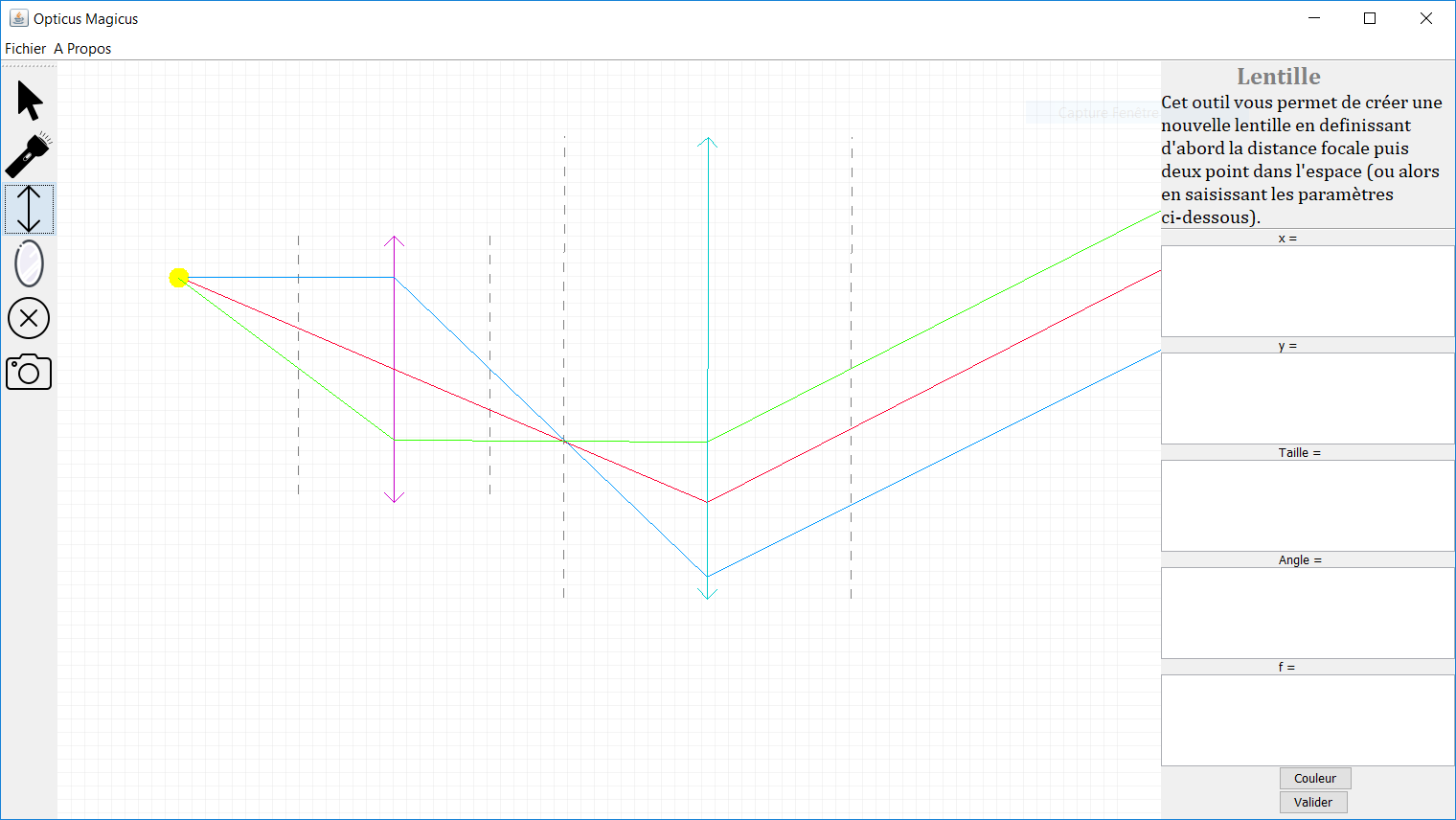
# Projet d’informatique de 2ème année : Opticus Magicus

## Introduction

Afin de mettre en œuvre l’ensemble des compétences acquises en informatique durant nos deux premières années à l’INSA de Lyon, notre objectif a été de concevoir un simulateur optique dénommé « Opticus Magicus ». En effet, l’optique constitue une partie importante des enseignements de physique du Premier Cycle, aussi bien en 1ère qu’en 2ème année. Nous avons donc réalisé un programme permettant de simuler les trajets de faisceaux lumineux en présences de plusieurs objets optiques tels que les lentilles convergentes et divergentes ou encore les miroirs. D’autre part, afin que le programme puisse être utilisé par un public très large, nous nous sommes particulièrement intéressés à la facilité d’utilisation. Afin de pouvoir développer le projet de manière méthodique, il nous a été nécessaire d’élaborer un cahier des charges, qui s’est enrichi au fur et à mesure.

Capture d’écran faite pour la simulation de la construction d’image dans un microscope

## Cahier des charges

Notre programme a été conçu dans un contexte éducatif et de découverte, s’adressant par exemple à des étudiants pouvant modéliser leurs expériences d’optique facilement. Le programme doit donc être intuitif et assez complet pour l’utilisateur. Pour cela, l’interface graphique doit être suffisamment épurée et les fonctionnalités doivent être facilement identifiables.

L’utilisateur doit pouvoir placer différents objets optiques respectant les lois physiques dans un espace plan :

* Une source de lumière pouvant être modélisée par un unique faisceau, dont la position et la direction du faisceau est choisie
* Des lentilles, dont la focale est déterminée par l’utilisateur (positive pour des lentilles convergentes et négative pour des lentilles divergentes)
* Des miroirs parfaits, c’est-à-dire des miroirs qui réfléchissent la totalité de la lumière reçue, mais aussi des miroirs semi-réfléchissants, qui réfléchissent une partie de la lumière et en transmettent une autre

Afin de rendre le programme le plus simple pour l’utilisateur et faciliter la compréhension, ce dernier doit pouvoir choisir la couleur des objets optiques ainsi que celle des faisceaux de lumière. Dans ce même but, la position et la taille des objets optiques doit pouvoir être choisie.

Le programme doit disposer d’une fonctionnalité de sauvegarde de l’état de la construction graphique, et donc d’une fonctionnalité d’importation de fichier pour ouvrir une certaine configuration. De plus, le programme doit permettre de prendre une capture d’écran de la zone de traçage directement à travers un bouton intégré à l’interface graphique.

## Problèmes posés

Les principaux problèmes rencontrés sont :

* La détermination de l’intersection entre les faisceaux et les objets optiques :

Ce problème nous a fait perdre du temps et nous avons modifié notre code de nombreuses fois pour que cela fonctionne normalement

* Le codage de la trajectoire du rayon transmis :

Le problème s’explique par la nécessité d’utiliser un mode de construction graphique, et donc tracer un autre faisceau, et déterminer d’autres points d’intersections

* S’assurer que le rayon transmis est du « bon » côté de l’objet optique :

En effet, cela demande d’être sur du coté sur lequel arrive le faisceau, ce qui a été résolu dans notre cas par l’utilisation du produit scalaire

## Principe de l’algorithme

### Interface graphique

Pour faciliter l’utilisation du programme, il suffit de lancer le run.bat, ce qui va ouvrir directement le programme. Afin de rendre le programme le plus intuitif possible, 5 grandes icônes sont présentes sur la barre d’outils, qui elle-même peut être placée où l’utilisateur le désire.

Afin de placer un objet optique, on a le choix entre saisir la position (x,y) de son centre, sa taille et son angle d’inclinaison par rapport à l’horizontale, ou bien on peut directement placet l’objet avec la souris. Tous les objets optiques sont sauvegardés dans la classe ZoneTracage dans une ArrayList d’objets optiques. Grâce à l’icône « Sélectionner », on peut même facilement repositionner les objets avec un « click and drag ». Les valeurs des tailles, positions et angle s’adaptent alors automatiquement.

Les systèmes optiques étudiés étant généralement composés de nombreuses lentilles ou miroirs, il est proposé à l’utilisateur de choisir la couleur de chaque objet optique grâce à une large palette de couleur en cliquant sur « Couleur ».

Les trois icônes suivantes permettent donc de choisir entre « Source », « Lentille », et « Miroir ». Afin de supprimer un élément, l’utilisateur peut soit sélectionner l’objet et appuyer sur « Supprimer », ou alors appuyer sur l’icône « Supprimer » et ensuite choisir le ou les objets à supprimer.

Afin de rendre le programme encore plus utile, pour les travaux de groupes par exemple, l’utilisateur doit pouvoir envoyer ou montrer ses constructions. Pour cela, il peut utiliser la dernière icône « Capture d’écran » qui permet d’enregistrer la zone de traçage à l’endroit souhaité sur l’ordinateur, ou bien sauvegarder la construction en cliquant sur « Fichier », puis « Sauvegarder ». Inversement, il peut ouvrir une construction en appuyant sur « Importer/Charger un fichier ».

**Gestion des rayons et des intersections :**

Il est assez compliqué de déterminer la trajectoire du rayon, effectivement l’affichage se fait en « int » tandis que nous travaillions en « double » et de plus nous avons pris beaucoup de temps à trouver une solution adéquate pour déterminer l’intersection entre un rayon et un objet optique. Pour cela nous avons donc utilisé des Line2D pour représenter les rayons mais aussi tous les autres objets optiques tels que les lentilles et miroirs. Cette Line2D avait en attribut deux points qui nous ont permis de déterminer l’équation de la droite de cette line et de déterminer une orientation (celle choisi est du point 1 vers le point 2). Pour l’intersection, nous avons utilisé une formule trouvée sur Wikipédia[[1]](#footnote-1), cette dernière permet de trouver l’intersection de deux droites formées chacune par deux points. Il ne nous restait plus à partir de cette formule de vérifier les intervalles de cette intersection pour vérifier si elles se trouvait bien sur l’objet optique avec lequel elle a intersecté.

Une des étapes la plus importante de notre programme est donc la mise à jour des rayons des sources. Voici le déroulement de cette-dernière :

* Tout d’abord on va parcourir la liste des objets optiques et on ne va traiter que les sources.
* Ensuite, pour chaque source on va parcourir sa liste de faisceaux qui contient tous les faisceaux qui sont dus à cette source et on va aussi parcourir le HashMap de faisceau qui est créé en même temps et qui permet à un faisceau de ne jamais interagir deux fois avec le même objet.
* Pour chaque faisceau, on va chercher les intersections de ce dernier avec tous les objets optiques de type miroir et lentille sauf celui avec lequel elle ne doit pas interagir (parfois aucun). On va éventuellement, trouver plusieurs intersections, dans ce cas on prend la plus proche de l’origine du rayon tout en vérifiant que l’intersection se trouve bien sur l’objet optique. Si on n’a trouvé aucune intersection avec un objet optique, dans ce cas on trouver l’intersection avec une des bordures du JPanel.
* Une fois que l’on a trouvé l’intersection, on remplace le faisceau que l’on traitait par le nouveau allant du premier point jusqu’à l’intersection et ensuite on rajoute un ou plusieurs rayon à la liste dans le cas du miroir, un seul dans le cas d’une lentille et on n’en rajoute pas dans le cas d’une bordure.
* Si un ou plusieurs rayons ont été rajoutés, on continue de parcourir la liste jusqu’à les avoir tous traité.

**Détermination des rayons de réfléchis ou réfractés :**

Pour déterminer le rayon de sortie d’une lentille, on doit déterminer le point d’intersection entre le plan focal de la lentille (du bon côté !) et un faisceau parallèle au premier faisceau passant par le centre optique de la lentille. On connaît alors 2 points de la 2ème partie du faisceau qui sont le premier point d’intersection et l’intersection avec le plan focal du rayon parallèle. On peut donc tracer le rayon sortant. Afin de déterminer ce « bon côté », il est possible d’utiliser le produit scalaire. En effet, en posant que la normale à nos objets optiques est vers la droite, ensuite on peut déterminer le plan focal de la lentille qui nous intéresse. Cela marche pour la lentille convergente, mais pour la lentille divergente, le processus est le même à la différence que le rayon sortant doit être la symétrie du rayon que l’on trouve avec la méthode du plan focal par rapport au point d’intersection du rayon incident. Car sinon pour une lentille divergente le rayon se trouve du mauvais côté.

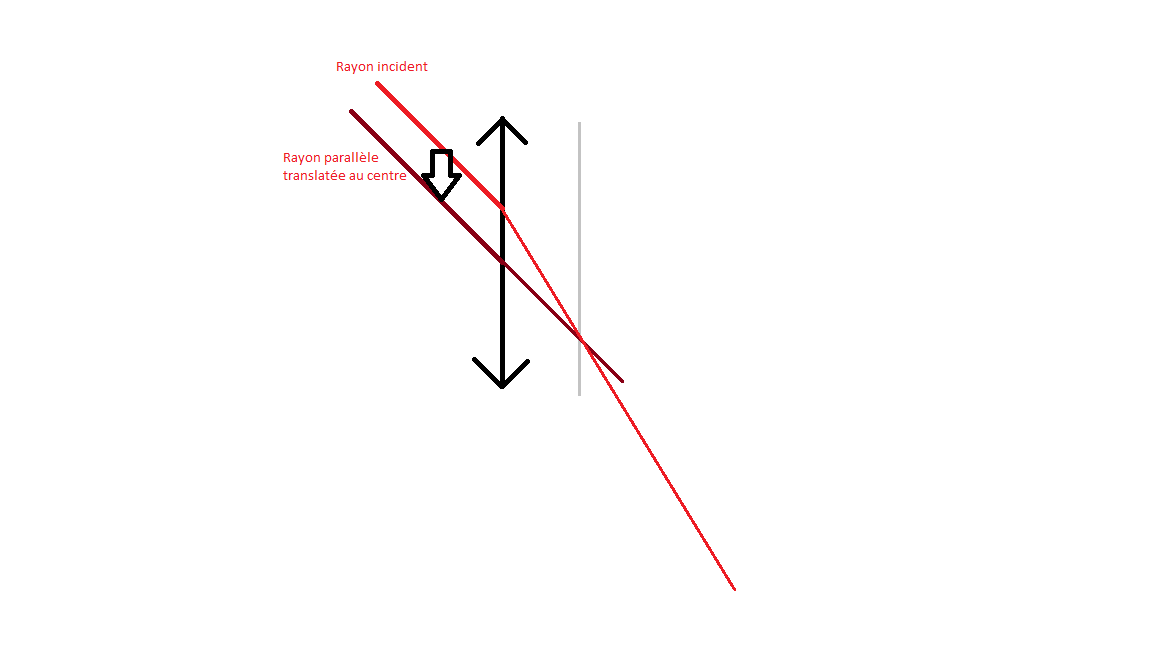


Figure 1. Méthode du rayon parallèle pour une lentille convergente

Pour le miroir, il fallait également déterminer le point d’intersection entre le faisceau et le miroir. Il fallait alors déterminer le symétrique de l’angle d’incidence par rapport à la normale du miroir. Ensuite, aussi bien pour les lentilles que pour les miroirs, il faut considérer la possibilité que le faisceau sortant rencontre à son tour un autre objet optique.

**Sélection des objets, focus et propriétés :**

Pour sélectionner les objets nous utilisons un mouseListener et nous testons si le click de la souris est à moins de 5 pixel d’un objet et si c’est le cas pour plusieurs objets, nous sélectionnons le plus proche. Pour savoir si un objet est sélectionné, il y a une variable de type ObjetOptique qui stock l’objet sélectionné. De plus ces objets ont une variable « focus » de type boolean qui permet de savoir si l’objet a le focus ce qui permet de tracer les deux croix et le carré au centre de l’objet qui sont des repères pour déplacer les objets. Pour les déplacer, nous regardons près de quel point ou si le clic est dans le carré pour déplacer l’objet. Si c’est le cas et que la souris est dragged, on déplace le point sélectionné ou l’objet entier si c’est un clic sur le carré. Lorsqu’un objet est sélectionné, on va appeler la méthode de propriété pour afficher le panel de propriétés correspondant au type d’objet et on va rentrer les valeurs des attributs de l’objets dans ce panel. Cela permet à l’utilisateur de visualiser les attributs de l’objet et de les modifier en rentrant une nouvelle valeur et en appuyant sur entrée. Lorsque que l’on clique dans le vide un resetFocus() est appelé et permet de désélectionné ce qui était sélectionné. On peut donc aussi modifier les attributs en changeant les valeurs dans le panel propriété et en appuyant sur entrée. Les valeurs de l’objet sont donc modifiés et on repaint la zone de traçage.

## La structuration des données

Afin de rendre notre code efficace, nous avons créé une classe mère abstraite « ObjetOptique » car les sources, les lentilles et les miroirs partagent tous les 3 des attributs communs comme une position du centre, un angle avec l’horizontale, une taille et une couleur. Ainsi, grâce au principe d’héritage appris cette année, cela permet d’éviter d’avoir plusieurs classes quasiment identiques.

Pour ce qui est de l’interface graphique, notre « FenetrePrincipale » est la classe qui code la JFrame et les 2 panels existants. La classe « BarreOutils » est la classe qui rassemble les 5 boutons utilisés de la barre.

La classe « Geometrie » rassemble différents calculs ou méthodes nécessaires pour le traçage des rayons. La plus importante de nos classes au niveau du code est « ZoneTracage ». En effet, c’est elle qui rassemble les différents calculs qui permettent de déterminer les points d’intersection et les rayons sortant des objets optiques.

- final double ratio

- JPanel mainPanel

- ZoneTracage panelDessin

FenetrePrincipale()

- JButton btnSelect, btnSource, btnLentille, btnMiroir, btnSuppr, btnScreenshot

+ static ActiveTool activeTool

- Propriete prop

BarreOutils()

+ actionPerformed()

+ static String EXTENSION

- final String description

 OpticusFilter()

+ accept()

+ getDescription()

 - final ImageIcon magicus

- JButton quitter, scrollHaut, scrollBas

- boolean verLeBas

- JscrollPane scroll

- JTextArea textArea

- int i

- boolean running

 Credit()

 + actionPerformed()

+ pause()

+ scroll()

 + static translateLine(),

+ static lineLine(),

+ static produitScalaire()

final Geometrie()

 - double f

- final int TAILLE\_MINIMALE

- Line2D.Double planFocal1, planFocal2

- static int compteNumero

- int numero

- boolean planFoc

 Lentille()

 + movePoint()

+ updatePlanFocal()

+ move()

+ setAffichagePlanFocal()

+ getAfficahegPlanFocal()

+ getFocal()

+ setFocal()

+ setCentreX()

+ setCentreY()

+ setAngle()

+ getNum()

+ draw()

- final double ecart

- static int compteNumero

- int numero

- boolean semiReflechissant

Miroir

+ setSemiReflechissant()

+ getSemiReflechissant()

+ getNum()

+ draw()

- double angle, taille, centrex, centrey

- Color couleur

- boolean focus

- Point2D.Double point1, point2

- Line2D.Double line

- transient JComponent parent

 ObjetOptique()

+ draw()

+ distancePoint()

+ move()

+ pointUpdate()

+ setAngle()

+ setCentreX()

+ setCentreY()

+ movePoint()

+ move()

+ setFocus()

+getCentrex()

+ getCentrey()

+ getTaille()

+ getAngle()

+ getPoint1()

+ getPoint2()

+ setColor()

+setParent()

+ getLine()

- static int compteNumero

- int numero

- ArrayList<Line2D.Double> tabFaisceau

Source()

+ draw()

 + setAngle()

+ movePoint()

+ getNum()

+ setCentreX()

+ setCentreY()

+ move()

+getTabFaisceau()

+ setTabFaisceau()

-int dist

- int location[]

- int cursors[]

ResizableBorder()

 + getBorderInsets()

+ isBorderOpaque()

+ paintBorder()

+ drawRectangles()

+ getCursor()

- JMenu menuFichier, submenu, menuAffiche, menuCredit

- JMenuItem itemImport , itemSave,itemQuit,itemCredit, menuItem2, menuItem3, menuItem4

- JRadioButtonMenuItem rbMenuItem

- JCheckBoxMenuItem cbMenuItem

- ZoneTracage panelDessin

Toolbar()

- final class ResultIntersectionAvecObjetOptique()

- static final int IFW

- Propriete prop

- int cursor

- ObjetOptique selectedObject

- Point2D selectedPoint, positionningPoint1, positionningPoint2

-ArrayList<ObjetOptique> listeObjet

- Line2D.Double postionningLinr

- Jframe parentFrame

- Line2D[] bordures

ZoneTracage()

 + mouseMoved()

+ mouseExited()

+ mouseClicked()

+ mouseEntered()

+ mousePressed()

+ mouseDragged()

 + mouseReleased()

+ resetFocus()

+ paintComponent()

+ updateInteraction()

+ intersectionAvecObjetOptique()

+ addObjetOptique()

+ removeObjetOptique()

+ setPropriete()

+ getSelectedObject()

+ getListObjetOptique()

+ setListObjetOptique()

Extends

Extends

Extends

- JLabel nomOutil, changerCouleur, description, labelFocal, labelTaille, labelX, labelY, labelAngle

- JCheckBox boxPlans, boxSemiReflet

- JTextField entreX, entreY, entreTaille, entreAngle, entreFocal

- JButton btnValider,btnCouleur, btnSupprimer

- ZoneTracage panelDessin

- Color couleurChoisi

- Font font1

Propriete()

 + propSource()

+ propLentille()

+ propSelect()

+ propMiroir()

+ propSuppr()

+ propScreenshot()

+ getEntreFocalValue()

+ setZoneTracage()

+ actionPerformed()

uses

uses

uses

uses

uses

uses

uses

uses

uses

1..\*

1

1..\*

1

0..\*

1

uses

1

1

1

1

1

1..\*

1..\*

1

1..\*

1

Schéma de la hiérarchie des classes :

## Problèmes connus et pistes d’amélioration

Le programme a pour objectif de visualiser le trajet des faisceaux de lumières à travers différents objets optiques, c’est pourquoi la notion d’équilibre d’intensité des rayons incidents, réfléchis et transmis (pour les miroirs semi-réfléchissants) est négligée dans cette version du programme.

D’autre part, notre programme fonctionne avec des sources représentées par des rayons uniques. On peut donc imaginer des sources ponctuelles émettant de nombreux rayons ou bien des « murs » de lumière, c’est-à-dire plusieurs rayons tous parallèles.

De même, les couleurs que l’on utilise ne servent qu’à éviter d’embrouiller l’utilisateur. Cependant, nous pouvons que notre programme prenne en compte les longueurs d’onde des faisceaux et ainsi adapter leurs comportements. Ainsi, il sera possible de créer d’autres objets optiques tels que les prismes.

Un bug connu est que si on arrive à faire en sorte que le point d’intersection d’une source soit exactement sur le point de départ du rayon, le rayon d’une source devient un point seul et ne redevient un rayon que si on réactualise la source (en la déplaçant ou en déplaçant un des points de la line de base.

Un second problème est que quand on met deux miroirs parallèles l’un en face de l’autre et une source perpendiculaire entre les deux, le programme va bugger car il va créer un nombre illimité de rayons sans jamais s’arrêter. Une solution serait de mettre une limite de nombre de rayons.

## Conclusion

Pour conclure, ce projet nous a permis de mener le développement d’un programme à partir de l’élaboration d’un cahier de charges jusqu’à son rendu. Ce projet nous a donc permis de mettre en application les connaissances pluridisciplinaires acquises durant notre formation à l’INSA afin de répondre à un besoin, ce qui constituera une part de notre futur métier d’ingénieur. Plus généralement, ce projet nous a permis d’améliorer nos compétences, aussi bien en informatique que des compétences nécessaires pour un bon ingénieur tels que le travail d’équipe, et une recherche efficace de solution au problème posé.

1. <https://en.wikipedia.org/wiki/Line%E2%80%93line_intersection> [↑](#footnote-ref-1)