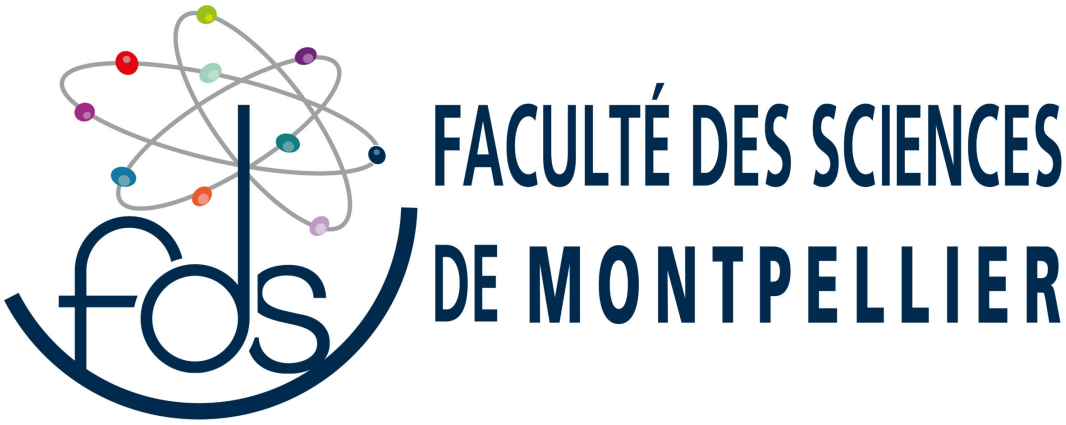
Université de Montpellier

M1 - IMAGINE - Programmation 3D
Compte Rendu - RayTracing phase 1

Etudiant :
Guillaume Bataille

Encadrant :
Noura Faraj
Marc HARTLEY

Année 2022-2023



Sommaire

1. Contexte et Objectif

Afin de pouvoir réaliser ce projet de rayTracing qui va nous permettre de rendre une image via des lancer de rayons, nous allons avoir 3 phases.
Dans cette premiere phase, nous allons essayer de comprendre le code et de faire nos premiers lancer de rayons !

2. La fonction RayTrace

RayTrace consiste à retourner le materiel (couleur) du point qu'on intersecte avec notre rayon (evidemment uniquement si le rayon touche quelque chose) !

```
1     Vec3 rayTrace(Ray const &rayStart, double znear)
2     {
3         // TODO appeler la fonction recursive
4         Vec3 color;
5         RaySceneIntersection RSI = computeIntersection(rayStart, znear);
6         if (RSI.intersectionExists) // Si y'a intersection
7         { // Si il y a bien une intersection avec un objet de la scène
8             if (RSI.typeOfIntersectedObject == 0)
9             {                                                          // On est avec une sphère (0)
10                color = spheres[RSI.objectIndex].material.diffuse_material; // Récup couleur
11            }
12            else if (RSI.typeOfIntersectedObject == 1)
13            {                                                          // On est avec une square (1)
14                color = squares[RSI.objectIndex].material.diffuse_material; // Récup couleur
15            }
16        }
17
18        return color;
19    }
20
```

3. Intersection avec une sphère

Afin de determiner une intersection avec une sphère, on défini une fonction intersectsphère qui va, avec l'aide d'équations, determiner si une intersection existe bien entre une sphère (Rayon et centre) et un rayon.

```
1   RaySphereIntersection intersect(const Ray &ray) const {
2       RaySphereIntersection intersection;
3       Vec3 d = ray.direction();
4       Vec3 o = ray.origin();
5       Vec3 c = m_center;
6       double r = m_radius;
7
8       //Equation d'intersection rayon-sphère
9       // t2 d.d + 2t d.(o-c) + ||o-c||² - r² = 0
10      double A,B,C;
11      A = d.dot(d,d);
12      B = 2 * (d.dot(d,o-c));
13      C = ( (o-c).length() * (o-c).length() ) - r*r ;
14
15      // discriminant : b*b - 4*a*c
16      double Discriminant;
17      Discriminant = B*B -(4 * A * C);
18
19      // Si on a des racines alors il y a deux intersections (ou une commune)
20      double racine1, racine2;
21      if (Discriminant >= 0){
22
23          racine2 = fmax( -B + sqrt(Discriminant), -B - sqrt(Discriminant)) ;
24          racine1 = fmin( -B + sqrt(Discriminant), -B - sqrt(Discriminant)) ;
25
26          racine1/= 2*A;
27          racine2/= 2*A;
28
29      intersection.intersectionExists = true;
30      intersection.t = racine1;
31      intersection.intersection = o + (d*racine1);
32      intersection.secondintersection = o + (d*racine2);
33      intersection.normal = intersection.intersection - c;
34      }
35      else{ // Pas d'intersection
36          intersection.intersectionExists =false;
37      }
38      return intersection;
39  }
```

4. Intersection avec un square

Tout comme pour sphère, on calcule l'intersection entre un rayon et un carré via une fonction. Dans un premier temps, on check si le rayon intersecte le plan du square, si oui on projette l'intersection plan sur le repère du carré et on determine si on est dans le carré ou non !

```
1     RaySquareIntersection intersect(const Ray &ray) const
2     {
3         RaySquareIntersection intersection;
4
5         Vec3 d = ray.direction();
6         Vec3 o = ray.origin();
7         Vec3 n = normalvec();
8
9         Vec3 a = bottomLeft();   // D
10        Vec3 a1 = bottomRight(); // C
11        Vec3 a2 = upRight();     // B
12        Vec3 a3 = upLeft();      // A
13
14        Vec3 inter;
15
16        double D = a.dot(a, n);
17
18        if (Vec3::dot(n, d) <= EPSILON_up && Vec3::dot(n, d) >= -EPSILON_down)
19        {
20            // std::cout << "dot : " << Vec3::dot(n, d) << std::endl;
21            intersection.intersectionExists = false;
22            return intersection;
23        }
24
25        double t = (D - o.dot(o, n)) / Vec3::dot(d, n); // Fonction de t
26        // double t = 0;
27
28        if (t > 0) // On est dans le plan
29        {
30            inter = o + (t * d); // Coordonnée du point d'intersection
31            // Changement de repère : on se mets dans le repère du square
32            // ABx = l'axe X du carré
33            // ABy = l'axe Y du carré
34            Vec3 ABx = a1 - a;
35            Vec3 ABy = a3 - a;
36            // Le vecteur entre le 0,0,0 du carré et le point intersection
37            Vec3 AC = inter - a;
38            // Les normes de projection du point intersection sur l'axe X et Y
39            double APx = Vec3::dot(ABx, AC) / ABx.length();
40            double APy = Vec3::dot(ABy, AC) / ABy.length();
41
```

```
42              // Si la norme Proj x < norme ABx alors on est dans le carré en X (cf en y)
43              if (APx <= ABx.length() && APx > 0 && APy <= ABy.length() && APy > 0)
44              {
45                  intersection.intersectionExists = true;
46                  intersection.t = t;
47                  intersection.normal = n;
48                  intersection.intersection = inter;
49                  intersection.u = APx;
50                  intersection.v = APy;
51              }
52              else
53              {
54                  intersection.intersectionExists = false;
55              }
56          }
57          return intersection;
58      }
```

5. Compute Intersection

Le compute intersection sert à être appeler a chaque lancer de rayon et il va calculer pour tout les elements de la scène leurs intersections. Cependant, chaque intersection (si elle existe) possède une valeur t, qui est sa distance entre l'origine du rayon et son intersection. On choisi donc de ne garder que le t le plus petit et donc de n'afficher que ce qui est vu en premier !

```
1   RaySceneIntersection computeIntersection(Ray const &ray, double znear)
2   {
3       RaySceneIntersection result;     // L'interesection la plus proche
4       RaySphereIntersection raysphere; // Definit mon IntersectionSphere
5       RaySquareIntersection raysquare; // Definit mon IntersectionSquare
6
7       for (unsigned int i = 0; i < spheres.size(); i++)
8       {                                          // Pour toute les sphères rencontrées;
9           raysphere = spheres[i].intersect(ray); // On teste l'intersection
10          if (raysphere.intersectionExists)
11          { // Si l'intersection a bien lieu
12              if (raysphere.t < result.t && raysphere.t != 0 && raysphere.t > znear)
13              { // On regarde si on a déja eu une intersection, si c'est pas le cas
14
15                  result.intersectionExists = true;         // On le mets a true
16                  result.objectIndex = i;                   // On recup l'id de l'élément touché
17                  result.typeOfIntersectedObject = 0;       // On set a 0 le type d'objet rencontré (avec 0 po
18                  result.t = raysphere.t;                   // On recup son t, distance entre la caméra et le
19                  result.raySphereIntersection = raysphere; // On récupère l'intersection
20              }
21          }
22      }
23      for (unsigned int j = 0; j < squares.size(); j++)
24      {
25          raysquare = squares[j].intersect(ray); // On teste l'intersection
26          if (raysquare.intersectionExists)
27          { // Si l'intersection a bien lieu
28              if (raysquare.t < result.t && raysquare.t != 0 && raysquare.t > znear)
29              { // On regarde si on a déja eu une intersection, si c'est pas le cas
30
31                  result.intersectionExists = true;         // On le mets a true
32                  result.objectIndex = j;                   // On recup l'id de l'élément touché
33                  result.typeOfIntersectedObject = 1;       // On set a 1 le type d'objet rencontré
34                                                            //(avec 0 pour sphère et 1 pour square )
35                  result.t = raysquare.t;                   // On recup son t, distance entre la caméra
36                                                            //et le point d'intersection.
37                  result.raySquareIntersection = raysquare; // On récupère l'intersection
38              }
39          }
40      }
```

```
41          return result;
42      }
```

6. Visuels

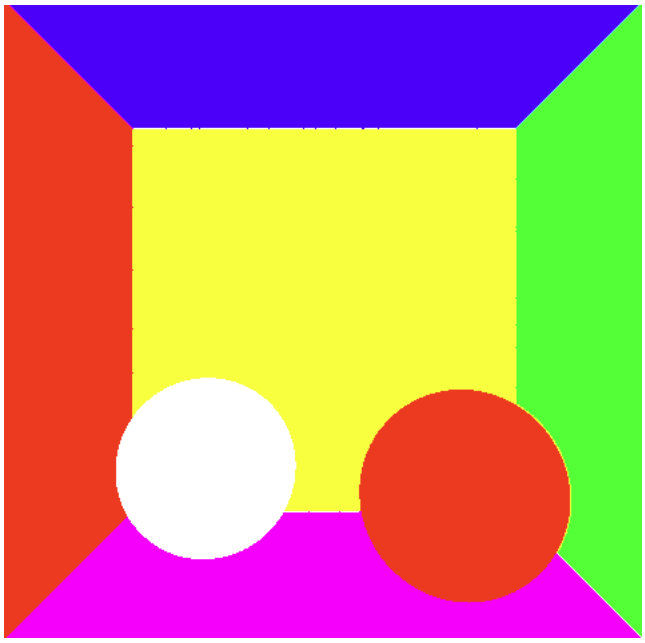


FIGURE 6.1 – Cornel box de la phase 1