**Université Clermont Auvergne**

**École Universitaire de Physique et d'Ingénierie**

**Diplôme** **: Master Imagerie et Technologie pour la Médecine (TechMed)**



Travail De Recherche Encadré

**Présenté par :** LEHARA Lyes

**Thème :** Bibliothèque de traitement d'images en C++

**Dirigé par :** Mme. PERY Emile

**Année universitaire :** 2024/2025.

Table de matière

[1. Définition d’une image : 4](#_Toc199779750)

[**1.1** **Codage d’une image (image numérique) :** 4](#_Toc199779751)

[2. Présentation de Projet : 5](#_Toc199779752)

[**2.1** **Namespace version 1.1 :** 5](#_Toc199779753)

[2.1.1 Définition : 5](#_Toc199779759)

[2.1.2 Implémentation des fonctions : 5](#_Toc199779760)

[2.1.3 Allocation d’images 5](#_Toc199779761)

[2.1.4 Création d'images particulières : 6](#_Toc199779762)

[2.1.5 Lecture et écriture des fichiers images au format brut (.RAW) : 8](#_Toc199779763)

[2.1.6 Conversion d’image : 9](#_Toc199779764)

[2.1.7 Conversion d'une image en fausses couleurs à l'aide d'une LUT : 10](#_Toc199779765)

[2.1.8 Tester des filtres LUT sur les différentes images : 12](#_Toc199779766)

[**2.2** **Namespace version 1.1 :** 13](#_Toc199779767)

[2.2.1 Introduction 13](#_Toc199779768)

[2.2.2 Implémentation des fonctionnalités de namespace v1.0 dans le cadre d’une class 13](#_Toc199779769)

[**2.3** **Namespace version 2.0 :** 14](#_Toc199779770)

[2.3.1 Implémentation des traitements d’images 14](#_Toc199779771)

[**2.4** **Conclusion et perspectives :** 20](#_Toc199779772)

[**2.5** **Bibliographie** 20](#_Toc199779773)

Liste des figures

[**Figure 1: Représentation d’image numérique.** 5](file:///C:\Users\lylehara\Downloads\TER-master\TER-master\Travaux%20Encadré%20de%20Recherche.docx#_Toc199668147)

[**Figure 2: Fonction d’allocation d’image.** 5](#_Toc199668148)

[**Figure 3: Création d’image blanche.** 6](file:///C:\Users\lylehara\Downloads\TER-master\TER-master\Travaux%20Encadré%20de%20Recherche.docx#_Toc199668149)

[**Figure 4: Création d’image damier.** 6](#_Toc199668150)

[**Figure 5: Création d’image sinusoïdale.** 7](#_Toc199668151)

[**Figure 6: Aperçu d’une image sinusoïdal** 7](file:///C:\Users\lylehara\Downloads\TER-master\TER-master\Travaux%20Encadré%20de%20Recherche.docx#_Toc199668152)

[**Figure 7: Aperçu d’une image damier** 7](file:///C:\Users\lylehara\Downloads\TER-master\TER-master\Travaux%20Encadré%20de%20Recherche.docx#_Toc199668153)

[**Figure 8: Aperçu d’une image Blanche** 7](file:///C:\Users\lylehara\Downloads\TER-master\TER-master\Travaux%20Encadré%20de%20Recherche.docx#_Toc199668154)

[**Figure 9: Lecteur d’un fichier brut** 8](#_Toc199668155)

[**Figure 10: Ecriture d’un fichier brut** 8](#_Toc199668156)

[**Figure 11: Conversion d’une image d’un type à un autre.** 9](#_Toc199668157)

[**Figure 12: Conversion d’une image RGB en niveau de gris** 9](#_Toc199668158)

[**Figure 13: Chargement de fichier LUT.** 10](file:///C:\Users\lylehara\Downloads\TER-master\TER-master\Travaux%20Encadré%20de%20Recherche.docx#_Toc199668159)

[**Figure 14: Application de LUT sur l’image.** 10](#_Toc199668160)

[**Figure 15: Conversion d’un pixel gris en RGB via une LUT** 11](file:///C:\Users\lylehara\Downloads\TER-master\TER-master\Travaux%20Encadré%20de%20Recherche.docx#_Toc199668161)

[**Figure 16: Résultat de l’application de LUT sur les différentes images** 12](#_Toc199668162)

[**Figure 17: Aperçu du code de la class Image** 13](#_Toc199668163)

[**Figure 18: Aperçu de code de la classe ImageRGB** 14](#_Toc199668164)

[**Figure 19: Aperçu de code de la classe Processing 1** 14](#_Toc199668165)

[**Figure 20: Aperçu de code de la classe Processing 2** 15](#_Toc199668166)

[**Figure 21: Aperçu de code de la classe Addition avec un scalaire** 15](#_Toc199668167)

[**Figure 22: Aperçu de code de la classe addition de deux images** 16](#_Toc199668168)

[**Figure 23: Résultat de l'addition de deux images de différentes dimensions** 16](file:///C:\Users\lylehara\Downloads\TER-master\TER-master\Travaux%20Encadré%20de%20Recherche.docx#_Toc199668169)

[**Figure 24: Résultat d'addition de deux images damier et sinusoïdal** 16](file:///C:\Users\lylehara\Downloads\TER-master\TER-master\Travaux%20Encadré%20de%20Recherche.docx#_Toc199668170)

[**Figure 25 : Aperçu de code de la classe égalisation d’histogramme** 18](file:///C:\Users\lylehara\Downloads\TER-master\TER-master\Travaux%20Encadré%20de%20Recherche.docx#_Toc199668171)

[**Figure 26 Image avant l’égalisation d’histogramme** 18](file:///C:\Users\lylehara\Downloads\TER-master\TER-master\Travaux%20Encadré%20de%20Recherche.docx#_Toc199668172)

[**Figure 27 Histogramme de l'image avant l'égalisation** 18](file:///C:\Users\lylehara\Downloads\TER-master\TER-master\Travaux%20Encadré%20de%20Recherche.docx#_Toc199668173)

[**Figure 28: Histogramme de l’image après l’égalisation** 18](file:///C:\Users\lylehara\Downloads\TER-master\TER-master\Travaux%20Encadré%20de%20Recherche.docx#_Toc199668174)

[**Figure 29: Image après l’égalisation d'histogramme** 18](file:///C:\Users\lylehara\Downloads\TER-master\TER-master\Travaux%20Encadré%20de%20Recherche.docx#_Toc199668175)

[**Figure 30: Aperçu de la class Convolution** 19](#_Toc199668176)

1. Introduction :

Le traitement d’images, qui est un sous-domaine du traitement du signal, englobe l'ensemble des méthodes et techniques appliquées aux images et vidéos dans le but d'en extraire des informations pertinentes ou d'améliorer leur perception visuelle. Avant toute phase de traitement, un prétraitement est souvent requis pour optimiser la qualité des images. Cela comprend, par exemple, des opérations de rehaussement de contraste, de suppression de bruit, de correction du flou, ainsi que des techniques de segmentation ou d'extraction de contours visant à isoler les éléments significatifs d'une image.

Ce rapport fournit une synthèse des différentes versions évolutives de notre bibliothèque de traitement d'images en C++. La version de base de notre projet repose sur une architecture fonctionnelle utilisant des templates, tandis que les versions ultérieures introduisent une approche orientée objet, en intégrant deux classes, Image et ImageRGB, ainsi que des surcharges d'opérateurs.

La dernière étape de notre projet consiste à mettre en œuvre des méthodes de prétraitement (rehaussement de contraste, suppression de bruit, correction du flou), ainsi que des techniques de segmentation et d'extraction de contours, sur diverses images.

1. Définition d’une image :

Une image, c’est une représentation visuelle d’une personne ou d’un objet, réalisée par des moyens comme la peinture, le dessin, la photo ou la vidéo. C’est aussi un ensemble organisé d’informations qui, une fois affichées à l’écran, forment quelque chose que l’œil humain peut reconnaître.

## Codage d’une image (image numérique) :

Une image numérique est une représentation visuelle d’un objet ou d’une scène, composée d’une grille de petits éléments appelés pixels. Chaque pixel contient une information, comme une couleur ou un niveau de gris, qui permet de reconstituer l’image dans son ensemble.

Cette image peut provenir d’une photo, d’un dessin ou d’une vidéo, et devient numérique grâce à un processus appelé numérisation, qui convertit une image réelle (analogique) en une matrice de valeurs numériques.

Mathématiquement, une image numérique est représentée par une fonction à deux dimensions, **f(x, y)**, où chaque **point (x, y)** correspond à un pixel, et **f(x, y)** indique l’intensité lumineuse à ce point. Cela signifie que chaque pixel est une mesure de la lumière captée à un endroit précis de l’image.



**Figure 1: Représentation d’image numérique.**

1. Présentation de Projet :

Le projet a été conçu pour faciliter et structurer le code destiné au traitement d'images avec C++.

Chaque version de la bibliothèque est encapsulée dans un bloc **namespace (v1.0**, **v1.1** et **v2.0).** Cela permet d’éviter les conflits de noms entre les différentes versions. L’utilisation des **templates** permet de créer **des fonctions** et des **classes génériques**, pour manipuler les différents types de données d’images

Il est aussi organisé en plusieurs **classes de base (mère) et classes dérivées**, afin de mettre en œuvre les principes de la programmation orienté objet.

1. Namespace version 1.1 :

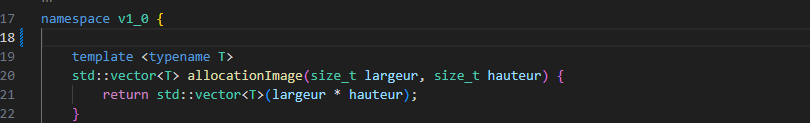
## Définition :

La version 1.0 utilise des fonctions de base pour travailler avec des images. Elle permet de créer différentes images comme des images blanches, des damiers ou encore des images sinusoïdales. Elle peut aussi lire et enregistrer des fichiers d’image au format brut (. Raw). En plus, elle peut transformer des images de 16 bits ou plus, en images plus simples de 8 bits. Elle permet aussi d’appliquer des filtres de couleur LUT. Toutes ces opérations sont faites en utilisant des tableaux dynamiques **std::vector** en C++.

## Implémentation des fonctions :

### Allocation d’images

Cette fonction sert à allouer dynamiquement une image sous forme d’un vecteur à une seule dimension dont la taille correspond à la largeur multipliée par la hauteur de l’image. C’est une fonction de type générique (Template), elle peut fonctionner avec n’importe quel type de données (int, float, uint16\_t, etc.).



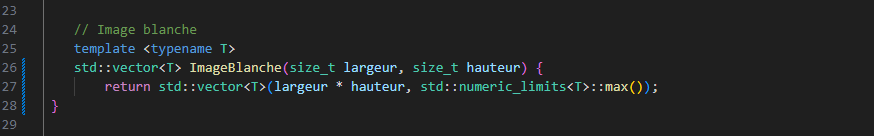
**Figure 2: Fonction d’allocation d’image.**

### Création d'images particulières :

#### Image Blanche

La fonction **ImageBlanche** permet de créer une image blanche de taille **hauteur** multiplié par la **longueur.** Chaque pixel est initialisé avec la valeur maximale possible du type générique *T* (**grâce à std::numeric\_limits<T>::max()**), ce qui correspond à la couleur blanche en niveaux de gris.

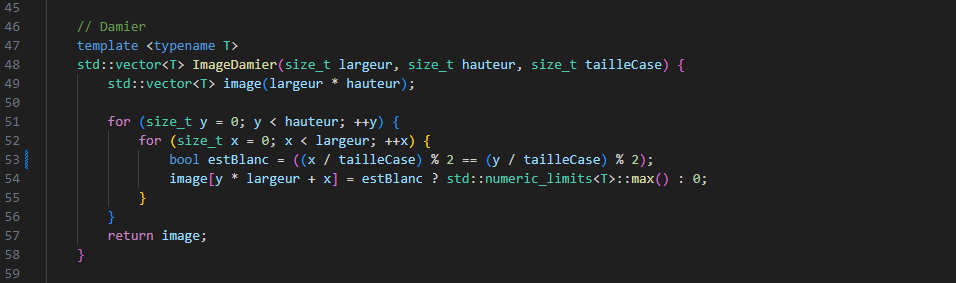
Exemple :

* + Si T = **uint8\_t**, alors max = 255
  + ****Si T = **uint16\_t**, alors max = 65535

**Figure 3: Création d’image blanche.**

#### Image Damier

Cette fonction générique ***ImageDamier*** crée une image en damier de dimensions données (haueur multiplié par longueur de l’image). Elle utilise deux boucles pour parcourir chaque pixel, et détermine si le pixel appartient à une case blanche ou noire en divisant les indices x et *y* par la taille de la case (**tailleCase**) modulo 2, puis en comparant leurs parités. Si la case est blanche, le pixel prend la valeur maximale 255 (blanc) ; sinon, il prend 0 (noir). L’image ainsi remplie est ensuite retournée.

****

**Figure 4: Création d’image damier.**

#### Image Sinusoïdale

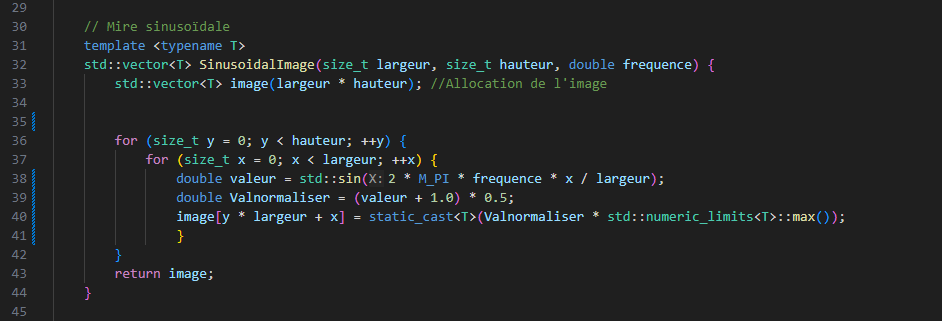
La fonction **SinusoïdaleImage** génère une image dont l’intensité des pixels varie selon une onde sinusoïdale. Elle prend en paramètre la largeur, la hauteur et la fréquence de l’onde qui représente le nombre de période de la fonction sinus.

* Pour chaque pixel, elle calcule une valeur sinus entre -1 et 1, puis la normalise entre 0 et 1 car les images utilisent des valeur positive (0 à 255) :

**double valeur = std::sin(2 \* M\_PI \* frequency \* x / width);**

**double valnormaliser = (val + 1.0) \* 0.5;**

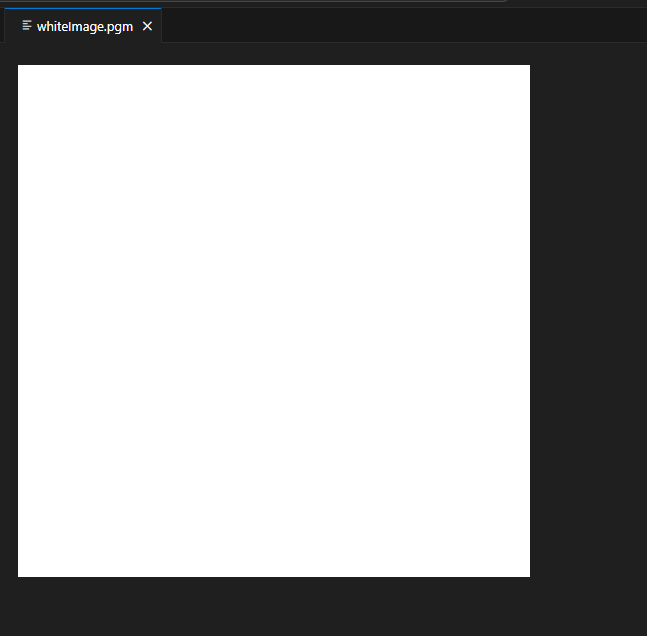
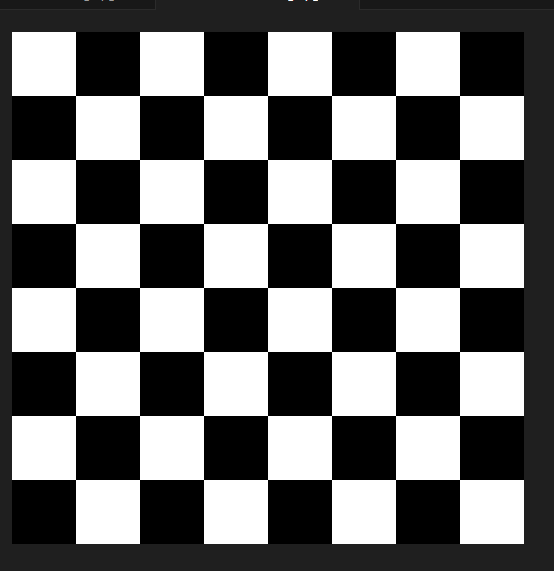
Puis la convertit en une intensité adaptée au type T (par exemple entre 0 et 255). L’image ainsi remplie est ensuite retournée.

****

**Figure 5: Création d’image sinusoïdale.**

* **Les Resultats obtenus**

Les figures ci-dessus montrent le résultat obtenu après l’appel aux fonctions.



**Figure 7: Aperçu d’une image damier**

**Figure 6: Aperçu d’une image sinusoïdal**

**Figure 8: Aperçu d’une image Blanche**

### Lecture et écriture des fichiers images au format brut (.RAW) :

#### Fichier Raw :

Un **fichier RAW** (du mot anglais *raw* signifiant *brut*) est un type de fichier généré par un appareil photo numérique ou un scanner, qui contient les données **brutes** captées par le capteur, **sans aucun traitement**.

#### Lecteur de fichiers images au format brut

La fonction ***lireImageRaw*** lit une image au format RAW et retourne ses pixels dans un vecteur. Elle prend en paramètre le nom du fichier **nomfichier**, les dimensions de l’image (la **haueur** et la **longueur**) et **bigEndian** comme Boolean.

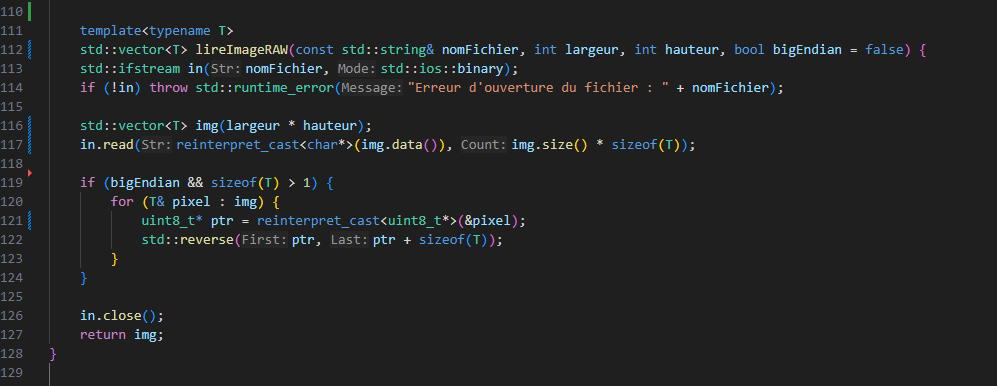
**Big** et **little endian** sont deux manières différentes de stocker les **octets** d’une valeur multi-octets (par exemple un entier sur 2 ou 4 octets) dans un fichier ou une mémoire.

* **Big-endian** → On stocke l’**octet le plus significatif** en premier (en tête).
* **Little-endian** → On stocke l’**octet le moins significatif** en premier.

**Exemple :** si on veut stocker la valeur **0x1234** (sur 2 octets) :

* En **big-endian** : 12 34
* En **little-endian** : 34 12

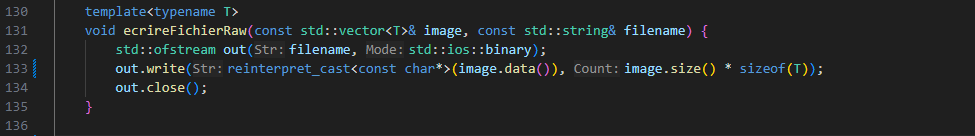
Ensuite, Elle lit les données pixel par pixel, en convertissant correctement les octets selon l’ordre spécifié, puis sauvegarde les valeurs dans un vecteur. Si le fichier ne s’ouvre pas, elle renvoie une image vide.



**Figure 9: Lecteur d’un fichier brut**

#### Ecriture de fichiers images au format brut

La fonction ***ecrireFichierRaw*** permet d’enregistrer une image (stockée dans un vecteur) dans un fichier brut (.Raw). Elle prend en paramètre l’image et le nom du fichier de destination



**Figure 10: Ecriture d’un fichier brut**

### Conversion d’image :

#### Conversion d’image d’un type à un autre :

La fonction ***convertImage*** permet de convertir une image représentée sous forme de vecteur (std::vector) de pixels d’un type source (SrcType : type des pixels d’entrée comme **uint8\_t, float,** etc.) vers un type destination (DstType : type des pixels de sortie **uint8\_t, float,** etc. ), avec une option pour ajuster dynamiquement la plage des valeurs (dynamique).

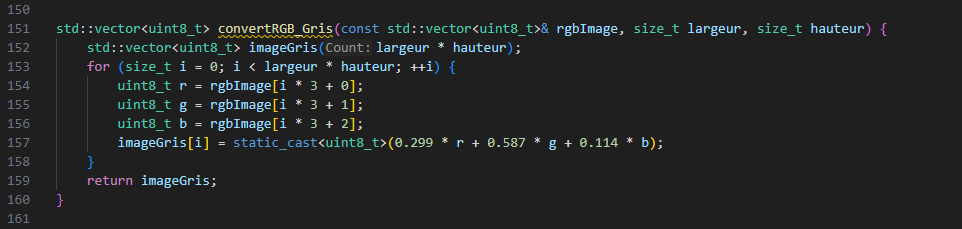


**Figure 11: Conversion d’une image d’un type à un autre.**

#### Conversion d’image de RGB vers niveau de gris :

La fonction ***convertRGB\_Gris*** transforme une image couleur **RGB** (stockée dans un **std::vector<uint8\_t>)** en une image en **niveaux de gris**. Elle parcourt chaque pixel de l’image puis, elle extrait les composantes rouge, verte et bleue, puis calcule une seule valeur de gris en appliquant une formule pondérée (0.299 pour le rouge, 0.587 pour le vert et 0.114 pour le bleu).

Enfin elle retourne le résultat comme une image en noir et blanc.



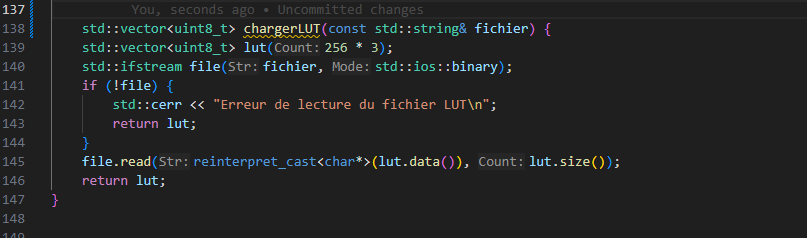
**Figure 12: Conversion d’une image RGB en niveau de gris**

### Conversion d'une image en fausses couleurs à l'aide d'une LUT :

#### Fichier LUT :

Une **LUT** (*Look-Up Table*, ou table de correspondance en français), est un tableau de valeurs numériques utilisé en traitement d’image, vidéo ou graphisme. Elle permet de modifier facilement les couleurs ou la luminosité de l’image.

#### Chargement de fichier LUT (le filtre LUT) :

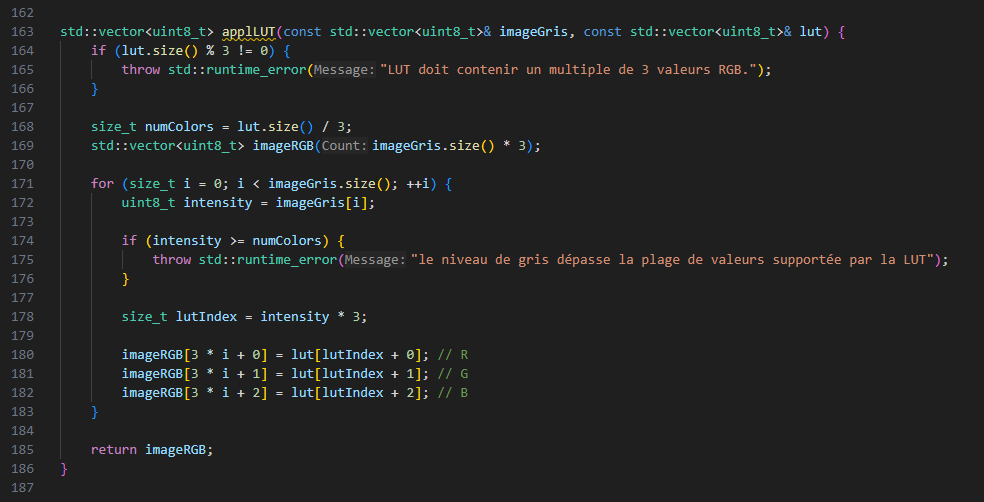


**Figure 13: Chargement de fichier LUT.**

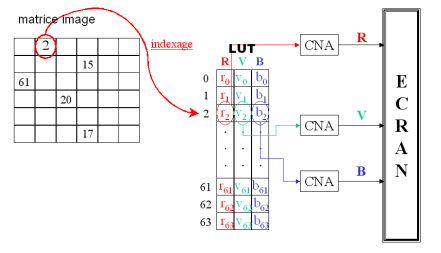
#### Application de fausse couleur à l’aide de LUT

La fonction ***applyLUT*** transforme une image de niveaux de gris en image couleur en remplaçant chaque pixel gris par une couleur définie dans une table LUT, selon son intensité.

Les **figures ci-dessous** montrent le code de la fonction **applyLUT** ainsi que le **schéma fonctionnel** illustrant le calcul des valeurs **RGB** à partir d’un pixel en niveaux de gris à l’aide de la table LUT.

****

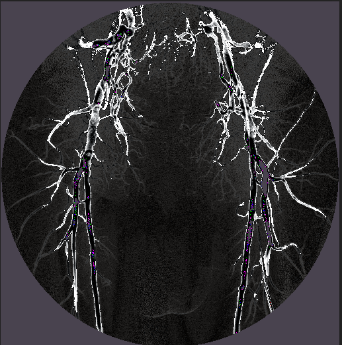
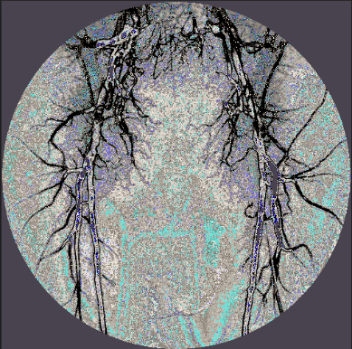
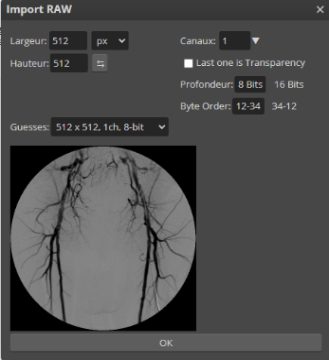
**Figure 14: Application de LUT sur l’image.**

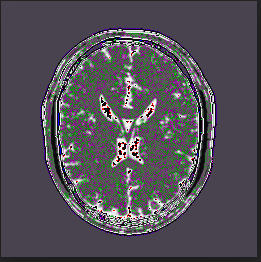
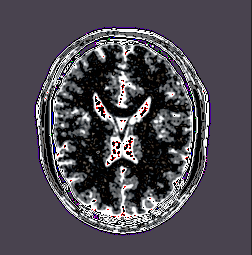
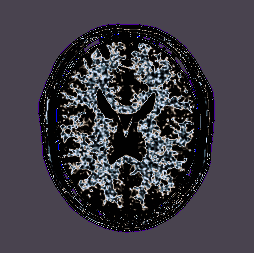
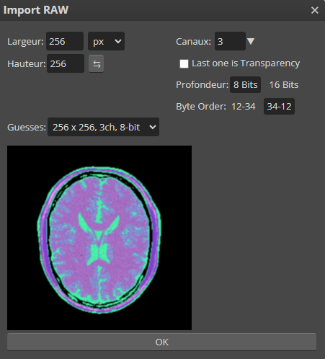


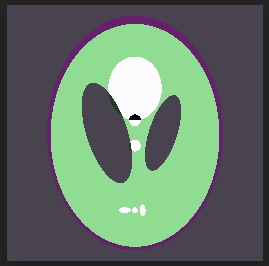
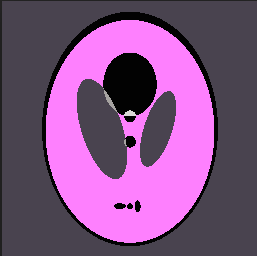
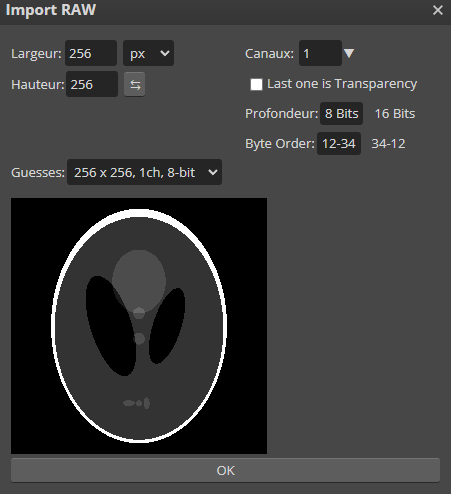
**Figure 15: Conversion d’un pixel gris en RGB via une LUT**

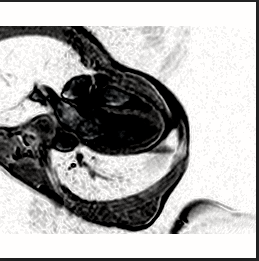
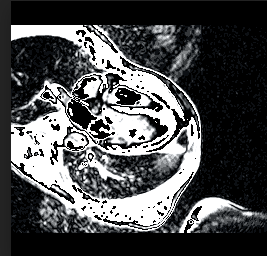
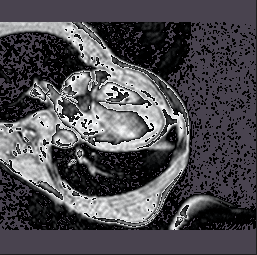
### Tester des filtres LUT sur les différentes images :

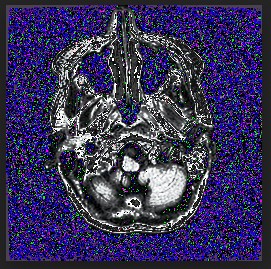
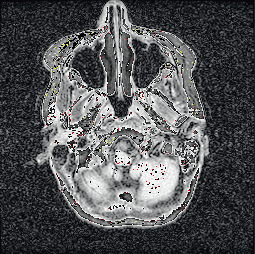
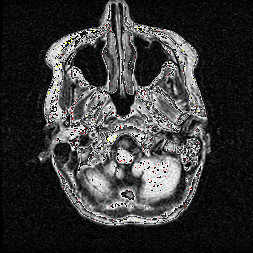
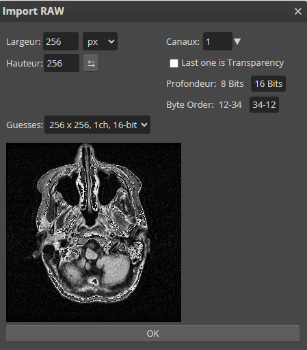
Les figures suivantes montrent le résultat obtenu après l’appel aux fonctions précédentes











**Figure 16: Résultat de l’application de LUT sur les différentes images**

## Namespace version 1.1 :

### Introduction

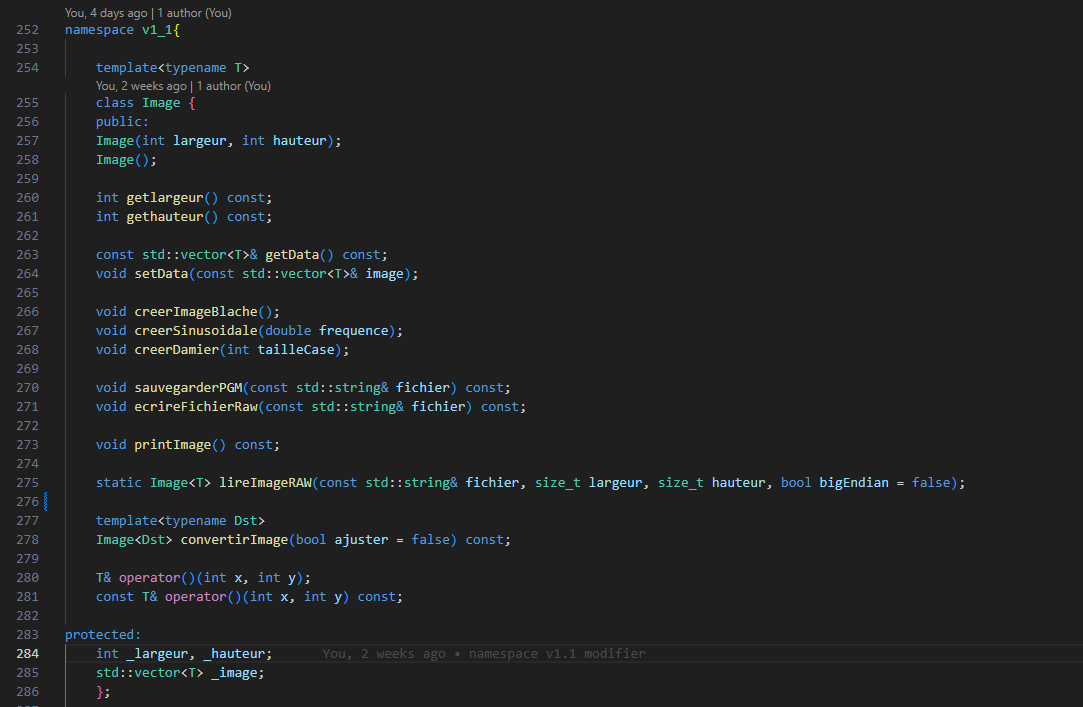
L’objectif du namespace v1\_1 est de reprendre les mêmes fonctionnalités de v1\_0 en adoptant une approche orientée objet. On utilise une classe générique pour coder des images en utilisant ***std::vector*,** ce qui rend la gestion plus claire et structurée.

Deux classes principales sont définies : ***Image***, qui gère les opérations de base (allocation d’image, création des images, lecture/écriture des fichier brut, conversion de type à un autre), et ***ImageRGB***, qui hérite de la class ***Image*** pour traiter les images couleur et appliquer des ***LUT*** spécifiques aux images en niveau de gris.

### Implémentation des fonctionnalités de namespace v1.0 dans le cadre d’une class

L’image ci-dessus représente la class Image de type ***template*** (un type générique), qui peut fonctionner avec n’importe quel type de données (float, uint8\_t, uint16\_t, …), elle encapsule les dimensions de l’image (\_**largeur**, \_**hauteur**), ainsi que la structure de stockage des pixels avec ***std::vector*** qui sert à stocker tous les pixels de l’image sous forme d’un tableau comme attributs de la class.

Elle fournit aussi des méthodes pour la création d’images (**blanche**, **sinusoïdale**, **damier**), la lecture/écriture de fichiers (**PGM**, **RAW**), l’impression, la conversion de type d’image, et la surcharge de l’opérateur **parenthèse ()** pour accéder aux ***pixel (i, j)***.

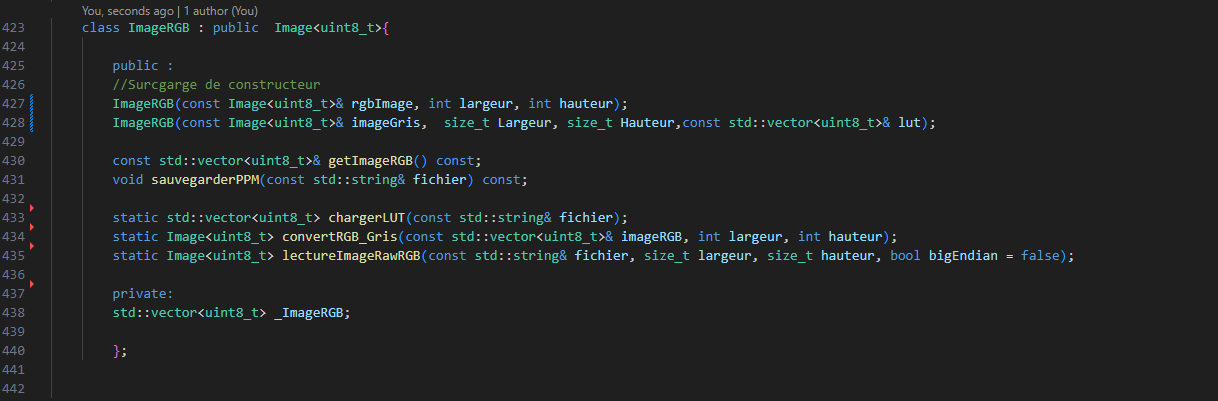


**Figure 17: Aperçu du code de la class Image**

La class ***ImageRGB*** est une class dérivé de la class ***Image*** conçue pour gérer les images en couleurs ***RGB***, elle prend une propriété ***\_ImageRGB*** pour accéder aux données de pixels d’une image ***RGB***. Elle inclut également des méthodes pour lire/écrire des fichiers PPM, charger une LUT binaire, convertir une image RGB en niveaux de gris.

Cette classe propose une **surcharge de constructeurs** :

* Le **premier constructeur** permet de créer une image **RGB** vide à partir d’une taille spécifiée (largeur \* hauteur).
* Le **deuxième constructeur** permet de **convertir une image en niveaux de gris en une image RGB** en utilisant une **LUT**.



**Figure 18: Aperçu de code de la classe ImageRGB**

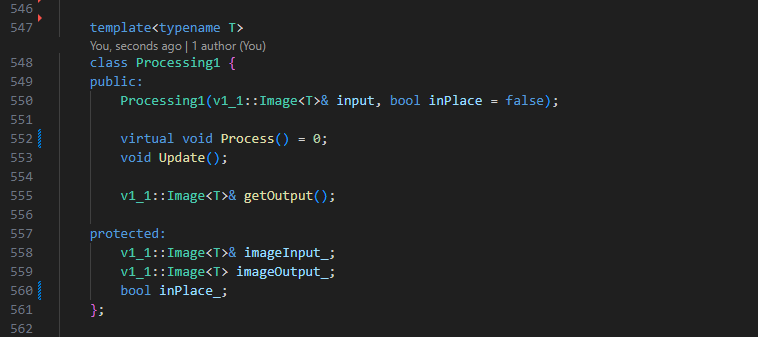
## Namespace version 2.0 :

Cette dernière version de la bibliothèque contient les bases de traitement d’image tel que, l’addition des images, égalisation d’histogramme et le filtre convolution.

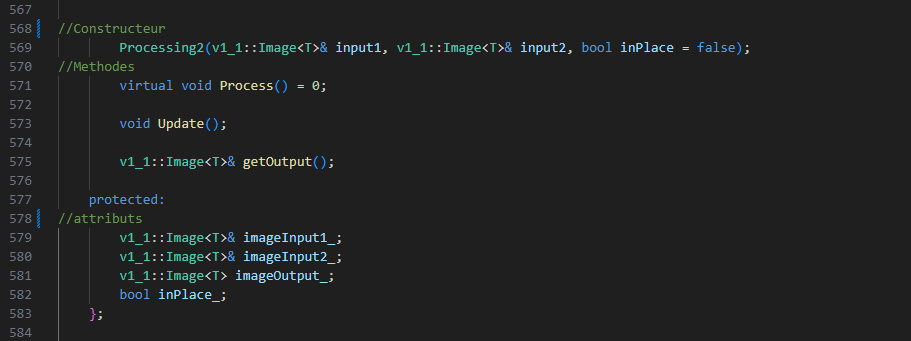
### Implémentation des traitements d’images

#### Processing 1&2

Les traitements utilisés dans cette version sont représentés sous forme d’une class template qui prend en paramètre aux plus deux images. Ces classes héritent soit de la classe abstraite Processing1, pour les traitements nécessitant une seule image en entrée, soit de la classe Processing2, pour ceux nécessitant deux images en entrée.



**Figure 19: Aperçu de code de la classe Processing 1**

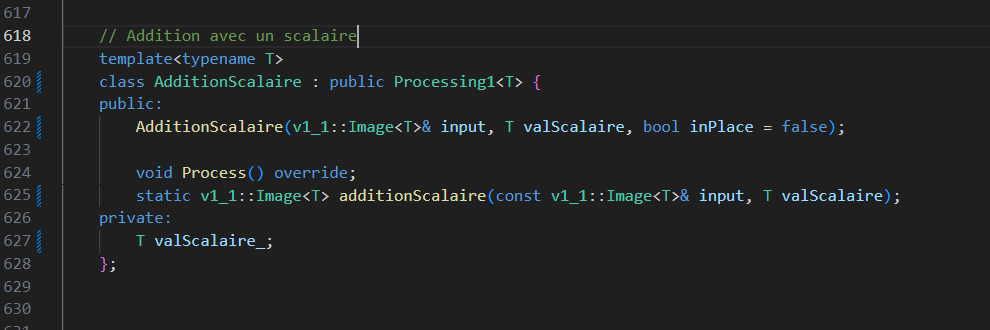


**Figure 20: Aperçu de code de la classe Processing 2**

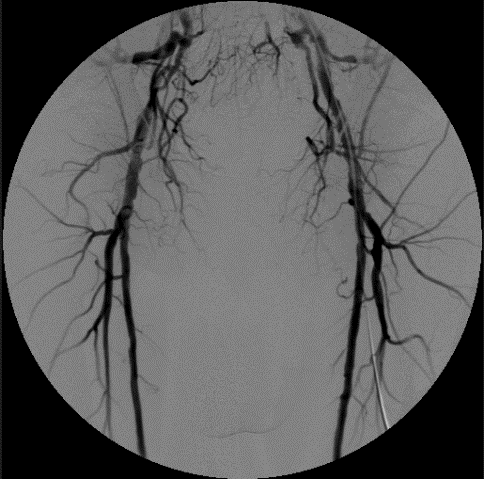
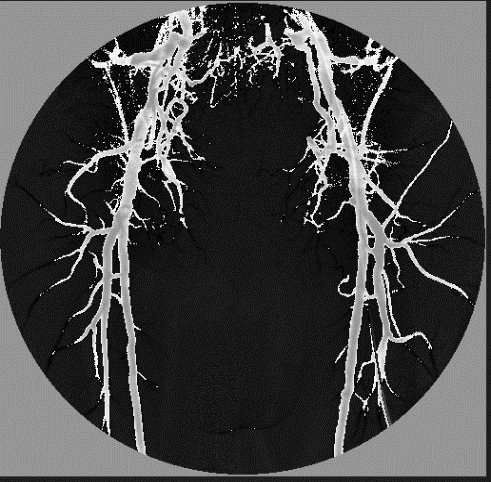
#### Addition des images

* Addition d’une valeur scalaire à une image :

La class **AdditionScalaire** est une class Template qui hérite de la class **Processing1**, ce qui signifié quelle prends une seule image en entrée. Elle permet d’ajouter une valeur scalaire a chaque pixel de cette image.  
Elle contient un constructeur pour initialiser l’image, la valeur scalaire (**valScalaire**) et **boolean** **inPlace** qui permet d’indiquer si l’opération modifie l’image originale, elle contient aussi la méthode **Process** effectue le traitement principal en surchargeant la méthode virtuelle de la class mère. Elle dispose également d’une méthode statique **additionScalair**, qui permet de faire le traitement sans qui créer des instances de la class.



**Figure 21: Aperçu de code de la classe Addition avec un scalaire**



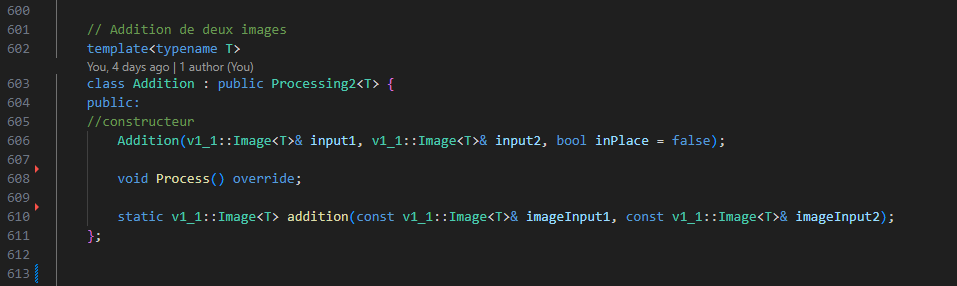
**Figure 22: Addition d'une image après l'addition avec un scalaire**

**Figure 23: Aperçu d'une image avant l'addition avec un scalaire**

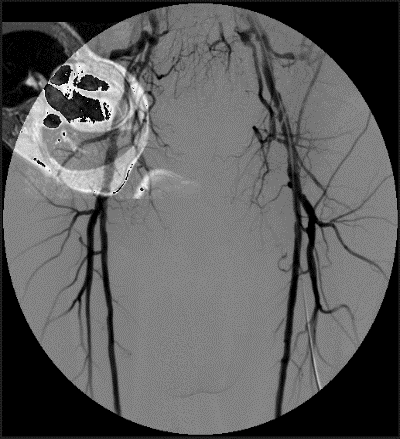
* Addition de deux images

La class **Addiition** est aussi une class **template** qui hérite de la class **Processing2**, indiquant qu’elle prend deux images en entrée. Son constructeur prend deux image **input1** et **input2** ainsi qu’un **boolean inPlace** qui indique si le résultat doit écraser la première image.  
L’addition de deux images se fait comme suit :

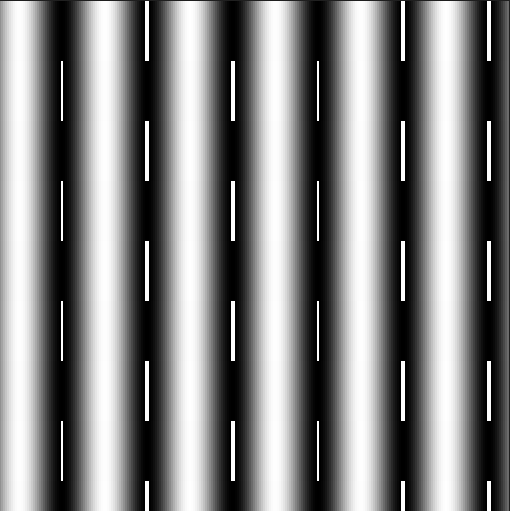
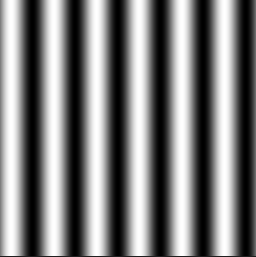
* + Si les deux images ont les mêmes dimensions, l’addition se fait pixel par pixel, pour avoir une image du même format en sortie.
  + Si leurs dimensions sont différentes, l’image la plus petite est d’abord étendue par l’ajout des zéros (padding), pour correspondre à la taille de l’image la plus grande avant d’effectue l’addition.



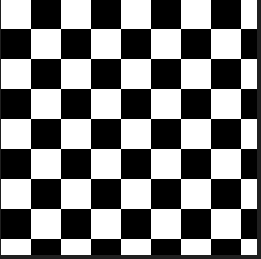
**Figure 24: Aperçu de code de la classe addition de deux images**

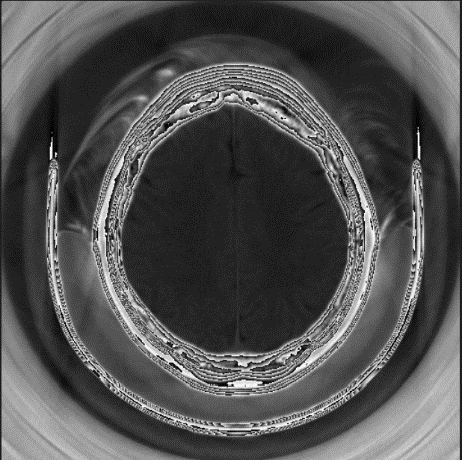
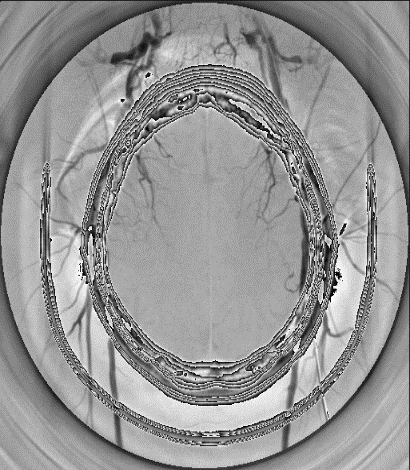


**Figure 25: Résultat de l'addition de deux images de différentes dimensions**



**Figure 26: Résultat d'addition de deux images damier et sinusoïdal de la même dimension**





**Figure 27: Résultat d'addition de deux images de même dimension**

#### Égalisation d'histogramme

L’égalisation d’histogramme est une technique de traitement d’image utilisée pour améliorer le contraste des images. Elle consiste à repartir les niveaux de gris d’une image de manière uniforme, pour que tous les niveaux de luminosité soient mieux représentés.

Il est calculé en trois étape principale :

* **Calcule de l’histogramme :**

Un histogramme d’image est une représentation graphique qui montre la répartition des pixels selon leur intensité lumineuse (de 0 à 255). Par exemple :

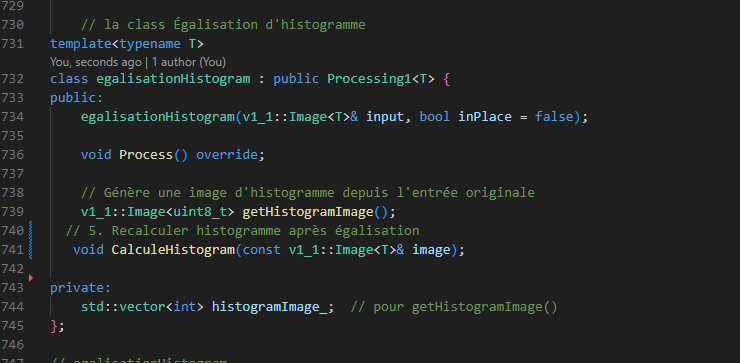
* + Une image sombre aura un histogramme concentré à gauche.
  + Une image très claire aura un histogramme concentré à droite
* **Calcule de la fonction de distribution cumulative**

La CDF indique pour chaque niveau d’intensité (de 0 à 255), combien de pixels dans l’image où les valeurs inferieurs ou égale a ce niveau

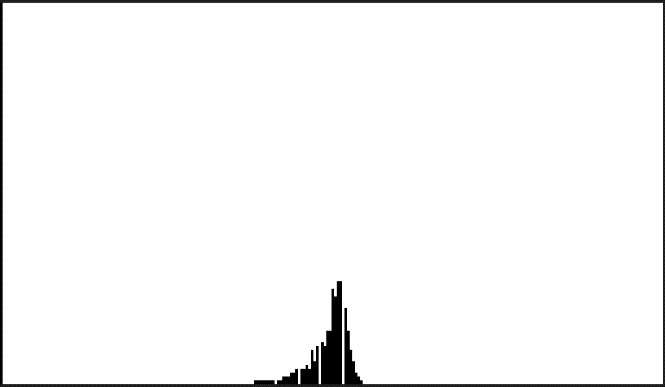
* **Transformation des niveaux de gris (via LUT)**

Consiste à créer une table LUT pour remplacer les anciennes valeurs de pixels par les nouvelles valeurs calculés

L’image suivante montre le code de la class d’égalisation d’histogramme

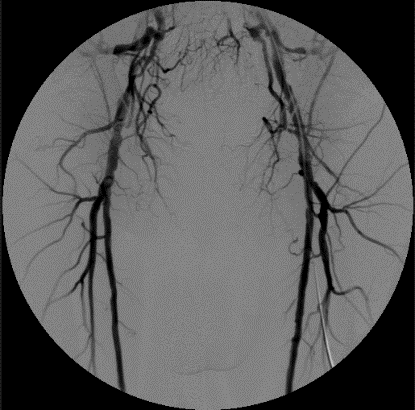
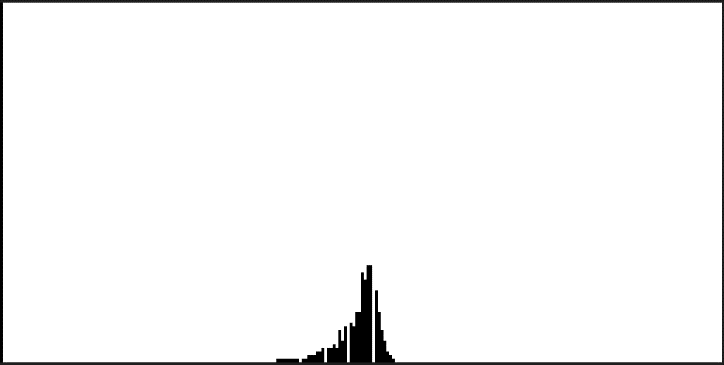
****

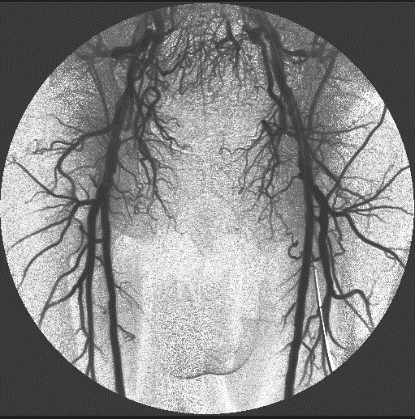
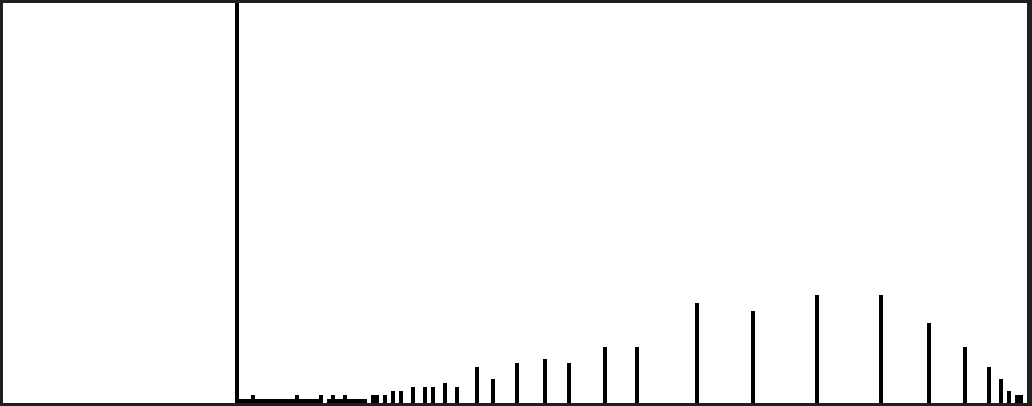
**Figure 28 : Aperçu de code de la classe égalisation d’histogramme**



**Figure 29 Image avant l’égalisation d’histogramme**

**Figure 30 Histogramme de l'image avant l'égalisation**

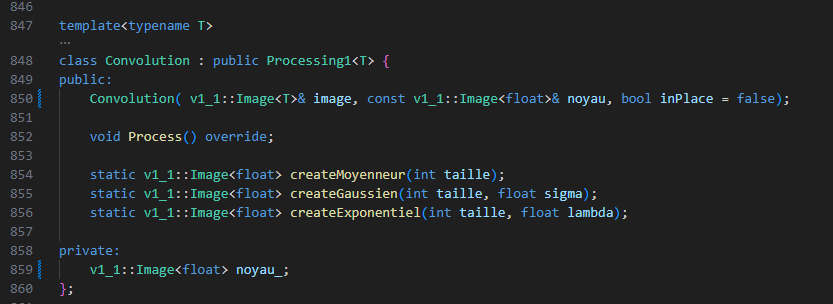




**Figure 32: Image après l’égalisation d'histogramme**

**Figure 31: Histogramme de l’image après l’égalisation**

#### Filtrage par convolution



**Figure 33: Aperçu de la class Convolution**

## Conclusion et perspectives :

## Bibliographie