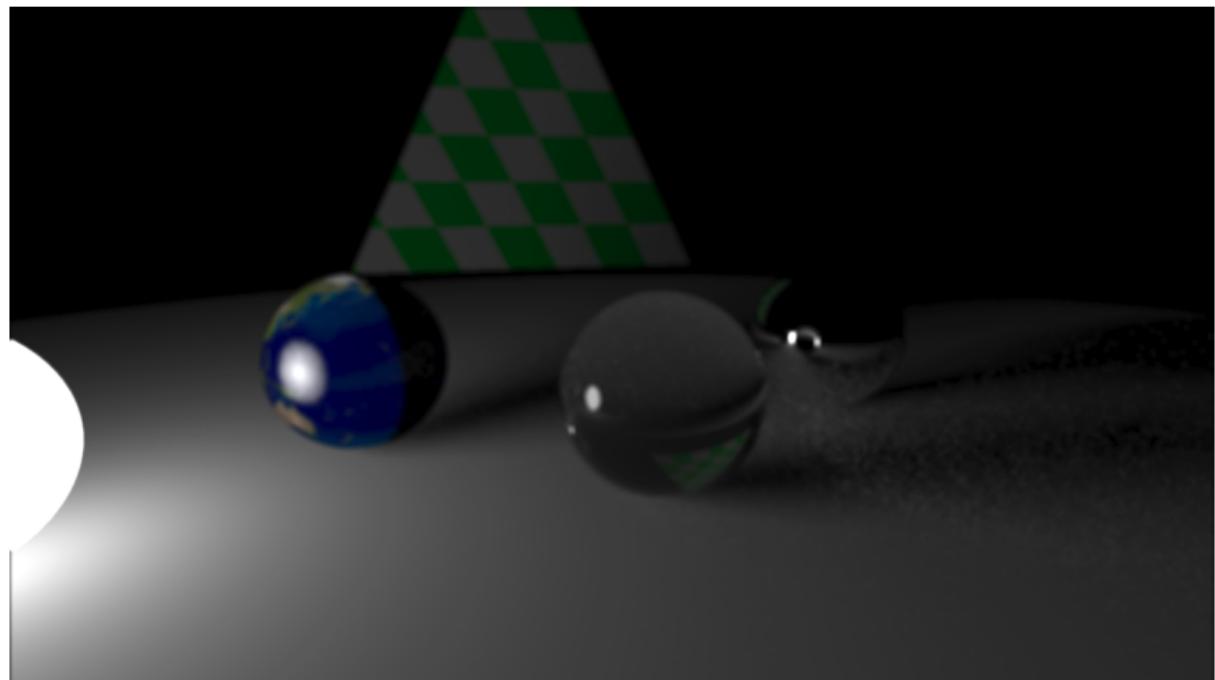


FIGURE – 100 samples, priority 0.5, gaussian 10/3.2, 56 seconds

# Un algorithme plus avancé

- On sépare l'image en deux parties : la couleur propre des objets et leur illumination.
- On applique un filtrage à l'illumination seulement :
  - Filtre médian : pour chaque pixel, on prend la médiane des pixels voisins.
  - Filtre gaussien : chaque pixel est moyenné avec ses voisins avec un coefficient  $G(x, y) = \frac{e^{\frac{x^2+y^2}{2\sigma^2}}}{2\pi\sigma^2}$ .
- On fusionne les deux images.

## Sans traitement

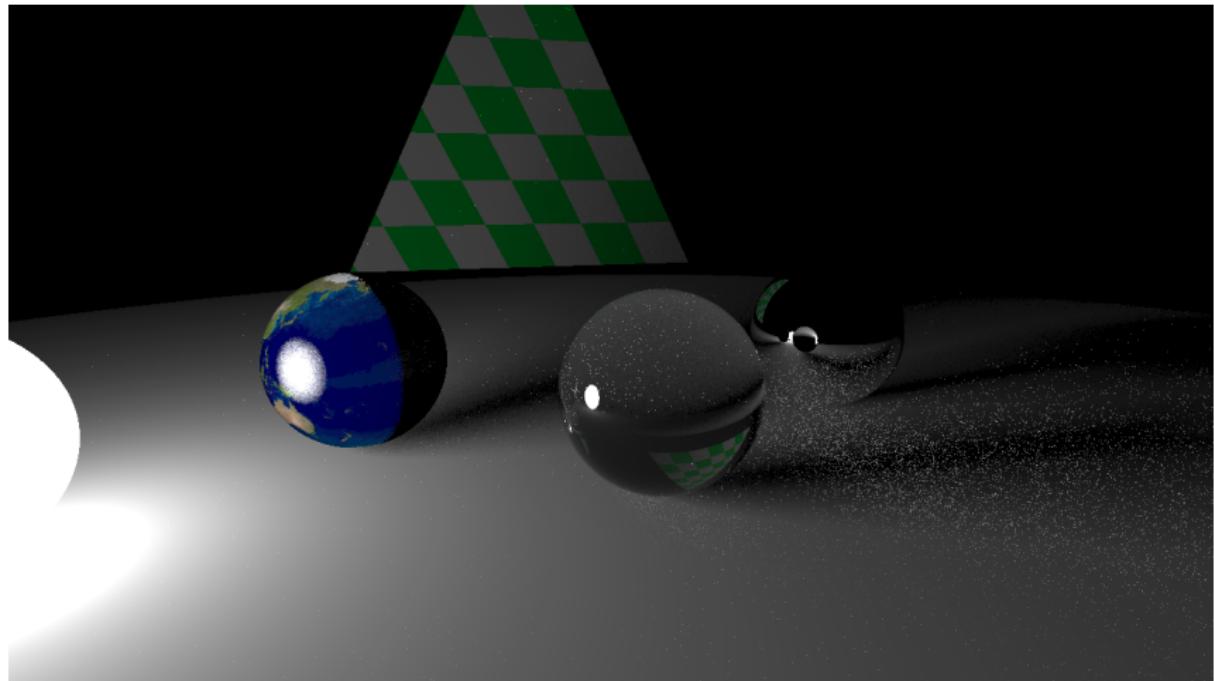


FIGURE – 100 samples, priority 0.5, 46 seconds

## Avec traitement

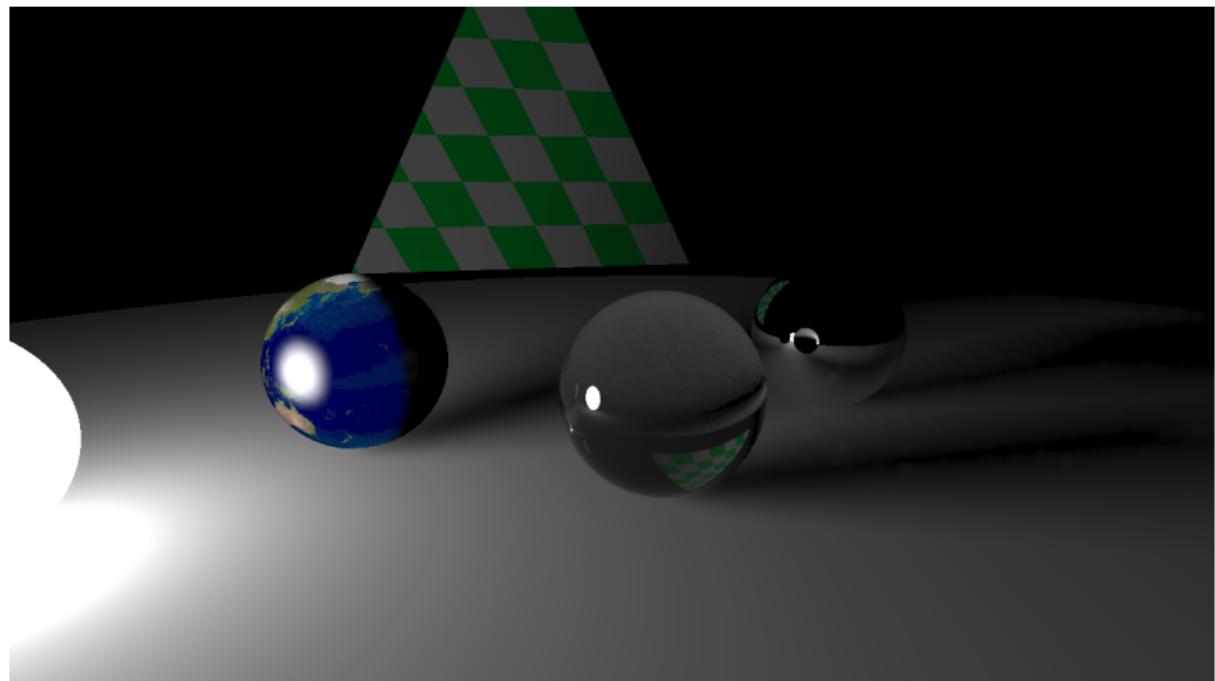
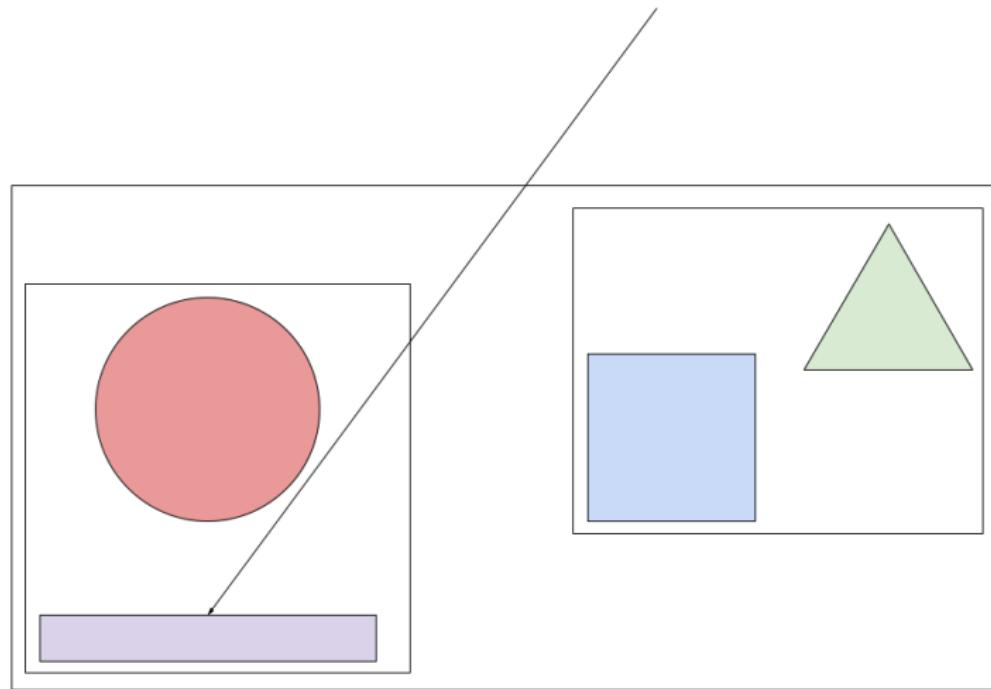


FIGURE – 100 samples, priority 0.5, processing 2/0.8, 62 seconds

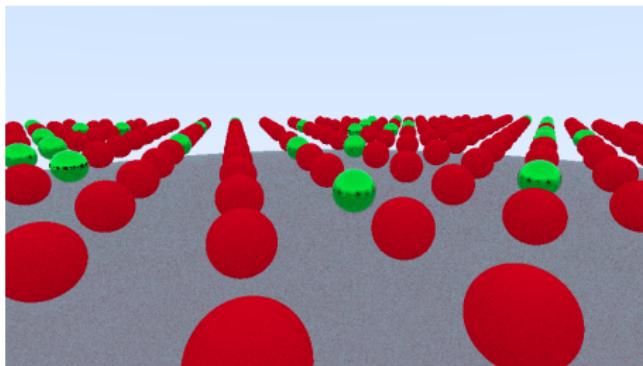
## Idée

En rangeant intelligemment les objets, on pourrait limiter le nombre de tests d'intersection.

# Bounding Volume Hierarchy (BVH)



# Comparaison



**FIGURE** – Pour une scène avec 400 sphères et 10 échantillons par pixel, on passe de 96 à 58 secondes.

# Pièce de bâtiment avec murs en bricks

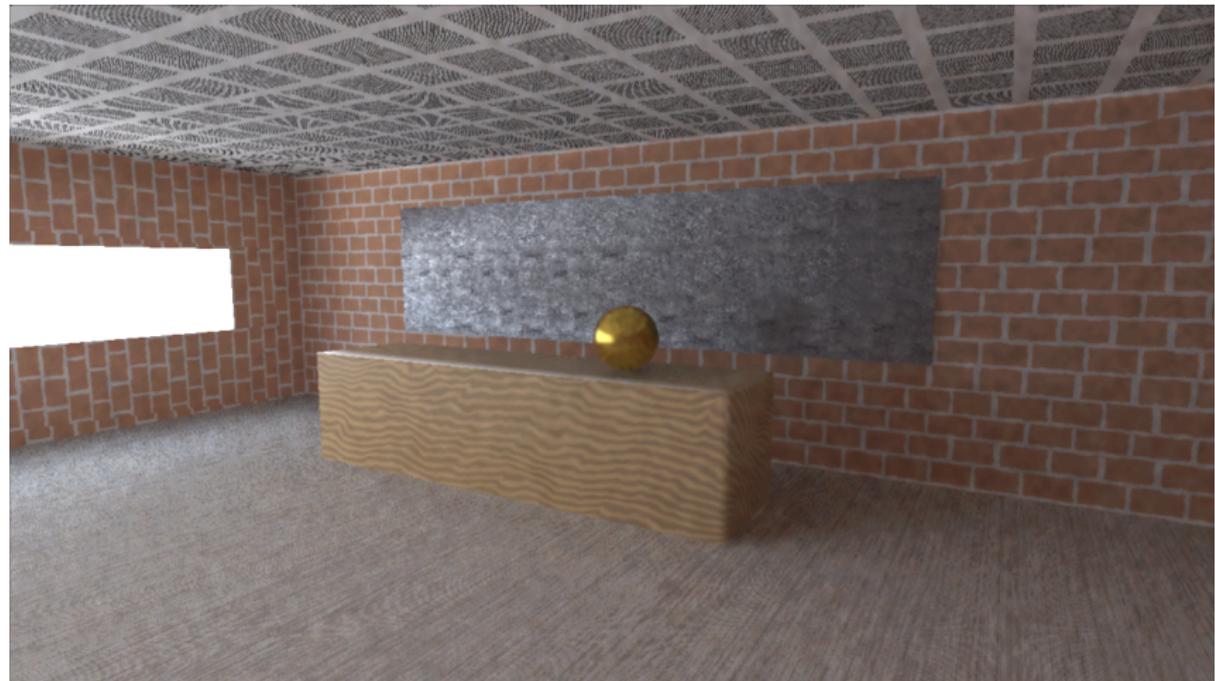


FIGURE – 100 samples, processing 2/0.8, 1274 seconds, average 0.148

# Pièce de bâtiment avec murs peints

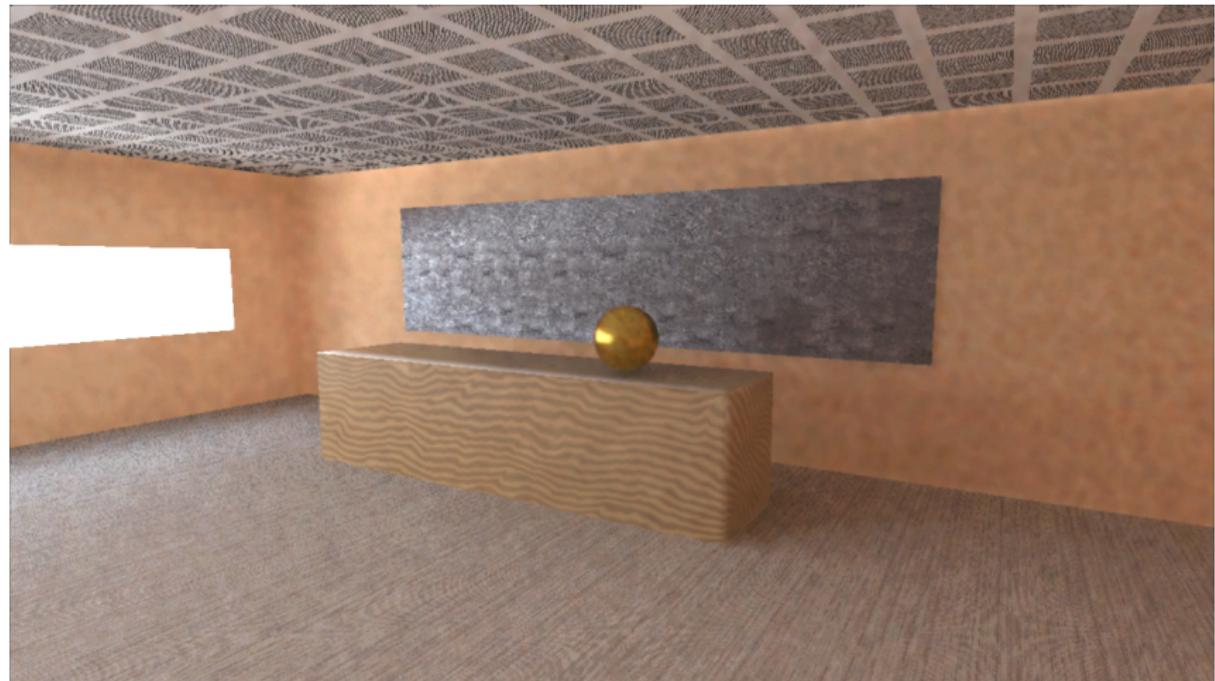


FIGURE – 100 samples, processing 2/0.8, 1224 seconds, average 0.172

# Tableau moins rugueux

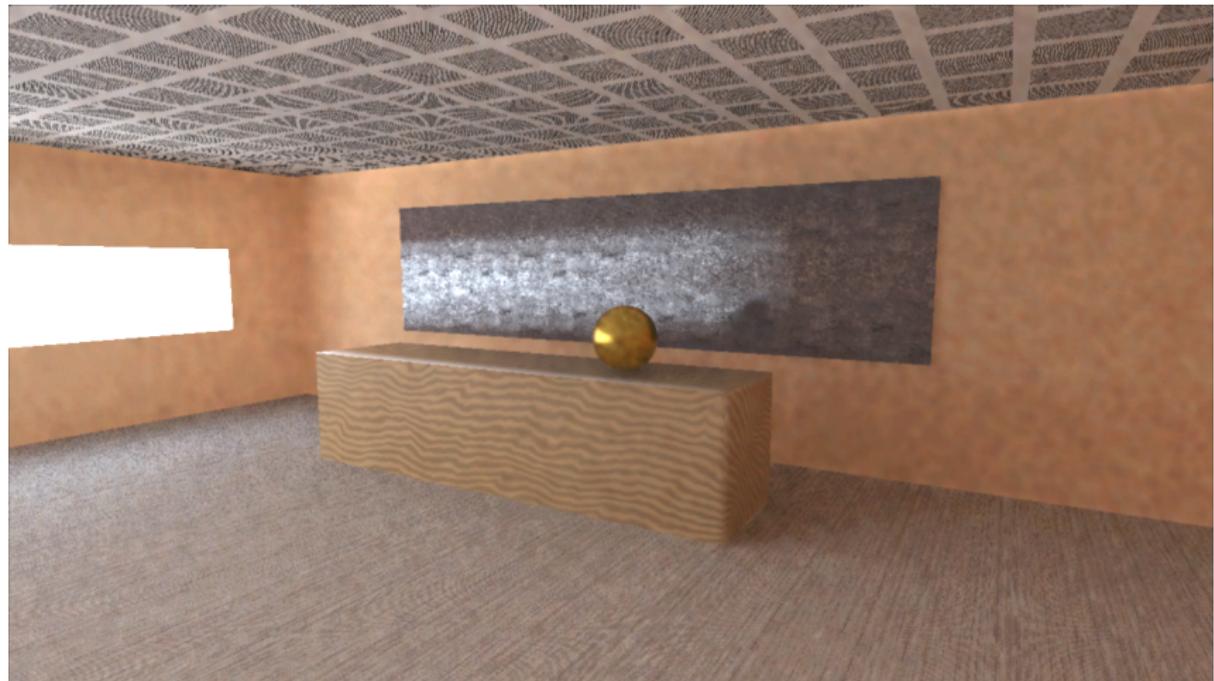


FIGURE – 100 samples, processing 2/0.8, 1418 seconds, average 0.172

## Conclusion et pistes d'améliorations

- L'implémentation de l'algorithme fonctionne.
- Il est possible de l'utiliser dans le domaine de l'architecture.
- Le programme a pu être optimisé, cependant pas suffisamment pour être utilisé en temps réel. Il existe tout de même plusieurs pistes d'améliorations :
  - utiliser l'accélération matérielle ;
  - précalculer certains éléments.

# Pièce de bâtiment avec murs peints

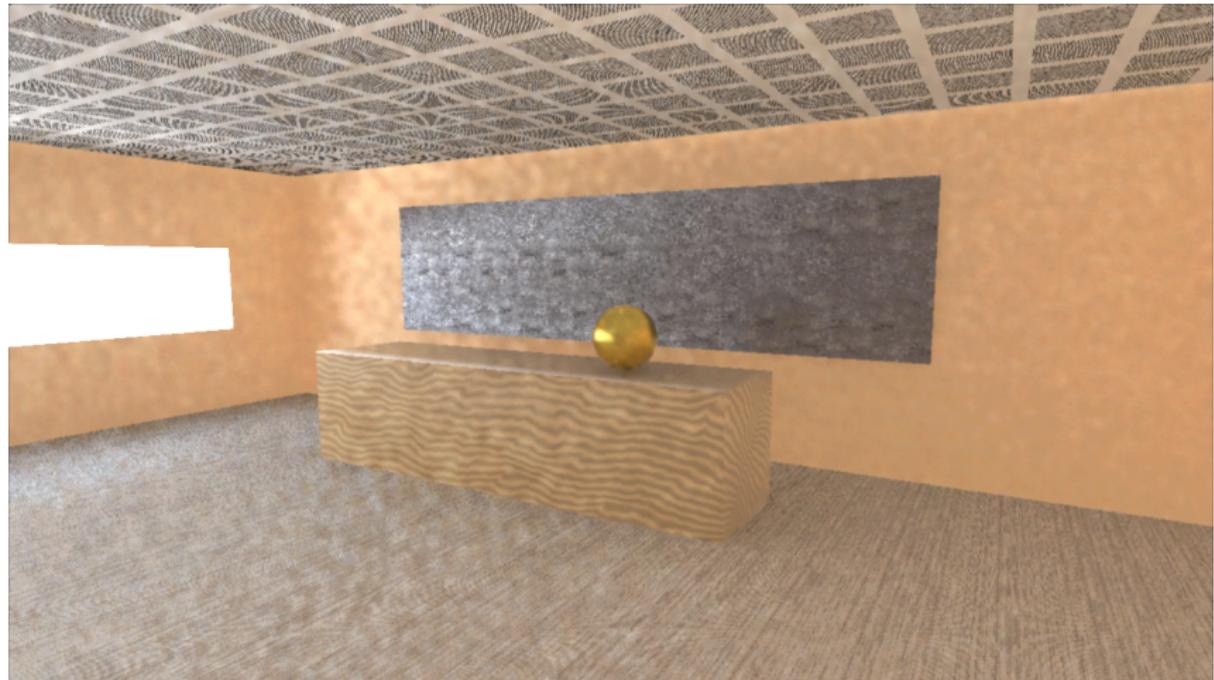


FIGURE – light 0.172, 25 samples, processing 2/0.8, 70 seconds, average 0.201

# Physicaly Based Rendering (PBR)

Comment modéliser la lumière de manière physiquement plausible ?

Les critères du PBR :

- ① Définir pour chaque surface une valeur de rugosité.
- ② Respecter le principe de conservation de l'énergie lumineuse.
- ③ Se baser sur une fonction de réflectivité bidirectionnelle.

# Grandeur importantes

L'énergie  $Q = \frac{hc}{\nu}$  en  $J$ .

La puissance ou flux  $\Phi = \frac{\partial Q}{\partial t}$  en  $W$ .

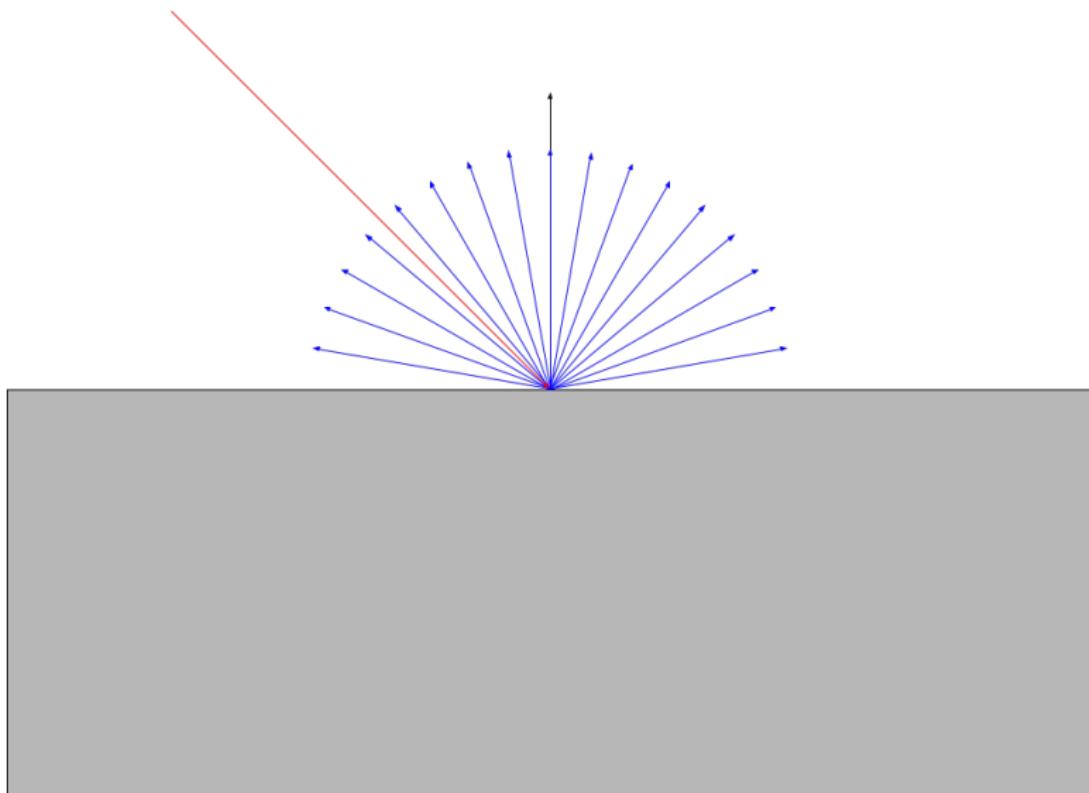
L'irradiance et l'exitance  $E = \lim_{\Delta A \rightarrow 0} \frac{\Delta \Phi}{\Delta A}$  en  $W.m^{-2}$ .

La luminance  $L = \lim_{\Delta \omega \rightarrow 0} \frac{\Delta E}{\Delta \omega}$  en  $W.m^{-2}.sr^{-1}$ .

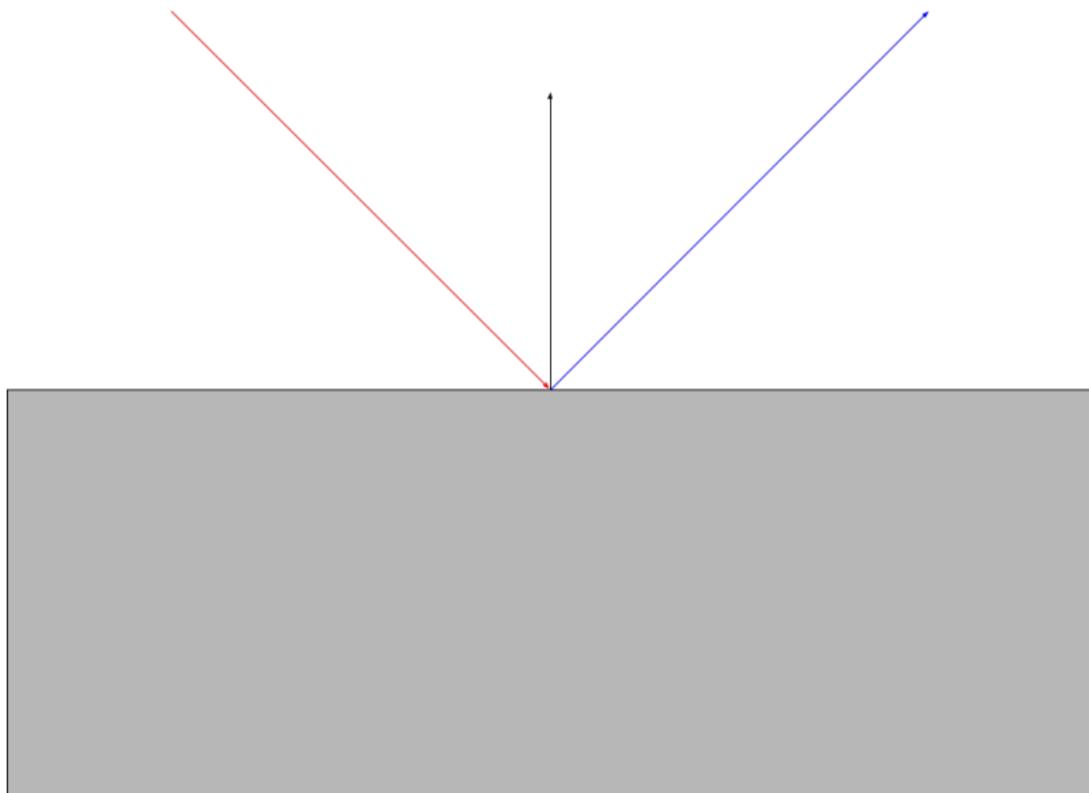
# Équation de rendu

$$\begin{aligned}L_o(x, \omega_o, \lambda, t) &= L_e(x, \omega_o, \lambda, t) \\&+ \int_{\Omega} f(x, \omega_i, \omega_0, \lambda, t) L_i(x, \omega_i, \lambda, t) (\omega_i \cdot n) d\omega_i\end{aligned}$$

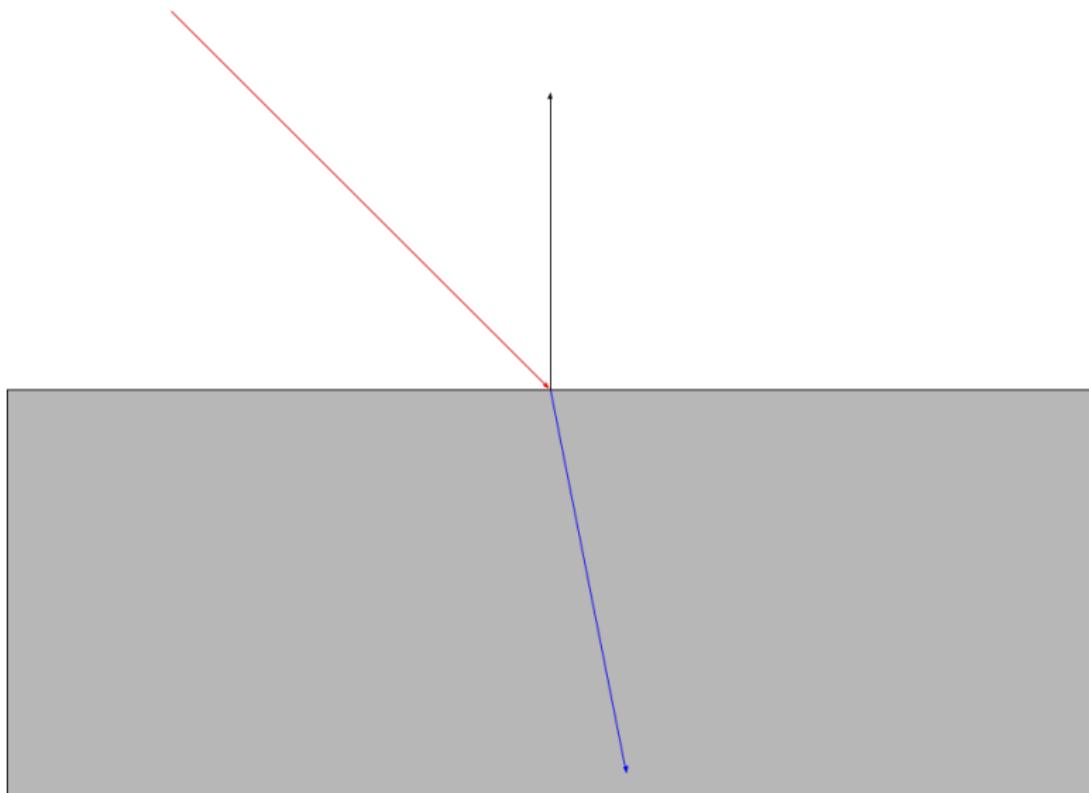
# Lambertien



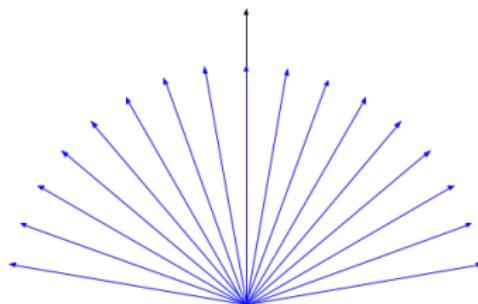
# Réflexion spéculaire



# Transmission spéculaire



# Lampe



On peut modéliser une surface réflexive réelle avec un nombre restreint de paramètres :

- Spectre diffus
- Spectre spéculaire
- Spectre émis
- Rugosité
- Normale