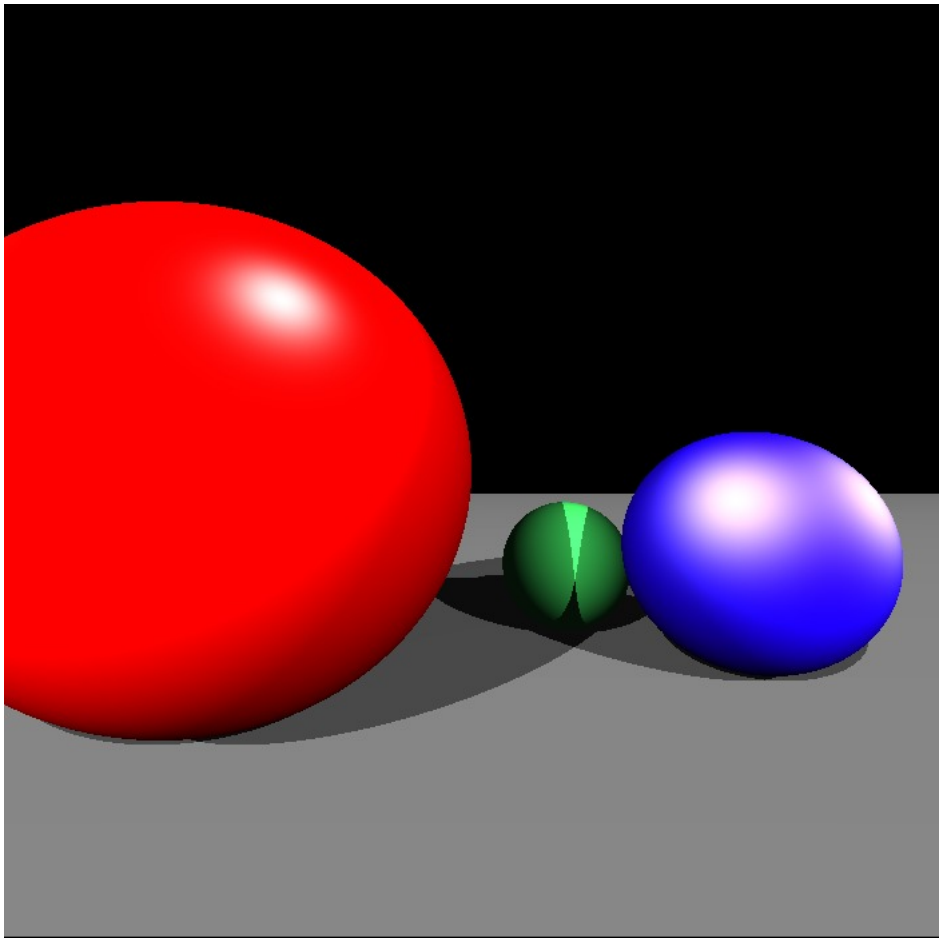
TP RENDU RAY TRACING

Valentin Bollier
Paul Gueneau

October 2020



1 Introduction

L'objectif du rendu projectif est d'obtenir une image 2D à partir d'un modèle 3D, pour cela il existe plusieurs approches dans ce TP nous allons nous intéresser à la méthode du ray tracing qui consiste à envoyer des rayons depuis la caméra sur différents objets de la scène 3D. Dans un premier temps nous verrons comment se traduit mathématiquement l'intersection des rayons avec les primitives de la scène 3D et comment le coder puis dans la seconde nous implémenterons le lancer de rayons dans la scène. On conclura sur les différences entre la méthode de rendu projectif par maillage et la méthode par lancer de rayons et leurs avantages et inconvénients respectifs.

2 Intersection avec les primitives

2.1 Cas du plan

Question 7

On a le système suivant :

$$(S) = \begin{cases} \langle x - x_p, n_p \rangle = 0 \\ x = x_0 + tu \end{cases}$$

$\Leftrightarrow$

$$(S) = \begin{cases} \langle x_0 + tu - x_p, n_p \rangle = 0 \\ x = x_0 + tu \end{cases}$$

$\Leftrightarrow$

$$(S) = \begin{cases} t\langle u, n_p \rangle + \langle x_0 - x_p, n_p \rangle = 0 \\ x = x_0 + tu \end{cases}$$

Et finalement :

$$(S) = \begin{cases} t = \frac{\langle x_p - x_0, n_p \rangle}{\langle u, n_p \rangle} \\ x = x_s + tu \end{cases}$$

Le résultat après l'implémentation de l'intersection du rayon avec le plan est donné figure 1.

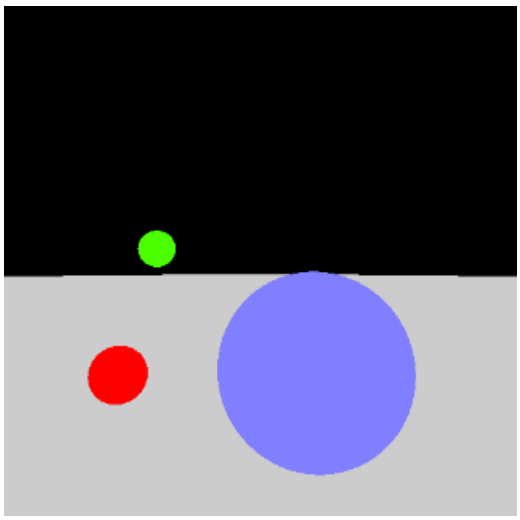


Figure 1: Intersection avec le plan

2.2 Cas de la sphère

Même principe que pour le plan, avec le système de départ :

$$(S) = \begin{cases} \|x - x_s\|^2 = r^2 \\ x = x_0 + tu \end{cases}$$

Après calculs on trouve :

$$(S) = \begin{cases} t^2 \|u\|^2 + 2t\langle u, x_0 - x_s\rangle + \|x_0 - x_s\|^2 - r^2 \\ x = x_0 + tu \end{cases}$$

On reconnaît une équation du second degré que l'on résout et qui nous donne deux solutions :

$$t = -\langle u, x_0 - x_s\rangle \pm \sqrt{\Delta}$$

avec

$$\Delta = \langle u, x_0 - x_s\rangle^2 - (\|x_0 - x_s\|^2 - r^2)$$

3 Lancé de rayons basiques

3.1 Recherche de l'intersection la plus proche

Question 12

On va avoir cinq cas distincts selon les valeurs de Δ :
- $\Delta < 0$ dans ce cas pas d'intersection avec la sphère.
- $\Delta = 0$ dans ce cas il y a une seule et unique intersection avec la sphère.
- $\Delta > 0$ c'est le cas le plus complexe que l'on sépare en trois sous-cas :

1er sous cas : $t_0 < 0$ et $t_1 < 0$ les deux racines sont négatives ce qui signifie que le rayon intersecte la sphère derrière son origine ce qui est physiquement impossible, on renverra donc false.

2ème sous-cas : $t_0 < 0$ et $t_1 > 0$ on a une racine positive et une négative ce qui signifie que l'origine du rayon se trouve à l'intérieur de la sphère, on choisira la racine positive.

3ème sous-cas : $t_0 < 0$ et $t_1 > 0$ les deux racines sont positives le rayon part de l'extérieur de la sphère et intersecte cette dernière en deux points, il faut alors déterminer le plus proche. Pour ce faire on prendra l'intersection pour laquelle la distance intersection-origine du rayon est la plus petite.

Le code la méthode compute_intersection qui s'occupe du calcul de l'intersection la plus proche se trouve en annexe 1. Après avoir implémenté cette méthode on obtient le résultat affiché figure 2, une partie de la sphère est recouverte par le plan.

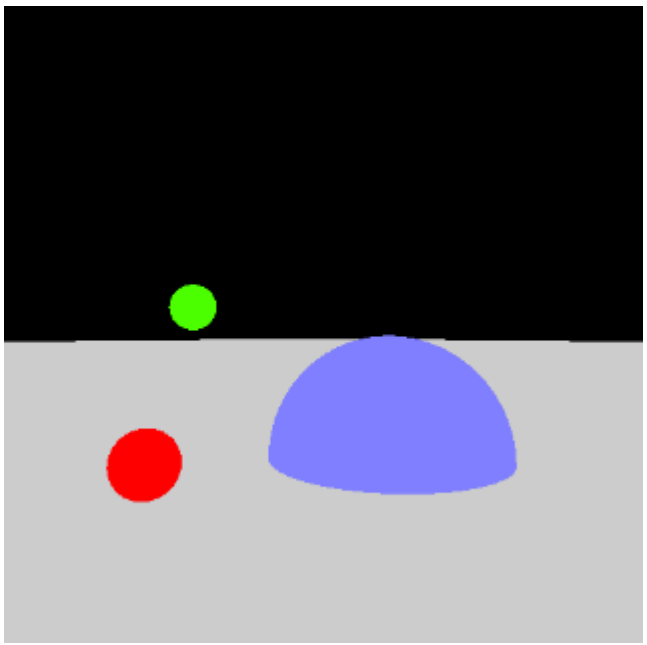


Figure 2: Intersection avec la sphère

3.2 Ombres et Illumination

La gestion des ombres se fait à l'aide de trois méthodes : *ray_trace* pour calculer les couleurs associés aux rayons dans la scène, *compute_illumination* pour la gestion de l'illumination ainsi que *is_in_shadow* une fonction retournant un booléen selon si le point est dans l'ombre ou non.

Voir le code en annexe 2.

Figure 3: Ombres

On rappelle le principe de l'illumination de Phong : c'est un modèle local donc le calcul de l'illumination se fait pour chaque pixel, il consiste à séparer la lumière en trois composantes: - La composante ambiante: K_a qui représente les lumières parasites provenant d'autres sources que celles considérées. - La composante diffuse: $Kd(\vec{L}.\vec{N})$ avec $\vec{L}$ le vecteur de lumière qui indique l'intensité de la lumière réfléchie en prenant une intensité constante quelle que soit sa direction. - La composante spéculaire: $Ks = K_s(\vec{R}.\vec{V})^\alpha$ avec α une constante liée au brillant du matériau $\vec{V}$ le vecteur pour la direction de vue de l'observateur et $\vec{R}$ le rayon réfléchi.
(K_a, K_d et K_s sont des constantes liées à chaque composante)

Le code est en annexe 3.

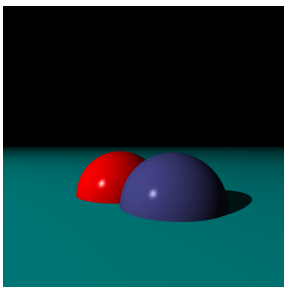


Figure 4: Illumination de Phong

4 Aller plus loin : Réflexion

On va implémenter la réflexion des rayons dans la classe *render_engine*, pour cela on utilise la formule suivante: $u' = \vec{u} - 2\langle \vec{u}, \vec{n} \rangle \vec{n}$

Toujours avec $\vec{u}$ le vecteur du rayon incident, $\vec{n}$ la normale au point d'intersection et $\vec{u'}$ le rayon réfléchi. Le principale problème est qu'il n'existe pas une seule

réflection. En effet, les rayons primaires envoyé depuis la caméra ne seront pas les seules à être réfléchis, les rayons incidents le seront également avec bien sûr un coefficient. La difficulté est de savoir gérer le nombre d'itérations de réflections et d'établir un compromis entre la vitesse et la fidélité du rendu. Dans notre cas, 5 itérations nous apporte un résultats plus que satisfaisant, comme montré sur la figure ci-dessous.

Pour l'affichage du code on se reportera à l'annexe 4.

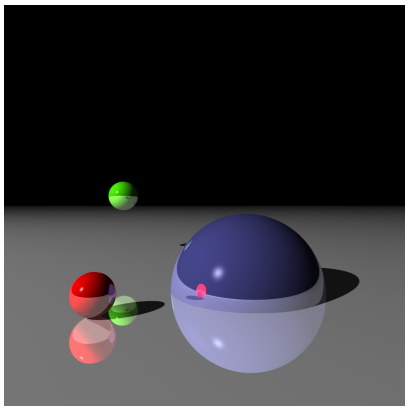
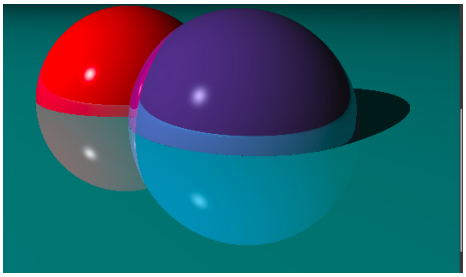


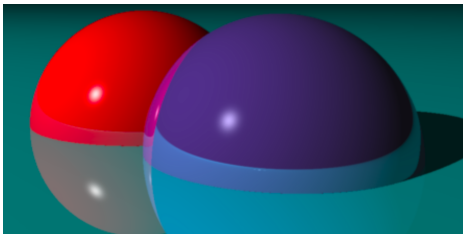
Figure 5: Résultat de la réflection

5 Anti aliasing

Dans la suite du TP, après avoir implémenté la réflection, on se propose de passer à l'optimisation du rendu : la netteté. L'anti-aliasing qui est connu de beaucoup de joueurs pour sa présence dans les paramètres graphiques des jeux permet moyennant des resources supplémentaires, d'améliorer la netteté de la scène. L'idée principale est de moyenner plusieurs rayons pour un seul pixel. Ainsi pour un pixel x_i, y_i, il est possible par différents paramètres , d'obtenir une nouvelle valeur en moyennant ses voisins. $(x_i, y_{i+1}), (x_{i+1}, y_i), (x_i, y_{i-1}), ((x_{i-1}, y_i))$.... sur le nombre totale de pixels



(a) Sans anti-aliasing



(b) Avec un anti-aliasing

Le rendu est nettement plus joli, la contrepartie étant le temps de rendu qui a quasiment doublé.

6 Réfraction

Après avoir implémenté une méthode d'anti-aliasing, il est question de pousser plus loin le travail sur l'optique de notre scène. En parallèle de la réflection et grâce aux lois de Snell-Descartes, nous savons que :

$$\eta_1 sin(i_1) = \eta_2 sin(i_2)$$

η représente l'indice de réfraction du milieu. Chaque milieu possède un indice propre, i_1 représente l'angle entre le rayon incident et la normale au point de contact, i_2 représente l'angle entre le nouveau rayon réfracté et l'inverse de la normale.

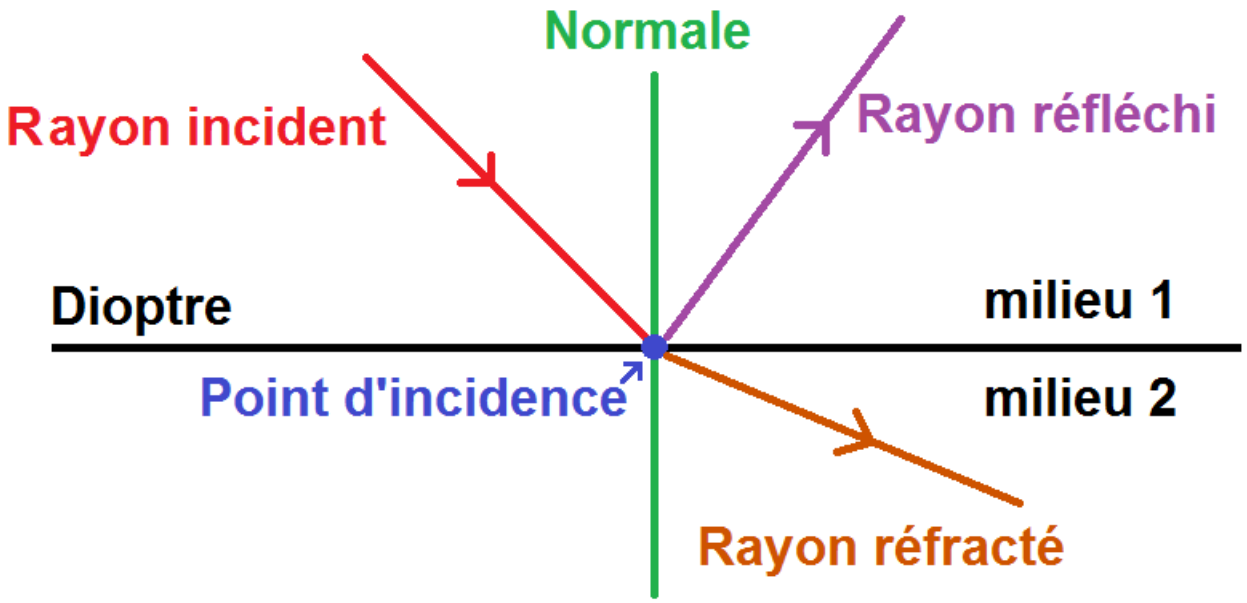


Figure 7: Schéma de la réfraction

Il y a cependant une condition supplémentaire, tout rayon ne peut être réfracté. Si les conditions ne le permettent pas, le rayon sera totalement réfléchi. Cette condition est de même implantée dans notre programme, le rayon peut être réfracté si :

$$\frac{\eta_1}{\eta_2} ||\vec{u} \wedge \vec{n}|| < 1$$

La direction du rayon est donnée par cette formule :

$$\vec{r} = \vec{u} + \alpha\vec{n} \text{ où } \alpha = \frac{\|\vec{u}\wedge\vec{n}\|}{tan(asin(\frac{n_1}{n_2}\|\vec{u}\wedge\vec{n}\|)} \; signe(\vec{u}.\vec{n}) \; -\vec{u}.\vec{n}$$

avec $\vec{u}$ la direction du rayon incident et $\vec{n}$ la normale au point d'intersection

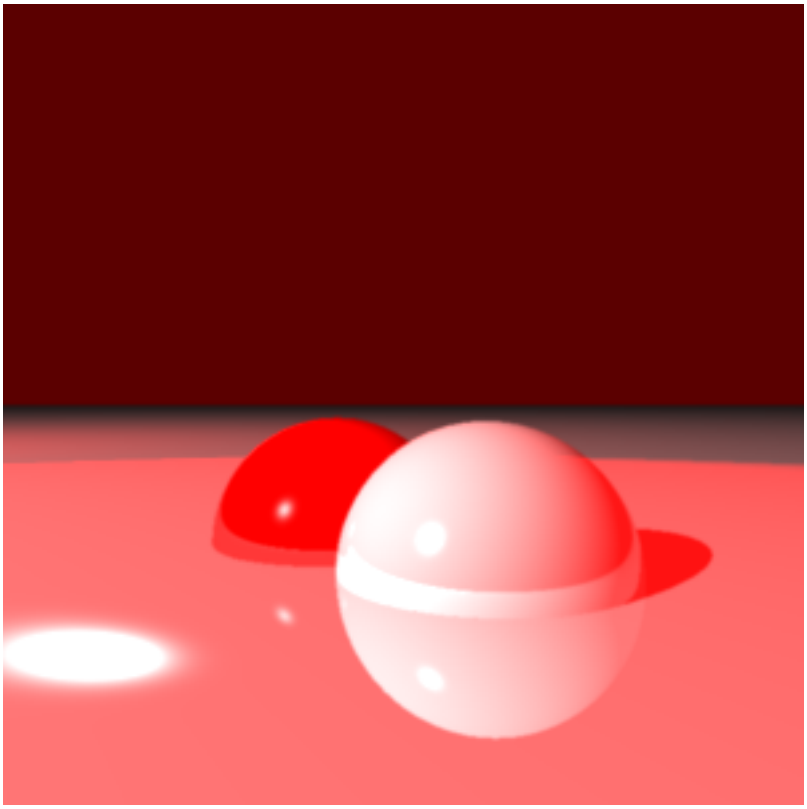


Figure 8: Résultat de la réfraction

Malgré un code très développé et plusieurs hypothèses solutionnant le problème, le rendu de la réfraction reste inachevé. Les rayons réfractés débordent dans toute la scène ou sont concentrés sur la gauche de l'image, ce qui donne une ambiance du Jugement Dernier.

7 Conclusion

Le ray-tracing s'avère être une bonne méthode de rendu avec un algorithme en apparence plutôt simple, cependant les choses se compliquent lorsqu'on a des surfaces plus complexes, en effet pour un plan, une sphère ou même un triangle il est aisé de déterminer le point d'intersection de façon analytique grâce aux équations mais pour des surfaces dont on ne connaît pas la fonction on sera obligés d'utiliser des méthodes d'approximation comme la méthode de Newton ou de descente de gradient.

Méthode	Avantages	Inconvénients
Maillage	Temps de création d'un modèle 3D réduit	Perte de qualité
Ray Tracing	Réalisme, Lissage	Temps de calculs longs

Un autre inconvénient du ray-tracing est qu'il ne prend pas en compte les caustiques, c'est-à-dire les rayons de lumière qui vont de la source à la composante diffuse en passant par la composante spéculaire, visuellement parlant un exemple pourrait être la lumière qui passe à travers une fenêtre renvoyée par un sol. D'autres aspects pour faire plus réel comme l'illumination globale ou la dispersion lumineuse sont aussi négligés.

8 Annexes

Annexe 1

Ombre

```
bool compute_intersection(ray const& r,
                          scene_parameter const& scene,
                          intersection_data& intersection,
                          int& index_intersected_primitive)
{
    int const N_primitive = scene.size_primitive();
    bool found_intersection = false;

    float intersection_var_relatif = 100000.0f;

    intersection_data intersection_var;

    for(int k = 0;k<N_primitive;k++)
    {
        primitive_basic const & primitive = scene.get_primitive(k);
        bool is_intersection = primitive.intersect(r,intersection_var);
        if(is_intersection)
        {    found_intersection = true;
            if (intersection_var.relative < intersection_var_relatif) {
                index_intersected_primitive = k;
                intersection = intersection_var;
                intersection_var_relatif = intersection_var.relative;
            }
        }
    }
    return found_intersection;
}
```

Annexe 2

Ombre

```
bool is_in_shadow(vec3 const& p,vec3 const& p_light, scene_parameter const& scene)
{
    ray rayon_shadow = ray(p,normalized(p_light-p));
    rayon_shadow.offset();

    //rayon_shadow.offset(1e10-5);
    int index_intersected_primitive;
    intersection_data intersection_var;
    if (compute_intersection(rayon_shadow, scene, intersection_var,index_intersected_primitive)) {
        if (norm(p_light-p) > intersection_var.relative) {
            return true;
        }
    }
```

Annexe 3

Illumination

```
color compute_illumination(material const& mat,intersection_data const& intersection,scene_parameter c
{    color c;

    vec3 const& p0 = intersection.position;

    int const N_light = scene.size_light();
    for(int k=0; k<N_light ; ++k)
    {
        light const& L = scene.get_light(k);
        bool const shadow = is_in_shadow(p0,L.p,scene);
        if(shadow==false)
            c +=  L.power*L.c*compute_shading(mat.shading(),mat.color_object(),p0,L.p,intersection.nor
           // mat.color_object();
        else
        {
            c += mat.color_object()/10.0f;
        }

    }

    return c;

}
```

Annexe 4

Réflexion

```
ray reflection(intersection.position, rayon.u()-2*dot
(rayon.u(),intersection.normal) * intersection.normal );
 reflection.offset();
intersection_data intersection_reflected;
int intersected_primitive_reflected = 0;  //current index of intersected reflected primitive

bool const is_intersection_reflected = compute_intersection(reflection,scene,intersection_reflected,
intersected_primitive_reflected);
sum_color += (mat.reflection()/(1+k))*compute_illumination(mat,intersection,scene);
rayon = reflection;
intersection = intersection_reflected;
intersected_primitive = intersected_primitive_reflected;
```

References

[1] Thibault Dupont. Cours ray tracing. *Cours CPE*, 2020.

[2] Nicolas Janey. `http://nicolas.janey.free.fr/RayTracing/lancer.htmAvantages`.

[3] Wikipedia contributors. Ray tracing (graphics) — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Ray_tracing_(graphics)&oldid=983699719`, 2020. [Online; accessed 16-October-2020].