# ACQUISITION ET VISUALISATION D'IMAGES COULEUR DE HAUTE DYNAMIQUE


**Présentation finale :**
BAYA Haytam
CHIKHI Yasmine

**ENCADRE PAR :**
**BIGUE Laurent**

# SOMMAIRE

# REMISE EN CONTEXTE

**HDR (High Dynamic Range) : Technologie permettant de capturer une large plage dynamique de luminosité.**


SOURCE : HDR10, HDR10+, HLG ET DOLBY VISION : QUELLES DIFFÉRENCES ENTRE LES STANDARDS HDR – FRANDROID.

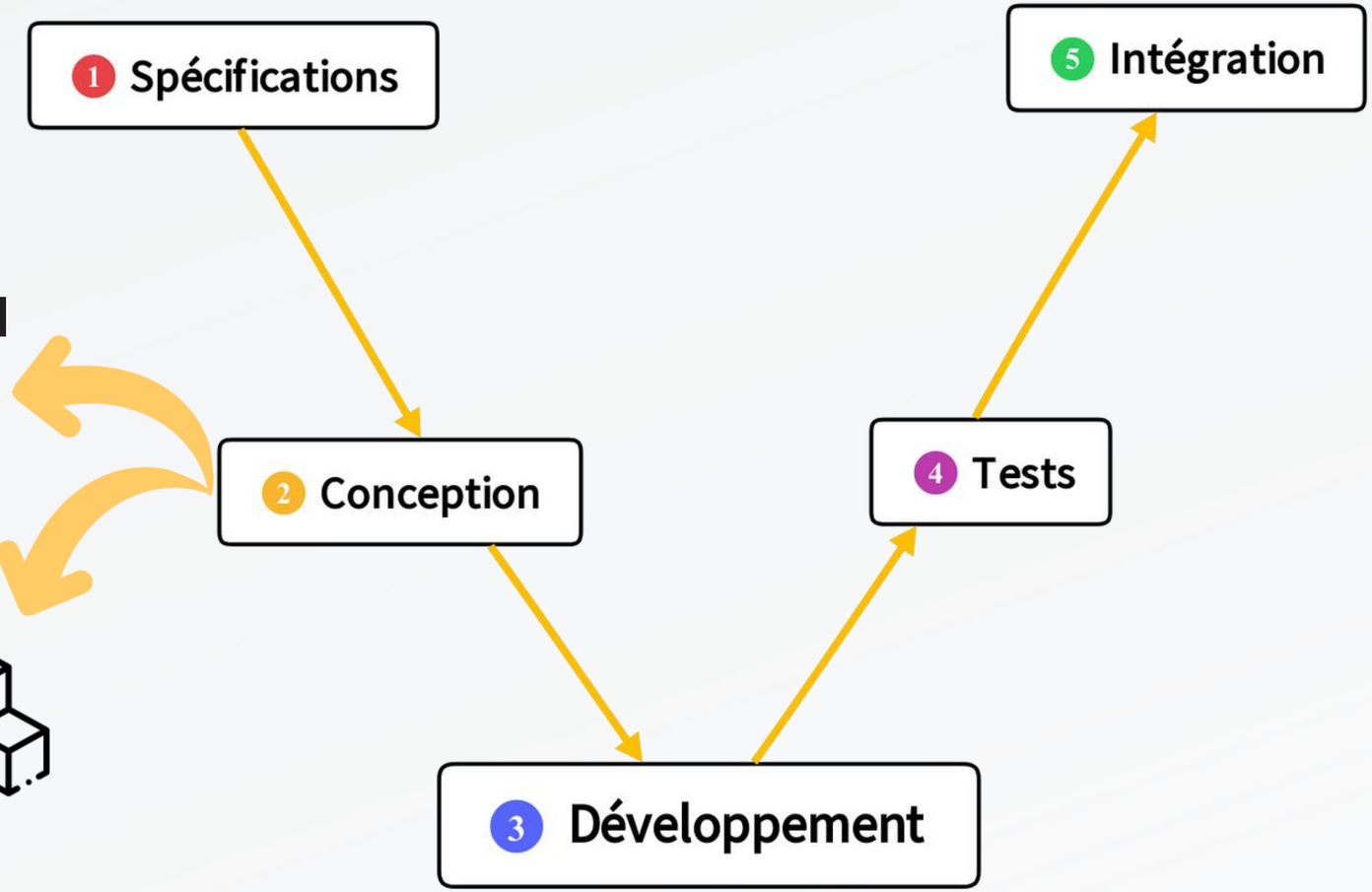

→ RGB24

→ BAYER24

**Première Partie :** Développement d'un système d'acquisition et de visualisation en temps réel d'images HDR au format **RGB24**.

**Deuxième Partie :** Intégration du format **Bayer24** au système, caractérisé par des spécifications plus contraignantes.

# METHODOLOGIE: CYCLE EN V




**ARENA API**

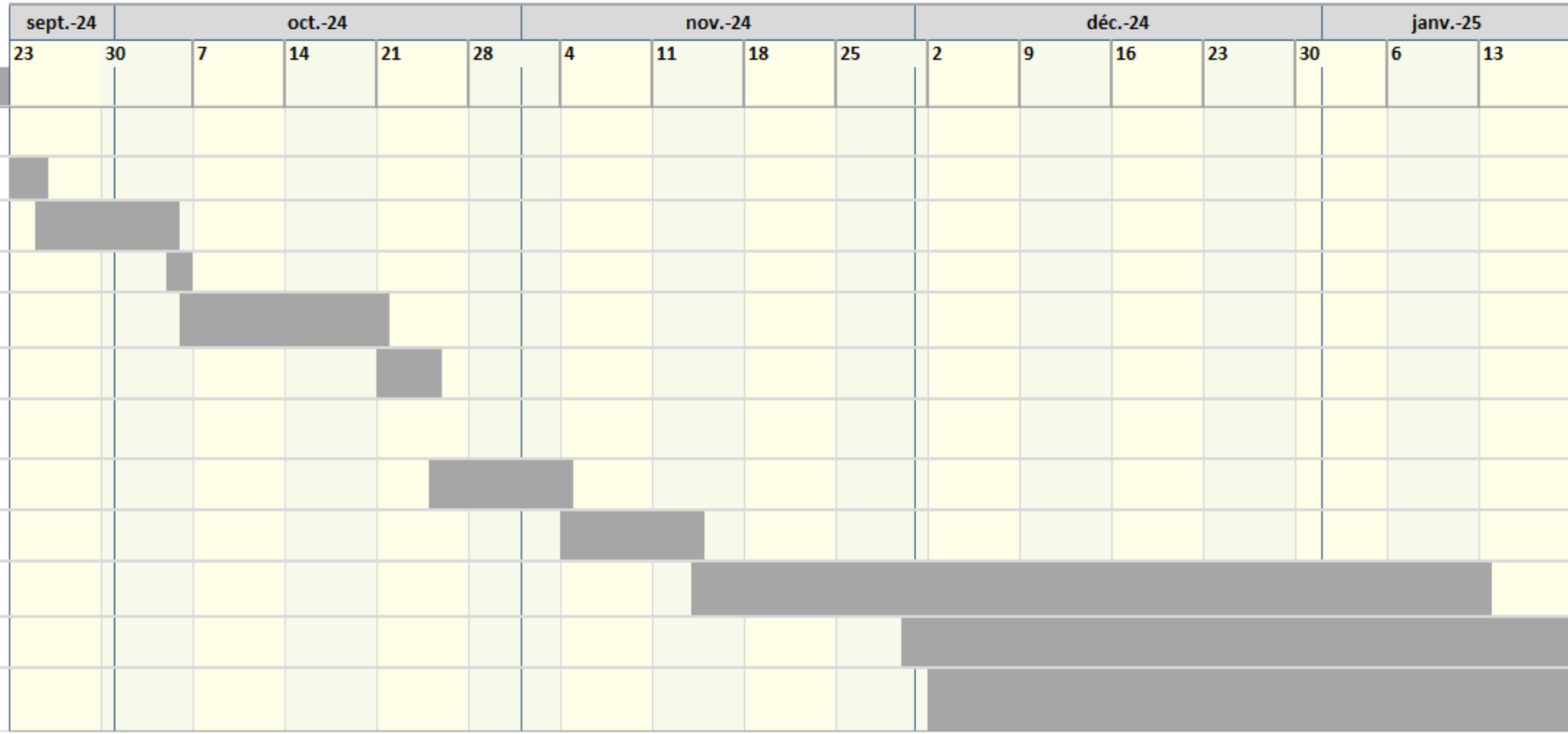
**Basée sur 3 modules :**

- **Acquisition**
- **Traitement**
- **flux temps réel**

# GESTION DE PROJET: GANTT

Acquisition et visualisation d'images couleur de haute dynamique

| Etapes: | AVANCEMENT | DÉBUT | FIN |
|---|---|---|---|
| **Phase 1 : Traitement des Images RGB** | | | |
| Rédaction des spécifications | 100% | 18/9/24 | 25/9/24 |
| Conception de l'architecture | 100% | 25/9/24 | 5/10/24 |
| Développement des modules | 100% | 5/10/24 | 6/10/24 |
| Réalisation des tests | 100% | 6/10/24 | 21/10/24 |
| Intégration des composants | 100% | 21/10/24 | 25/10/24 |
| **Phase 2 : Traitement des Images Bayer** | | | |
| Rédaction des spécifications | 100% | 25/10/24 | 4/11/24 |
| Conception de l'architecture | 100% | 4/11/24 | 14/11/24 |
| Développement des modules | 100% | 14/11/24 | 13/1/25 |
| Réalisation des tests | 100% | 30/11/24 | 22/1/25 |
| Intégration des composants | 100% | 2/12/24 | 31/1/25 |



# SPECIFICATIONS (CAHIER DES CHARGES)

**Contraintes matérielles et calcul de la performance cible du flux temps réel.**

**GIGABIT ETHERNET**

890 Mbps

BANDE PASSANTE

$$\text{FPS}_{\max} = \frac{\text{Bande passante (890 Mbps)}}{\text{Taille d'une image (16.08 Mb)}}$$

**FPS max** $\approx 6.92$ **FPS.**

- Résolution de l'image : 2880 × 1860 pixels.
- Taille par pixel : 24 bits/pixel (Bayer24).
- Taille totale d'une image : 16.08 MB.

DÉBIT D'UNE IMAGE

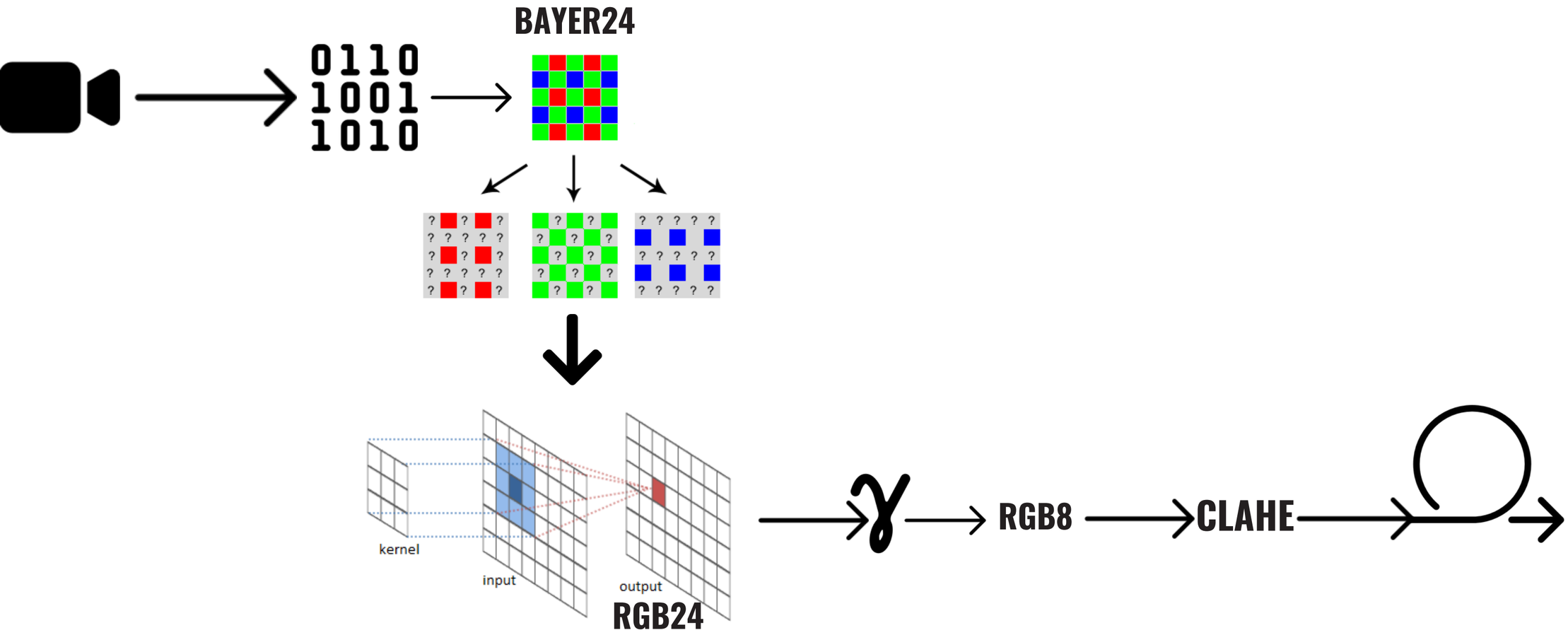
RGB

**FPS max :** 2.3 FPS.

**Il faut aller 3 fois plus vite !**

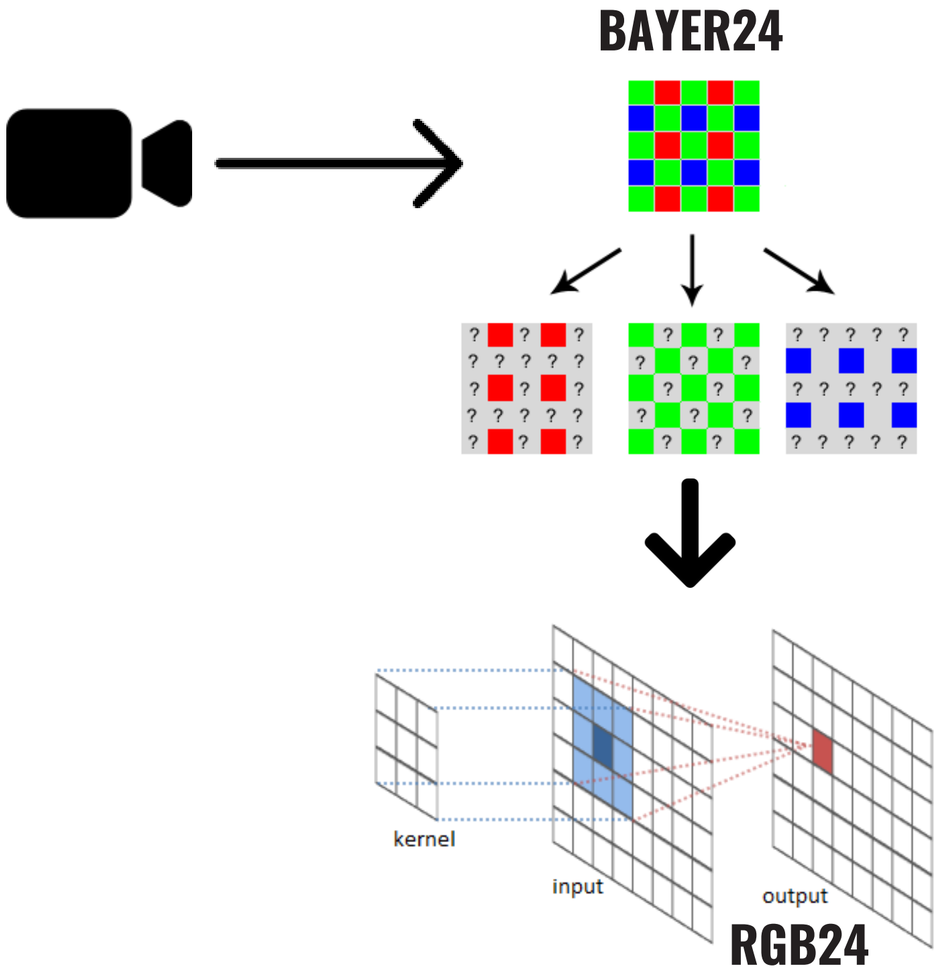
# DEVELOPPEMENT

Acquisition → Affichage → Flux temps réel



# ACQUISITION

## Acquisition et Reconstruction de l'Image

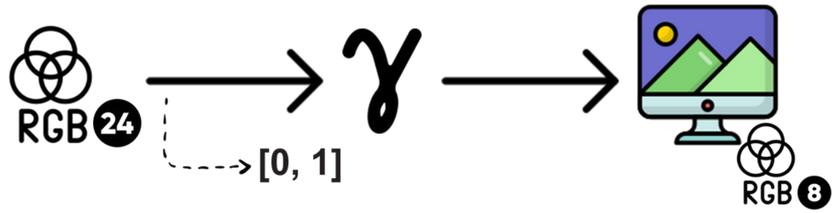



- **1 Acquisition (Bayer24) :** Le capteur capture une **image brute en mosaïque** (1 seule couleur par pixel).
- **2 Séparation des canaux :** Extraction des matrices **R, G, B**.
- **3 Dématriçage (Convolution) : Interpolation** des couleurs pour reconstruire l'image.
- **4 Image finale (RGB24) :** Pixels complets avec **R, G, B**.

Impact du coût en temps (0.01 s) sur le FPS (6.9 FPS) :

- Coût ajouté $\approx$ 0.01 s par image.
- FPS cible : 6.9 FPS.
- FPS cible après ajout : 6.45 FPS.

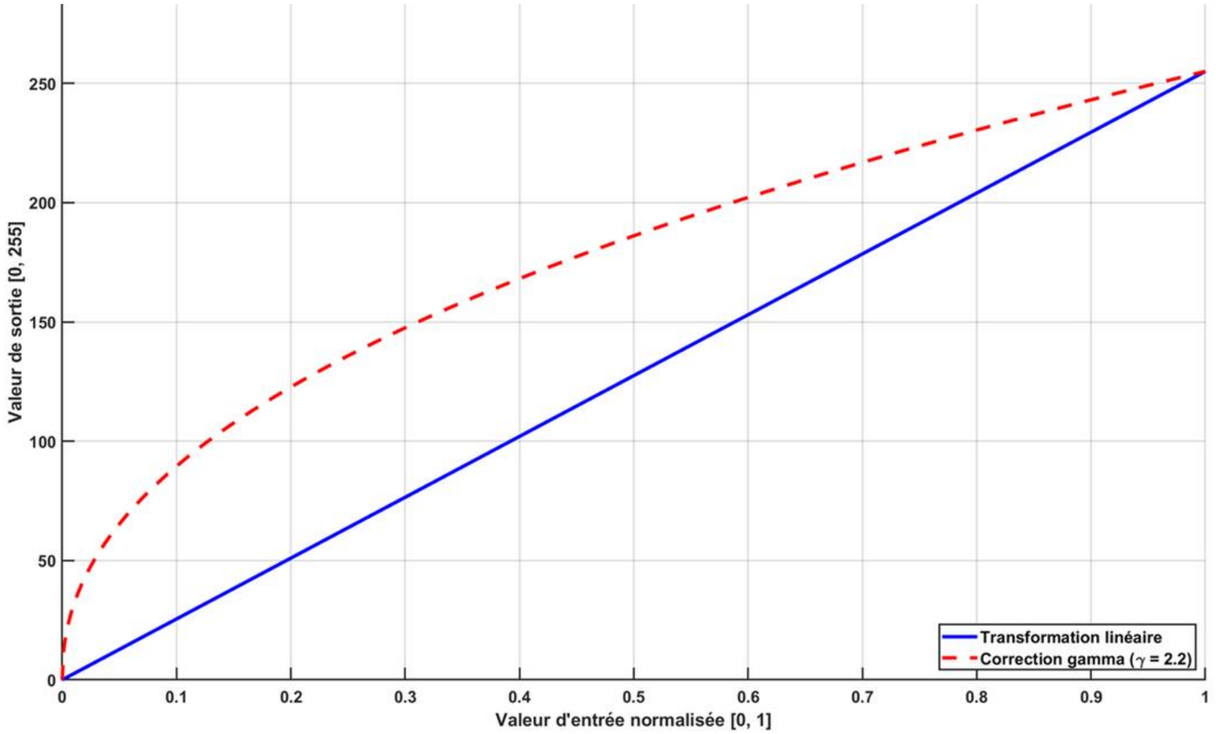
# AFFICHAGE



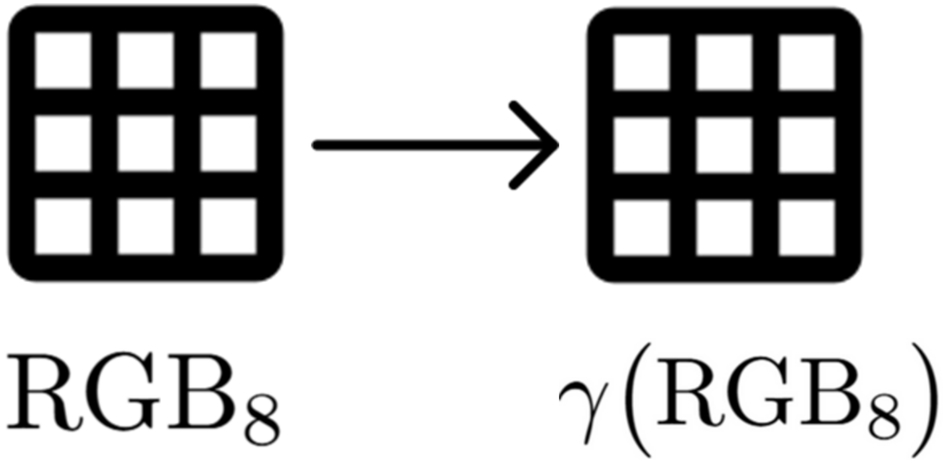

- **Formule utilisée :**

$$I_{\text{sortie}} = I_{\text{entrée}}^{\frac{1}{\gamma}}$$

- **Objectif :**
  - Éclaircir les zones sombres et compresser les hautes lumières.
  - Préserver les détails visuels tout en rendant les images compatibles avec les écrans RGB8.



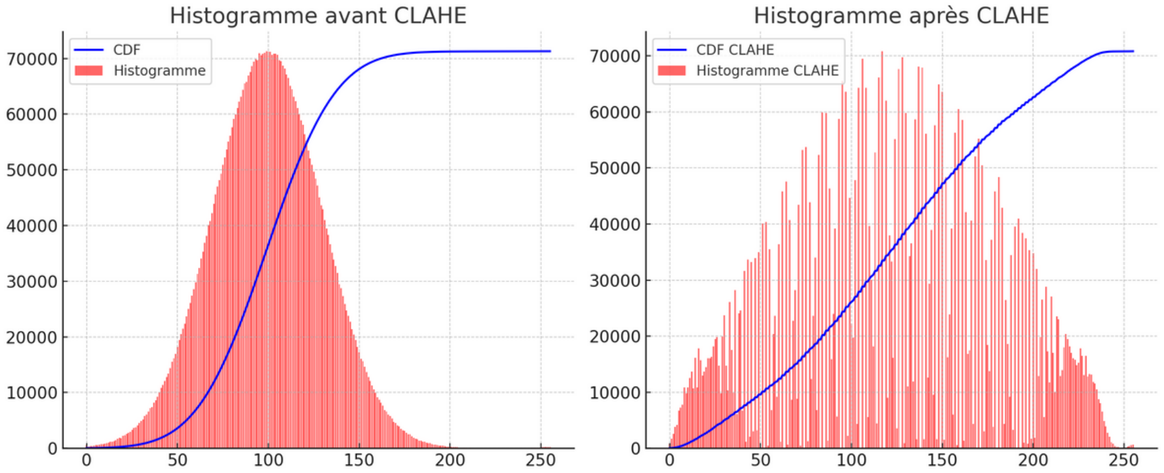

**PRECALCUL DE GAMMA: LUT**



$\mathrm{RGB}_8$ $\gamma(\mathrm{RGB}_8)$

# AFFICHAGE

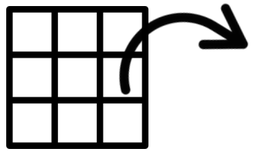
RGB8 → CLAHE







Détails plus visibles après CLAHE.



$$I' = \text{round}\left(\frac{\text{CDF}(I) - \text{CDF}_{\min}}{(M \times N) - \text{CDF}_{\min}} \times (L-1)\right)$$

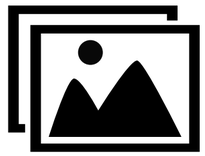
- $\text{CDF}(I)$ : Fonction de distribution cumulative (CDF) de l'intensité $I$
- $L$ : Nombre total de niveaux de gris (souvent 256 pour une image 8 bits)

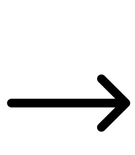
**Impact du coût en temps (0.008 s) sur le FPS (6.45 FPS) :**

- Coût ajouté ≈ 0.008 s par image.
- FPS cible : 6.45 FPS.
- FPS cible après ajout : 6.13 FPS.

# OPTIMISATIONS: COMPRESSION ET ENREGISTREMENT TEMPS REEL

## Thread principal





IMAGES TRAITEES

Mémoire tampon

CODEC FFV1

LIBÉRATION DE LA FILE D'ATTENTE

-Acquisition/Reconstruction
- Gamma
- CLAHE

Flux temps réel

## Thread encodage

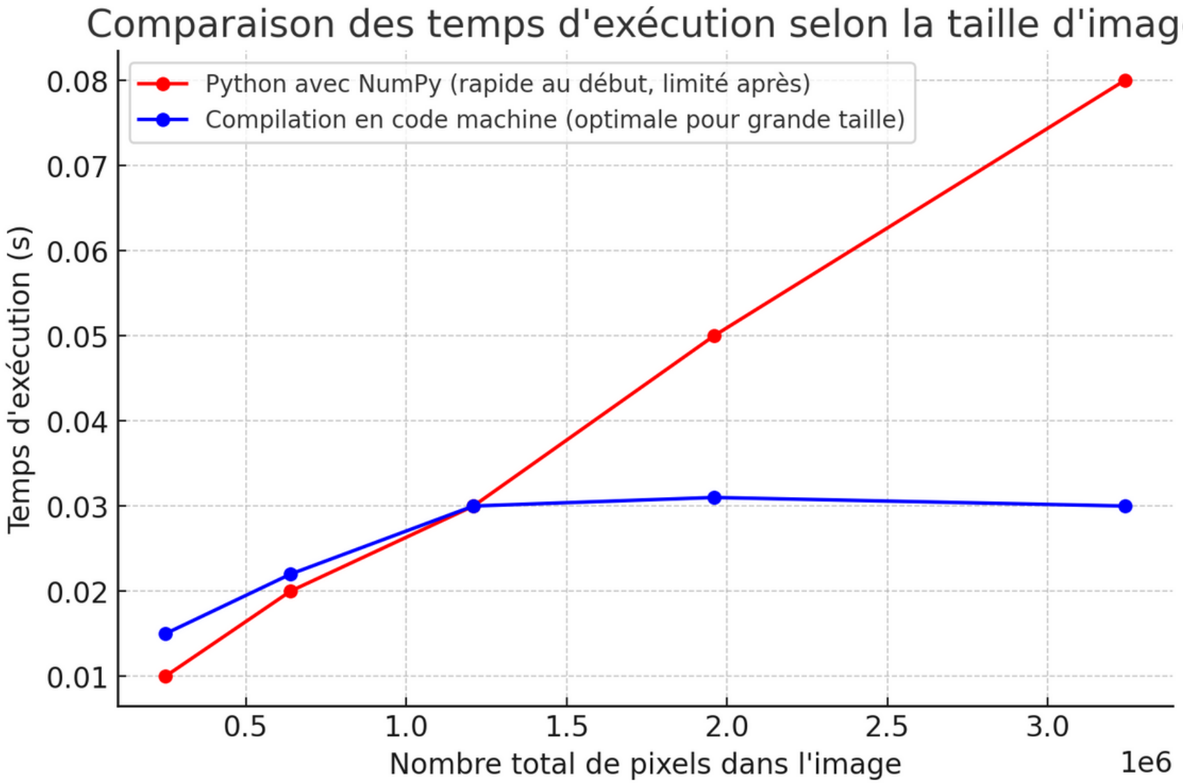
- Récupération des images
- Compression sans perte avec FFV1

ÉCRITURE DE LA VIDÉO

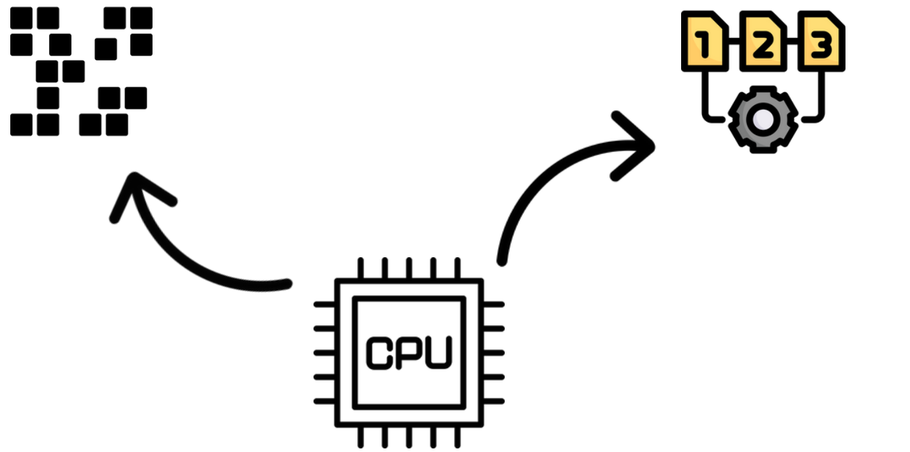
30% de gain en mémoire

Pratiquement aucune influence sur le FPS grâce à l'architecture parallélisée du programme.

# OPTIMISATIONS: APPLICATION DE LA LUT AVEC PYTHON VS CODE COMPILÉ



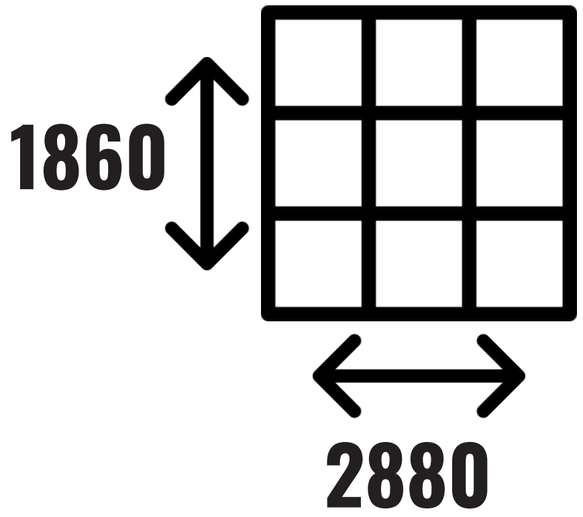

- Accès mémoire dispersé
- Accès séquentiel en mémoire (cache-friendly)



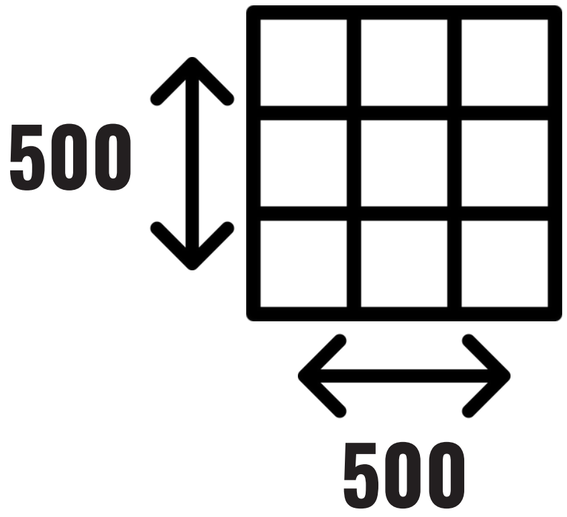

**Conclusion :**

- Avec Python (0.08s), la baisse de FPS est marquée (4.11 FPS).
- Avec Compilation en Code Machine (0.03s), on conserve une meilleure fluidité (5.18 FPS).

# Impact de la résolution sur l'efficacité des optimisations




- **Python** : 4,1 FPS
- **Code compilé** : 5,15 FPS




- **Python** : 10 FPS
- **Code compilé** : 20 FPS

× 2

# RESULTAT: INTERFACE LOGICIEL




# CONCLUSION

**Points Forts :**

- **Compatibilité** : Fonctionne avec tout appareil GenICam via des **fonctions génériques**.
- **Modularité** : Optimisations facilement intégrables.
- **Adaptabilité** : Gestion dynamique des résolutions sans perte de stabilité.
- **Accessibilité** : Fonctionne sans GPU ni matériel spécifique.

**Perspectives :**

- **Optimisation du FPS** : Mieux séparer la logique du programme de la charge machine.
- **IA au lieu de CLAHE** : Éviter l'amplification du bruit.

# MERCI !

# Calcul de la CDF

La fonction de distribution cumulative (CDF) d'un niveau de gris $I$ se définit par :

$$\mathrm{CDF}(I) = \frac{1}{M \times N} \sum_{k=0}^{I} h(k),$$

où :

- $h(k)$ est l'histogramme, c'est-à-dire le nombre de pixels dont la valeur d'intensité est $k$.
- $M \times N$ est le nombre total de pixels de l'image (largeur $\times$ hauteur).

La CDF est donc une fonction qui, pour chaque niveau de gris $I$, renvoie la proportion cumulative de pixels ayant une intensité inférieure ou égale à $I$.

# Noyaux de Convolution pour le Dématriçage

## Noyaux Utilisés

**R/B (`kernel_r_b`) :**

$$\begin{bmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ \frac{1}{2} & 1 & \frac{1}{2} \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \end{bmatrix}$$

**G (`kernel_g`) :**

$$\begin{bmatrix} 0 & \frac{1}{4} & 0 \\ \frac{1}{4} & 1 & \frac{1}{4} \\ 0 & \frac{1}{4} & 0 \end{bmatrix}$$

## Pourquoi ces valeurs ?

L'interpolation bilinéaire privilégie les voisins directs ($\frac{1}{2}$) par rapport aux diagonales ($\frac{1}{4}$), assurant une reconstruction rapide et fluide.