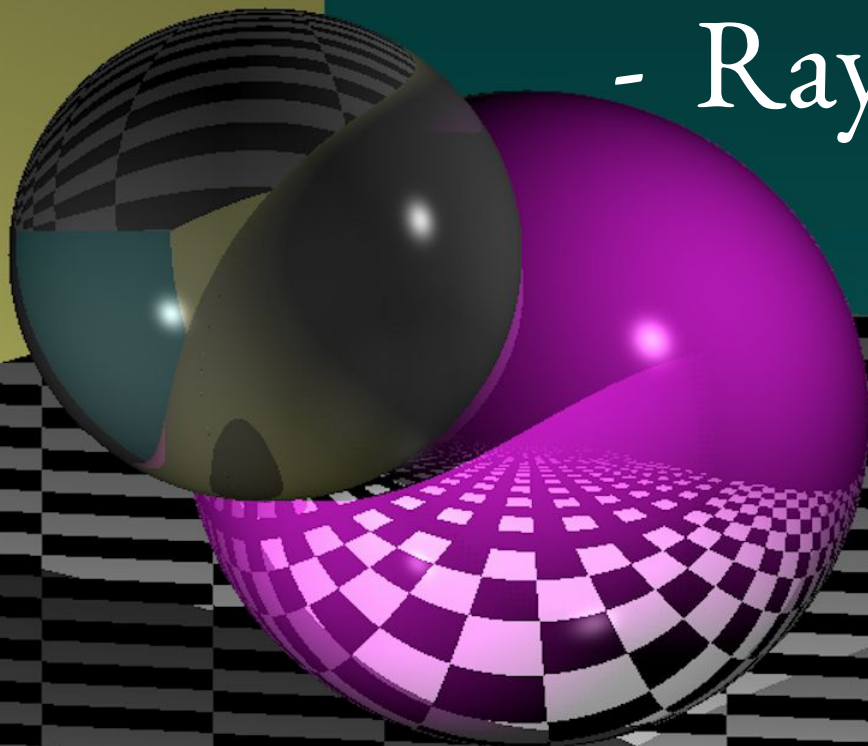


Moteur de rendu - Raytracing -



Bodinier
Puech

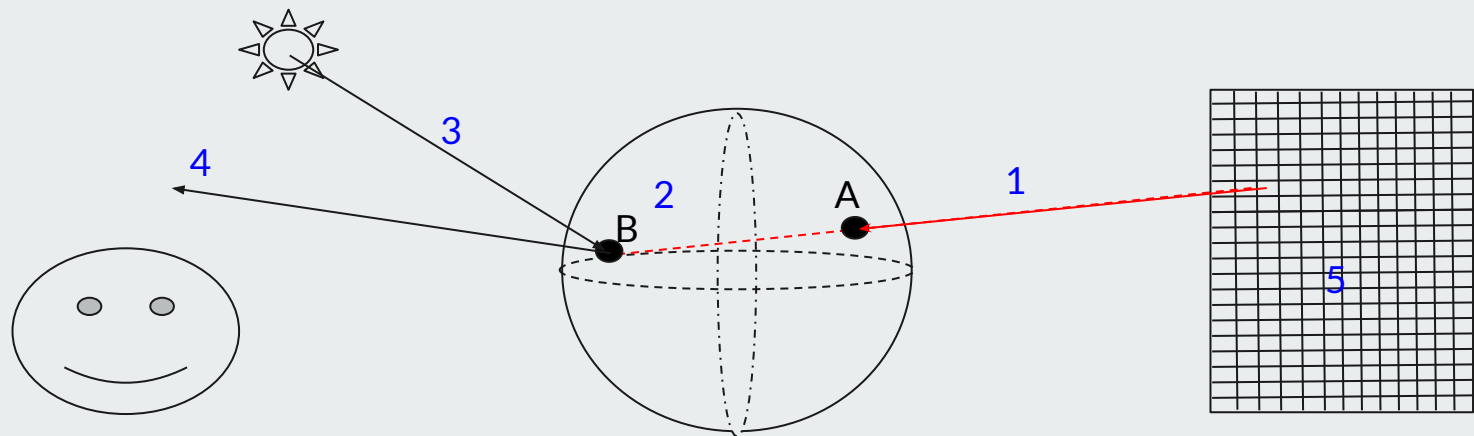
I/ Raytracing de base



Premier objectif :

- Afficher des sphères
- Gérer l'éclairement en fonction de l'incidence et de la distance à la lumière
- Gérer les ombres

Principe de base :



1 : On part du pixel (x,y) et on lance un rayon

2 : On calcule des intersections éventuelles avec des objets

3 : Si on trouve une intersection telle que l'objet est au premier plan (A : OK, B : PAS OK) on calcule l'éclairement et la couleur en fonction des propriétés des objets et lumières

4 : On fait partir un rayon réfléchi et/ou réfracté, on regarde à son tour s'il intersecte un objet, ...

5 : Lorsque l'on a fini tous les rayons secondaires, on fait partir un rayon du pixel suivant



N'ayant pas utilisé de librairies déjà existantes, nous avons dû définir nos propres classes, méthodes, attributs et fonctions

Classes de base : rayon

```
4  class point {
5      public :
6
7      float x, y, z;
8  };
9
10
11  class vecteur {
12      public :
13
14      float x, y, z;
15
16      void normalize(){
17          float norm = sqrtf(x*x + y*y + z*z);
18
19          x /=norm;
20          y /=norm;
21          z /=norm;
22      }
23  };
```

```
81  class ray
82  {
83      public :
84
85      point start;
86      vecteur dir;
87  };
```

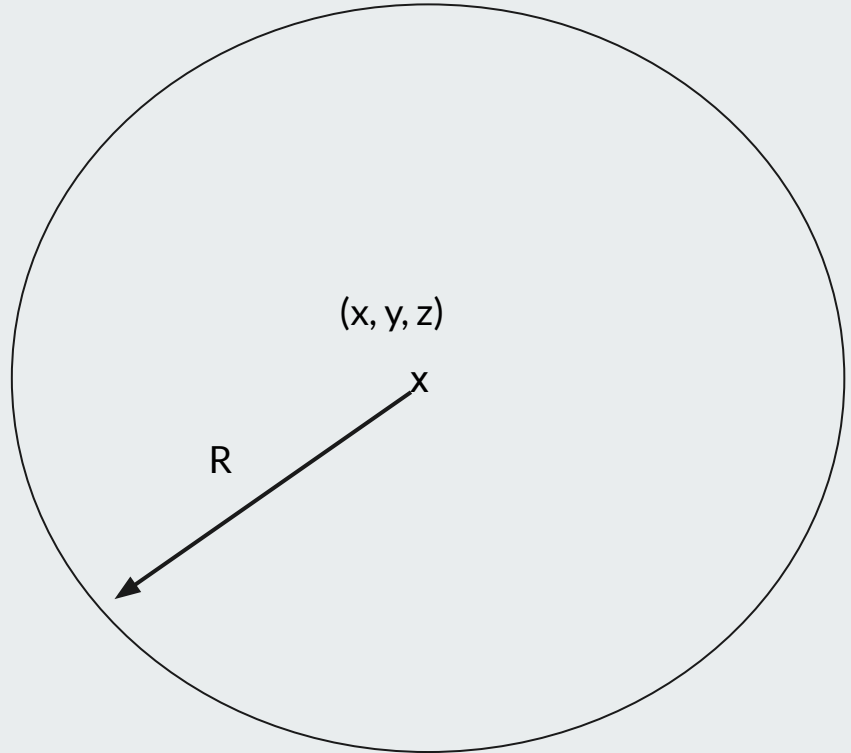
Classes de base : la lumière



```
72  class light
73  {
74      public :
75
76      point pos;
77      float red, green, blue;
78  };
79
```

Classes de base : la sphère

```
48  class sphere {  
49      public :  
50  
51      point pos;  
52      float size;  
53      int material;  
54  };
```




La classe scène

La classe scène va être la description de tous les objets et paramètres de notre rendu (matériaux, positions, paramètres physiques, ...)

```
88
89  class scene
90  {
91      public :
92          scene(){};
93          //scene(const scene &){};
94
95          vector<material> matTab;
96          vector<sphere> sphTab;
97          vector<plan> planTab;
98          vector<paraboloid> parabTab;
99          vector<parallelo> paraTab;
100         vector<light> lgtTab;
101         int sizex, sizey;
102         int nbRebond;
103         int enableSpec;
104         float sVal, sIntensity;
105         bool shadow;
106         bool phong;
107
108     private :
109
110 };
```


Calcul d'intersection (sphère)


$$M \in \text{Rayon donc } \begin{cases} x = x_0 + d_1 t \\ y = y_0 + d_2 t \\ z = z_0 + d_3 t \end{cases}$$

$$M \in \text{Sphère donc } (x - x_c)^2 + (y - y_c)^2 + (z - z_c)^2 = R^2$$

$$M \in \text{Rayon} \cap \text{Sphère} \Leftrightarrow (x_0 + d_1 t - x_c)^2 + (y_0 + d_2 t - y_c)^2 + (z_0 + d_3 t - z_c)^2 = R^2$$

Implémenté par la fonction hitSphère

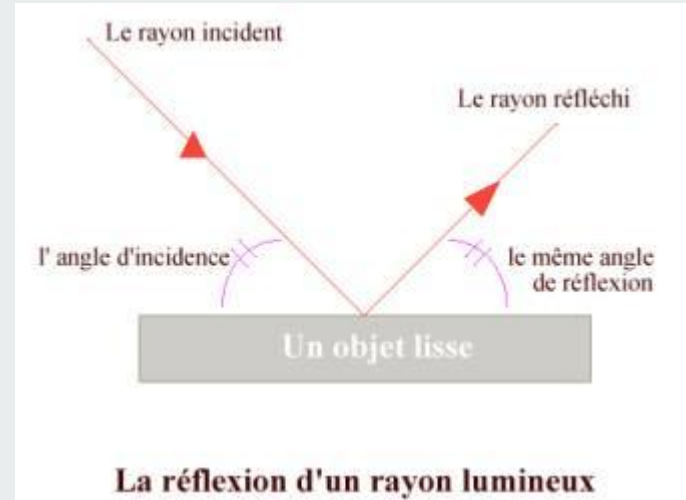
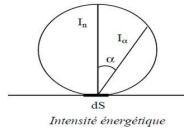
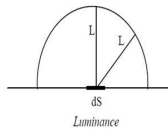
Calculs d'éclairement et de rayons réfléchis

Prise en compte l'influence de la distance à la source lumineuse et à l'incidence des rayons sur l'objet :

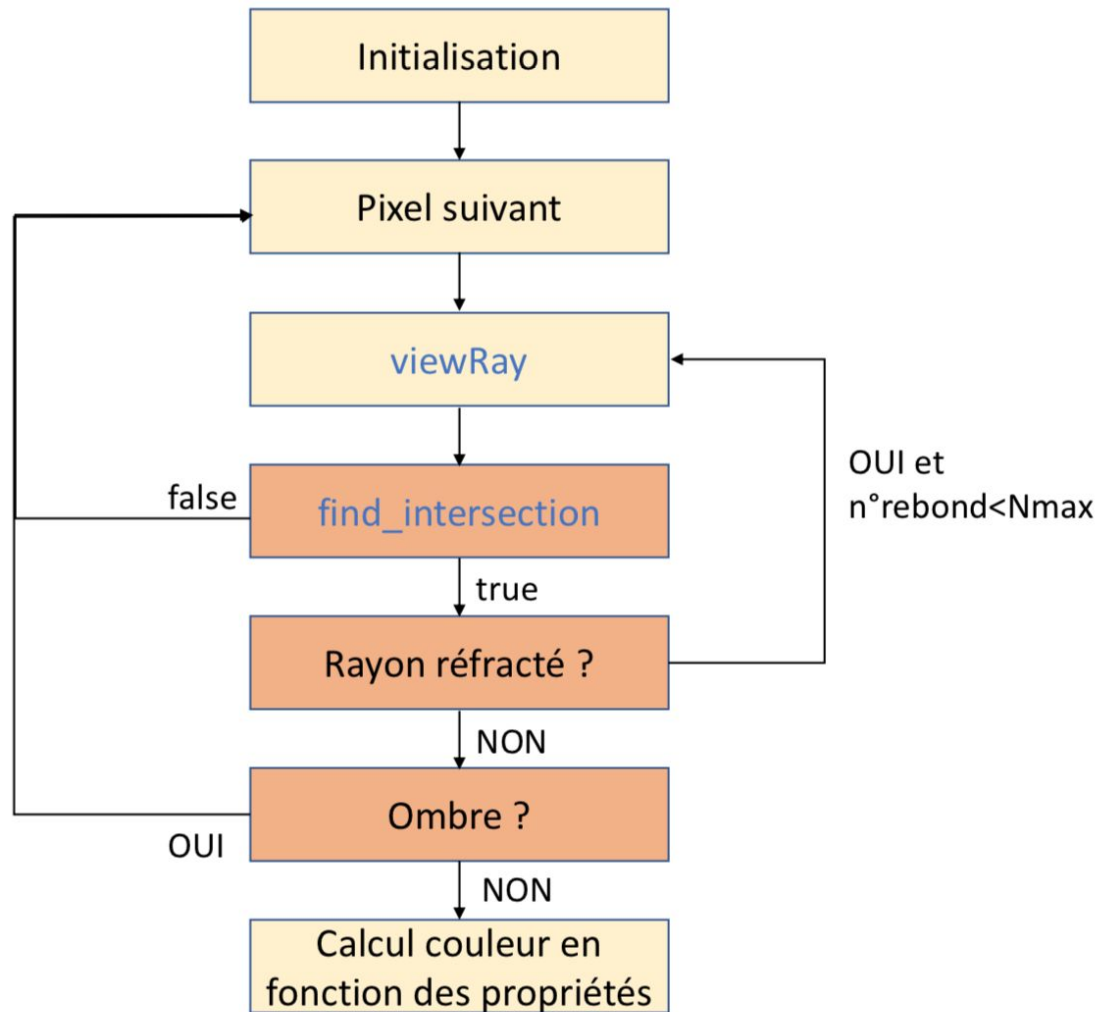
- Produit scalaire entre le viewray et le rayon lumineux nous donne un coefficient de lambert

Ainsi l'indicatrice d'émission est une sphère tangente en O à la surface émettrice lorsque celle-ci suit la loi de Lambert

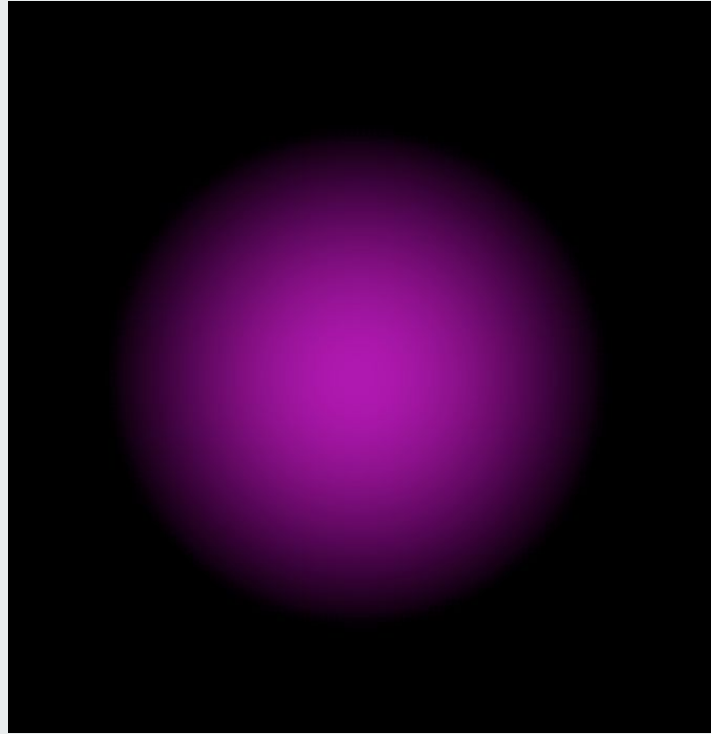
$$\frac{dI_x}{dS} = L \cos \alpha$$



$$\vec{r} = \vec{i} - 2x \langle \vec{i} | \vec{n} \rangle x \vec{n}$$



Premier résultat



II/ Ajout de fonctionnalités




- Gestion d'objets multiples
- Plan
- Paraboloïdes

- Transparence
- Réfraction
- Spécularité
- Texture

Complétons les classes

```
31  class material
32  {
33      public :
34
35      float red, green, blue, reflection, opacity, refraction;
36  };
56  class plan {
57      public :
58
59      vecteur normale;
60      float d;
61      int material;
62      int texture0n;
63  };
```

Calcul d'intersection (plan)


$$M \in \text{Rayon} \text{ donc } \begin{cases} x = x_0 + d_1 t \\ y = y_0 + d_2 t \\ z = z_0 + d_3 t \end{cases}$$

$$M \in \text{Plan} \Leftrightarrow \exists (a, b) \in [0,1]^2 \text{ tels que } \begin{cases} x = ax_{AB} + bx_{AC} + x_A \\ y = ay_{AB} + by_{AC} + y_A \\ z = az_{AB} + bz_{AC} + z_A \end{cases}$$

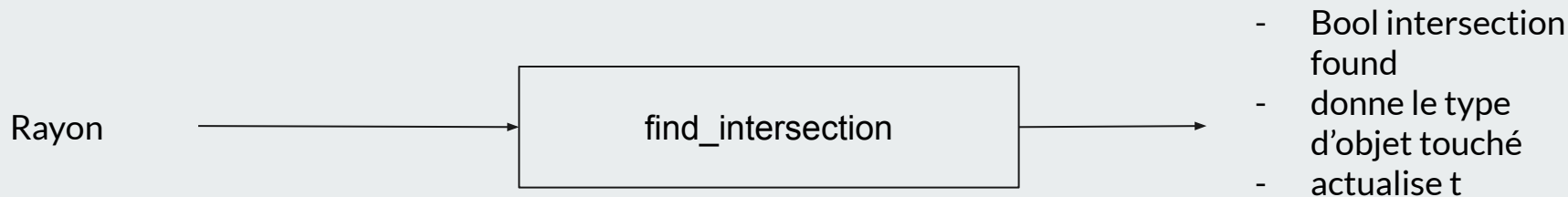
$$M \in \text{Plan} \cap \text{Rayon} \Leftrightarrow \begin{pmatrix} x_{AB} & x_{AC} & -d_1 \\ y_{AB} & y_{AC} & -d_2 \\ z_{AB} & z_{AC} & -d_3 \end{pmatrix} \begin{pmatrix} a \\ b \\ t \end{pmatrix} = \begin{pmatrix} x_0 - x_A \\ y_0 - y_A \\ z_0 - z_A \end{pmatrix}$$

Implémenté par la fonction *hitPlan*

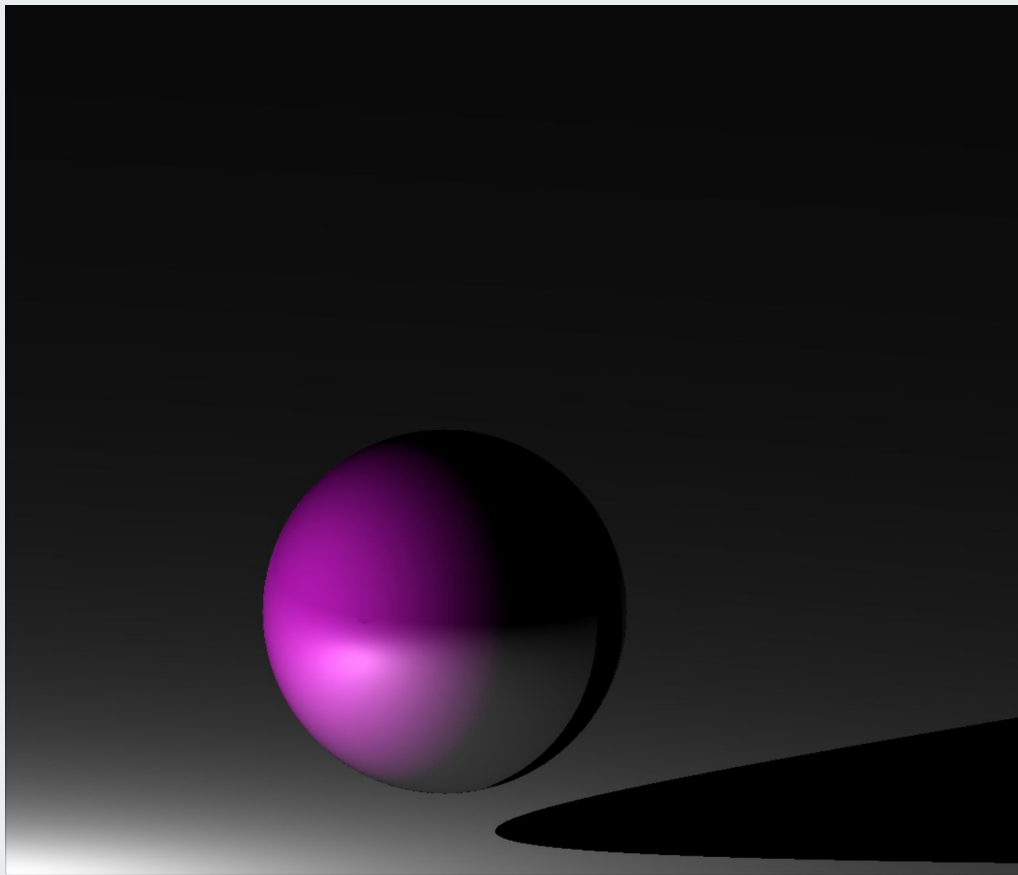
Fonction find_intersection



Cette fonction est primordiale



Affichage d'un plan infini

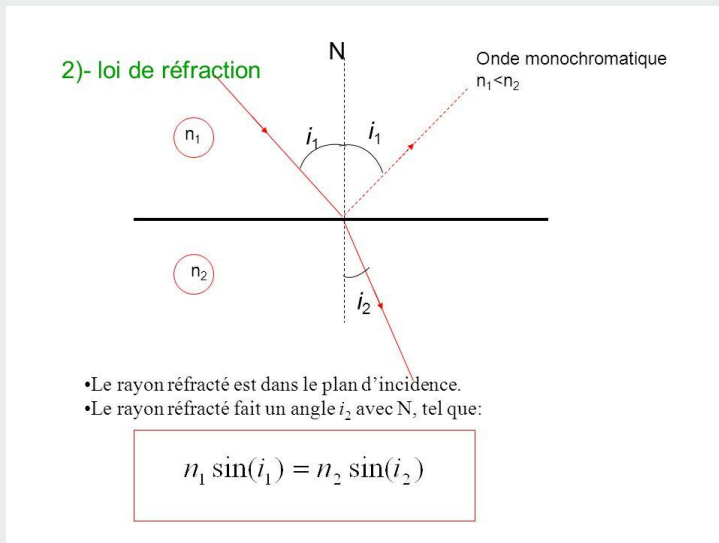


Transparence, réfraction

Transparence :

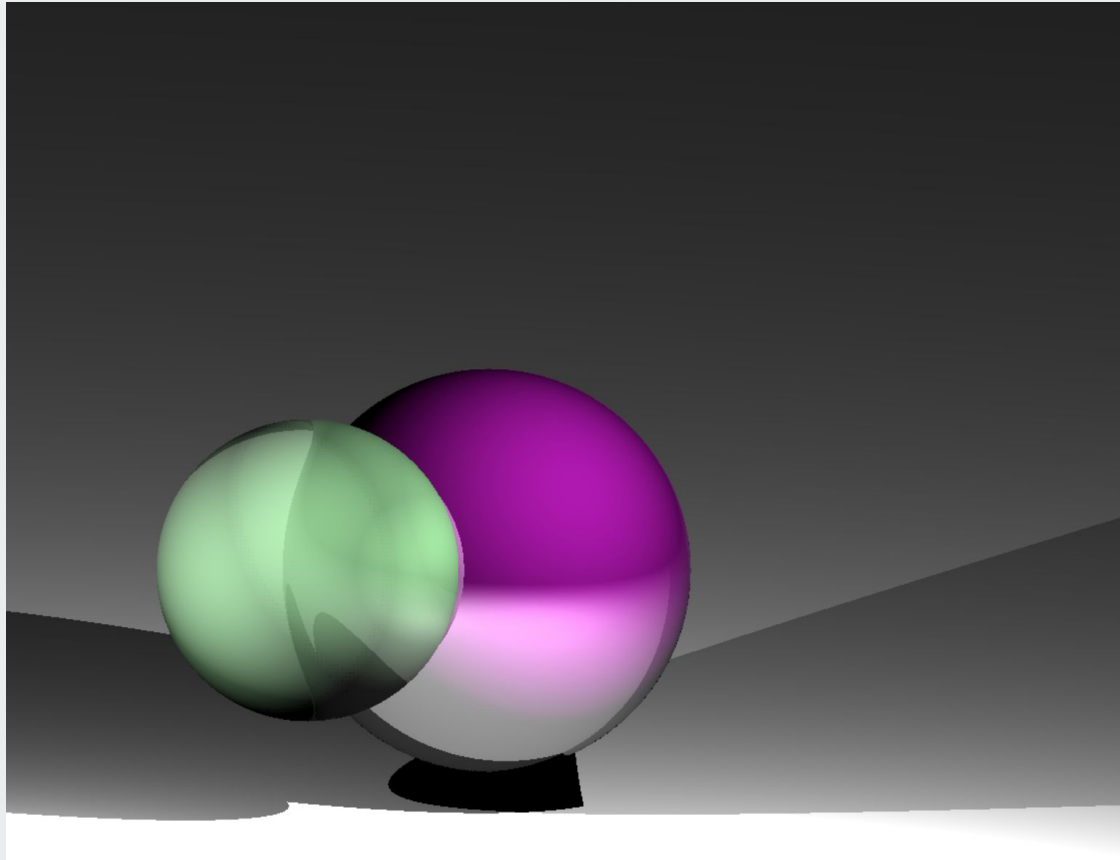
On spécifie dans les attributs de propriétés que le matériau est transparent (concrètement on autorise à des éléments placés derrière lui d'être coloriés aussi)

Loi de
Snell-Descartes :

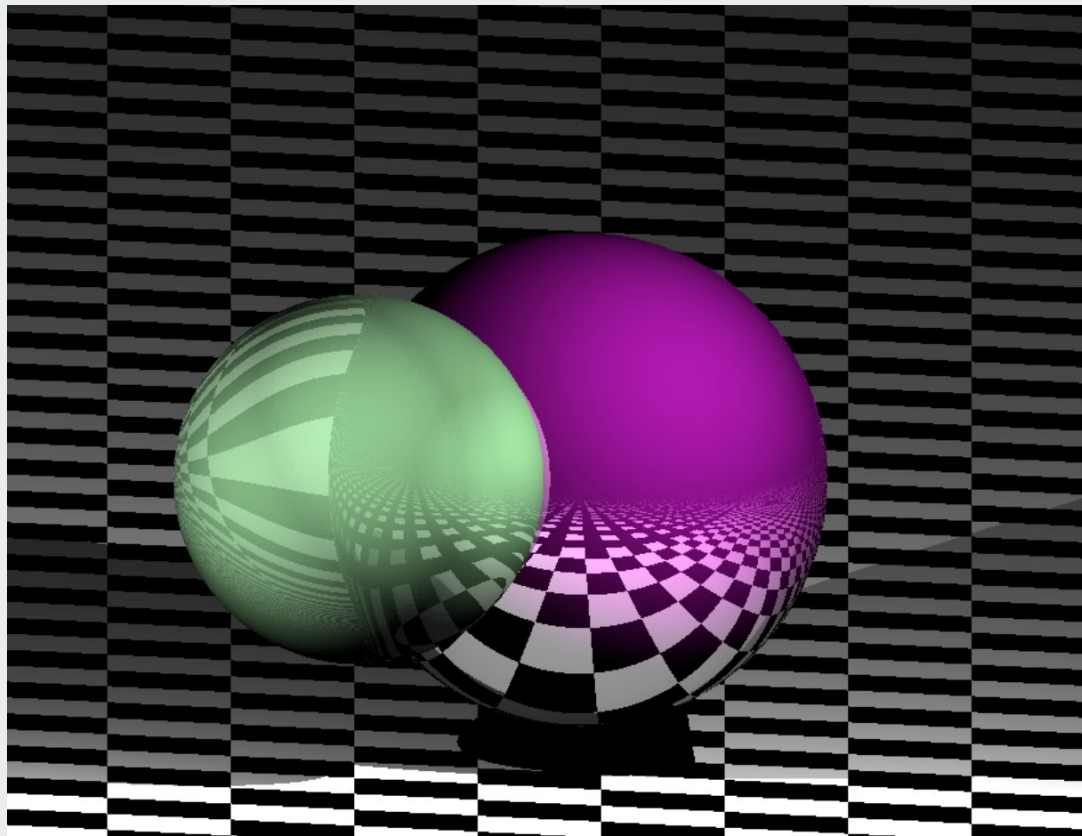


$$\vec{r} = \vec{i} - 2x \langle \vec{i} | \vec{n} \rangle x \vec{n}$$

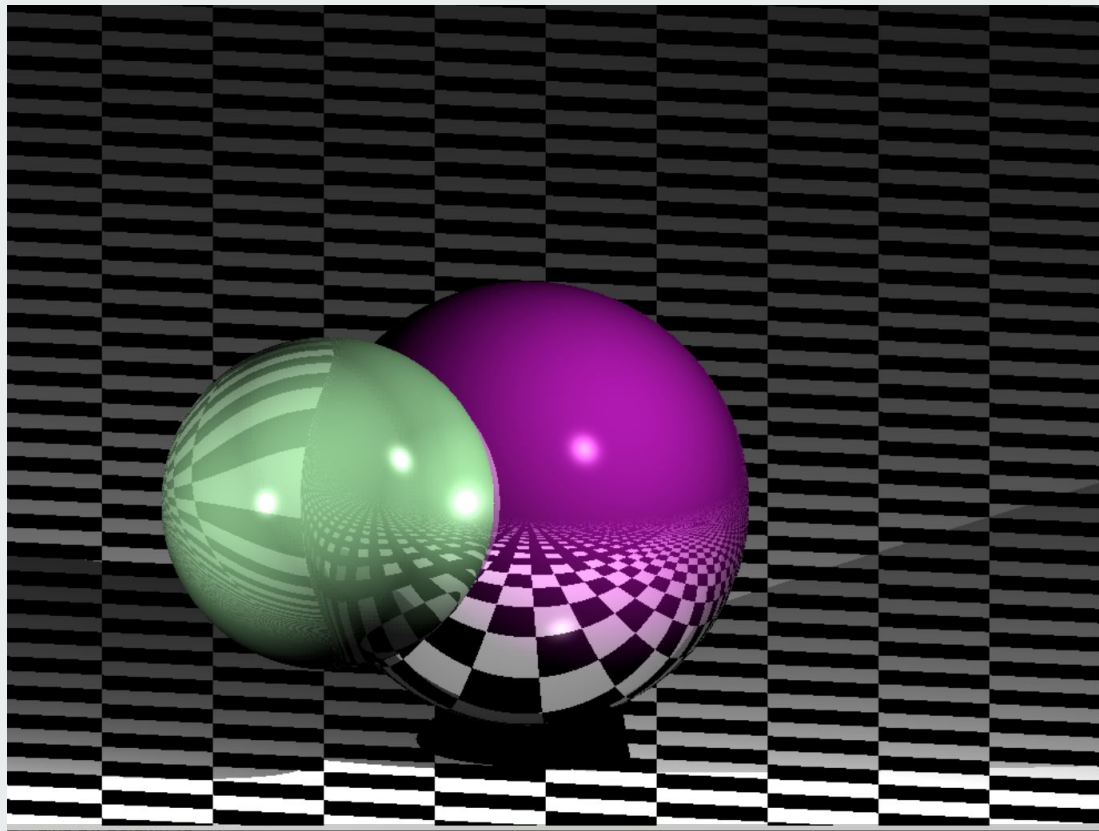
Transparence et réfraction



Texture procédurale



Spécularité

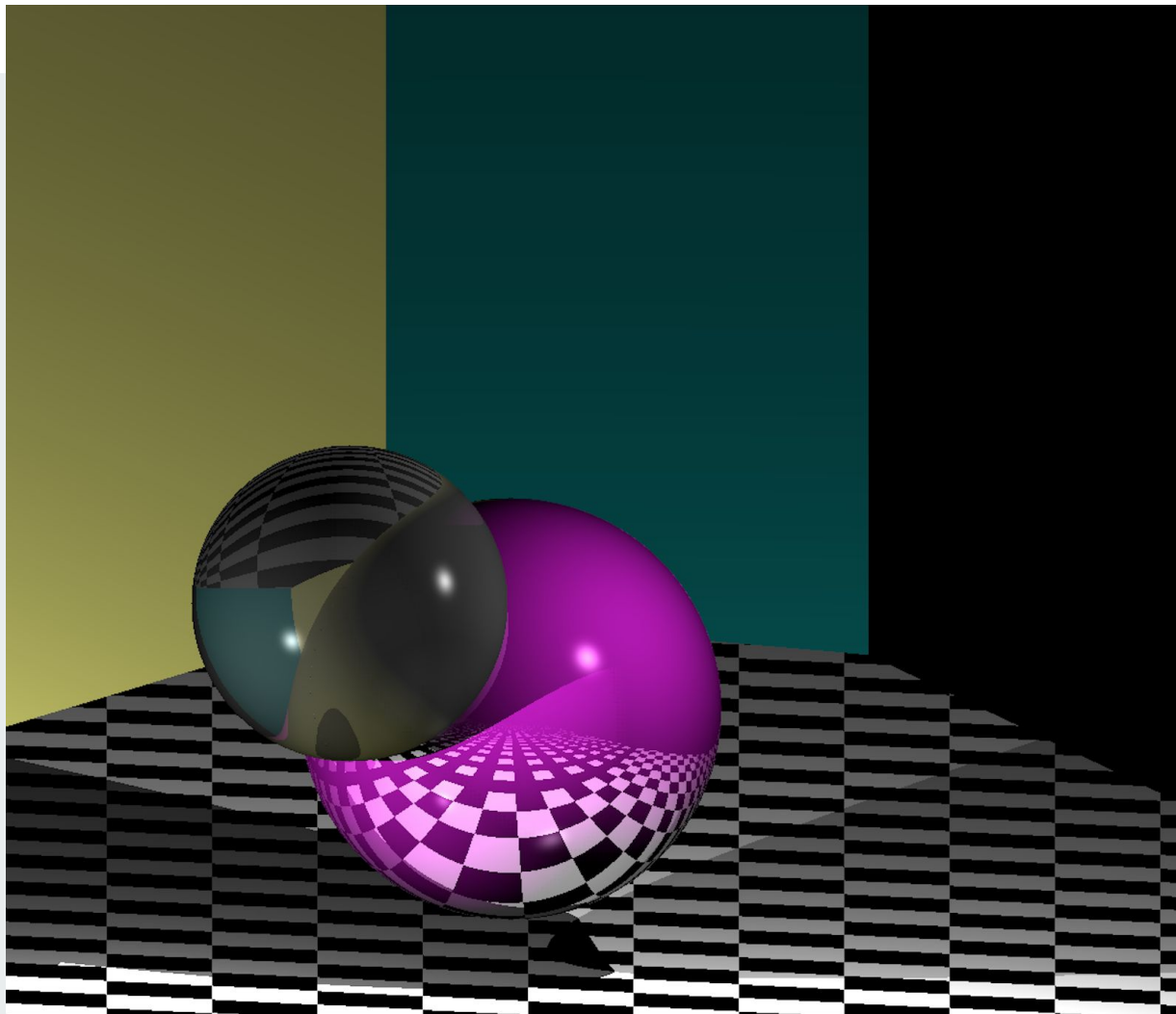


$$k_{\text{spec}} = \|R\| \|V\| \cos^n \beta = (\hat{R} \cdot \hat{V})^n$$

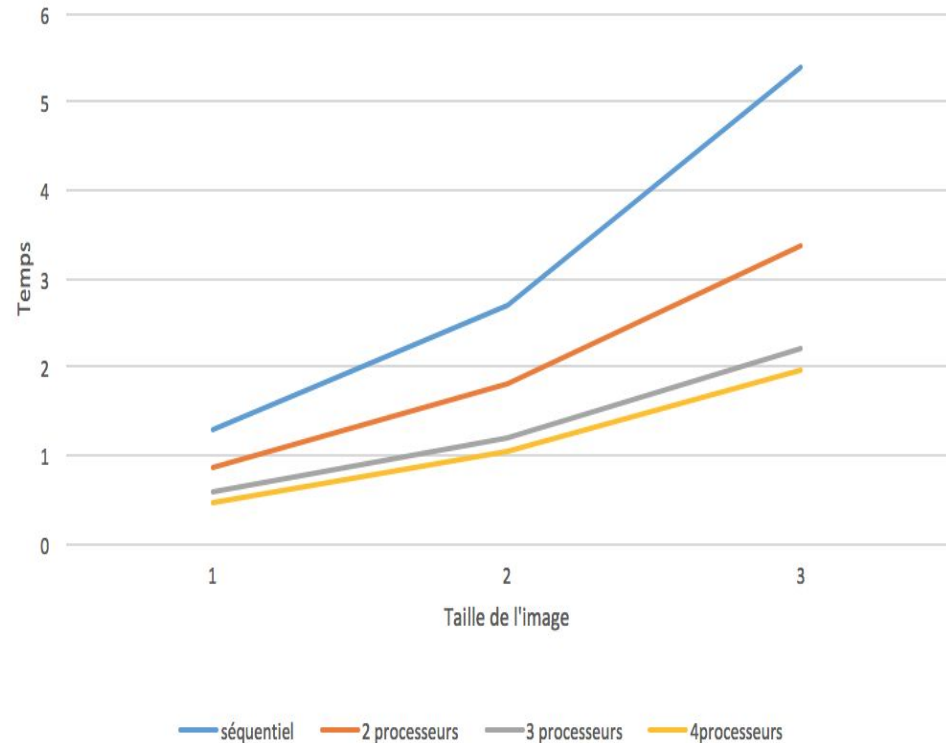
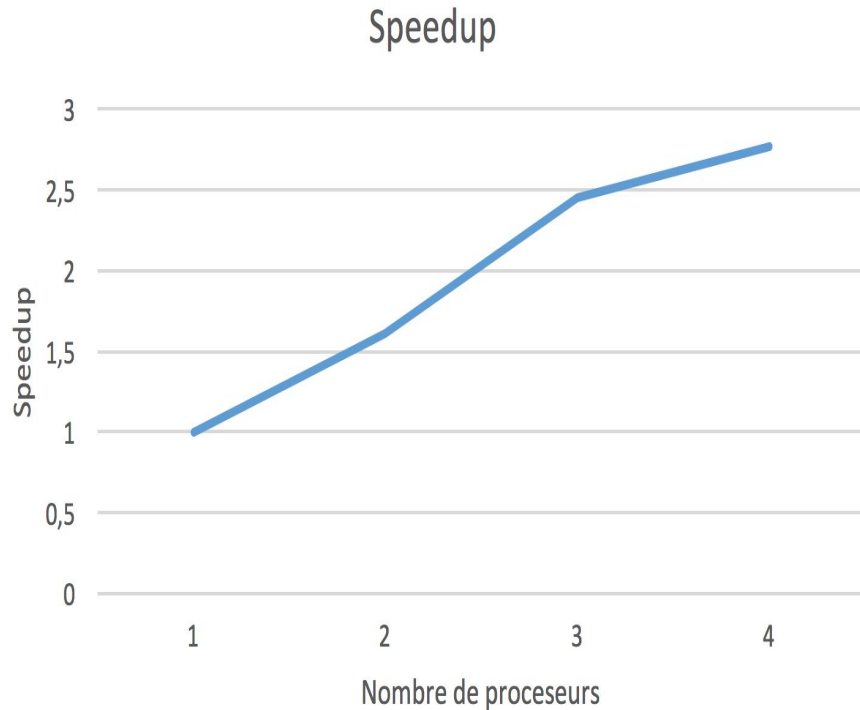
III/ Parallélisme et gestion de scène

Stratégie de parallélisation en client-serveur avec MPI :

Permet de faire des images de plus grande dimensions, plus complexes, avec plus d'objets



Analyse des performances



IV/ Perspectives d'ouverture



DANS UN PREMIER TEMPS :

- Affichage de triangles 2D dans l'espace 3D : avec des triangles on peut créer une variété de formes infinie -> *(nous avons déjà les fonctions et algorithmes)*
- Mapping de plans et de sphères -> établir les fonctions de correspondance
- Bump mapping ->
- Améliorer l'ergonomie de l'interface pour le fichier scène -> XML

PERSPECTIVE SUR PLUS LONG TERME:

- Création d'animations 3D en générant plusieurs images par seconde et en animant les objets (par exemple faire rebondir une sphère par terre)
- Nécessite l'implémentation de nouvelles méthodes dans les classes (masse, rigidité, ...)
- Nécessite la prise en compte de la mécanique newtonienne
- Nécessite l'optimisation des fonctions