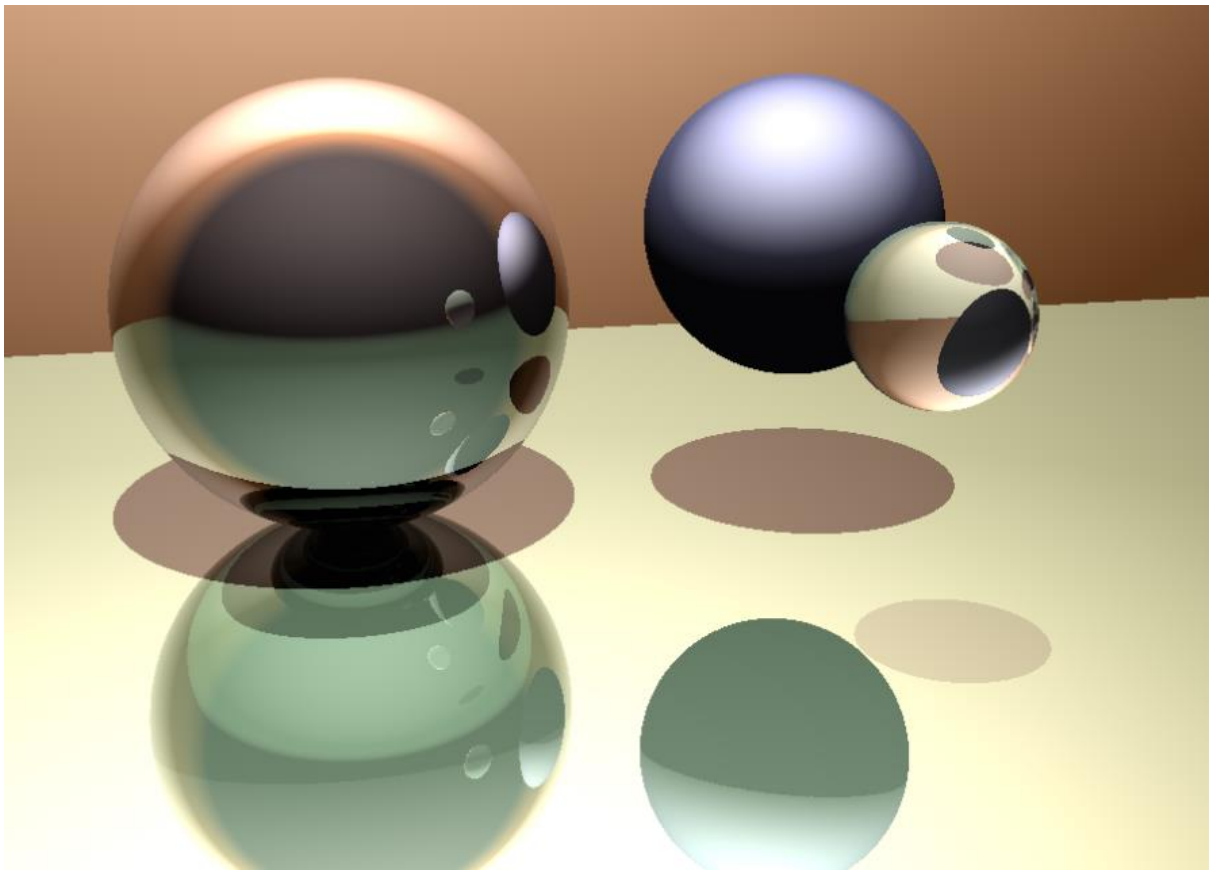
Rapport de l’application aux lancers de rayon



Amphoux Gabriel G5A
Duraysseix Romain G5A

Année universitaire 2018/2019

Université de Limoges

Table des matières

Rappel de l'énoncé

Compléter l'ensemble des "trous" (repérés par "A FAIRE") présents dans le code fourni (attention, ce code ne doit pas être diffusé !). Pensez à fournir des jeux d'essai / tests lorsque nécessaire, et profitez de cette question pour décrire l'architecture globale de votre projet (vous êtes libres de poursuivre le développement en respectant le code fourni ou de proposer vos propres solutions, différentes)

Analyse du travail à faire

Tout d'abord, nous avons dû analyser et comprendre le code déjà fourni. Cette étape fut difficile. Puis, il a fallu que nous réfléchissons par où commencer pour réaliser ce projet. Nous avons dû faire de nombreux schémas pour essayer de comprendre le fonctionnement. Lors des premières séances, nous avons passé notre temps à essayer de comprendre le fonctionnement et cela nous a permis de bien commencer.

Pour réaliser cette image, nous devons commencer par faire toutes les fonctions et les procédures de la base 3D comme le calcul pour normaliser un vecteur. Ensuite, nous avons à faire le calcul de l'image de la caméra et le lancer de rayon où l'objet prend la couleur de l'impact du rayon sur l'objet pour avoir le décor de la scène sur l'image. Puis, il faut ajouter les sphères au décor. Il faut donc faire l'intersection des sphères par rapport aux lancers de rayon pour les afficher. De plus, il faut aussi gérer la lumière ponctuelle de la scène pour faire les ombres des objets par rapport à la lumière. Il faut aussi modifier le lancer de rayon pour que l'objet prend la couleur de l'impact du rayon sur l'objet mais en fonction de l'illumination. Puis, la lumière crée un effet de réfraction sur les sphères, il faut donc gérer ce cas dans le lancer de rayon, ce qui va permettre d'afficher tous les reflets des sphères par rapport à la lumière. Pour finir, il faut gérer la matière des sphères pour savoir comment la sphère réagit par rapport à la lumière car en fonction de la matière, cette dernière réagit de différentes façons.

Architecture globale de notre projet

L’architecture globale de notre projet, est la suivante. Nous avons tout d’abord commencé par voire ce que nous devions faire en premier lieu. Nous avons donc rempli le fichier bases3d.cpp grâce aux indications de bases3d.hpp. Ensuite, nous avons effectué des tests sur nos vecteurs pour ne pas partir sur des mauvaises bases.

Ensuite nous avons cherché à comprendre sur quoi nous orienter. Nous avons essayé de remplir camera.cpp ainsi que rayon.cpp, après ça nous avons eu nos premiers résultats.

Ensuite nous voulions afficher des sphères, nous nous sommes donc penchés sur sphere.cpp. Après cela, nous avons pu avoir nos sphères, mais cela ressemblait à un dessin en 2 dimensions. Nous avons donc essayé de gérer les ombres pour avoir un rendu qui ressemblait à de la 3 dimensions.

Pour la diffraction et la réflexion nous avons retravaillé rayon.cpp, nous avons rajouté quelques lignes et le résultat que vous pouvez voire est donc notre rendu final.

Algorithmes

camera.cpp

```
////////////////////////////////////////////////////////////////////////////
//
// Cette procédure permet de calculer une image.
//
// Entrées : complexite : Entier
//               pm : Une map de pixel
//               lo : Liste d'objet 3D
//               ll : Liste de lumiere
//
// Sorties : pm : Une map de pixel
//              lo : Liste d'objet 3D
//              ll : Liste de lumiere
//
//
////////////////////////////////////////////////////////////////////////////
Procédure Calculer_image(complexite : Entier ; pm : PixelMap, lo : Liste<Objet3D>, ll : Liste<Lumiere>)
Début
        Initialistation :
        foyer : Point3D // Foyer optique de la camera
        droite : Vecteur3D // Vecteur partant sur la droite dans le plan de l'ecran
        dx, dy : Double // dimension des macro-pixels
        x, y : Entier // Position dans l'image du pixel en cours de calcul
        hg : Point3D // Position du pixel au centre du premier macro-pixel de l'ecran (en haut a gauche)
        pt : Point3D  // Position de l'intersection entre le rayon a lancer et l'ecran
        ray : Rayon  // Rayon que l'on lance
        vect : Vecteur3D // Vecteur directeur du rayon que l'on lance
        index : Entier // Indice du pixel en cours de traitement

        // On calcule la position du foyer de la camera
        foyer -> centre - (dir * dist)
        // On calcule le vecteur unitaire "droite" du plan
        droite -> dir.Cross(haut)
        // On calcule le deltaX et le deltaY
        dx -> largeur / pm.Largeur()
        dy -> hauteur / pm.Hauteur()
        // On calcule la position du premier point de l'ecran que l'on calculera
        hg -> centre + (droite * ((dx / 2) - (largeur / 2))) + (haut * ((hauteur / 2) - (dy / 2)))
        // Pour chaque pixel de l'image a calculer
        index -> 0
        Pour y De 0 A pm.Hauteur() Faire
        Pour x De 0 A pm.Largeur() Faire
                        // On calcule la position dans l'espace de ce point
                pt -> hg + (droite * (dx * x)) - (haut * (dy * y))
                        // On prepare le rayon qui part du foyer et qui passe par ce point
                ray.Orig(pt)
                vect -> pt - foyer
                vect.Normaliser()
                ray.Vect(vect)
```

```
                ray.Milieu(1)
                        // Et on enregistre la couleur du rayon dans l'image
                pm.Map(index++, ray.Lancer(lo, ll, complexite))
        FinFaire
        FinFaire
Fin
```

lumiere.cpp

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//
// Cette fonction permet de gérer l'éclairage ponctuelle (principalement les ombres).
//
// Entrées : r : Rayon
//           i : Intersection du rayon
//           p : Point3D
//           lo : Liste d'Objet3D
//
// Sorties : i : Intersection du rayon
//           p : Point3D
//           lo : Liste d'Objet3D
//
// Retourne : La couleur de l'objet en fonction de son éclairage
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
Fonction RVB Lumiere_Ponctuelle (r : Rayon ; i : Intersection3D, p : Point3D, lo : Liste<Objet3D>)
Début
        Initialisation :
        ray : Rayon ray
        imp : Intersection3D // Intersection de l'impact du rayon
        pos : Point3D // Position de l'impact
        res : RVB
        li : C_Liste_Intersection

        //Initialisation du rayon
        ray.Orig(p)
        ray.Milieu(1)
        ray.Vect(Vecteur3D(ray.Orig(), this->Pos()))

        // Au depart, le point au bout du rayon est noir


        // On recherche l'impact du rayon
        Si (ray.Intersections(li, lo)) Alors

        Si (li.Premier() != 0) Alors
                imp -> Pointeur li.Premier()

                // On calcule la position de l'impact (s'il y en a un)
                pos -> ray.Orig() + (ray.Vect() * imp.Dist())
                Si (this->Pos().X() > pos.X()) Alors
                res -> RVB(0, 0, 0)
                Fin Si
                // On peut economiser de la memoire en vidant des maintenant la liste des intersections
(avant les appels recursifs)
```

```
                li.Vider()
        Sinon
                //Calcule de l'ombre en fonction de la normale
                Double cosAngle -> (ray.Vect() * i.Norm()) / (ray.Vect().Longueur() * i.Norm().Longueur())
                res -> res + i.Objt()->Couleur() * couleur * cosAngle
        Fin Si

        Sinon
        // Le rayon n'a rien touche
        Fin Si
        Retourne res
Fin
```

rayon.cpp

```
////////////////////////////////////////////////////////////////////////////////
//
// Cette fonction permet de gérer le lancer de rayon.
//
// Entrées : recur : Entier
//              lo : Liste d'objet 3D
//              ll : Liste de lumiere
//
// Sorties : lo : Liste d'objet 3D
//             ll : Liste de lumiere
//
// Retourne : La couleur de l'objet s'il y une intersection
//
////////////////////////////////////////////////////////////////////////////////
Fonction RVB Lancer(recur : Entier ; lo : Liste<Objet3D>, ll : Liste<Lumiere>)
Début
        Initialisation :
        li : C_Liste_Intersection // Liste d'intersections
        imp : Intersection3D // Intersection de l'impact du rayon
        pos : Point3D // Position de l'impact
        res : RVB // Couleur du résultat
        ray : Rayon // Rayon secondaire

        // Au départ, le point au bout du rayon est noir
        res -> RVB(0, 0, 0)

        // On recherche l'impact du rayon
        Si (Intersections(li, lo)) Alors
        Si (li.Premier() != 0) Alors
                imp -> Pointeur li.Premier()
                // On calcule la position de l'impact (s'il y en a un)
                pos -> orig + (vect * imp.Dist())

                // On peut economiser de la memoire en vidant des maintenant la liste des intersections
(avant les appels recursifs)
                li.Vider()

                Si (imp.Objt()->Kr() > 0) Alors
                        ray.Orig(pos)
                        ray.Vect(this->Vect())
```

```
                        ray.Vect().Normaliser()
                        ray.Vect(ray.Vect().Reflechir(imp.Norm()))
                        res -> res + ray.Lancer(lo, ll, recur--) * imp.Objt()->Kr()
                Fin Si
                Pour (ll.Premier() De ll.Courant() A ll.Suivant()) Faire
                res -> res + ll.Courant()->Illumination(ray, imp, pos, lo)

                Fin Faire

        Sinon
                // Le rayon n'a rien touche
        Fin Si
        Sinon
        // Le rayon n'a rien touche
        Fin Si

        Retourne res
Fin
```

sphere.cpp

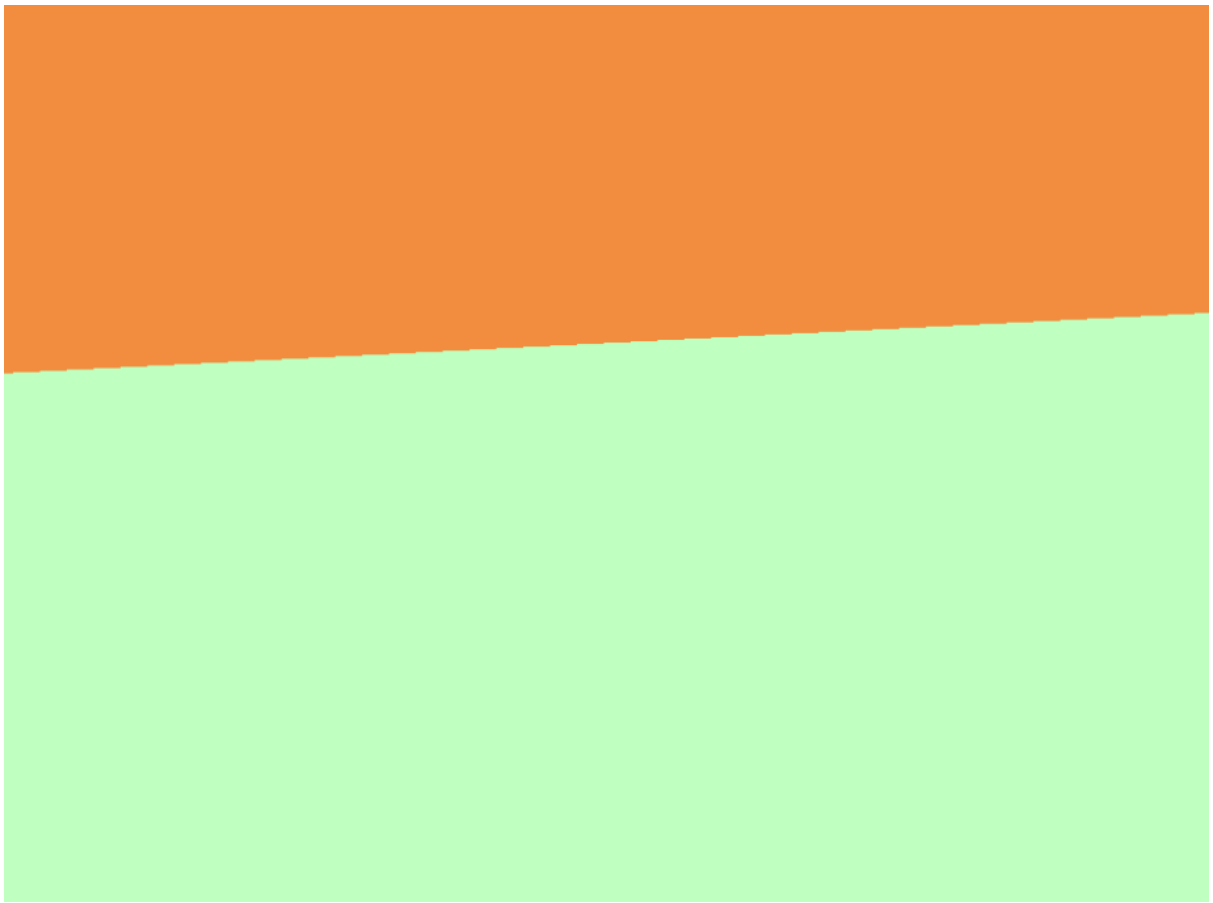
```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//
// Cette fonction permet de gérer les intersections des rayons et de la sphère.
//
// Entrées : r : Rayon
//              l : Liste d'intersection
//
// Sorties : r : Rayon
//             l : Liste d'intersection
//
// Retourne : Un entier qui indique si le rayon a rencontrer une intersection
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////
Fonction Entier Intersection(;r : Rayon, l : C_Liste_Intersection)
Début
        Initialisation :
        //Initialisation des valeurs pour le calcul de delta
        a : Double -> 1
        b : Double -> (r.Orig() - this->Centre())*2.0*r.Vect()
        c : Double -> (r.Orig() - this->Centre())*(r.Orig() - this->Centre()) - this->SphereRayon()*this-
>SphereRayon()
        delta : Double -> b * b - 4 * a*c // Calcul de delta
        racine1 : Double
        racine2 : Double // Si delta est > 0 alors il y a deux racines
        vecteur1 : Point3D
        vecteur2 : Point3D //S'il y a deux racines il y a deux point d'intersection
        i1 : Intersection3D
        i2 : Intersection3D //Si il y a deux racines il y a deux intersection à rajouter

        Si (delta < 0) Alors
        // le rayon n'est pas en intersection avec la sphère
        Retourne 0;
        Sinon
        Si (delta = 0) Alors
```

```
            // il existe un point d'intersections avec la sphère
            racine1 -> (-b) / (2 * a)

            // On initialise l'intersection
            vecteur1 -> r.Orig() + (r.Vect() * racine1)

            i1 -> Intersection3D(racine1, this, 0)
            i1.Norm(Normale(vecteur1))

            // Puis on la rajoute dans la liste des intersections
            l.Ajouter(i1)

            Retourne 1
        Sinon
            racine1 -> (-b + Racine(delta)) / (2 * a)
            racine2 -> (-b - Racine(delta)) / (2 * a)

            vecteur1 -> r.Orig() + (r.Vect() * racine1)
            vecteur2 -> r.Orig() + (r.Vect() * racine2)


            // On initialise l'intersection
            i1 -> Intersection3D(racine1, this, 0)
            i1.Norm(Normale(vecteur1))

            i2 -> Intersection3D(racine2, this, 0)
            i2.Norm(Normale(vecteur2))

            // Puis on la rajoute dans la liste des intersections
            l.Ajouter(i1)
            l.Ajouter(i2)

            Retourne 1

        Fin Si

        Fin Si
Fin
```

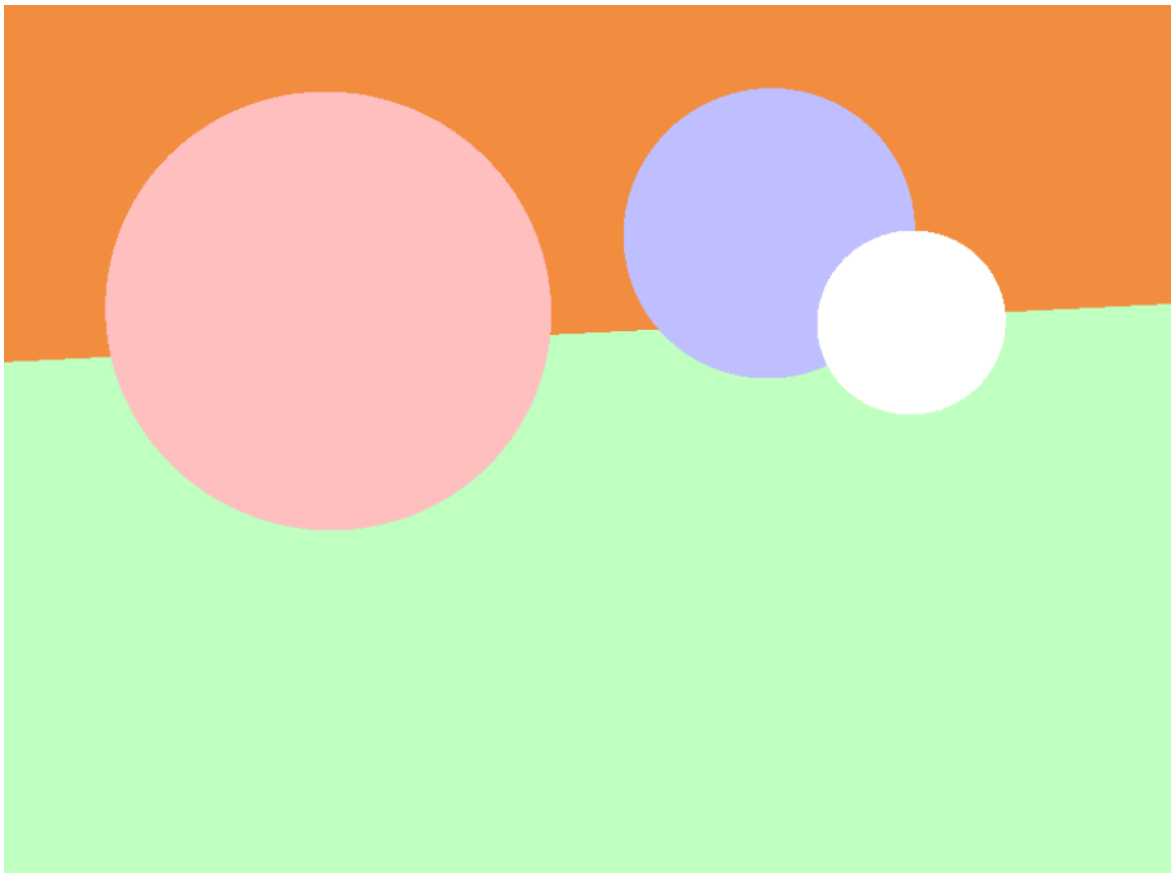
Jeux d’essai

Tout d’abord, nous avons une image toute noire.

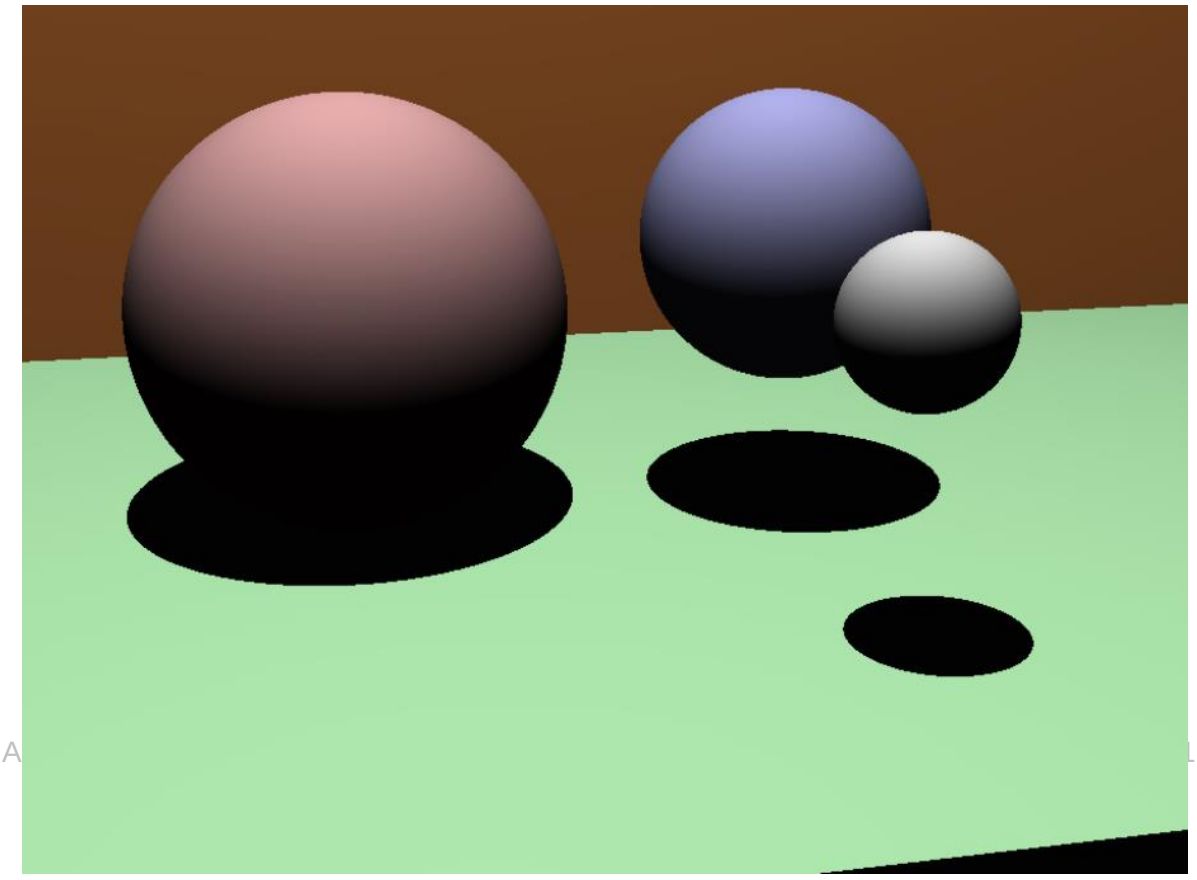
Mais une fois les fonctions et les procédures de la base 3D faites, le calcul d’image de la caméra et le lancer de rayon fait, nous avons le décor de la scène.



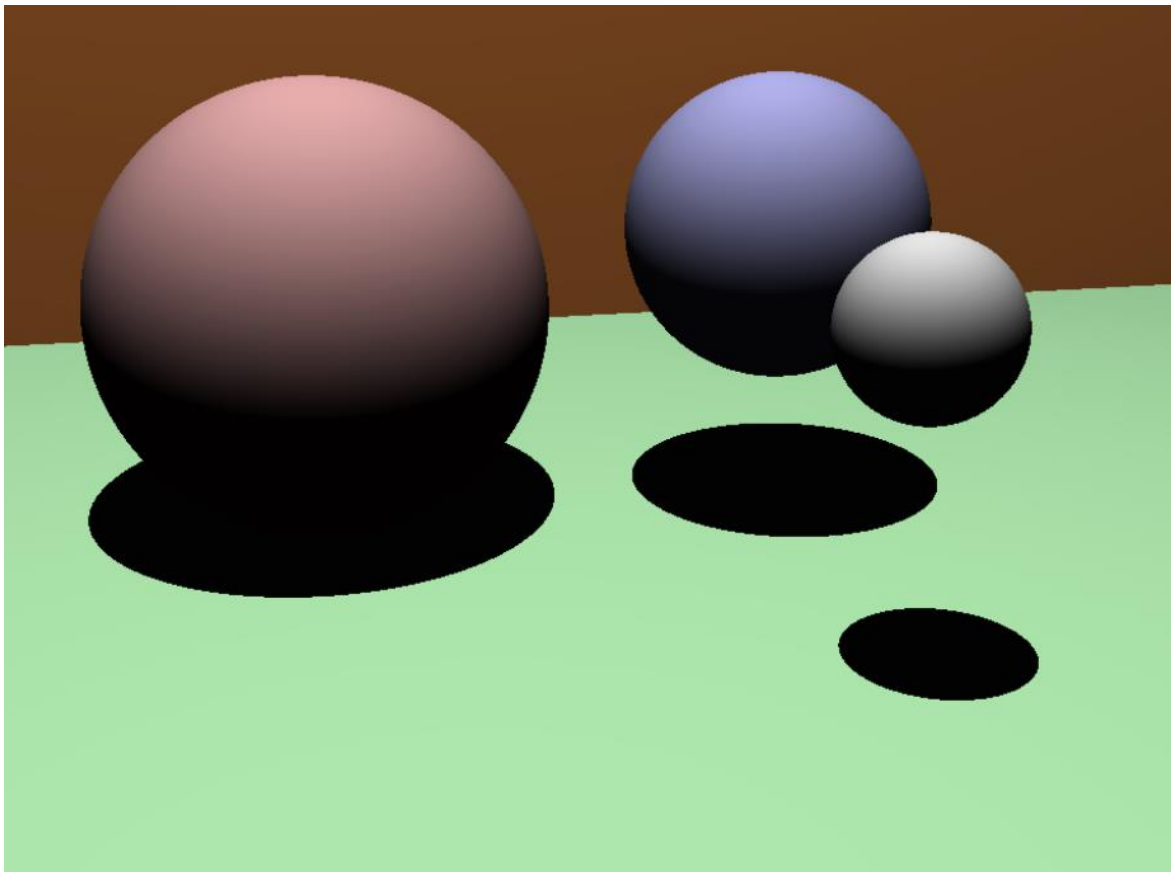
Ensuite, nous avons fait l’intersection des sphères pour avoir ces dernières sur l’image.



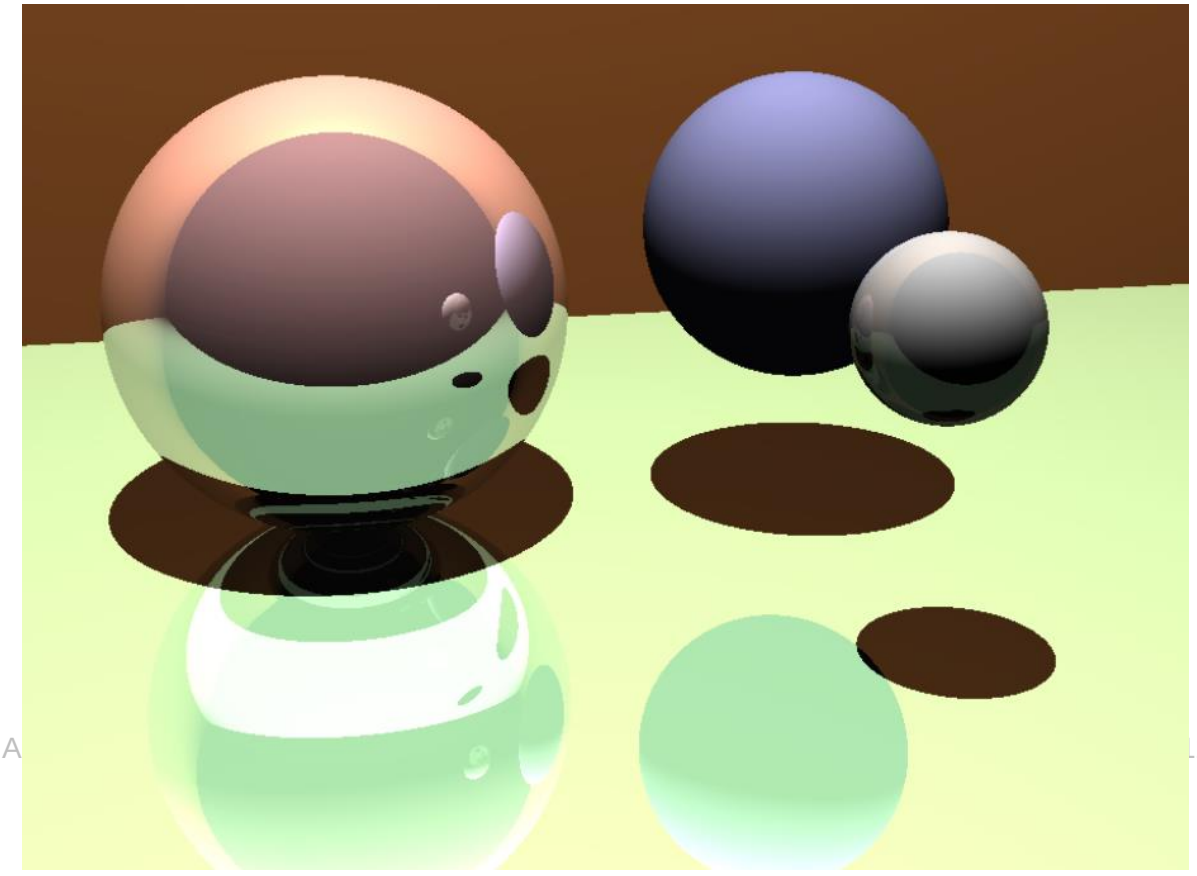
Puis, nous avons gérer la lumière ponctuelle. Ce qui nous a amené à changer le lancer de rayon.



A partir de ce moment, notre image commençait vraiment à ressembler à l'image finale. Nous avons juste déplacé de 1 en "z" la caméra pour ne plus avoir l'ombre en bas à droite.



Pour finir, nous avons gérer les reflets des sphères par rapport à la lumière.



Remarques

Nous n’avons pas réussi à reproduire parfaitement l’image final mais nous sommes quand même contents du résultat que nous avons obtenu. Grâce à ce projet nous avons beaucoup appris du développement 3D même si nous avons trouvé cela très difficile à causes des difficultés en mathématiques et en physique.