Synthèse d'Image - Lancer de rayon - TD02

Premier raytracer

L'objectif de ce TD est d'implémenter des fonctions testant l'intersection entre un rayon et des formes géométriques afin d'afficher des premiers éléments à l'écran.

Exercice 01 : Des structures pour notre programme

Nous allons commencer à créer des structures pour représenter les différents objets de notre programme.

1. **Créez un fichier *include/raytracer.h* dans lequel vous définirez une structure "Ray" représentant un rayon. Un rayon est définit par une position d'origine et une direction.**
2. **Dans le fichier *include/raytracer.h*, créez une structure "Intersection" avec un champ de position et un champ de couleur.**
3. **Créez un fichier *include/shape.h* dans lequel vous définirez une structure "Sphere" représentant une sphère, ainsi qu'une structure "Cube" représentant un cube. Une sphère est définie par un centre, un rayon et une couleur. Un cube est définit par un point minimum, un point maximum (les deux points de sa diagonale) et une couleur.**
4. **Ajoutez les fonctions `createRay(...)`, `createCube(...)` et `createSphere(...)` (trouvez les signatures par vous même) afin de faciliter la création de ces objets.**

Exercice 02 : Implémentation des fonctions d'intersection

Il est maintenant temps d'implémenter et de tester nos équations d'intersection.

1. **Dans le fichier include/raytracer.h, ajoutez les deux signatures de fonction suivantes :**

```
int intersectsSphere(Ray r, Sphere s, Intersection*);
```

Cette fonction devra tester si le rayon intersecte la sphère.
Si c'est le cas, elle devra remplir les champs de l'intersection dont le pointeur sera passé en paramètre et retourner 1. S'il n'y a pas d'intersection, la fonction devra retourner 0.

```
int intersectsCube(Ray r, Cube s, Intersection*);
```

Cette fonction devra tester si le rayon intersecte le cube.
Si c'est le cas, elle devra remplir les champs de l'intersection dont le pointeur sera passé en paramètre et retourner 1. S'il n'y a pas d'intersection, la fonction devra retourner 0.

2. **Créez un fichier src/raytracer.c dans lequel vous implémenterez ces deux fonctions à partir des équations d'intersection trouvées à la fin du TD01.**
3. **Dans votre fichier main.c, testez vos fonctions d'intersection en créant plusieurs Spheres, Cubes, Rayons avec des paramètres différents, en en appelant vos fonctions d'intersection avec ces différents objets.**

Exercice 03 : Structure de notre scène

On appelle scène l'ensemble des objets que l'on désire potentiellement voir apparaître à l'écran (en fonction de la position de la caméra).

1. **Dans le fichier *raytracer.h*, créez une structure Scene, qui contiendra un tableau (de taille fixe pour le moment) de Spheres et un tableau de Cubes.**

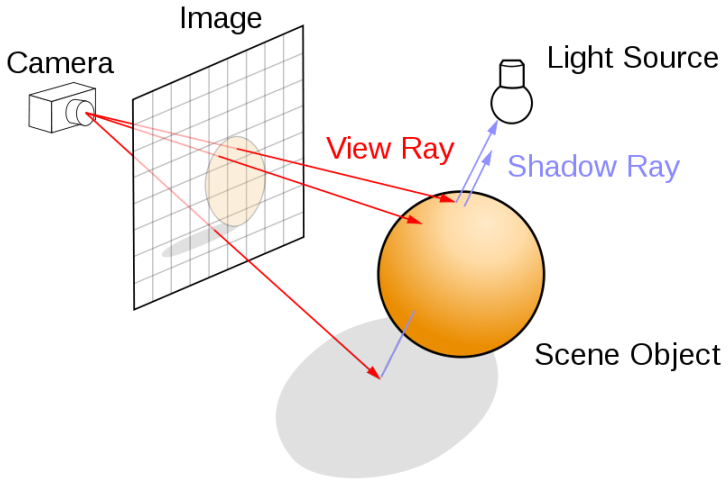
2. **Ajoutez la fonction `Scene createScene()`. Cette fonction doit créer et initialiser les différents champs de la structure.**

3. **Ajoutez les fonctions `void addCubeToScene(Scene* scene, Cube cube)` et `void addSphereToScene(Scene* scene, Sphere sphere)`. Vous devrez définir, comme toujours, la signature des fonctions dans le header, et l'implémentation des fonctions dans le fichier .c correspondant.**

4. **Ecrivez la fonction `int throwRayOnScene(Ray ray, const Scene* scene, Intersection* intersection)` qui boucle sur tous les objets d'une scène afin de trouver l'intersection la plus proche avec le rayon passé en paramètre (la fonction renvoie 1 s'il y a une intersection, sinon 0).**

Exercice 04 : Notre premier raytracing !

Nous avons maintenant toutes les briques en place pour effectuer notre premier raytracing ! Mais comment définir la position et la direction des rayons que nous allons lancer ? Rappelez vous cette image :



Il faut considérer que l'image finale affichée à l'écran est représentée par un Quad placé dans la scène juste devant la caméra, et que la caméra va lancer des rayons à travers chaque pixel de l'image et détecter les intersections avec les objets de la scène.

Pour ce premier raytracer, nous utiliserons les valeurs suivantes :

- Caméra positionnée en (0, 0, 0) et regardant dans la direction (0, 0, -1) (axe négatif des Z).
- Image positionné sur le plan Z = -1, dans un carré de taille 2 centré en (0, 0, -1). Cela implique que les coordonnées des coins du carré seront :
  - (−1, −1, −1)
  - (1, −1, −1)
  - (1, 1, −1)
  - (−1, 1, −1)

Notre **framebuffer** (la tableau qui contient les couleurs de nos pixels) sera donc représenté virtuellement par ce quad. Chaque pixel possède donc une position 3D située dans le quad virtuel, et les rayons seront lancés à travers ces pixels.

Pour ce projet, nous utiliserons la SDL pour stocker et afficher une image à l'écran. Notre framebuffer sera représenté par un objet de type **SDL_Surface***.

Pour chaque pixel (i, j) de notre framebuffer (de notre image), il sera donc facile d'obtenir :

- Sa position x dans le repère de la scène : x = -1 + 2 * (i / imageWidth)
- Sa position y dans le repère de la scène : y = 1 - 2 * (j / imageHeight)
- Sa position z dans le repère de la scène : z = -1.
- Le rayon qui le traverse, ayant pour origine (0, 0, 0) (la position de la caméra) et passant par le point (x, y, z). Notre caméra étant centrée sur l'origine, (x, y, z) est donc aussi le vecteur directeur du rayon.

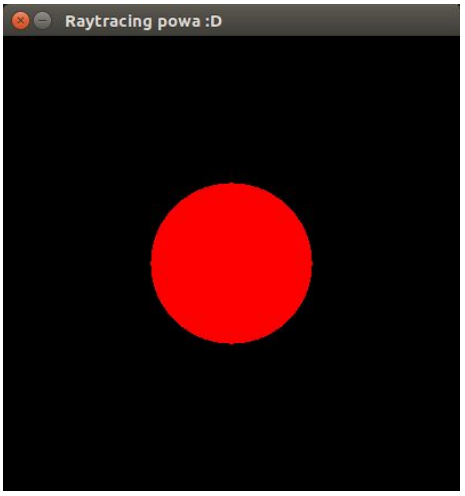
1. **Copiez les fichier *sdl_tools.h* et *sdl_tools.c* dans votre projet. Ils fournissent une fonction `PutPixel()` qui remplira un objet SDL_Surface* avec une couleur donnée à une position de pixel donnée.**

2. **Créez une fonction `void simpleRaytracing(const Scene* scene, SDL_Surface* framebuffer)`.**

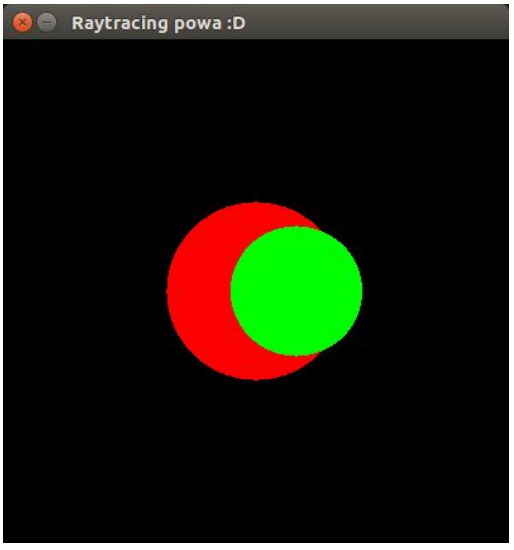
   Cette fonction itère sur chacun des pixels du framebuffer (les dimensions du framebuffer sont stockés dans framebuffer->w et framebuffer->h). Pour chaque pixel (i, j) elle calcule le rayon R(i, j) selon la formule donnée précédemment puis envoie ce rayon dans la scène. Si une intersection est trouvée elle place la couleur du point d'intersection à la position (i, j) du framebuffer (avec la fonction PutPixel de sdl_tools.h). Sinon elle ne touche pas au pixel. Vous placerez cette fonction dans les fichiers raytracer.h et raytracer.c.

3. **Remplacez le contenu de votre fichier par le contenu du fichier main_raytracer.c. Ce nouveau template initialise la SDL et créé une fenêtre, un framebuffer, et se charge de l'afficher dans un mini boucle de rendu.**

4. **Ajoutez à votre scène une sphère à la position (0, 0, -3) de rayon 1 et de couleur rouge. Appelez la fonction `simpleRaytracing` sur votre scène, et lancez votre programme. Vous devrez voir un cercle rouge s'afficher à l'écran :**



5. **Ajoutez une deuxième sphère à votre scène, à la position (0, 0, -2), de rayon 0.5 et de couleur verte. Vous devrez voir la deuxième sphère s'afficher juste devant la première :**



6. **Inversez les deux lignes d'ajout de sphères, et constatez que le résultat est parfaitement identique, prouvant que les intersections détectées sont bien les plus proches de la caméra.**
7. **Ajoutez d'avantage de sphères ainsi que des cubes à votre scène et vérifiez que tout s'affiche correctement.**

Félicitations, vous venez de développer votre tout premier raytracer. Ce dernier est encore très basique (la structure de la scène est très contraignant, il n'y a pas de lumière, etc.) mais la suite des TDs nous montrera comment l'améliorer.

Bientôt … ? :)