RAPPORT D'INGÉNIEUR
PROJET DE DEUXIÈME ANNÉE
FILIÈRE F4 : CALCUL ET MODÉLISATION SCIENTIFIQUES

Intégration du modèle de caméra plénoptique dans un logiciel de rendu

Présenté par : Sandy LE PAPE et Florent PAPINI

Tuteur de projet :
Charles-Antoine NOURY
Professeur référent :
Christophe TILMANT

Date de la soutenance :
le mardi 20 mars 2018
Durée du projet :
60h chacun

Année scolaire 2017-2018

Remerciements

Nous aimerions tout d'abord remercier notre tuteur de projet, Charles-Antoine NOURY, pour avoir eu la gentillesse de se montrer disponible et à notre écoute. Il a su nous aiguiller au fur et à mesure du projet afin de nous aider à progresser dans notre travail. Nous aimerions également remercier tout particulièrement Céline TEULIERE, qui a su nous éclairer sur les connaissances 3D nécéssaires pour la pleine compréhension du sujet. Nous remercions également Christophe TILMANT en tant que professeur référent de ce projet. Enfin, nous tenions aussi à remercier Loïc YON et Gilles LEBORGNE, qui nous ont aidés, pour le premier à nous sortir d'une difficulté dans le code et pour le second brièvement pour la mise en place des équations d'optique. Enfin, nous aimerions remercier l'ISIMA, qui nous a permis de travailler de longues heures au sein de ses locaux, ainsi que l'Institut PASCAL, dans lequel ont eu lieu de nombreuses réunions de travail avec notre tuteur.

Table des illustrations

Résumé

Le but de ce projet est d'implémenter la **caméra plénoptique** dans le **logiciel de rendu POV-Ray**. Ce logiciel crée les objets de la scène par **tracé de rayons**. Le code **C++**, que l'on doit compléter pour ce type de caméra, est disponible en Open-Source sur un dépôt **GitHub**. Pour générer une image, il est nécessaire d'écrire un fichier de script dans le langage propre à POV-Ray décrivant la géométrie et les volumes présents dans la scène et d'exécuter le code C++ s'occupant de la génération de la scène codée dans le script.

À ce jour, le modèle de caméra plénoptique semble fonctionnel, mais ne peut être modifié aisément par un fichier de configuration des paramètres.

Mots-clés : caméra plénoptique, logiciel de rendu, POV-Ray, tracé de rayons, C++, GitHub.

Abstract

The goal of this project is to implement the **plenoptic camera** in the **rendering software POV-Ray**. This software creates the objects of the scene by **ray-tracing**. The **C++** code, which has to be fitted with this type of camera, is available in Open-Source on a **GitHub** repository. To generate an image, it is necessary to write a script file in the language specific to POV-Ray describing the geometry and the volumes in the scene and to execute the C++ code dealing with the scene generation coded in the script.

Currently, this model of plenoptic camera seems functional, but cannot be modified easily by a parameters configuration file.

Keywords : plenoptic camera, rendering software, POV-Ray, ray-tracing, C++, GitHub.

Table des matières

Introduction

Ce projet nous a été confié par l'Institut PASCAL, situé sur le Campus des Cézeaux, aux côtés d'écoles d'Ingénieurs telles que l'ISIMA, cette dernière faisant maintenant partie de l'Institut d'Informatique d'Auvergne.

L'Institut Pascal est un laboratoire de recherche ayant une ferme volonté de développer des systèmes novateurs via une approche multidisciplinaire à prédominance physique. Il propose notamment à des étudiants la possibilité de se spécialiser autour d'un axe majeur de développement de l'Insitut.

C'est au sein de l'axe de recherche *Image, Systèmes de Perception, Robotique* que nous a été confié le projet. Cette filière de l'Institut approfondit l'étude de la caméra plénoptique, qui a la capacité de capturer une image 3D. L'essentiel du travail ayant précédé le nôtre sur cette caméra s'est situé au niveau de son calibrage, utilisant directement des images brutes, plutôt que d'utiliser la méthode classique de reconstruction de l'image [7].

Il y a maintenant un réel besoin de visualiser l'image que l'on pourrait obtenir avec une telle caméra. La caméra étant relativement peu développée encore aujourd'hui, très peu de logiciels de rendus 3D sont munis de cette dernière.

Jusqu'à présent, la visualisation d'images de caméra plénoptique ne pouvait se faire via un logiciel de tracé de rayons. Le logiciel au sein duquel sera intégré l'implémentation de la nouvelle caméra est POV-Ray.

Afin de présenter au mieux l'un des points clés qu'est le tracé de rayons, nous devrons nous pencher tout d'abord sur des concepts fondamentaux d'infographie. Il conviendra ensuite d'exposer les outils et méthodes utilisés en amont de l'implémentation. Enfin, nous présenterons l'obtention de différents rendus en faisant varier différents paramètres.

Chapitre 1

Contexte du projet

1.1 Principe du logiciel

1.1.1 Qu'est-ce qu'un logiciel de rendu ?

Un logiciel de rendu est un outil informatique permettant la construction puis la visualisation réaliste d'une scène grâce à un code écrit par l'utilisateur. Ce type de logiciels calcule une ou plusieurs images 3D, permettant la restitution fidèle de la projection 3D ainsi que tous les effets de lumière.

La plupart de ces logiciels de rendu disposent habituellement d'un logiciel de modélisation 3D intégré, plus couramment appelé modeleur 3D, dans lequel on crée la à la main (mise en place de l'éclairage, géométrie des objets,...). Le principe de ces logiciels de rendu est de générer une image réaliste à partir de ces informations et réglages fixés dans le logiciel de modélisation intégré : cette technique est appelée synthèse.

POV-Ray ne dispose pas de modeleur 3D intégré : les modèles ainsi que les éléments indispensables à la création d'une scène doivent donc être décrits dans un script. Il est cependant possible d'utiliser un modeleur dédié spécifiquement à POV-Ray ou d'importer un fichier sous le format exigé par le logiciel.

POV-Ray est avant tout un logiciel de tracé de rayons.

1.1.2 POV-Ray, un logiciel de tracé de rayons

Afin de générer une image réaliste de la scène créée, les logiciels de rendu utilisent différentes techniques de calcul.

En guise d'introduction, rappelons tout d'abord la manière dont l'Homme visualise son environnement extérieur. L'œil est doté d'une rétine, récepteur de lumière équivalent au capteur d'un appareil photographique. Tout humain peut voir un objet à condition que la rétine soit éclairée par la lumière émise par ce dernier (cf FIGURE 1.1). La plupart des logiciels de rendu utilisent ce mode de pensée, notamment ceux utilisant la rastérisation.

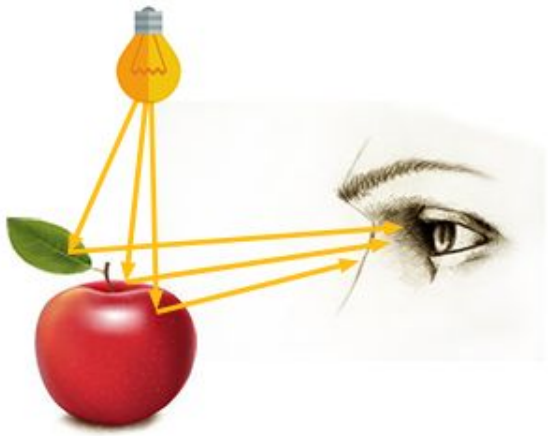


FIGURE 1.1 – Vision d'une image par l'œil humain [2]

On distingue 2 techniques principales : la rastérisation (*rasterization* en anglais) et le tracé de rayons (*ray-tracing* en anglais).

Le processus de rastérisation est couramment utilisé dans ce type de logiciels : il consiste à convertir une image que l'on souhaite construire (souvent transformée en amont sous forme d'une primitive, i.e. une forme géométrique basique) en une représentation matricielle 2D. Chaque point de l'image (correspondant à une case de la matrice) contiendra une certaine quantité d'informations importantes telle que la couleur ou la profondeur par exemple. Il convient alors de déterminer quelles sont les cases qui sont occupées par la primitive (cf partie supérieure FIGURE 1.2).

Un logiciel de tracé de rayons, en revanche, tel que POV-Ray prend le contrepied de cette vision. En effet, cette méthode de calcul consiste, à partir d'un pixel de l'image que l'on souhaite créer, à calculer la trajectoire de chaque rayon lumineux rencontrant l'objet virtuel de la scène (cf partie inférieure FIGURE 1.2).

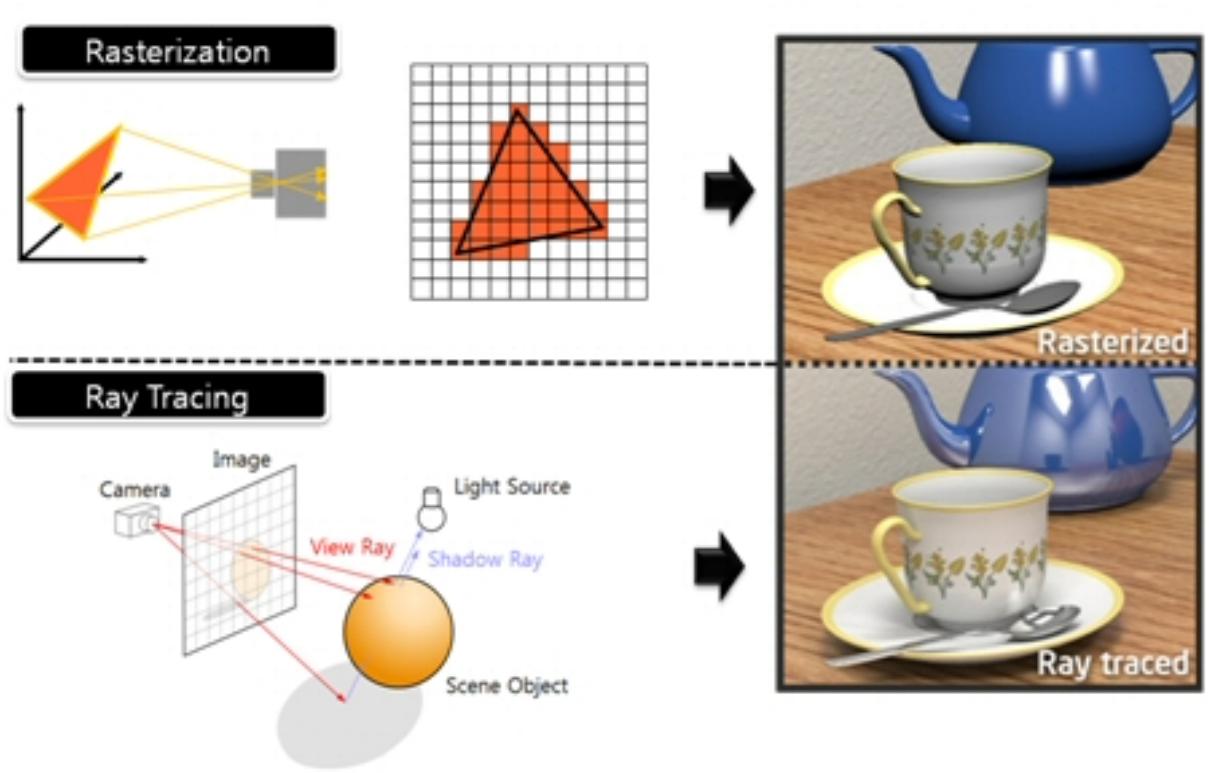


FIGURE 1.2 – Comparatif de différentes techniques de calcul utilisées par les logiciels de rendus

Le principal avantage d'un tel logiciel est de pouvoir créer une image d'une qualité nettement supérieure à celle obtenue par un logiciel de rendu classique.

Pour comprendre comment fonctionne un tel logiciel, il est d'abord nécessaire de comprendre dans quel domaine scientifique il s'inscrit.

1.1.3 Qu'est-ce que l'Infographie ?

On définit communément ce large domaine qu'est l'Infographie comme l'ensemble des techniques utilisées en Informatique pour créer et traiter des images numériques. Naturellement, l'Infographie 3D a pour but de représenter des volumes en perspective dans un espace à 3 dimensions.

Ainsi, le fonctionnement même de ce logiciel se repose sur des notions basiques de cette activité qu'il est nécessaire de définir afin de comprendre pleinement les objets que nous allons manipuler par la suite.

1.2 Notions élémentaires d'Infographie

1.2.1 Les 3 éléments indispensables à une scène 3D

Une Scène 3D doit comporter 3 éléments fondamentaux (cf figure FIGURE 1.3) afin d'être visualisée correctement :

- **la caméra** : c'est à partir de son emplacement que le logiciel va générer le rendu de notre scène. Il faudra donc s'assurer du cadrage de la scène, donc que la direction dans laquelle elle pointe soit sur l'objet construit ;

- **la source de lumière** : elle sert de manière évidente à éclairer la scène visualisée (on se contentera d'une seule source dans notre scène ; exemple avec plusieurs sources de lumière en FIGURE 1.3).

- **l'objet 3D à visualiser** (une pyramide de sphères en FIGURE 1.3).

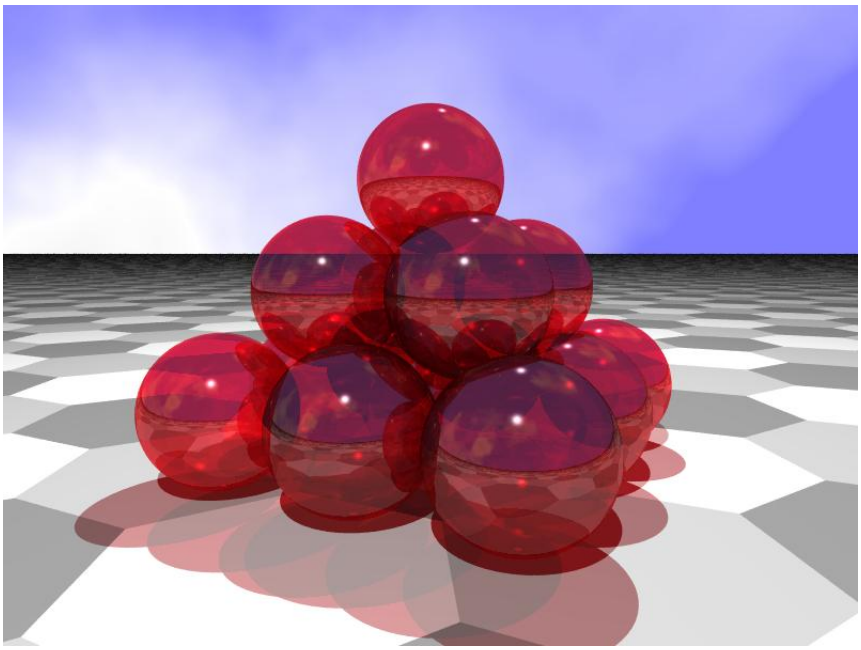


FIGURE 1.3 – Rendu généré par POV-RAY [4]

L'ensemble du projet se concentrera principalement sur l'objet *caméra* de la scène car le but du projet est de coder la caméra plénoptique. Il faut maintenant pouvoir connaître la position de cette caméra par rapport aux autre objets de la scène, d'où l'introduction de la notion de repères.

1.2.2 Repères associés à ces éléments

Chaque scène 3D créée est associée à un repère général : on l'appelle repère de la scène 3D ou encore repère monde. Très souvent, il correspond au repère de base dans lequel s'effectue la plupart des calculs. Ce repère contient tous les autres repères existants.

Chaque élément de la scène a son repère associé, à savoir la caméra, la source de lumière et l'écran. Il est indispensable dans la mesure où chaque élément de la scène va être localisé par des coordonnées exprimées par rapport à ce repère.

Il sera souvent nécessaire d'effectuer des calculs de changements de base, ie des calculs permettant de passer de l'expression des coordonnées d'un repère à leur expression dans un autre (cf FIGURE 1.4). Il sera question dans ce projet de trouver la relation entre les coordonnées spatiales d'un point dans la scène 3D avec le point associé dans l'image prise par la caméra : cette opération est appelée calibrage (*calibration* en anglais). L'opération effectuée utilise une formalisation mathématique matricielle qui sera explicitée plus tard.

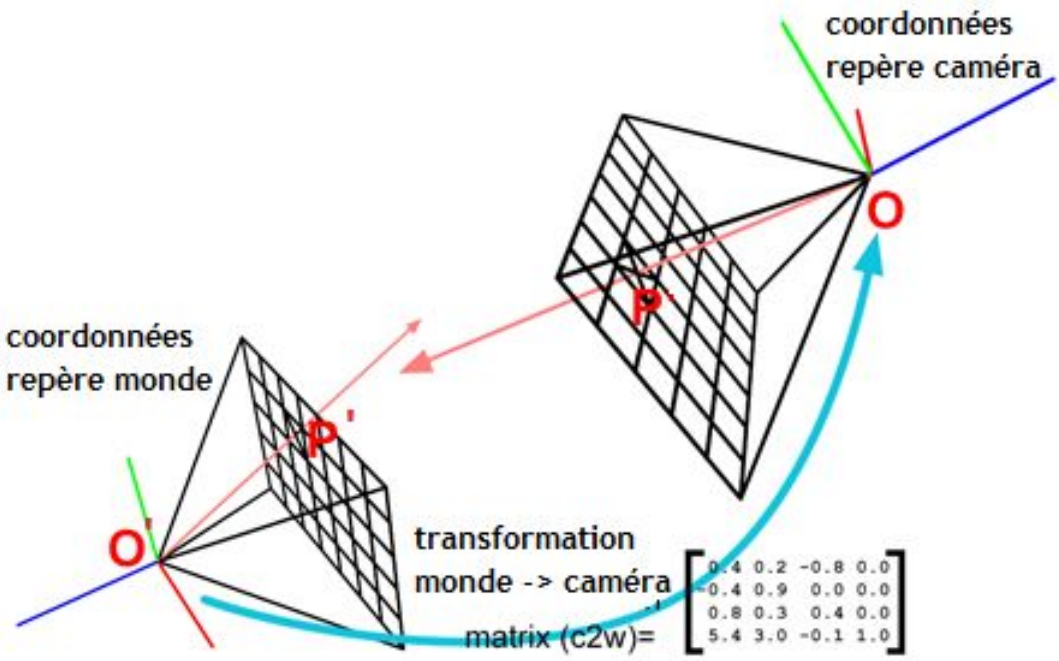


FIGURE 1.4 – Passage du système de coordonnées de la scène 3D à celui de l'écran [2]

Dans POV-RAY, le repère de la caméra (a fortiori celui de la caméra plénoptique) est orthogonal et correspond à (O, cameraRight, cameraUp, cameraDirection) (cf FIGURE 1.5). Le repère de l'écran est orthonormé entre -0.5 et +0.5 unités. Le repère de la lumière et celui d'un objet de la scène 3D ne seront pas utilisés dans ce projet.

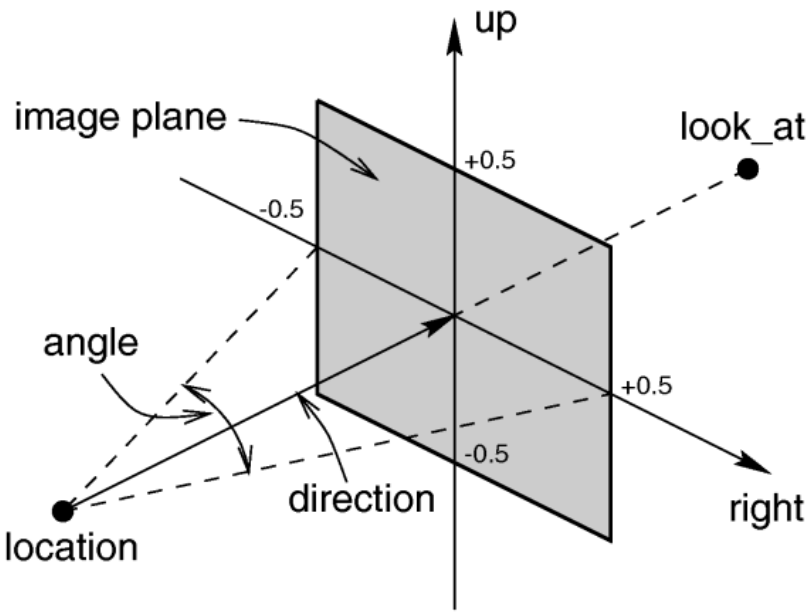


FIGURE 1.5 – Placement et repère associés à la caméra dans POV-RAY [1]

Maintenant que nous connaissons les différents éléments que comporte une scène 3D et que nous pouvons les repérer dans l'espace, étudions la manière de tracer les objets 3D dans la scène.

1.2.3 Tracé des éléments 3D de la scène

Dans POV-RAY comme dans tout logiciel de tracé de rayons, un rayon, contribuant à la création d'un pixel de l'image numérique, est définit par une origine (que l'on notera O) et une direction (souvent normalisée que l'on notera R) (cf FIGURE 1.6).

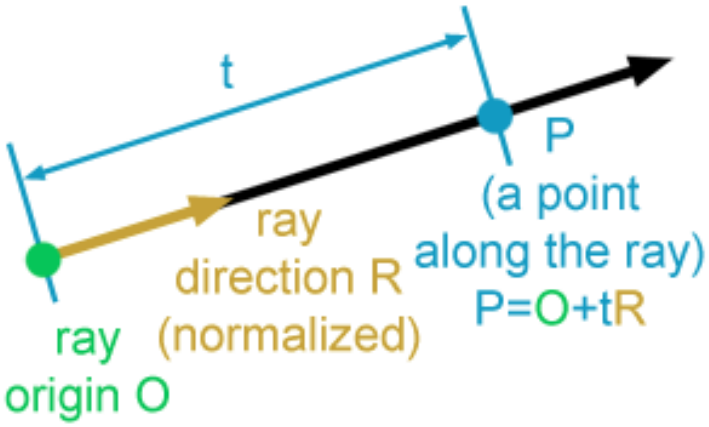


FIGURE 1.6 – Modélisation d'un rayon dans un logiciel de rendu [2]

La formalisation mathématique de l'expression d'un rayon est :

$$P = O + t \times R$$

où t est la longueur entre l'origine et le point d'intersection du rayon avec l'objet virtuel.

Ces rayons tracés vont nous être utile pour résoudre les problèmes de visibilité et rassembler les informations à propos des couleurs et des effets d'ombre de l'objet. Nous allons donc envoyer un rayon : s'il vient à intersecter l'objet virtuel, on calcule alors sa couleur au pixel correspondant (cf FIGURE 1.7). Ce type de rayons, lançés depuis l'origine du repère de la caméra, sont nommés rayons primaires (*primary rays*), aux côtés de rayons s'occupant davantage des effets d'ombre, de diffusion,... nommés rayons secondaires (pas l'objet de ce projet).

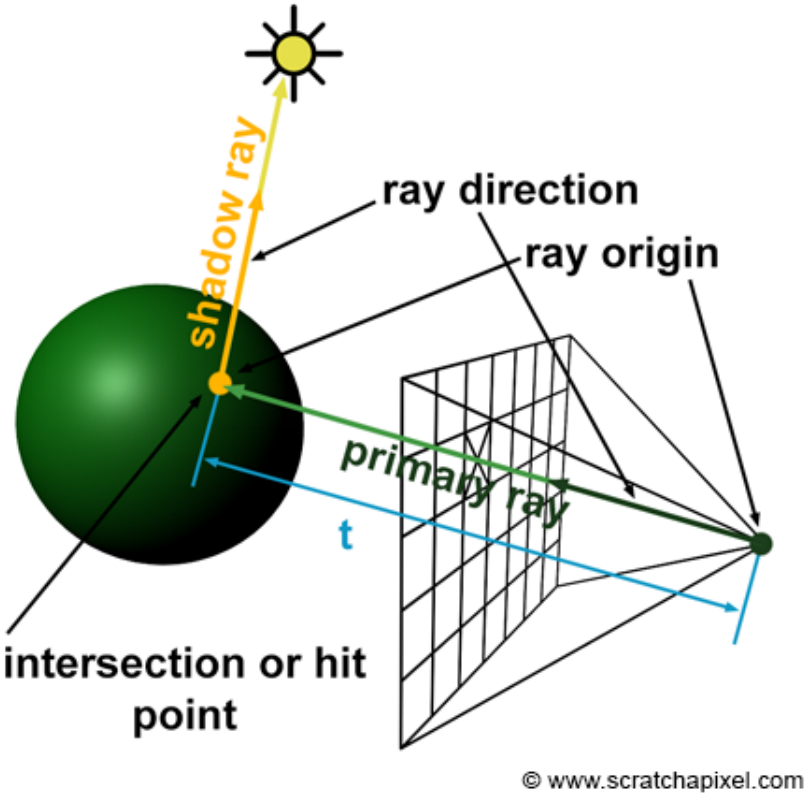


FIGURE 1.7 – La notion de rayon primaire pour le tracé de rayons [2]

Penchons-nous maintenant sur la façon de coder une telle scène 3D et toutes ses composantes dans le logiciel utilisé.

1.3 Création d'une scène dans POV-Ray

1.3.1 Création d'une scène 3D

Sous POV-Ray, la création d'une scène 3D se décompose en 3 parties correspondant aux 3 parties principales d'une scène 3D.

L'objet *caméra* se code via l'utilisation de 2 vecteurs (cf FIGURE 1.3) :

- le vecteur **location** $< x, y, z >$ où x, y et z représentent les coordonnées du centre de la caméra dans la scène 3D,

- le vecteur **look_at** $< x, y, z >$ représentant la direction dans laquelle regarde la caméra.

L'objet *light_source* s'initialise différemment : les premières coordonnées du vecteur indiquent la position de la source lumineuse dans la scène (pas de direction requise). On indiquera également la couleur de cette lumière.

Enfin, il est possible de mettre une couleur au fond d'écran : nous l'avons colorié en gris dans l'image de référence. Mais qu'en est-il des volumes géométriques de l'objet à créer ?

1.3.2 Code des formes géométriques

Il est courant d'introduire un plan dans la scène, de manière à y "déposer" l'objet d'étude. Nous l'avons mis dans le plan (O,y,z) de la scène.

Il serait inutile de détailler le code de chaque solide géométrique introduit dans la scène : leur implémentation dépend des propriétés de la forme considérée (un diamètre sera introduit pour les objets circulaires par exemple,...)

Nous obtenons ainsi, par l'utilisation de POV-Ray, une image, qui sera une image de référence (cf Figure 1.8) pour toute la suite du projet. Elle servira à juger de la cohérence de l'implémentation de nos futurs modèles : c'est le seul moyen de vérifier la justesse de notre modèle.

Une fois tous ces éléments d'infographie maîtrisés, le problème peut être davantage précisé.

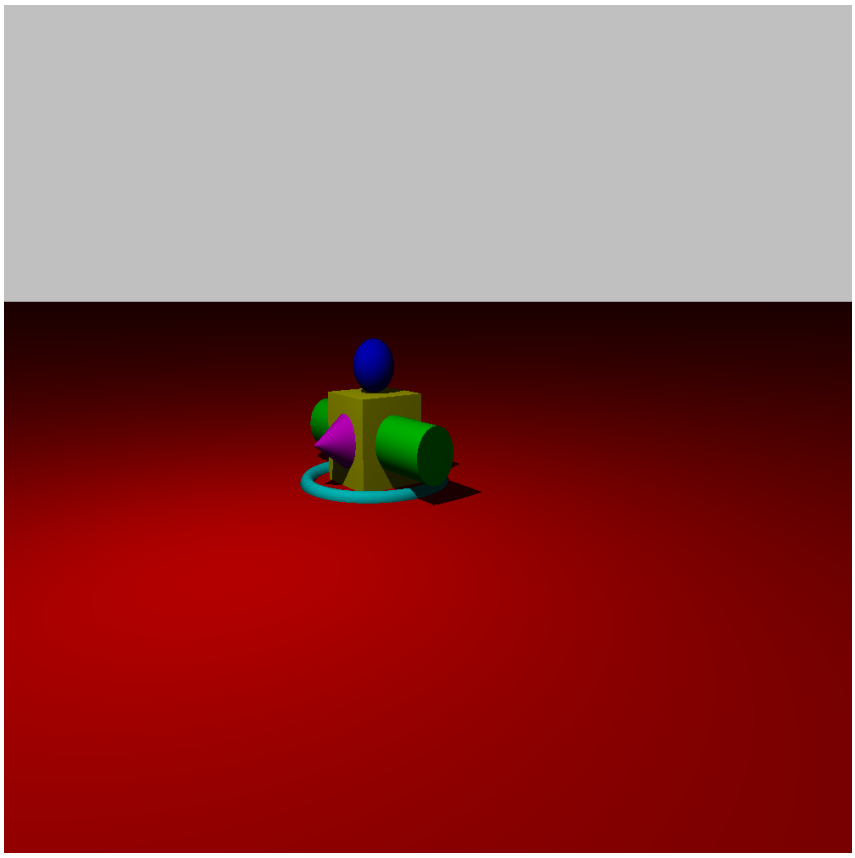


FIGURE 1.8 – Rendu du modèle sténopé [5]

1.4 Analyse du problème

Notre projet étant directement lié à la caméra plénoptique, il est nécessaire de connaître les différents éléments qui la compose et comprendre son fonctionnement si on veut epsérer pouvoir l'implémenter.

La caméra plénoptique est un appareil photographique numérique qui a la capacité de capturer une image 3D. En effet, contrairement à un appareil traditionnel mesurant uniquement l'intensité lumineuse en chaque point du capteur photosensible (2D), cette caméra mesure également la direction empruntée par chaque rayon lumineux dont le point d'incidence se trouve sur le capteur, permettant l'ajout de l'information de relief 3D à la scène photographiée.

Cela s'effectue via l'introduction d'un *light field*, un vecteur à quatre composantes représentant la répartition de la lumière dans l'espace 4D, les 2D du plan du capteur et les 2D du plan de la lentille. À l'issue d'une seule prise, on récolte donc une quantité d'informations importante, qui aurait été perdue par une photographie classique. Cette caméra permet, à elle seule, de remplacer un système coûteux et encombrant (de multiples caméras disposées en différents endroits d'une scène) par une unique caméra.

Cette caméra est composée de 3 principaux éléments : un capteur photosensible, une matrice de microlentilles et une lentille mince principale (cf FIGURE 1.9 ET 1.10).

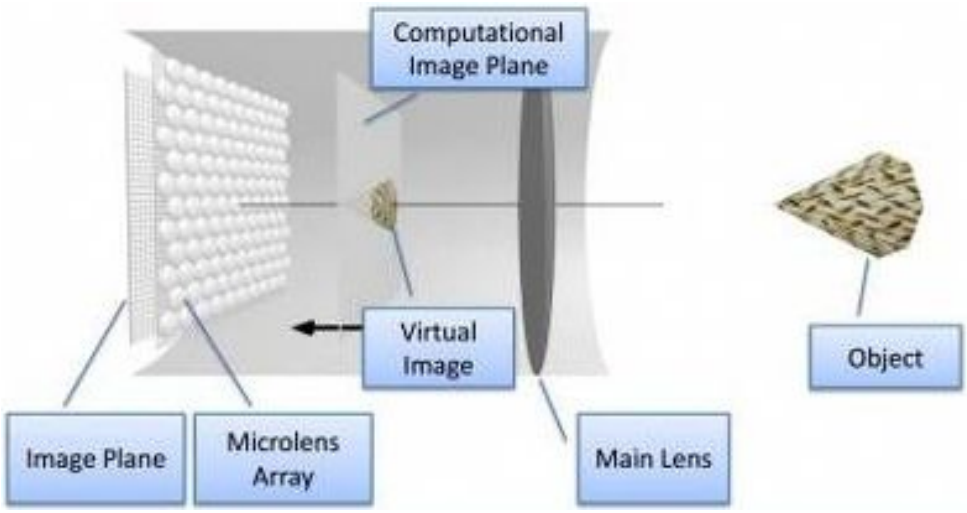


FIGURE 1.9 – Les différentes composantes de la caméra plénoptique

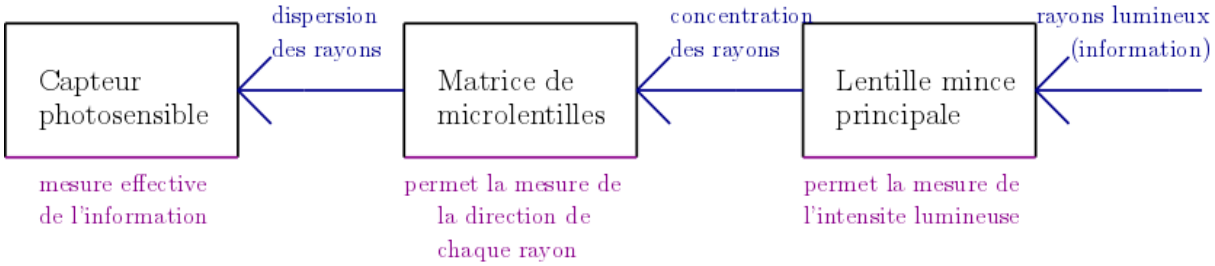


FIGURE 1.10 – Schéma fonctionnel associé

La capacité de capturer la direction des rayons lumineux d'une scène est possible grâce au système de matrice de microlentilles. Les microlentilles dispersent les rayons afin de fragmenter l'image sur le capteur et ainsi, obtenir une multitude d'images prises de points de vue différents. Ces images sont ensuite traitées informatiquement afin de reconstituer la véritable scène lors du rendu.

La capacité de canaliser les rayons dans le système de microlentilles est apporté par la lentille principale.

Voici une image du rendu final (cf FIGURE 1.11) à obtenir pour une caméra plénoptique classique (un zoom désigné par l'encart vert permet de mieux voir ce que contient l'image).

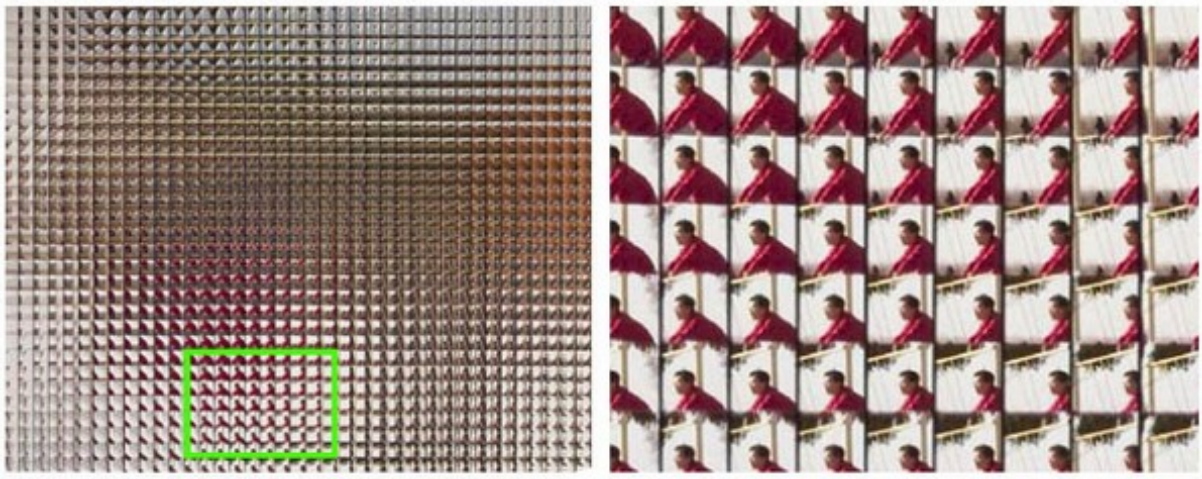


FIGURE 1.11 – Rendu du modèle plénoptique et zoom [10]

La caméra plénoptique est utilisée à grande échelle dans les domaines tels que l'imagerie 3D, en plein développement, l'industrie d'aide visuelle 3D pour les robots, ou encore le contrôle de processus industriels.

L'essentiel de notre travail va donc se résumer à trouver l'expression du rayon primaire partant de la caméra plénoptique qui va permettre de créer l'ensemble des pixels sur l'image de l'objet photographié que l'on va visualiser sur le logiciel. Avant de trouver l'expression de ce rayon, cela nécessite de déterminer son origine précise, ce qui va nécessiter plusieurs changements de repères.

Avant de se lancer tête baissée dans l'implémentation du système optique du modèle plénoptique, nous allons traiter les systèmes optiques unitaires simples des ses différents composantss. Une fois leur implémentation effectuée, leur concaténation va permettre d'obtenir le modèle final.

Vous trouverez le cahier des charges en annexe 1 de ce rapport.

Chapitre 2

Réalisation et conception

Avant de commencer cette chapitre, nous vous invitons à consulter les 2 diagrammes de Gantt résumant l'avancée prévue du projet et l'avancée réelle. Vous trouverez ces documents en annexe 2 du rapport.

2.1 Modèle du sténopé

2.1.1 Principe

Le sténopé (appelé *pinhole camera* en anglais) est un système optique simple constitué d'une boîte fermée (plus communément appelée chambre) dont l'un de ses côtés est perçé d'un petit trou (cf FIGURE 2.1). Afin d'éviter toute réflexion lumineuse, seul le petit trou doit laisser passer la lumière (il est donc d'usage de recouvrir la surface intérieure de la boîte d'un matériau mat et noir).

La propagation rectiligne de la lumière via ce petit trou permet la création d'une image de l'environnement extérieur sur le côté opposé. En effet, chaque point de la surface d'un objet reflète les rayons de la lumière qui l'éclaire dans toutes les directions. Le trou perçé dans la boîte en laisse passer un certain nombre, dont le point d'impact appartiendra au plan de projection intérieur à la boîte, créant une image renversée.

Parmi les principales caractéristiques de ce modèle, nous pouvons citer :

- l'obtention d'une image d'une perspective parfaite, du fait de la projection centrale

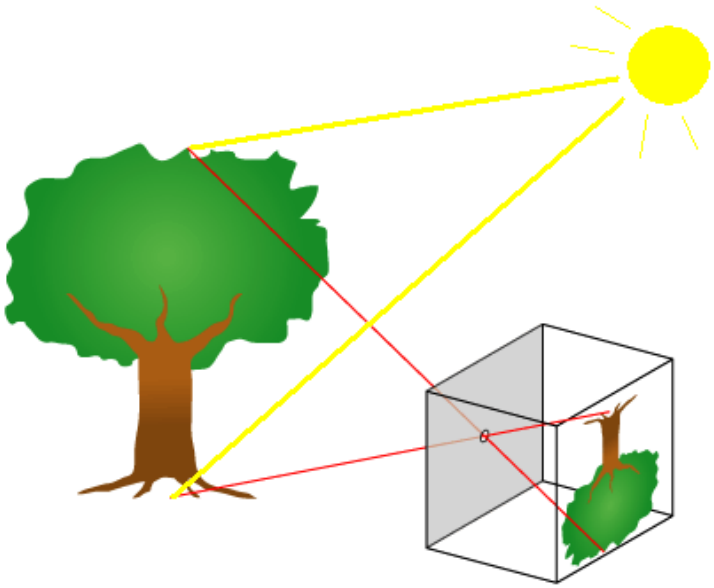


FIGURE 2.1 – Principe d'un sténopé [9]

obtenue grâce au petit trou ;

- une profondeur de champ infinie, permettant l'obtention d'une netteté identique selon que l'on se place loin ou près de l'objet. Rappelons que la profondeur de champ correspond à la zone de l'espace dans laquelle doit se trouver l'objet photographié de telle sorte à ce que l'image de celui-ci soit considérée nette par l'oeil humain.

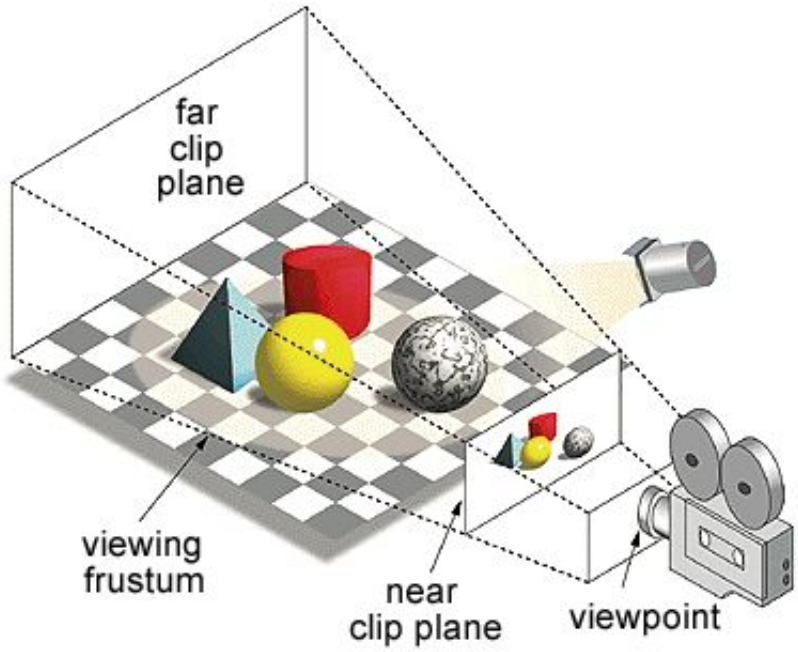


FIGURE 2.2 – Modélisation 3D de la caméra perspective (sténopé) [8]

Le principal inconvénient de ce dispositif est de trouver le diamètre idéal du trou tel que la diffraction n'opère pas trop (trou trop petit) et que l'image ne soit pas trop trouble ou trop lumineuse (trou trop grand).

2.1.2 Modélisation et mise en équation

Comme expliqué à la sous-section précédente, la propagation de la lumière est rectiligne dans ce modèle : un rayon sera donc représenté sous forme d'une droite orientée de la source de lumière (objet) vers l'image (cf FIGURE 2.3). Le repère de référence est celui de la caméra, d'origine O. On note $(x_p^{'}, y_p^{'}, z_p^{'})$ les coordonnées d'un pixel dans l'espace 3D et (x_p, y_p, z_p) les coordonnées du pixel correspondant sur le capteur.

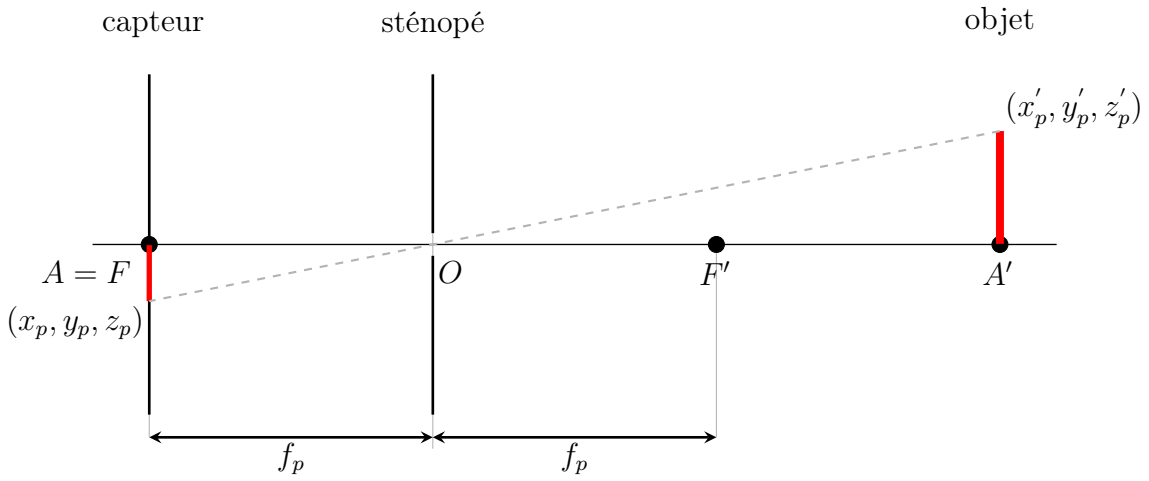


FIGURE 2.3 – Schématisation du modèle sténopé

Il est donc nécessaire de trouver un relation de passage qui permette de convertir un point (pour nous un pixel) exprimé dans le repère monde à un point exprimé dans le repère du plan image (calibrage). Cette transformation nécessite plusieurs transformations intermédiaires que voici :

coord repère monde → coord repère caméra → projection repère caméra dans plan image → coord repère image via tranf affine

La représentation matricielle du problème donne ([3]) :

$$\begin{pmatrix} x_p \\ y_p \\ z_p \end{pmatrix} = \begin{pmatrix} k_u & 0 & c_u \\ 0 & k_v & c_v \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} f_p & 0 & 0 & 0 \\ 0 & f_p & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \times \left(\begin{array}{ccc|c} & R_{axe}(\theta) & & \begin{matrix} t_x \\ t_y \\ t_z \end{matrix} \\ \hline 0 & 0 & 0 & 1 \end{array}\right) \times \begin{pmatrix} x_p^{'} \\ y_p^{'} \\ z_p^{'} \\ 1 \end{pmatrix}$$

où f_P est la focale du sténopé, k_u, k_v correspondent aux facteurs d'agrandissement de l'image, $(c_u, c_v)^T$ le vecteur contenant les coordonnées du centre optique de la caméra projeté sur le plan image, $(t_x, t_y, t_z)^T$ (resp. $R_{axe}(\theta)$) contenant les informations de translation(s) (resp. rotation(s)) du passage du repère monde au repère caméra.

N.B. : on remarque ici l'utilisation de coordonnées homogènes (ajout d'un 1 en quatrième composante du vecteur pixel capteur) afin de conserver la justesse du produit matriciel.

La mise en équation se réduit à l'écriture d'équations de droites selon cameraRight (x) et cameraUp (y) dans le repère de la caméra :

$$f(x) = ax + b \;=\; \frac{x_p^{'}}{2f_p}x \;=\; \frac{x_p}{f_p}x \quad \text{et} \quad f(y) = ay + b \;=\; \frac{y_p^{'}}{2f_p}y \;=\; \frac{y_p}{f_p}y$$

N.B. : l'ordonnée à l'origine est ici commune à toutes les droites (en réalité, le petit trou percé se modélise par un point), d'où b = 0.

2.2 Modèle à une lentille mince

2.2.1 Principe et schématisation

La lentille mince est un composant présent dans de nombreux systèmes optiques classiques, permettant à tout rayon lumineux la traversant d'être réfraction. Pour un pixel de l'image en A, il existe une infinité de rayons qui, en passant par la lentille, convergent en un point précis formant le pixel correspondant de l'objet en A' (cf FIGURE 2.4).

On qualifie de mince la lentille lorsque son épaisseur est négligeable devant le rayon de courbure de ses faces : cela permet de négliger la distance entre les 2 dioptres (que l'on représente sur le schéma pour distinguer le composant) et nous considérons donc un seul point O au centre de la lentille. Les rayons passant par le centre de la lentille ne sont pas déviés (fonctionnent comme le sténopé), du fait de l'hypothèse précédente et les autres passent soit par le foyer objet F, soit par le foyer image F' (cf les traits pointillés FIGURE 2.4).

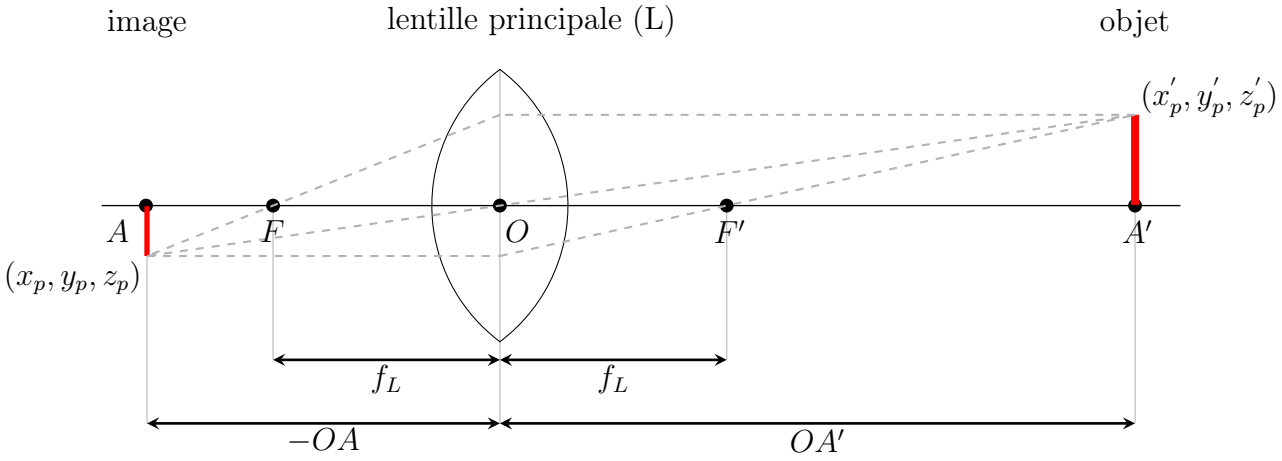


FIGURE 2.4 – Schématisation du modèle à une lentille mince

2.2.2 Comparaison avec le modèle sténopé

Le modèle à une lentille mince diffère du modèle sténopé lorsque l'on regarde le résultat d'une projection dans chacun d'entre eux. Le modèle sténopé ne renvoie qu'une image inversée réduite de l'objet tandis que le modèle à une lentille mince renvoie une image inversée avec un grossissement relatif aux caractéristiques de la lentille.

La notion de distance focale nous permet de calculer la position ainsi que la dimension de l'image. Cependant, elle n'est pas toujours visible selon la position de la focale par rapport à l'objet et le rendu peut être flou.

On remarquera que le modèle sténopé n'est rien de plus que le modèle à une lentille mince où seuls les rayons passant par le centre (ie non déviés) sont considérés.

2.2.3 Mise en équation

Le but est donc de trouver l'expression mathématique reliant la position d'un pixel de l'espace 3D à sa position correspondante sur le capteur (calibrage). En supposant que la lentille mince ne puisse subir de transformation linéaire, on trouve :

$$\begin{pmatrix} x_p \\ y_p \\ z_p \\ 1 \end{pmatrix} = \begin{pmatrix} k_u & 0 & c_u \\ 0 & k_v & c_v \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} f_L & 0 & 0 & 0 \\ 0 & f_L & 0 & 0 \\ 0 & 0 & F_L & 0 \end{pmatrix} \times \left(\begin{array}{ccc|c} & R_{axe}(\theta) & & \begin{matrix} t_x \\ t_y \\ t_z \end{matrix} \\ \hline 0 & 0 & 0 & 1 \end{array}\right) \times \begin{pmatrix} x'_p \\ y'_p \\ z'_p \\ 1 \end{pmatrix}$$

On va appliquer la formule de conjugaison pour exprimer le pixel de l'espace 3D en fonction de celui du capteur : elle ne peut s'appliquer que pour les lentilles minces.

On a, d'après la relation de conjugaison (en prenant les notations de la FIGURE 2.3) :

$$\frac{1}{\overline{OA'}} - \frac{1}{\overline{OA}} = \frac{1}{f_L} \quad \Leftrightarrow \quad \frac{1}{\overline{OA'}} = \frac{1}{\overline{OA}} + \frac{1}{f_L} \quad \Leftrightarrow \quad \overline{OA'} = \frac{f_L \times \overline{OA}}{f_L + \overline{OA}}$$

N.B. : on raisonne ici avec des grandeurs algébriques, ce qui signifie ici qu'une distance est positive si le vecteur la représentant est dans le même sens que l'axe optique (ici cameraDirection), et négative pour le sens opposé.

On reprend les mêmes notations que dans la partie 2.1.2. En omettant la notation algébrique, on obtient :

$$z_p^{'} = \frac{f_L \times (-OA)}{f_L - OA}$$

Déterminons maintenant les angles α_x et α_y :

$$\alpha_x = \arctan\left(\frac{y_p}{f_L}\right) \quad \text{et} \quad \alpha_y = \arctan\left(\frac{x_p}{f_L}\right)$$

On peut maintenant déterminer $y_p^{'}$ en fonction de $z_p^{'}$:

$$\tan(\alpha_x) = \frac{y_p^{'}}{z_p^{'}} \quad \Leftrightarrow \quad y_p^{'} = \tan(\alpha_x) \times z_p^{'} = \frac{y_p \times z_p^{'}}{f_L}$$

Les calculs sont analogues pour $x_p^{'}$. On obtient au final :

$$\boxed{x_p^{'} = \frac{y_p \times z_p^{'}}{f_L} \quad \text{et} \quad y_p^{'} = \frac{y_p \times z_p^{'}}{f_L}}$$

2.3 Modèle de la caméra à une matrice de microlentilles

2.3.1 Principe et schématisation

Ce modèle consiste à considérer non plus une seule lentille mais une matrice de microlentilles identiques. C'est cette matrice qui va nous permettre de multiplier les points de vue et, a fortiori, obtenir les informations 3D d'une scène photographiée.

L'ensemble des microlentilles fonctionnent toutes sur le principe du sténopé, c'est-à-dire que tous les rayons lumineux considérés passent obligatoirement par les centres des microlentilles (cf FIGURE 2.5).

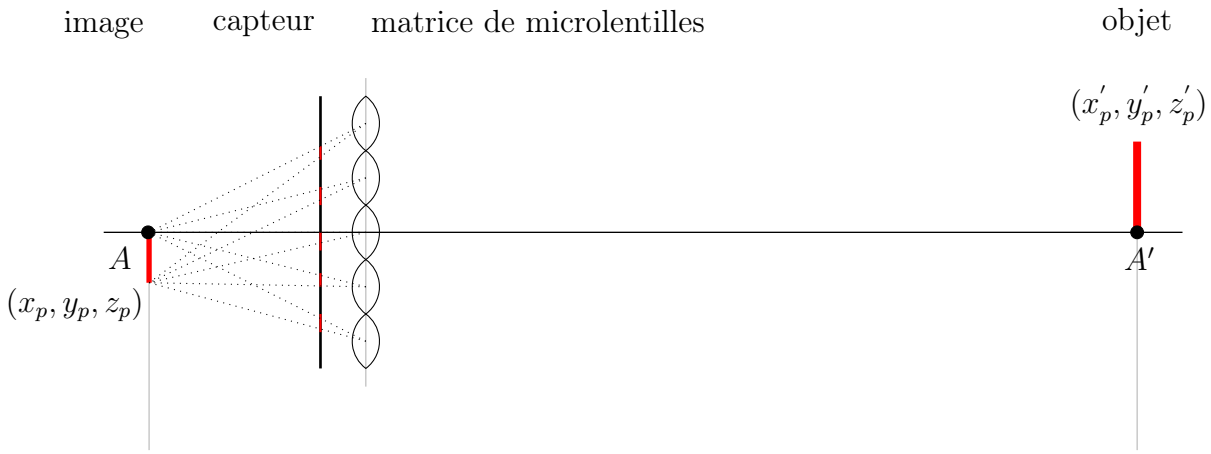


FIGURE 2.5 – Schématisation du modèle à une matrice de microlentilles

La problématique est toujours la même : on cherche à exprimer la position d'un pixel de l'image en fonction de la position du pixel de l'objet photographié. Seulement, dans ce cas, un pixel peut potentiellement se projeter à travers plusieurs microlentilles, ce qui implique la considération de nouveaux paramètres physiques que sont le cône de visualisation et l'ouverture d'une microlentille.

2.3.2 Cône de visualisation et considérations physiques

Tous les pixels de l'image ne se projettent pas dans la même microlentille de la matrice mais une même microlentille peut permettre à plusieurs pixels de se projeter. Ainsi, on définit un cône de visualisation pour chaque microlentille, ie une zone de l'espace où tout pixel qui s'y trouve se projette dans cette microlentille.

Ce cône de visualisation est formé grâce à l'image d'un disque de la forme de la lentille principale qui est présente dans la caméra plénoptique (partie 2.4). Afin de vérifier qu'un pixel se projette dans une microlentille, on vérifie que la droite passant par le pixel (de coordonnées $(x_{pixel}, y_{pixel}, z_{pixel})^T$) et le centre de la microlentille (de coordonnées $(x_{centre}, y_{centre}, z_{centre})^T$) coupe le disque (plus simple à implémenter qu'un test sur les angles). Pour cela, on calcule les coefficients directeurs de la droite en 3 dimensions :

$$coeff_x = \frac{x_{centre} - x_p}{z_{centre} - z_p} \qquad coeff_y = \frac{y_{centre} - y_p}{z_{centre} - z_p}$$

Posons r le rayon du disque et d la distance entre le disque et l'écran.

On projette le pixel sur le disque, de coordonnées $(x_{proj}, y_{proj}, z_{proj})^T$:

$$x_{proj} = coeff_x \times d - x_{centre} \qquad y_{proj} = coeff_y \times d - y_{centre}$$

Pour vérifier que la projection est sur le disque on regarde si :

$$\boxed{x_{proj}^2 + y_{proj}^2 \leq r^2}$$

Si c'est le cas, alors on trace ce rayon ; sinon, on passe à la microlentille suivante. Si, à la fin de ces vérifications, aucune microlentille ne correspond, on noircit le pixel.

On impose également une ouverture pour chaque lentille de la matrice, cette ouverture étant directement liée à la "quantité" de lumière disponible pour former l'image de l'objet virtuel. On considère dans notre cas que les cônes de vision ne se chevauchent pas ; sinon, cela donnerait aux pixels des couleurs incohérentes. D'autre part, divers paramètres viennent complexifier les équations, notamment le nombre de microlentilles ainsi que l'orientation de la matrice de microlentilles qui peut, selon les cas, subir une rotation autour de un ou plusieurs axes.

2.3.3 Matrices de rotation et tracé de rayons

La matrice de microlentilles peut subir 3 rotations différentes. En effet, on se place ici dans un espace euclidien à 3 dimensions dont l'origine du repère est le centre de la matrice de microlentilles. On cherche donc à faire subir une rotation au repère de la matrice initialement droite (changement de repère par rotation). Afin de gérer plus facilement ces rotations, nous faisons ici appel à des matrices de rotation qui nous permettent de stocker l'information liée à la rotation autour d'un axe. Ci-dessous les matrices de rotation exprimées selon leur axe (exemple : $R_x(\theta)$: rotation autour de l'axe x) :

$$R_x(\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos(\theta) & -sin(\theta) \\ 0 & sin(\theta) & cos(\theta) \end{pmatrix} R_y(\theta) = \begin{pmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{pmatrix} R_z(\theta) = \begin{pmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Une fois toutes ces matrices posées, on les applique aux coordonnées de chaque centre des microlentilles afin d'obtenir les nouvelles coordonnées de position après rotation. Cela

nous donne un nouveau avec un nouveau repère. La formule générale qui vient est :

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = R_x(\theta) \times R_y(\phi) \times R_z(\alpha) \times \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Ces calculs nous permettent de faire partir, d'un pixel donné, un rayon passant par l'un des nouveaux centres de microlentilles. Cependant, il faut trouver dans quelle microlentille le rayon doit passer : cette information nous est donnée grâce au cône de vision.

2.4 Concaténation des modèles et obtention du modèle plénoptique

2.4.1 Principe de la caméra plénoptique

La caméra plénoptique est le produit "final" de la combinaison des deux modèles précédents : on assemble dans un même système optique un capteur, une matrice de microlentilles et une lentille mince.

L'avantage par rapport à une simple matrice de lentilles est de pouvoir canaliser les rayons au travers de la lentille principale. Sans cette dernière, les rayons peuvent rencontrer l'écran selon un large choix possibles d'angles et il est donc plus difficile de reconstituer une scène en 3D à partir de telles informations. Ainsi, l'ajout de cette lentille principale a pour effet de concentrer les rayons sur une zone particulière en suivant le même principe que celui d'une loupe.

On effectue toujours le même travail pour la recherche du pixel par rapport à la scène. Cependant, on ajoute une nouvelle *couche* à la modélisation avec ce nouveau système optique.

2.4.2 Fusion des 2 modèles précédents

La confection de ce modèle consiste simplement à concaténer les deux modèles décrits précédemment (cf FIGURE 2.6).

Très peu de nouveautés dans les calculs concernant ce modèle car ils ont déjà été

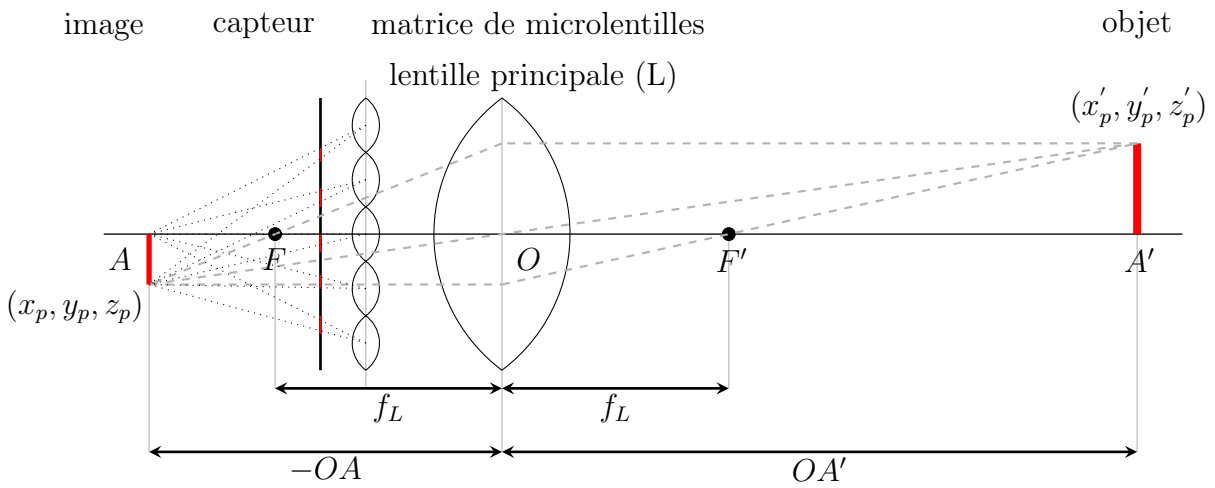


FIGURE 2.6 – Schématisation du modèle plénoptique

tous effectués (ceux pour la lentille principale d'une part, et ceux pour la matrice de microlentilles d'autre part).

On fixe comme repère de calcul le repère de la lentille principale : ainsi, seul l'origine du rayon, partant maintenant du pixel projeté (ds le plan de la lentille principale) n'est plus calculé dès le début mais une fois les calculs pour la matrice de microlentilles effectués.

Si le pixel se projette dans une microlentille, on pense à recalculer la distance du plan image par rapport à la matrice de microlentilles. Le code concernant la lentille principale a été stocké dans un vecteur à 3 composantes pour faciliter le calcul avec ray.Direction.

Chapitre 3

Résultats et discussions

3.1 Architecture du code et points d'action

3.1.1 Code Open Source

Ce logiciel étant régulièrement enrichi par la contribution de nombreux internautes, POV-Ray est un logiciel open source, c'est-à-dire qu'il est accessible par tous en lecture et écriture. Il est disponible sur le Git de POV-Ray dont le lien est le suivant :

`https://github.com/POV-Ray/povray`

Rappelons que ce code volumineux va servir à obtenir un rendu réaliste de l'image dont la géométrie et les formes que l'on souhaite visualiser ont été fixées dans le fichier de script. Il contient de nombreux fichiers contenant les fonctions mathématiques nécessaires aux transformations que l'on doit coder,...

3.1.2 Architecture globale du code

3.1.3 Principaux fichiers à modifier

L'intégration de notre travail dans le code déjà existant a nécessité la modification de quelques fichiers que voici :

- **tracepixel.h et .cpp** : c'est le fichier .cpp principal où se situe toute l'implémentation des algorithmes de tracé de rayons selon les différents types de caméra existants.

L'une des fonctions qu'elle contient, *CreateCameraRay*, a dû être complétée afin de s'adapter à la caméra plénoptique : elle prend en entrée les coordonnées d'un pixel, la taille de l'écran (en pixels) où est formée l'image ainsi qu'un rayon et retourne le rayon tracé selon une origine et une direction calculées.

- **camera.h et .cpp** : on déclare les caméras que l'on a implémentées sous forme de macro. Pour l'instant, nous avons changé la caméra par défaut mais le fichier de configuration sera à modifier par la suite ;

- **reservedwords.h et .cpp** : on ajoute les token qui seront utilisés principalement dans l'analyseur de code (appelé *parser* en anglais) ;

- **parser.cpp** : on ajoute le cas de la caméra plénoptique en associant le type de la caméra et son token (on se contente du strict minimum dans le code, tout comme effectué pour les CYC_CAMERA) ;

3.2 Algorithmes et obtention de rendus

3.2.1 Implémentation des modèles physiques

Nous nous sommes grandement basés sur le code de la caméra perspective déjà existant afin d'implémenter correctement nos modèles.

Le premier modèle (à une lentille mince) doit redéfinir l'origine du rayon à chaque passage par le *case* (contrairement au sténopé) et le tracer. Le principe de l'algorithme est le suivant :

Fonction `modèle à une lentille mince`(rayon, coordonnées du pixel, taille de l'écran) :
- passage des coordonnées en pixels aux coordonnées dans [-0.5 ; 0.5] ;
- initialisation de la focale et de la distance écran - lentille ;
- calcul de l'origine du rayon selon les coordonnées du pixel traité ;
- calcul des coordonnées du pixel image ;
- calcul de la direction de l'unique rayon selon les coordonnées calculées précédemment ;

Retourner tracé du rayon primaire ;
Fin

Algorithme 1 – *La fonction* MODÈLE À UNE LENTILLE MINCE

Le second modèle (matrice de microlentilles) consiste à vérifier si on doit calculer le centre d'une nouvelle microlentille, auquel cas le pixel ne peut se projeter dans aucune des lentilles précédentes pour l'instant :

Fonction `modèle à une matrice de microlentilles`(rayon, coordonnées du pixel, taille de l'écran) :
- passage des coordonnées en pixels aux coordonnées dans [-0.5 ; 0.5] ;
- initialisation du nombre de lentilles ;
- initialisation du diamètre de la microlentille et de la lentille principale ;
- initialisation de la distance matrice - lentille et écran - lentille ;
- initialisation des coordonnées du centre de la lentille / microlentille, de référence, du pixel ;
- calcul de la rotation ;

Tant que (pixel ne peut pas se projeter dans une microlentille) **faire**
- passage au centre de microlentille suivant ;
- transformation du point ayant subi une rotation ;
- déplacement de l'origine du rayon ;
- calcul des coefficients directeurs des droites passant par le pixel et le centre de la microlentille ;
- calcul des coordonnées du pixel projeté ;

Si (pixel appartient au cône de vision) **Alors**
- calcul de la direction du rayon à tracer dans la microlentille courante ;
- trouvé = vrai ;

Fin Si

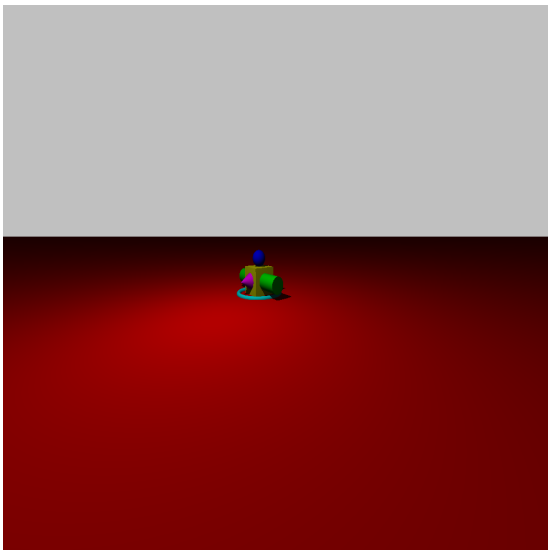
Fait
Si (trouvé) **Alors**
Retourner tracé du rayon primaire ;
Fin Si
Fin

Algorithme 2 – *La fonction* MODÈLE À UNE MATRICE DE MICROLENTILLES

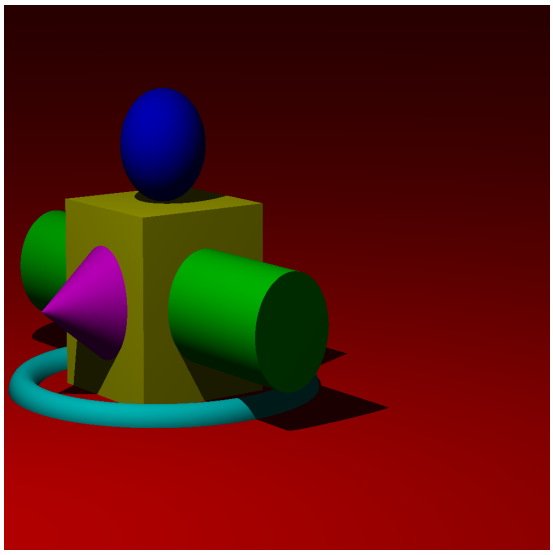
3.2.2 Rendus selon différents paramètres des modèles introductifs

Rappelons que le modèle sténopé est le modèle basique par défaut dans POV-Ray et que nous n'avons donc pas eu à l'implémenter. Le rendu que POV-Ray nous génère le rendu de référence de la partie 1.3.

Le modèle à une lentille mince fait apparaître à l'écran une image plus ou moins "zoomée", selon la focale (exprimée en mm) fixée (cf FIGURE 3.1 ET 3.2) et une distance écran-lentille de 50 mm.

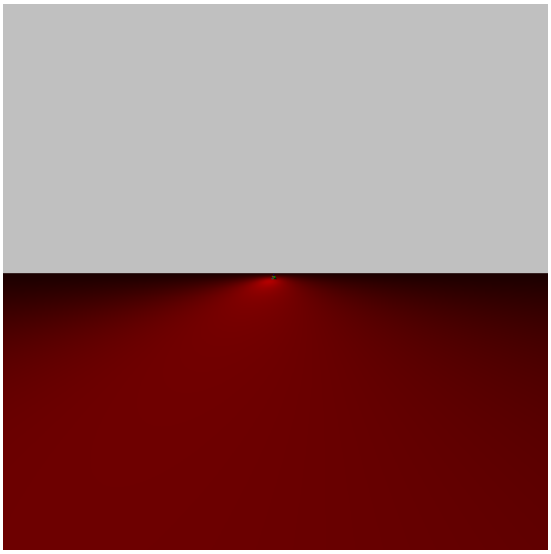


(a) *pour $f = 0.5$ mm*

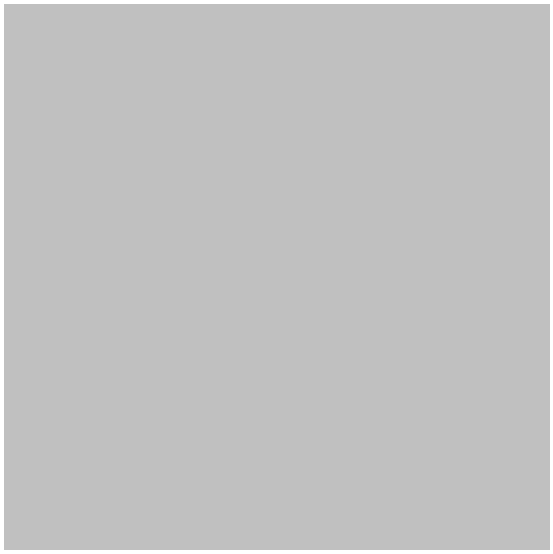


(b) *pour $f = 5$ mm*

FIGURE 3.1 – Rendus du modèle à 1 lentille mince pour des focales classiques



(a) *pour $f = 0.05$ mm*



(b) *pour $f = 50$ mm*

FIGURE 3.2 – Rendus du modèle à 1 lentille mince pour des focales extrêmes

L'image a été générée à partir d'un fichier **.pov** qui permet, comme décrit en partie 1.3, de déclarer les objets que nous souhaitons voir sur la scène (ici, une scène sur fond gris, dans laquelle a été disposé un plan rouge servant de "base" pour un ensemble d'objets

géométriques divers).

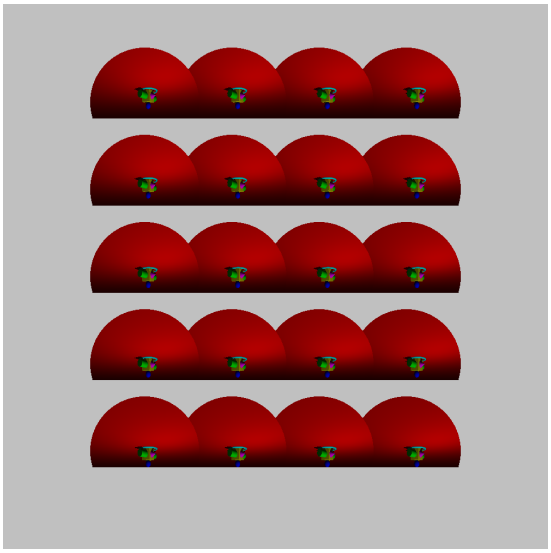
Une fois la lentille mince implémentée, nous avons dû fixer deux valeurs fondamentales : la distance focale ainsi que la distance entre le capteur et la lentille (notons la **d**). On constate que :

- **si f** $\geq$ **d**, il n'y a aucune image qui apparaît à l'écran, ce qui est normal car l'image créée est image virtuelle et donc, non visualisable (cf FIGURE 3.2 (b)) ;

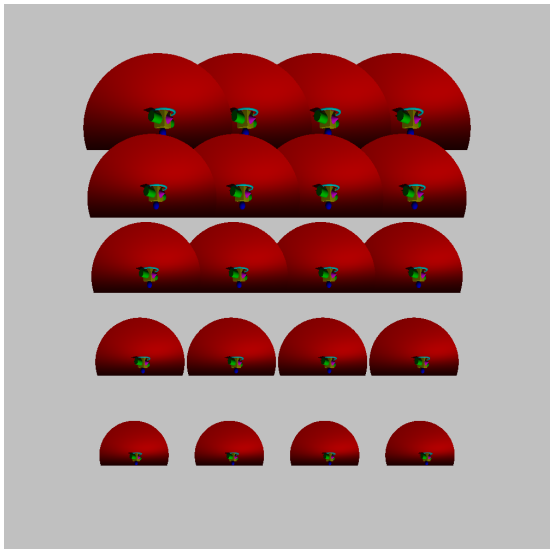
- **f** $<$ **d**, l'image créée est réelle et peut être visualisée. Plus la focale est grande, plus l'image est de grande taille.

Concernant le modèle à une matrice de microlentilles, nous obtenons les rendus ci-dessous (cf FIGURE 3.3). Remarquons que la modification de certains paramètres modifient instantanément le rendu (ci-dessous, on effectue une rotation de la matrice de lentilles de 20 $^\circ$ selon chaque axe de rotation du plan de la caméra).

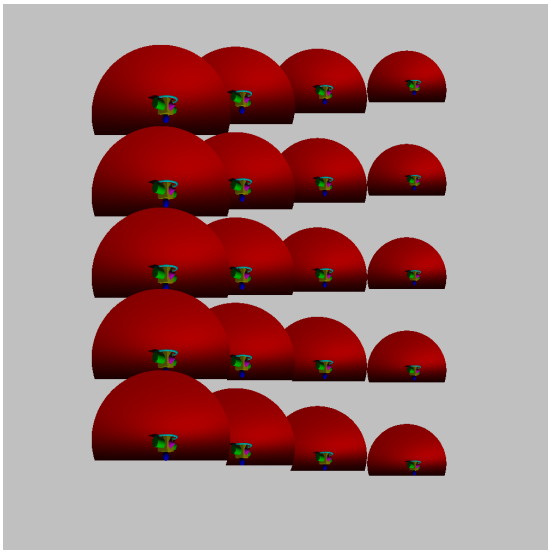
Par exemple, on constate que la rotation selon l'axe cameraRight (axe horizontal) de la matrice de microlentilles (cf FIGURE 3.3 (b)) est cohérente (les images supérieures sont de plus grande taille que les images inférieures ; elles se chevauchent même). Les autres images sont également cohérentes (axe vertical et axe normal au plan de la feuille).



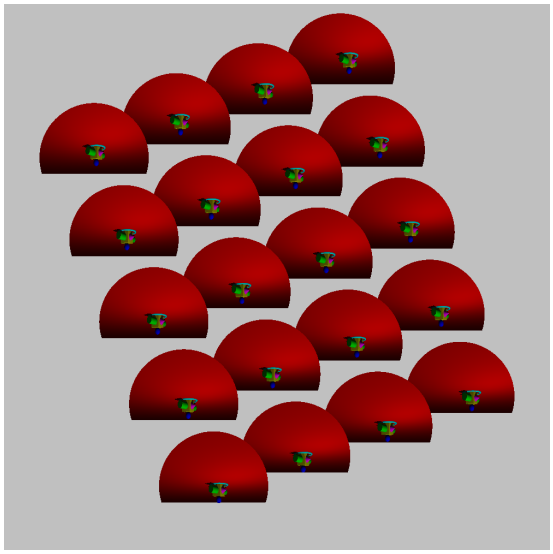
(a) *sans rotation de la matrice*



(b) *rotation autour de cameraRight*



(c) *rotation autour de cameraUp*

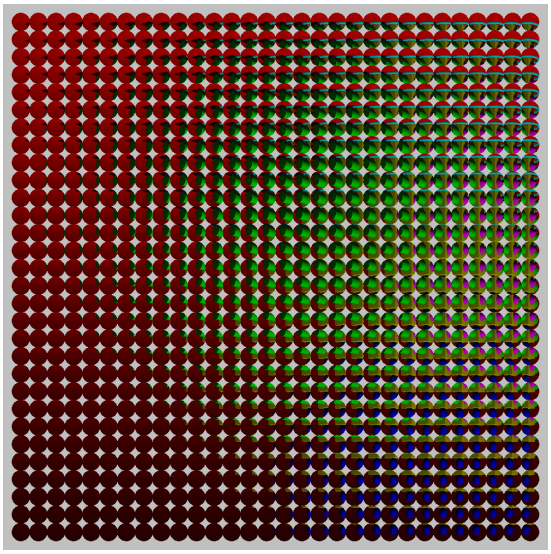


(d) *rotation autour de cameraDirection*

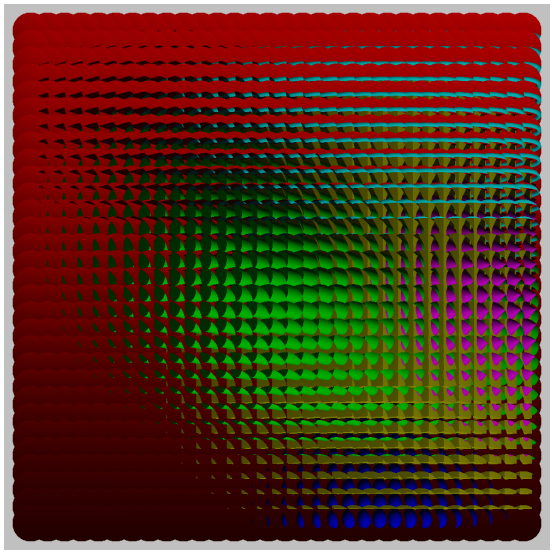
FIGURE 3.3 – Rendus du modèle à une matrice de microlentilles avec ou sans rotation

3.2.3 Rendus selon différents paramètres du modèle final

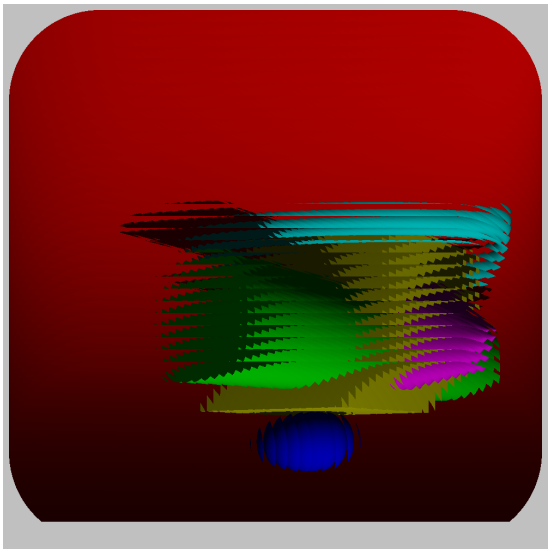
D'après les quelques images que nous avons de disponibles sur internet, on suppose que les résultats obtenus sont corrects. Tout d'abord, nous avons choisi de faire varier la distance que l'on notera d (sans unité car on travaille avec un rapport) entre l'écran et la matrice de microlentilles. Nous obtenons (cf FIGURE 3.4) :



(a) Rendu du modèle plénoptique avec d = 1/16



(b) Rendu du modèle plénoptique avec d = 1/8



(c) Rendu du modèle plénoptique avec d = 5/8

FIGURE 3.4 – Rendus du modèle plénoptique en faisant varier d

Le paramétrage le plus satisfaisant semble être celui avec d =1/8. Les microlentilles ne se chevauchent pas trop et nous donnent de multiples prises de vues, nous transmettant une information 3D. Le premier rendu présente l'avantage de distinguer chaque microlentille existante dans la matrice (en l'occurence 30).

La caméra plénoptique a de nombreux paramètres que l'on doit pouvoir régler manuellement (14 en réalité), et que l'on regroupe selon le composant où il agit (écran, matrice de microlentilles ou lentille principale). Sans toutefois lister exhaustivement les paramètres réglables, on se propose d'en mettre en évidence quelques uns d'entre eux. Nous avons pour l'instant pu en implémenter, nous le pensons, 5 d'entre eux.

Premièrement, il est possible de modifier la taille de l'écran (cf FIGURE 3.5), sans réduire la taille des autres composants.

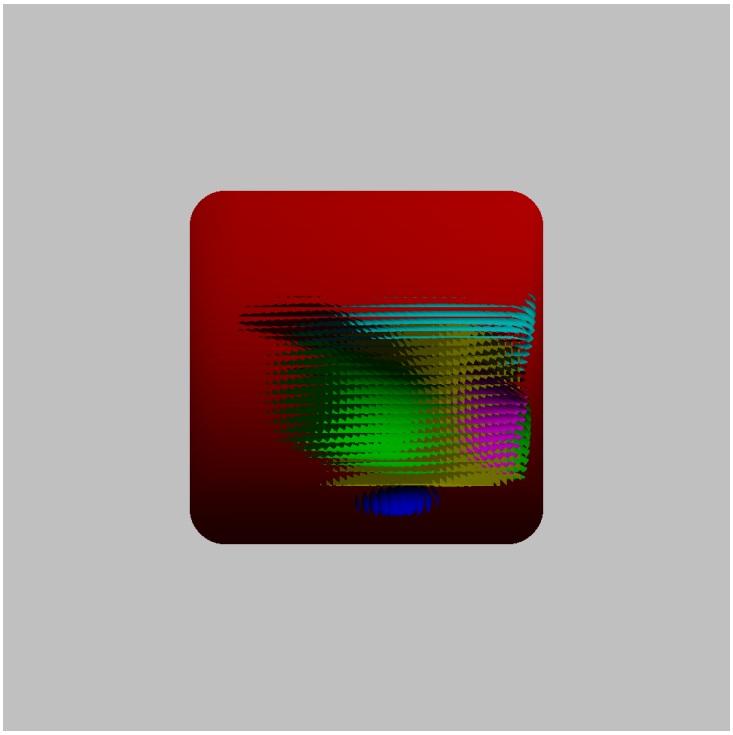


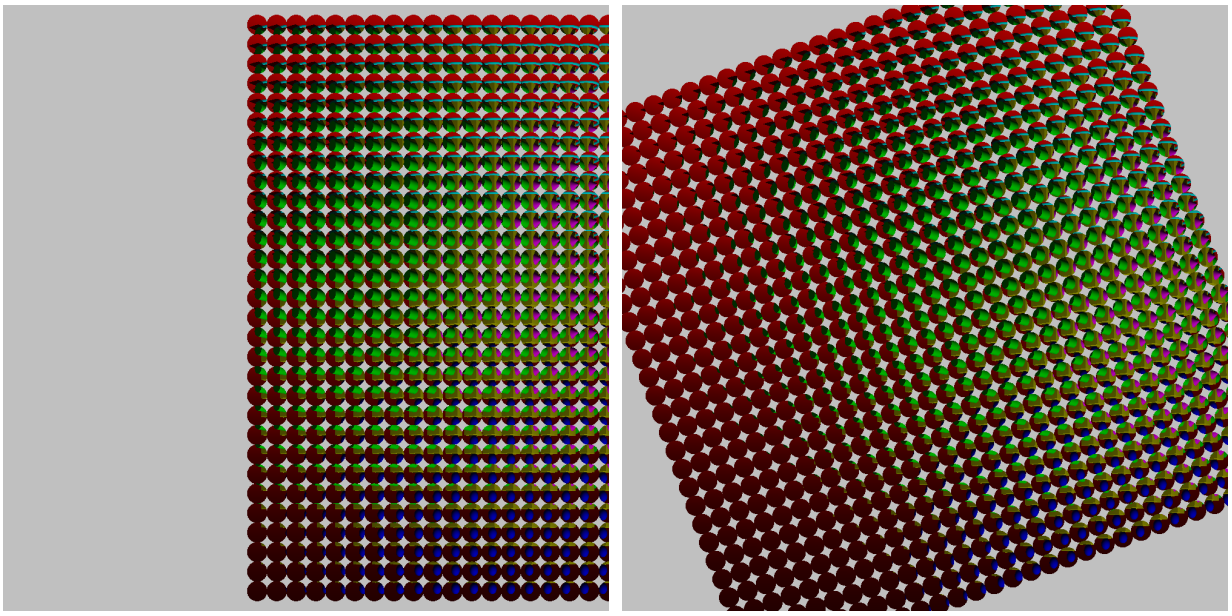
FIGURE 3.5 – Rendu du modèle plénoptique une augmentation par 2 de la dimension du capteur

On remarque qu'en augmentant par 2 les dimensions de l'écran, l'image est plus petite qu'avant. En effet, il n'y a pas de rayons qui se projettent ailleurs.

Effectuons maintenant des transformations à la matrices de microlentilles au modèle plénoptique (cf FIGURE 3.6).

On remarque que les transformations marchent de la même façon que celles pour le modèle de microlentilles simple car le modèle plénoptique utilise celui-ci comme partie intégrante de son modèle.

Enfin, générons une image où la focale est à 10 (on a fixé un d de 5/8) (cf FIGURE 3.7).



(a) Rendu du modèle plénoptique avec une translation de la matrice de lentilles de 0.4

(b) Rendu du modèle plénoptique avec une rotation de la matrice de lentilles de 20 °

FIGURE 3.6 – Rendus du modèle plénoptique en faisant varier différents paramètres

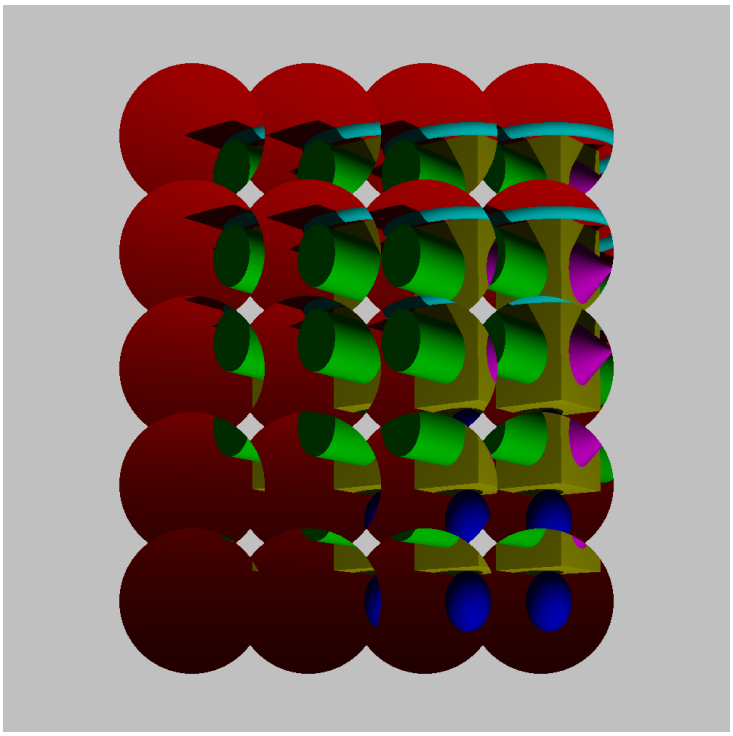


FIGURE 3.7 – Rendus du modèle plénoptique en faisant varier différents autres paramètres

Du fait du faible nombre de microlentilles, on voit partiellement l'objet mais l'effet plénoptique est clairement visible.

3.3 Discussions et perspectives

A l'issue de ce travail, on remarque le potentiel de la caméra plénoptique : ce n'est pas une caméra apportant seulement des informations 3D : elle permet également, sous différents angles de vue, de remonter à l'information en profondeur. Cette fonctionnalité permet l'amélioration de la qualité des images obtenues et ses applications sont multiples : les images générées par de telles caméras peuvent, par exemple, améliorer les diagnostics dans la milieu médical, et éventuellement, aider à repérer des tumeurs impossible à détecter avec une caméra classique [6].

Avec un peu plus de temps, nous aurions pu terminer la création du fichier de configuration. Certains paramètres de la caméra tels que la distorsion aurait demandé un travail important, tant sur l'acquisition des connaissances requises (physiques et mathématiques) que sur la recherche de son implémentation. Sinon, il serait éventuellement possible de publier le code déjà entamé en Open-Source sur POV-Ray afin qu'un travail collaboratif entre internautes puisse avoir lieu.

Conclusion

Notre objectif était d'implémenter le modèle de caméra plénoptique puis de l'intégrer dans le logiciel de tracé de rayons POV-Ray. Il était ensuite question de créer un fichier de configuration dans lequel il était possible de modifier à souhait les paramètres que l'on désirait régler, plutôt que de les modifier à même le code, obligeant à recompiler à chaque modification. L'objectif initial de ce projet a été rempli, et le résultat obtenu est celui attendu. L'appréhension du sujet nous a réclamé plus de temps que prévu, ce qui ne nous a pas permis de réaliser dans son ensemble, l'extension de cet objectif initial : la création du fichier de configuration.

Le modèle de la lentille mince a été bien plus difficile à implémenter que ce qu'il paraissait pour nous au début du projet. C'est après une compréhension profonde des techniques de tracés utilisées que nous avons pu résoudre les problèmes.

Nous avons eu l'occasion d'augmenter notre maîtrise en programmation en langage C++ et nous nous sommes familiarisés avec le langage intégré au logiciel POV-Ray pour créer une scène 3D. Nous avons alors été conscients de l'importance ainsi que du potentiel de tels logiciels de tracés, en l'ayant étudié sous tous ces aspects (visualisation de la scène 3D, modification du code interne au logiciel,...).

L'objectif intial du projet a été réalisé. L'avancement du projet de configuration permet d'entrevoir de nombreuses perspectives d'amélioration. La structure du code du tracé de rayons primaires par un switch case est peu pertinent car il oblige à modifier le prototype de la fonction globale, quand bien même la caméra plénoptique serait concernée. Remplacer chaque *case* par une fonction pourrait être une belle avancée, contribuant à la modularité du code.

Bibliographie

[1] POV-Ray documentation. http://www.povray.org/documentation/.

[2] Scratchapixel 2.0. Learn computer graphics from scratch ! https://www.scratchapixel.com.

[3] Thierry Chateau. Geometry for vision. http://chateaut.free.fr/CV/L2_VisionGeometry2014_4handout.pdf.

[4] David J. Eck. A fully computer-generated image created with the free ray-tracing program, pov-ray. http://math.hws.edu/eck/graphicsPics/povray1.html.

[5] Gyscos. POV-Ray. https://openclassrooms.com/courses/pov-ray.

[6] Aurélie Montmerle Bonnefois Marie-Thérèse Velluet Kevin Cossu, Guillaume Druart. Caméras plénoptiques pour l'imagerie tridimensionnelle. https://www.techniques-ingenieur.fr/base-documentaire/mesures-analyses-th1/mesures-tridimensionnelles-et-etats-de-surface-42409210/cameras-plenoptiques-pour-l-imagerie-tridimensionnelle-r1393/.

[7] Charles-Antoine Noury. Calibrage de caméra plénoptique à partir des images brutes.

[8] PC Reviews. Definition of : viewing frustum. https://www.pcmag.com/encyclopedia/term/61771/viewing-frustum.

[9] SEOS. 3d models - some basic principles. http://www.seos-project.eu/modules/3d-models/3d-models-c02-p01.html.

[10] Tonyee. Lytro light field camera — the camera of the future ? https://tonyee.wordpress.com/tag/plenoptic-camera/.

Glossaire

calibrage : opération qui consiste à déterminer une relation entre les coordonnées d'un point dans la scène 3D et celles associées dans l'image prise par la caméra.

capteur : élément de la caméra permettant de recevoir la lumière. Dans POV-Ray, le capteur est composé d'un million de pixels.

dioptre : surface séparant deux milieux transparents.

distance focale (aussi appelée focale) : distance séparant la lentille de son foyer.

foyer : point vers lequel convergent les rayons lumineux après leur passage dans un système optique (une lentille dans ce projet).

image : forme créée par la visualisation d'un objet à travers un système optique.

image virtuelle : désigne l'image (non visible par l'œil humain) formée par des rayons prolongés (lorsque les rayons divergent à la sortie de la lentille).

macro : permet le remplacement par le préprocesseur du texte fournit après le define en C++ à tous les endroits du code.

modeleur : logiciel de modélisation permettant la création de scènes en 3 dimensions composées de primitives.

objet virtuel : objet qui ne peut être touché mais qui peut être visualisé car situé en aval du système optique.

primitive (*infographie*) : forme géométrique de base à partir de laquelle il est possible de construire des formes plus complexes.

réfraction : désigne le changement de direction d'un rayon en changeant de milieu (chaque matériau ayant un indice de réfraction distinct).

repère : ensemble de trois axes othogonaux deux à deux permettant de définir un système de coordonnées.

Scène 3D : espace créé dans tout logiciel de modélisation où l'on peut placer des volumes, des lumières et des caméras.

token : la plus petite unité qui peut être traitée par un compilateur C++.

Annexe 1 - Cahier des charges initial du projet

PROPOSITION DE PROJET ISIMA

Filière : 4 **Ce projet concerne plutôt la :** ☐ 2ième année ☐ 3ième année

Intitulé du projet : Intégration du modèle de caméra plénoptique dans un logiciel de rendu

---

Personne responsable : Charles-Antoine Noury

Email : charles_antoine.noury@uca.fr **Téléphone :** +336 50 17 63 10

Entreprise ou Laboratoire : Institut Pascal

Adresse : _Campus Universitaire des Cézeaux - 4 Avenue Blaise Pascal
TSA 60026 / CS 60026 - 63178 Aubière Cedex - FRANCE

Mots clés décrivant le projet : caméra plénoptique, rendu, c++

Description du travail :

Une caméra plénoptique est un type de caméra particulier qui grâce à une matrice de micro-lentilles permet de capturer l'équivalent de plusieurs images en une seule prise de vue. Cette caractéristique est particulièrement intéressante car contrairement aux caméras classiques elle permet d'obtenir des informations 3D sur la scène ou de changer la mise au point d'une image après avoir pris le cliché.

L'Institut Pascal a développé un modèle de projection pour ce type de caméra, c'est-à-dire le modèle qui permet de calculer pour chaque point 3D d'une scène, la position où il sera projeté sur le capteur, et donc la position correspondante dans l'image. Inversement, en connaissant les paramètres de la caméra il est possible de calculer pour chaque pixel de l'image, le rayon 3D correspondant.

Ce modèle est connu et validé par des simulations simples telles que la projection d'un cube ou d'une mire. L'objectif de ce projet est de pouvoir simuler l'acquisition d'images plénoptiques sur des scènes 3D complexes. Pour cela, on souhaite intégrer le modèle de caméra plénoptique dans un logiciel de « lancer de rayons ». Le lancer de rayons permettra à partir d'un rayon 3D de récupérer son intersection avec les objets 3D de la scène et donc sa couleur. Dans ce projet on utilisera le logiciel POV-Ray, libre et multi-plateformes, qui permet de faire du lancer de rayons pour obtenir du rendu de scènes 3D.

Dans un premier temps les étudiants devront se familiariser avec le principe du rendu par lancer de rayons avec l'aide du tuteur et prendre en main le logiciel POV-Ray en réalisant des rendus utilisant des modèles de caméras classiques déjà existants dans le logiciel.
Il s'agira ensuite d'intégrer le modèle de caméra plénoptique dans POV-Ray et d'obtenir différentes simulations de rendu avec ce modèle. Les caméras plénoptiques ayant davantage de paramètres que les caméras classiques, ceux-ci devront être intégrés comme paramètres modifiables dans les fichiers de configuration des rendus.

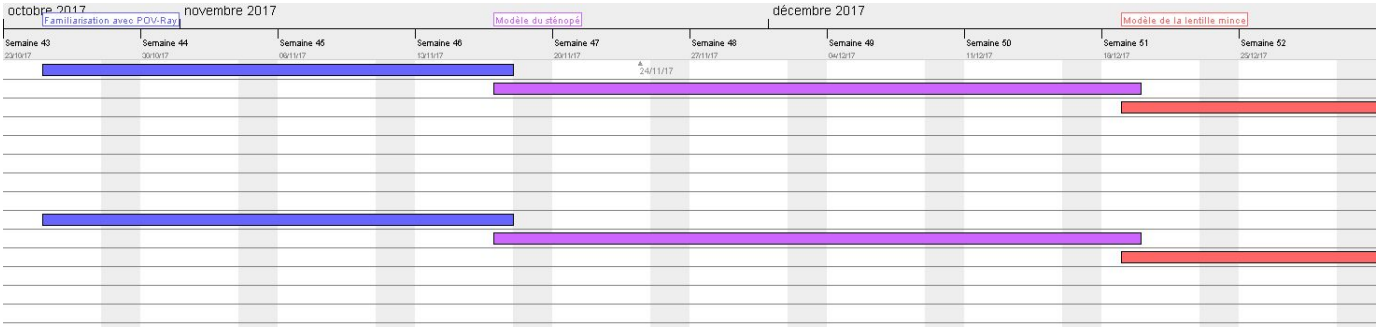
Le développement se fera en c++, en utilisant git pour le suivi. Toutes les informations concernant le modèle de caméra plénoptique seront fournies par l'encadrant du projet.

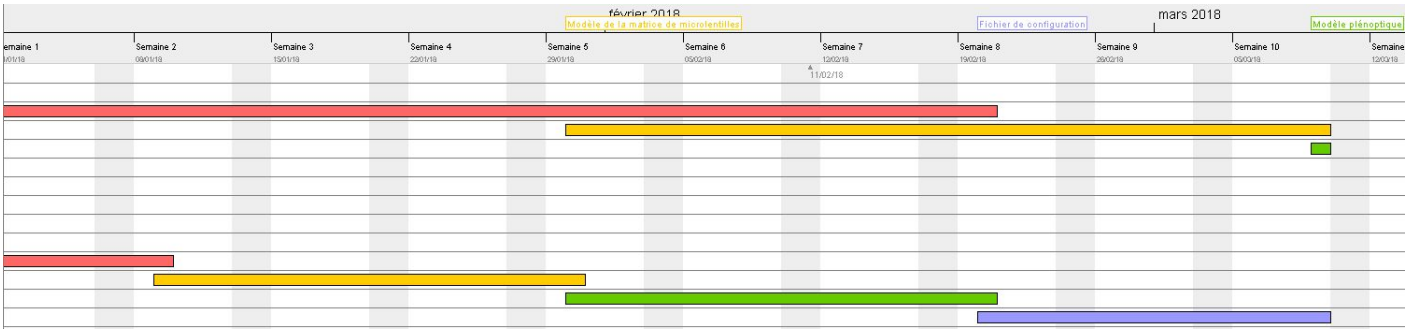
Environnement de travail (éventuellement) :
Outils / langages utilisés : c++, git, POV-Ray

Document à retourner au plus tard le 2 octobre 2017 à :

Annexe 2 - Diagrammes de Gantt prévisionnel et réel

Ci-dessous les diagrammes de Gantt prévisionnel (établi en début de projet) et réel (celui qui a été suivi, compte tenu des difficultés du projet). Les blocs supérieurs correspondent au diagramme réel et les blocs inférieurs correspondent au prévisionnel. L'image étant trop longue pour être contenue sur une seule page, elle a dû être scindée en deux.





Voici les tableaux de durées réelles et prévues pour chaque tâche du projet (couleur bleue : Familiarisation avec POV-Ray ; couleur grise : Fichier de configuration).

Plannification réelle :

Nom	Date de début	Date de fin
Familiarisation avec POV-Ray	25/10/17	17/11/17
Modèle du sténopé	17/11/17	19/12/17
Modèle de la lentille mince	19/12/17	20/02/18
Modèle de la matrice de microlentilles	30/01/18	09/03/18
Modèle plénoptique	09/03/18	09/03/18

Plannification prévue :

Familiarisation avec POV-Ray	25/10/17	17/11/17
Modèle du sténopé	17/11/17	19/12/17
Modèle de la lentille mince	19/12/17	09/01/18
Modèle de la matrice de microlentilles	09/01/18	30/01/18
Modèle plénoptique	30/01/18	20/02/18
Fichier de configuration	20/02/18	09/03/18

On constate que les difficultés à s'adapter au code C++ de la communauté POV-Ray et à comprendre les fondamentaux du *ray-tracing* a considérablement ralenti notre progression. On remarque néanmoins que le temps prévisionnel consacré au modèle plénoptique a été surestimé puisque, une fois le code pour la lentille mince et la matrice de microlentilles fonctionnels, l'implémentation du modèle plénoptique a été l'affaire de quelques minutes (concaténation des deux modèles, à quelques lignes de code près).

Annexe 3 - Code du projet

La totalité du code du projet se trouve sur le lien GITHUB du tuteur. Le voici :

`https://github.com/charlybigoud/povray`

Le fichier principal où la majorité du code a été inséré est "**tracepixel.cpp**". Le chemin à partir de la racine est : `source > core > render > tracepixel.cpp`. La fonction à regarder principalement est **CreatePrimaryRay**.