



IN204

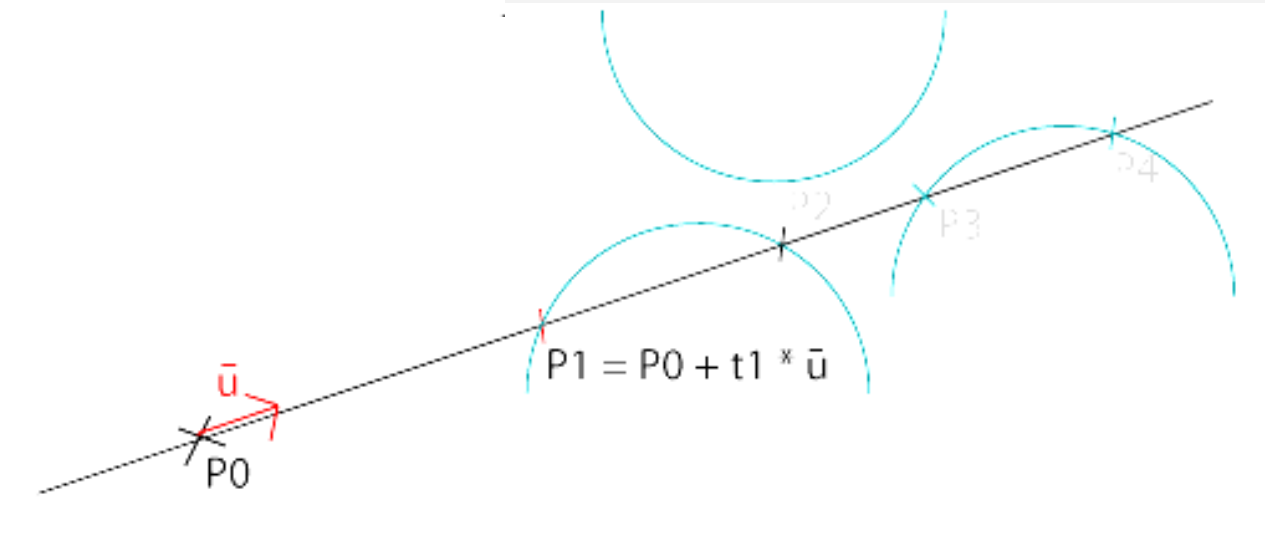
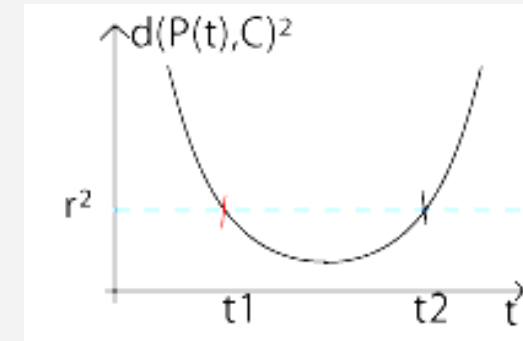
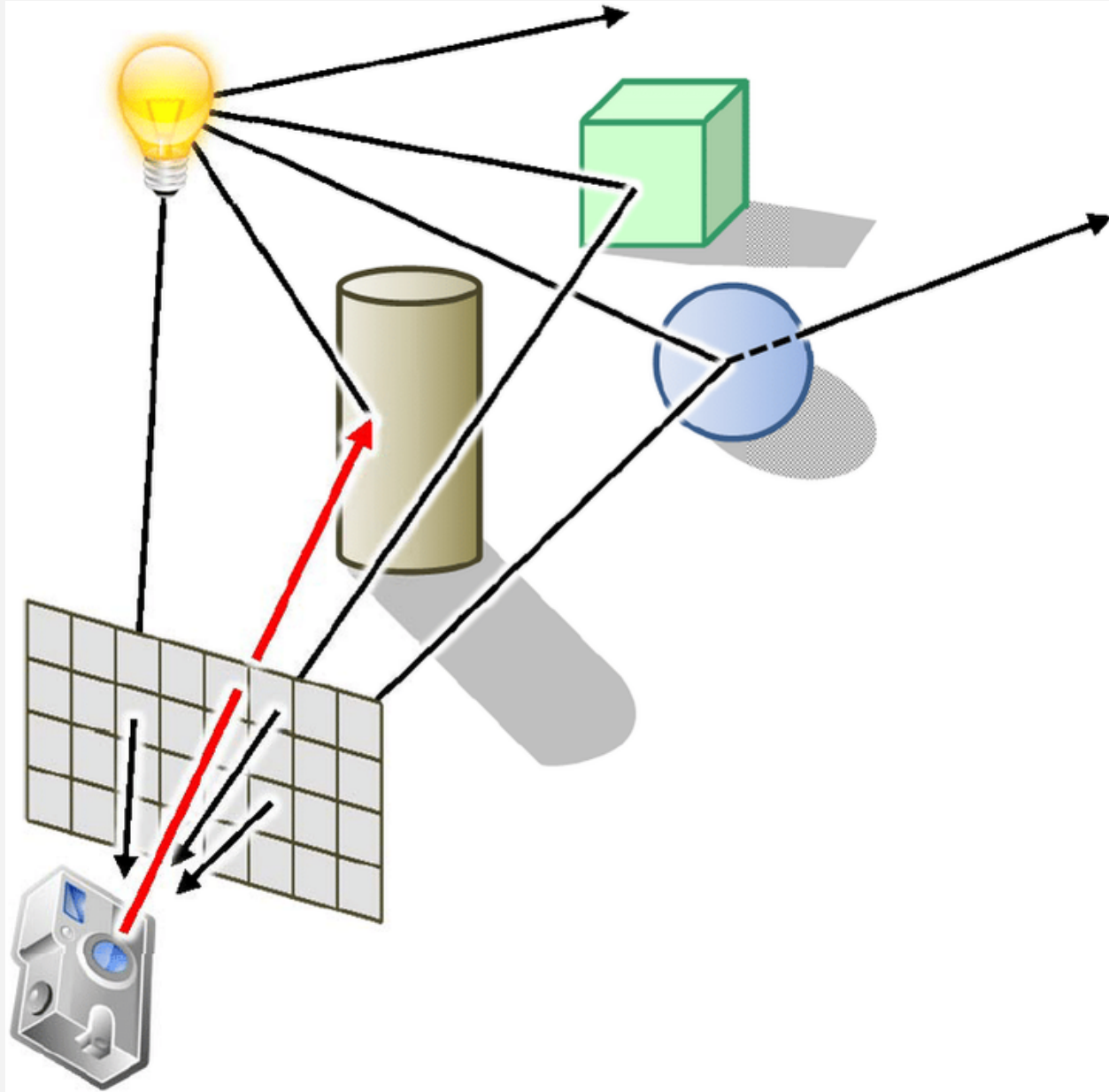
Ray Tracing

Axel DUMONT/Valentin QUONIAM-BARRE

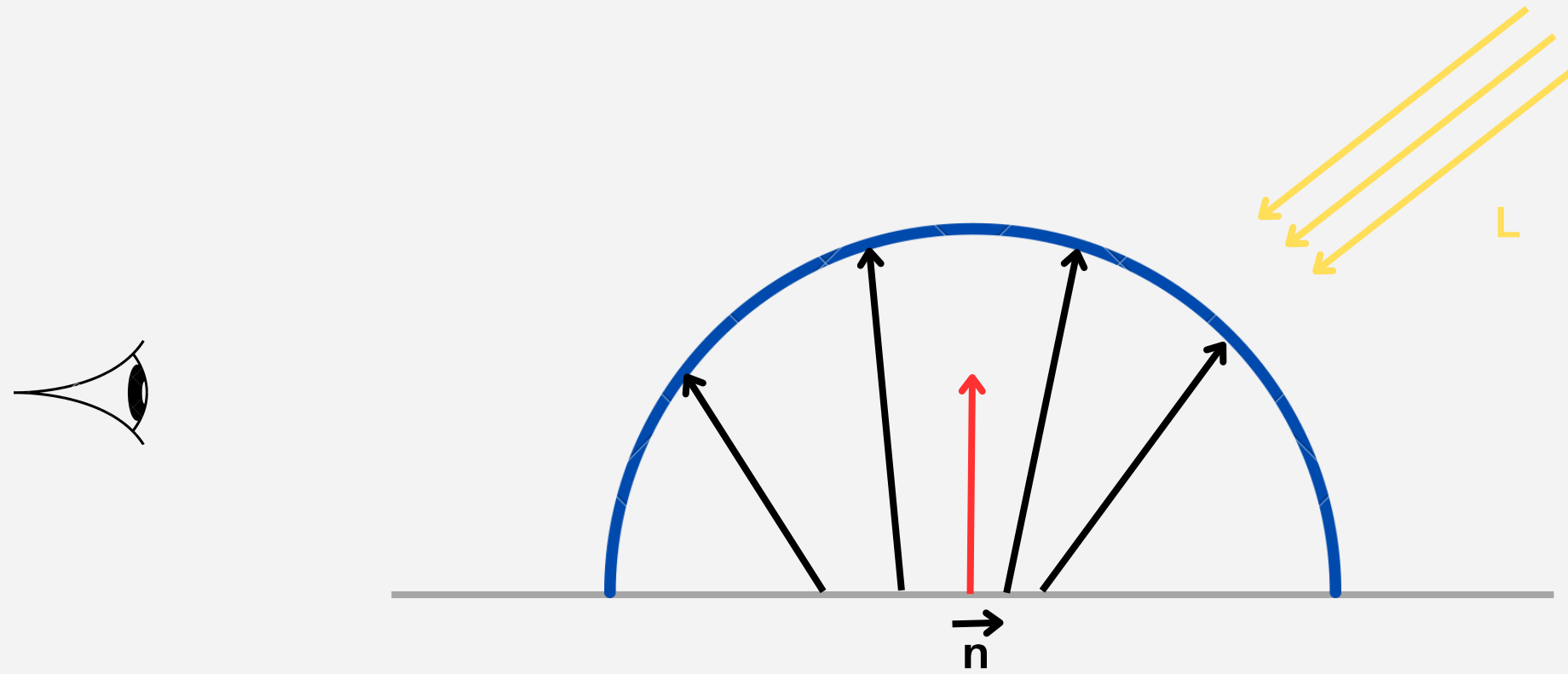
Qu'est ce que le Ray Tracing



Comment le Ray Tracing fonctionne



Comment le Ray Tracing fonctionne



Construction d'une scène

scene2.txt

1000 800	—————→	Dimensions de l'images
2 2 3	—————→	Nombre de matériaux, nombre d'objets, nombre de lumières
0.8 0.6 0.0 0.7 1.0 1.0 1.0 50	—————→	Description du 1er matériau (réflexion, etc)
0.0 1.0 1.0 0.7 1.0 1.0 1.0 60		
350.0 500.0 0.0 150 0 sphere 0 0 0	—————→	Description du 1er objet
650.0 400.0 0.0 90 1 sphere 0 0 0		
1000.0 100.0 -10000.0 0.6 0.7 0.2	—————→	Description de la 1ère lumière
0.0 100.0 -10000.0 0.6 0.7 0.2		
300.0 200.0 -100.0 1.0 0.0 0.0		



Implémentation

Dossier cpp

```
bool init(char* inputName, scene &myScene)
```

main.cpp



Ouvre la scène (scene.txt) et utilise raytrace.cpp

```
bool draw(char *outputName, scene &myScene)
```

raytracer.cpp



Fais le raytracing et stocke dans un fichier .tga

(test.cpp)



Test utilisé dans la création de nouveaux objets






Implémentation

Dossier hpp

conversion_tga_png	—————→	Deux fichiers hpp pour la conversion des images
objects.hpp	—————→	Définit un objet par son centre et une taille et ajoute plusieurs fonctions pour savoir si un rayon rencontre un objet
structs.hpp	—————→	Définit les structures 3D, ainsi que les couleurs, le matériau, la lumière
header_tga.hpp	—————→	Initialise le fichier tga de sortie



Implémentation (quelques fonctions, struct et class)

```
struct object {
    point pos;
    float size;
    int material;
    string type;
    float angle_rot_x;
    float angle_rot_y;
    float angle_rot_z;
};
```

```
struct light {
    point pos;
    color couleur;
};
```

```
struct material {
    color diffuse;
    float reflection;
    color specular;
    float power;
};
```

```
bool hitSphere(const ray &r, const object &s, float &t)
```

```
struct color {
    float red, green, blue;
};
```

```
// Fonction pour tourner un point dans le repère du cube
point transformPoint(const point &p, const object& c){

    float angle_x = c.angle_rot_x * M_PI /180;    // Conversion de l'angle en radians
    float angle_y = c.angle_rot_y * M_PI /180;
    float angle_z = c.angle_rot_z * M_PI /180;

    point q;
    // Matrice de rotation en X
    float matrix_x[3][3] = {
        {1, 0, 0},
        {0, cos(angle_x), -sin(angle_x)},
        {0, sin(angle_x), cos(angle_x)}
    };

    // Matrice de rotation en Y
    float matrix_y[3][3] = {
        {cos(angle_y), 0, sin(angle_y)},
        {0, 1, 0},
        {-sin(angle_y), 0, cos(angle_y)}
    };

    // Matrice de rotation en Z
    float matrix_z[3][3] = {
        {cos(angle_z), -sin(angle_z), 0},
        {sin(angle_z), cos(angle_z), 0},
        {0, 0, 1 }
    };

    // Soustraire le centre pour la rotation par rapport au centre
    float x = p.x - c.pos.x;
    float y = p.y - c.pos.y;
```

```
struct ray {
    light start;
    vecteur dir;
};
```

```
float z = p.z - c.pos.z;

    // Appliquer la rotation en X
    float x1 = matrix_x[0][0] * x + matrix_x[0][1] * y + matrix_x[0][2] * z;
    float y1 = matrix_x[1][0] * x + matrix_x[1][1] * y + matrix_x[1][2] * z;
    float z1 = matrix_x[2][0] * x + matrix_x[2][1] * y + matrix_x[2][2] * z;

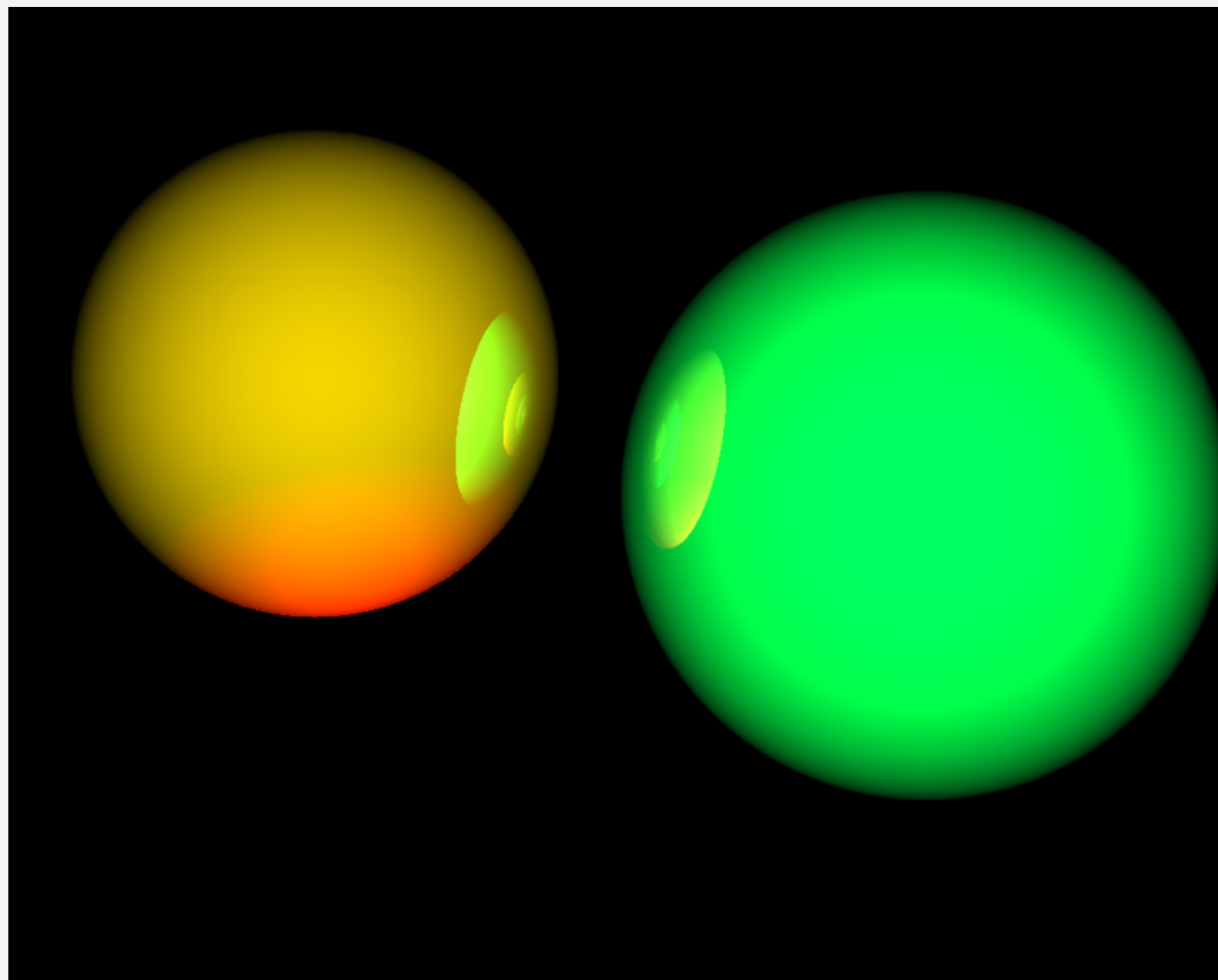
    // Appliquer la rotation en Y
    float x2 = matrix_y[0][0] * x1 + matrix_y[0][1] * y1 + matrix_y[0][2] * z1;
    float y2 = matrix_y[1][0] * x1 + matrix_y[1][1] * y1 + matrix_y[1][2] * z1;
    float z2 = matrix_y[2][0] * x1 + matrix_y[2][1] * y1 + matrix_y[2][2] * z1;

    // Appliquer la rotation en Z
    float x3 = matrix_z[0][0] * x2 + matrix_z[0][1] * y2 + matrix_z[0][2] * z2;
    float y3 = matrix_z[1][0] * x2 + matrix_z[1][1] * y2 + matrix_z[1][2] * z2;
    float z3 = matrix_z[2][0] * x2 + matrix_z[2][1] * y2 + matrix_z[2][2] * z2;

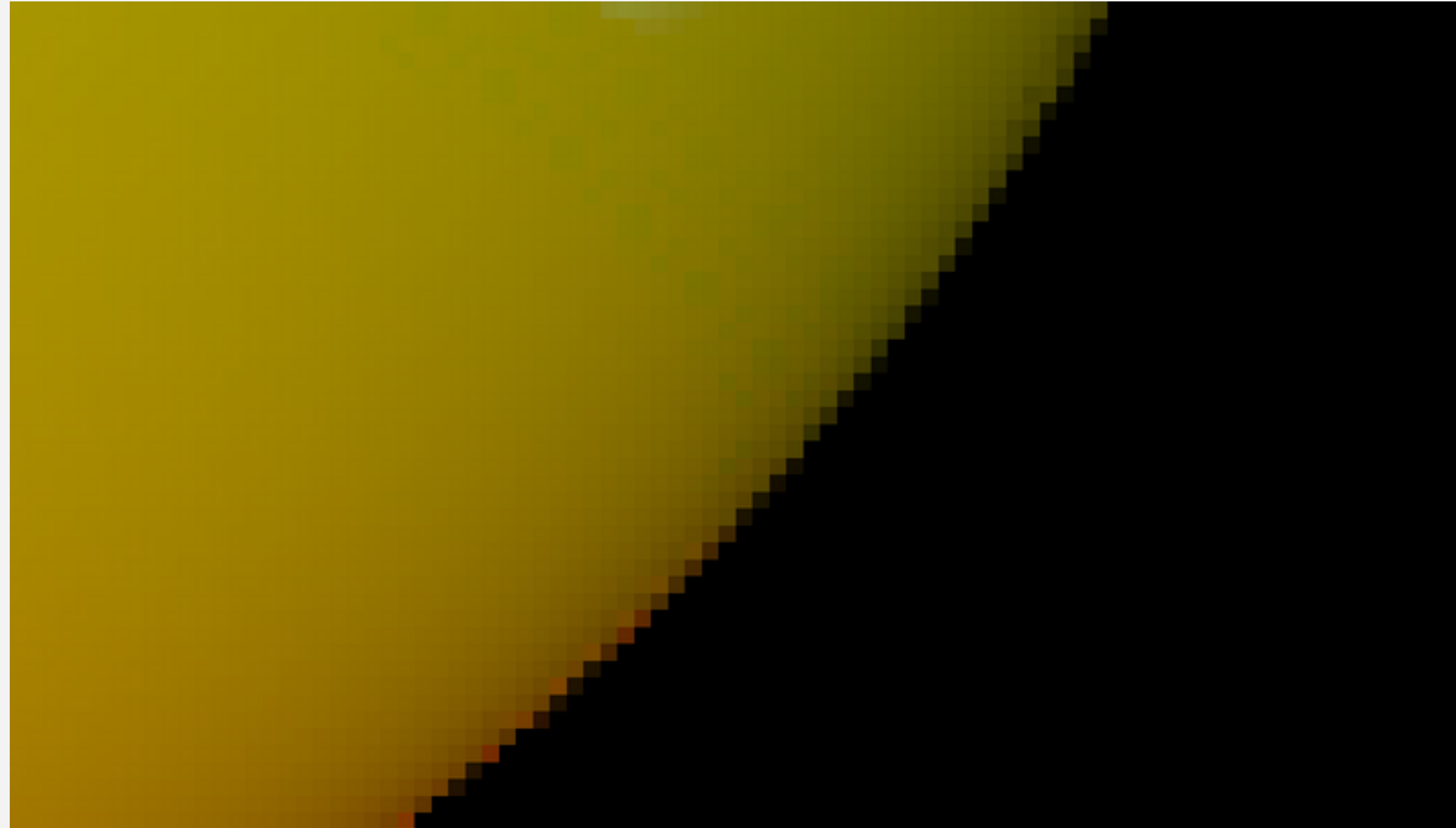
    // Ajouter le centre du cube pour recentrer le sommet
    q.x = x3 + c.pos.x;
    q.y = y3 + c.pos.y;
    q.z = z3 + c.pos.z;

    return q;
}
```

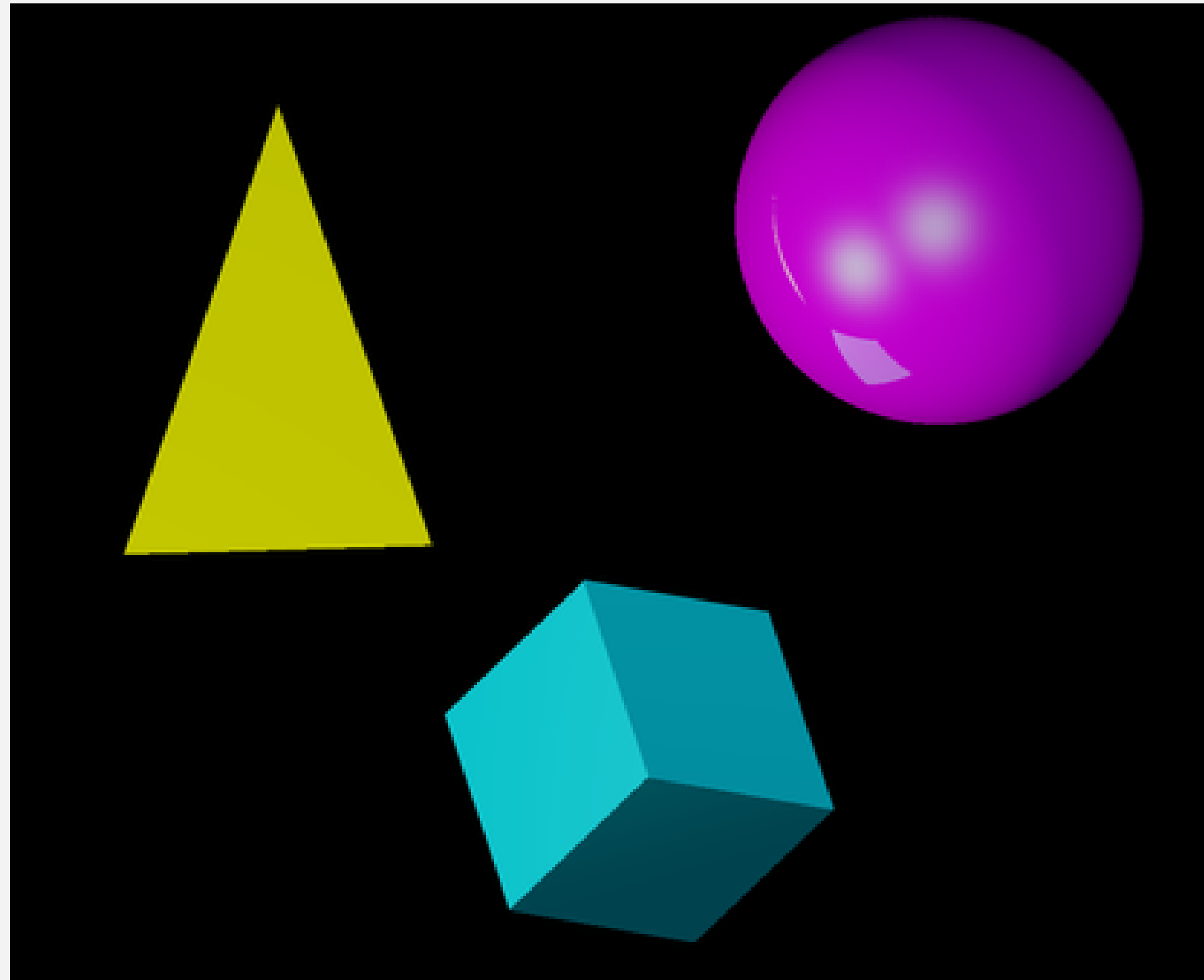

Premiers résultats



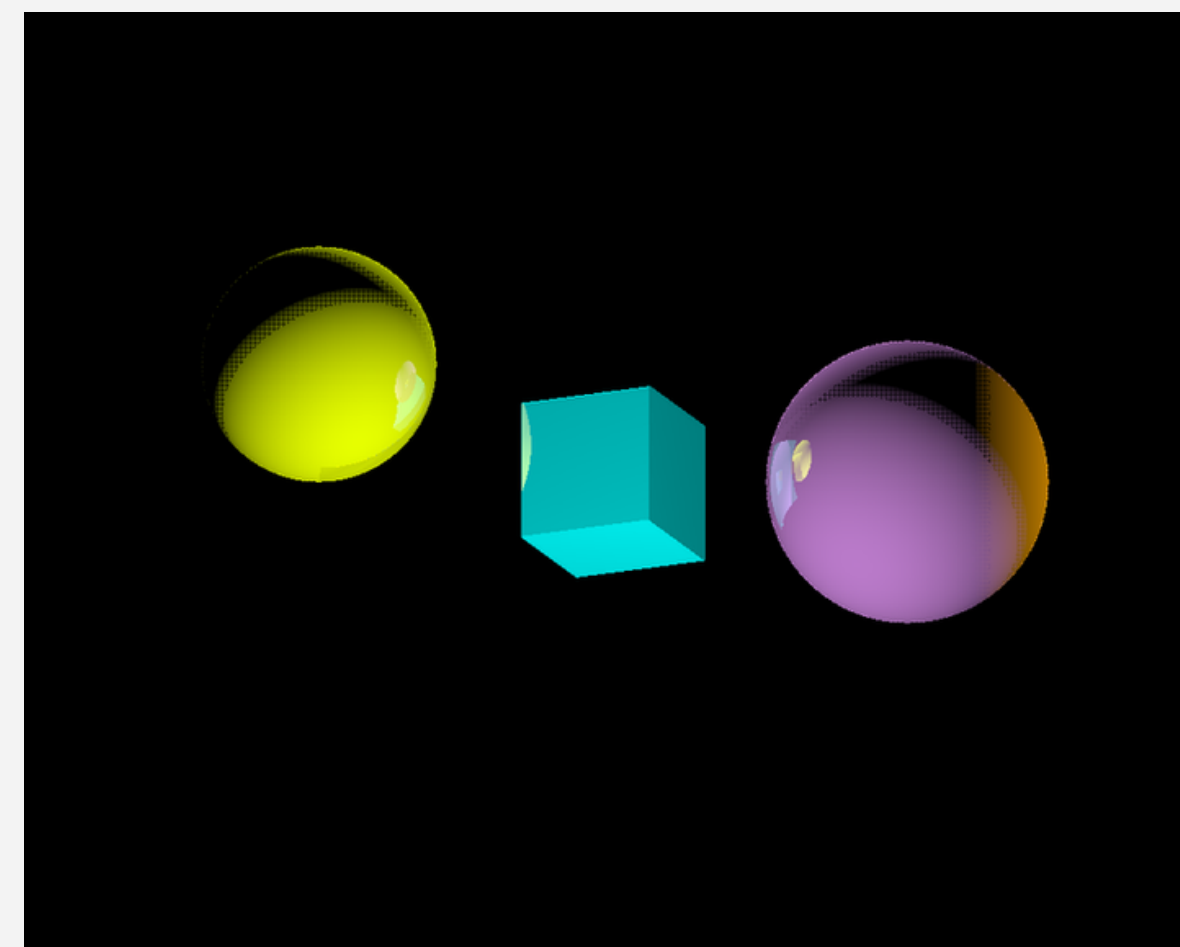
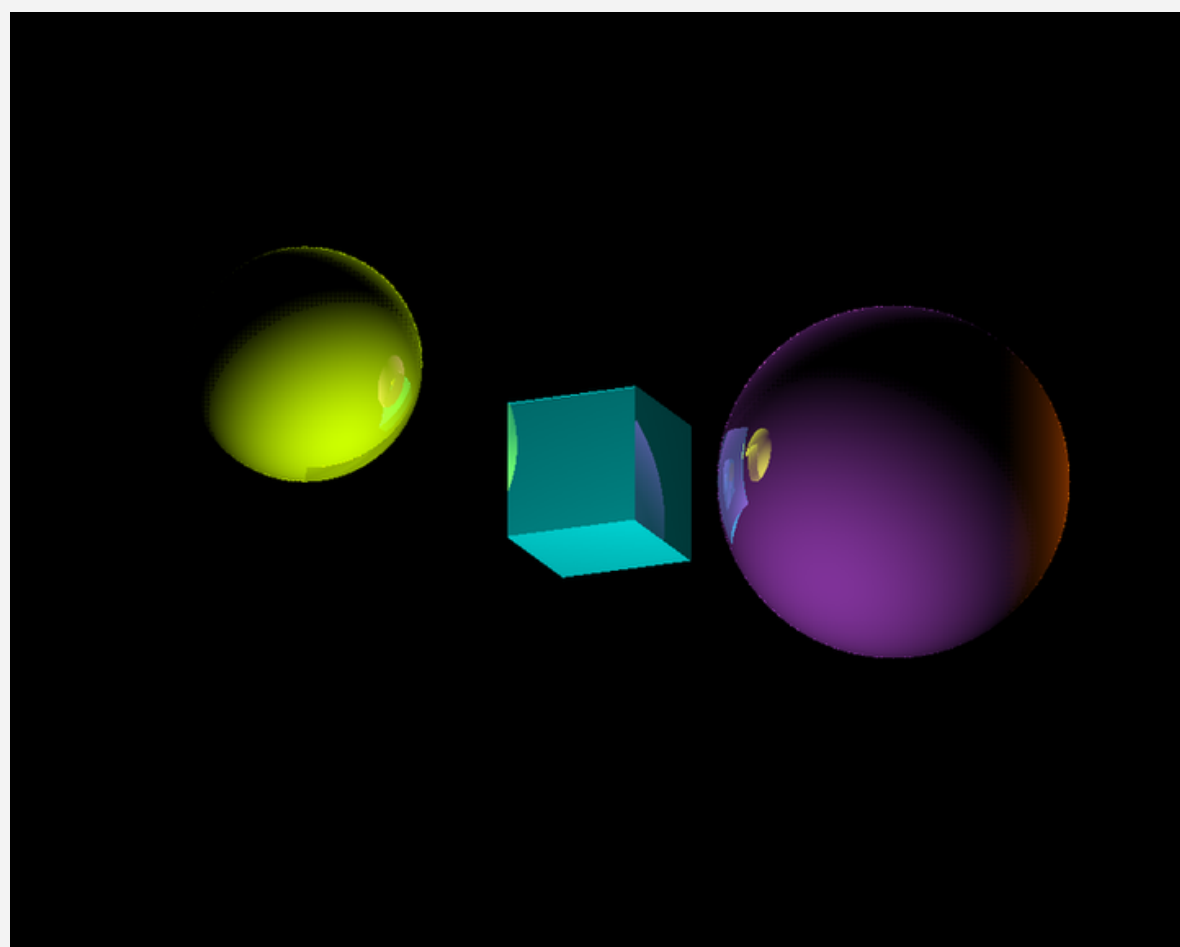
Ajout de l'antialiasing



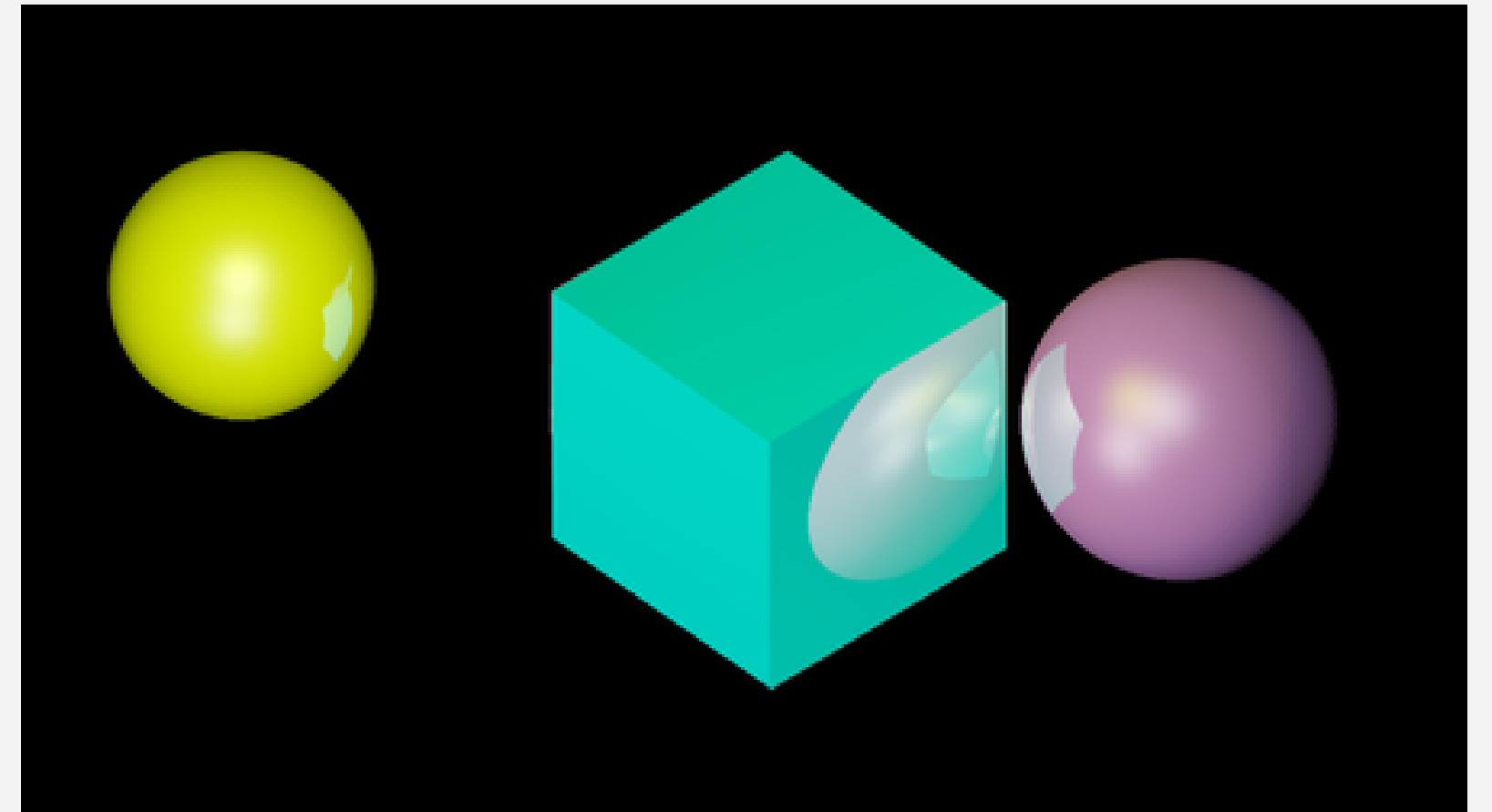
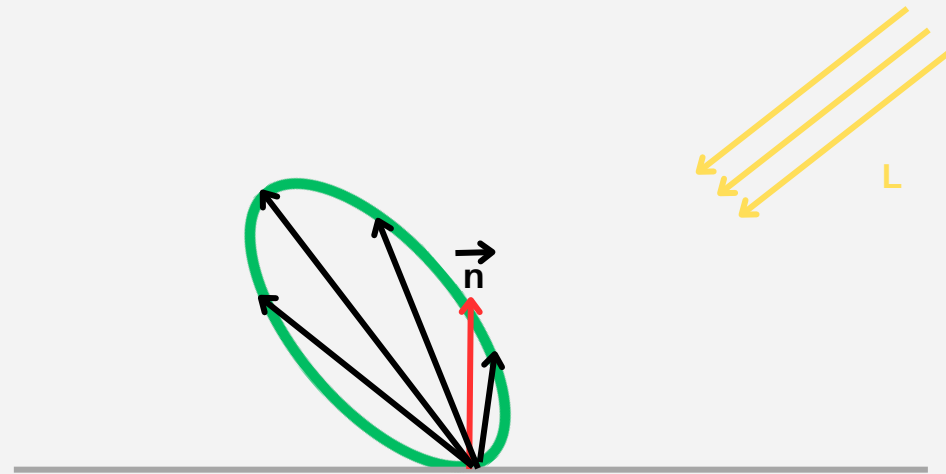
Ajout de formes différentes



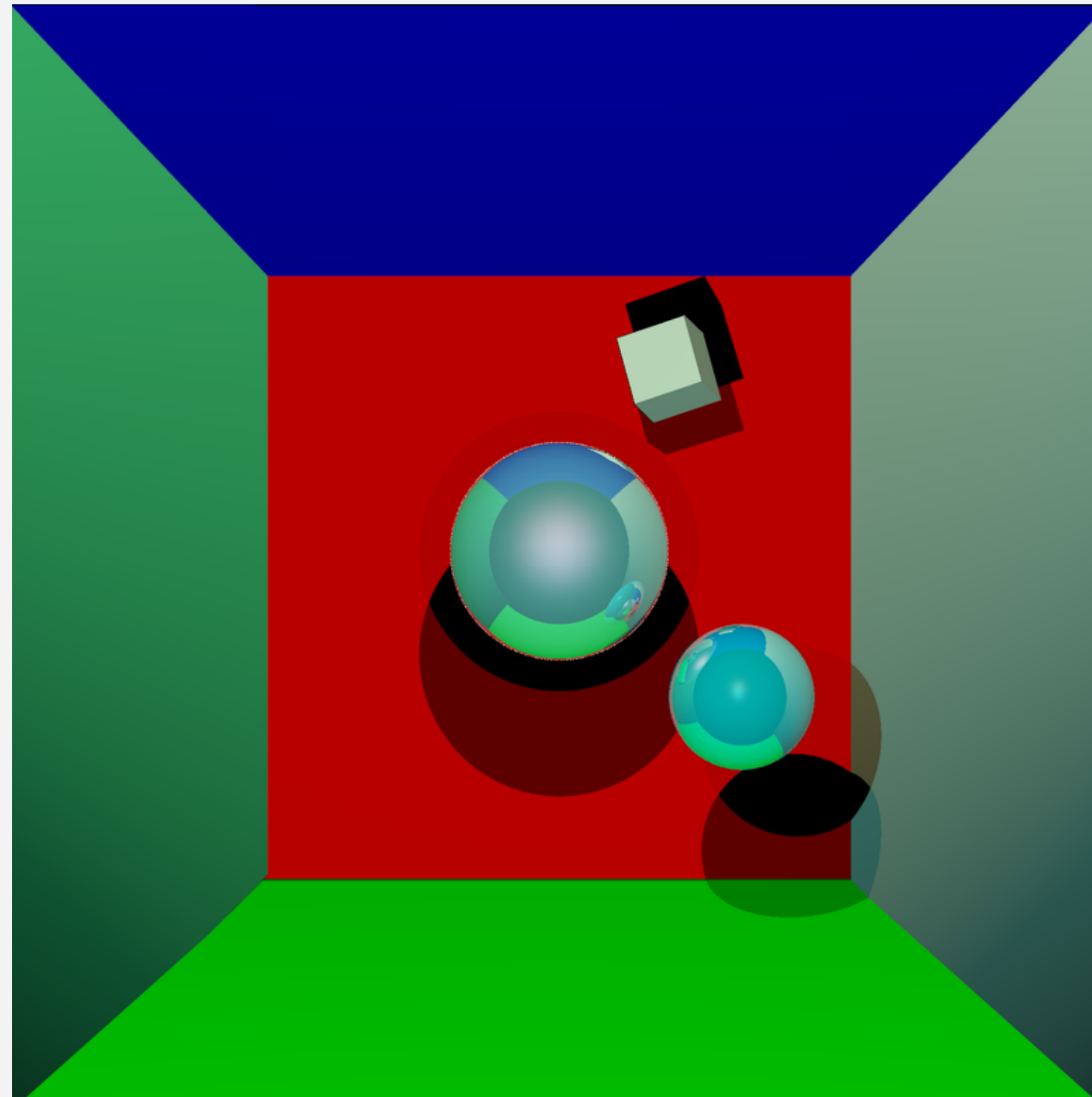
Modification de l'exposition



Eclairage Blinn-Phong




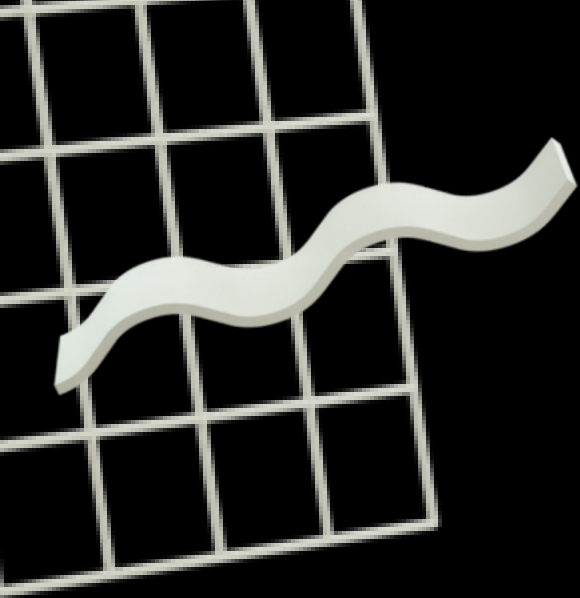
Résultat final



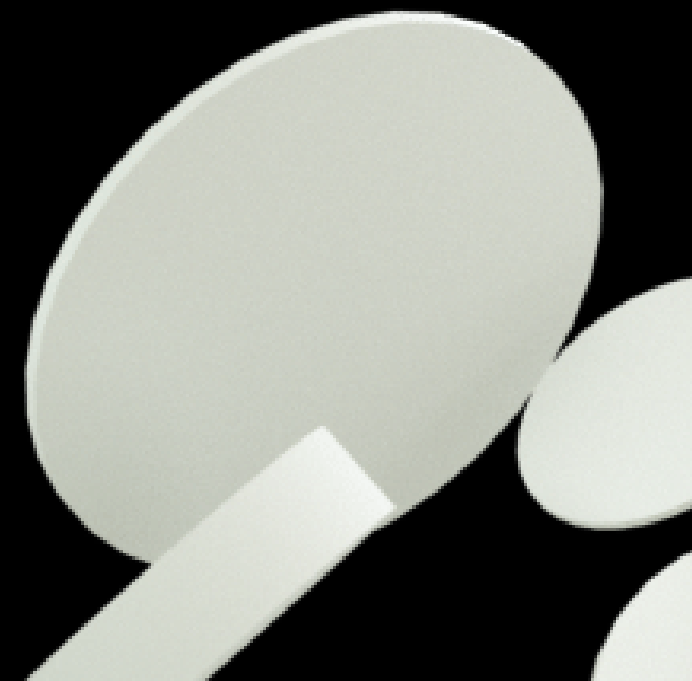


Pistes d'amélioration

- Changement de la classe objet pour des formes quelconques
 - Optimisation du temps par la parallélisation
 - Interface pour générer des scènes
- 



**Merci pour votre
écoute**





Bibliographie

- <http://www.massal.net/article/raytrace/page1.html>
 - *Ray Tracing in One Weekend*, Peter Shirley
 - Ray Tracing in One Weekend C++ Tutorial on YouTube,
<https://www.youtube.com/watch?v=nQ3TRft18Qw>
- 