ColoReduce

Figure 1: l'image de gauche est transformée par réduction à 4 couleurs pour donner le résultat visible à droite

1. Introduction

Le but de ce projet est de modifier une image (Fig 1) en réduisant l'ensemble des couleurs à un petit nombre prédéfini dans une table lue en entrée. Pour atteindre ce but l'intensité de chaque pixel de l'image source subit un *seuillage* pour coder sa couleur avec l'une des couleurs de la table prédéfinie, suivi d'un ou de plusieurs *filtrages* (Fig 2). Nous utiliserons le format d'image **ppm** en entrée et en sortie car il est simple à exploiter pour notre projet.

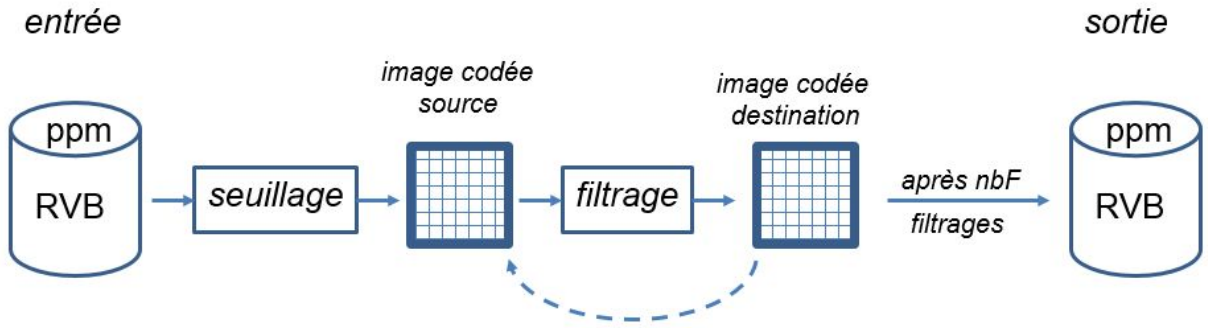


Fig. 2 : vue d'ensemble des transformations appliquées à une image dans ce projet. Le format **ppm** représente une image couleur avec un triplet Rouge-Vert-Bleu pour chaque pixel. L'image codée contient un seul entier pour chaque pixel ; cet entier représente l'indice d'une des couleurs de la table des couleurs fournie au programme. Après le nombre de filtrages demandé, le programme affiche l'image finale en format PPM dans le terminal.

2. Spécifications

2.1 Le format PPM [source : http://fr.wikipedia.org/wiki/Portable_pixmap]:

Le format PPM contient la description d'une image couleur. Chaque pixel est codé par 3 valeurs entières (rouge, vert, bleu) comprises entre 0 et une valeur maximum **MAX** qui pour ce projet sera toujours **255**. La Figure 3 montre la structure d'un fichier PPM pour une image de 3 colonnes x 2 lignes:

- Les 2 caractères « P3 » qui indiquent qu'il s'agit d'un fichier de format PPM.
- Taille de l'image en nombre de colonnes **nbC** et en nombres de lignes **nbL**.

- Valeur entière d'intensité maximale possible pour chaque couleur **MAX**.
- Valeurs entières des couleurs des pixels, en commençant par la ligne du haut de l'image. Si l'image est grande une ligne de l'image est répartie sur plusieurs lignes du fichier.

```
P3
3 2
255
255   0   0     0 255   0     0   0 255
255 255   0   255 255 255     0   0   0
```

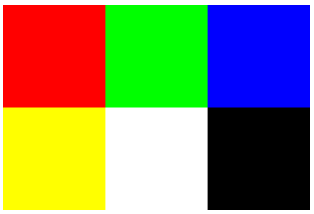
Image correspondante (agrandie) : 

Fig. 3 : exemple de fichier PPM [source : http://fr.wikipedia.org/wiki/Portable_pixmap]

2.2 Informations à fournir au programme en plus de l'image en format PPM

La figure 4 donne un exemple complet des informations à fournir en entrée. Le programme lit **le nombre de couleur**, puis chacune des **couleurs utilisées pour le filtrage** (détails 2.3)**,** puis la liste **des seuils** définissant la transformation (détails 2.4), ensuite vient le **nombre de filtrages** (détails 2.5 et 2.6), et enfin les informations d'une image en format PPM (détails 2.1). Des espaces, tabulations et retours à la ligne peuvent être rajouté pour plus de lisibilité.

```
3
25 25 25   25 50 255    255 255 255
0.3 0.7
2
P3
3 2
255
255   0   0     0 255   0     0   0 255
255 255   0   255 255 255     0   0   0
```

Fig 4 : on fournit d'abord le nombre réduit de couleurs (3) puis les valeurs des composantes RVB des 3 couleurs, suivi par les seuils (0.3 et 0.7) dont le nombre est déduit du nombre de couleurs, le nombre de filtrages (2), et enfin l'image en format PPM.

2.3 Table des couleurs

L'utilisateur indique le nombre **R**éduit de Couleurs **nbR** (minimum:**2**, maximum:**255**). Chaque couleur est codée par 3 valeurs entières entre 0 et MAX pour rouge , vert et bleu (Fig 4 ; 2ième ligne).

Le programme doit les mémoriser dans une table des couleurs de taille **nbR+1** car la première couleur de cette table (pour l'indice de couleur **0)** doit OBLIGATOIREMENT être la **couleur noire**

de valeur (0 0 0) qui est toujours utilisée par le programme pour créer des contours dans l'étape de *filtrage* (2.6).

De ce fait, les couleurs définies par l'utilisateur en entrée seront associées à des indices compris entre **1** et **nbR**. Dans l'exemple de la Figure 4 la couleur (25 25 25) sera rangée à l'indice 1 de la table et correspond au code 1 utilisé à l'étape du seuillage et ainsi de suite : la couleur suivante (25 50 255) correspond au code 2 et la couleur (255 255 255) au code 3.

2.4 Table des seuils

Après la table des couleurs, on doit ensuite fournir (**`nbR-1`**) seuils intermédiaires dans l'intervalle **] 0. , 1. [** (Fig 4 ; 3ième ligne). Les seuils sont rangés dans une table des seuils de taille **`nbR+1`** avec les conventions suivantes qui seront exploitée à l'étape du seuillage (2.6.1):

- Le seuil rangé à l'indice 0 vaut toujours 0. (seuil implicite).
- Le seuil rangé à l'indice nbR vaut toujours 1. (seuil implicite).

Les valeurs explicitement fournies pour les seuils intermédiaires occuperont les indices compris entre **1** et **`nbR-1`**. Ces valeurs doivent être distinctes et croissantes.

Remarque sur le test d'*égalité* des seuils: Deux seuils sont considérés comme distincts si l'écart **`delta_seuil`** entre leurs valeurs est supérieur ou égal à une tolérance de type double : EPSILON= **`0.001`**. Sachant cela, l'expression C++ qui détecte l'erreur sur les seuils doit être :

```
if(delta_seuil < 0.001) alors affichage d'un message d'erreur
```

2.5 Nombre de passages du filtrage (détails en 2.6**)**

Ensuite, un entier positif **`nbF`** indique le nombre de fois que le *filtrage* est appliqué.

2.6 Codage des pixels de l'image fournie en entrée

La transformation des couleurs s'effectue en deux étapes de *seuillage* et de *filtrage* (Fig 2).

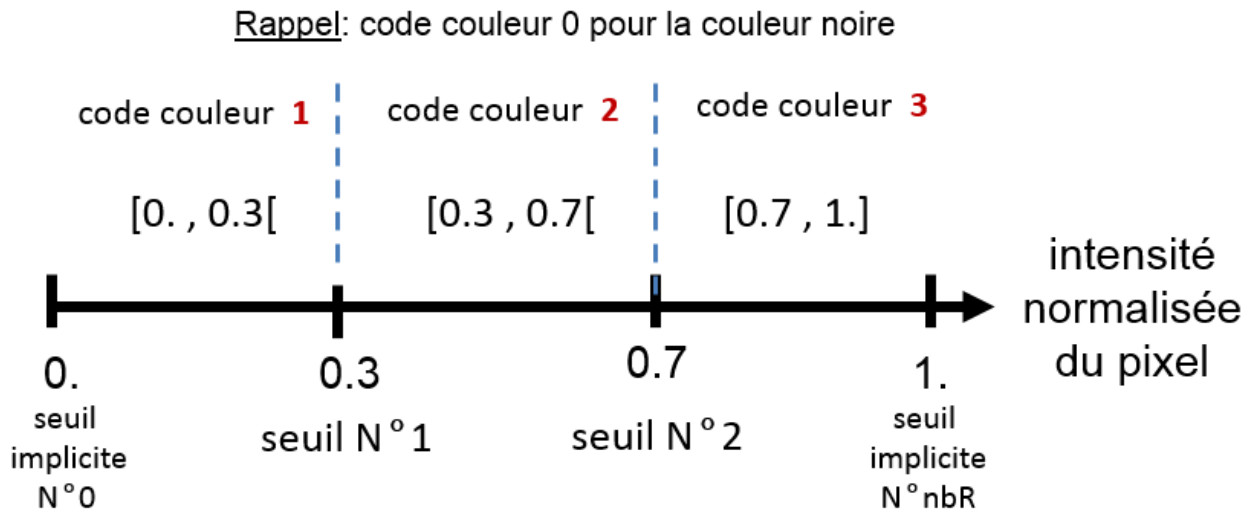


Fig 5 : détermination du code de couleur initial d'un pixel parmi la table des couleurs par comparaison de son intensité normalisée avec les seuils de la table des seuils.

2.6.1 Seuillage

Dans la première étape de **seuillage** (Fig 2) le programme initialise, au moment de la lecture de chaque pixel, ***l'indice entier utilisé comme code de couleur*** pour ce pixel : (Fig 5)

- Calcul de l'intensité normalisée I_N de chaque pixel : $I_N = \sqrt{R^2 + V^2 + B^2}/(\sqrt{3}\,MAX)$
  Où R, V, B désignent l'intensité entière du pixel et MAX la valeur maximum (section 2.1).

- Le **code de couleur** du pixel d'intensité I_N vaut **i** si :

$$\textbf{seuil N° i-1} \leq I_N < \textbf{seuil N° i}$$

Exception: le seuil max N° nbR est inclus dans l'intervalle associé au code de couleur **nbR**

2.6.2 Filtrage

La seconde étape est le **filtrage** qui travaille directement sur l'unique indice entier du code de couleur de chaque pixel de l'*image source,* pour produire une nouvelle *image destination* (Fig 2). Pour chaque pixel, on examine le code de couleur de ses ***8 pixels voisins*** dans l'*image source* (Fig 6) pour déterminer sa valeur dans l'*image destination*.

Règles du filtrage :

- Si au moins **6** des pixels voisins ont le même code de couleur **X,** alors le pixel examiné prends la valeur de code **X** dans l'*image destination* (Fig 6a et 6c). Ceci est indépendant de la valeur initiale du pixel.

- Sinon le pixel examiné prend le code **0** (couleur noire) dans l'*image destination* (Fig 6b, 6d)

Le filtrage est appliqué **nbF** fois ; chaque nouveau filtrage s'applique sur l'*image destination* obtenue au filtrage précédent (Fig 2).

cas particulier : ce traitement ne s'applique pas aux pixels du bord de l'image qui prendront tous la couleur noire, d'indice 0, dans l'*image destination*.

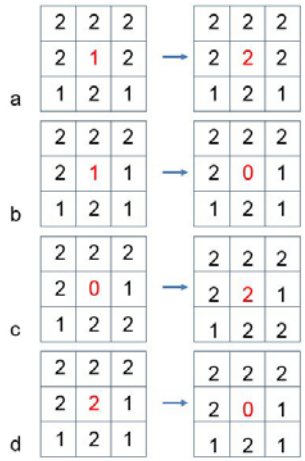


Fig 6 : transformation du pixel central, indiqué en rouge, de l'image codée *source* (gauche) vers l'image *destination* (droite)

2.7 Fin du programme

Après les nbF filtrages, l'image destination est affichée dans le terminale en format PPM. Il est autorisé d'avoir nbF nul ; cela permet de vérifier le résultat du seuillage (seul) en format PPM.

3. Mise en œuvre en langage C++

Nous mettons à disposition un fichier source à compléter nommé **projet19_proto.cc** ; il contient une fonction **main()** et un ensemble de quatre fonctions préfixées par « `error_` » *qu'il faudra utiliser pour l'affichage des messages d'erreurs* sur nbR, une valeur de couleur ou de seuil, et nbF.

3.1 Clarté et structuration du code avec des fonctions

La clarté de votre code sera prise en compte pour le rendu. Nous vous demandons de respecter les conventions de programmation (disponible en PDF sur le moodle de ce cours).
Dans tous les cas, l'affichage des messages d'erreurs doit seulement être fait avec les fonctions fournies, sans les modifier.

3.2 Mémorisation et structuration des données

Pour la mémorisation de données, nous vous recommandons d'utiliser des **vector** pour représenter des tableaux à un ou deux indices. Par exemple :

- **`vector<double>`** pour la table des seuils
- **`vector<vector<int>>`** pour l'image codée source ou destination

Cet outil de structuration des données convient parfaitement pour transmettre, ajouter et éventuellement modifier des données dans des fonctions.

3.3 Variables locales ou globales ?

Toutes les variables ou tableaux utilisés pour ce projet seront déclarés ***localement*** et transmis en paramètres aux fonctions **seulement** si c'est nécessaire. Aucune exception ne sera admise.

3.4 Précision sur les nombres à virgule

La totalité de ce projet doit utiliser le type **`double`** pour les nombres à virgule. C'est le cas du programme de démo avec lequel votre programme sera comparé du point de vue de l'image produite en sortie. Il est donc important de travailler avec la même précision (cela exclu donc le type `float` mais aussi le `long double`, etc).

3.5 Redirection des entrées-sorties

Ce mécanisme simple de *redirection* vu en cours/série en semaine 6 permet un test très efficace du projet sans changer une seule ligne de code. On peut également rediriger la sortie standard (l'affichage) vers un fichier texte. C'est très utile pour récupérer facilement l'image finale pour la visualiser avec d'autres outils (série de la semaine 5). Le projet sera évalué de cette manière.

3.6 Fichiers test :

Construire un ensemble de scénarios de tests d'un programme, pour lesquels on connaît la solution, est très important pour valider votre programme. Nous le faisons partiellement pour vous en vous fournissant quelques fichiers de tests. A vous de compléter avec vos propres fichiers plus élaborés ou juste différents.
Vos trouvez dans le dossier *tests* deux dossiers : *Elementary* et *Advanced*. Le premier contient 4 fichiers tests pour la détection d'erreur et deux fichiers tests corrects et très simples. Le second répertoire contient des cas plus élaborés (nb de couleurs, de filtrage, taille image).

4. Rendus :

Il faudra fournir :

1) **Votre code source imprimé** avec une mise en page portrait, police de caractères utilisée par défaut par **geany**, ou Courrier New de taille 10. Votre objectif est de ne pas avoir de passages à la ligne parasites (wrapping) ; il est autorisé d'avoir 87 caractères au maximum

par ligne. Vous devez indenter le code à l'intérieur de chaque fonction ainsi que le code contrôlé par les instructions de contrôle. Vous pouvez vérifier votre mise en page avec la commande Print Preview de geany.

2) **Votre code source devra être téléchargé (upload)** à l'aide du lien qui sera mis à disposition sur **moodle** (Topic 12). Vous êtes responsables de vérifier que l'upload s'est bien passé en le téléchargeant (download) dans votre compte et en examinant que tout est bien présent. Vous pouvez toujours faire un nouvel upload jusqu'à la date limite.

3) **Un rapport imprimé** d'au maximum une feuille de papier A4 recto-verso (2 pages) écrit à l'ordinateur. Ce rapport doit être amené dans les casiers devant la salle INJ 141.

Vous devez avoir téléchargé votre code pour le **dimanche 9 décembre à 23h59** et livré votre code imprimé avec votre rapport (avec votre prénom et nom) avant le **lundi 10 décembre à midi**.

4.1 Rapport

Le Rapport ne contient PAS de page de titre, ni de table des matières
Le Rapport contient :

a) Résultat de la phase **d'analyse** (max 1 page, police de taille 11) :
Décrire l'organisation générale du programme en faisant ressortir la mise en oeuvre des principes *d'abstraction* et de *ré-utilisation*.

b) Pseudocode de votre **algorithme** pour **l'étape de filtrage** décrite dans la section **2.6.2**. (pas de code source)

c) Décrire dans un paragraphe l'ordre de complexité **O(…)** de **cet algorithme** en fonction de la taille de l'image **nbL** x **nbC** et éventuellement d'autres paramètres si c'est important dans votre approche.

4.1 Barème indicatif (12pts):

Ce barème est indicatif et provisoire. Il sera éventuellement modifié par la suite.

(2pt) rapport : description de la mise en œuvre du principe d'abstraction et de réutilisation, analyse , pseudocode du filtrage et coût calcul.
(4pt) Lisibilité, structuration du code et conventions de programmation du cours
(2pt) votre programme fonctionne correctement avec les fichiers du dossier *elementary*
(2,5pt) votre programme fonctionne correctement avec les fichiers du dossier *advanced* (timemout avec un temps d'exécution 2 fois plus long que celui de la version de démonstration)
(1,5pt) fichiers de tests qui seront fournis rendus publics après le rendu (timemout avec un temps d'exécution 2 fois plus long que celui de la version de démonstration)

Dernier rappel : le projet d'automne est INDIVIDUEL. Le détecteur de plagiat sera utilisé selon les recommandations du SAC.