

# Rapport Tp3 ISIR

Jeremy Vacher

Mars 2024

## Table des matières

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Réduisons les temps de calcul</b>	<b>2</b>
<b>3</b>	<b>Sources lumineuses surfaciques : le quad</b>	<b>2</b>
<b>4</b>	<b>Moins de bruit</b>	<b>3</b>

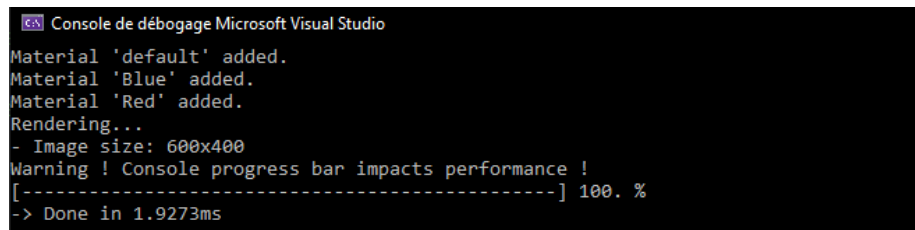
## 1 Introduction

Dans ce TP, nous verrons une méthode nous permettant d'améliorer la vitesse d'exécution de notre programme. Nous verrons aussi une autre source lumineuse permettant d'adoucir les ombres et rendre un résultat plus réel.

## 2 Réduisons les temps de calcul

Pour réduire les temps de calcul nous rajoutons une ligne permettant de réaliser dans une zone parallèle les lignes traitant nos pixels.

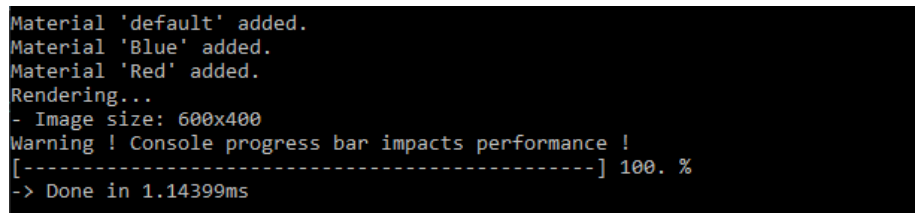
Avant de rajouter la ligne permettant la parallélisation notre temps d'exécution est de :



```
Microsoft Visual Studio Debug Console
Material 'default' added.
Material 'Blue' added.
Material 'Red' added.
Rendering...
- Image size: 600x400
Warning ! Console progress bar impacts performance !
[-----] 100. %
-> Done in 1.9273ms
```

FIGURE 1 – Temps d'exécution sans zone parallèle

En rajoutant **pragma omp parallel** for notre temps d'exécution est de :



```
Material 'default' added.
Material 'Blue' added.
Material 'Red' added.
Rendering...
- Image size: 600x400
Warning ! Console progress bar impacts performance !
[-----] 100. %
-> Done in 1.14399ms
```

FIGURE 2 – Temps d'exécution avec zone parallèle

## 3 Sources lumineuses surfaciques : le quad

Dans cette partie, nous allons ajouter un nouveau type de source lumineuse ayant la forme d'un **quadrilatère**. Pour ce faire, nous allons rajouter une nouvelle classe prenant en paramètre la **position** d'un coin, les deux **arrêtes** de la position, la **couleur** et la **puissance** de la source lumineuse. Nous avons aussi la **normale** et l'**aire** de la surface quadratique.

Nous allons commencer par réaliser une méthode permettant de retourner un **LightSample**. Pour cela, on prend une **position aléatoire** sur notre quadrilatère en utilisant les arrêtes. Ensuite on calcule la **direction** du rayon entre la position aléatoire sur le quadrilatère et le point observé.

On calcule la **PDF**, en utilisant la **distance** entre la position aléatoire et le point observé, l'**angle** entre la normale au quadrilatère et la **direction** du

rayon. On calcule aussi le **facteur géométrique** qui correspond à la distance au carré divisé par notre angle. Enfin, on calcule la **radiance** en multipliant la puissance de la lumière à la couleur et on retourne le LightSample.

```
LightSample QuadLight::sample( const Vec3f & p_point ) const
{
    Vec3f _posRandom = _position + ( randomFloat() * _u )
    + ( randomFloat() * _v );
    float _distance = glm::distance( _posRandom, p_point );
    Vec3f _direction = glm::normalize( _posRandom - p_point );
    float cosAngle = glm::dot( _n, _direction );
    float _geometricFactor = ( _distance * _distance ) / cosAngle;
    float _pdf = ( 1.f / _area ) * _geometricFactor;
    Vec3f _radiance = ( _color * _power ) / _pdf;

    return LightSample( _direction, _distance, _radiance, _pdf );
}
```

On obtiens alors l'image ci-dessous :

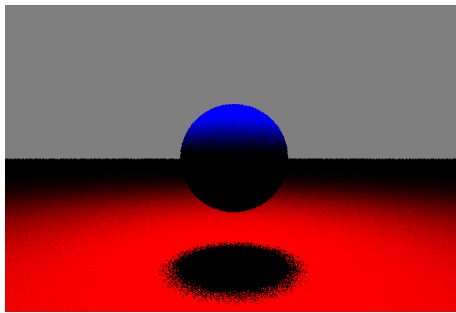
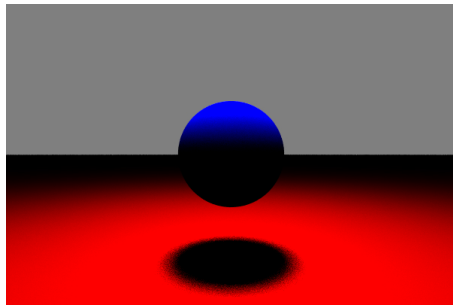


FIGURE 3 – Résultat de l'exercice 2

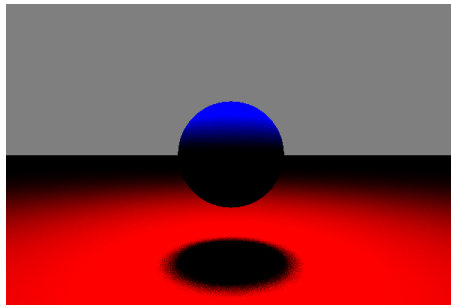
## 4 Moins de bruit

Dans cette partie, nous allons rajouter un attribut permettant de lancer plusieurs **rayons d'ombrages** et un **booléen** permettant de savoir si la sources lumineuses est surfaciques. Dans l'intégrateur on rajoute une condition pour vérifier que nous avons une surface et une boucle pour lancer nos rayons d'ombrage. Enfin, on calcule notre couleurs finales en divisant par le nombre de rayon d'ombrage.

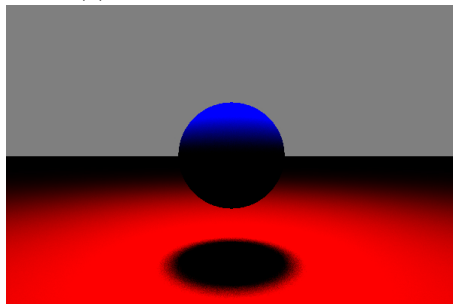
Lorsque l'on rajoute l'**anti-aliasing** nous obtenons un meilleur résultat puisque l'on lance plusieurs rayons pour chaque pixel de l'image.



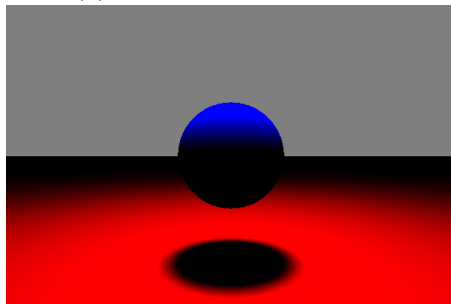
(a) Résultat de l'exercice 3 a



(b) Résultat de l'exercice 3 b



(c) Résultat de l'exercice 3 c



(d) Résultat de l'exercice 3 d

FIGURE 4 – Résultat de l'exercice 3