



ENSGTI
ÉCOLE D'INGÉNIEURS



Programmation Orientée Objet (Python)

TP5 : Introduction à la programmation orientée-objet (POO) suite

[A.Martin](#)

2026-2027



Visual Studio Code



Objectif

Dans cette seconde séance sur la programmation orientée objet (POO), vous allez enrichir le code du TP précédent, en ajoutant la possibilité de suivre la trajectoire d'un rayon lumineux lorsqu'il rencontre des objets géométriques (cercle, rectangle).

Au terme de ce TP, vous saurez :

- expliquer le principe du tracé de rayons en optique géométrique ;
- implémenter l'intersection entre un rayon et un objet ;
- calculer la direction réfléchie (loi de la réflexion) ;
- visualiser et analyser les impacts ;
- (bonus) déterminer le premier impact dans une scène et suivre les rebonds.

Durée estimée : 4 heures

1. Introduction

En optique géométrique, on modélise la lumière par des rayons (demi-droites orientées). Lorsqu'un rayon rencontre une surface réfléchissante, sa direction de sortie $\mathbf{d}'$ est donnée par la loi de la réflexion (Snell-Descartes) : « l'angle d'incidence est égal à l'angle de réflexion, mesuré par rapport à la normale locale $\mathbf{n}$ ». En suivant ces interactions successives, on simule des systèmes simples : miroirs, cavités, obstacles, etc.

Ce procédé est appelé tracé de rayons (ray casting) et dans ce TP, nous allons en implémenter une version simplifiée dans laquelle :

- les objets sont des cercles et des rectangles ;
- les rayons rebondissent un certain nombre de fois sur les parois ;
- la réflexion est supposée parfaite, sans absorption.

2. Classes supplémentaires

Nous allons enrichir le module précédent par de nouvelles classes :

- La classe **Rayon** représente un rayon lumineux avec son point d'origine et sa direction.
- La classe **Impact** décrit une intersection entre un rayon et une figure. Pour cela, on stocke la figure touchée par le rayon, la position de l'impact, la normale à la surface et la distance parcourue.
- Les classes **CercleOptique** et **RectangleOptique** représentent la version « optique » des figures. Elles implémentent une méthode `intersection_rayon()` qui teste si le rayon coupe la figure.
- La classe **SceneOptique** regroupe plusieurs figures et permet de suivre un rayon pas à pas avec ses rebonds.

On munira aussi notre module de la fonction indépendante `reflechir_direction()`, qui applique la loi de la réflexion pour calculer la direction du rayon après impact.

Pourquoi la POO aide ici ?

Avec une approche purement fonctionnelle, il faudrait distinguer les cas :

```
if type(fig) == Cercle: intersection_cercle(fig, rayon)
if type(fig) == Rectangle: intersection_rectangle(fig, rayon)
```

Avec la POO, chaque figure implémente sa méthode `intersection_rayon()`. La scène n'a donc pas besoin de savoir si c'est un cercle ou un rectangle : c'est le **polymorphisme**. On gagne ainsi en clarté, en extensibilité et en maintenance.

Le diagramme UML de la figure 1 résume cette architecture.

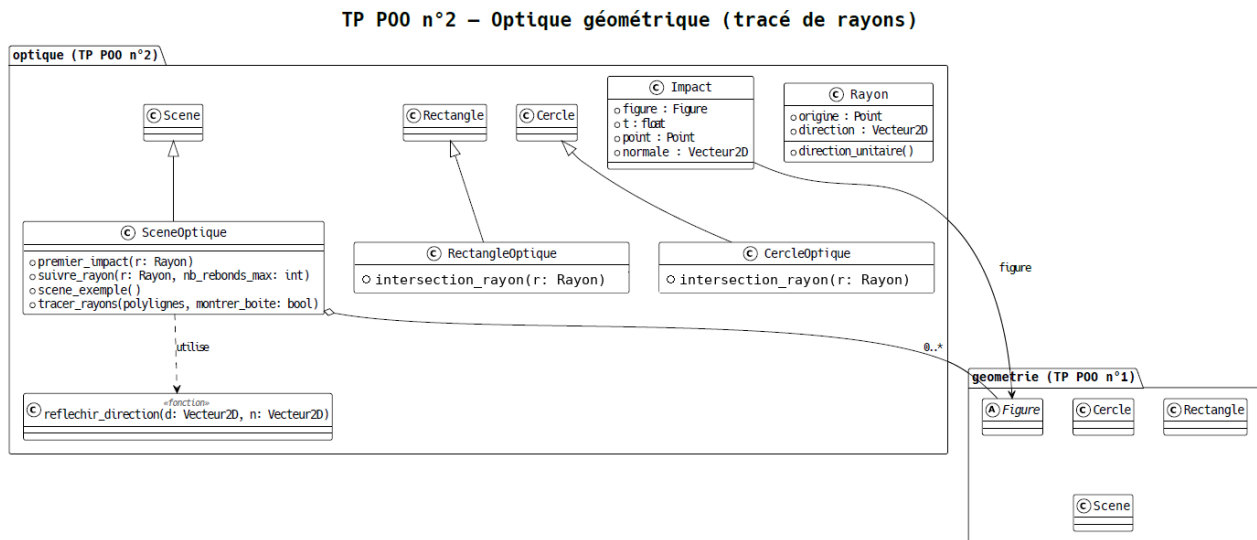


Figure 1 – Diagramme de classes du module « optique ».

3. Algorithme de suivi d'un rayon

Voici un pseudo-code pour `SceneOptique.suivre_rayon()` ; on suppose que $d=1$ et que chaque impact fournit P et une normale unitaire n .

```

Entrée : rayon initial ( $O_0$ ,  $d_0$  unitaire),  $nb\_rebonds\_max$ 
Sortie : liste de points [ $O_0$ ,  $P_1$ ,  $P_2$ , ...,  $P_k$ ]

points = [ $O_0$ ]
 $O = O_0$ 
 $d = d_0$ 

POUR  $i = 1$  À  $nb\_rebonds\_max$  FAIRE
    impact = premier_impact( $O$ ,  $d$ )      # renvoie ( $P$ ,  $n$ ,  $t$ ) avec  $t \geq 0$ , sinon None

    SI impact est None ALORS
        points.ajouter( $O + T\_max * d$ ) # prolonge "vers l'infini" pour tracer la sortie
        QUITTER
    FIN SI

    ( $P$ ,  $n$ ,  $t$ ) = impact
    points.ajouter( $P$ )

    # réflexion ( $n$  unitaire) ;  $d$  restera unitaire
     $d = d - 2 * (d \cdot n) * n$ 

    # décalage anti ré-impact numérique (epsilon petit)
     $O = P + epsilon * n$ 
FIN POUR

RETOURNER points

```

Recommandations numériques : choisir $\varepsilon \in [10^{-9}, 10^{-6}]$ en fonction des tailles de la scène ; garder $\|d\| = 1$ pour que t soit en unités de longueur.

4. Travail à réaliser

4.1 Étape préparatoire

1. Ouvrez un terminal.
2. Dans votre dossier de travail « tp_python », désarchivez le fichier ressources « tp5.zip » disponible sur Moodle :

```
unzip tp5.zip
```

3. Déplacez-vous dans le dossier « tp5 ». Vous devriez voir l'arborescence suivante :

```
tp5/  
|-- src/  
|   |-- geometrie.py    # module "geometrie"  
|   |-- optique.py      # fichier Python à compléter  
|   |-- __init__.py  
|-- tests/  
|   |-- test_optique.py  # tests fournis  
|   |-- __init__.py  
|-- requirements.txt
```

4. Créez l'environnement virtuel « env_tp5 », activez-le et installez les bibliothèques nécessaires :

```
pip install -r requirements.txt
```

5. Désactivez ensuite l'environnement virtuel (vous l'activerez depuis VSCode).

4.2 Développement des classes

Vous allez implémenter les classes dans l'ordre ci-dessous. Après chaque étape, vous lancerez le test fourni pour visualiser vos résultats :

```
pytest --cov=src --cov-report=term-missing -v
```

Étape 1 – Loi de la réflexion

Implémentez la fonction indépendante `reflechir_direction(d, n)`, qui calcule la direction réfléchie d'un rayon lumineux incident.

Cette fonction reçoit :

- **d** : direction incidente ;
- **n** : normale unitaire à la surface au point d'impact ;

et doit renvoyer la direction réfléchie unitaire **d'** selon la loi de la réflexion.

Principe

Lorsqu'un rayon lumineux rencontre une surface plane, l'angle d'incidence θ_i et l'angle de réflexion θ_r (mesurés par rapport à la normale) sont égaux :

$$\theta_i = \theta_r$$

Le vecteur réfléchi **d'** s'exprime directement par la relation vectorielle suivante :

$$\mathbf{d}' = \mathbf{d} - 2(\mathbf{d} \cdot \mathbf{n}) \mathbf{n}$$

avec $\|\mathbf{n}\| = 1$.

Pseudo-code

```

Entrées :
  d = (dx, dy) : direction incidente
  n = (nx, ny) : normale unitaire à la surface

Sortie :
  d_reflechie = (dx', dy') : direction réfléchie (unitaire)

Étapes :
1. Calculer le produit scalaire :
   s = d · n = dx*nx + dy*ny

2. Appliquer la formule de réflexion :
   d' = d - 2*s*n

3. Normaliser le vecteur réfléchi :
   norme = racine_carree(d'_x^2 + d'_y^2)
   d' = d' / norme

4. Retourner d'

```

La figure 2 est le résultat d'un test positif avec un rayon incident (bleu) qui est réfléchi (orange), et une normale (verte).

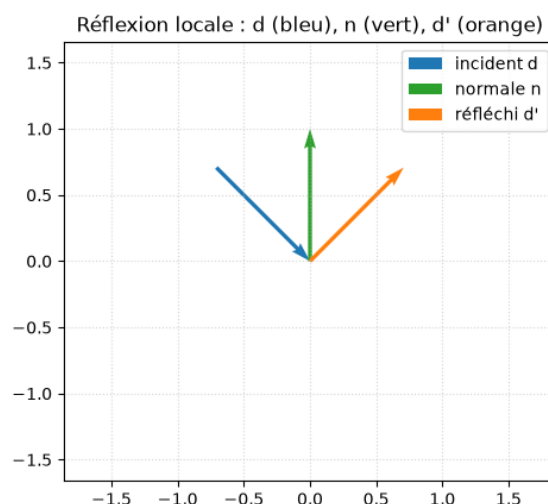


Figure 2 – Résultat du test de la fonction `reflechir_direction(d, n)`.

Étape 2 - Cercle optique

Implémentez la méthode `CercleOptique.intersection_rayon(0, d)` qui détermine si le rayon

$$M(t) = \mathbf{O} + t \mathbf{d}, \quad t \geq 0$$

coupe un cercle de centre $\mathbf{C}$ et de rayon R , et renvoie :

- t : distance paramétrique du premier impact (le plus petit $t \geq 0$) ;
- $\mathbf{P}$: point d'impact ;
- $\mathbf{n}$: normale sortante unitaire au point d'impact ;
- `None` si le rayon ne coupe pas le cercle.

Principe

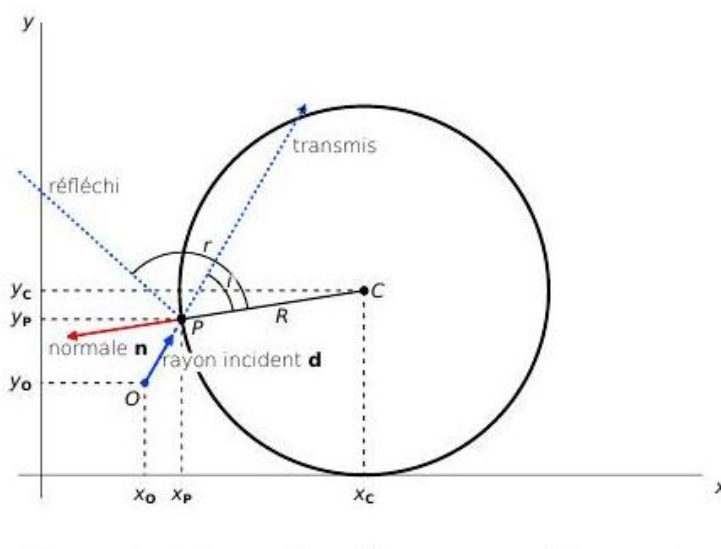


Figure 3 – Intersection d'un rayon et d'un cercle.

Un point du rayon vérifie :

$$\|M(t) - C\|^2 = R^2$$

En posant $\mathbf{u} = \mathbf{O} - \mathbf{C}$, on obtient une équation quadratique en t :

$$(\mathbf{d} \cdot \mathbf{d}) t^2 + 2(\mathbf{u} \cdot \mathbf{d}) t + (\mathbf{u} \cdot \mathbf{u} - R^2) = 0$$

Le discriminant $\Delta = b^2 - 4ac$ indique la nature de l'intersection :

- si $\Delta < 0$: pas d'intersection ;
- si $\Delta \geq 0$: deux solutions t_1 et t_2 ;

Le premier impact correspond au plus petit $t_i \geq 0$.

On en déduit :

$$\mathbf{P} = \mathbf{O} + t \mathbf{d} \quad , \quad \mathbf{n} = \frac{(\mathbf{P} - \mathbf{C})}{\|\mathbf{P} - \mathbf{C}\|}$$

Pseudo-code

Entrées :

$\mathbf{O} = (O_x, O_y)$: origine du rayon
 $\mathbf{d} = (d_x, d_y)$: direction du rayon
 $\mathbf{C} = (C_x, C_y)$: centre du cercle
 R : rayon

Sortie :

(t, P, n) ou None

Étapes :

1. $\mathbf{u} = \mathbf{O} - \mathbf{C}$

$a = \mathbf{d} \cdot \mathbf{d}$

$b = 2 * (\mathbf{u} \cdot \mathbf{d})$

$c = (\mathbf{u} \cdot \mathbf{u}) - R^2$

2. $\Delta = b^2 - 4ac$

Si $\Delta < 0 \rightarrow$ RETOURNER None # pas d'intersection réelle

3. $t_1 = (-b - \text{racine_carree}(\Delta)) / (2a)$

$t_2 = (-b + \text{racine_carree}(\Delta)) / (2a)$

```

4. Garder le plus petit  $t \geq 0$  :
    $t = \min(t_1, t_2)$  avec  $t \geq 0$ 
   Si aucun  $t \geq 0 \rightarrow$  RETOURNER None

5. Calculer le point d'impact :
    $P = O + t \cdot d$ 

6. Calculer la normale sortante unitaire :
    $n = (P - C) / ||P - C||$ 

7. RETOURNER ( $t, P, n$ )

```

Exemple

La figure 4 résulte d'un test positif de la méthode `intersection_rayon()` avec un cercle (bleu), un rayon incident (tirets bleus) et son point d'impact (rouge). La normale est quant à elle tracée en orange.

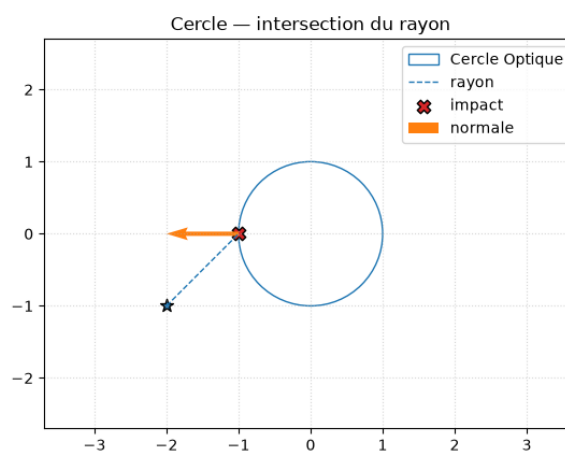


Figure 4 – Résultat du test de la méthode `intersection_rayon()` de la classe `CercleOptique`.

Étape 3 – Rectangle optique

Implémentez la fonction `RectangleOptique.intersection_rayon()` (rectangle aligné sur les axes) par la méthode des bandes. Cette méthode doit déterminer si le rayon

$$M(t) = O + t \cdot d, \quad t \geq 0$$

coupe le rectangle défini par les bornes $[x_{min}, x_{max}] \times [y_{min}, y_{max}]$, et renvoyer :

- t : distance paramétrique le long du rayon ;
- P : point d'impact ;
- n : normale sortante unitaire à la face touchée ;
- None si aucune intersection n'existe.

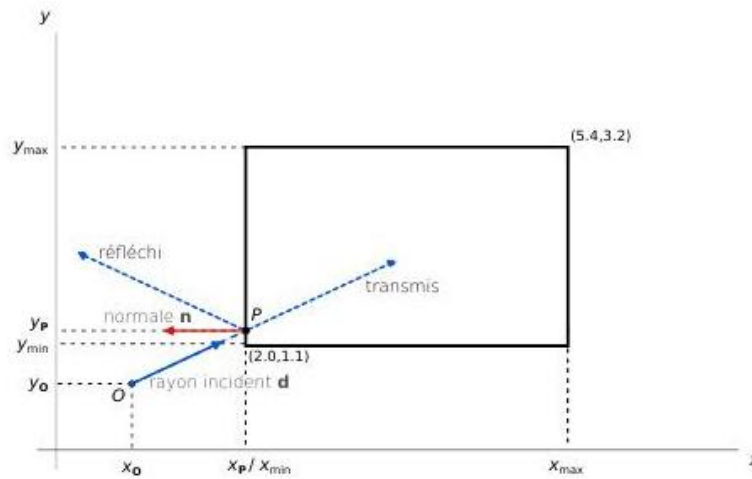


Figure 5 – Intersection d'un rayon et d'un rectangle.

Le principe repose sur la **méthode des bandes** : on calcule les intervalles de valeurs de t pour lesquels le rayon est à l'intérieur du rectangle selon les axes x et y . S'il existe une portion commune positive de ces intervalles, le rayon traverse le rectangle.

Rappel du principe

1. Pour chaque axe, on détermine les bornes d'entrée et de sortie du rayon dans la bande :

- Pour x :

$$t_1 = \frac{x_{\min} - O_x}{dx}, \quad t_2 = \frac{x_{\max} - O_x}{dx}$$

puis $t_{x\min} = \min(t_1, t_2)$, $t_{x\max} = \max(t_1, t_2)$,

- On applique le même raisonnement pour y .

2. L'intervalle global d'intersection est :

$$t_{\text{entrée}} = \max(t_{x\min}, t_{y\min}), \quad t_{\text{sortie}} = \min(t_{x\max}, t_{y\max})$$

3. Si $t_{\text{entrée}} > t_{\text{sortie}}$: le rayon ne traverse pas le rectangle. Si $t_{\text{sortie}} < 0$: l'intersection est derrière l'origine. Sinon, le premier impact valide est obtenu pour $t = \max(t_{\text{entrée}}, 0)$.

4. On calcule ensuite le point d'impact et la normale sortante :

$$\mathbf{P} = \mathbf{O} + t \mathbf{d}, \quad \mathbf{n} \in \{(-1, 0), (1, 0), (0, -1), (0, 1)\}$$

selon la face touchée.

La figure 6 illustre le principe de la méthode des bandes :

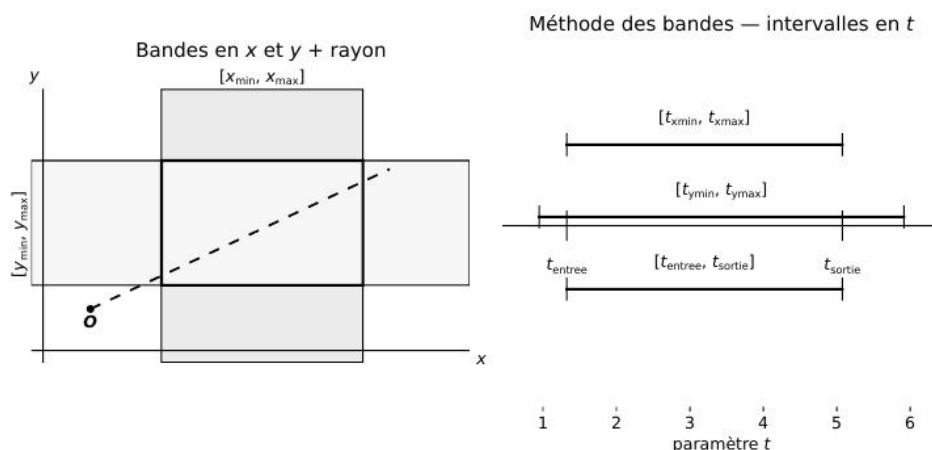


Figure 6 – Méthode des bandes.

Le pseudo-code de la méthode des bandes est le suivant :

```

Entrées :
  O = (Ox, Oy) : origine du rayon
  d = (dx, dy) : direction du rayon
  rectangle : bornes xmin, xmax, ymin, ymax

Sortie :
  (t, P, n) ou None

Constante :
  EPS = 1e-9   # tolérance flottante

Étapes :
1. Si |dx| < EPS :
    si Ox < xmin ou Ox > xmax -> aucun impact (None)
    sinon txmin = -infini ; txmax = +infini
    Sinon :
      t1x = (xmin - Ox) / dx
      t2x = (xmax - Ox) / dx
      txmin = min(t1x, t2x)
      txmax = max(t1x, t2x)

2. Même chose pour y -> obtenir tymin et tymax.

3. Calculer :
    tentree = max(txmin, tymin)
    tsortie = min(txmax, tymax)

4. Si tentree > tsortie -> aucun impact.
   Si tsortie < 0 -> aucun impact.

5. Premier point valable :
    t = max(tentree, 0)
    P = O + t.d

6. Déterminer la normale selon la face touchée :
    si |Px - xmin| < EPS -> n = (-1, 0)
    si |Px - xmax| < EPS -> n = ( 1, 0)
    si |Py - ymin| < EPS -> n = (0, -1)
    si |Py - ymax| < EPS -> n = (0,  1)

7. Retourner (t, P, n)

```

La figure 7 résulte d'un test positif de la méthode `intersection_rayon()` avec un rectangle (bleu), un rayon incident (tirets bleus) et son point d'impact (rouge). La normale de la face touchée est tracée en orange.

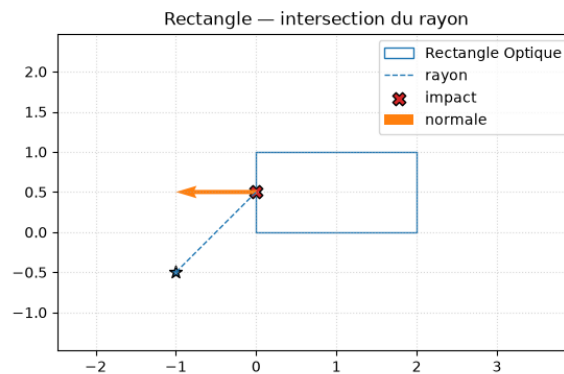


Figure 7 – Résultat du test de la méthode `intersection_rayon()` de la classe `RectangleOptique`.

Les étapes 4 et 5 suivantes sont des **exercices bonus**. Elles ne sont pas exigées pour la validation, mais permettent d'aller plus loin si vous avez terminé les étapes précédentes.

Étape 4 - Premier impact dans une scène (exercice bonus)

Implémentez la méthode `premier_impact()` de la classe `SceneOptique`. Elle doit renvoyer l'impact le plus proche du rayon parmi toutes les figures de la scène.

La figure 8 résulte d'un test positif de la méthode `premier_impact()` avec une scène contenant un cercle et un rectangle, un rayon incident (tirets bleus), l'impact (croix rouge) et la normale (orange).

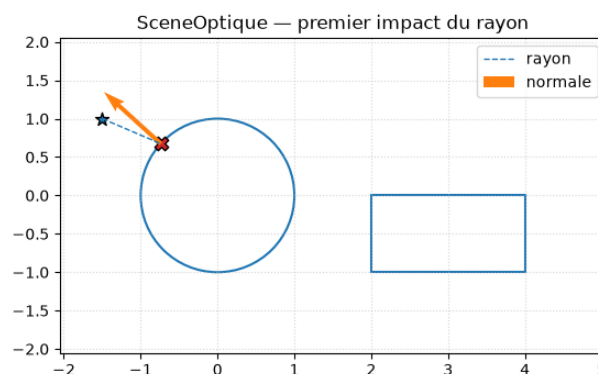


Figure 8 – Résultat du test de la méthode `premier_impact()` de la classe `SceneOptique`.

Étape 5 – Suivi de rayon avec rebonds (exercice bonus)

Implémentez la méthode `suivre_rayon()` de la classe `SceneOptique`. Cette méthode doit suivre un rayon dans la scène, en calculant ses rebonds successifs (jusqu'à un nombre maximal donné). À chaque impact, la direction est mise à jour par la loi de la réflexion.

Pourquoi décaler légèrement l'origine du rayon après impact (EPS) ?

Si on reprend exactement le point d'impact comme nouvelle origine, le calcul numérique peut considérer que le rayon « recoupe » la même surface à l'itération suivante (boucle infinie).

Solution : $\mathbf{O}_{\text{nouveau}} = \mathbf{P} + \varepsilon \mathbf{n}$ où $\mathbf{P}$ est l'impact, $\mathbf{n}$ la normale locale, et ε une très petite valeur (1e-6).

Cela place la nouvelle origine juste à l'extérieur de la surface.

La figure 9 résulte d'un test positif de la méthode `suivre_rayon()`, avec une scène contenant une boîte rectangulaire avec un obstacle circulaire. Trois rayons différents sont lancés depuis la gauche :

- leur trajet est affiché en tirets colorés,
- le point de départ est marqué par une étoile,
- le point de sortie par une croix.

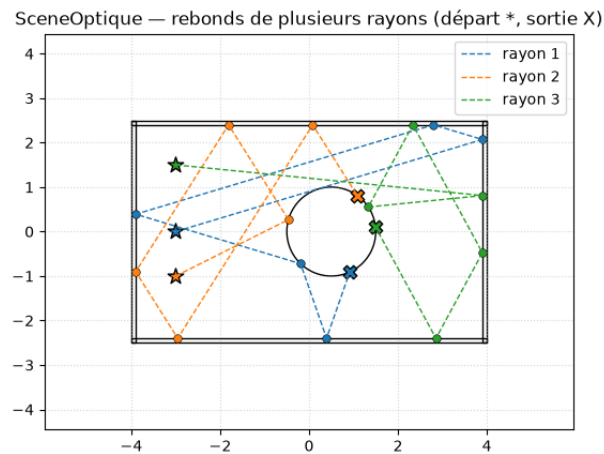


Figure 9 – Résultat du test de la méthode `suivre_rayon()` de la classe `SceneOptique`.

Résumé

Dans ce TP, vous avez appris à :

- implémenter le tracé de rayons dans une scène en deux dimensions ;
- utiliser des intersections rayon/objet afin de calculer les impacts ;
- mettre en œuvre la loi de la réflexion ;
- suivre un rayon avec plusieurs rebonds et visualiser sa trajectoire .