Université de Montpellier
Master 1 - IMAGINE

ColorHarmony

Harmonisation des couleurs

Rapport de Projet : Harmonie des couleurs
Projet informatique — HAI804I & HAI809I

Étudiants :
M. Camille BERNADAS
M. Khélian LARVET

Encadrants :
Mme. Bianca JANSEN VAN RENSBURG
M. William PUECH

Année : 2021 - 2022

Nous tenons à remercier nos encadrants Mme. Bianca JANSEN VAN RENSBURG ainsi que M. William PUECH pour leur accompagnement, leur bienveillance et leurs précieux conseils tout au long de ce projet qui nous ont permis de le porter à son état actuel. Nous remercions également Pascal LARVET, pour sa relecture attentive de ce rapport.

Table des matières

Introduction **2**

État de l'art **4**

1 Harmonie des couleurs **5**
1.1 Définition 5
1.2 Types d'harmonies 5
1.3 Algorithme d'harmonisation 6
1.3.1 Préparation des variables selon le type d'harmonie choisi 6
1.3.2 Conversion vers l'espace HSL 6
1.3.3 Application de l'harmonisation 6
1.3.4 Conversion vers l'espace RGB 7
1.3.5 Apparition d'artéfacts et perte de cohérence 7

2 Choix des couleurs et scores **9**
2.1 Choisir une couleur pertinente 9
2.2 Calculer un score 10

3 Interfaçage Gtkmm **11**
3.1 Widgets 11
3.2 Signaux 12

4 Bilan et perspectives d'amélioration **14**
4.1 Avancement du projet 14
4.2 Difficultés rencontrées 14
4.3 Améliorations possibles 14

Annexes **15**

A Extraits de code **16**
A.1 Conversion RGB - HSL 16
A.2 Algorithme d'harmonisation 17
A.3 Algorithme K-means (couleur dominante) 18

Bibliographie **19**

Introduction

Dans le cadre des modules "Analyse et traitement des images" et "Codage et compression multimédia" du second semestre de M1 IMAGINE, nous avons développé l'application **ColorHarmony** dont le dépôt est disponible à l'adresse suivante :

`https://gitlab.etu.umontpellier.fr/e20170006995/color-harmony`

Cette application permet d'harmoniser les couleurs d'une image selon plusieurs types d'harmonies avec une aide à la selection de la couleur à harmoniser.

Motivations du projet

Notre première motivation sur ce projet a été la possibilité d'analyser les différentes harmonies qui sont couramment utilisées dans les théories des couleurs et comprendre pourquoi ces combinaisons donnent un résultat esthétiquement plaisant.

Nous avons également été intéressés par l'impact des couleurs dans une image. En effet, les couleurs peuvent être considérées comme un support de communication visuel permettant un échange d'informations implicites entre l'artiste et le spectateur.

Ce projet nous semblait donc parfait pour comprendre comment les couleurs peuvent se combiner pour donner des résultats visuellement attrayants et par la même occasion, analyser comment implémenter diverses méthodes pour harmoniser des images.

Objectifs du projet et cahier des charges

Quatre objectifs ont été fixés :

- Le premier a été l'approfondissement de notre compréhension sur le domaine des couleurs et des harmonies. Puis l'analyse des différents procédés utilisés pour transformer les couleurs dans une image.
- Le second s'est porté sur l'implémentation d'une méthode traditionnelle pour harmoniser les couleurs d'une image.
- Le troisième fût la mise en place de fonctionnalités pour attribuer un score et aider l'utilisateur dans le choix de la couleur à harmoniser pour une image donnée.
- Le dernier s'est porté sur la réalisation d'un logiciel avec une interface graphique permettant aux utilisateurs d'accéder aux différentes fonctionnalités et par la même occasion d'afficher rapidement l'image résultante pour une couleur et une harmonie choisie.

Technologies utilisées

Le choix a été fait en accord avec nos compétences respectives et dans le respect de l'unique contrainte imposée : il nous fallait pouvoir manipuler des images au format PPM. Nous avons aussi choisi d'implémenter une interface graphique pour faciliter l'utilisation de notre outil.

C++ :
Langage de programmation compilé permettant la programmation sous de multiples paradigmes. Il nous permet d'améliorer nettement les performances de notre projet.

Bibliothèque Gtkmm :
Ensemble de bibliothèques logicielles permettant de réaliser une interface graphique en C++. Il nous permet d'avoir un aperçu rapide après l'application d'une harmonie sur une image.

État de l'art

Problématique

Une grande partie de ce que nous percevons et ressentons à propos d'une image est vécue à travers les couleurs. Leur utilisation est donc critique, surtout dans le domaine du design et de la visualisation pour exprimer une information implicite particulière.

À l'heure actuelle, il n'existe aucune définition formelle concernant le principe d'harmonie des couleurs. Nous pouvons décrire ce concept par les ensembles de couleurs qui par leur combinaison, produisent un sentiment esthétique agréable par la perception visuelle.

De manière plus concrète, des couleurs harmoniques peuvent être créées en choisissant des couleurs en fonction de leur position relative dans un espace colorimétrique (roue chromatique HSL).

Notre objectif est donc de développer des outils permettant une harmonisation automatique des couleurs pour une image donnée.

État de l'art

De nombreux travaux développent plusieurs techniques d'harmonisation des couleurs pour permettre aux artistes d'améliorer la composition des couleurs de leurs images.

La méthode traditionnelle la plus connue consiste à récupérer dans un premier temps un histogramme de teinte de l'image puis d'appliquer un modèle harmonique. Un modèle harmonique est défini comme des secteurs de teinte dans l'espace colorimétrique HSL (complémentaire, analogique, triadique, etc).
Ce modèle (optimal ou arbitraire) permet ensuite de modifier les teintes de l'image en fonction d'un angle dans l'espace HSL par rapport à l'histogramme de l'image. Un traitement supplémentaire est nécessaire pour assurer la fluidité spatiale. En effet, l'application effectue des modifications pixel par pixel sans prendre en compte les formes décrites par l'image. La fluidité spatiale peut ainsi être assurée par l'une des méthodes suivantes :

- une optimisation "graph-cut" (régions spatiales contiguës).
- une segmentation par décalage moyen.
- une segmentation binaire optimale.

Nous avons également constaté l'existence d'autres méthodes, qui utilisent des espaces colorimétriques différents comme par exemple l'espace HCL. Nous avons donc choisi d'utiliser les méthodes traditionnelles les plus courantes car elles sont plus rapides à implémenter et bien suffisantes pour nos objectifs. Nous avons aussi une contrainte de temps à respecter dans le cadre de ce projet.

Chapitre 1

Harmonie des couleurs

1.1 Définition

Dans la théorie des couleurs, l'harmonie des couleurs fait référence à l'effet agréable obtenu en utilisant certaines combinaisons de couleurs. Nous pouvons créer ces harmonies facilement dans l'espace colorimétrique TSL (ou HSL en anglais) avec les composantes : teinte, saturation, luminosité.

Dans notre cas, nous définissons une harmonie en choisissant une teinte à l'aide d'un angle dans la roue chromatique (intervalle [0 ; 360[). Par défaut notre application choisit un angle de 0, donc du rouge pur.

À partir de cette valeur, nous pouvons calculer les autres valeurs de teinte selon le type d'harmonie utilisée. Chacune de ces valeurs définit ce qu'on appellera une "bande", qui est un ensemble de couleur ayant la même valeur de teinte mais avec des valeurs différentes de saturation et de luminosité.

Enfin, nous définissons la quantité du spectre des couleurs que l'on souhaite conserver. Cette valeur, elle aussi comprise dans l'intervalle [0 ; 360[, sert à définir la surface totale utilisée par les bandes sur la roue chromatique.

1.2 Types d'harmonies

Il existe plusieurs types d'harmonie et nous avons choisi d'implémenter les suivants :

Harmonie monochrome

Harmonie de couleurs semblables

Harmonie complémentaire

Harmonie complémentaire séparée

Harmonie triadique

Harmonie carrée

1.3 Algorithme d'harmonisation

1.3.1 Préparation des variables selon le type d'harmonie choisi

L'utilisateur peut donc choisir la teinte principale de l'harmonisation, le type d'harmonie souhaité et la largeur de bande à utiliser sur la roue chromatique pour une image donnée. Une fois ces informations envoyées, nous pouvons commencer le processus d'harmonisation.

Selon le type d'harmonie choisi, nous calculons les valeurs des centres des autres bandes :
- **Harmonie de couleurs semblables et monochrome :** pas d'autre bande ;
- **Harmonie complémentaire :** une autre bande éloignée de 180 degrés ;
- **Harmonie complémentaire séparée :** deux autres bandes, la première éloignée de 150 degrés et la seconde de 210 degrés ;
- **Harmonie triadique :** deux autres bandes, éloignées de 120 degrés de la précédente.
- **Harmonie carrée :** trois autres bandes, éloignée de 90 degrés de la précédente.

La largeur de bande à utiliser sur la roue chromatique permet de définir la largeur des bandes dont la valeur du centre a été définie. On considère cette quantité comme un angle que l'on réparti de manière égale entre chaque bande. Par exemple, dans le cas d'une harmonie carrée, cet angle x est divisé en 4 et chaque bande a une largeur de x/4.

Ainsi, avec un angle de 360 degrés, l'image originale n'est pas du tout modifiée : on conserve l'intégralité des couleurs présentes. Avec un angle de 180 degrés, on divise par deux le nombre de teintes différentes présentes dans l'image. Avec un angle proche de 0 degré, on essaie de ne conserver que les teintes pures définies par les bandes.

L'exception à ces règles est l'harmonie monochrome, où la valeur renseignée par l'utilisateur est ignorée par le programme qui utilise un angle de 1 degré.

1.3.2 Conversion vers l'espace HSL

Pour pouvoir facilement changer la teinte d'un pixel, il est nécessaire de convertir sa couleur de l'espace RGB à l'espace HSL (voir annexe A.1), pour obtenir les trois composantes : Teinte, Saturation et Luminosité.

Notons que les nuances de gris n'ont pas de valeur de teinte définie. Nous leur attribuons alors la teinte par défaut.

1.3.3 Application de l'harmonisation

Le processus d'harmonisation (voir annexe A.2) s'exécute indépendamment pour chaque pixel de l'image ce qui engendrera des problèmes que nous détaillerons dans la sous-section 1.3.5.

Ainsi, pour chaque pixel de l'image, nous calculons sa valeur dans l'espace HSL, puis nous cherchons quelle est la bande la plus proche de la valeur de teinte du pixel. La bande la plus proche est considérée comme la valeur de référence.

Nous pouvons ensuite calculer la distance signée entre la valeur de référence et la valeur de teinte du pixel, comprise dans l'intervalle]-180, 180] lorsqu'une seule bande est utilisée. On passe alors dans un modèle centré sur la valeur de référence.

Cette distance signée est divisée par la demi-longueur d'influence de la bande, c'est-à-dire la distance maximale entre le centre de la bande et un point qui l'a comme référence. Cette demi-longueur est définie par le nombre de bandes.

```
float influenceSize = (float)(360. / bands.size());
float radius = influenceSize / 2.;
```

Nous obtenons alors une valeur flottante dans l'intervalle [-1, 1]. Cette valeur est multipliée par la demi-largeur de la bande. Cette largeur correspondant à la largeur de bande à utiliser sur la roue chromatique.

Enfin, on ajoute la valeur de référence pour retourner dans un modèle basé sur la roue chromatique.

```
float outColor = (signedDistance / radius) * (bandSize / 2) + desiredBand;
```

1.3.4 Conversion vers l'espace RGB

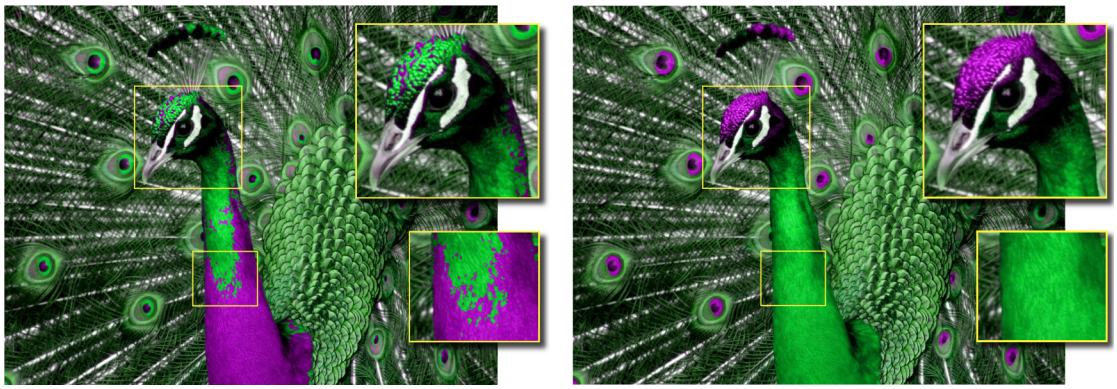
Pour pouvoir sauvegarder l'image au format PPM, il faut reconvertir l'ensemble des pixels de l'image harmonisée de l'espace HSL vers l'espace RGB.

1.3.5 Apparition d'artéfacts et perte de cohérence

Le processus d'harmonisation s'exécutant indépendamment pour chaque pixel, il est possible de remarquer des artefacts. Ces artefacts apparaissent lorsque deux pixels faisant partie d'une même zone de cohérence, ont des couleurs très légèrement différentes et sont attribuées à deux bandes différentes. Ce qui provoque l'apparition d'un fort contraste entre les deux pixels.

Apparition d'artefacts

Ces artefacts peuvent-être corrigés en implémentant un système de coupe graphique de l'image en plusieurs zones de même appartenance ([3]). Ainsi nous pourrions traiter les pixels de chaque zone de façon à ce que le contraste entre eux soient minimisés.



Correction des artefacts avec des zones contigües

Chapitre 2

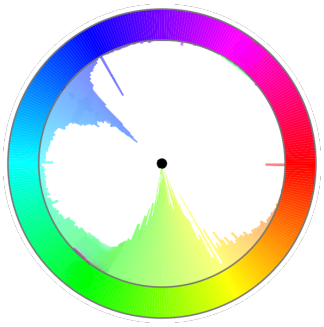
Choix des couleurs et scores

La couleur principale peut être sélectionnée manuellement par l'utilisateur, mais il est nécessaire de faciliter la sélection de cette couleur.

2.1 Choisir une couleur pertinente

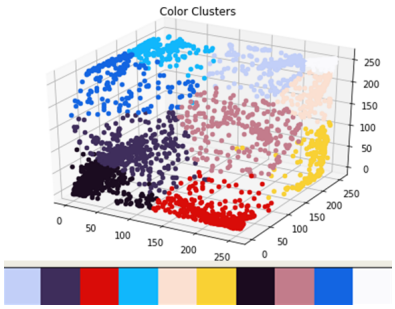
Pour les utilisateurs n'étant pas des artistes, il est difficile de choisir une couleur pertinente afin de conserver l'ambiance initiale de l'image. C'est pourquoi nous avons mis à disposition deux méthodes permettant de récupérer la couleur dominante pour une image donnée.

La première méthode consiste à analyser l'histogramme des teintes de l'image dans l'espace de couleur HSL afin de renvoyer la teinte la plus utilisée. Cette méthode permet ainsi d'avoir rapidement un résultat qui sera constant pour une image donnée.



Histogramme sur la teinte

La seconde méthode utilise l'algorithme du partitionnement en k-moyennes ou K-means en anglais (Code disponible à l'annexe A.3). Cette méthode permet, sur une image, de récupérer les n couleurs dominantes. Dans notre cas, nous renvoyons la couleur ayant le plus gros cluster pour n=5 avec 5 itérations.



Partitionnement en 10-moyennes

Cette méthode peut s'avérer être relativement lente en fonction des paramètres passés (itérations et nombre de cluster) et de la dimension de l'image initiale, c'est pourquoi nous devons réduire sa taille avant d'utiliser cette méthode pour réduire la compléxité.
Les résultats de cette méthode sont par conséquent variables dans le temps car l'initialisation des clusters est aléatoire dans l'image.

Image originale

K-means (K=5)

Image harmonisée

Couleur dominante

2.2 Calculer un score

Le PSNR est utile pour mesurer la proximité entre une image compressée comparée à l'originale. Il nous permet de mesurer la ressemblance entre l'image harmonisée et l'image d'origine. Néanmoins, il ne prend pas en compte la qualité visuelle de reconstruction et ne peut être reconnu comme une mesure objective de la qualité visuelle d'une image.

Il est difficile de mesurer la qualité d'une harmonisation car l'appréciation d'une harmonie de couleurs est subjective. De ce fait, il n'existe pas vraiment de score pour mesurer la qualité visuelle d'une image et l'harmonie de ses couleurs.

Nous avons donc créé notre propre score qui vient compléter l'usage du PSNR. En produisant l'histogramme des teintes utilisées dans notre image, nous pouvons compter combien de teintes ne sont pas utilisées. Plus ce nombre est proche de 360, plus notre score intermédiaire est elevé.

Ce score intermédiaire permet de calculer la différence entre le nombre de teintes utilisées dans l'image d'origine et l'image après le processus d'harmonisation. Plus le nombre de teintes utilisées a diminué, plus ce score final est grand. On considère alors avoir bien harmonisé l'image, en réduisant les nuances de couleurs moins harmonieuses.

Si ce score de teintes non utilisées est grand et que le PSNR l'est aussi, alors l'image produite a été harmonisée tout en restant similaire à l'image d'origine.

Chapitre 3

Interfaçage Gtkmm

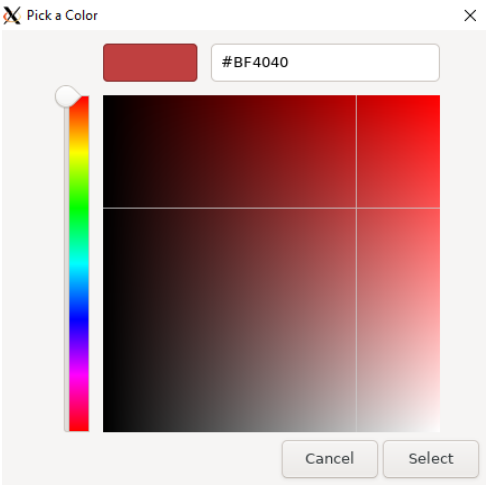
3.1 Widgets

Comme la plupart des bibliothèques graphiques, Gtkmm fonctionne avec le principe des « widgets ». Les widgets sont des fonctions interactives permettant l'affichage d'informations variées comme par exemple des boutons ou des zones de texte.

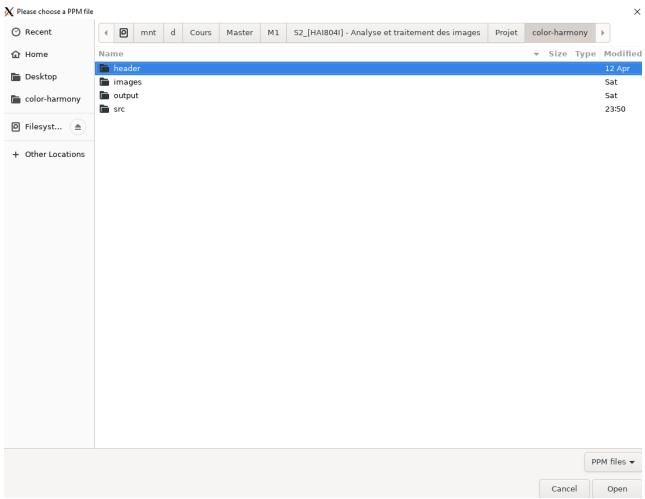
Nous nous sommes limités à des widgets simples dans l'objectif de rendre l'application ergonomique. Voici l'ensemble des widgets utilisés dans notre application :

- Zones de textes
- Zones de saisie
- Images
- Boutons
- Boutons de couleur
- Listes de choix

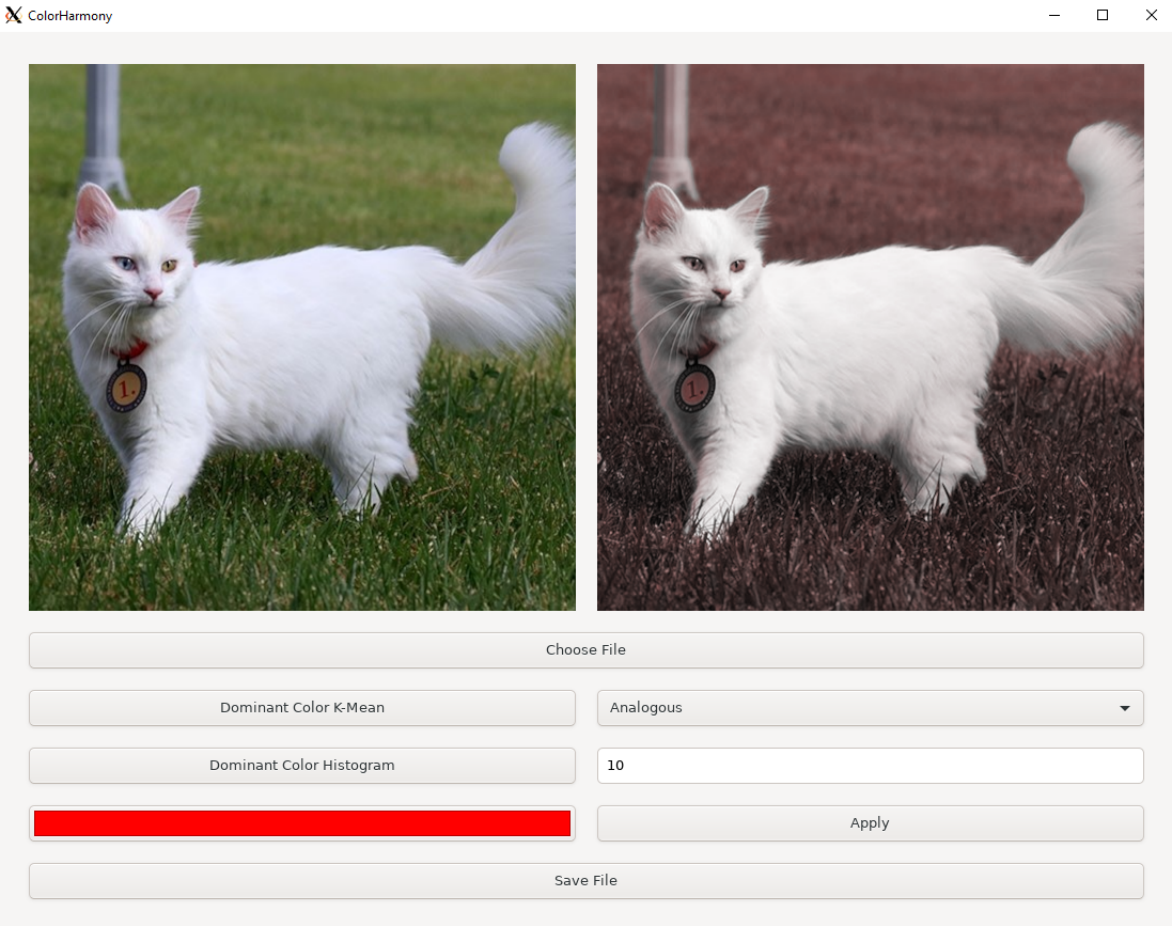
Ainsi notre application possède 10 « widgets » et permet de modifier avec aisance les différents paramètres d'harmonisation. Elle permet également d'avoir un aperçu rapide sans être obligé de sauver l'image pour voir les changements apportées par l'harmonisation.
L'image de gauche indique l'image originale importée, celle de droite affiche l'image originale modifiée par l'harmonisation.



Choix de la couleur



Choix du fichier



Interface de l'application

Voici une liste exhaustive des fonctionnalités accessibles via notre interface :
- Importer une image (au format PPM) ;
- Récupérer la couleur dominante de l'image originale selon son histogramme de teinte ;
- Récupérer la couleur dominante de l'image originale selon le plus gros cluster formé par l'algorithme K-means ;
- Changer la couleur d'harmonisation via un sélecteur de couleur ;
- Modifier le type d'harmonisation à l'aide d'une liste de choix ;
- Modifier les angles formés (en degrés, de [0 ; 360[) dans l'histogramme de teinte pour harmoniser l'image ;
- Appliquer l'harmonisation avec les paramètres séléctionnés sur l'image originale ;
- Sauver l'image harmonisée (au format PPM) ;

3.2 Signaux

Les signaux permettent de notifier un évènement sur un widget parmi une liste d'actions possibles comme par exemple `signal_clicked()` ou `signal_changed()`. Ces signaux favorisent la programmation événementielle et ainsi permettent de faciliter l'interaction de l'utilisateur avec notre application.

Chaque signal peut être surchagé par une méthode pour produire un évènement particulier. Dans notre cas, la plupart des signaux permettaient l'actualisation des données passées entre l'interface et nos fonctions.

Mais un signal en particulier a été intéressant à mettre en place : celui des `FileChooserDialog`.

Les `FileChooserDialog` sont des fenêtres particulières qui permettent d'importer ou d'exporter des fichiers dans l'arborescence du système. Nous avons donc surchargé un signal pour permettre de les manipuler. Cette surcharge se fait en quatre étapes :

Dans un premier temps nous fabriquons le fenêtre de dialogue avec les différentes intéractions possibles et nous la signalons comme éphémère avec la fonction `set_transient_for()` ;

```
// Dialog
Gtk::FileChooserDialog dialog("Choose a file", Gtk::FILE_CHOOSER_ACTION_OPEN);
dialog.add_button("_Cancel", Gtk::RESPONSE_CANCEL);
dialog.add_button("_Open", Gtk::RESPONSE_OK);
dialog.set_transient_for(*this);
```

Puis nous indiquons sa localisation dans l'arborescence du système à l'aide de la fonction `get_current_folder()` ;

```
// Location
dialog.set_current_folder(dialog.get_current_folder());
```

Par la suite, nous pouvons créer des filtres pour indiquer à l'utilisateur les fichiers supportés par l'application en utilisant les formats fournis par la classe `FileFilter` ;

```
// Filters
auto filter_ppm = Gtk::FileFilter::create();
filter_ppm->set_name("PPM files");
filter_ppm->add_pattern("*.ppm");
dialog.add_filter(filter_ppm);
```

Enfin nous lançons la fenêtre de dialogue et nous attendons la réponse de l'utilisateur en analysant le résultat retourné par la fonction `run()` ;

```
// Run
const int result = dialog.run();

// Handle response
switch (result)
{
    case Gtk::RESPONSE_OK:
    {
        [...]
        break;
    }
    case Gtk::RESPONSE_CANCEL:
    {
        printf("Cancel clicked\n");
        break;
    }
    default:
    {
        printf("Unexpected clicked\n");
        break;
    }
}
```

Chapitre 4

Bilan et perspectives d'amélioration

4.1 Avancement du projet

A l'heure actuelle, notre application est totalement fonctionnelle, et suit au mieux le cahier des charges qui nous a été donné. Ainsi notre application donne la capacité à un utilisateur d'importer une image au format PPM, de modifier cette image selon plusieurs critères, et enfin de l'exporter au format PPM si l'image résultante satisfait l'utilisateur.

4.2 Difficultés rencontrées

Notre première difficulté a été d'implémenter correctement un algorithme d'harmonisation basé sur un centrage sur la valeur cible et une réduction vers un intervalle de type [-1,1]. À plusieurs reprises, l'implémentation était adaptée à un certain type d'harmonie mais provoquait des erreurs lorsqu'un autre type d'harmonie était sélectionné.

Nous avons également eu des difficultés à mettre en place une interface graphique avec Gtkmm. En effet, la documentation étant dépréciée et certains modules n'étant plus à jour, nous avons été contraints de consulter plusieurs forums pour pouvoir produire une interface robuste et efficace.

4.3 Améliorations possibles

Plusieurs perspectives d'améliorations sont possibles pour ce projet dans le but de le rendre plus complet, stable et efficace telles que :

- La capacité de traiter d'autres formats d'image que PPM.
- Utiliser la saturation et la luminosité de l'espace HSL pour harmoniser des images.
- Exploiter d'autres espaces de couleur tel que CIE LCh permettant de réduire les calculs pour harmoniser les couleurs d'une image.
- Implémenter une optimisation "Graph cut" permettant la segmentation des régions spatiales contigües afin d'éviter les artéfacts qui réduisent la cohérence de l'image.
- Minimiser la perte de précision avec les conversions successives entre les différents espaces de couleurs et entre les nombres entiers et flottants.

Annexes

Annexe A

Extraits de code

A.1 Conversion RGB - HSL

```
void Vec3HSL::convertRGBtoHSL(int r_in, int g_in, int b_in)
{
    float r = (float)r_in / 255.;
    float g = (float)g_in / 255.;
    float b = (float)b_in / 255.;

    float Xmax = std::max(r, std::max(g, b));
    float Xmin = std::min(r, std::min(g, b));

    float chromaRange = Xmax - Xmin;
    float hue;
    if (chromaRange == 0.)
    {
        hue = 0.;
    }
    else
    {
        if (Xmax == r){hue = 60. * (0. + (g - b) / chromaRange);}
        if (Xmax == g)
        {hue = 60. * (2. + (b - r) / chromaRange);}
        if (Xmax == b)
        {hue = 60. * (4. + (r - g) / chromaRange);}
    }

    while (hue < 0.)
    {hue += 360.;}
    while (hue >= 360.)
    {hue -= 360.;}

    this->H = hue;
    this->L = (Xmax + Xmin) / 2.;

    if (this->L == 0. || this->L == 1.)
    {this->S = 0;}
    else
    {this->S = chromaRange / (1 - abs(2 * Xmax - chromaRange - 1));}
}
```

A.2 Algorithme d'harmonisation

```
void harmonize(Vec3HSL **tabHSL, int nH, int nW, std::vector<float> bands, float
↪ bandSize)
{
    bool first = true;

    for (int i = 0; i < nW; i++)
    {
        for (int j = 0; j < nH; j++)
        {
            float color = tabHSL[i][j].getH();

            float desiredBand = bands[0];
            float signedDistance = Vec3HSL::distanceHue(color, desiredBand);
            float minDistance = signedDistance;
            if (minDistance < 0.)
            {
                minDistance *= -1;
            }

            float influenceSize = (float)(360. / bands.size());
            float radius = influenceSize / 2.;

            for (size_t i = 1; i < bands.size(); i++)
            {
                float signedDist = Vec3HSL::distanceHue(color, bands[i]);
                float dist = signedDist;
                if (dist < 0.)
                {
                    dist *= -1;
                }
                if (dist < minDistance)
                {
                    signedDistance = signedDist;
                    minDistance = dist;
                    desiredBand = Vec3HSL::fixHue(bands[i]);
                }
            }

            float outColor;

            outColor = (signedDistance / radius) * (bandSize / 2) + desiredBand;

            tabHSL[i][j].setH(outColor);
        }
    }
}
```

A.3 Algorithme K-means (couleur dominante)

```
Vec3Color kMeansDominantColor(Vec3RGB **imageRGB int nbIter, int nbCentroid){
    // Init
    srand(time(0));
    std::vector<CompMean> kMeanVect(nbCentroid);

    // Choose random centroid
    for (int i = 0; i < nbCentroid; ++i){
        kMeanVect[i].setCentroid(imageRGB[rand() % nW][rand() % nH]);
    }

    // Convergence : For each iterations
    for (int nbI = 0; nbI < nbIter; ++nbI){
        // Clear Vectors
        for (int clear = 0; clear < nbCentroid; ++clear){
            kMeanVect[clear].clearEnsList();
        }

        // For each Pixel
        for (int i = 0; i < nW; i++){
            for (int j = 0; j < nH; j++){
                // Dist Min for each CompMean
                Vec3Color pxTarget = imageRGB[i][j];
                int idxComp = 0; double distMin = INT_MAX;
                for (int i = 0; i < nbCentroid; ++i){
                    double d = kMeanVect[i].getCentroid().distVecColor(pxTarget);
                    if (distMin > d){
                        distMin = d;
                        idxComp = i;
                    }
                }
                // Attribution pxTarget
                kMeanVect[idxComp].addEnsList(pxTarget);
            }
        }

        // Compute Centroids (average)
        for (int c = 0; c < nbCentroid; c++){
            kMeanVect[c].computeCentroid();
        }
    }

    // Return the centroid with ensList size max
    int idxMax = 0; int nbEnsMax = kMeanVect[0].getEnsList().size();
    for (int i = 0; i < nbCentroid; i++){
        if (nbEnsMax < kMeanVect[i].getEnsList().size()){
            idxMax = i;
            nbEnsMax = kMeanVect[i].getEnsList().size();
        }
    }
    return kMeanVect[idxMax].getCentroid();
}
```

Bibliographie

[1] Christel Chamaret. Color harmony : experimental and computational modeling. Image Processing [eess.IV]. Université Rennes 1, 2016. English. ffNNT : 2016REN1S015ff. fftel-01382750
https://tel.archives-ouvertes.fr/tel-01382750/document

[2] Palette-based image decomposition, harmonization, and color transfer. 2018.
https://arxiv.org/pdf/1804.01225.pdf

[3] Color Harmonization. 2006.
https://igl.ethz.ch/projects/color-harmonization/harmonization.pdf

[4] Color Harmony for Image Indexing. 2009.
https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.332.7221&rep=rep1&type=pdf

[5] A Geometric Approach to Harmonic Color Palette Design. 2013.
https://arxiv.org/pdf/1709.02252.pdf

[6] Color Scheme Analysis of Illustrations. 2020.
https://profnote.github.io/2020-10-27-color-harmony/

[7] Wikipedia - HSL and HSV
https://en.wikipedia.org/wiki/HSL_and_HSV