Lancer de Rayons orienté objet
IN204

Yu WANG

08/03/2021

Introduction

- Le lancer de rayons est une technique utilisée pour rendre des images sur un ordinateur.
- Il comprend des rayons lumineux simulés qui obligent la caméra à quitter l'objet dans la scène, puis à pénétrer dans la source de lumière à partir de l'objet. En calculant l'intersection de la lumière et de l'objet, nous pouvons obtenir la couleur de chaque pixel de l'image à créer. Le lancer de rayons est largement utilisé dans la génération d'images de jeux vidéo et la recherche en physique optique.

L'algorithme principal

Algorithme 1 Ray tracing

Input $std :: vector < Object* >$, $std :: vector < Source* >$
Output: $pixels[Width * Height]$

```
1:  for each pixel of the image do
2:      ray ← GENERATERAY(Camera, xPostion, yPosition)
3:      for i = 0 to number of objects do
4:          intersections[i] ← FINDINTERSECTIONS(ray, objects)
5:      end for
6:      indexClosest ← FINDINDEXCLOSEST(intersections)
7:      if indexClosest == −1 then
8:          pixel ← BLACK
9:      else
10:         pixel ← GETCOLOR(indexClosest, ray, intersections, objects, sources)
11:     end if
12: end for
```

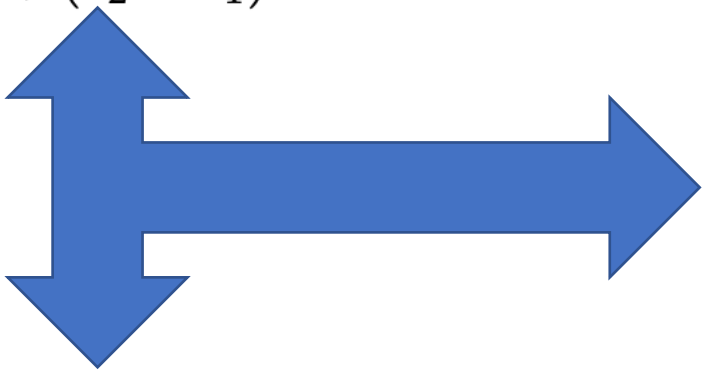
L'algorithme principal

Rayon

$$x = x_1 + (x_2 - x_1)t$$

$$y = y_1 + (y_2 - y_1)t$$

$$z = z_1 + (z_2 - z_1)t$$



Sphère

$$(x - x_c)^2 + (y - y_c)^2 + (z - z_c)^2 = r^2$$

$$at^2 + bt + c = 0$$

$$a = (x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2$$

$$b = -2[(x_2 - x_1)(x_c - x_1) + (y_2 - y_1)(y_c - y_1) + (z_c - z_1)(z_2 - z_1)]$$

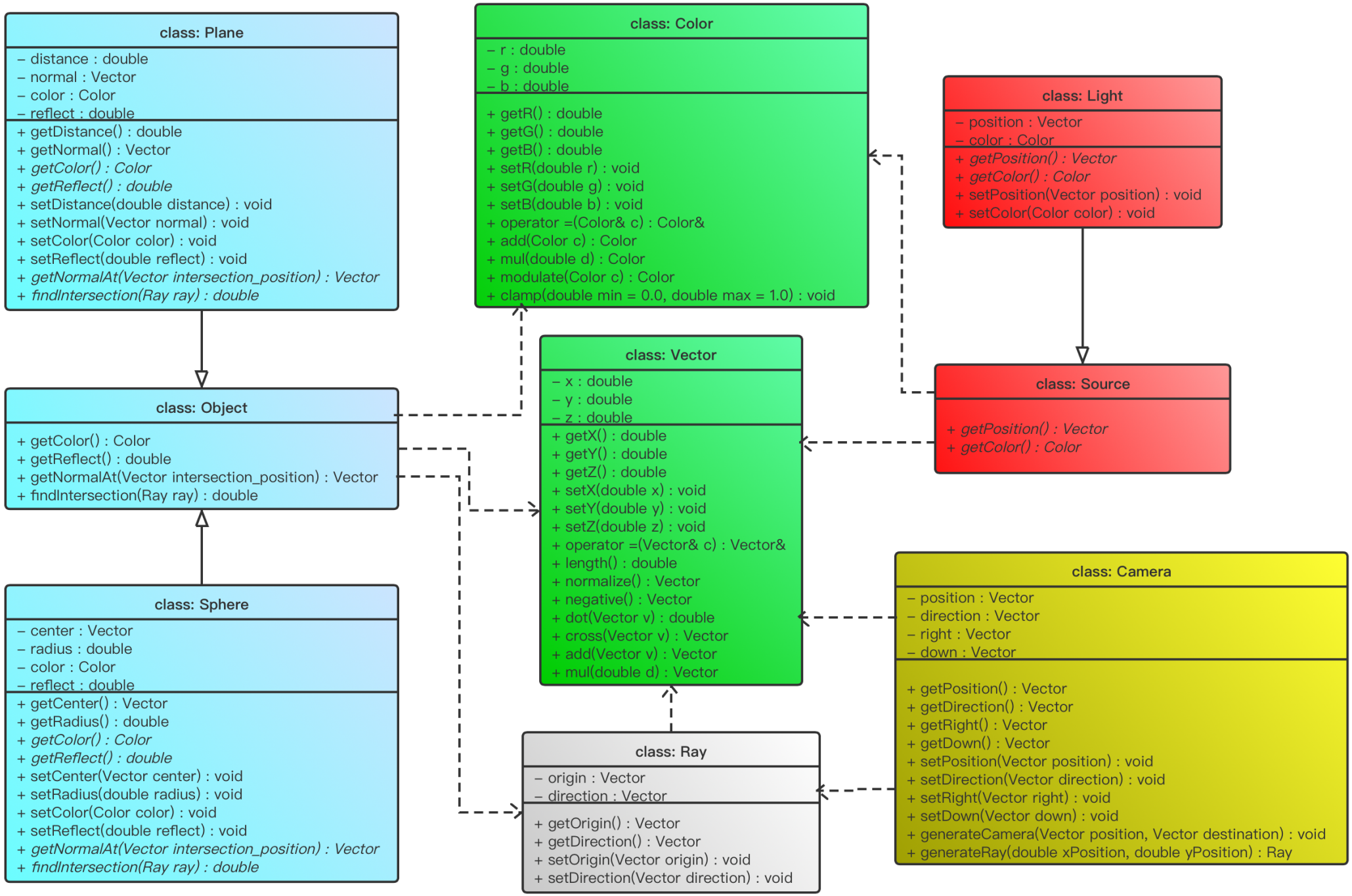
$$c = (x_c - x_1)^2 + (y_c - y_1)^2 + (z_c - z_1)^2 - r^2$$

L'algorithme principal

Algorithme 2 getColor

Input $indexClosest, ray, intersections, objects, sources$
Output: $finalColor$

```
1:  objectColor ← GETCOLOR(objects[indexClosest])
2:  finalColor ← 0.2 * objectColor
3:  for i = 0 to number of sourcea do
4:      if shadow == false then
5:          sourceColor ← GETCOLOR(sources[i])
6:          finalColor ← finalColor+ MODULATE(objectColor, sourceColor)
7:          reflect ← GETREFLECT(objects[indexClosest])
8:          if 0 < reflect <= 1 then
9:              specular ← DOT(reflectionDirection, lightDirection)
10:             if specular > 0 then
11:                 specular ← POW(specular, 10)
12:                 finalColor ← finalColor+ MUL(sourceColor, specular * reflect)
13:             end if
14:         end if
15:     end if
16: end for
```


class: Plane
– distance : double
– normal : Vector
– color : Color
– reflect : double
+ getDistance() : double
+ getNormal() : Vector
+ getColor() : Color
+ getReflect() : double
+ setDistance(double distance) : void
+ setNormal(Vector normal) : void
+ setColor(Color color) : void
+ setReflect(double reflect) : void
+ getNormalAt(Vector intersection_position) : Vector
+ findIntersection(Ray ray) : double
class: Color
– r : double
– g : double
– b : double
+ getR() : double
+ getG() : double
+ getB() : double
+ setR(double r) : void
+ setG(double g) : void
+ setB(double b) : void
+ operator =(Color& c) : Color&
+ add(Color c) : Color
+ mul(double d) : Color
+ modulate(Color c) : Color
+ clamp(double min = 0.0, double max = 1.0) : void
class: Light
– position : Vector
– color : Color
+ getPosition() : Vector
+ getColor() : Color
+ setPosition(Vector position) : void
+ setColor(Color color) : void
class: Object
+ getColor() : Color
+ getReflect() : double
+ getNormalAt(Vector intersection_position) : Vector
+ findIntersection(Ray ray) : double
class: Vector
– x : double
– y : double
– z : double
+ getX() : double
+ getY() : double
+ getZ() : double
+ setX(double x) : void
+ setY(double y) : void
+ setZ(double z) : void
+ operator =(Vector& c) : Vector&
+ length() : double
+ normalize() : Vector
+ negative() : Vector
+ dot(Vector v) : double
+ cross(Vector v) : Vector
+ add(Vector v) : Vector
+ mul(double d) : Vector
class: Source
+ getPosition() : Vector
+ getColor() : Color
class: Sphere
– center : Vector
– radius : double
– color : Color
– reflect : double
+ getCenter() : Vector
+ getRadius() : double
+ getColor() : Color
+ getReflect() : double
+ setCenter(Vector center) : void
+ setRadius(double radius) : void
+ setColor(Color color) : void
+ setReflect(double reflect) : void
+ getNormalAt(Vector intersection_position) : Vector
+ findIntersection(Ray ray) : double
class: Ray
– origin : Vector
– direction : Vector
+ getOrigin() : Vector
+ getDirection() : Vector
+ setOrigin(Vector origin) : void
+ setDirection(Vector direction) : void
class: Camera
– position : Vector
– direction : Vector
– right : Vector
– down : Vector
+ getPosition() : Vector
+ getDirection() : Vector
+ getRight() : Vector
+ getDown() : Vector
+ setPosition(Vector position) : void
+ setDirection(Vector direction) : void
+ setRight(Vector right) : void
+ setDown(Vector down) : void
+ generateCamera(Vector position, Vector destination) : void
+ generateRay(double xPosition, double yPosition) : Ray

Parallélisation

Nombre de processus	Temps de calcul des pixels (s)	Speed Up
1	23.0059	1
2	13.3755	1.72
4	7.3524	3.13
8	6.2431	3.67

Anti aliasing

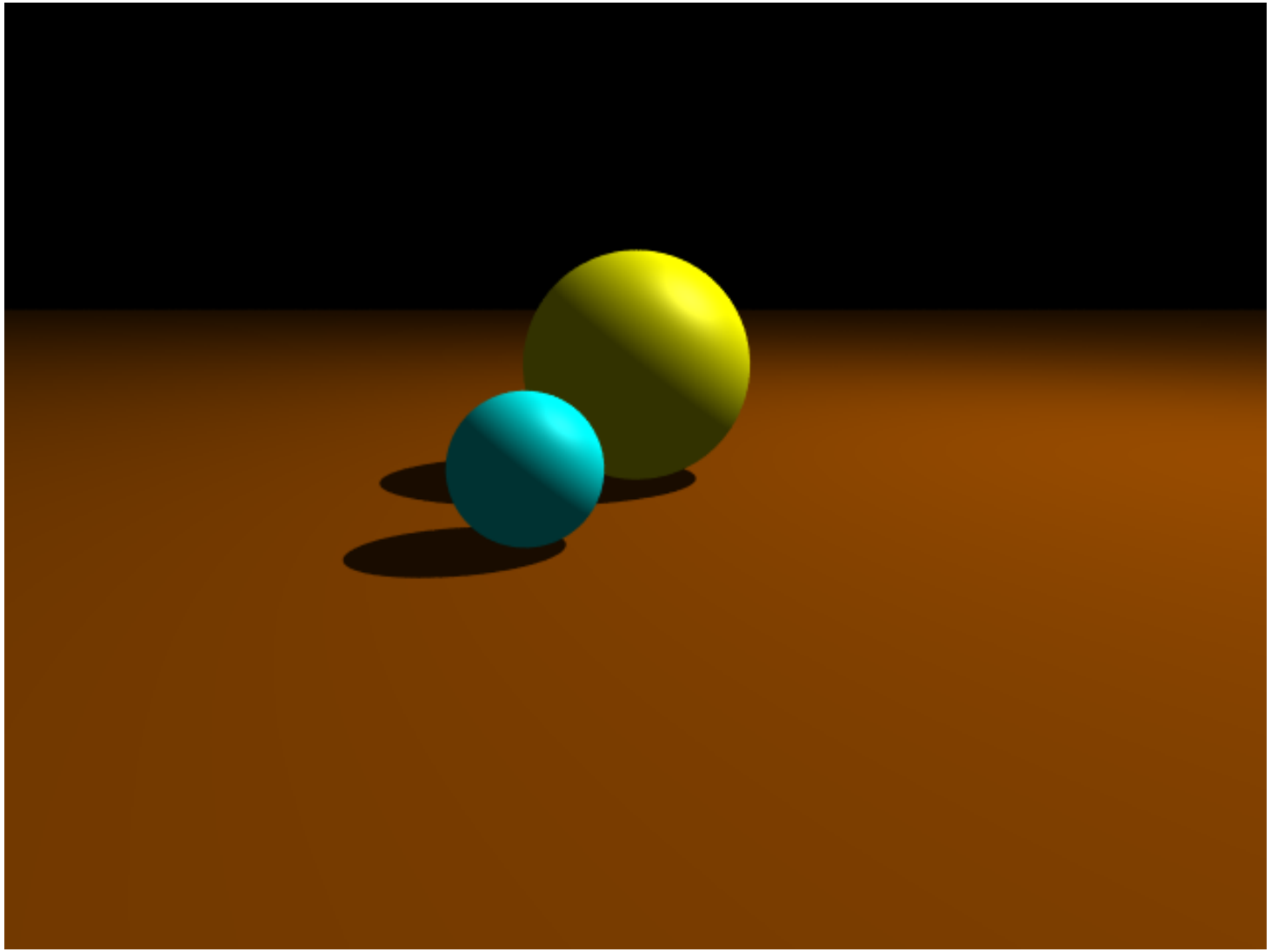
```
double xPosition = (double)(x - 0.5*Height)/(double)sample ;
double yPosition = (double)(y - 0.5*Width)/(double)sample;
```

```
const int kNumPixelSamples = 64;

double xPosition = (double)(x - 0.5*Height + rand()/double(RAND_MAX) -
    0.5)/(double)sample ;
double yPosition = (double)(y - 0.5*Width + rand()/double(RAND_MAX) -
    0.5)/(double)sample;

pixels[x*Width+y] = pixels[x*Width+y].mul(1/(double)kNumPixelSamples);
```

Résultats



Conclusion et discussion

- La principale difficulté rencontrée est liée à l'algèbre, c'est-à-dire à la définition du système de coordonnées et aux calculs entre différents vecteurs.

- On constate que le projet a encore beaucoup à faire.
  - La classe d'objet et la classe de source lumineuse prennent en charge l'héritage pour créer des scènes plus complexes.
  - Afin de rendre le programme plus facile à utiliser pour les utilisateurs, il peut être intéressant de mettre en œuvre une interface dans laquelle nous pouvons définir tous les composants de la scène à créer, ce qui permettra de définir la scène de manière plus simple.
  - Nous pouvons calculer la réflexion secondaire plus précisément pour obtenir une vue plus réaliste de la scène

Références

- [1] Monsuez, B. : notes de cours, Programmation Orientée Objet. (2020)
- [2] Juvigny, X. : notes de cours, Programmation Parallèle. (2020)
- [3] *Ray tracing* : https://fr.wikipedia.org/wiki/Ray_tracing
- [4] *Rayito* : https://github.com/Tecla/Rayito
- [5] *Introduction to Ray Tracing: a Simple Method for Creating 3D Images* : https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-ray-tracing