MAURIN Christina
ODOR Thibault

Harmonie des Couleurs

Compte Rendu 6

Développement de l'application avec Qt Creator (suite) :

Cette semaine nous avons continué le développement de l'application avec Qt Creator. La semaine précédente, nous avions modifié nos fenêtres de dialogue pour que ce soit plus lisible pour l'utilisateur, nous avons décidé de continuer dans cette lancée et de rendre notre application encore plus fonctionnelle dans plusieurs situations.

Dans un premier temps, nous nous sommes dit que l'utilisateur voudrait probablement avoir la possibilité d'afficher des images autres qu'au format carré mais en paysage ou portrait. A l'origine, il était possible de modifier ces images cependant, l'affichage n'était pas "acceptable". En effet, soit l'image était entièrement visible mais n'était pas centrée (ci-dessous, à gauche), soit elle remplissait tout l'espace mais elle avait un effet comme "zoomé" qui ne permettait donc pas de voir toute l'image (ci-dessous, à droite).

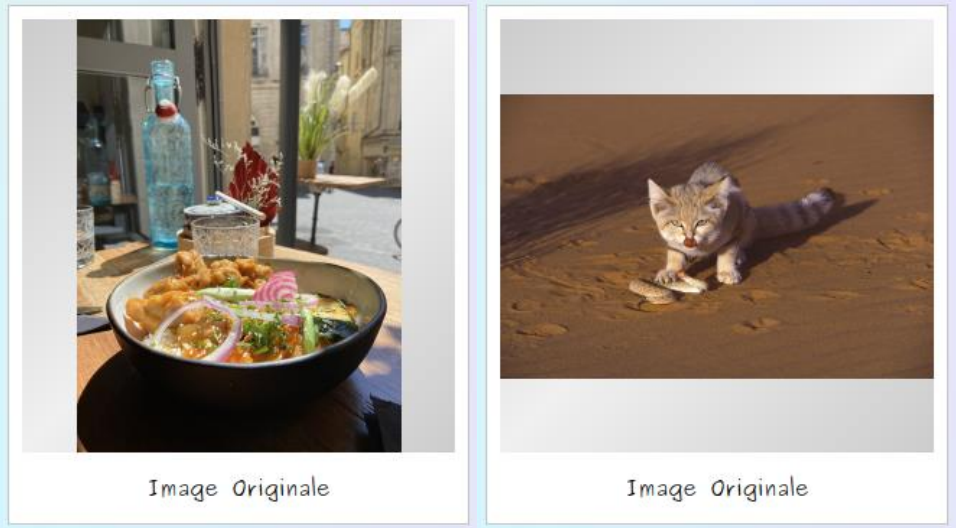
Nous avons modifié des lignes de codes afin de contraindre l'image à la valeur de la taille du label correspondant dans un premier temps, puis, on vérifie si la largeur est supérieure à la longueur (ou l'inverse), afin d'avoir l'image visible entièrement, que ce soit au format portait ou paysage.

```
QImage image(cheminFichier);
    if(image.width()>image.height()){
      image = image.scaledToWidth(ui->init_image_label->width(),Qt::SmoothTransformation);
    }else{
      image = image.scaledToHeight(ui->init_image_label->height(),Qt::SmoothTransformation);
    }
    if(!image.isNull()){
      ui->init_image_label->setFixedSize(ui->init_fond_pola->size());
      ui->init_image_label->setPixmap(QPixmap::fromImage(image));
    }
```

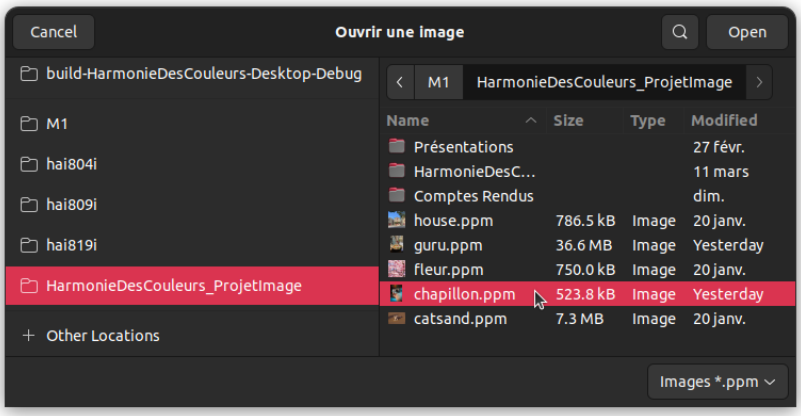
Cela fait, nous avions encore la couleur gris foncé en fond du "polaroid" et cela donnait l'impression à l'utilisateur que ce n'était pas fait pour prendre en compte ces images au ratio autre que 1:1. C'est pourquoi, nous avons donc ajouté un gradient afin de faire un effet "papier glacé" plus agréable à l'œil.

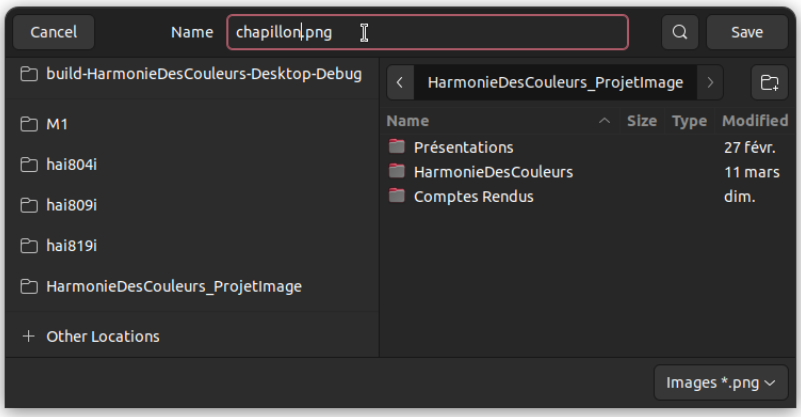
```
ui->init_fond_pola->setStyleSheet(QString("background-color : qlineargradient(spread: pad,
x1:0, y1:0, x2:1, y2:1, stop: 0 rgba(204, 204, 204, 255), stop: 0.5 rgba(240, 240, 240, 255),
stop: 1 rgba(204, 204, 204, 255))"));
```

MAURIN Christina
ODOR Thibault



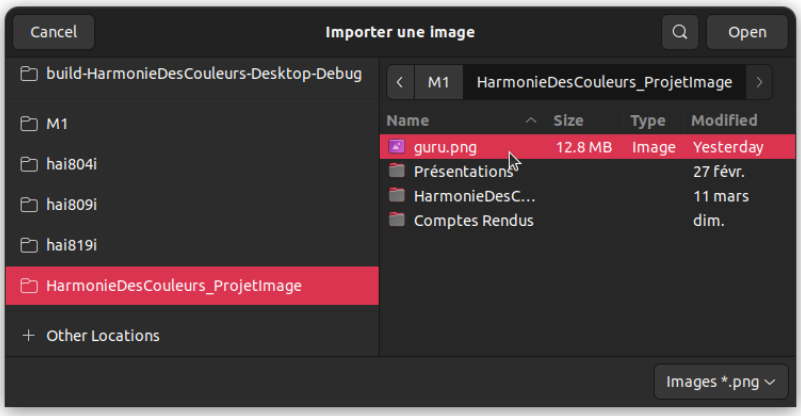
Ensuite, nous avons réfléchi au fait que nous étions peu à utiliser le format ppm pour des images. C'est pourquoi, nous nous sommes penchés sur la question de la conversion de fichiers. Et si nous voulions enregistrer l'image au format PNG ? Qt Creator ne convertit pas les fichiers mais il peut les sauvegarder au format désiré. C'est donc de cette manière que nous avons pu utiliser cette fonctionnalité pour que notre image ppm devienne une image png.





Ce faisant, nous avons également voulu donner la possibilité à l'utilisateur d'importer son propre fichier png. Néanmoins, cela s'est avéré plus difficile car en effet, nous utilisons le format ppm dans nos algorithmes. Nous avons réussi à contourner le problème de la même façon que précédemment en enregistrant le fichier au format ppm dans le dossier tmp. Nous avons donc alors récupéré le nouveau chemin pour travailler sur l'image.

```
cheminFichier = QFileDialog::getOpenFileName(this, tr("Importer une image"),
QDir::homePath(), tr("Images *.png")); // Ouvrir le chemin de l'image de base png
QImage imageOldExt(cheminFichier); // Ouverture image png de base
imageOldExt.save("/tmp/imageOldExt.ppm","ppm"); // Enregistrement en ppm dans tmp
QImage imagePPM("/tmp/imageOldExt.ppm") // Ouverture de la nouvelle image ppm
cheminFichier = "/tmp/imageOldExt.ppm"; // Remplacement du chemin du fichier
```

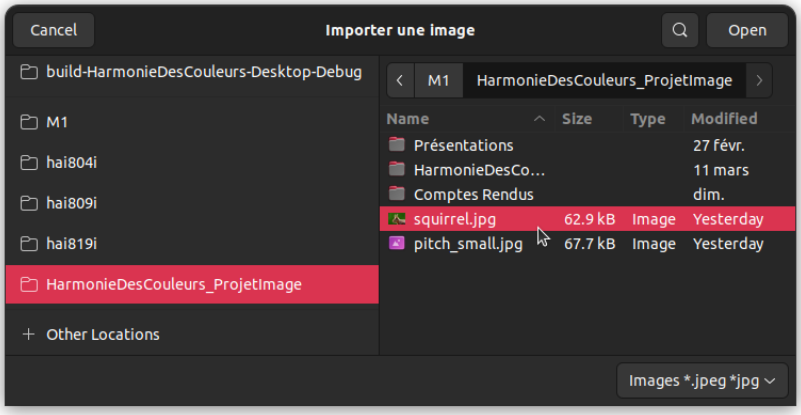


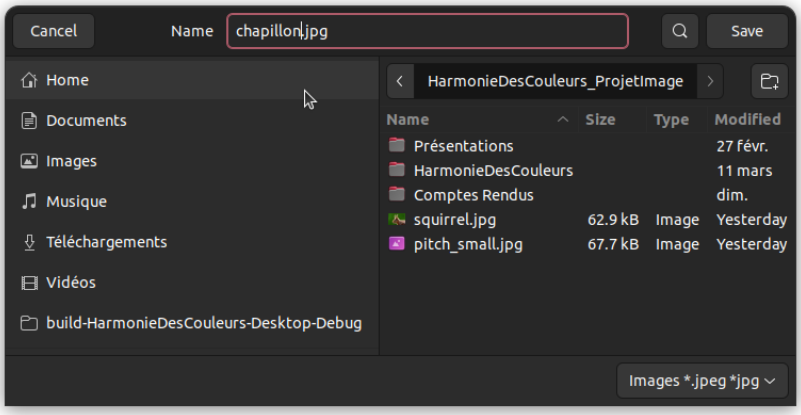
Puisque nous avions réussi à convertir au format PNG, pourquoi ne pas le faire au format JPEG ? Surtout lorsque l'on sait que c'est le format le plus utilisé lorsqu'on prend une photo. Afin de ne pas réécrire plusieurs fois la même fonction en modifiant seulement png par jpg, nous avons modifié les fonctions pour qu'elles soient plus "standardisées". En effet, nous leur avons donné en paramètre un entier qui est utilisé lorsqu'on appuie sur l'intitulé du menu correspondant. De ce fait, nous avons pu facilement avoir notre version jpg.

```
//Exporter boutons
connect(ui->actionExportPNG, &QAction::triggered, this, [=](){
        on_actionExport(1);
    });

connect(ui->actionExportJPEG, &QAction::triggered, this, [=](){
        on_actionExport(2);
    });

//Importer boutons
connect(ui->actionImportPNG, &QAction::triggered, this, [=](){
        on_actionImport(1);
    });

connect(ui->actionImportJPEG, &QAction::triggered, this, [=](){
        on_actionImport(2);
    });
```

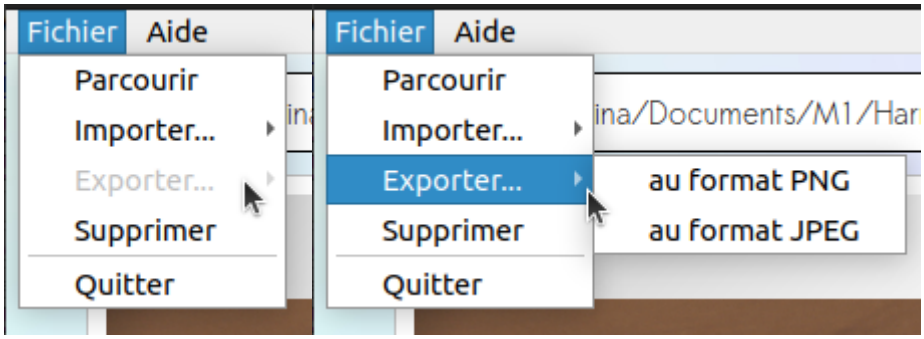
Extrait du début de la fonction ***on_actionImport***(*int format*):

```
/* Utiliser une image png ou jpeg */
void HarmonieDesCouleurs::on_actionImport(int format) //Appui pour choisir un fichier image *.png
{
    QString filter;
    if(format == 1){ //PNG
        filter = tr("Images *.png");
    }else if(format == 2){ //JPEG
        filter = tr("Images *.jpeg *jpg");
    }
    if(!cheminFichier.isEmpty()){
        importeCheminFichier.clear();
        cheminFichier.clear();
        nomFichier.clear();
    }

    cheminFichier = QFileDialog::getOpenFileName(this, tr("Importer une image"), QDir::homePath(), filter);
    qDebug() << "cheminFichier sélectionné : " << cheminFichier;
    importeCheminFichier = cheminFichier;
    qDebug() << "importeCheminFichier sélectionné : " << importeCheminFichier;
    importeFichier = true;

    QFileInfo fileInfo(cheminFichier);
    nomFichier = fileInfo.fileName();
```

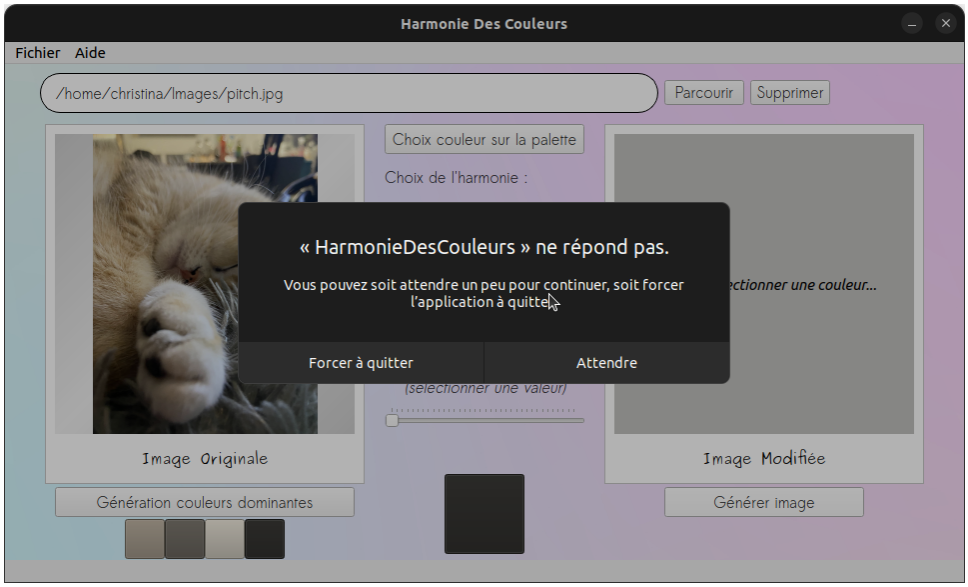




Également, nous avons fait en sorte de ne pas pouvoir appuyer sur “Exporter...” lorsque l’on n’a pas encore généré l’image modifiée. Plus tard, nous avons aussi remarqué que si l’on avait généré une image puis importé une autre, on pouvait encore utiliser “Exporter...”, or, cela aurait exporté la précédente génération. Nous avons donc fait en sorte que cela ne se produise plus.



En ayant toutes les conditions remplies ci-dessus, nous avons donc testé plusieurs images. Nous avons constaté avec effroi qu'une fenêtre de dialogue *"Harmonie des couleurs ne répond pas : forcer à quitter | attendre"* pouvait apparaître plusieurs fois selon la taille et le poids du fichier utilisé. Cela peut inciter alors l'utilisateur à quitter l'application croyant qu'elle ne marche pas alors qu'au contraire, elle prend simplement du temps pour exécuter le code.



Nous avons donc décidé de chercher une solution pour y remédier. Cela impliquait donc deux de nos fonctions : la fonction qui génère les couleurs dominantes et celle qui génère l'image modifiée.

```
QThread* thread = new QThread;
QObject::connect(thread, &QThread::started, [=]() {
        QApplication::setOverrideCursor(Qt::WaitCursor);
        QWidget *mainWindow = QApplication::activeWindow();
        mainWindow->setEnabled(false);

        couleursDominantes(nomImage, couleur1, couleur2, couleur3, couleur4);
        ui->d_color_1->setStyleSheet(QString("background-color: rgb(%1, %2, %3)")
                        .arg((int)couleur1.r).arg((int)couleur1.g).arg((int)couleur1.b));
        ui->d_color_2->setStyleSheet(QString("background-color: rgb(%1, %2, %3)")
                        .arg((int)couleur2.r).arg((int)couleur2.g).arg((int)couleur2.b));
        ui->d_color_3->setStyleSheet(QString("background-color: rgb(%1, %2, %3)")
                        .arg((int)couleur3.r).arg((int)couleur3.g).arg((int)couleur3.b));
        ui->d_color_4->setStyleSheet(QString("background-color: rgb(%1, %2, %3)")
                        .arg((int)couleur4.r).arg((int)couleur4.g).arg((int)couleur4.b));
        genDomColor = true;
        ui->d_color_1->setEnabled(true);
        ui->d_color_2->setEnabled(true);
        ui->d_color_3->setEnabled(true);
        ui->d_color_4->setEnabled(true);

        mainWindow->setEnabled(true);
        QApplication::restoreOverrideCursor();
        thread->quit();
});
thread->start();
```

Exemple d'utilisation de QThread dans notre code

Thread 34 in group i1 exited.

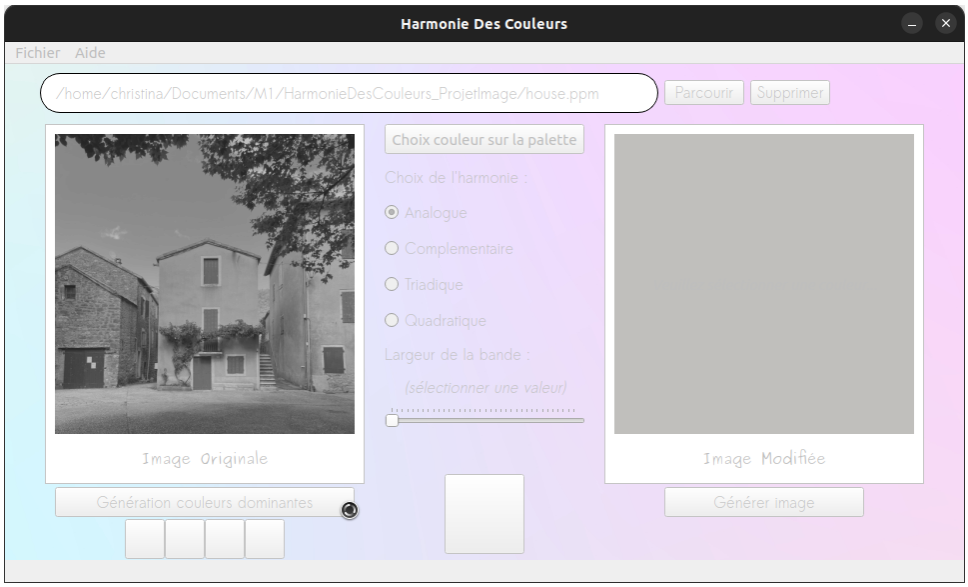
Après plusieurs essais et problèmes de conversions de types, nous avons fini par trouver une solution avec QThread utilisé sous la forme d'une lambda expression. De plus, durant ce moment d'attente, nous avons changé notre curseur sous la forme d'un curseur "wait" afin de faire comprendre à l'utilisateur que le programme est en train de calculer les résultats : QApplication::setOverrideCursor(Qt::WaitCursor).

Après implémentation, nous avions remarqué un bug qui, lorsqu'une image (au format png ou jpeg) est importée deux fois et que l'on appuie sur le bouton permettant de générer les couleurs dominantes, faisait planter l'application. Après avoir essayé à maintes reprises de corriger les bugs en pensant que cela venait du thread généré, nous

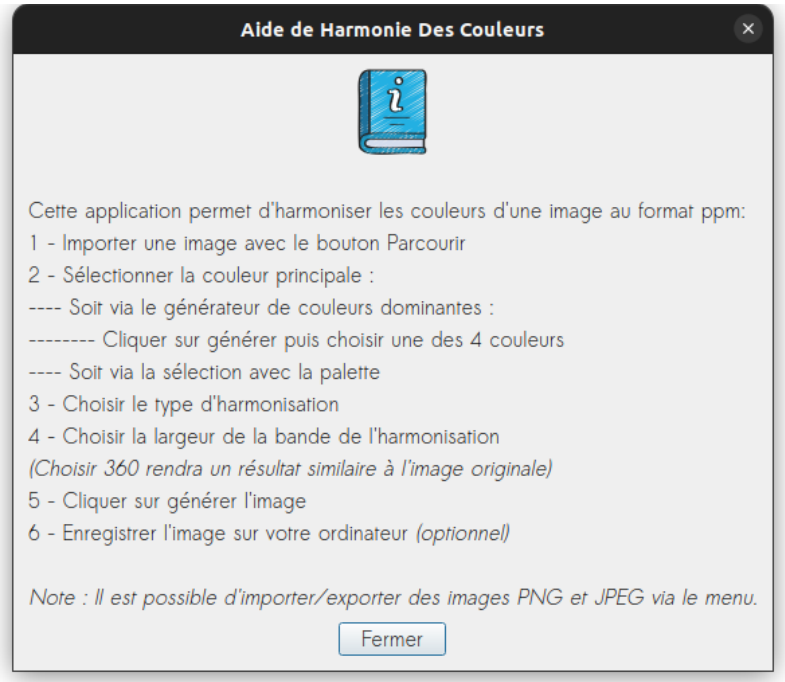
avons finalement réussi à le corriger dans la semaine. Nous avons ainsi ajouté les lignes de code suivantes, placées juste avant l'appel de la fonction *couleursDominantes()*.

```
char nomImageArray[chemin.size() + 1];
strncpy(nomImageArray, nomImage, chemin.size());
nomImageArray[chemin.size()] = '\0';
```

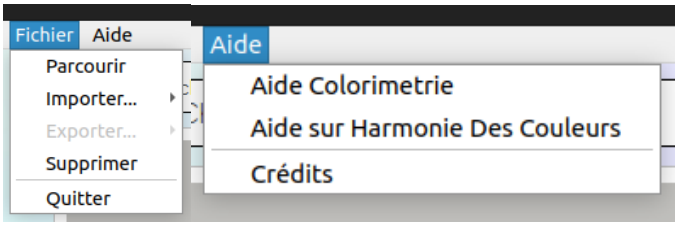
Plus tard, nous nous sommes rendu compte que l'on pouvait toujours interagir avec les autres éléments de l'interface durant les calculs, comme notamment le bouton "supprimer". Nous avons donc rajouté dans notre thread des lignes de code permettant de griser l'interface pour ne pas pouvoir l'utiliser le temps du chargement : *mainWindow->setEnabled(false|true)*.



Nous avons modifié l'aide afin de faire comprendre à l'utilisateur qu'il peut utiliser des images PNG/JPEG dans la barre du menu.



Enfin nous avons renommé Menu en Fichier et déplacé Crédits dans Aide, car c'est ce qui est le plus conventionnel dans les applications.



MAURIN Christina
ODOR Thibault

Algorithme :

Nous avons également grandement amélioré notre algorithme cette semaine.

La première chose que nous avons réussi à mettre en place est l'identification des zones, pour ce faire nous utilisons dans un premier temps une image moyennée à l'aide de Kmean. Ainsi si on reprend notre image « house.ppm » :

Droite : Image de base / Gauche : Image moyennée

Ensuite, on prend un point dans notre image et on itère pour chaque pixel de la même couleur en marquant à chaque fois les points qui ont été visités. Si toute une zone est visitée (si tous les points voisins à notre zone sont d'une couleur différente à celle du point de base) alors on passe au point suivant qui n'a pas déjà été visité. Cette méthode fonctionne sur le principe du BFS.

On peut alors visuellement le représenter sous cette forme, où, pour chaque zone identifiée précédemment, on a affecté une couleur aléatoire :

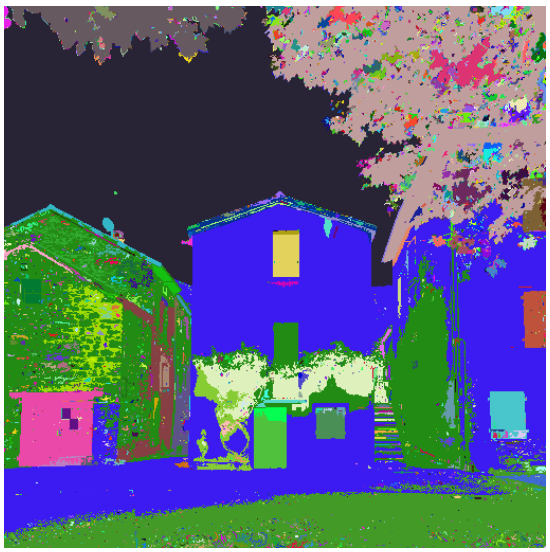


Image segmentée

Ainsi, maintenant que nous avons différentes zones, il est possible d'utiliser une image qui contient des artéfacts comme ci-dessous :

Image harmonisée mais contenant des artefacts

On voit sur cette image que le ciel est "coupé en deux" vers la fin car la couleur n'est pas la même sur l'image originale (bleu/bleu turquoise).

Ensuite, pour chaque zone identifiée, on récupère la couleur la plus présente en fonction de l'image à fixer. On passe la couleur RGB en HSL. Et ainsi on affecte chaque pixel de la zone le même H que la couleur la plus présente.

Image « fixé »

Ainsi, comme on peut voir, le ciel n'est plus "coupé en deux" et les couleurs sont réparties plutôt équitablement.

Cependant, trouvant que l'image n'avait plus l'air vraiment harmonisée, nous avons testé de réappliquer la même harmonisation et obtenons alors un résultat beaucoup plus convaincant :

Image harmonisée puis « fixé »

Ainsi on peut voir que les feuilles paraissent moins jaunâtres, cependant on remarque que les barreaux sur la droite sont un peu étranges car, comme on a pu le voir sur l'image d'identifications des zones les barreaux, les feuilles et la fenêtre du milieu ne forment qu'une zone.

Pour la semaine prochaine :

Pour la semaine prochaine, nous vérifierons le fonctionnement global de l'application (afin qu'elle soit au point pour la future présentation vidéo) et les éventuels bugs que nous n'aurions pas discernés auparavant.

Également, avec les recherches algorithmiques que nous avons effectuées cette semaine, il nous faudrait ajouter un bouton qui permet à l'utilisateur, s'il le souhaite, de « fixer » l'image (avec la méthode vue plus haut) afin de réduire au maximum les artéfacts.

Enfin, comme on l'a vu précédemment, notre méthode permettant de fixer les artefacts possède encore quelques soucis. De ce fait, il nous faudrait tester de nouvelles approches comme l'utilisation de la couleur moyenne ou la variance à la place de la couleur dominante.

MAURIN Christina
ODOR Thibault

Sources :

https://www.techniques-de-peintre.fr/theorie-des-couleurs/harmonie-coloree/

https://en.wikipedia.org/wiki/HSL_and_HSV

https://doc.qt.io/

https://en.wikipedia.org/wiki/Graph_cuts_in_computer_vision

https://en.wikipedia.org/wiki/Breadth-first_search