Université de Montpellier

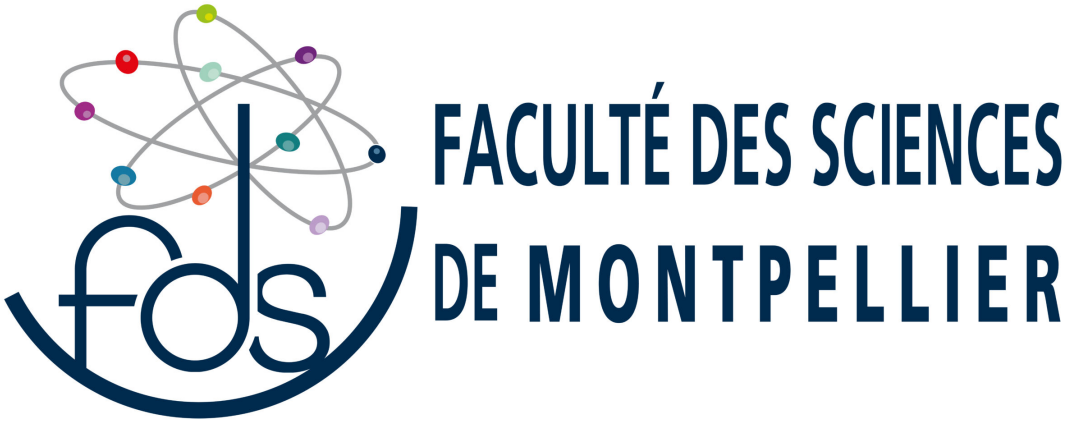
M1 - IMAGINE - Programmation 3D
Compte Rendu - RayTracing phase 3

Etudiant :
Guillaume Bataille

Encadrant :
Noura Faraj
Marc HARTLEY

Année 2022-2023

Mon git :



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 troisième phase, nous allons calculer la reflection d'un rayon pour avoir des objet reflechissant et nous allons gérer les meshs avec le calcul d'intersection de mesh pour avoir des modèles plus interessant.

2. Visuels

Voila des visuels pour montrer ou j'en suis fin phase 3 mais je vous invite a regarder mes nombreux fails dans le repertoire de mon projet.

Voici le visuel demandé par la prof pour montrer la ou en est :

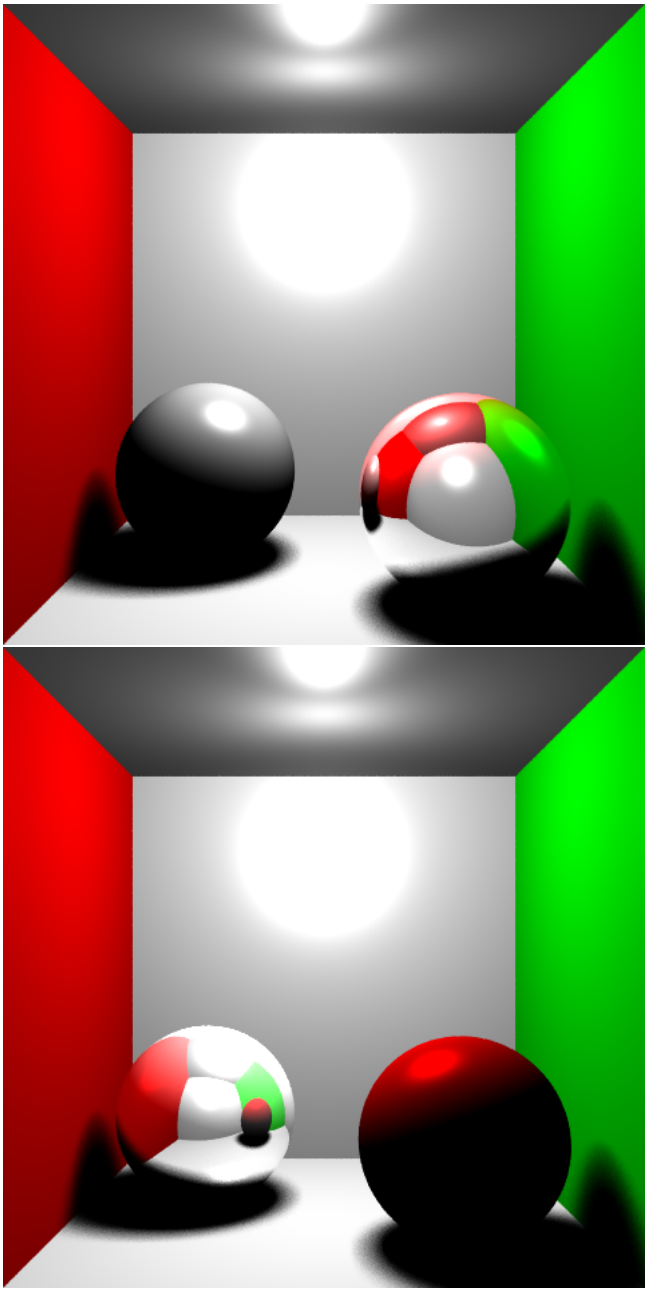


FIGURE 2.1 – Original soft shadow + reflection

A partir d'ici j'ai changé le square de la light d'ou la différence dans les shadow :

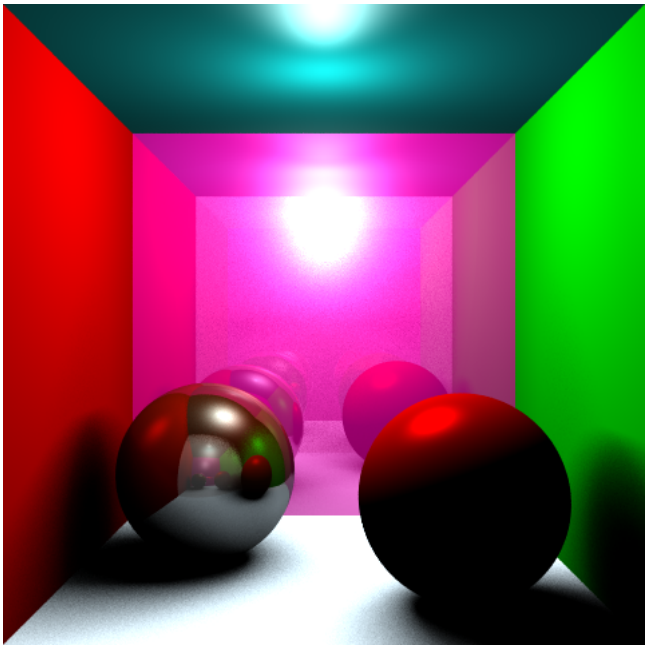


FIGURE 2.2 – Reflection

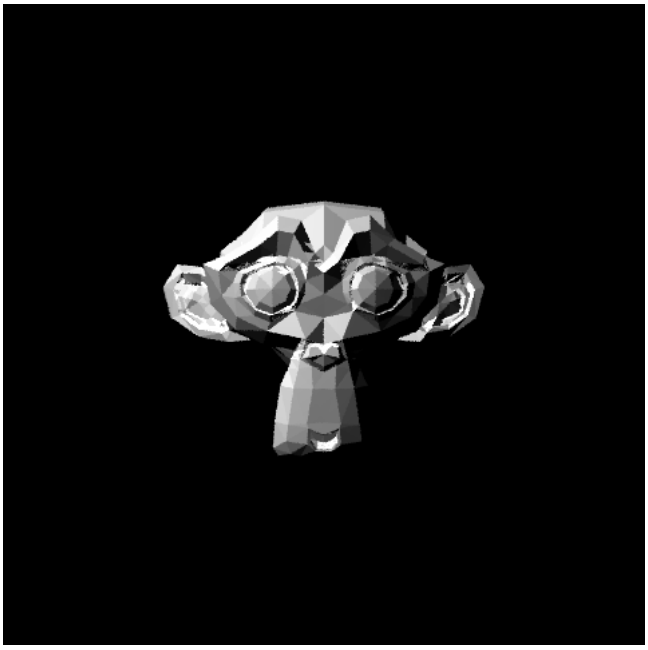


FIGURE 2.3 – Intersection avec Suzanne non interpolés

FIGURE 2.4 – Intersection avec Suzanne normales interpolées

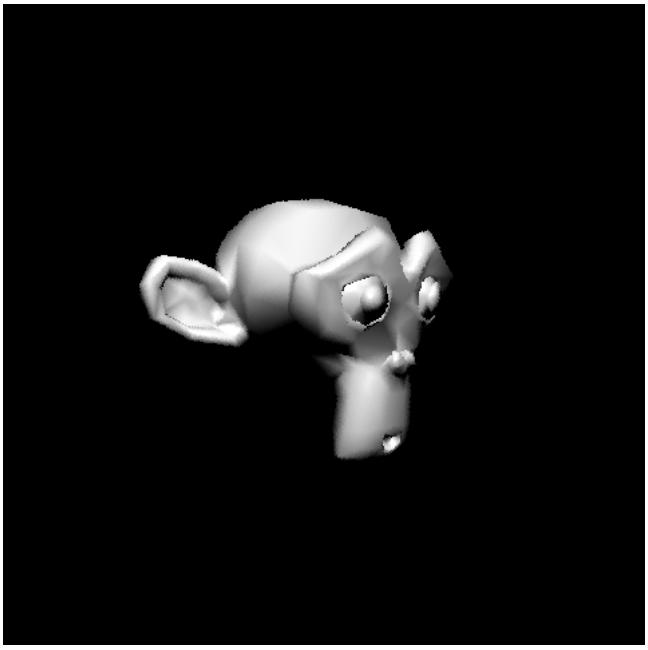


FIGURE 2.5 – Intersection avec Suzanne normales interpolées

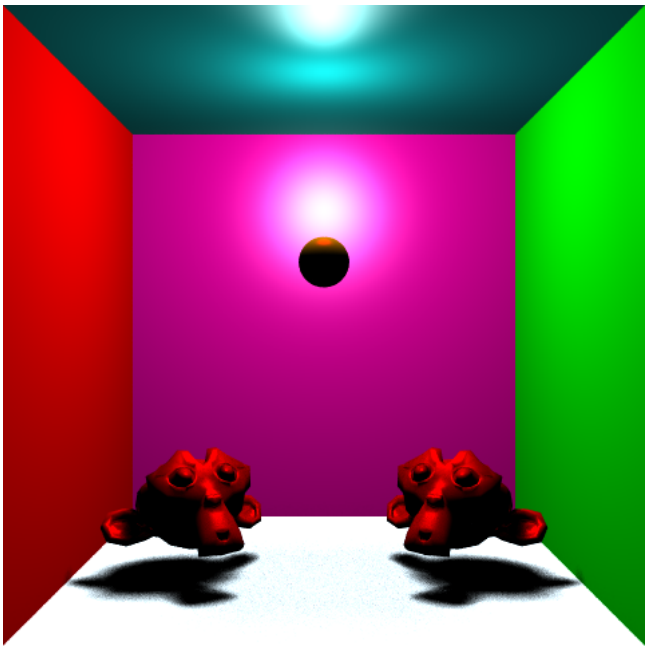


FIGURE 2.6 – suzanne x 2 qui prient pour la kit kat ball (soft shadow + mesh rendering)

3. Reflection

Le but est de compute le vecteur de reflection a partir de la normale et de l'incident a chaque calcul de rayon (car simple) puis si ce qu'on touche est de type mirroir (ou s'il est un peut spécular) on bounce et on retire un au bounce restant

```
1       Vec3 rayTraceRecursive(Ray ray, int NRemainingBounces, double znear)
2       {
3
4           // TODO appeler la fonction recursive
5           Vec3 color;
6           int nb_current_bounce = 0;
7           RaySceneIntersection RSI = computeIntersection(ray, znear,false);
8           if (!RSI.intersectionExists) // Si y'a pas d'intersection
9               return Vec3(0, 0, 0);
10
11          Material mat = getRayMaterial(RSI); // On recupère le material courant
12          color = mat.ambient_material;       // Récup couleur ambiant
13          phong(RSI, color, mat, ray);        // On appelle phong pour y ajouter la couleur de l'intersection (dif
14          shadow(RSI, color, mat);            // On appelle shadow qui va compute les ompbres dans color
15
16          // Recursive part :
17          //      Reflection sur mirroir
18          if (mat.type == Material_Mirror && NRemainingBounces > 0)
19          { //On teste si on est avec mirroir ET si il reste des rebonds
20              // Parametrage du nouveau point de depart du rayon
21              Vec3 new_origin = RSI.position;            //La position de l'intersection devient l'origine du rayo
22              Vec3 new_direction = RSI.bounce_direction; // La direction bounce (reflexion via old direction et no
23              Ray Ray_bounce = Ray(new_origin, new_direction);
24              color += rayTraceRecursive(Ray_bounce, NRemainingBounces - 1, 0.001) / NRemainingBounces; // On reti
25          }
26          // znear =0.001 c'est pour pas qu'il se cogne sur lui même ou sur son voisin
27
28          return color;
29      }
30
```

4. Triangle.h

Comme les mesh sont composés de triangle, avant de gerer un mesh, on gère un triangle via une méthode d'interesect pour un triangle.
Pour se faire, on utilise l'intersect.square jusqu'a être sur le plan du triangle. Puis on va utiliser les barycentres en verifiant si a partir d'un des trois coté, le triangle est du coté positif (vers le triangle) ou négatif (hors du triangle) et cela via l'aire du triangle coté + point intersect.
Si tout les points sont du bon coté, on est sur le triangle ! Et en plus on peut utiliser l'aire de ses triangles pour pondérer la normale et la couleur.

```
1
2       RayTriangleIntersection getIntersection( Ray const & ray ) const {
3
4           RayTriangleIntersection intersection;
5           Vec3 d = ray.direction();
6           Vec3 o = ray.origin();
7           Vec3 n = getnorm();
8           float area = getarea();
9           n.normalize();
10
11          Vec3 s0 = gets0();
12          Vec3 s1 = gets1();
13          Vec3 s2 = gets2();
14
15          float u,v;
16
17          Vec3 s0s1 = s1 - s0;
18          Vec3 s0s2 = s2 - s0;
19          Vec3 N = Vec3::cross(s0s1,s0s2);
20          float denom = Vec3::dot(N,N);
21
22          Vec3 inter;
23
24          double D = Vec3::dot(s0, n);
25
26          // 1) check that the ray is not parallel to the triangle:
27
28           if ((Vec3::dot(n, d) <= 0.01) && (Vec3::dot(n, d) >= -0.01))
29          {
30              intersection.intersectionExists = false;
31              return intersection;
32          }
33
34          // 2) check that the triangle is "in front of" the ray:
35          double t = (D - Vec3::dot(o, n)) / Vec3::dot(d, n); // Fonction de t
36          double orientation = Vec3::dot(d,  n);
37          if (t > 0 && orientation <= 0) // On est dans le plan et on est face au plan (pour ne pas render le dos
```

```
38          //if(t > 0)
39          {
40          // 3) check that the intersection point is inside the triangle:
41          // CONVENTION: compute u,v such that p = w0*c0 + w1*c1 + w2*c2, check that 0 <= w0,w1,w2 <= 1
42
43              inter = o + (t * d); // Coordonnée du point d'intersection
44              Vec3 C;
45              // edge 0
46              Vec3 edge0 = s1 - s0;
47              Vec3 vp0 = inter - s0;
48              C = Vec3::cross(edge0,vp0);
49              if (Vec3::dot(N,C) < 0){ // Sort de l'aire
50              intersection.intersectionExists = false;
51              return intersection;
52              }
53
54              // edge 1
55              Vec3 edge1 = s2 - s1;
56              Vec3 vp1 = inter - s1;
57              C = Vec3::cross(edge1,vp1);
58              u = Vec3::dot(N,C);
59              if (u < 0){ // Sort de l'aire
60              intersection.intersectionExists = false;
61              return intersection;
62              }
63
64              // edge 2
65              Vec3 edge2 = s0 - s2;
66              Vec3 vp2 = inter - s2;
67              C = Vec3::cross(edge2,vp2);
68              v = Vec3::dot(N,C);
69              if ( v < 0){ // Sort de l'aire
70              intersection.intersectionExists = false;
71              return intersection;
72              }
73              // ON est bien interect
74              u /= denom;
75              v /= denom;
76              //std::cout<<" u :"<< u <<"  v : "<<v<<"  DENOM : " <<denom <<std::endl;
77
78                  intersection.intersectionExists = true;
79                  intersection.t = t;
80                  intersection.w0 = u;
81                  intersection.w1 = v;
82                  intersection.w2 = 1 - u - v;
83                  intersection.normal = n;
84                  intersection.intersection = inter;
85                  intersection.bounce_direction = d - 2 * intersection.normal *( Vec3::dot(d,intersection.normal))
86                  intersection.tIndex;
87          }
88
```

```
89            return intersection;
90        }
91
```

5. Box.h

Afin d'optimiser et de ne pas parcourir tout les triangles des mesh à chaque recherche d'un mershintersect, on utilise une bounding box a compute une seule fois.
Elle est calculé lors du buildarray d'un mesh et elle est construite en parcourant tout les sommets d'un mesh et en determinant BBmin et BBmax, puis on peut construire la box.
Enfin on défini les triangles composant la box pour faciliter le calcul de si oui ou non on tappe la cornell box (via un intersect box).

```
1
2  class Box{
3      //Attributs
4      public:
5      std::vector<Vec3> vertices;
6      std::vector<Vec3> triangles;
7      Vec3 BB_min;
8      Vec3 BB_max;
9      //Constructor :
10     Box(Vec3 BB_min, Vec3 BB_max){
11         this->BB_min = BB_min;
12         this->BB_max = BB_max;
13         buildbox();
14     }
15     Box(){}
16
17        void buildbox() {
18         vertices.resize(8);
19         triangles.resize(12);
20
21         vertices[0]=BB_min;
22         vertices[1]=Vec3(BB_min[0]  ,BB_min[1]  ,BB_max[2]  );
23         vertices[2]=Vec3(BB_min[0]  ,BB_max[1]  ,BB_min[2]  );
24         vertices[3]=Vec3(BB_min[0]  ,BB_max[1]  ,BB_max[2]  );
25         vertices[4]=Vec3(BB_max[0]  ,BB_min[1]  ,BB_min[2]  );
26         vertices[5]=Vec3(BB_max[0]  ,BB_min[1]  ,BB_max[2]  );
27         vertices[6]=Vec3(BB_max[0]  ,BB_max[1]  ,BB_min[2]  );
28         vertices[7]=BB_max;
29
30         triangles[0]=Vec3(0,1,2 );// Good
31         triangles[1]=Vec3(3,2,1 );
32
33         triangles[2]=Vec3(0,4,1 );// Good
34         triangles[3]=Vec3(5,1,4 );
35
36         triangles[4]=Vec3(0,4,2 );// Good
37         triangles[5]=Vec3(6,2,4 );
38
```

```
39          triangles[6]=Vec3(7,5,6 );// Good
40          triangles[7]=Vec3(4,6,5 );
41
42          triangles[8]=Vec3(7,3,6 );// Good
43          triangles[9]=Vec3(2,6,3 );
44
45          triangles[10]=Vec3(7,5,3 );// Good
46          triangles[11]=Vec3(3,1,5 );
47      }
48
49        RayTriangleIntersection intersect( Ray const & ray) {
50          RayTriangleIntersection result;
51          result.intersectionExists = false;
52          for (Vec3 t : triangles) // pour chaque triangle de ma box
53          {
54              Vec3 s0 = vertices[t[0]];
55              Vec3 s1 = vertices[t[1]];
56              Vec3 s2 = vertices[t[2]];
57              Triangle current_triangle = Triangle(s0,s1,s2);
58              RayTriangleIntersection current_intersect = current_triangle.getIntersection(ray);
59
60              if (current_intersect.intersectionExists){
61                  result = current_intersect;
62              }
63          }
64          return result;
65        }
66
```

6. Mesh.h

On a notre triangle intersect qui permet de trouver l'intersection d'un triangle, et la box pour opti. On peut simplement définir mesh intersect comme : si le rayon courant.intersectbox est faux, on arrete tout, sinon on parcours tout les triangles composant le mesh et on applique triangle.intersect jusqu'a trouver celui avec le t le plus petit.

```
1
2     RayTriangleIntersection intersect( Ray const & ray, bool shadowchecking ) const {
3         RayTriangleIntersection closestIntersection;
4         closestIntersection.t = FLT_MAX;
5         //RayTriangleIntersection BBox_interesection;
6         Box Bounding_box = Box(BB_min,BB_max);
7         //std::cout<<BB_max<<BB_min<<std::end;
8         bool touchingbox = Bounding_box.intersect(ray).intersectionExists;
9         if (!touchingbox){
10            closestIntersection.intersectionExists = false; // SI PAS TOUCHE BOX ON RETURN FALSE
11            return closestIntersection;
12        }
13        // Note :
14        // Creer un objet Triangle pour chaque face
15        for(size_t i = 0; i < triangles.size();i++) // Pour tout les triangles de mon mesh
16        {
17            //On recup les positions des 3 sommets de chaque triangle
18            MeshVertex s0 = vertices[triangles[i].v[0]];
19            MeshVertex s1 = vertices[triangles[i].v[1]];
20            MeshVertex s2 = vertices[triangles[i].v[2]];
21            //std::cout<<s0<<" "<<s1<<" "<<s2<<std::endl;
22            float triangleScaling = 1.000001;
23            //triangleScaling =1;
24            Triangle current_tri = Triangle(s0.position*triangleScaling,s1.position*triangleScaling,s2.position*
25
26            RayTriangleIntersection current_intersect = current_tri.getIntersection(ray);
27            if (current_intersect.intersectionExists && current_intersect.t < closestIntersection.t){ // Si on i
28                closestIntersection = current_intersect;
29                closestIntersection.tIndex = i;
30                closestIntersection.normal = closestIntersection.w0 * s0.normal + closestIntersection.w1 * s1.no
31                closestIntersection.normal.normalize();
32            }
33        }
34        //Triangle
35        // Vous constaterez des problemes de précision
36        // solution : ajouter un facteur d'échelle lors de la création du Triangle : float triangleScaling = 1.0
37        return closestIntersection;
38    }
39
```

7. Remarques

Pour effectuer le tp nous avons :
- Optimiser le calcul de square.intersect en le remplaçant comme l'intersect de deux triangles
- Ajouter des variables gloables define SCENENB, CURRENTSCENE, SHADOWTYPE ;
- Ajouter des scenes pour les meshs.
- Corriger un soucis de square light qui était bien trop proche
- Ajouter un pourcentage de progression du rendu dans le terminal

FIGURE 7.1 – Barre de progres + temps pris pour un rendu

A retenir :
-Tester regulièrement le code
-Tester toujours avec les param de rendu basse resolution pour voir si ça marche avant d'attendre 15min pour un rendu mauvais

Prochainement : -On essaiera d'avoir des refraction propre car actuellement c'est trop brouillon pour le mettre dans le rapport