Master 1 Informatique
ISICG

Introduction à la Synthèse d'Images Réalistes

Rapport TP+projet

Réalisé par :
TRABELSI Ayoub

Encadrant :
Monsieur MARIA Maxime

2022-2023

Table des matières

Table des figures

1 Introduction

Au cours de ce semestre, en nous appuyant sur la base de code proposé [1], nous avons exploré et appliqué diverses techniques de synthèse d'images réalistes.
Dans ce rapport, je présenterai les différentes méthodes mises en œuvre pour réaliser les travaux pratiques et les améliorations.

Vous trouverez ci-dessous les liens Git pour accéder au code :

- Git personnel :

- Git Unilim :

2 Lancer de rayons

2.1 Introduction

Le lancer de rayons fonctionne en traçant des rayons depuis la caméra vers les objets de la scène.
Lorsqu'un rayon rencontre un objet, il peut être réfléchi, réfracté ou absorbé, selon le matériau de l'objet.

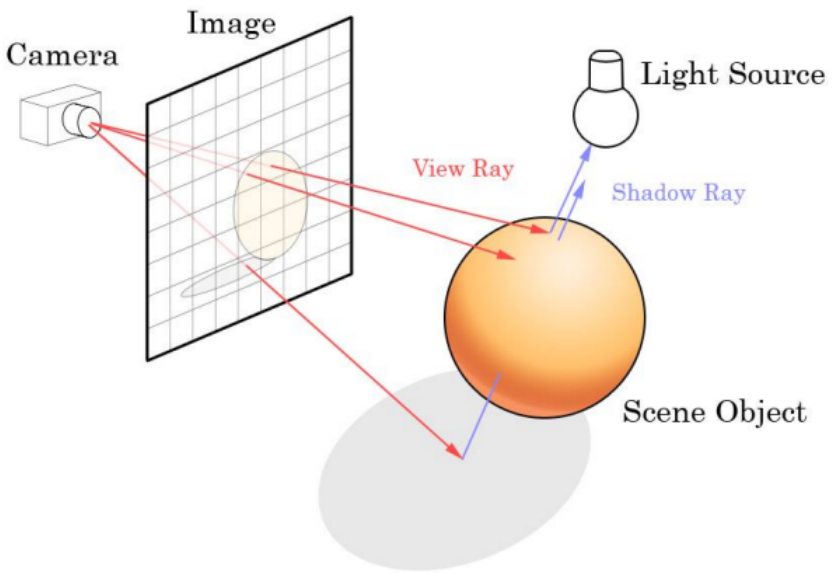


FIGURE 1 – Lancer de rayons [2]

2.2 Caméra, rayons et projection perspective

Dans ce moteur, c'est la caméra qui se charge de générer ces rayons. Pour le moment, nous ne traiterons qu'une caméra perspective (classe **PerspectiveCamera**)
- Pour calculer la direction des rayons primaires, il faut tout d'abord calculer la position et les dimensions de la fenêtre virtuelle (viewport) à travers laquelle ils vont passer.

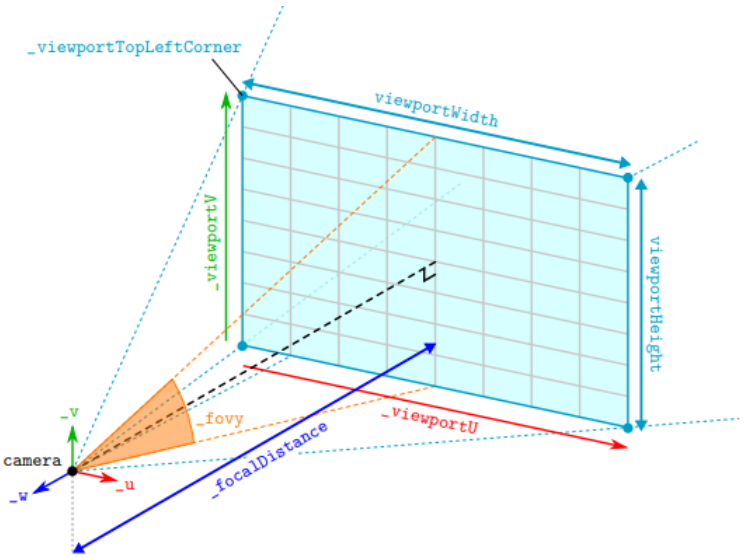


FIGURE 2 – Lancer de rayons [3]

En utilisant la figure ci-dessus on va implémenter la fonction **updateViewport** qui met à jour le viewport en fonction de la configuration de la caméra.

---

updateViewport

$viewportHeight \leftarrow 2 * tangente(fovy(radian)/2) * distanceFocale$
$viewportWidth \leftarrow viewportHeight * aspectRatio$
$viewportU \leftarrow u * viewportWidth$
$viewportV \leftarrow v * viewportHeight$
$viewportTopLeftCorner \leftarrow positionCamera - (w * distanceFocale) + (viewportV/2) - (viewportU/2)$

En utilisant le viewport, pour générer les rayons primaires la fonction **generateRay** prend en paramètres une position sur le viewport et retourne un rayon.

---

generateRay

$viewportHeight \leftarrow 2 * tangente(fovy(radian)/2) * distanceFocale$
$Direction \leftarrow viewportTopLeftCorner + (viewportU * positionX) - (viewportV * positionY) - positionCamera$
return $Rayon(positionCamera, normale(Direction))$

2.3 Objet

La géométrie des objets est décrite par une hiérarchie de classes dont la classe mère est abstraite (**BaseGeometry**). Pour ajouter des objets géométriques il faut créer une classe pour l'objet en héritant de la classe **BaseObject** Cette classe contient deux attributs : un nom et un matériau (classe **BaseMaterial**).
Cette classe doit implémenter la méthode **intersect**.

Après l'implémentation de la méthode **SphereGeometry : :intersect**(en passant le rayon en paramètre), la méthode **RayCastIntegrator : :Li**(pour prendre en compte l'angle entre la normale et la direction du rayon) et la méthode **SphereGeometry : :computeNormal**(qui retourne la normale de la sphère au point p). on obtient ce résultat.

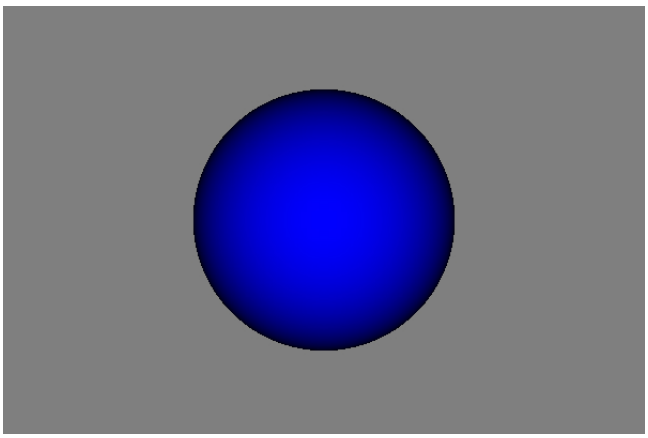


FIGURE 3 – TP1-3c

On va maintenant lancer plusieurs rayons par pixel et de faire la moyenne des couleurs obtenues (**Antialiasing**)pour atténuer le problème du crénelage.

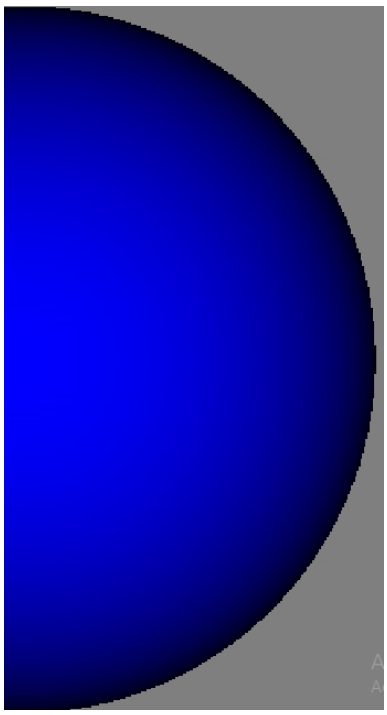


FIGURE 4 – Sans-Antialiasing

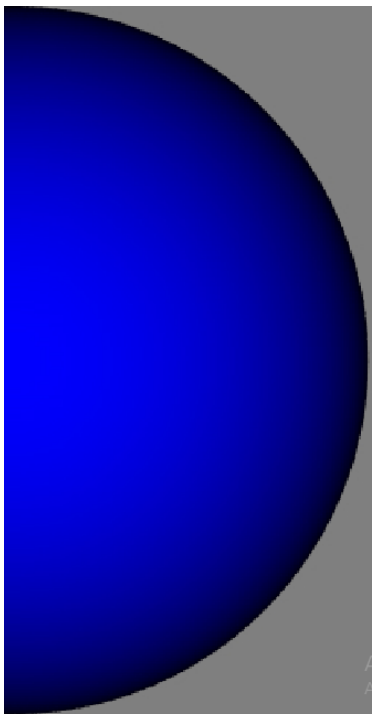


FIGURE 5 – Avec-Antialiasing(32 rayons)

3 Éclairage et calcul d'ombres portées

Dans cette partie, nous allons prendre en compte des sources lumineuses ponctuelles et calculer les ombres portées dans notre scène.

3.1 Plan

Maintenant que nous avons utilisé la sphère dans la partie précédente, nous allons ajouter un plan à notre scène. Pour ce faire, nous allons créer une nouvelle classe appelée **PlaneGeometry**, qui hérite de la classe **BaseGeometry** et décrit la géométrie d'un plan. Nous allons également créer une classe **Plane**, qui hérite de la classe **BaseObject** et décrit un objet plan.

Après l'ajout d'un **Plan** horizontal et d'un matériau **ColorMaterial** rouge, on a effectué l'association de ces deux dans **"Scene.cpp"**. On obtient cette figure.

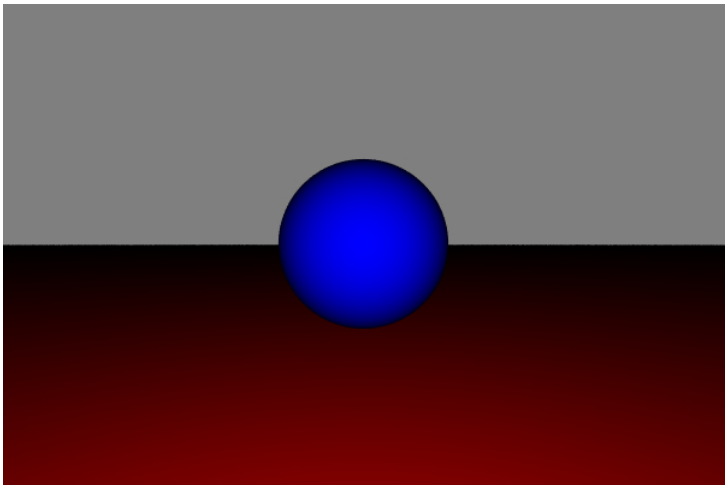


FIGURE 6 – Plan-Sphère

3.2 Sources lumineuses ponctuelles

On a la classe abstraite **BaseLight** sert de base à la hiérarchie de sources lumineuses.
Pour représenter une source lumineuse ponctuelle. On a ajouté la classe **PointLight**, héritant de **BaseLight**.

PointLight : :sample(point)

$Direction \leftarrow Normale(Position - point)$
$distance \leftarrow Distance(Position, point)$
$pdf \leftarrow 1$
$Radiance \leftarrow (Color * Power)/(distance * distance)$
return $Light(Direction, distance, Radiance, pdf)$

Jusque-là, notre intégrateur ne prend pas en compte les sources lumineuses.
Pour cela on va ajouter l'intégrateur **DirectLightingIntegrator**.
Dans l'intégrateur on doit calculer l'éclairage direct dans la méthode **directLighting**.

---

directLighting(Light,hitRecord,ray)

$lightSimple \leftarrow Light.sample(hitRecord.point)$
$Color = mtl- > getColor() * lightSample.radiance * cosTheta.$
return $Color$

La méthode **Li** doit calculer l'éclairage des objets en fonction des sources lumineuses de la scène.

Li(Scene,Ray,Tmin,Tmax)
  if $intersection(Ray, Scene)$ **then**
    $LightSamplelightsample \leftarrow Scene.getLights().sample(Hitrecord.point)$
    $RayRayShadow(Hitrecord.point, lightsample.direction)$
    $RayShadow.offset(Hitrecord.normal)$
    if $NOTScene.intersect(RayShadow, Tmin, Tmax, Hitrecord)$ **then**
     return $directLighting(Scene.getLights(), Hitrecord, Ray)$;
    end if
  else
      return $backgrounColor$
  end if

Après l'ajout d'une source lumineuse ponctuelle située en (1, 10, 1) de couleur blanche et de puissance 100 et lancer des rayons d'ombrage.

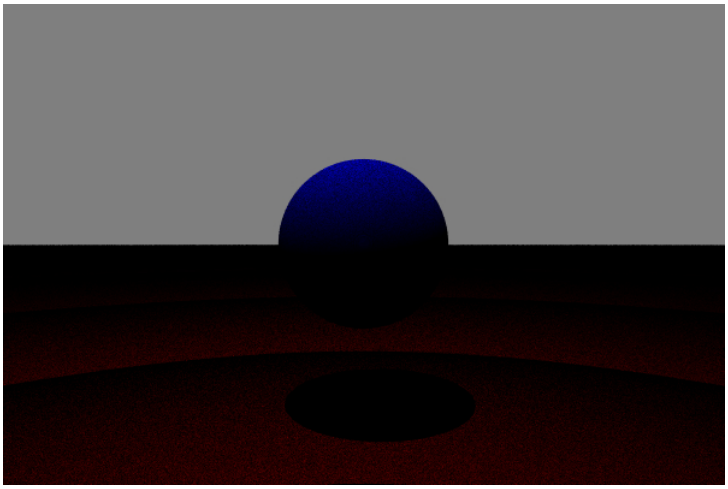


FIGURE 7 – Ombre(intersect)

En utilisant la mèthode **intersectAny** (retourne true si une intersection valide entre TMin et TMax est trouvée et false sinon)à la place de **intersect** (cherche l'intersection la plus proche entre le rayon et la scène)

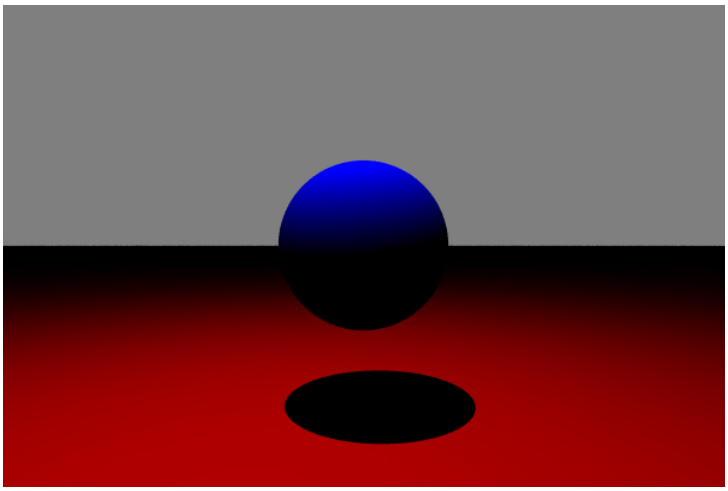


FIGURE 8 – Ombre(intersectAny)

4 Sources lumineuses surfaciques et ombres douces

Dans cette partie, nous allons prendre en compte des sources lumineuses surfaciques et calculer des ombres douces.
Pour diminuer les temps de calcul, nous allons paralléliser notre boucle de rendu en ajoutant #pragma omp parallel for au-dessus de la boucle traitant les lignes de pixels.

4.1 Sources lumineuses surfaciques (quad)

Dans la partie précédente notre scène était uniquement éclairée par une source lumineuse ponctuelle.
Dans le monde réel, les sources ponctuelles n'existent pas, les ombres dures produisent donc une image visuellement peu réaliste.
Donc nous allons ajouter une source lumineuse surfacique (quad)

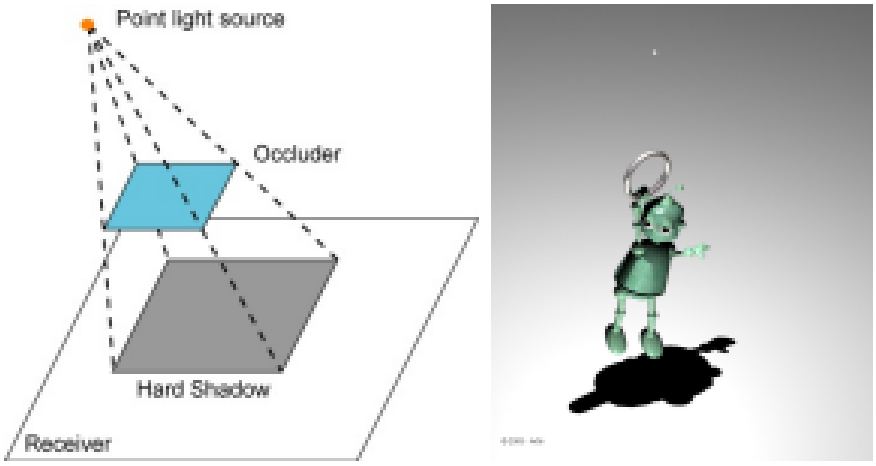


FIGURE 9 – Source luminuese ponctuelle[4]

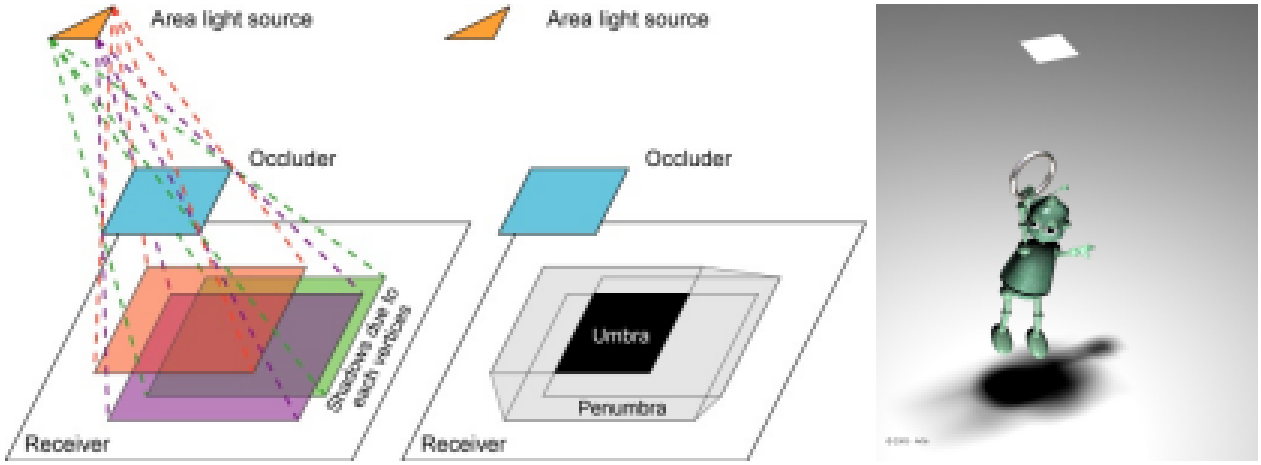


FIGURE 10 – Source luminuese surfacique[4]

Pour cela on va ajouter une classe **QuadLight** héritant de **BaseLight**.
la méthode **QuadLight : :sample** retourne un **LightSample**, à partir d'une position passée en paramètre.

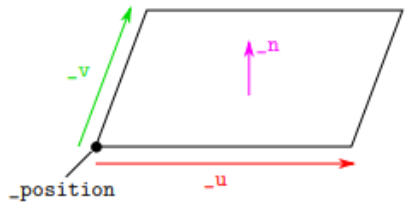


FIGURE 11 – Schéma d'une source lumineuse quad [4]

QuadLight : :sample(point)

$DeltaU \leftarrow \_u * randomFloat()$
$DeltaV \leftarrow \_v * randomFloat()$
$position \leftarrow \_position + DeltaU + DeltaV$
$Direction \leftarrow normale(position - point)$
$cos \leftarrow dot(normale, Direction)$
$distance \leftarrow distance(\_point, position)$
$facteur \leftarrow (distance * distance)/cos$
$pdf \leftarrow (1/area) * facteur$
$radiance \leftarrow (color * power)/pdf$
return $Lightsample(Direction, distance, Radiance, pdf)$

Après l'ajout d'une source lumineuse surfacique (quad) dans Scene.cpp située en (1, 10, 2) de vecteurs _u (-2, 0, 0) et _v(0, 0, 2) de couleur blanche et avec une puissance de 40 et en désactivant l'anti-aliasing, on obtient cette figure.

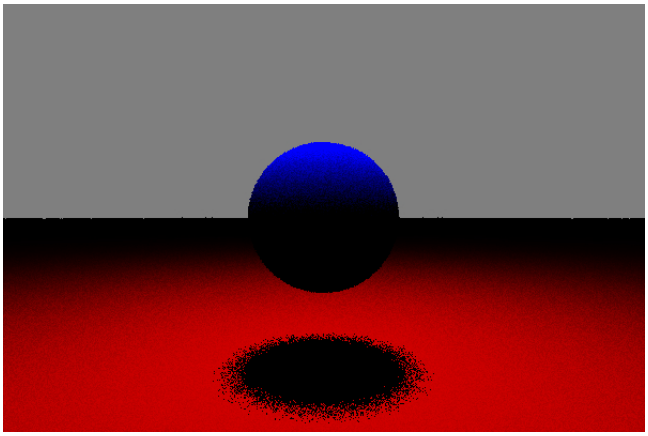


FIGURE 12 – Quad light(Sans anti-aliasing)

4.2 Moins de bruit

La génération aléatoire d'une direction pour tester l'éclairage d'un quad induit bien évidemment du bruit dans notre image. De plus, vu que nous lançons qu'un seul rayon, il n'y a pas d'ombres douces.
Pour pallier ce problème, il suffit de lancer plusieurs rayons d'ombrage...
Nous allons modifier la méthode **DirectLightingIntegrator : :Li** pour lancer **_nbLightSamples** rayons d'ombrage en cas de source surfacique (_nbLightSamples étant un attribut de notre intégrateur).
Pour chaque point observé, l'éclairage correspondra à la moyenne des **_nbLightSamples** contributions lumineuses.

Li(Scene,Ray,Tmin,Tmax)

if $intersection(Ray, Scene)$ **then**
  for lum **to** $liste\_lumières\_scène$ **do**
    if $Scene.getLights().at(i).getIsSurface()$ **then**
      for $j = 0$ **to** $\_nbLightSamples$ **do**
        $LightSamplelightsample \leftarrow Scene.getLights().sample(Hitrecord.point)$
        $RayRayShadow(Hitrecord.point, lightsample.direction)$
        $RayShadow.offset(Hitrecord.normal)$
        if $NOTScene.intersect(RayShadow, Tmin, Tmax, Hitrecord)$ **then**
          $Color \leftarrow Color + directLighting(Scene.getLights(), Hitrecord, Ray);$
        end if
      end for
      $color \leftarrow color/\_nbLightSamples$
    else
      $LightSamplelightsample \leftarrow Scene.getLights().sample(Hitrecord.point)$
      $RayRayShadow(Hitrecord.point, lightsample.direction)$
      $RayShadow.offset(Hitrecord.normal)$
      if $NOTScene.intersect(RayShadow, Tmin, Tmax, Hitrecord)$ **then**
        $Color \leftarrow Color + directLighting(Scene.getLights(), Hitrecord, Ray);$
      end if

return $Color$

  return $backgrounColor$
    end if

Aprés la Réactivation de l'anti-aliasing. on obtient ces résultats.

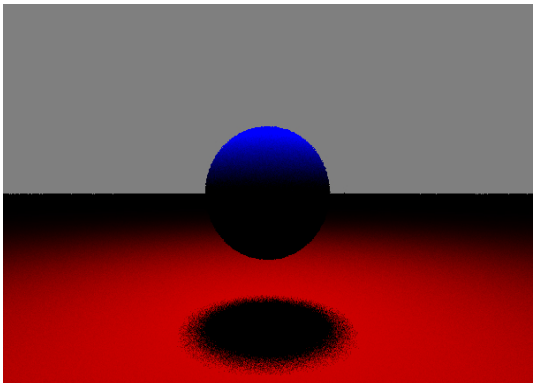


FIGURE 13 – nbLightSamples=4

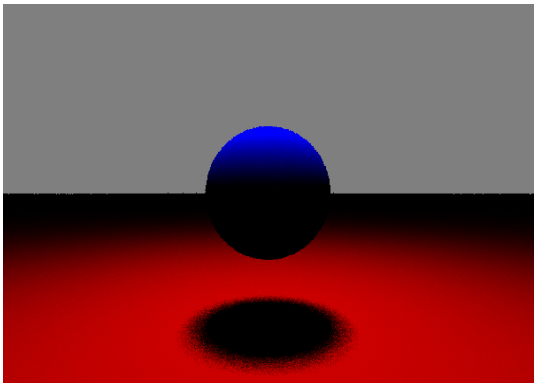


FIGURE 14 – nbLightSamples=8

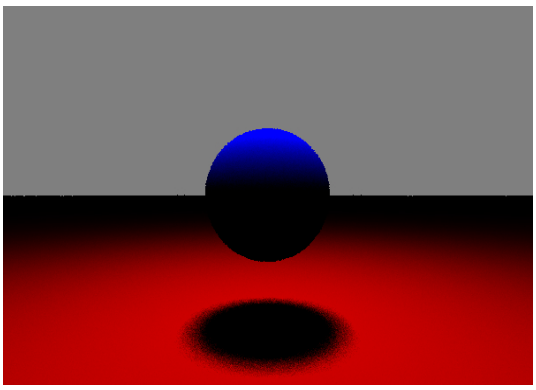


FIGURE 15 – nbLightSamples=16

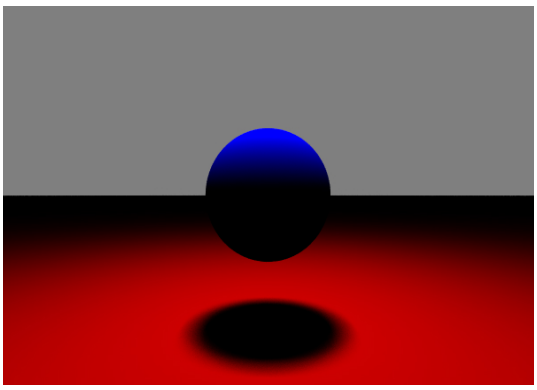


FIGURE 16 – nbLightSamples=32

5 Triangle, maillage et BVH

Jusqu'à maintenant, nous n'avions qu'une sphère et un plan comme primitives géométriques. Généralement en informatique graphique, les objets complexes sont représentés par un ensemble de triangles organisés dans un maillage. Dans ce TP, nous allons donc faire en sorte de pouvoir gérer ces maillages triangulaires.

5.1 Premiers maillages

On déclare la fonction **TriangleMeshGeometry : :intersect** proposée par **Möller et Trumbore**.
En utilisant la scène donnée en **Annexe A** dans le TP4 [5] et en mettant la caméra à (0, 2, 6).
On obtient cette figure.

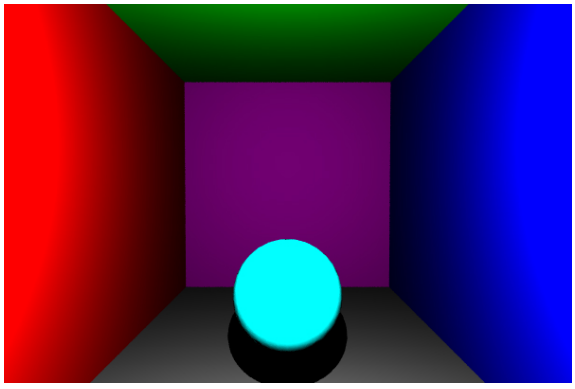


FIGURE 17 – uvsphere

5.2 Boîtes englobantes

Le calcul de l'image est trop long en l'état. En effet, pour chaque pixel, nous testons l'ensemble des primitives géométriques linéairement afin de trouver l'intersection la plus proche. Dans cette partie, nous allons utiliser un volume englobant autour de notre maillage dans le but de diminuer la complexité globale du calcul de l'image. Le principe est simple : avant de chercher une intersection avec les triangles composant le maillage, nous allons commencer par tester s'il y a une intersection avec le volume englobant du maillage. Si le rayon ne touche pas ce volume, alors il ne touche aucun triangle du maillage.
On modifie la méthode **MeshTriangle : :addVertex** pour modifier l'AABB en fonction du sommet ajouté avec la **méthode ABBB : :extend**.
Puis on implémente **AABB : :intersect** et **MeshTriangle : :intersect** (pour tester l'intersection avec l'**ABBB : :extend** avant de tester l'ensemble des triangles).

— La fonction **ABBB : :extend** est utilisé pour construire une boîte englobante qui entoure tous les points ou boîtes englobantes pertinents.

— La fonction **MeshTriangle : :intersect** ajoute un sommet à une collection de sommets et met à jour la boîte englobante pour inclure ce nouveau sommet, ce qui permet de maintenir la boîte englobante à jour au fur et à mesure de l'ajout des sommets.

— La fonction **AABB : :intersect** teste l'intersection entre un rayon et une boîte englobante.

— **MeshTriangle : :intersect**teste l'intersection entre un rayon et un triangle de maillage.

Maintenant le rendu est plus rapide.

6 Matériaux et BRDFs

Dans cette partie, nous allons ajouter des BRDF au moteur pour définir de nouveaux matériaux pour nos objets.

$$L_o = L_e + \int_\Omega L_i \cdot f_r \cdot \cos\theta \cdot d\omega$$

FIGURE 18 – Equation du rendu[6]

la **BRDF**, c'est la fonction **fr** dans l'équation du rendu.

6.1 Matériau parfaitement diffus : modèle de Lambert

Le modèle de Lambert représente des surfaces purement diffuses, pour lesquelles la lumière est réfléchie de manière uniforme dans toutes les directions.

$$f_r = \frac{k_d}{\pi}$$

FIGURE 19 – Lambert[6]

La BRDF associée est plus que simple à évaluer puisqu'il s'agit d'une constante.
Elle est donc indépendante des directions d'observation et d'incidence.
Où kd est le coefficient de réflectance diffus, représenté ici par un triplet RGB.
En Appliquant ce nouveau matériau aux deux objets.

On obtient cette figure.

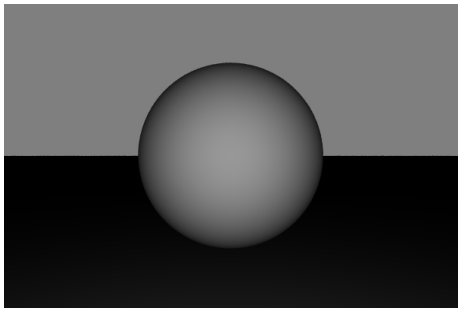


FIGURE 20 – BRDF Lambert

6.2 Matériau diffus et rugueux : modèle d'Oren-Nayar

Le modèle d'**Oren-Nayar** est utilisé pour représenter des matériaux mats en prenant en compte leur rugosité. Ce modèle suppose que la surface est composée de micro-facettes en forme de "V". La rugosité de la surface est caractérisée par un paramètre unique appelé la pente des facettes.

$$f_r = \frac{k_d}{\pi} \cdot (A + (B \cdot max(0, \cos(\phi_i - \phi_o)) \cdot \sin\alpha \cdot \tan\beta)$$

FIGURE 21 – BRDF Oren-Nayar[6]

Avec :

$$A = 1 - 0.5\frac{\sigma^2}{\sigma^2 + 0.33} \;\; ; \;\; B = 0.45\frac{\sigma^2}{\sigma^2 + 0.09} \;\; ; \;\; \alpha = max(\theta_i, \theta_o) \;\; ; \;\; \beta = min(\theta_i, \theta_o)$$

On implémente la fonction **evaluate** qui, retourne le résultat de la BRDF, à partir de la normale à la surface et des directions d'observation et d'incidence.
En Appliquant ce nouveau matériau aux deux objets, On obtient ces résultats en variant la rugosité.

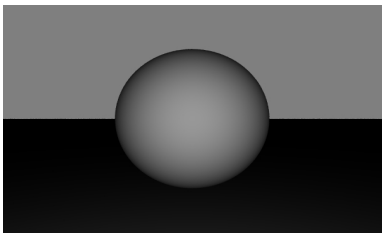


FIGURE 22 – Oren-Naya(rugosité=0)

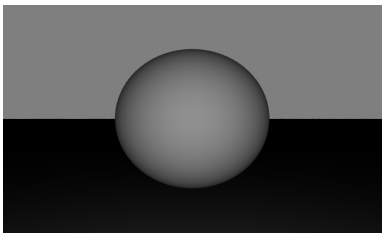


FIGURE 23 – Oren-Naya(rugosité=0.2)

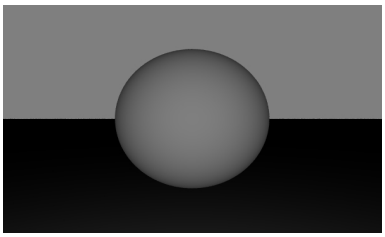


FIGURE 24 – Oren-Naya(rugosité=0.4)

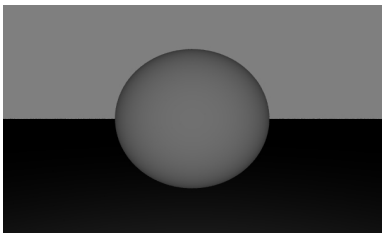


FIGURE 25 – Oren-Naya(rugosité=0.6)

6.3 Matériau « plastique » : modèle de Phong

Le modèle de **Phong** propose de représenter le reflet spéculaire d'une surface comme un cône aligné avec la direction du miroir par rapport à la direction d'incidence de la lumière. Ce modèle est entièrement empirique et n'est pas physiquement réaliste.

$$f_r = \frac{k_d}{\pi} + \frac{k_s}{\cos\theta_i} \cdot \cos\alpha^s$$

FIGURE 26 – BRDF Phong[6]

On implémente la fonction **evaluate** qui, qui n'évaluera que la partie spéculaire de la BRDF.
On ajoute une classe **PlasticMaterial** utilisant deux BRDF, **LambertBRDF** et **PhongBRDF**, calculant respectivement les parties diffuse et spéculaire de la BRDF.
En Appliquant ce nouveau matériau aux deux objets, On obtient ces résultats qui montrent une sphère grise (70% diffuse et 30% spéculaire) pour plusieurs valeurs de s.

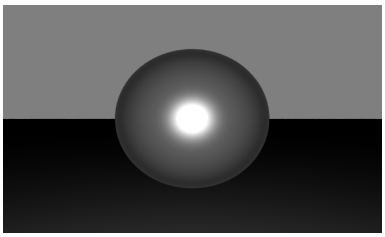


FIGURE 27 – Phong(s=8)

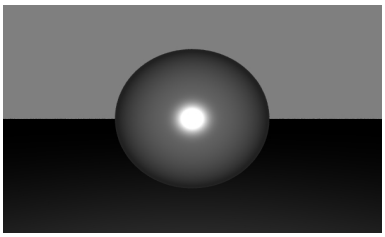


FIGURE 28 – Phong(s=16)

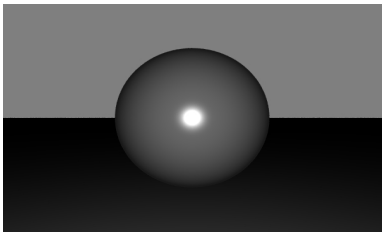


FIGURE 29 – Phong(s=32)

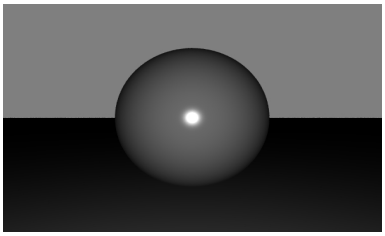


FIGURE 30 – Phong(s=64)

6.4 Matériau physiquement réaliste : modèle de Cook-Torrance

Nous allons maintenant implémenter une BRDF à micro-facettes en utilisant le modèle de **CookTorrance**

$$f_{rs} = \frac{DFG}{4(\omega_o \cdot n)(\omega_i \cdot n)}$$

FIGURE 31 – CookTorrance[6]

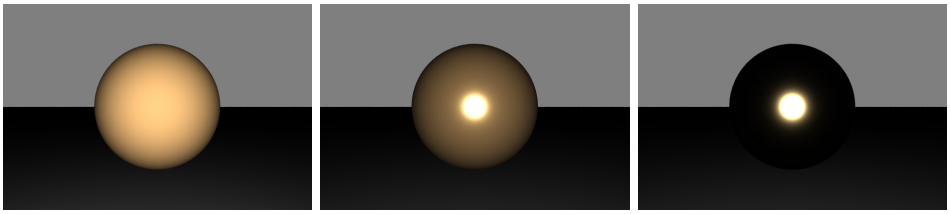
Avec :

$$D(h) = \frac{\alpha^2}{\pi((n \cdot h)^2(\alpha^2 - 1) + 1)^2}$$

$$G(\omega_i, \omega_o, n) = G_1(n \cdot \omega_o)G_1(n \cdot \omega_i)$$
$$G_1(x) = \frac{x}{x(1-k)+k}$$
$$k = \frac{(\sigma+1)^2}{8}$$

$$F(\omega_o, h, F_0) = F_0 + (1 - F_0)(1 - (h \cdot \omega_o))^5$$

On implémente la fonction **evaluate** qui, retourne le résultat de la BRDF.
En Appliquant un matériau doré F0 = (1, 0.85, 0.57) et une rugosité de 0.3 On obtient ces résultats en variant la Metalness.



(a) (b) (c)

FIGURE 32 – Cook-Torrance(Metalness=0/0.5/1)

7 Réflexions, réfractions

Danc cette partie, nous allons ajouter les phénomènes lumineux de réflexion et de réfraction.
Nous allons rester dans le contexte simple de la méthode de Whitted qui a introduit le lancer de rayons en tant que processus récursif.

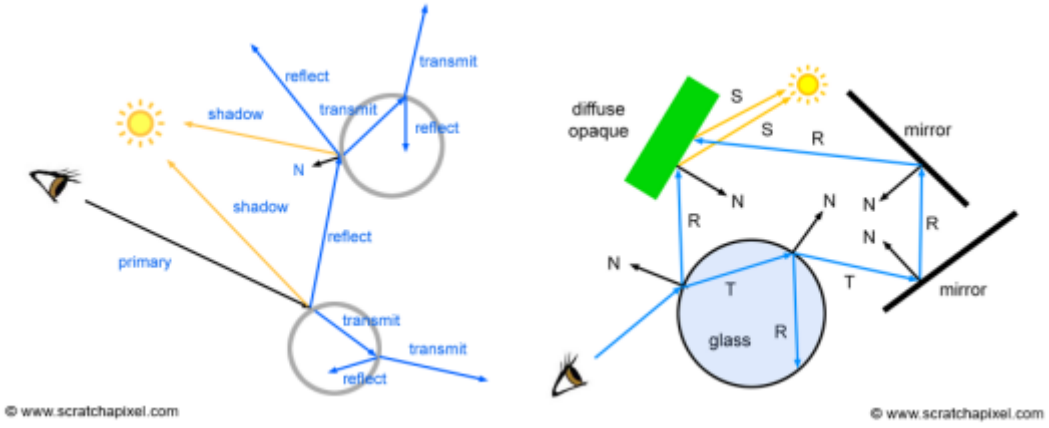


FIGURE 33 – Lancer de rayons récursif[16]

Maintenant il faut créer un nouvel intégrateur **WhittedIntegrator** à partir de **DirectLightingIntegrator** permettant de gérer la récursivité du lancer de rayons.

```
1  trace(scene, rayon)
2      if <intersection rayon/scene>
3          if <materiau mirroir>
4              rayonReflexion = <calculer reflexion>
5              trace(scene, rayonReflexion)
6          else if <materiau transparent>
7              rayonReflexion = <calculer reflexion>
8              trace(scene, rayonReflexion)
9              rayonRefraction = <calculer refraction>
10             trace(scene, rayonRefraction)
11         else
12             <calculer eclairage direct>
```

FIGURE 34 – La fonction récursive[16]

Pour le calcul d'éclairage direct, j'ai utilisé la fonction **DirectLightingIntegrator : :Li** puisque c'est déjà définit dans la classe **DirectLightingIntegrator**.
Pour représenter les objets parfaitement spéculaires, J'ai ajouté un nouveau matériau **MirrorMaterial**, avec la méthode **isMirror** pour identifier dans l'intégrateur si le matériau est un miroir ou non.
Pour représenter les objets transparents, J'ai ajouté un nouveau matériau **TransparentMaterial** avec la méthode **isTransparent** pour identifier dans l'intégrateur si le matériau est transparent ou non.

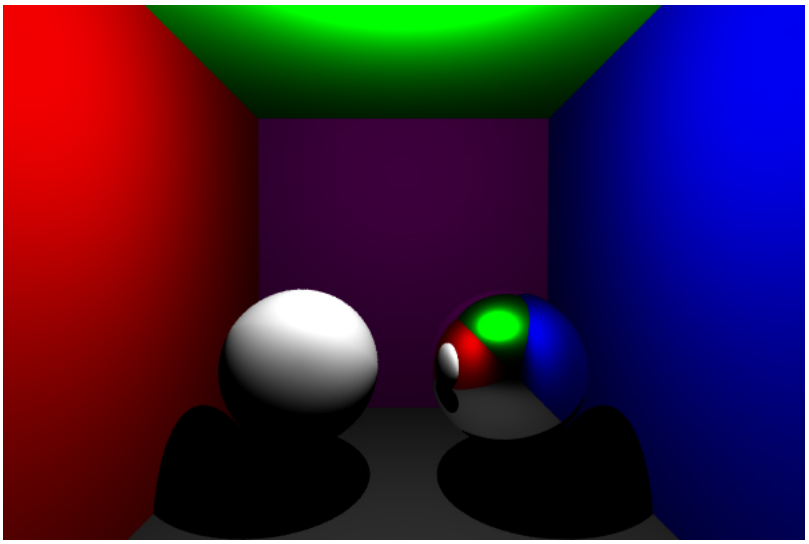


FIGURE 35 – Sphère1 miroir

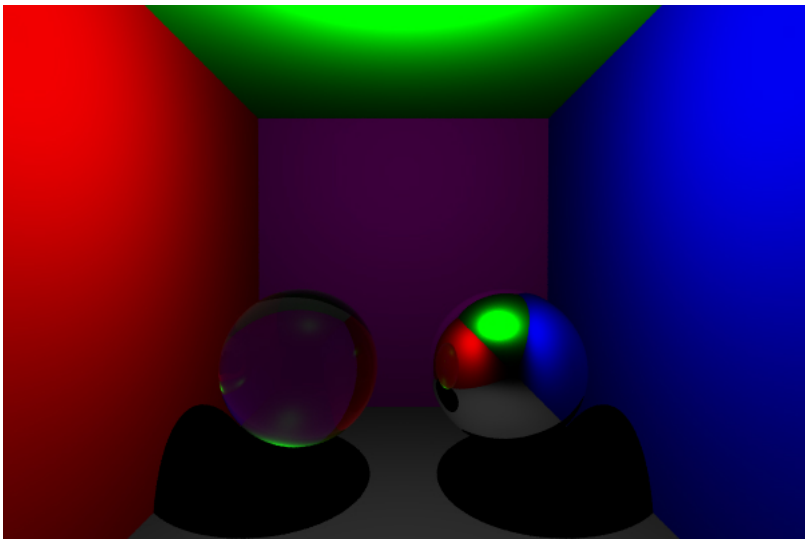


FIGURE 36 – Sphère2 transparent

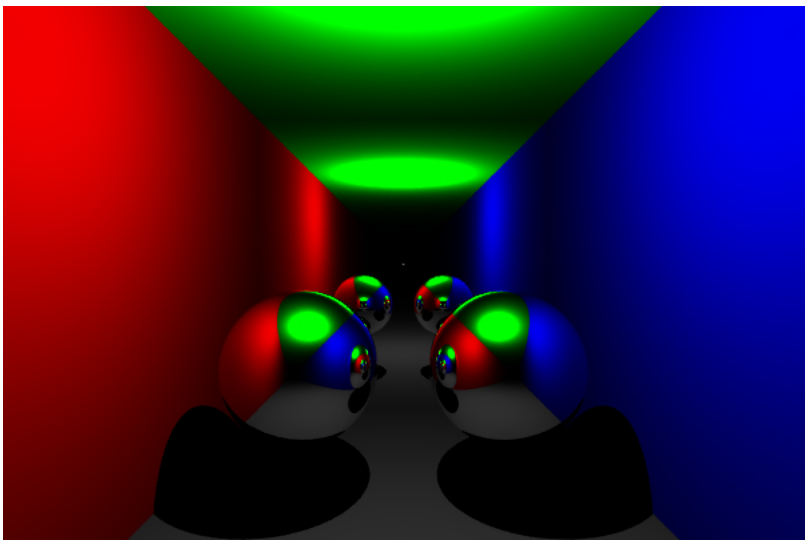


FIGURE 37 – Miroir(Sphère1,Sphère2,Plan)

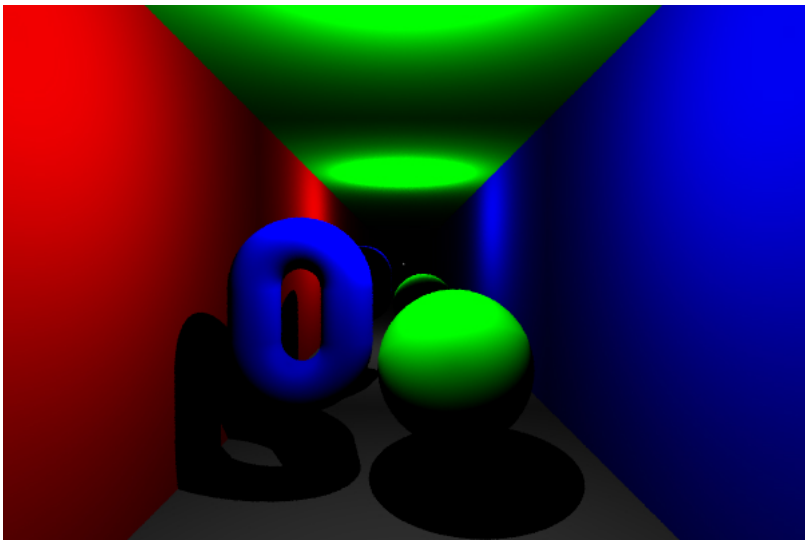


FIGURE 38 – Plan Miroir

8 Surface implicit et sphere tracing

Dans cette partie, nous allons faire le rendu de surfaces implicites via un algorithme de ray-marching et plus particulièrement celui du sphere-tracing.

Au début, il faut implémenter les méthodes **intersect** et **intersectAny** de la classe **ImplicitSurface**, pour déterminer l'intersection entre un rayon et la surface(ça marche avec toutes les surfaces en utilisant la fonction _sdf qui va être different dans chaque surface implicite).

```
bool ImplicitSurface::intersect( const Ray & p_ray,
                                 const float p_tMin,
                                 const float p_tMax,
                                 HitRecord & p_hitRecord ) const
{
    for ( float i = p_tMin; i < p_tMax;) //profondeur minimale ---> profondeur maximale
    {
        Vec3f point_i   = p_ray.pointAtT( i );//point sur le rayon à la profondeur i
        float distance = _sdf( point_i );     // distance entre le point et la surface implicite
        i += distance;
        if ( distance < _minDistance ) //une intersection avec la surface implicite à point_i
        {   // Intersection found, fill p_hitRecord.
            p_hitRecord._point  = p_ray.pointAtT( i );
            p_hitRecord._normal = _evaluateNormal( p_hitRecord._point );
            p_hitRecord.faceNormal( p_ray.getDirection() );
            p_hitRecord._distance = i;
            p_hitRecord._object   = this;
            return true;
        }
    }
    return false;
}
```

FIGURE 39 – intersect

```
bool ImplicitSurface::intersectAny( const Ray & p_ray, const float p_tMin, const float p_tMax ) const
{
    /// TODO
    for ( float i = p_tMin; i < p_tMax; ) // profondeur minimale ---> profondeur maximale
    {
        Vec3f point_i  = p_ray.pointAtT( i ); // point sur le rayon à la profondeur i
        float distance = _sdf( point_i );     // distance entre le point et la surface implicite
        i += distance;
        if ( distance < _minDistance ) // une intersection avec la surface implicite à point_i
        { // Intersection found, fill p_hitRecord.
            return true;
        }
    }
    return false;
}
```

FIGURE 40 – intersectAny

Pour avoir un shading correct, il faut évaluer la normale de la surface au point d'intersection en utilisant la fonction **_evaluateNormal**.

```
virtual Vec3f _evaluateNormal( const Vec3f & p_point ) const
{/// TODO
//http://rodolphe-vaillant.fr/entry/87/normal-to-an-implicit-surface
    float e = 0.00001;
    return glm::normalize( Vec3f( 1, -1, -1 ) * _sdf( p_point + Vec3f( e, -e, -e ) )
                           + Vec3f( -1, -1, 1 ) * _sdf( p_point + Vec3f( -e, -e, e ) )
                           + Vec3f( -1, 1, -1 ) * _sdf( p_point + Vec3f( -e, e, -e ) )
                           + Vec3f( 1, 1, 1 ) * _sdf( p_point + Vec3f( e, e, e ) ) );
}
```

FIGURE 41 – evaluateNormal

Pour ajouter des surfaces implicites à notre scène, il faut créer des classes dérivant de la classe **Implicit-Surface**, avec l'implémentation de la méthode **_sdf** pour calculer la distance de la surface par rapport à un point.

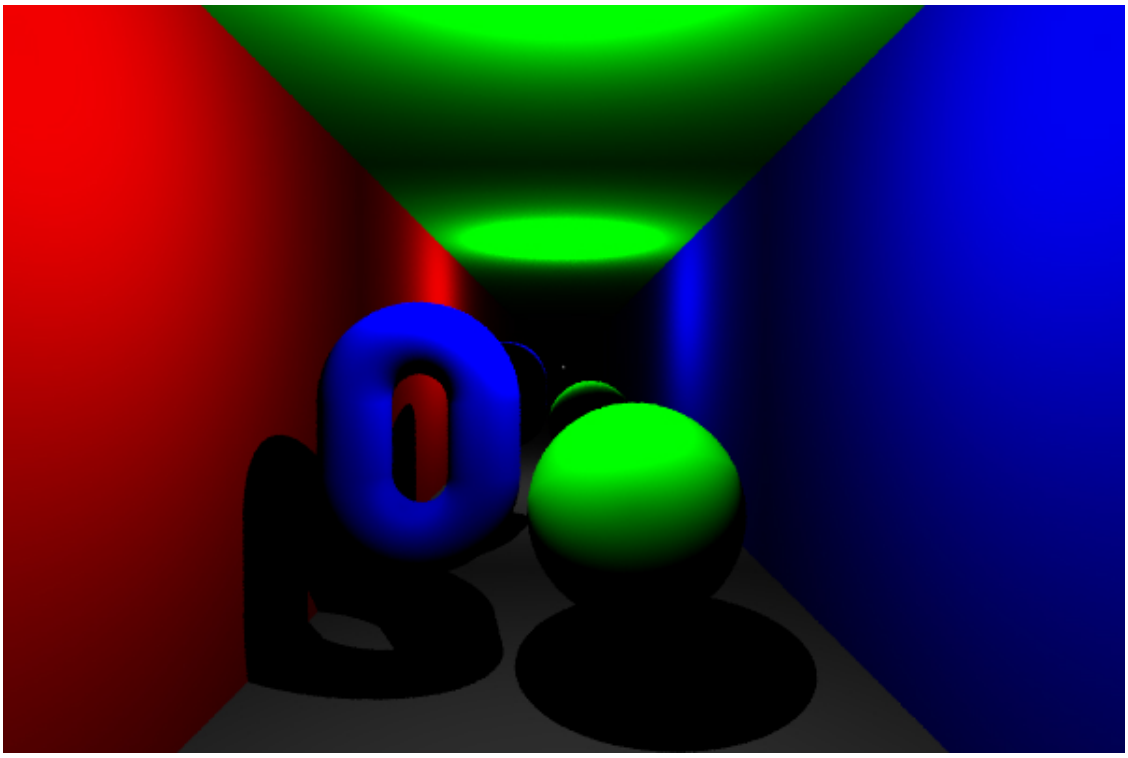


FIGURE 42 – Surfaces implicites 1

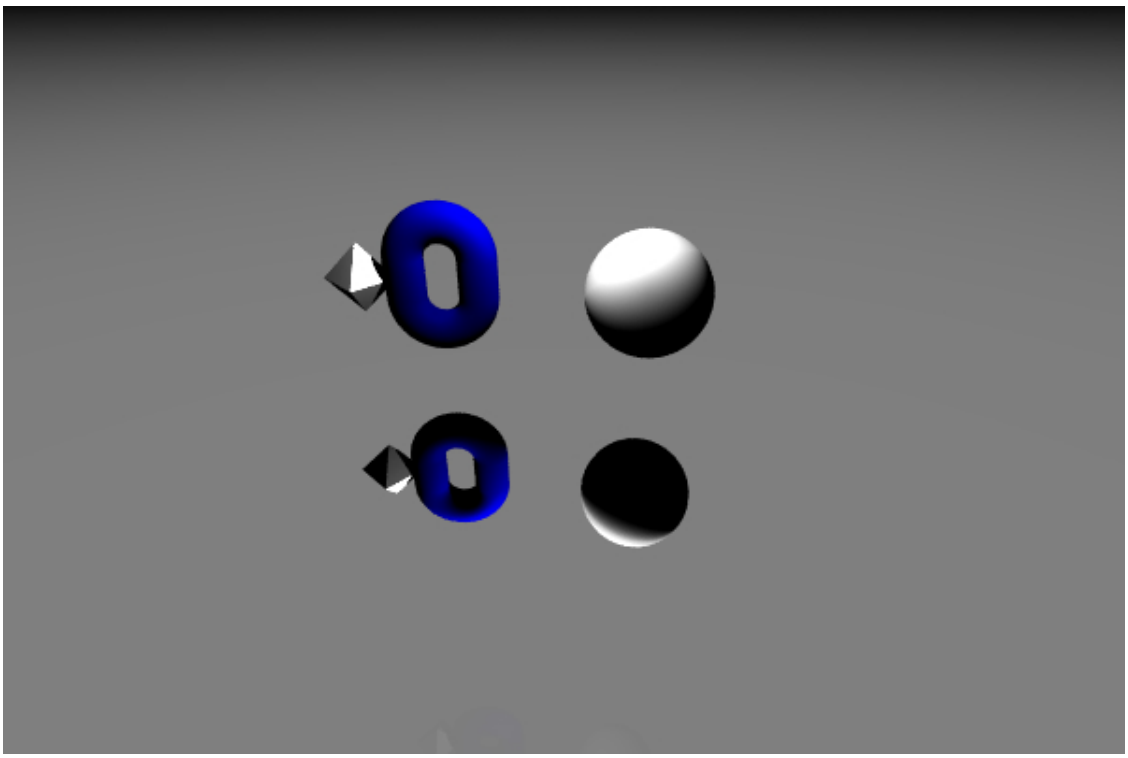


FIGURE 43 – Surfaces implicites 2

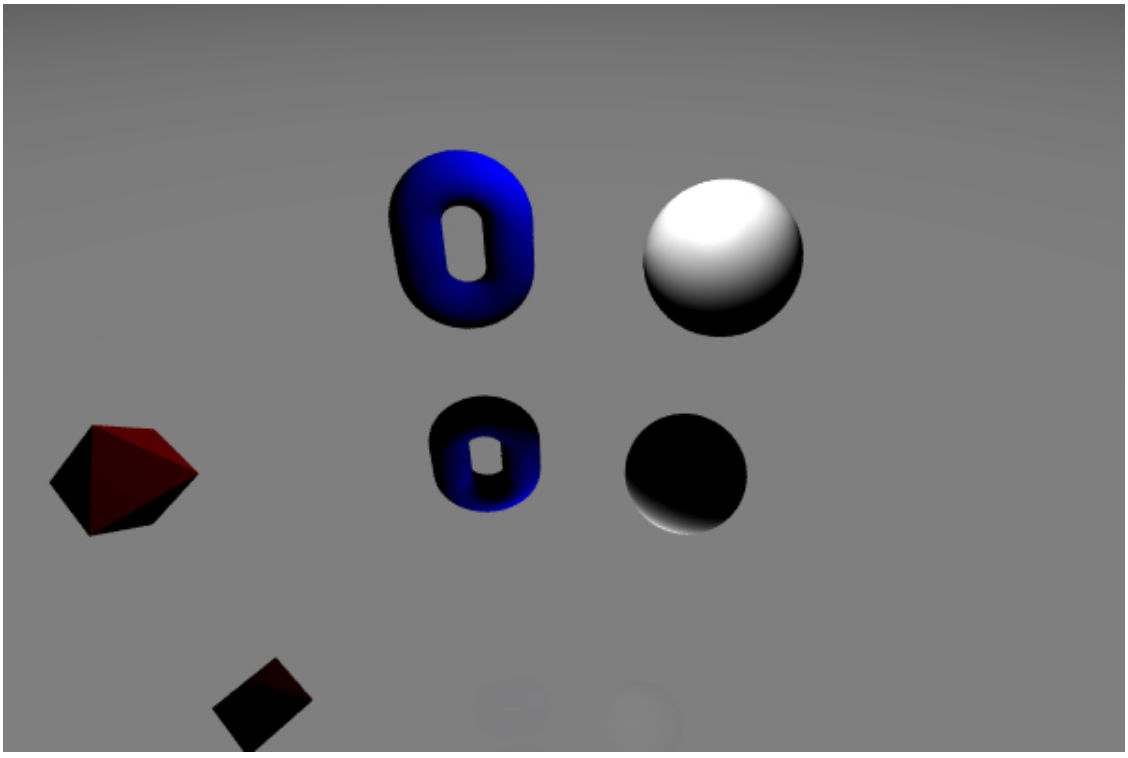


FIGURE 44 – Surfaces implicites 3

9 Spot Light

Dans cette partie, j'ai essayé d'ajouter une source lumineuse (SpotLight), mais vu la limite du temps, je ne suis pas arrivé à la finaliser.

point lights

- light is emitted equally from a point $\mathbf{S}$ in all directions
- simulate local lighting, different at each surface point $\mathbf{P}$
- light direction: $\mathbf{l} = (\mathbf{S} - \mathbf{P})/|\mathbf{S} - \mathbf{P}|$
- light color: $L = k_l/|\mathbf{S} - \mathbf{P}|^2$

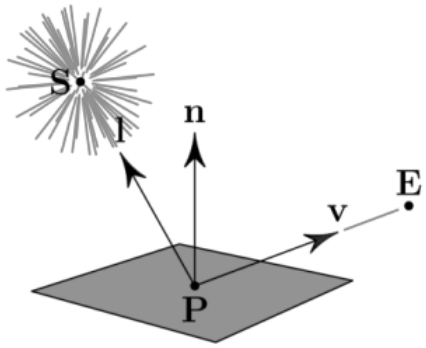


FIGURE 45 – Point Light[24]

spot lights

- same as points lights, but only emits in a cone around $\mathbf{d}$
- simulate theatrical lights
- cone falloff model arbitrary
- light direction: $\mathbf{l} = (\mathbf{S} - \mathbf{P})/|\mathbf{S} - \mathbf{P}|$
- light color: $L = k_l \cdot attenutation/|\mathbf{S} - \mathbf{P}|^2$

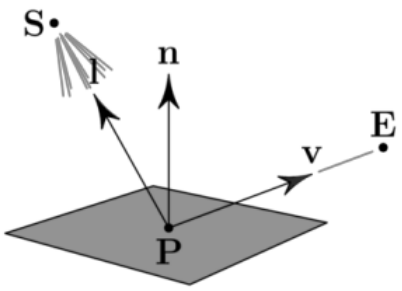


FIGURE 46 – Spot Light[24]

10 Conclusion

Dans ce module, j'ai acquis de nombreuses connaissances et compétences en Synthèse d'Images Réalistes. Malgré les difficultés que j'ai rencontrées tout au long du semestre, je suis content des résultats que j'ai obtenus. Malheureusement avec les contraintes de temps, je n'ai pas pu implémenter des améliorations à mon code, mais j'ai essayé d'ajouter une source lumineuse (SpotLight).
Je tiens à exprimer ma profonde gratitude envers mes professeurs du module ISIR pour leur soutien inestimable tout au long de ce semestre. Leurs conseils ont été d'une grande aide dans la réalisation de ces TPs.

11 Bibliographie

Ci-dessous une collection de références vers les différentes ressources que j'ai utilisé lors de la réalisation de ce projet.

Références

[1] http://git.unilim.fr/mariap02/rt_isicg_2023.git

[2] https://community-sciences.unilim.fr/pluginfile.php/75648/mod_resource/content/1/1_Creating_a_Raytracer.pdf

[3] https://communitysciences.unilim.fr/pluginfile.php/121474/mod_resource/content/6/22_23_ISIR_TP1_Raycasting.pdf

[4] https://community-sciences.unilim.fr/pluginfile.php/124193/mod_resource/content/9/22_23_ISIR_TP3_Sources_lumineuses_surfaciques_et_ombres_douces.pdf

[5] https://community-sciences.unilim.fr/pluginfile.php/127115/mod_resource/content/6/22_23_ISIR_TP4_Triangle_maillage_et_BVH.pdf

[6] https://community-sciences.unilim.fr/pluginfile.php/193325/mod_resource/content/2/22_23_ISIR_TP5_Materiaux_et_BRDF.pdf

[7] https://mathinsight.org/spherical_coordinates

[8] https://en.wikipedia.org/wiki/Phong_reflection_model

[9] https://en.wikipedia.org/wiki/Lambertian_reflectance

[10] http://www.codinglabs.net/article_physically_based_rendering_cook_torrance.aspx

[11] https://raytracing.github.io/

[12] https://raytracing.github.io/books/RayTracingInOneWeekend.html

[13] https://raytracing.github.io/books/RayTracingTheNextWeek.html

[14] https://raytracing.github.io/books/RayTracingTheRestOfYourLife.html

[15] https://pbr-book.org/3ed-2018/Light_Transport_I_Surface_Reflection/Sampling_Light_Sources

[16] https://community-sciences.unilim.fr/pluginfile.php/193944/mod_resource/content/1/22_23_ISIR_TP6_Reflexion_refraction.pdf

[17] https://en.wikipedia.org/wiki/Fresnel_equations

[18] https://en.wikipedia.org/wiki/Snell%27s_law

[19] https://iquilezles.org/articles/distfunctions/

[20] https://damassets.autodesk.net/content/dam/autodesk/research/publications-assets/pdf/rendu-realiste-de-surfaces.pdf

[21] https://www.scratchapixel.com/

[22] https://www.labri.fr/perso/pbenard/teaching/pghp/slides/LancerRayon.pdf

[23] http://rodolphe-vaillant.fr/entry/87/normal-to-an-implicit-surface

[24] https://pellacini.di.uniroma1.it/teaching/graphics13a/slides/03_raytracing.pdf