**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 Introduction 5](#_Toc199857748)

[2 Définition d’une image 5](#_Toc199857749)

[2.1 Codage d’une image (image numérique) 5](#_Toc199857750)

[3 Présentation de Projet 6](#_Toc199857751)

[4 Namespace version 1.0 6](#_Toc199857752)

[4.1 Définition : 6](#_Toc199857753)

[4.2 Implémentation des fonctions 6](#_Toc199857754)

[4.2.1 Allocation d’images 6](#_Toc199857755)

[4.2.2 Création d'images particulières 7](#_Toc199857756)

[4.2.2.1 Image Blanche 7](#_Toc199857757)

[4.2.2.2 Image Damier 7](#_Toc199857758)

[4.2.2.3 Image Sinusoïdale 8](#_Toc199857759)

[4.2.3 Lecture et écriture des fichiers images au format brut (.RAW) : 9](#_Toc199857760)

[4.2.3.1 Fichier Raw : 9](#_Toc199857761)

[4.2.3.2 Lecteur de fichiers images au format brut 9](#_Toc199857762)

[4.2.4 Ecriture de fichiers images au format brut 10](#_Toc199857763)

[4.2.5 Conversion d’image 10](#_Toc199857764)

[4.2.5.1 Conversion d’image d’un type à un autre 10](#_Toc199857765)

[4.2.5.2 Conversion d’image de RGB vers niveau de gris 11](#_Toc199857766)

[4.2.6 Conversion d'une image en fausses couleurs à l'aide d'une LUT 11](#_Toc199857767)

[4.2.6.1 Fichier LUT 11](#_Toc199857768)

[4.2.6.2 Chargement de fichier LUT (le filtre LUT) 12](#_Toc199857769)

[4.2.6.3 Application de fausse couleur à l’aide de LUT 12](#_Toc199857770)

[5 Tester des filtres LUT sur les différentes images 13](#_Toc199857771)

[6 Namespace version 1.1 14](#_Toc199857772)

[6.1 Introduction 14](#_Toc199857773)

[6.2 Implémentation des fonctionnalités de namespace v1.0 dans le cadre d’une class 14](#_Toc199857774)

[7 Namespace version 2.0 16](#_Toc199857775)

[7.1 Implémentation des traitements d’images 16](#_Toc199857776)

[7.1.1 Processing 1&2 16](#_Toc199857777)

[7.1.2 Addition des images 17](#_Toc199857778)

[7.1.2.1 Addition d’une valeur scalaire à une image 17](#_Toc199857779)

[7.1.2.2 Addition de deux images 18](#_Toc199857780)

[7.1.3 Égalisation d'histogramme 20](#_Toc199857781)

[7.1.4 Filtrage par convolution 21](#_Toc199857782)

[7.1.4.1 Définition 21](#_Toc199857783)

[7.1.4.2 Types des filtres (noyaux) 22](#_Toc199857784)

[8 Conclusion et perspectives : 23](#_Toc199857785)

[9 Bibliographie 25](#_Toc199857786)

Liste des figures

[Figure 1: Représentation d’image numérique. 6](file:///C:\Users\lylehara\Downloads\TER\Travail%20de%20Recherche%20Encadré.docx#_Toc199858015)

[Figure 2: Fonction d’allocation d’image. 7](#_Toc199858016)

[Figure 3: Création d’image blanche. 7](file:///C:\Users\lylehara\Downloads\TER\Travail%20de%20Recherche%20Encadré.docx#_Toc199858017)

[Figure 4: Création d’image damier. 8](#_Toc199858018)

[Figure 5: Création d’image sinusoïdale. 8](#_Toc199858019)

[Figure 6: Aperçu d’une image sinusoïdal 9](file:///C:\Users\lylehara\Downloads\TER\Travail%20de%20Recherche%20Encadré.docx#_Toc199858020)

[Figure 7: Aperçu d’une image damier 9](file:///C:\Users\lylehara\Downloads\TER\Travail%20de%20Recherche%20Encadré.docx#_Toc199858021)

[Figure 8: Aperçu d’une image Blanche 9](file:///C:\Users\lylehara\Downloads\TER\Travail%20de%20Recherche%20Encadré.docx#_Toc199858022)

[Figure 9: Lecteur d’un fichier brut 10](#_Toc199858023)

[Figure 10: Ecriture d’un fichier brut 10](#_Toc199858024)

[Figure 11: Conversion d’une image d’un type à un autre. 11](#_Toc199858025)

[Figure 12: Conversion d’une image RGB en niveau de gris 11](#_Toc199858026)

[Figure 13: Chargement de fichier LUT. 12](file:///C:\Users\lylehara\Downloads\TER\Travail%20de%20Recherche%20Encadré.docx#_Toc199858027)

[Figure 14: Application de LUT sur l’image. 12](#_Toc199858028)

[Figure 15: Conversion d’un pixel gris en RGB via une LUT 13](file:///C:\Users\lylehara\Downloads\TER\Travail%20de%20Recherche%20Encadré.docx#_Toc199858029)

[Figure 16: Résultat de l’application de LUT sur les différentes images 14](#_Toc199858030)

[Figure 17: Aperçu du code de la class Image 15](#_Toc199858031)

[Figure 18: Aperçu de code de la classe ImageRGB 16](#_Toc199858032)

[Figure 19: Aperçu de code de la classe Processing 1 16](#_Toc199858033)

[Figure 20: Aperçu de code de la classe Processing 2 17](#_Toc199858034)

[Figure 21: Aperçu de code de la classe Addition avec un scalaire 17](#_Toc199858035)

[Figure 22: Addition d'une image après l'addition avec un scalaire 18](file:///C:\Users\lylehara\Downloads\TER\Travail%20de%20Recherche%20Encadré.docx#_Toc199858036)

[Figure 23: Aperçu d'une image avant l'addition avec un scalaire 18](file:///C:\Users\lylehara\Downloads\TER\Travail%20de%20Recherche%20Encadré.docx#_Toc199858037)

[Figure 24: Aperçu de code de la classe addition de deux images 18](#_Toc199858038)

[Figure 26: Résultat d'addition de deux images damier et sinusoïdal de la même dimension 19](file:///C:\Users\lylehara\Downloads\TER\Travail%20de%20Recherche%20Encadré.docx#_Toc199858039)

[Figure 25: Résultat de l'addition de deux images de différentes dimensions 19](file:///C:\Users\lylehara\Downloads\TER\Travail%20de%20Recherche%20Encadré.docx#_Toc199858040)

[Figure 27: Résultat d'addition de deux images de même dimension 19](file:///C:\Users\lylehara\Downloads\TER\Travail%20de%20Recherche%20Encadré.docx#_Toc199858041)

[Figure 28 : Aperçu de code de la classe égalisation d’histogramme 20](file:///C:\Users\lylehara\Downloads\TER\Travail%20de%20Recherche%20Encadré.docx#_Toc199858042)

[Figure 29 Histogramme de l'image avant l'égalisation 21](file:///C:\Users\lylehara\Downloads\TER\Travail%20de%20Recherche%20Encadré.docx#_Toc199858043)

[Figure 30 Image avant l’égalisation d’histogramme 21](file:///C:\Users\lylehara\Downloads\TER\Travail%20de%20Recherche%20Encadré.docx#_Toc199858044)

[Figure 31: Histogramme de l’image après l’égalisation 21](file:///C:\Users\lylehara\Downloads\TER\Travail%20de%20Recherche%20Encadré.docx#_Toc199858045)

[Figure 32: Image après l’égalisation d'histogramme 21](file:///C:\Users\lylehara\Downloads\TER\Travail%20de%20Recherche%20Encadré.docx#_Toc199858046)

[Figure 33: image explicative sur la convolution 22](file:///C:\Users\lylehara\Downloads\TER\Travail%20de%20Recherche%20Encadré.docx#_Toc199858047)

[Figure 34: Aperçu de la class Convolution 23](#_Toc199858048)

# 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.

# 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.**

# 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.

# Namespace version 1.0

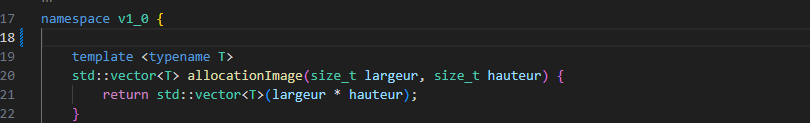
## Définition :

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.**

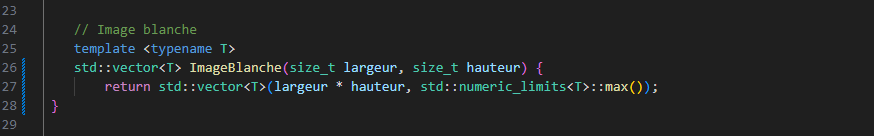
### Création d'images particulières

#### 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 :

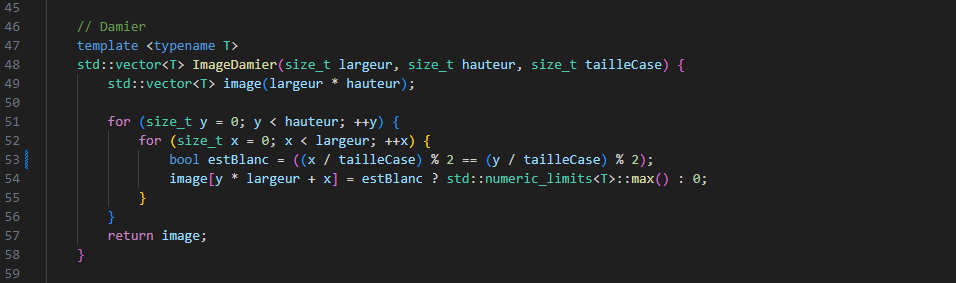
* Si T = **uint8\_t**, alors max = 255
* Si T = **uint16\_t**, alors max = 65535

****

**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

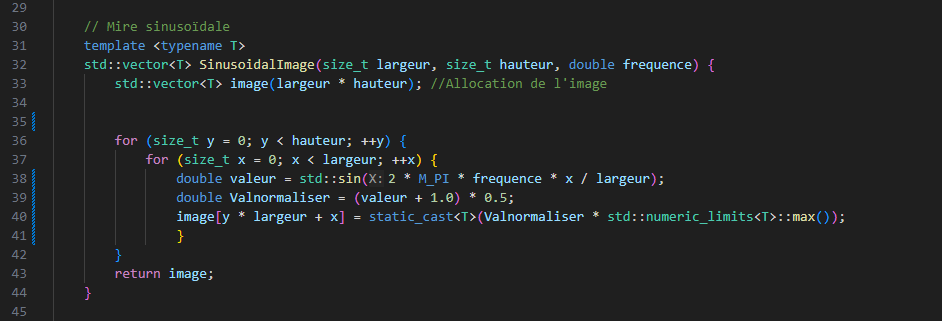
La fonction **SinusoïdaleImage** génère une image dont l’intensité des pixels varie selon une onde sinusoïdale. Elle prend en paramètre la largeur, la hauteur et la fréquence de l’onde qui représente le nombre de période de la fonction sinus.

* 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;**

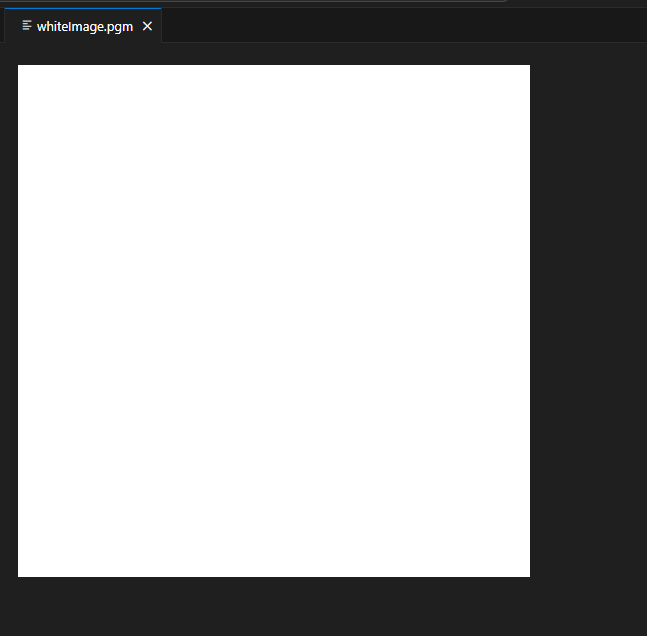
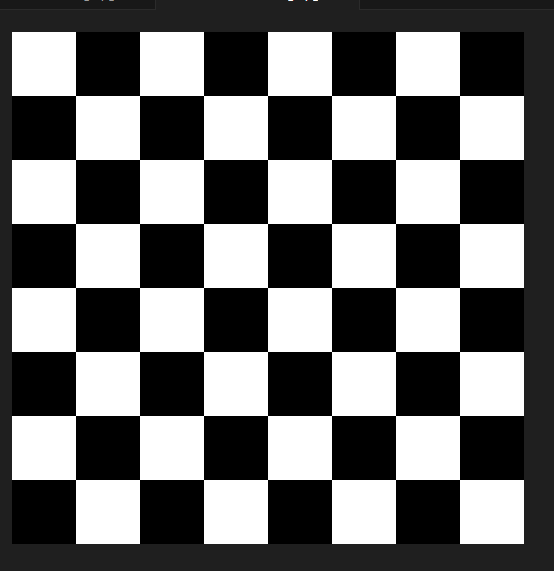
Puis la convertit en une intensité adaptée au type T (par exemple entre 0 et 255). L’image ainsi remplie est ensuite retournée.

****

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

* **Les Resultats obtenus**

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



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

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

**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.

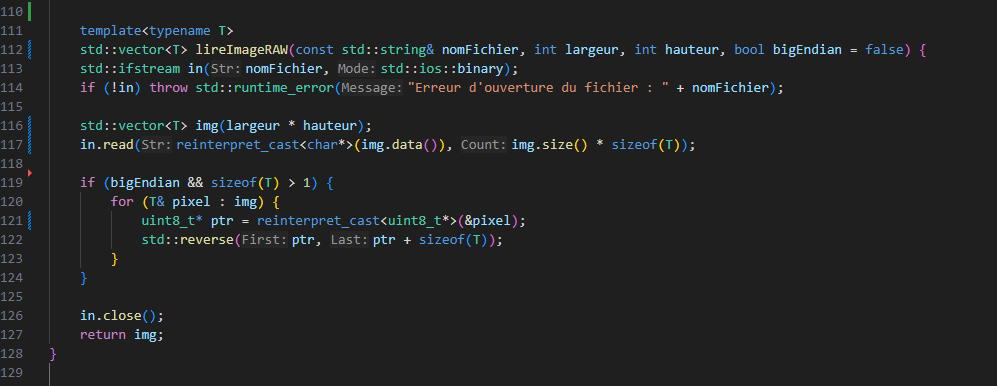
**Big** et **little endian** sont deux manières différentes de stocker les **octets** d’une valeur multi-octets (par exemple un entier sur 2 ou 4 octets) dans un fichier ou une mémoire.

* **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

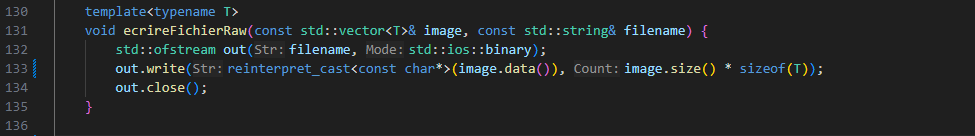
Ensuite, Elle lit les données pixel par pixel, en convertissant correctement les octets selon l’ordre spécifié, puis sauvegarde les valeurs dans un vecteur. Si le fichier ne s’ouvre pas, elle renvoie une image vide.



**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).

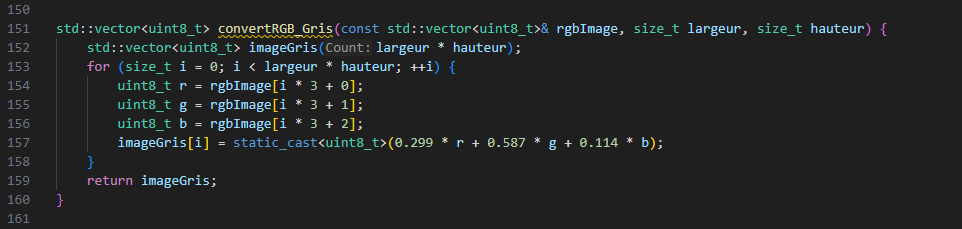


**Figure 11: Conversion d’une image d’un type à un autre.**

#### 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**

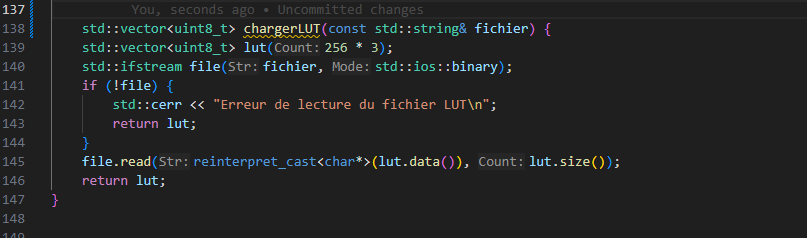
### Conversion d'une image en fausses couleurs à l'aide d'une LUT

#### 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)

La fonction **chargerLUT** permet de lire un fichier binaire contenant une table correspondance (Look-Up Table) et retourne un vecteur de uint8\_t en sortie

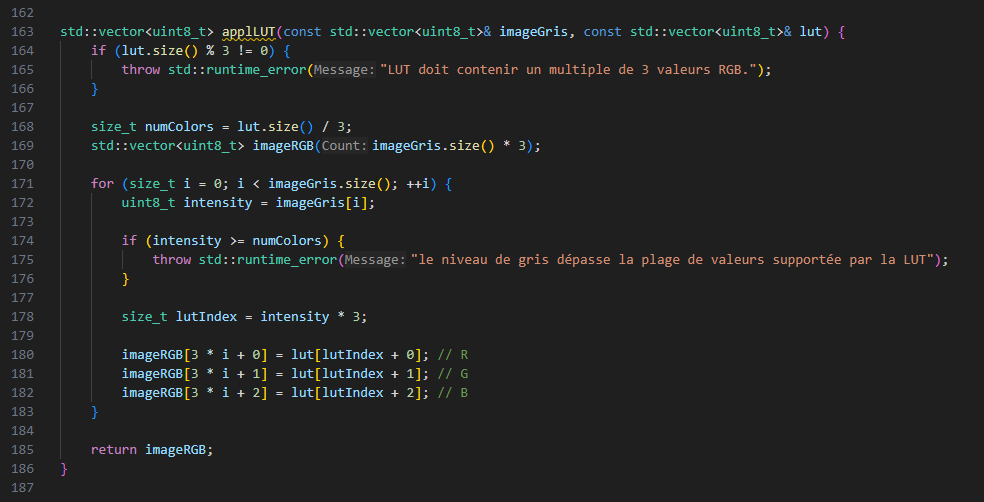


**Figure 13: Chargement de fichier LUT.**

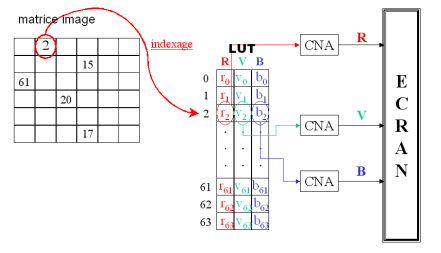
#### Application de fausse couleur à l’aide de LUT

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.

****

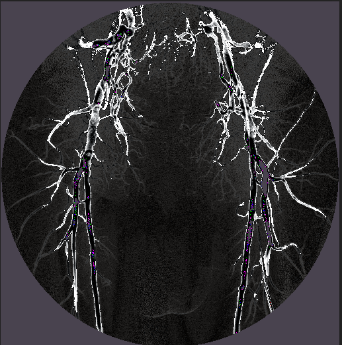
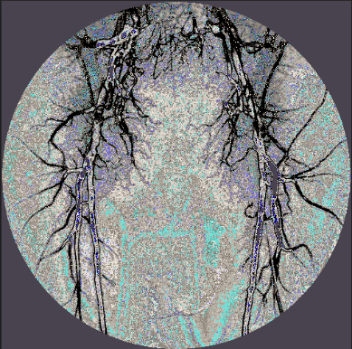
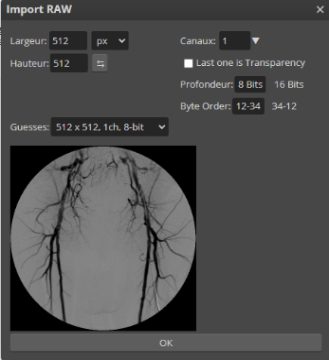
**Figure 14: Application de LUT sur l’image.**

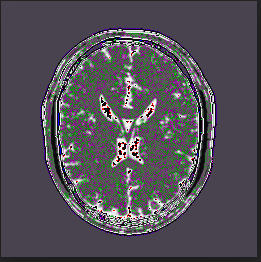
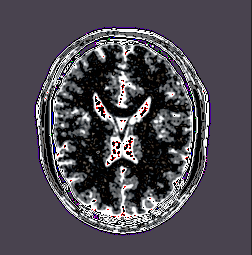
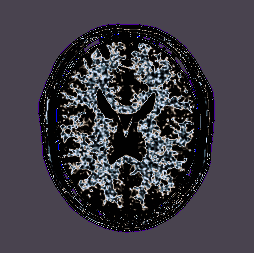
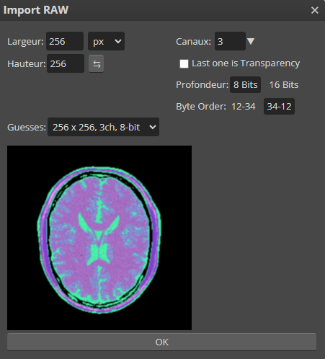


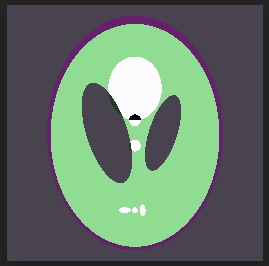
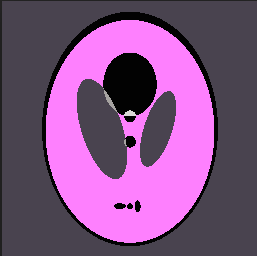
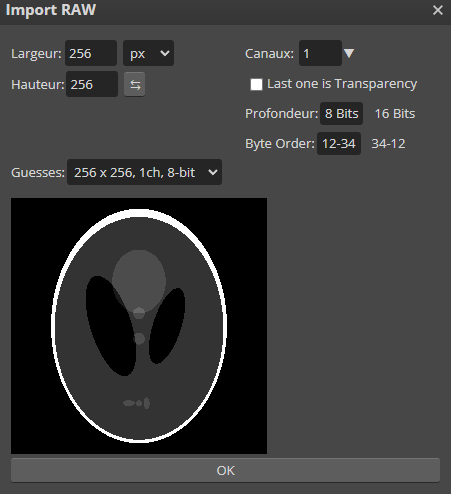
**Figure 15: Conversion d’un pixel gris en RGB via une LUT**

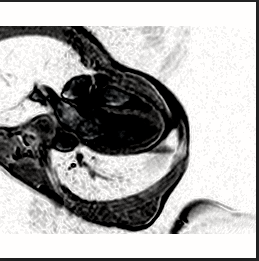
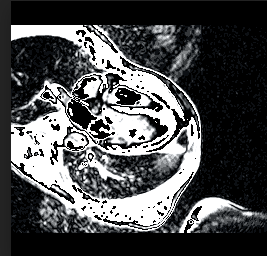
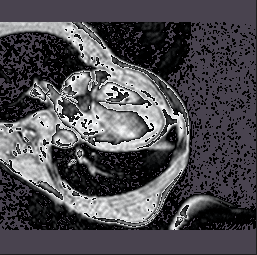
# Tester des filtres LUT sur les différentes images

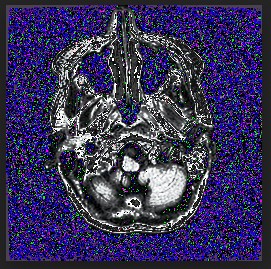
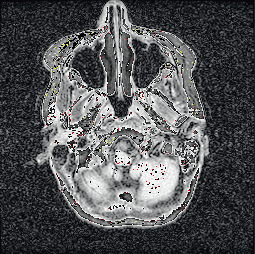
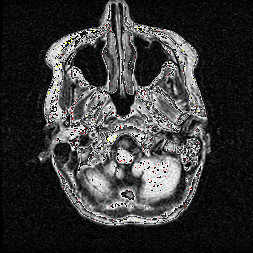
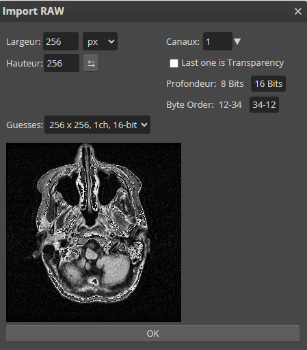
Les figures suivantes montrent le résultat obtenu après l’appel aux fonctions précédentes











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

# Namespace version 1.1

## Introduction

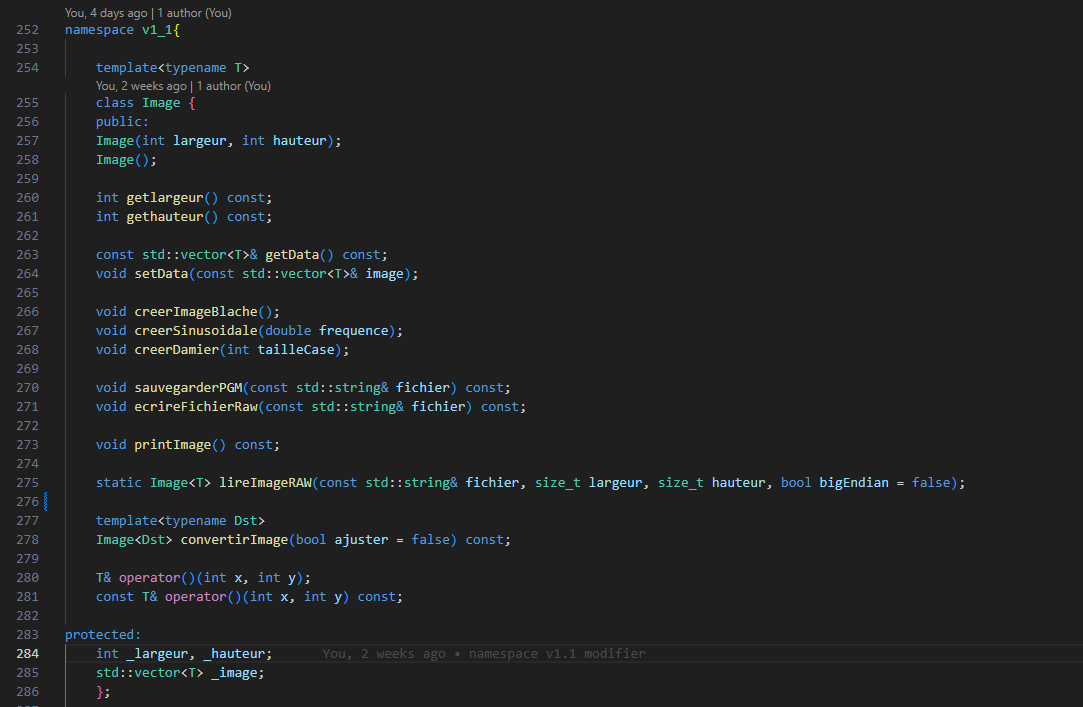
L’objectif du namespace v1\_1 est de reprendre les mêmes fonctionnalités de v1\_0 en adoptant une approche orientée objet. On utilise une classe générique pour coder des images en utilisant ***std::vector*,** ce qui rend la gestion plus claire et structurée.

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)***.

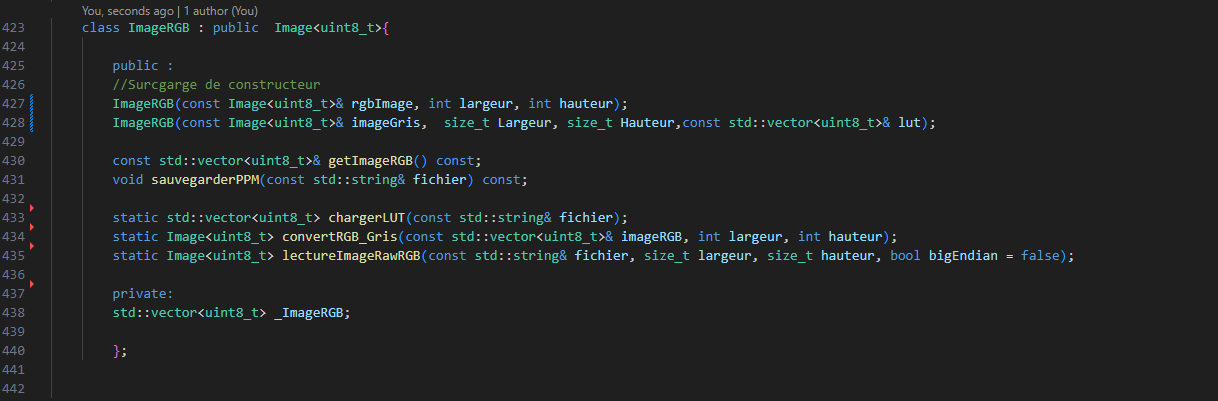


**Figure 17: Aperçu du code de la class Image**

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**

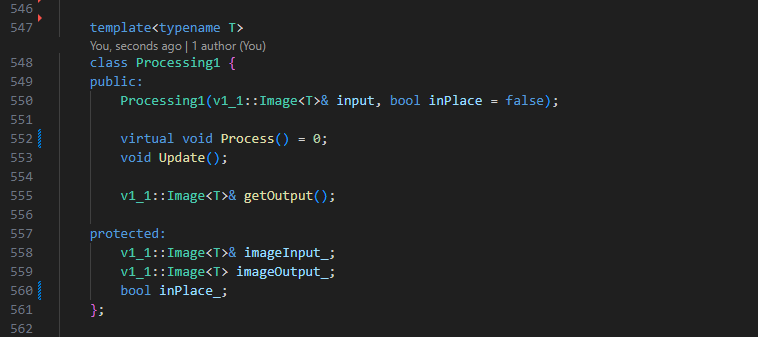
# Namespace version 2.0

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.

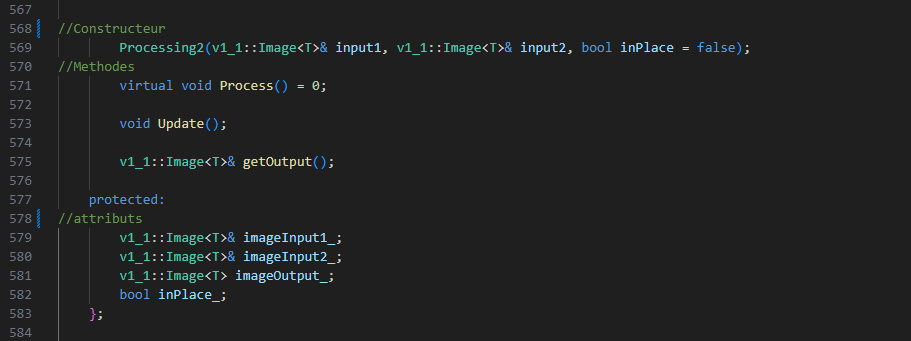
## Implémentation des traitements d’images

### 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**

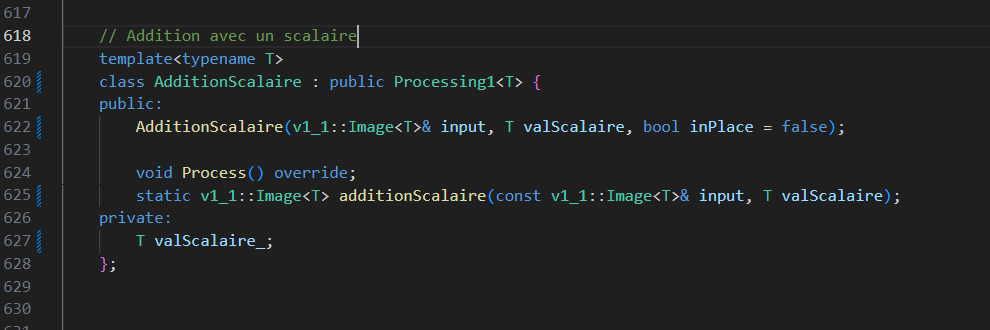


**Figure 20: Aperçu de code de la classe Processing 2**

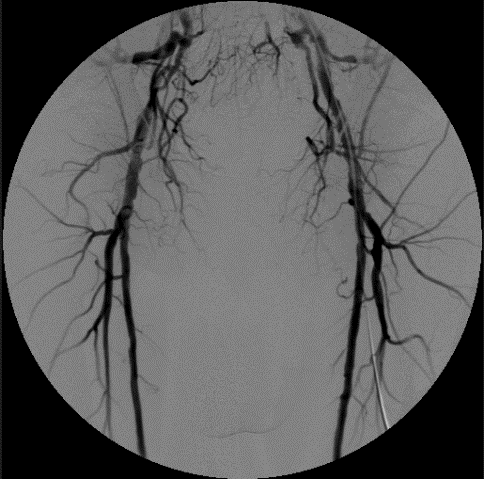
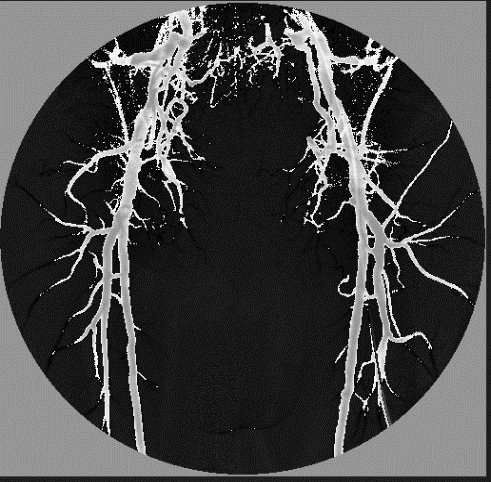
### Addition des images

#### 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**



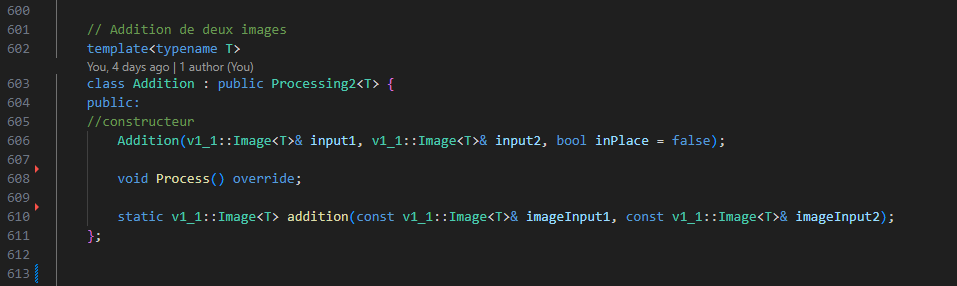
**Figure 22: Addition d'une image après l'addition avec un scalaire**

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

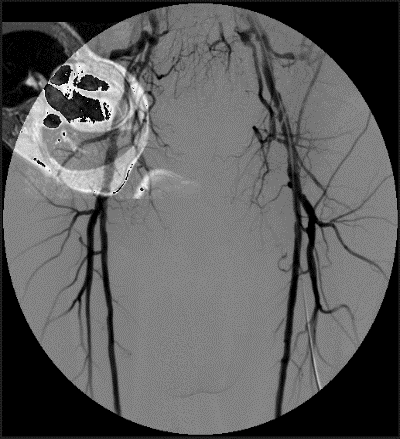
#### Addition de deux images

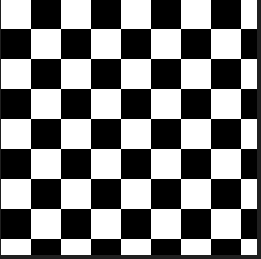
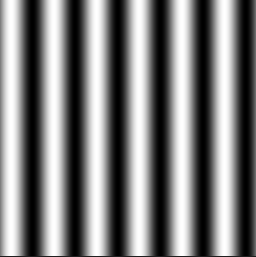
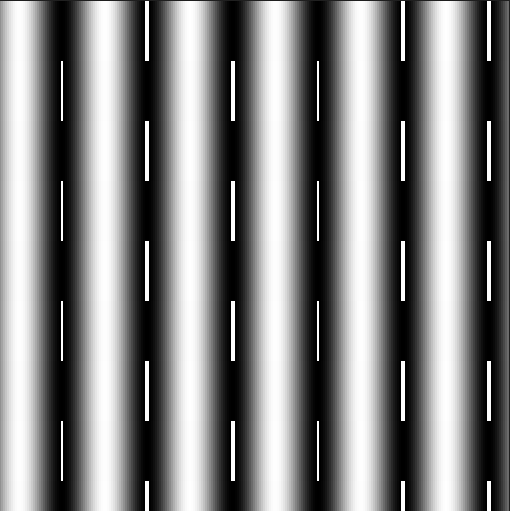
La class **Addiition** est aussi une class **template** qui hérite de la class **Processing2**, indiquant qu’elle prend deux images en entrée. Son constructeur prend deux image **input1** et **input2** ainsi qu’un **boolean inPlace** qui indique si le résultat doit écraser la première image.  
L’addition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



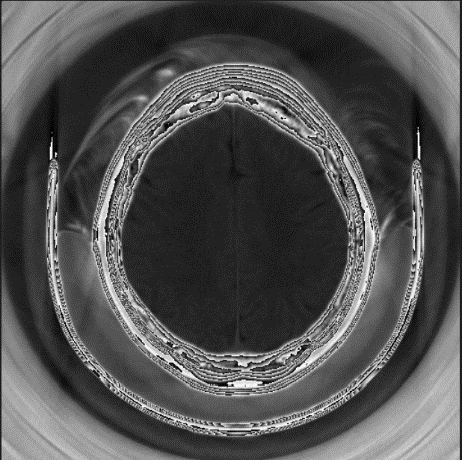
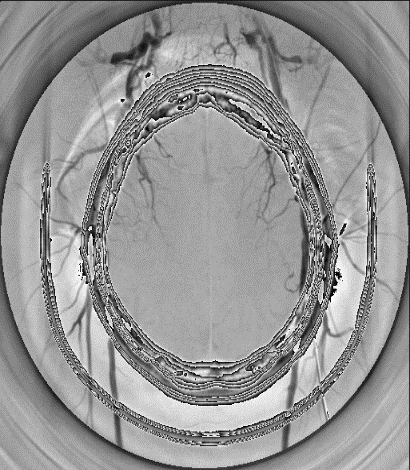
**Figure 24: Aperçu de code de la classe addition de deux images**





**Figure 25: Résultat d'addition de deux images damier et sinusoïdal de la même dimension**

**Figure 26: Résultat de l'addition de deux images de différentes dimensions**



**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 :

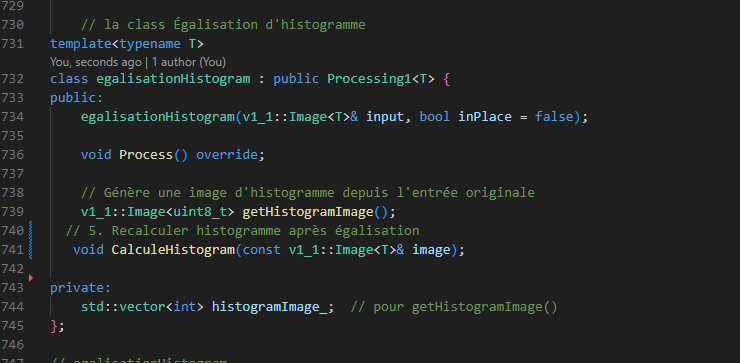
* + Une image sombre aura un histogramme concentré à gauche.
  + Une image très claire aura un histogramme concentré à droite
* **Calcule de la fonction de distribution cumulative**

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

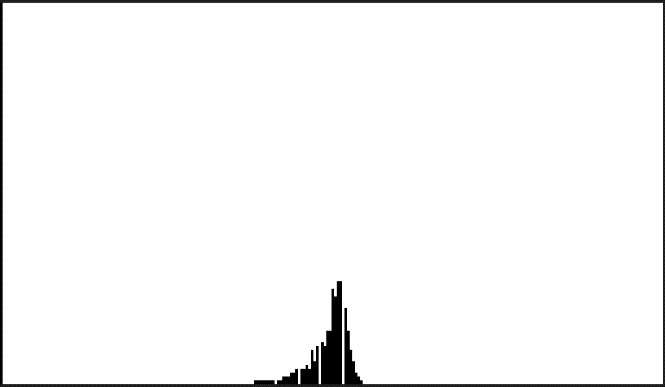
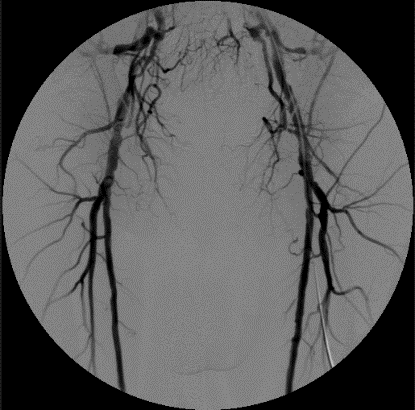
* **Transformation des niveaux de gris (via LUT)**

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

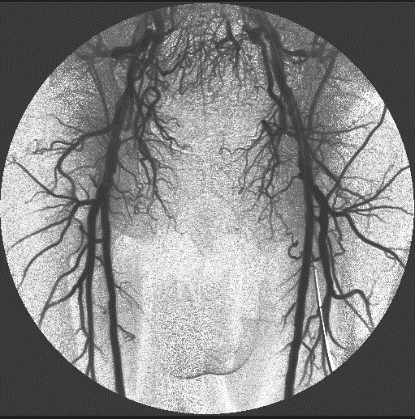
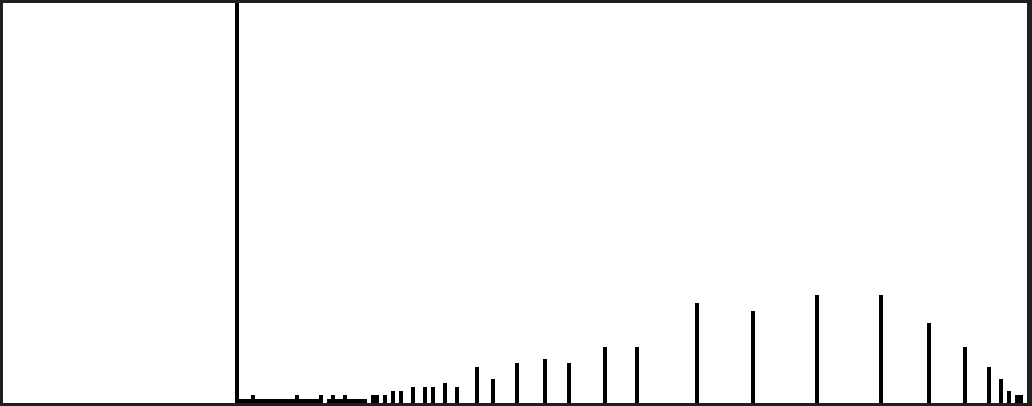
****

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



**Figure 29 Histogramme de l'image avant l'égalisation**

**Figure 30 Image avant l’égalisation d’histogramme**



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

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

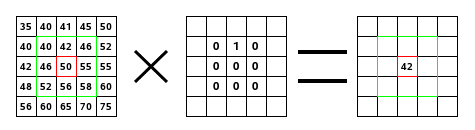
### Filtrage par convolution

#### Définition

La **convolution** est une technique utilisée en traitementd’images. Elle consiste en une opération de **multiplication de deux matrices** de tailles différentes (généralement une petite et une grande), mais de même dimensionnalité semblable (par exemple 1D, 2D), produisant une **nouvelle matrice** (également de même dimensionnalité).

La convolution est donc le **traitement d’une matrice** (une image) par une autre petite matrice appelée **matrice de convolution**, le filtre ou noyau (kernel).

Le **filtre parcourt toute la matrice de l’image** de manière incrémentale et génère une nouvelle matrice constituée des résultats de la multiplication.

 Le **filtre étudie successivement chacun des pixels** de l'image. Pour chaque pixel, que nous appellerons **« pixel initial »,** il **multiplie la valeur de ce pixel et de chacun des 8 pixels qui l'entourent** par la valeur correspondante dans le noyau. Il **additionne l'ensemble des résultats** et le pixel initial prend alors la valeur du résultat final.  
Vite un exemple, simple :  
À gauche se trouve la **matrice de l'image H** : chaque pixel est indiqué par sa valeur. Le **pixel initial** est encadré de rouge. La **zone d'action du noyau** est encadrée de vert. Au centre, se trouve le **noyau** et, à droite, le **résultat de la convolution**.  
Voici ce qui s'est passé : le filtre a **lu successivement**, de gauche à droite et de haut en bas, les pixels de la zone d'action du noyau et il a **multiplié chacun d'eux par la valeur correspondante du noyau X**, puis **additionné les résultats**. Le pixel initial a pris la valeur **42**.

**Figure 33: image explicative sur la convolution**

Sous la forme suivante :

= (40\*0) + (42\*1) + (46\*0) + (46\*0) + (50\*0) + (55\*0) + (52\*0) + (56\*0) + (58\*0) = **42**.

#### Types des filtres (noyaux)

En traitement d’image plusieurs types de filtres (noyaux) peuvent utilisés selon le cas recherché.

Voici quelques filtres les plus courant :

* **Filtre Moyenneur :**

Est utilisé pour réduire le bruit et obtenir un lissage uniforme de l’image, mais il peut flouter les contours.

Il remplace chaque pixel de l’image par la moyenne de ces voisins. Par exemple si on prend un noyau de 3x3 contient des coefficients égaux à 1/9. Ça forme est :

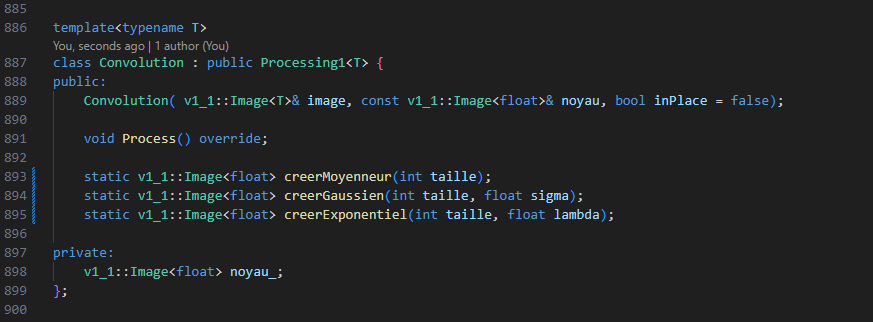
* **Filtre Gaussien :**

Ce filtre est appliqué lorsqu’on souhaite un flou plus naturel tout en préservant les bords. Il repose sur la distribution gaussienne, il applique des poids plus élevés au **pixel central** et plus faibles aux pixels plus éloignés. La matrice de son noyau est représentée par :

* **Filtre exponentiel :**

Le filtre exponentiel dit aussi filtre de lissage, est principalement utilisé pour rajouter de flou dans image et pour réduire certains types de bruit. La matrice de son noyau est représentée par :

L’image suivante montre le code de la classe Convolution qui hérite de la classe **Processing1**. Elle contient un constructeur prenant en paramètre une image d’entrée, un noyau de convolution ainsi qu’un booléen optionnel **inPlace** pour déterminer si le traitement modifie directement l’image originale, elle contient aussi une méthode **Process()** qui permet d’appliquer l’opération de convolution. La classe propose également trois méthodes statiques : **creerMoyenneur()**, **creerGaussien()** et **creerExponentiel()**, qui permettent respectivement de générer des noyaux de convolution de type moyenneur, gaussien et exponentiel



**Figure 34: Aperçu de la class Convolution**

# Conclusion

Le traitement d’image joue un rôle fondamental dans plusieurs domaines notamment la vision ordinateur, robotique et l’intelligence artificielle. Il sert à améliorer la qualité des images pour facilité l’interprétation dans les traitements plus complexe comme la classification ou l’apprentissage automatique.

Le travail réalisé dans le cadre de ce rapport s’inscrit sur le développement d’une bibliothèque de traitement d’images en C++ structuré en trois version successives (namespace v1.0, namespace v1.1, et namespace v2.0).

Namespace v1.0 est une sous bibliothèque qui regroupe les fonctionnalités de base pour le traitement d’images. Elle inclut les fonctions pour allouer dynamiquement des images, la création des image blanche, damier et sinusoïdal, la lecteur et l’écriture des fichier images en format brut (. RAW) selon leur type de codage **big** ou **little endian**, ainsi que la conversion entre types d’images. Elle contient également des fonctions qui permet de transformer des images en couleur **RGB** en niveau de gris (noir et blanc) en utilisant une table correspondance (**LUT**).

# Bibliographie

1. <https://wp.unil.ch/risk/files/2015/12/8.-Filtres-et-convolution.pdf>
2. <https://openclassrooms.com/fr/courses/4470531-classez-et-segmentez-des-donnees-visuelles/5026661-filtrez-une-image>
3. <https://perso.esiee.fr/~perretb/I5FM/TAI/convolution/index.html>
4. <https://perso.ensta-paris.fr/~manzaner/Cours/Poly/Poly_Chap2_Filtrage.pdf>