

POLYTECH NICE SOPHIA

Projet C++ : Semestre S7

ALOUI Ghaieth

KSIKS Marwan

AOURARH Yassine

Octobre 2024

Table des matières

I - Positionnement du problème	3
II - Projet 1 : Modélisation d'un comportement addictif	3
1 - Présentation du modèle mathématique	3
1.1 - Modèle pour une Personne	3
1.2 - Modèle pour deux Personnes	4
2 - Présentation du programme C++	4
2.1 - Diagramme de Classes UML	5
2.2 - Explication du Code implémenté	5
III - Projet 2 : Techniques de Compression d'images	7
1 Présentation du programme C++	7
1.1 Diagramme de Classes UML	7
2 Relations entre les classes et fonctionnement du programme	8
2.1 Classe CompresseurAbstrait	8
2.2 Classe CompressionImage	9
2.3 Classe CompresseurFactory	10
2.4 Classe ProcesseurImage	10
2.5 Classe GestionnaireImage	10
2.6 Usage de OpenCV dans le programme	11
2.7 Usage dans main.cpp	11
3 Analyse des performances du code	11
3.1 Précision supérieure en C++	12
3.2 Compromis entre performance et temps d'exécution	12
3.3 Résumé des avantages	12
IV Conclusion	12
V - Annexe : Mode d'usage et installation du Programme	13
1 Prérequis	13
2 Présentation de l'arborescence des projets	13
3 Installation des dépendances	13
4 Compilation et exécution des projets	13
5 Remarques finales	14

I - Positionnement du problème

Ce projet, s'inscrivant dans une logique de programmation orientée objet, impose une approche différente de celle utilisée en Python. En effet, la programmation orientée objet en C++ repose sur des concepts tels que l'encapsulation, l'héritage, et le polymorphisme, qui diffèrent dans leur implémentation par rapport à Python. De plus, C++ exige une gestion explicite de la mémoire et impose une structuration plus rigide des classes, et objets, ce qui nécessite une adaptation des méthodes et des stratégies de conception utilisées en Python pour tirer parti des spécificités et performances offertes par le langage C++. Dans la suite, nous revisiterons donc deux projets réalisés en Python pendant la MAM3, en les adaptant à une implémentation en C++. Nous le raccourcirons si le rapport est trop long.

II - Projet 1 : Modélisation d'un comportement addictif

1 - Présentation du modèle mathématique

Dans cette section, nous présentons un modèle mathématique décrivant les dynamiques d'addiction. L'objectif de ce modèle est de comprendre l'évolution des comportements addictifs au fil du temps et d'analyser les interactions mutuelles entre deux individus. Nous commencerons par décrire le modèle pour une seule personne, où l'on modélise l'intensité de l'addiction en fonction de plusieurs paramètres. Ensuite, nous étendrons le modèle à deux personnes, en tenant compte des effets d'influence et d'interactions mutuelles.

1.1 - Modèle pour une Personne

Le modèle pour une personne décrit l'évolution de plusieurs variables clés associées à l'addiction au fil du temps.

Les équations qui décrivent cette évolution sont données par :

$$\begin{aligned} C(t+1) &= (1-d)C(t) + b \min\{1, 1-C(t)\}A(t) & C(0) &= C_0 \\ S(t+1) &= S(t) + p \max\{0, S_{\max} - S(t)\} - hC(t) - kA(t) & S(0) &= S_0 \\ E(t+1) &= E(t) - m_E E(t) & E(0) &= E_0 \\ A(t) &= V(t) + R(\lambda(t)) \times q(1 - V(t)), & R(\cdot) &\sim \text{Poisson}(\lambda) \\ \lambda(t+1) &= \lambda(t) + m_\lambda, & \lambda(0) &= \lambda_0 \end{aligned}$$

Lois de comportement :

- $\psi = C - S - E$
- $V = \min\{1; \max\{\psi, 0\}\}$
- $\psi = \psi(C, S, E)$ en général
- Addiction $\iff V \approx 1$

où :

- $C(t)$ représente l'intensité de fringale ou de désir à l'instant t .
- $S(t)$ est l'intensité de self-control, indiquant la capacité de contrôle de soi.
- $E(t)$ représente l'influence sociétale (telle que la famille, les amis ou les normes légales) à un moment donné.

- $A(t)$ représente le niveau d'addiction exercé, correspondant au passage à l'acte.
- $d, b, p, h, k, m_E, m_\lambda$ sont des paramètres influençant la dynamique des différentes variables.
- $V(t)$ est une fonction dépendant de la différence entre fringale, self-control et influence sociétale, représentant le niveau de vulnérabilité.
- $R(X(t))$ suit une distribution de Poisson, modélisant des événements aléatoires influençant l'intensité de l'addiction.
- $\lambda(t)$ représente le taux de Poisson utilisé pour générer ces événements.

1.2 - Modèle pour deux Personnes

Le modèle pour deux personnes est une extension naturelle du modèle pour une personne, avec l'introduction d'interactions entre deux individus. Les dynamiques internes de désir ($C(t)$), de self-control ($S(t)$) et d'addiction ($A(t)$) restent similaires à celles du modèle pour une personne. Toutefois, nous ajoutons des termes d'influence réciproque entre les deux individus, notés i et j . Ces interactions sont modélisées à travers les coefficients α , β , et γ , qui traduisent l'influence que l'individu j exerce sur l'individu i , et vice-versa.

Les équations de la dynamique pour deux personnes sont les suivantes :

$$\begin{aligned} C_i(t+1) &= (1-d)C_i(t) + \alpha A_j(t)C_i(t) + b \min\{1, 1-C_i(t)\}A_i(t) \\ S_i(t+1) &= S_i(t) + p_j \max\{0, S_{\max} - S_i(t)\} - hC_i(t) - kA_i(t) \\ p_j &= p + \beta \exp(-\gamma A_j(t)) \end{aligned}$$

où :

- $C_i(t)$ représente l'intensité de fringale de l'individu i à l'instant t .
- $S_i(t)$ est le niveau de self-control de l'individu i .
- $A_i(t)$ est l'intensité de l'addiction pour l'individu i .
- α , β , et γ sont les coefficients qui modélisent l'influence réciproque entre les deux individus.
- p_j est un paramètre de renforcement pour l'individu i , influencé par l'individu j .
- d, b, h, k sont les mêmes paramètres que dans le modèle pour une personne.

Ce modèle met en évidence l'influence réciproque entre deux personnes, où l'addiction d'un individu peut affecter à la fois sa propre dynamique et celle de l'autre individu, créant ainsi une interaction complexe.

2 - Présentation du programme C++

Après avoir compris le problème et les équations qui le décrivent, il est essentiel de réfléchir attentivement à l'approche à adopter pour implémenter ces équations dans un programme C++. Ce programme doit être optimisé non seulement en termes de mémoire et de temps d'exécution, mais également en assurant une représentation fidèle du modèle mathématique. De plus, il doit être conçu de manière à être réutilisable et facile à manipuler.

Pour atteindre ces objectifs, nous avons choisi d'organiser le code en différentes classes. Ces classes représentent à la fois le modèle pour une et deux personnes ainsi que les système d'équations associés, permettant ainsi une gestion modulaire et évolutive du programme.

2.1 - Diagramme de Classes UML

Afin d'illustrer la phase de conception du code et les éléments qu'il contient, nous présentons ci-dessous un diagramme de classes UML simplifié. Ce diagramme met en évidence les classes créées ainsi que les relations entre elles :

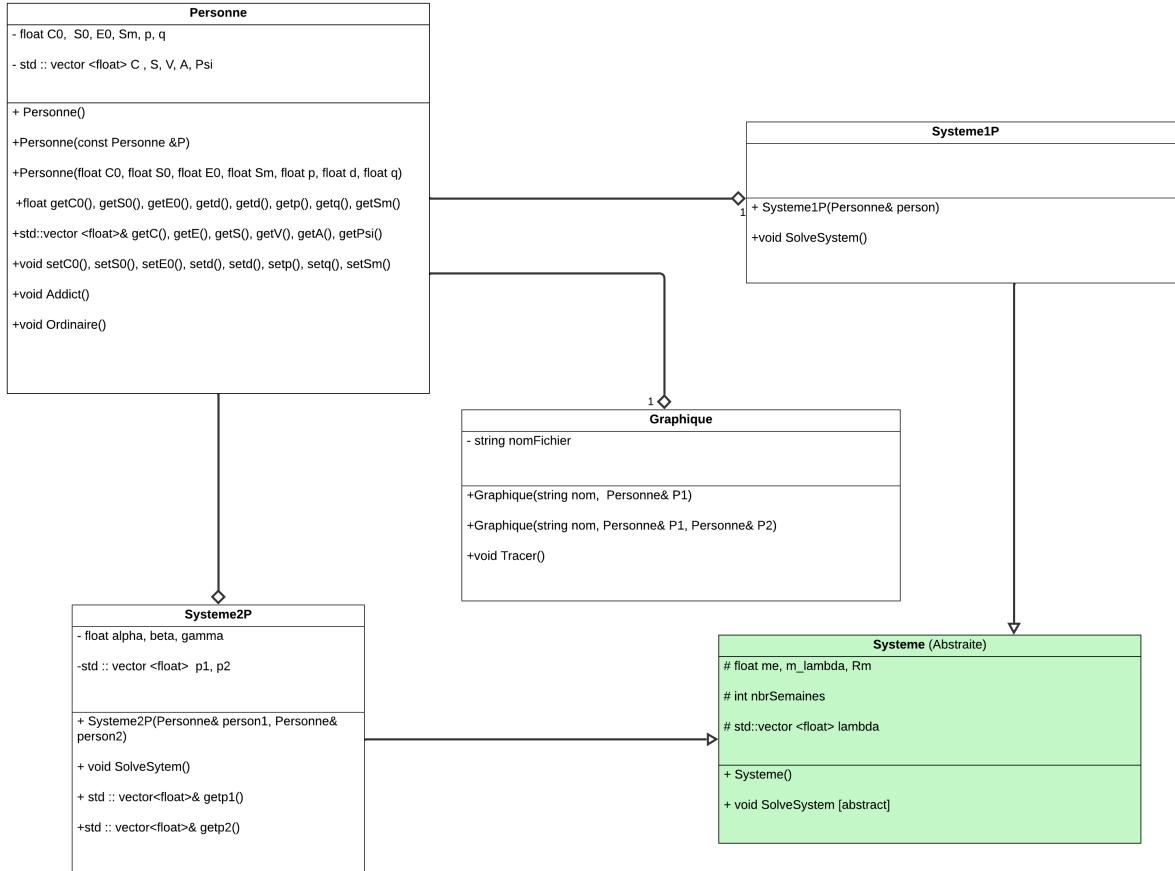


FIGURE 1 – Diagramme de classes UML du programme C++

2.2 - Explication du Code implémenté

Personne : La classe **Personne** nous permet de créer un individu et les caractéristiques liés à son addiction. Les attributs de la classe **Personne** sont indiqués dans le diagramme. On définit ensuite les différents constructeurs (par défaut, par copie et spécifique). Cela nous permettra d'instancier des objets Personnes avec des comportements différents que l'on fera interagir par la suite. Dans cette logique, on commence par implémenter les getters et les setters classiques, permettant de modifier et d'accéder à chacun des attributs de l'individu. Ensuite, on crée deux setters qui permettent à l'utilisateur de définir une personne ordinaire et une personne addict plus simplement sans avoir d'idée sur les ordres de grandeur des paramètres.

Système : Comme nous l'avons vu dans la partie 1, deux situations se présentent, la première correspond à un individu seul (qui n'interagit avec personne mais qui est tout de même influencé par la société) et la seconde, à deux individus qui se côtoient et qui s'influencent mutuellement. Pour décrire la dynamique de ces deux situations, deux systèmes différents sont nécessaires. Elles possèdent cependant des caractéristiques

communes que l'on regroupe dans la classe abstraite `Système`. On y définit donc les paramètres caractérisant notre système. De plus, cette classe est abstraite car elle contient la méthode abstraite “`solveSysteme()`” qui sera implémentée plus tard dans les classes `Système1P` et `Système2P`.

Système1P : La classe `Système1P` permet la résolution du système modélisant le comportement d'une personne seule. Cette classe hérite de la classe `Système`, en effet, elle constitue un système particulier. Elle contient un attribut : une personne, un constructeur spécifique qui prend en arguments une personne. Dans le .cpp, il est nécessaire que le constructeur appelle le constructeur de la classe mère (`Système`). Cela permet à l'objet de la classe `Système1P` de correctement hérité des attributs de la classe `Système`. Ensuite on définit la méthode abstraite `solveSystem` en l'adaptant à notre système. Dans cette méthode, on initialise les différents paramètres (pour éviter d'appeler les getters à chaque itération), puis on résout le système en faisant `nbrsemaines-1` fois itérations (si on le fait `nbrsemaines` fois on sort du tableau).

La difficulté ici (pour `Système2P` aussi) est la génération d'un nombre selon une distribution de Poisson qui se fait en 4 étapes : tout d'abord, on initialise la graine, puis on initialise un générateur de nombres aléatoires avec cette graine, ensuite on définit une distribution de poisson de paramètre `lambda[i]`, et enfin, on génère un nombre aléatoire en utilisant la distribution de Poisson.

Système2P : La classe `Système2P` concerne la résolution du modèle pour deux personnes. Ses attributs (détailé dans le diagramme UML) permettent de définir le Système à deux personnes. Son constructeur prend en argument deux personnes et appelle le constructeur de `Système` pour les mêmes raisons que `Système1P`. Enfin, nous redéfinissons la méthode abstraite `solveSysteme` en l'adaptant au système pour deux personnes.

Graphique : Cette classe permet de générer des fichiers .csv nécessaires pour tracer des courbes avec Gnuplot. Le constructeur de la classe prend en paramètres le nom souhaité pour le fichier .csv, ainsi que la ou les personnes dont on veut tracer les courbes. La méthode `Tracer()` génère ensuite le fichier en enregistrant les données nécessaires.

Comparaison entre le Code Python et le Code C++ du Projet d'Addiction

Dans cette étude, nous comparons les deux versions du code (Python et C++) pour le projet d'addiction selon trois aspects clés : temps d'exécution, flexibilité, et extensibilité/reutilisabilité.

Temps d'Exécution

La version C++ est significativement plus rapide que la version Python. Le C++ permet une gestion fine des ressources et réduit la surcharge mémoire en utilisant des références (`Personne&`) et des destructeurs personnalisés. La manipulation efficace des données via `std::vector` améliore la performance des calculs, un atout essentiel pour des simulations intensives.

Flexibilité

La transition vers la programmation orientée objet en C++ a introduit une meilleure modularité grâce aux classes `Personne`, `Système1P`, et `Système2P`. Chaque composant peut être modifié indépendamment, facilitant les ajustements du modèle ou des paramètres sans réécrire de larges portions de code.

Extensibilité et Réutilisabilité

La structure orientée objet rend le code C++ plus extensible et réutilisable que la version Python. Cette organisation permet d'ajouter de nouvelles classes ou de modifier les classes existantes sans altérer la base du code. Cette approche modulaire rend le système plus adaptable à des extensions futures, tout en assurant une meilleure maintenabilité du code.

III - Projet 2 : Techniques de Compression d'images

La compression d'image est devenue un besoin crucial avec la croissance exponentielle des données visuelles. La transformée en cosinus discrète (DCT) est une méthode largement utilisée dans les algorithmes de compression comme JPEG. Elle permet de transformer une image de l'espace spatial vers l'espace fréquentiel, où les détails perceptibles par l'œil humain sont séparés des informations plus subtiles. En comprimant ces dernières, on réduit efficacement la taille des images tout en maintenant une bonne qualité visuelle. Dans ce projet, la DCT est appliquée sur des blocs de 8x8 pixels, et une quantification est ensuite utilisée pour optimiser le stockage des données.

1 Présentation du programme C++

Le programme de compression d'images est structuré de manière modulaire et extensible, en utilisant des concepts tels que l'héritage et le polymorphisme pour gérer différents types de compresseurs.

1.1 Diagramme de Classes UML

Le diagramme UML illustre l'architecture du programme, basée sur une classe abstraite `CompresseurAbstrait`, qui définit les interfaces pour la compression, la décompression, et d'autres opérations. La classe `CompressionImage`, dérivée de cette classe abstraite, implémente la compression par DCT. La `CompresseurFactory` permet de créer des objets compresseurs de manière dynamique, tandis que `ProcesseurImage` et `GestionnaireImage` offrent des outils pour gérer les images et évaluer la performance de la compression.

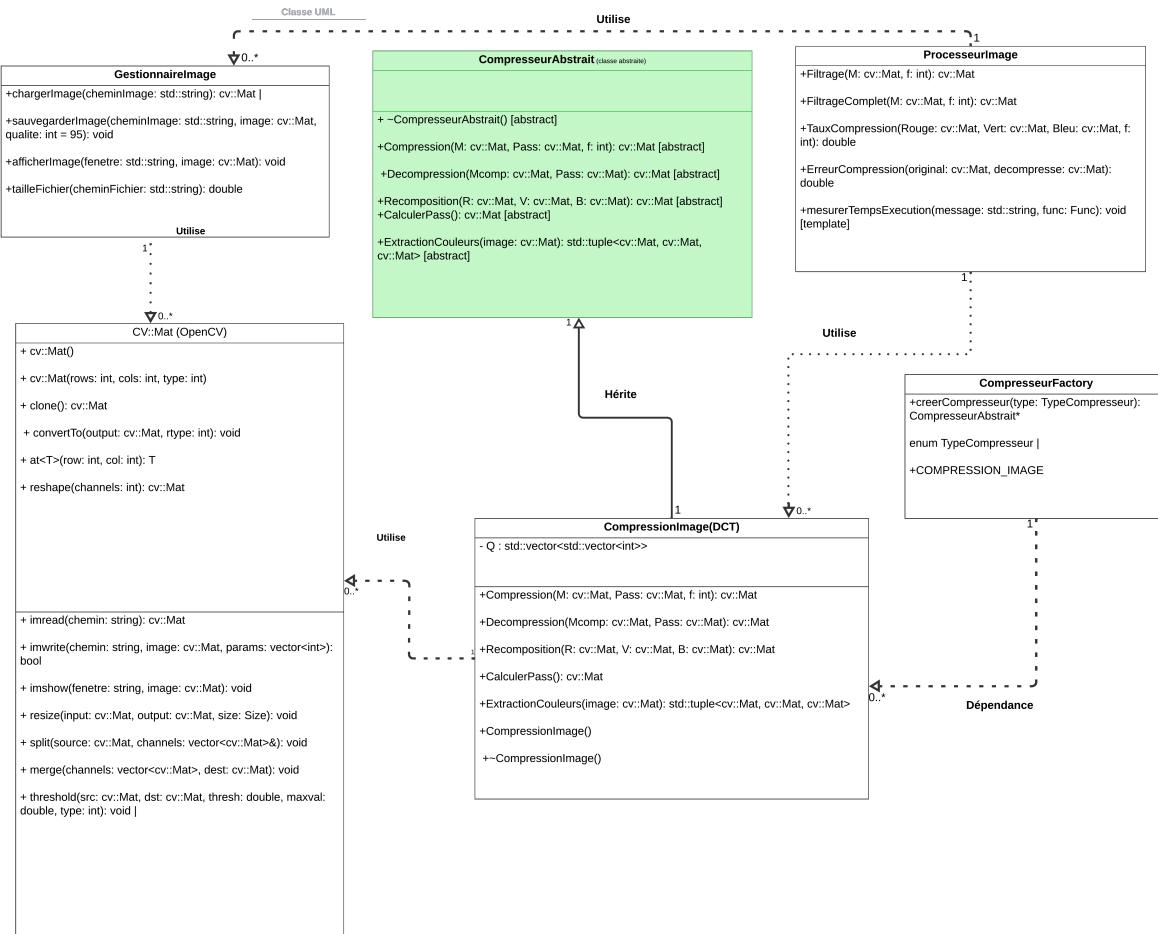


FIGURE 2 – Diagramme de classes UML du programme C++

2 Relations entre les classes et fonctionnement du programme

Le programme est structuré autour de plusieurs classes interconnectées pour implémenter la compression d'images via la DCT (Discrete Cosine Transform). Chaque classe joue un rôle bien défini dans le processus de compression et de gestion des images. Les sections suivantes détaillent chaque classe et leur rôle dans l'architecture du programme.

2.1 Classe CompresseurAbstrait

Objectif : La classe **CompresseurAbstrait** établit une base commune pour tous les compresseurs futurs. Elle impose une structure pour les opérations essentielles comme la compression, la décompression, ou l'extraction des couleurs.

Fonctionnement : Cette classe abstraite définit les méthodes que toutes les implementations spécifiques de compresseurs doivent obligatoirement fournir : - Compression et décompression des données. - Gestion des canaux de couleur.

Cela garantit l'extensibilité du programme en permettant l'ajout futur d'autres algorithmes de compression sans modifier le reste du code.

```
1 class CompresseurAbstrait {
2     public:
```

```

3     virtual cv::Mat Compression(const cv::Mat& M, const cv::Mat
4       & Pass, int f) = 0;
5     virtual cv::Mat Decompression(const cv::Mat& Mcomp, const
6       cv::Mat& Pass) = 0;
7     virtual ~ComptesseurAbstrait() {}
8 };

```

Listing 1 – Extrait de la classe ComptesseurAbstrait

2.2 Classe CompressionImage

Objectif : La classe `CompressionImage` implémente les détails de la compression et de la décompression via la transformation DCT (Discrete Cosine Transform). Elle gère également la séparation et la recomposition des canaux de couleur.

Fonctionnement :

- **Compression :**
 - Chaque bloc 8x8 de l'image est transformé à l'aide de la matrice DCT (`Pass`) et de sa transposée.
 - Un filtrage est appliqué pour conserver uniquement les basses fréquences.
 - La quantification réduit encore les données en divisant les coefficients par une matrice prédéfinie (`Q`).
- **Décompression :**
 - Les coefficients quantifiés sont multipliés par la matrice `Q` pour retrouver leurs valeurs approximatives.
 - La DCT inverse reconstruit chaque bloc dans l'espace spatial, formant ainsi l'image décompressée.

```

1 cv::Mat CompressionImage::Compression(const cv::Mat& M, const cv::
2   Mat& Pass, int f) {
3   cv::Mat PassT;
4   cv::transpose(Pass, PassT);
5   cv::Mat Mcomp = cv::Mat::zeros(M.size(), CV_64F);
6   // Appliquer la DCT et la quantification bloc par bloc
7   for (int i = 0; i < M.rows; i += 8) {
8     for (int j = 0; j < M.cols; j += 8) {
9       cv::Mat bloc = M(cv::Rect(j, i, 8, 8));
10      cv::Mat dctBloc = Pass * bloc * PassT;
11      // Filtrage et quantification
12      for (int x = 0; x < 8; ++x) {
13        for (int y = 0; y < 8; ++y) {
14          if (x + y >= f) dctBloc.at<double>(x, y) = 0;
15          Mcomp.at<double>(i + x, j + y) = round(dctBloc.
16            at<double>(x, y) / Q[x][y]);
17        }
18      }
19    }
20  }
21  return Mcomp;
22 }

```

Listing 2 – Extrait de CompressionImage

2.3 Classe CompresseurFactory

Objectif : La classe `CompresseurFactory` simplifie la création d'objets dérivés de `CompresseurAbstrait` en utilisant le design pattern *Factory*. Cela permet de masquer les détails de l'instanciation et de gérer dynamiquement différents types de compresseurs.

Fonctionnement : Cette classe crée des instances spécifiques de compresseurs, comme `CompressionImage`, en fonction du type spécifié (`TypeCompresseur`). Ce mécanisme garantit une extensibilité future pour intégrer de nouveaux algorithmes de compression.

```

1 CompresseurAbstrait* CompresseurFactory::creerCompresseur(
2     TypeCompresseur type) {
3     if (type == TypeCompresseur::COMPRESSION_IMAGE) {
4         return new CompressionImage();
5     }
6     return nullptr;
}
```

Listing 3 – Extrait de CompresseurFactory

2.4 Classe ProcesseurImage

Objectif : La classe `ProcesseurImage` offre des outils pour évaluer les performances de la compression. Elle calcule notamment : - Le taux de compression. - L'erreur moyenne entre l'image originale et l'image décompressée.

Fonctionnement :

- **Taux de compression :** Mesure le ratio entre les coefficients nuls et non nuls après compression.
- **Erreur de compression :** Basée sur la norme euclidienne, elle quantifie la différence entre les images originale et décompressée.

```

1 double TauxCompression(const cv::Mat& Rouge, const cv::Mat& Vert,
2                         const cv::Mat& Bleu, int f) {
3     int total_pixels = Rouge.rows * Rouge.cols;
4     cv::Mat RougeComp = FiltrageComplet(Rouge, f);
5     int non_zero_rouge = cv::countNonZero(abs(RougeComp) > 0);
6     return (1.0 - static_cast<double>(non_zero_rouge) /
7             total_pixels) * 100;
}
```

Listing 4 – Extrait de ProcesseurImage

2.5 Classe GestionnaireImage

Objectif : Cette classe gère les interactions avec les fichiers d'image, notamment leur chargement, affichage et sauvegarde.

Fonctionnement : Elle utilise des fonctions OpenCV comme `imread()` pour lire une image depuis un fichier et `imwrite()` pour l'enregistrer. Ces outils facilitent le traitement des images au format matriciel.

```

1 cv::Mat GestionnaireImage::chargerImage(const std::string&
2   cheminImage) {
3   return cv::imread(cheminImage, cv::IMREAD_COLOR);
}
```

Listing 5 – Extrait de GestionnaireImage

2.6 Usage de OpenCV dans le programme

OpenCV joue un rôle essentiel dans la gestion et le traitement des images tout au long du programme. Voici les principaux usages de cette bibliothèque :

- **Chargement et sauvegarde :** Les fonctions `cv::imread()` et `cv::imwrite()` permettent de charger et de sauvegarder des images au format JPEG ou PNG.
- **Traitement matriciel :** Les images sont représentées sous forme de matrices (`cv::Mat`), facilitant les opérations mathématiques comme la DCT ou le filtrage.
- **Séparation des canaux :** `cv::split()` permet de diviser une image en ses canaux rouge, vert et bleu.
- **Transformation DCT :** Les fonctions `cv::dct()` et `cv::idct()` sont utilisées pour effectuer la transformation et son inverse.

2.7 Usage dans main.cpp

Le fichier `main.cpp` orchestre l'ensemble du processus de compression et de décompression. Les étapes principales incluent :

1. Chargement de l'image via `GestionnaireImage`.
2. Création d'un compresseur via `CompresseurFactory`.
3. Application de la compression et décompression sur les canaux R, V, B.
4. Recomposition et sauvegarde de l'image décompressée.

```

1 Mat image = GestionnaireImage::chargerImage("image.jpg");
2 auto compresseur = CompresseurFactory::creerCompresseur(
3   TypeCompresseur::COMPRESSION_IMAGE);
4 tie(Rouge, Vert, Bleu) = compresseur->ExtractionCouleurs(image);
5 Mat imageDecompressee = compresseur->Recomposition(Rouge, Vert,
6   Bleu);
GestionnaireImage::sauvegarderImage("image_decompressee.jpg",
7   imageDecompressee);
```

Listing 6 – Extrait de main.cpp

3 Analyse des performances du code

Le programme a été testé pour comparer les performances de la version C++ avec la version Python, notamment au niveau du taux de compression, de l'erreur moyenne et du temps d'exécution. Les résultats obtenus sont résumés dans le tableau ci-dessous.

Critère	Python	C++
Taux de compression (%)	94	94
Erreur de compression moyenne (%)	6.63	0.35
Temps d'exécution (secondes)	32	0.6

TABLE 1 – Comparaison des performances entre Python et C++

3.1 Précision supérieure en C++

- **Optimisation native** : C++ compile directement en instructions machine, réduisant les erreurs d'arrondi et offrant un meilleur contrôle sur les types (`float`, `double`).
- **Gestion explicite de la mémoire** : La gestion mémoire en C++ minimise les copies inutiles et optimise l'utilisation des ressources, augmentant la précision des calculs.
- **Utilisation native d'OpenCV** : En C++, les matrices cv : :Mat d'OpenCV optimisent la gestion mémoire et évitent les copies inutiles

3.2 Compromis entre performance et temps d'exécution

- **Réduction de la précision** : Utiliser des types moins précis (`float` au lieu de `double`) peut accélérer les calculs, mais au prix d'une légère augmentation de l'erreur de compression.
- **Paramètres ajustables** : Une compression plus rapide peut être obtenue en réduisant la fréquence de filtrage (`f`) ou en augmentant les valeurs de quantification, bien que cela diminue la qualité visuelle de l'image.

3.3 Résumé des avantages

La version C++ offre une précision supérieure et des temps d'exécution plus rapides grâce à ses optimisations natives et sa gestion fine des ressources. Les compromis entre qualité et rapidité peuvent être ajustés selon les besoins spécifiques, l'utilisateur peut choisir la fréquence.

IV Conclusion

Ce projet a démontré l'efficacité du C++ dans l'optimisation d'un algorithme de compression d'image. Initialement écrit en Python, le passage au C++ a permis de réduire significativement le temps d'exécution tout en maintenant un taux de compression constant et une erreur moindre. La meilleure gestion de la mémoire, ainsi que l'architecture modulaire basée sur l'héritage et le polymorphisme, garantit une extensibilité future.

En effet, grâce à la classe abstraite `CompresseurAbstrait` et au design pattern `Factory`, il est facile d'intégrer d'autres algorithmes de compression, tels que la compression par ondelettes ou sans perte. Il serait également possible d'optimiser la DCT en effectuant plusieurs tests d'ajustement sur le filtrage et la matrice de quantification.

La compression réalisée dans ce projet est un succès, car elle a permis de réduire la taille d'une image de 1700ko à 380ko ($f=10$). De plus, l'utilisateur peut encore diminuer la taille de l'image en choisissant une fréquence plus basse.

V - Annexe : Mode d'usage et installation du Programme

Cette annexe fournit un guide complet pour installer et exécuter les projets **Addiction** et **Compression d'image**. Vous apprendrez à configurer et compiler les deux projets en utilisant **CMake** et les bibliothèques externes **OpenCV** et **Gnuplot**.

1 Prérequis

Assurez-vous d'avoir les éléments suivants installés sur votre système :

- **CMake** (version minimale : 3.10 pour Compression d'image et 3.14 pour Addiction)
- **C++17** ou supérieur
- **OpenCV** pour le projet Compression d'image
- **Gnuplot** pour le projet Addiction

2 Présentation de l'arborescence des projets

Chaque projet contient plusieurs fichiers C++ et un fichier **CMakeLists.txt** spécifique qui gère la configuration et la compilation.

Organisation des instructions spécifiques :

- **Compression d'image** : utilise **OpenCV** pour la manipulation d'images.
- **Addiction** : utilise **gnuplot** pour la visualisation graphique des données.

3 Installation des dépendances

Installation (OpenCV)	Addiction (gnuplot)
<ul style="list-style-type: none"> — Pour Ubuntu : <pre>sudo apt update sudo apt install libopencv-dev</pre> — Pour macOS (avec Homebrew) : <pre>brew update brew install opencv</pre> — Pour Windows : <ol style="list-style-type: none"> 1. Télécharger OpenCV : https://opencv.org/releases/ 2. Extraire et configurer le PATH : <pre>C:\opencv\build</pre> 	<ul style="list-style-type: none"> — Installation de gnuplot : <pre># Pour Ubuntu : sudo apt update sudo apt install gnuplot</pre> — # Pour macOS avec Homebrew : <pre>brew install gnuplot</pre>

4 Compilation et exécution des projets

Étapes générales de compilation (communes aux deux projets) :

```
cd build
cmake ..
make
```

Projet Compression d'image	Projet Addiction
Compilation et exécution : <pre>mkdir build cd build cmake .. make ./compression_image_executable</pre>	Compilation et exécution : <pre>mkdir build cd build cmake .. make ./addiction_executable</pre>

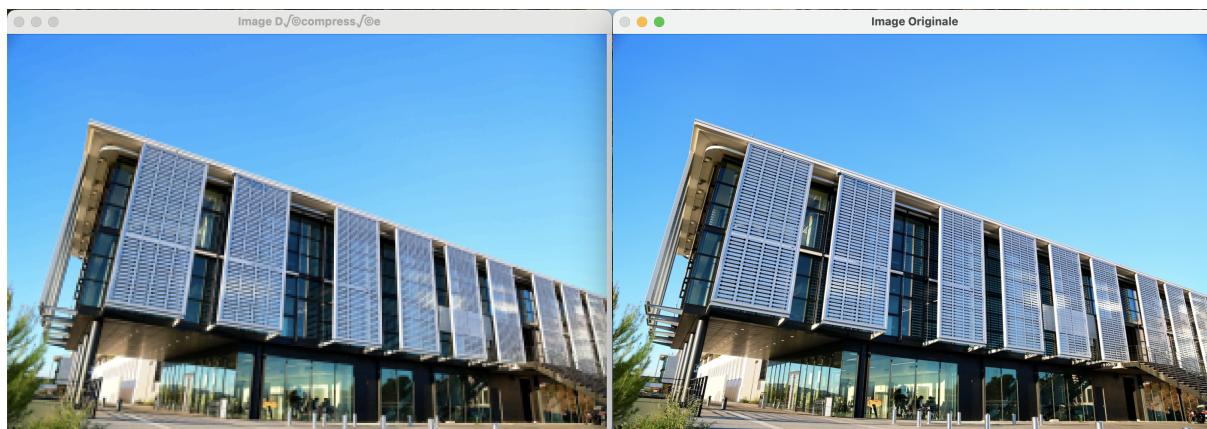


FIGURE 3 – Image de Référence Compressée

```
g_aloui@MacBook-Air-de-Ghaieth build % ./test
Taille de l'image initiale : 1648.32 kB
Entrez la fréquence de filtrage (ex: 10), plus la fréquence est basse, plus l'image va être compressée !: 10
Extraction des canaux de couleur : 0.00367896 secondes.
Calcul de la matrice DCT : 1.5709e-05 secondes.
Compression des canaux : 0.237163 secondes.
Décompression des canaux : 0.212029 secondes.
Recomposition de l'image : 0.0140334 secondes.
Taille de l'image décompressée : 384.981 kB
L'image a bien été compressée, elle est passée de 1648.32 kB à 384.981 kB.
Taux de compression : 93.8311%.
Calcul du taux de compression : 0.082878 secondes.
Erreur de compression (moyenne) : 0.00351084
Calcul de l'erreur de compression : 0.0351145 secondes.
```

FIGURE 4 – Sortie attendue du Programme

5 Remarques finales

Les projets sont conçus pour être facilement configurables et extensibles. En cas de modification des dépendances ou des bibliothèques, mettez à jour les fichiers `CMakeLists.txt` en conséquence.