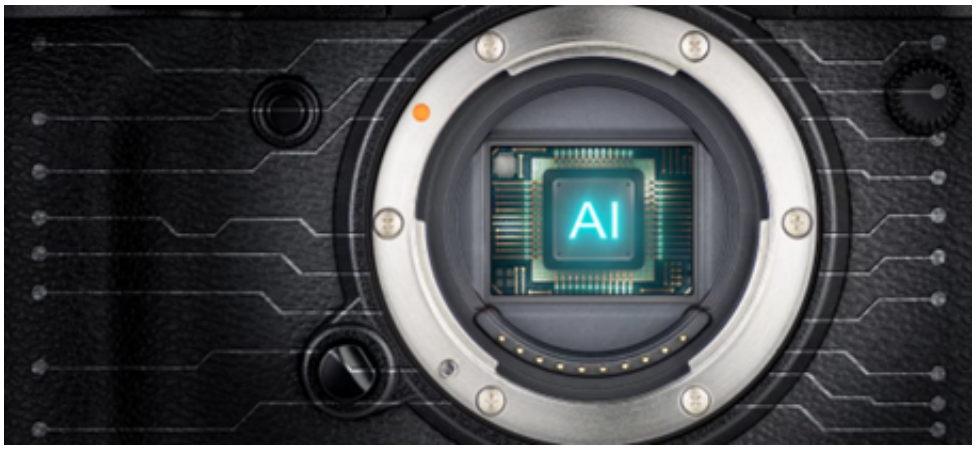
ensil-ensci ÉCOLE D'INGÉNIEURS DE LIMOGES



Rapport Bibliographique

PROJET MIX 4A

GENERATION D'IMAGE EN VUE DE L'ENTRAINEMENT D'UNE IA POUR LE SUIVI D'EVENEMENTS SPORTIFS

Alexis GUEVARA
Justin N'GBLOGNI

15 décembre 2025

Table des matières

1 Introduction

Dans un monde ou l'intelligence artificielle (IA) occupe une place de plus en plus importante dans tout le domaine de la vie humaine, la détection des images par ordinateur constitue un outil de poids pour l'analyse automatique dans des événements sportifs par exemple. Cependant, pour qu'elle puisse interpréter les différentes scènes, elle doit être alimentée par des images représentatives de la réalité, c'est dans ce cadre que s'inscrit notre projet de 4 année. L'objectif de notre travail est de comprendre et modéliser le processus de formation d'une image numérique à partir d'une caméra perspective, afin de générer des images synthétiques destinées à l'entraînement d'un modèle d'IA. Pour atteindre cet objectif, nous allons étudier plusieurs aspects théoriques :

- La compréhension du modèle de la caméra perspective
- La définition et la manipulation des différents repères de référence (repère monde, repère caméra, repère images, repère pixel)
- Les matrices de passages qui permettent le passage entre les repères
- La préparation de données pour l'apprentissage d'une IA

En somme, nous cherchons à établir un lien solide entre la théorie de la « Vision Artificielle » et sa mise en pratique.

2 Histoire de la caméra perspective

L'histoire de la caméra perspective retrouve ses origines dans l'observation du comportement de la lumière. Dès l'Antiquité, des savants comme Aristote avaient remarqué qu'une image inversée pouvait se former à travers un petit trou lorsqu'un rayon lumineux le traversait. Ce principe connu sous le nom de « Camera Obscura » à été étudié de manière approfondie au XI siècle par le scientifique arabe Ibn al-Haytham (Alhazen) dans son livre d'Optique. Il explique que la lumière se déplace en ligne droite et que les images se forment par projection sur une surface opposée. Au cours de la Renaissance, des artistes et chercheurs tels que Filippo Brunelleschi et Leon Battista Alberti ont perfectionné ces connaissances pour créer les règles de la perspective et la notion de « Point de Fuite » apparait. Au fil des siècles, la « Camera Obscura » a évolué. Le simple trou a été remplacé par des lentilles optiques, des miroirs ont été ajoutés pour redresser l'image, et l'appareil s'est miniaturisé pour devenir un outil scientifique et artistique. Ces évolutions ont conduit directement à l'invention de la caméra moderne et à la modélisation géométrique que nous appelons aujourd'hui caméra perspective.

3 Analyse du besoin

Dans le but de connaître le véritable intérêt de notre projet, l'analyse du besoin nous permettra de mettre en lumière son utilité, d'identifier sur quoi il agit et de préciser son objectif final.

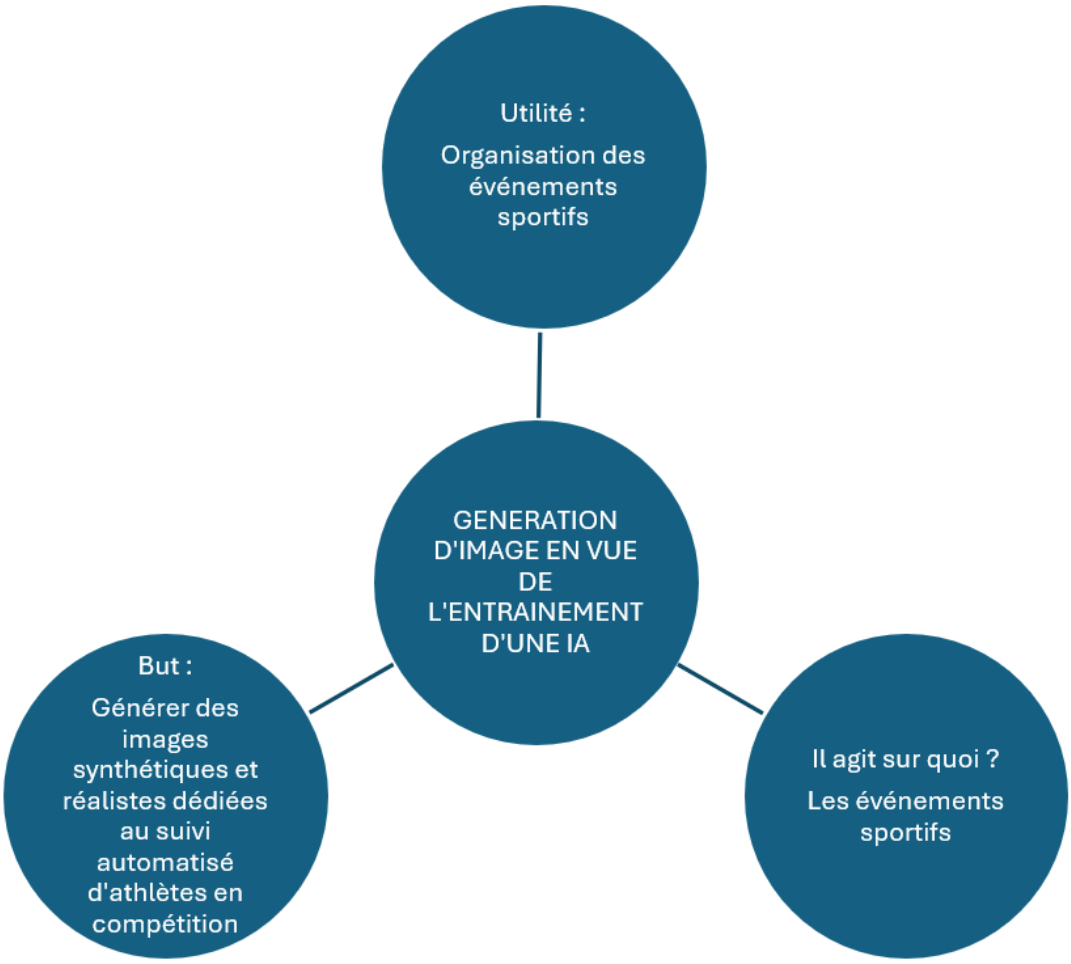


FIGURE 1 – Analyse du besoin du système « Génération d'images en vue de l'entraînement d'une IA »

Cette analyse nous conduit naturellement à la construction d'un cahier des charges fonctionnel et d'un diagramme pieuvre, outils qui structurent les interactions entre le système de génération d'images et son environnement (athlètes, caméras, conditions météo, modèles IA, etc.)

Le Cahier des Charges Fonctionnel est structuré autour de trois types de fonctions distinctes, garantissant à la fois la performance du système, sa qualité d'usage et la rigueur de sa conception.

- **FP (Fonctions Principales) :** Ces fonctions définissent le but direct du système.
- **FC (Fonctions Contraintes) :** Également appelées Fonctions de Service, elles définissent les **limites et les conditions** dans lesquelles le système doit fonctionner.
- **FV (Fonctions de Validation) :** Ces fonctions sont introduites pour formaliser et structurer la démarche de conception.

N°	Fonction	Description / Critère de satisfaction
FP1	Générer une image réaliste d'un athlète en action.	Qualité du rendu (perspective, éclairage) suffisante pour l'entraînement d'IA.
FP2	Varier les conditions d'environnement (Météo, Éclairage...).	Le système doit permettre de faire varier : la météo, l'éclairage, le type de stade, l'occlusion et le masquage.
FP3	Modéliser le dossard et ses données.	Le dossard généré doit inclure un rectangle blanc et des chiffres (numéros/couleurs) positionnés correctement.
FV1	Valider la projection géométrique.	Le système doit permettre de projeter avec succès :1). un point isolé ; 2). une forme simple (carré/triangle) ; 3). la surface du dossard complet.
FC1	Utilisation simple.	Utilisation du générateur de banque d'images facile à paramétrer et à exécuter.
FC2	Fournir des images claires.	Résolution suffisante et clarté des chiffres pour l'identification par l'IA

TABLE 1 – Extrait du Cahier des Charges Fonctionnel

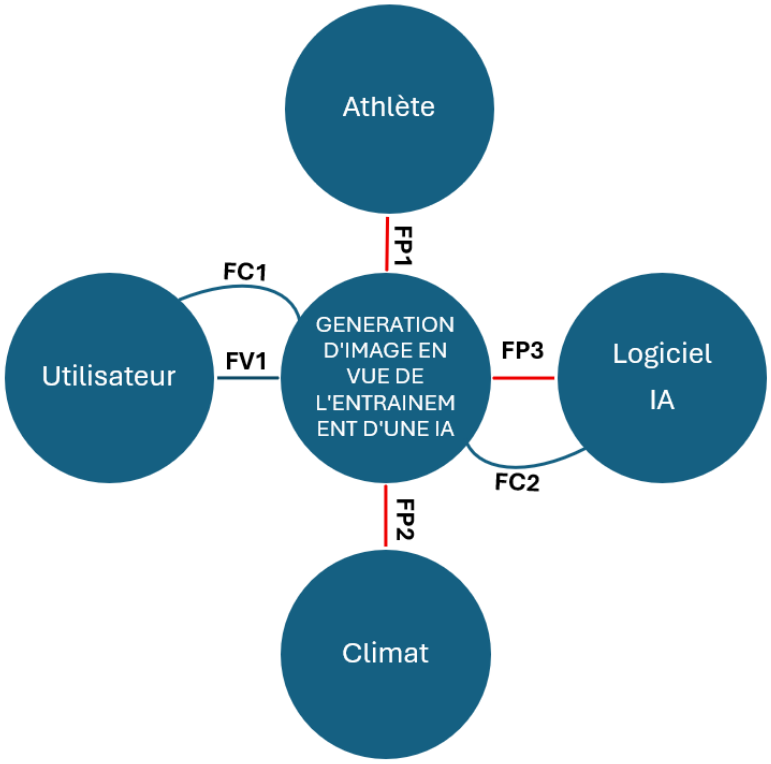


FIGURE 2 – Diagramme pieuvre du système « Génération d'images en vue de l'entraînement d'une IA »

4 Illustrations

Modélisation mathématique du modèle pinhole

Soit $R_w = (O_w, \vec{\imath}_w, \vec{\jmath}_w, \vec{k}_w)$ le repère monde.

Soit une caméra constituée de l'ensemble *écran + système focal*. Nous modélisons le système focal par un point O_c appelé *pinhole* (ou trou sténopéique), et l'écran par un rectangle $ABCD$ de largeur W (largeur de l'image) et de hauteur H (hauteur de l'image).

L'écran est situé à une distance F du point O_c ($O_cO_e = F$). Par le point O_c passe une droite perpendiculaire au plan (O_iAB), dont l'intersection avec ce plan définit le point O_e, centre du rectangle O_iABC. L'ensemble (O_c, O_iABC) constitue le système appelé *caméra*. Ce système est positionné dans l'espace, et chacun de ses points possède des coordonnées exprimables dans le repère monde $R_w = (O_w, \vec{\imath}_w, \vec{\jmath}_w, \vec{k}_w)$.

Définissons un repère orthonormé direct R_c d'origine O_c lié à notre système focal. Soit $R_c = (O_c, \vec{\imath}_c, \vec{\jmath}_c, \vec{k}_c)$, tel que $\vec{k}_c$ soit suivant l'axe optique (O_cO_e).

Définissons ensuite un repère orthonormé direct R_e, d'origine O_e, dans le plan de l'écran, lié à celui où se formera l'image. Soit : $R_e = (O_e, \vec{\imath}_e, \vec{\jmath}_e, \vec{k}_e)$ tel que $\vec{\imath}_e = \frac{\vec{O_IA}}{\|\vec{O_IA}\|}$, $\vec{\jmath}_e = \frac{\vec{CO_I}}{\|\vec{CO_I}\|}$ et $\vec{k}_e = \vec{k}_c$.

Définissons ensuite un repère orthonormé R_i d'origine O_i dans la plan de l'image et liée à notre image. Soit $R_i = (O_i, \vec{\imath}_i, \vec{\jmath}_i, \vec{k}_i))$, tel que $\vec{\imath}_i = \frac{\vec{O_iA}}{\|\vec{O_iA}\|}$, $\vec{\jmath}_i = \frac{\vec{O_iC}}{\|\vec{O_iC}\|}$ et $\vec{k}_i = \vec{k}_c$

Soit maintenant M un point de l'espace observé par la caméra et repéré par ces coordonnées dans le repère R_c. On a $\vec{O_cM} \begin{pmatrix} x_M \\ y_M \\ z_M \end{pmatrix}_{\mathcal{B}_c}$. L'image du point M se forme sur l'écran grâce au rayon lumineux passant par le point O_c (pinhole), sans déviation, jusqu'à l'écran, où il rencontre le plan (O_iABC) au point I.Les coordonnées du point I sont données dans le repère de l'écran R_e de la façon suivante :

$$\vec{O_eI} \begin{pmatrix} x_I \\ y_I \\ 0 \end{pmatrix}_{\mathcal{B}_e}$$

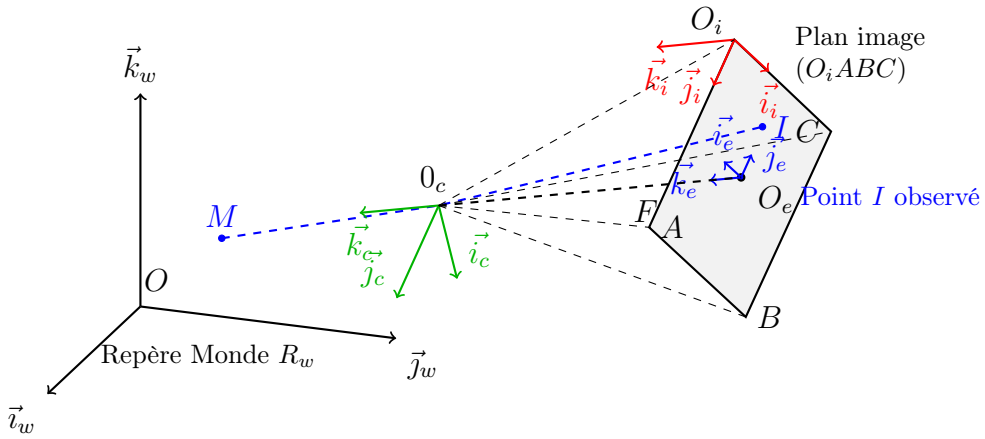
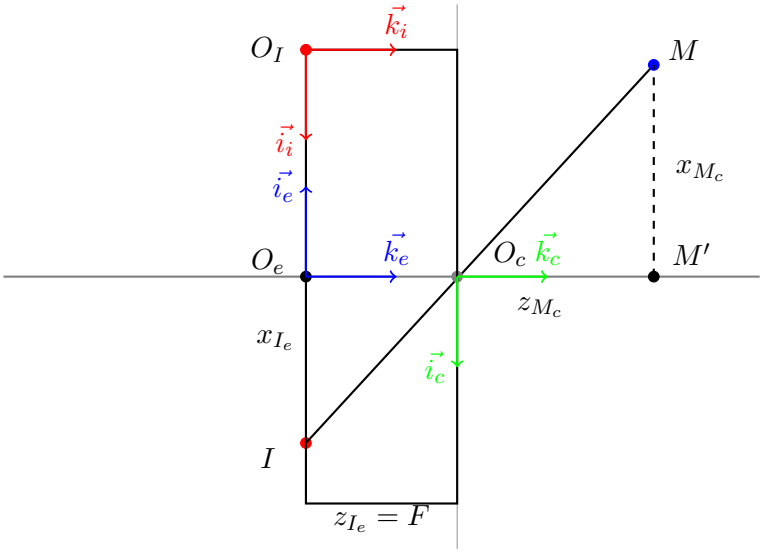


FIGURE 3 – Modélisation de la caméra pinhole avec plan image $(ABCD)$: formation du point image I du point objet M, grâce au rayon lumineux passant par le pinhole P.

La chaine de conversion de l'information sera la suivante :

FIGURE 4 – Pipeline du flux de l'information

Plaçons-nous dans le plan $(O_c, \vec{i_c}, \vec{k_c})$ pour mieux résoudre la partie associée à la matrice K.



En analysant la figure, on remarque que les triangles O_cO_eI et O_CMM' sont semblables. Cela étant, on peut déduire de façon triviale la relation :

$$\frac{|x_{I_e}|}{|-F|} = \frac{|x_{M_c}|}{|z_{M_c|}}.$$

Comme x_{I_e} et x_{M_c} sont de mêmes signes , on a alors $x_{I_e} = \dfrac{x_{M_c}}{z_{M_c}} F$ (1) .

Par analogie, on déduit que dans le plan $(O_c, \vec{j_c}, \vec{k_c})$, on a :

$$\frac{|y_{I_e}|}{|-F|} = \frac{|y_{M_c}|}{|z_{M_c|}}.$$

Soit $y_{I_e} = \dfrac{y_{M_c}}{z_{M_c}} F$ (2)

Vu que l'image ne se formera que dans le plan (O_IAB), $y_{I_e} = 0$. Faisons donc fi de cette coordonnée. De (1) et (2), on forme ce système :

$$\begin{cases} x_{I_e} = \dfrac{x_{M_c}}{z_{M_c}} F & (1) \\ y_{I_e} = \dfrac{y_{M_c}}{z_{M_c}} F & (2) \end{cases}$$

On écrit donc la matrice de passage des coordonnées du repère de la camera R_c vers le repère R_e de l'écran avec les coordonnées homogènes de la façon suivante :

$$\begin{bmatrix} x_{I_e} \\ y_{I_e} \\ 1 \end{bmatrix}_{\mathcal{B}_e} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{x_{M_c}}{z_{M_c}} \\ \frac{y_{M_c}}{z_{M_c}} \\ 1 \end{bmatrix}_{\mathcal{B}_c}$$

soit

$$z_{M_c} \begin{bmatrix} x_{I_e} \\ y_{I_e} \\ 1 \end{bmatrix}_{\mathcal{B}_e} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{M_c} \\ y_{M_c} \\ z_{M_c} \end{bmatrix}_{\mathcal{B}_c}$$

avec :

$$K' = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Remarque :

La distance focale est exprimée en millimètre, il en est de même pour les coordonnées du point :

$$\overrightarrow{O_c M} = \begin{pmatrix} x_{M_c} \\ y_{M_c} \\ z_{M_c} \end{pmatrix}_{\mathcal{B}_c},$$

Ceci n'étant pas le cas des coordonnées du point image I dans le repère, R_e qui sont elles exprimées en pixels.

Pour passer des millimètres en pixels, nous introduisons de nouveaux paramètres α_x et α_y tel que : $f_x = \dfrac{f}{\alpha_x}$ et $f_y = \dfrac{f}{\alpha_y}$. Ainsi K' devient K :

$$K = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1/\alpha_x & 0 & 0 \\ 0 & 1/\alpha_y & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & 0 \\ 0 & f_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

5 La matrice extrinsèque

La matrice extrinsèque nous permet de savoir comment notre caméra est placée et orientée dans le monde réel. Pour bien comprendre, imaginons que nous posons notre caméra sur un trépied quelque part dans les tribunes d'un stade. Nous devons alors connaître deux informations essentielles :

1. Dans quelle direction la caméra regarde-t-elle ? (le centre du terrain, le poteau de corner, le ciel... ?)
2. Où exactement est-elle située dans le stade ? (à 15 m du sol, à 30 m de la ligne de touche, etc.)

La matrice extrinsèque répond exactement à ces deux questions. C'est la réponse mathématique à : où est ma caméra et dans quelle direction regarde-t-elle par rapport au monde réel ?

La matrice des paramètres **intrinsèques** ne dépend pas de la scène observée. Ainsi, une fois estimée (calibrée), elle peut être réutilisée tant que la focale reste fixe (cas d'un objectif sans zoom). En revanche, si une image issue de la caméra est redimensionnée par un facteur d'échelle, tous ces paramètres doivent être mis à l'échelle (multipliés ou divisés, respectivement) par le même facteur.

La matrice combinée rotation-translation $\begin{bmatrix} R & t \end{bmatrix}$ est le produit d'une transformation projective et d'une transformation homogène. La transformation projective 3×4 convertit un point 3D exprimé dans le repère monde en un point 2D dans le plan image, mais en coordonnées normalisées de caméra :

$$\begin{bmatrix} x_{M_c} \\ y_{M_c} \\ z_{M_c} \end{bmatrix}_{\mathcal{B}_c} = z_{M_c} \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}_{\mathcal{B}_c} = \begin{bmatrix} R \,|\, t \end{bmatrix} \begin{bmatrix} X_{M_w} \\ Y_{M_w} \\ Z_{M_w} \\ 1 \end{bmatrix}_{\mathcal{B}_w} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} X_{M_w} \\ Y_{M_w} \\ Z_{M_w} \\ 1 \end{bmatrix}_{\mathcal{B}_w}$$

où $x' = \dfrac{x_{M_c}}{z_{M_c}}$ et $y' = \dfrac{y_{M_c}}{z_{M_c}}$ sont les coordonnées normalisées du point image (exprimées en unités de distance focale).

6 Fusion de paramètres intrinsèques et extrinsèques

On passe ensuite aux coordonnées en pixels grâce à la matrice intrinsèque K :

$$z_{M_c} \begin{bmatrix} x_{I_e} \\ y_{I_e} \\ 1 \end{bmatrix}_{\mathcal{B}_e} = K \, z_{M_c} \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}_{\mathcal{B}_c} = K \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}_{\mathcal{B}_w}$$

Posons $\lambda = z_{M_c}$, on obtient donc l'équation finale de projection :

$$\lambda \begin{bmatrix} x_{I_e} \\ y_{I_e} \\ 1 \end{bmatrix}_{\mathcal{B}_e} = K \begin{bmatrix} R \,|\, t \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}_{\mathcal{B}_w}$$

7 Représentation des repères image et écran

Maintenant que nous avons les coordonnées du point (objet) dans le repère écran, il faudrait recalculer ses coordonnées, cette fois-ci non pas dans le repère de l'écran, mais dans le repère image. Réutilisons nos repères de départ : $R_i = (O_i, \vec{\imath}_i, \vec{\jmath}_i, \vec{k}_i)$ est le repère image (la convention en traitement d'image oblige que l'origine O_i soit généralement au coin supérieur gauche de l'image) et $R_e = (O_e, \vec{\imath}_e, \vec{\jmath}_e, \vec{k}_e)$ est le repère écran (origine O_e au centre optique de l'image, aussi appelé point principal).

Dans le cadre de ce projet, nous utilisons une matrice de transformation **homogène** 3×3. Cette matrice $\mathbf{R}$ est conçue pour opérer sur des points définis en coordonnées homogènes, facilitant l'intégration des rotations et translations en une seule opération matricielle.

La matrice de rotation $\mathbf{R}$ autour de l'axe k_e par un angle θ est donnée par la forme générale :

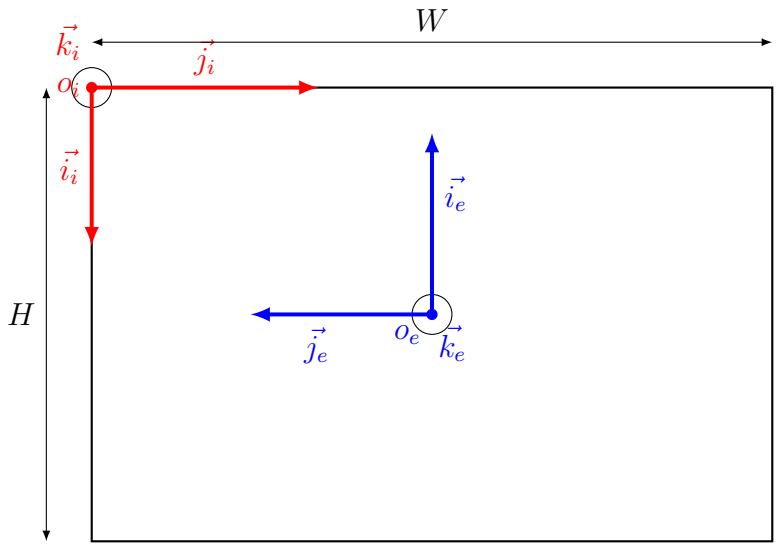


FIGURE 5 – Représentation des repères Image et Ecrans

$$\mathbf{R}(\theta) = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{1}$$

La transformation spécifique requise pour relier les axes de nos repères Écran et Image est une **rotation de** 180° **(π radians)**. Cette rotation permet d'aligner la direction des axes, notamment si l'un utilise une convention où l'axe vertical est dirigé vers le haut (comme dans le $\mathcal{R}_e$) et l'autre vers le bas (comme dans le repère Image $\mathcal{R}_i$).

En appliquant l'angle $\theta = \pi$ (180°) à la matrice générale, nous obtenons les valeurs suivantes :

— $\cos(\pi) = -1$

— $\sin(\pi) = 0$

Ce qui simplifie la matrice de rotation $\mathbf{R}$ en une matrice de **symétrie centrale** dans le plan (x, y) :

$$\mathbf{R} = \begin{pmatrix} \cos(\pi) & -\sin(\pi) & 0 \\ \sin(\pi) & \cos(\pi) & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2}$$

Nous connaissons les coordonnées du point I dans le repère $\mathbf{R}_e$. Nous souhaitons maintenant les déterminer dans le repère $\mathbf{R}_i$. Nous aurons donc les équations suivantes :

$$\overrightarrow{O_e I_e} = \begin{pmatrix} x_{I_e} \\ y_{I_e} \\ 0 \end{pmatrix} \tag{3}$$

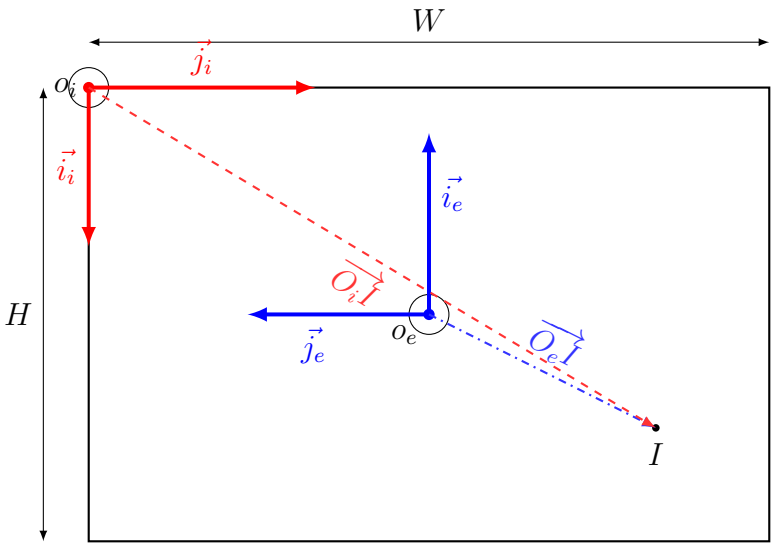


FIGURE 6 – Exemple d'illustration

$$\begin{aligned}\overrightarrow{O_iI_e} &= \mathbf{T}_{e\to i}\cdot\overrightarrow{O_eI_e}\\ &= \begin{pmatrix}-1 & 0 & 0 & W/2\\ 0 & -1 & 0 & H/2\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{pmatrix}\cdot\begin{pmatrix}x_{I_e}\\ y_{I_e}\\ 0\\ 1\end{pmatrix}\\ &= \begin{pmatrix}W/2-x_{I_e}\\ H/2-y_{I_e}\\ 0\\ 1\end{pmatrix}\end{aligned}$$

Posons $A = \begin{bmatrix}-1 & 0 & 0 & W/2\\ 0 & -1 & 0 & H/2\\ 0 & 0 & 1 & 0\\ 0 & -1 & 0 & 1\end{bmatrix}$ Donc on a ; $\begin{bmatrix}v\\ u\\ 0\end{bmatrix}_{\mathcal{B}_i} = A.\begin{bmatrix}x_{I_e}\\ y_{I_e}\\ 0\\ 1\end{bmatrix}_{\mathcal{B}_e}$

Ainsi, nous avons finalement les coordonnées d'un point dans le repère image $(\mathbf{R}_i)$. Nous pouvons donc retenir ces deux équations donnant les coordonnées d'un point image formé sur l'écran de la caméra à partir des coordonnées d'un point objet repéré dans le monde :

$$\begin{cases}\lambda\begin{bmatrix}x_{I_e}\\ y_{I_e}\\ 1\end{bmatrix}_{\mathcal{B}_e} = K\,[R\,|\,t]\begin{bmatrix}X_w\\ Y_w\\ Z_w\\ 1\end{bmatrix}_{\mathcal{B}_w}\\ \begin{bmatrix}v\\ u\\ 0\end{bmatrix}_{\mathcal{B}_i} = A.\begin{bmatrix}x_{I_e}\\ y_{I_e}\\ 0\\ 1\end{bmatrix}_{\mathcal{B}_e}\end{cases}$$

Par la suite, nous allons chercher à faire cela pour différents points. L'objectif à partir de plusieurs points représentant un objet réel, ici notre dossard, reproduire son image

dans le plan image. Pour y parvenir, nous allons d'abord le faire pour des cas simples. Nous prendrons des figures géométriques simples comme un triangle, un carré.

8 Identification des limites du champ d'observation de notre caméra en fonction des paramètres intrinsèque.

Énumérons les conditions pour qu'un objet puisse être observé par la caméra. Nous en identifions trois :

1. Condition de profondeur :
   L'objet est devant la caméra : cette condition triviale, mais très importante, nous permet d'écrire que :$z_{M_c} > 0$ (Souvent, on utilise même une distance minimale "Near clip" : $z_{M_c} > Z_{min}$)
2. Condition sur la verticale (H)
   Nous savons que l'abscisse du point image dans le repère de l'écran est donnée par $f_x \cdot \frac{x_{M_c}}{z_{M_c}}$ et pour que ce point soit sur l'écran, il doit être entre $-H/2$ et $+H/2$.
   L'inégalité à respecter :
   $$-\frac{H}{2} \leq f_x \cdot \frac{x_{M_c}}{z_{M_c}} \leq \frac{H}{2}$$
   Ce qui revient à dire (pour contraindre la position X) :
   $$|x_{M_c}| \leq \frac{H \cdot z_{M_c}}{2f_x}$$
3. Condition sur la largeur (W)
   C'est la même logique pour la largeur. L'ordonnée du point projeté est donnée par $f_y \cdot \frac{x_{M_c}}{z_{M_c}}$ et doit donc être entre $-W/2$ et $+W/2$.L'inégalité à respecter :
   $$-\frac{W}{2} \leq f_y \cdot \frac{y_{M_c}}{z_{M_c}} \leq \frac{W}{2}$$
   Ce qui revient à dire :
   $$|y_{M_c}| \leq \frac{W \cdot z_{M_c}}{2f_y}$$

Tout au long de la suite de notre projet, nous utiliserons ces conditions pour définir la position de notre caméra par rapport à notre objet afin d'éviter des valeurs abérantes.

9 Tests réalisés

Ici, nous allons expliquer les différents essais que nous avons réalisés. Nous avons réussi à passer du repère monde au repère image, tout cela de manière virtuelle. Pour cela, nous avons codé en Python en raison de sa simplicité et de la disponibilité de librairies puissantes de calcul matriciel et de représentation 3D comme **numpy** et **matplotlib**. C'est pour cela que, dès l'entame de notre code, on importe ces librairies comme suit :

```
import numpy as np
import matplotlib.pyplot as plt
```

FIGURE 7 – Importation des dépendances nécessaires au projet.

9.1 Définition du Modèle de Caméra et des Matrices Fondamentales

Notre code initialise les paramètres cruciaux de la projection en utilisant les concepts de la vision par ordinateur, notamment en définissant les matrices intrinsèques et extrinsèques.

9.1.1 Matrice Intrinsèque (K) et Matrice de Transformation (A)

La matrice **K** modélise la transformation du repère Caméra vers le repère Écran (coordonnées normalisées), utilisant les paramètres de focale (f_x, f_y) et du centre optique (c_x, c_y). La matrice **A** gère le passage du repère Écran centré à la grille de pixels $(\mathbf{R}_i)$, en appliquant la rotation de 180° et la translation nécessaire $(W/2, H/2)$. Pour choisir les valeurs de ces paramètres, nous nous sommes rapportés à une caméra bon marché, une caméra Raspberry Pi 8MP (IMX219).

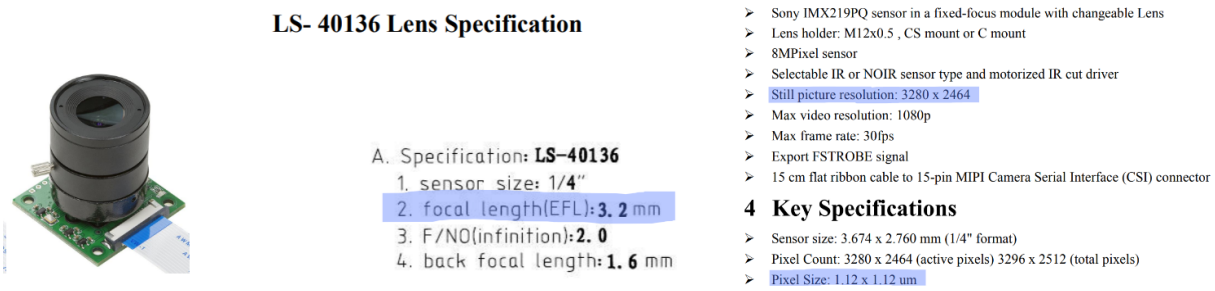


FIGURE 8 – Choix d'une caméra modèle et identification des spécifications

On peut tirer de la datasheet de cette caméra les informations suivantes : $\begin{cases} \text{f} = 3.2 \text{ mm} \\ \text{W} = 3280 \text{ pixels} \\ \text{H} = 2464 \text{ pixels} \\ \alpha_x = \alpha_y = 1.12 \mu m \end{cases}$

```python
fx = 2857 # Focale en pixels fx = 3.2/0.00112
fy = 2857 # Focale en pixels fy = 3.2/0.00112
cx = 0
cy = 0
s = 0

H = 2464 # Hauteur de l'image en pixels
W = 3280 # Largeur de l'image en pixels

K = np.array([[fx, s, cx],
              [0, fy, cy],
                [0,   0,   1]]) # Matrice des paramètres intrinsèques de la caméra

A = np.array([[-1, 0, 0, W/2],
              [0, -1, 0, H/2],
                [0, 0, 1, 0]])  # Matrice de transformation des coordonnées écran aux coordonnées image
```

FIGURE 9 – Définition des matrices intrinsèques **K** et de la matrice de transformation **A** en Python.

```python
class camera_intrinseque:
    def __init__(self, K , A):
        self.K = K
        self.A = A
    def project_point(self, P_camera):
        """
        Projette un point 3D dans le repère de la caméra sur le plan image.
        P_camera : coordonnées du point dans le repère de la caméra (x, y, z)
        Retourne les coordonnées 2D du point projeté sur le plan image (u, v)
        """
        x, y, z = P_camera
        if z <= 0:
            return None  # Le point est derrière la caméra
        I_ecran = K @ P_camera[:3]
        # Conversion en coordonnées inhomogènes
        I_ecran = np.round(I_ecran[:2] / I_ecran[2])
        I_image = A @np.hstack((I_ecran, np.array([0., 1.])))
        v, u = I_image[:2]
        return np.array([v, u])

camera = camera_intrinseque(K, A)
```

FIGURE 10 – Classe caméra_intrinsèque

Grâce ensuite à la classe **caméra_intrinsèque** , nous pouvons créer un objet caméra possédant des caractéristiques spécifiques et possédant une méthode permettant de calculer les coordonnées d'un point image à partir des coordonnées dans le repère caméra.

9.1.2 Transformation Extrinsèque et la Classe `Repere`

Ici nous définissons une classe nommée **Repere**, qui pourrait avoir comme attribut l'origine du repère et la matrice de rotation du repère vers le monde. Cette classe nous permettra par la suite de définir les différents repères de travail. Cette classe est essentielle pour gérer le changement de base et la transformation des coordonnées d'un point 3D défini dans le **Repère Monde** vers le **Repère Caméra** (méthode `coordonnees_point_dans_ce_repere`). Ce calcul prendra en compte les coordonnées de l'origine du nouveau repère défini dans le repère Monde. Cette classe aussi, grâce à la méthode `coordonnees_point_dans_repere_monde`, nous permet de déterminer les coordonnées d'un point dans le monde si on peut le situer par rapport à la caméra.

```python
class Repere:

    def __init__(self, origine = np.array([0, 0, 0]), R=np.eye(3)):
        self.rotation = R
        self.i = np.array([1, 0, 0]) @ self.rotation
        self.j = np.array([0, 1, 0]) @ self.rotation
        self.k = np.array([0, 0, 1]) @ self.rotation
        # self.i = self.rotation[:, 0]
        # self.j = self.rotation[:, 1]
        # self.k = self.rotation[:, 2]
        self.origine = origine
    def coordonnees_point_dans_ce_repere(self, point_global):
        """
        Calcule les coordonnées d'un point global dans ce repère.
        """
        vecteur = point_global - self.origine
        x = np.dot(vecteur, self.i)
        y = np.dot(vecteur, self.j)
        z = np.dot(vecteur, self.k)
        return np.array([x, y, z])
    def coordonnees_point_dans_repere_monde(self, point_local):
        """
        Calcule les coordonnées d'un point local dans le repère monde.
        """
        return (point_local[0] * self.i +
                point_local[1] * self.j +
                point_local[2] * self.k +
                self.origine)
```

FIGURE 11 – Implémentation de la classe Repere et des méthodes de changement de coordonnées.

9.2 Validation de la Projection Géométrique (Fonction FV1)

La fonction `project_point` est le cœur de la validation. Elle réalise l'étape **FV1** en enchaînant toutes les transformations : du repère Monde (3D) au repère Image (2D).

Pour y parvenir, nous créons une instance de la classe repère pour spécifier le repère de la caméra et une autre instance pour définir une caméra comme suit :

```python
camera = camera_intrinseque(K, A)
```

0.0s Python

```python
Repere_W = Repere(origine=O_w)
Repere_C = Repere(O_c, R=R_word_to_camera )
```

0.0s Python

FIGURE 12 – Création d'une caméra et des repères

Notons que le repère Monde est défini par son origine $O_w = (0, 0, 0)$. Le repère Caméra est placé en $O_c = (0, l, 0)$, sur l'axe des ordonnées du repère Monde, à une distance l de

l'origine. La matrice de rotation $\mathbf{R}$ correspond à l'alignement de l'axe optique de la caméra avec l'axe des ordonnées (Y_w) du repère Monde, obtenu par deux rotations successives de $\pi/2$: une autour de Y_w, puis autour de X_w.

En suite nous suivons la logique de l'algorithme de projection.

9.2.1 Algorithme de Projection 3D → 2D

L'algorithme de projection se décompose en trois phases :

1. **Transformation Extrinsèque :** Le point $\mathbf{P}_{\text{monde}}$ est converti en $\mathbf{P}_{\text{camera}}$ via la classe `Repere`.
2. **Projection Pin-Hole :** Le point $\mathbf{P}_{\text{camera}}$ est projeté sur le plan écran à l'aide de la matrice $\mathbf{K}$.
3. **Conversion Pixel :** Les coordonnées écran sont transformées en coordonnées pixel ($\mathbf{R}_i$) via la matrice $\mathbf{A}$, incluant la rotation de 180° et la translation.

Notre code utilise une fonction de test unitaire pour valider d'abord le changement de repère et la projection d'un point isolé M_{world}.

```python
def project_point(M_world):
    M_camera = Repere_C.coordonnees_point_dans_ce_repere(M_world[:3])
    I_image  = camera.project_point(M_camera)
    print(I_image)
    return I_image
```

FIGURE 13 – Fonction `project_point` et validation sur un point isolé.

9.2.2 Test sur une Forme Géométrique (Triangle)

Pour satisfaire l'exigence **FV1** de la validation sur une forme simple, un triangle ($\alpha\beta\gamma$) est modélisé dans l'espace, puis ses sommets sont projetés.

1. Les coordonnées $\alpha_{\text{camera}}, \beta_{\text{camera}}, \gamma_{\text{camera}}$ sont définies, simulant la position du dossard.

```python
d = 15
alpha_camera = np.array([d, d, 1, 1])
beta_camera = np.array([-d, d, 1, 1])
gamma_camera = np.array([0, -d, 1, 1])
```

   Nous visualisons bien les trois points formant le triangle pour nous assurer de la forme géométrique définie.

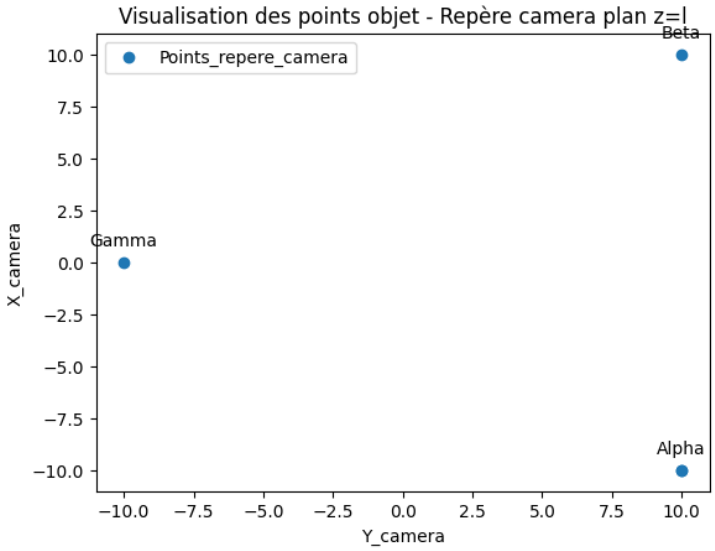


FIGURE 14 – Visualisation du triangle dans le plan $Y_c = l$ ($\mathbf{R}_c$)

2. Nous utilisons ces coordonnées pour retrouver les coordonnées des points du triangle dans le repère monde (repère d'où le calcul d'image doit partir).

```
alpha_monde = Repere_C.coordonnees_point_dans_repere_monde(alpha_camera[:3])
beta_monde = Repere_C.coordonnees_point_dans_repere_monde(beta_camera[:3])
gamma_monde = Repere_C.coordonnees_point_dans_repere_monde(gamma_camera[:3])
```

✓ 0.0s Python

```
print(alpha_monde, beta_monde, gamma_monde)
```

✓ 0.0s Python

```
[ 10.   0. -10.] [10.  0. 10.] [-10.   0.   0.]
```

3. Par notre fonction `project_point` nous les projetons pour obtenir les coordonnées pixels et donc $\alpha_{\text{image}}, \beta_{\text{image}}, \gamma_{\text{image}}$.

```
alpha_image = project_point(alpha_monde)
beta_image = project_point(beta_monde)
gamma_image = project_point(gamma_monde)
```

La visualisation du triangle projeté confirme que l'ensemble des transformations (extrinsèques, intrinsèques, et $\mathbf{A}$) sont correctement appliqués, positionnant le dossard dans le repère image ($\mathbf{R}_i$) avec l'orientation adéquate. On se rend ainsi compte d'un effet symétrique de l'image, en effet, l'image formée est une projection inversée (à l'envers) et renversée (gauche/droite) par rapport à l'objet réel.

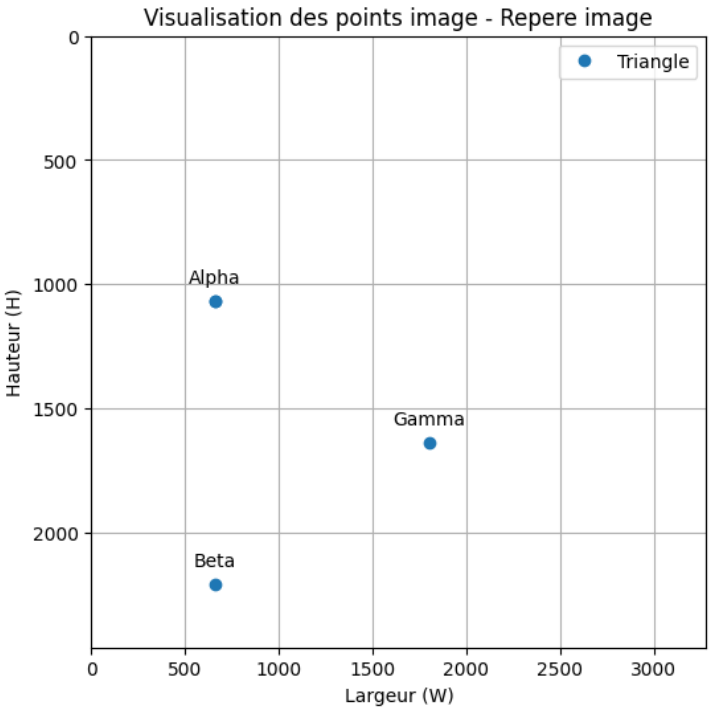


FIGURE 15 – Visualisation du triangle projeté dans le repère image ($\mathbf{R}_i$).

10 Conclusion et Perspectives

Le travail mené dans le cadre de ce projet de 4^{e} année a permis d'établir la chaîne complète de la projection d'image par caméra perspective, un jalon essentiel pour la génération d'images synthétiques destinées à l'entraînement d'une IA pour le suivi d'athlètes. Après avoir défini l'analyse des besoins et formalisé les exigences via le Cahier des Charges Fonctionnel (CdCF), nous avons validé le modèle géométrique en Python. L'implémentation a démontré le succès de la fonction FV1 (Validation de la projection) : en utilisant les matrices intrinsèques $\mathbf{K}$ et extrinsèques $[\mathbf{R}|\mathbf{t}]$ au sein d'une fonction de projection, nous avons transformé un point unique, puis une forme géométrique (un triangle) définie dans le repère Monde (3D) en coordonnées finales 2D (pixels). Ce processus confirme la faisabilité de positionner le dossard modélisé (FP3) avec une perspective mathématiquement exacte. Pour les travaux futurs, l'effort devra se concentrer sur l'automatisation de la chaîne de génération (FP2), notamment en intégrant le masquage et la variation des conditions d'environnement (météo, éclairage), pour construire la banque d'images. Idéalement, une intégration à un moteur de rendu 3D permettrait de gérer plus facilement la texture et les effets d'éclairage pour garantir le réalisme requis (FP1) de l'image finale.

11 Sources

— Musée Suisse de L'appareil Photographique :
https://www.cameramuseum.ch/ressources/base-de-connaissance/la-camera-obscura-et-son-histoire
— Vidéo YouTube : *Photographie - La Camera Obscura*
https://www.youtube.com/watch?v=F3poXuvt94E
— De Pictura :
https://fr.wikipedia.org/wiki/De_pictura
— Universiteit van Amsterdam :
https://staff.fnwi.uva.nl/r.vandenboomgaard/IPCV20162017/LectureNotes/CV/PinholeCamera/PinholeCamera.html
— OpenCV :
https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html
— Vidéo YouTube : *Computer Vision : The Camera Matrix*
https://www.youtube.com/watch?v=Hz8kz5aeQ44&t=20s
— Lien Git vers le code Python du projet :
https://github.com/Justinamenngblogni/G-n-rateur_d-image_ENSIL.git