Travaux Pratiques en Traitement d'Images

Tawsif Gokhool
École d'ingénieurs 3iL

February 23, 2021

1 TP 1 - La prise en main de la maquette du logiciel traitement d'images

Ce rapport de TP décrit la mise en route ainsi que la structuration du projet utilisée pour l'implémentation des algorithmes en traitement d'images discuté dans le cours.

Le nom d'utilisateur ainsi que le mot de passe est: user
Les dépendances du projet sont:

- QT 5.X- l'IDE fourni
- OpenCV 3.14 - bibliothèque de traitement d'images

1.1 Le répertoire du projet

Le projet se trouve dans: `/home/user/TIL2021/TP_I2_Etud/`. Ouvrez un terminale et naviguez en ligne de commande dans le répertoire en tapant: `cd TIL2021/TP_I2_Etud/` . Ou tout simplement, ouvrez le navigateur de fichier et parcourez l'arborescence.

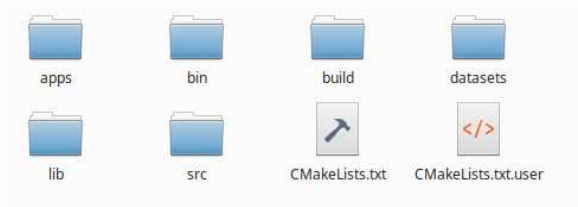


Figure 1: contenu du répertoire

- apps - contient les fichiers exemples de lancement des algorithmes
- bin - contient les exécutables
- build - répertoire de compilation du projet
- datasets - les jeux de données à utiliser
- lib - contient l'extension *.a de la bibliothèque statique
- src - la source de notre bibliothèque statique contenant le fichiers *.cpp et *h
- CMakeLists.txt - la programmation en cmake de toute la structure (A ne pas manipuler sauf instruction)

1.2 Importation du Projet

Afin d'ouvrir le projet sur notre IDE de Qt, nous utilisons le logiciel `Qt creator`. Pour importer le projet, on lance d'abord `Qt creator` soit dans un terminale avec la commande `qtcreator &` ou dans l'explorateur d'application avec l'icône "loupe" se trouvant en bas du menubar.

Figure 2: Le menubar

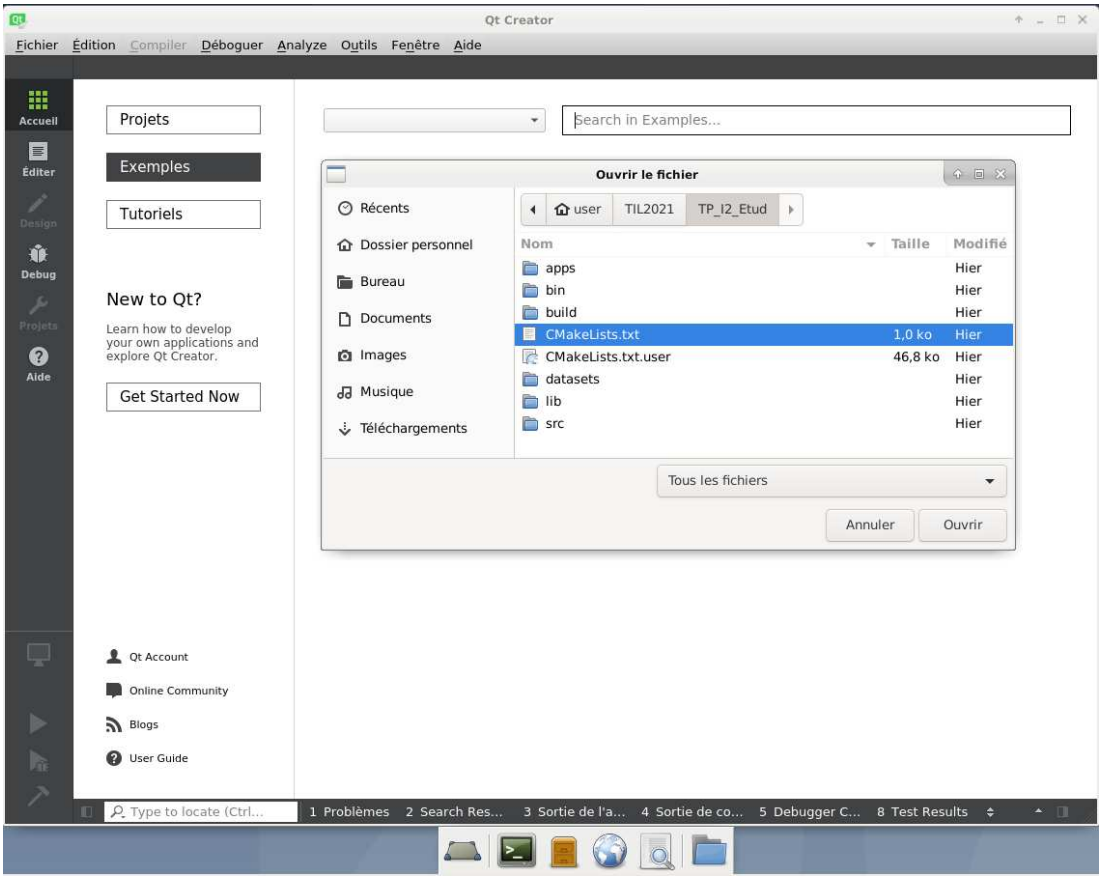


Figure 3: Ouvrir un projet sous l'IDE de QT

1. Click sur ouvrir. Le compilateur de QT compile le projet à son compte et ouvre le projet `IMAGE_PROC_TP`

2. Click sur le projet pour le dérouler. On aperçoit maintenant l'arborescence de tous les composants du projet

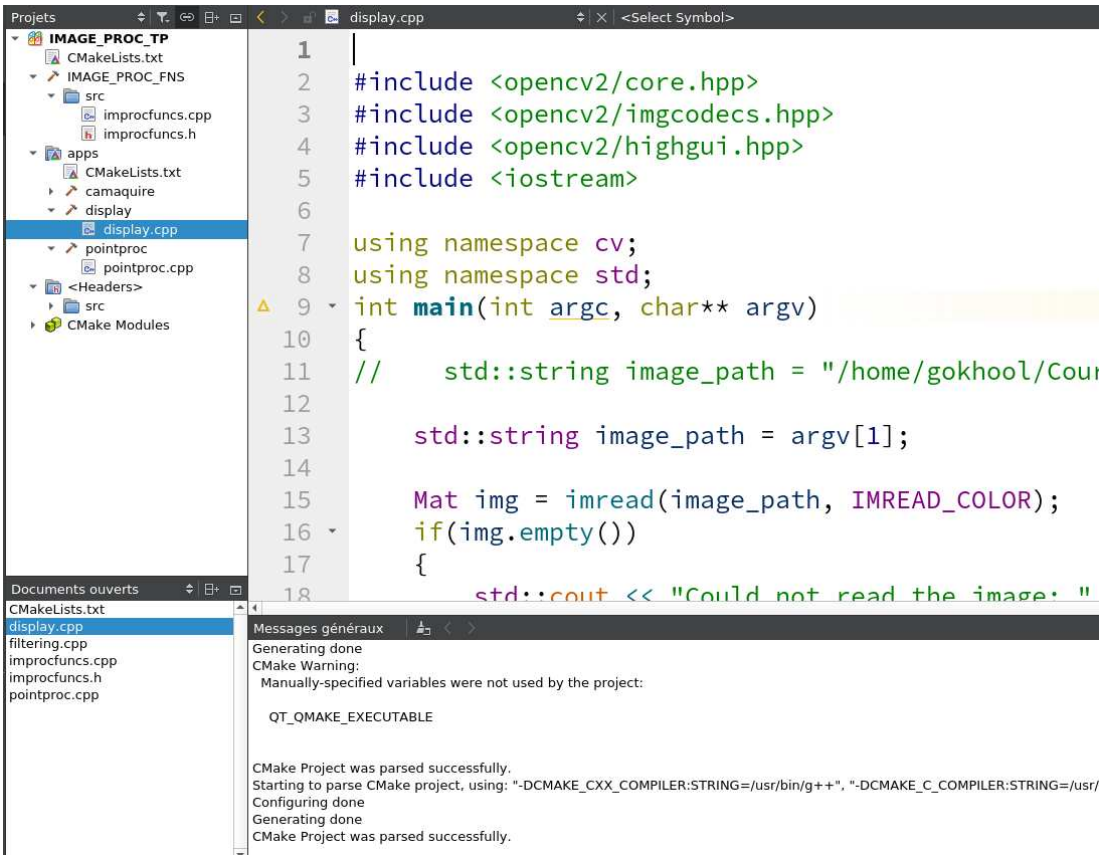


Figure 4: Aperçu de l'arborescence du projet

1.3 Compilation et mise en route

Dans le répertoire du projet, `/TIL2021/TP_I2_Etud/`, la compilation, se fait comme suite:

1. mkdir build // créez le répertoire build si seulement ce n'est pas fait
2. cd build // pointez vers le répertoire build directory
3. cmake ../ // compilez les librairies et les dépendences avec cmake
4. make -j7 // compilez le code

Les executables se trouvent dans `/TIL2021/TP_I2_Etud/bin` et un exécutable en particulier est lancé du terminale comme suit: `./display <glisser une image>` . C'est utile de lancer deux terminales côte à côte afin de voir la compilation et ensuite lancer les éxecutables du répertoire `/bin` comme nous démontre la figure suivante. Les images se trouvent dans le répertoire `/TIL2021/TP_I2_Etud/bin`

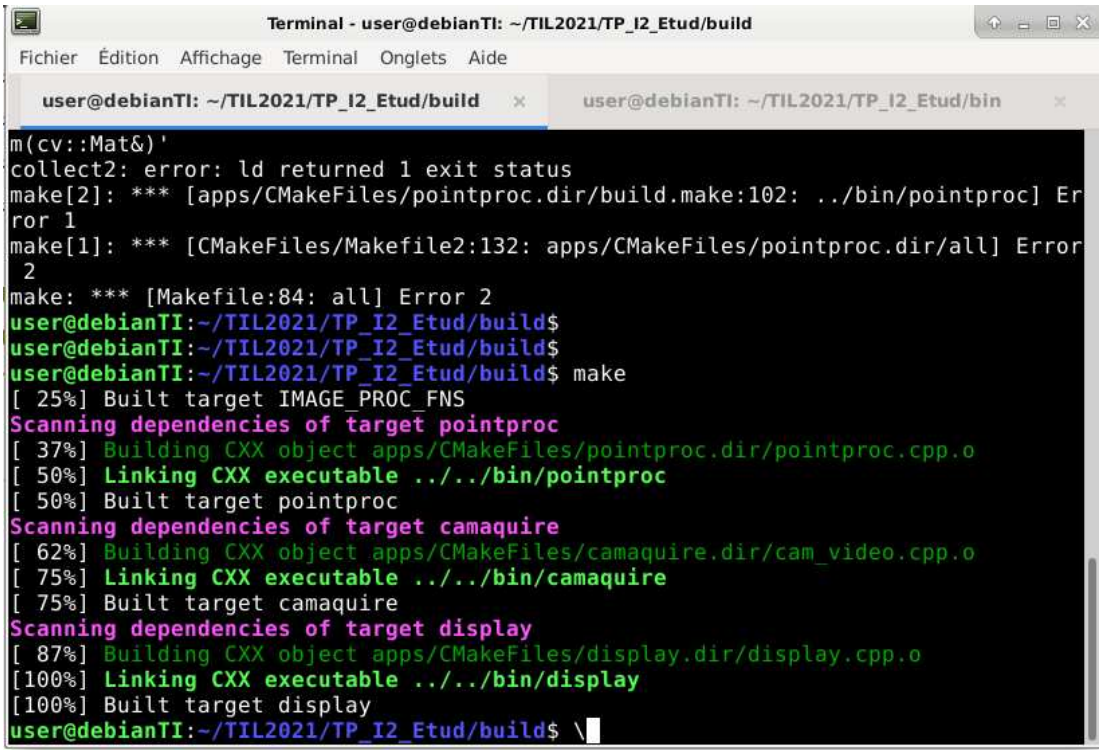


Figure 5: lancement de deux terminales côte à côte par la commande `Shift+ctlr+T`

1.3.1 Contenu des sous répertoires

1. Le répertoire `src` contient les fichiers:
   (a) `improcfuncs.h` contenant les signatures des fonctions de la bibliothèque qu'on développera dans ce cours
   (b) `improcfuncs.cpp` contenant l'implémentation des fonctions déclarées dans (a).
2. Le répertoire `apps` contient des exemples d'appels de la bibliothèque en occurence:
   (a) `cam_video.cpp` : une application de lancement de la webcam de votre ordinateur

(b) `display.cpp` : une application pour le chargement et ensuite l'affichage d'une image

(c) `pointproc.cpp` : une application d'exemple d'accès des pixels d'une image en opencv

(d) `CMakeLists.txt` : l'inclusion des applications en dessus dans le projet et la génération des éxecutables

1.4 Compte rendu

Un rapport complet et détaillé vous sera exigé à la fin de ce cours, soit **la semaine 10**. Cependant, la progression doit être graduelle et fluide tout au long de ces 5 scéances de TP. Ce dernier décrira tous les exercices qui vous seront confiés au cours des 5 scéances. Chaque approche doit être élaboré suivi de son implémentation algorithmique et enfin l'analyses de diverses résultats obtenus. Les notes vont être réparties (sur la note finale) comme suite:

- 3 points pour les algorithmes (maquette finale du projet en C++ incluant la compilation ludique de tous les algos. TOUT DOIT MARCHER!)
- 3 points pour la rédaction du rapport
- des points bonus pour un algorithme de votre choix qui n'a pas été abordé dans ce cours (à définir).

2 TP2 - Transformation de l'intensité dans le domaine spatiale

2.1 Introduction

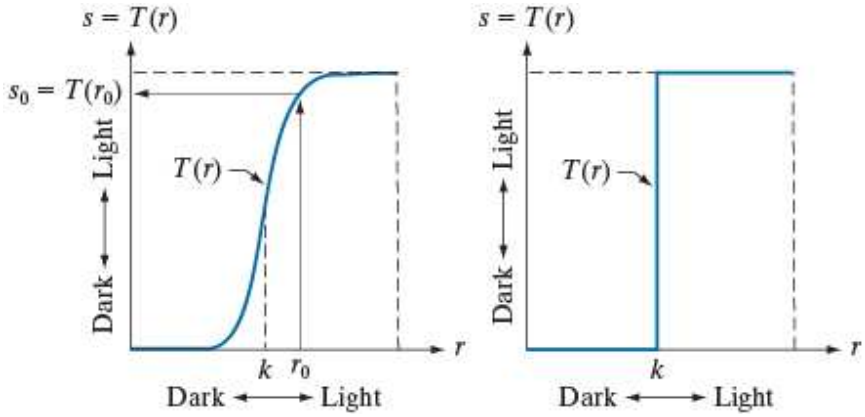


Figure 6: Fonctions de transformation d'intensité (a) l'extension dynamique (contrast stretching)(b) seuillage (Thresholding)

La transformation de l'intensité dans le domaine spatiale est définie par

$$s = T(r), \tag{1}$$

d'où $T(\cdot)$ est une fonction strictement croissante et peu prendre la forme de la figure 6(a) par exemple. l'intérêt principale de ce genre de transformation c'est d'améliorer l'intensité, $\boldsymbol{\mathcal{I}}$, donnée par la sortie s, afin de la rentre plus réelle. Ce correcteur intervient suite à des "déformations" subi par exemple par la sortie obtenue d'un appareil photographique suite à un éclairage trop faible de la scène en question ou un temps de pose insuffisant.

En analysant la courbe de la figure 6(a), on en ressort qu'une petite bande de valeur en niveau de gris dans l'image, sur l'axe r peut être transformée en une bande plus étendue résultant à une éclaircissement (ou assombrissement) des niveaux de gris de l'image sur une certaine zone pré-définie. Intuitivement, l'étendue de la bande peut être lu sur l'axe s de la figure. Cette approche est communément connu sous ***l'extension dynamique*** tout en permettant d'améliorer le contraste de l'image; ou *contrast stretching* dans la littérature anglaise. On peut désormais formuler mathématiquement cette approche comme suite:

$$\boldsymbol{\mathcal{I}}' = \frac{\boldsymbol{\mathcal{I}}(x,y) - G_{\min}}{G_{\max} - G_{\min}} \times 255, \tag{2}$$

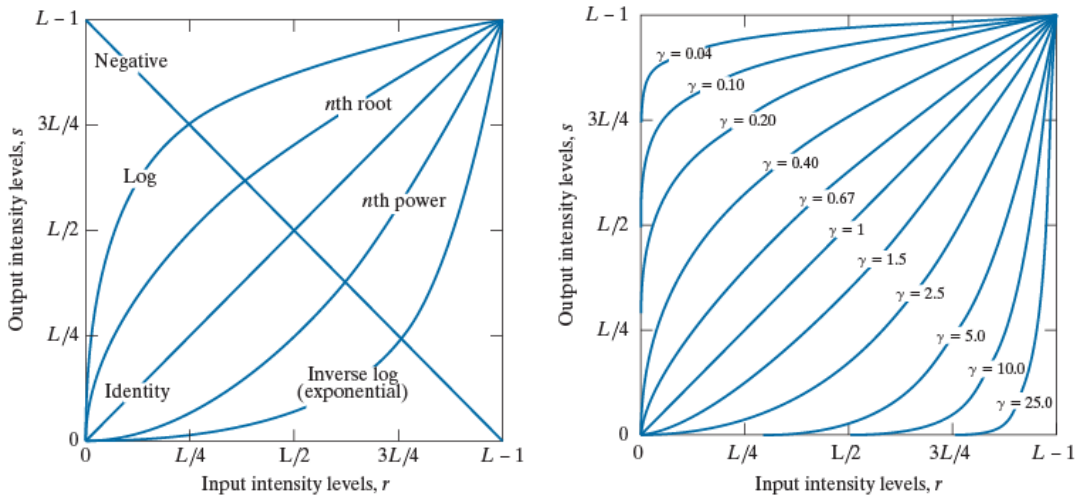


Table 1: transformés de (a)log et (b) de gamma

où $G_{\max}$ correspond à r_0 et $G_{\min} = k$ dans la figure 6(a).

Par la suite, on utilisera, le même raisonnement pour la transformation de `log` et de `gamma`, γ, comme nous illustre les figures du tableau 1.

Pour le transformé de log,

$$s = 255 \times \texttt{log}(r/\lambda) \tag{3}$$

d'où λ est un facteur d'échelle donné par $\lambda = \texttt{log}(255)$

Similairement, le transformé exponentiel `exp` est donnée par:

$$s = 255 \times \texttt{exp}(r/\lambda) \tag{4}$$

A noter que les facteurs d'échelles sont toujours variable selon l'application en cause pour avoir un résultat satisfaisant. Pour ce TP, nous choisissons $\lambda = 255/10000$ par exemple.

Et, le transformé de γ *(power-law)* est donnée par:

$$s = cr^{\gamma} \tag{5}$$

Pour ce TP, nous choisissons $c = 1$ par exemple.

Par la suite, nous décrirons les étapes à suivre pour implémenter toutes ces équations. En occurrence, un guide pratique étape par étape sur le transformé de `log` vous sera introduit et le reste des équations sont à votre charge à titre d'exercices et **vous seront exigés dans le rapport finale**.

2.2 Example d'implémentation

Ici un exemple d'implémentation d'une fonction vous êtes instruit. Pour la suite d'autres implémentations, on utilisera le même cheminement.

On commence par un exemple simple du transformé de `log`. La signature du fonction insérée dans le fichier header `src/improcfuncs.h` est donnée par:

```
void log_transform(const cv::Mat &image);
```

La fonction dessus inclut une entré **par référence** de la structure matricielle de l'image en `opencv`, et le type de retour est `void`, soit, nul. Donc le syntaxe générale d'une fonction est donné comme suit:

```
type_de_retour nom_de_la_fonction(type_1 entrée_1, ··· type_n entrée_n);
```

On passe ensuite par l'implémentation de cette fonction à insérer dans `src/improcfuncs.cpp`.

```
1     void ip::log_transform(const cv::Mat &image)
2 {
3     int value = 0; // declaration d'une valeur entiere
4
5     // on declare le facteur d'echelle en double
6     double scale = log(255.0);
7
8     //Inilialization du tableau de correspondence (Look up table)
9     std::vector<float> lut(256);  // utilisation de la bibliotheque standard
10    lut[0] = 0; // allocation de la premi re cellule
11
12    // Remplissement du tableau de correspondence
13    for (int i = 1; i<256; i++)
14        lut[i] = (int) (255*log( i/scale));
15
16    // declaration d'une matrice type cv::Mat
17    // le constructeur prend en entre: taille en row, taille en colonne,
18    //type de matrice, initialisation    ZERO
19    cv::Mat imglog(image.rows, image.cols, CV_8UC1, Scalar(0));
20
21    // On parcours la structure matricielle en 2d pour chercher
22    //la correspondence du valeur en niveau de gris
23        for(int i = 0 ; i< image.rows; i++)
24            for(int j = 0 ; j< image.cols; j++)
25            {
26                // acquisition de la valeur de l'image en niveau de gris
27                 value = image.at<uchar>(i,j);
28
29                 // attribution de la valeur dans la nouvelle matrice imglog
30                 //donnee par le lut
```

```
                imglog.at <uchar>(i,j) = (int) lut[value];
            }

        // Affichage de la matrice, soit l'image en opencv
        cv::namedWindow("image log", 1);
        imshow("image_log", imglog );
        waitKey(0); // on attend un retour clavier
} // fin de la fonction

```

Lisez et comprenez attentivement le code développé en dessus. L'explication donnée en commentaires est "self sustained".

Ensuite, on procède à l'appel de cette fonctionalité dans le `main` du programme. Afin de pouvoir faire appel à plusieurs <`main`>, on les structure dans un répertoire que `CMake` va gérer. Sans cette structuration, le compilateur nous donnera des erreurs. Ce répertoire est nommé `apps` dans notre cas. C'est ici que l'appel de la fonction s'éxecutera. Si on regarde le fichier `pointproc.cpp` en particulier, on note le développement du `main` ici.

```
    // declaration des headers d'opencv
    #include <opencv2/opencv.hpp>
    #include <opencv2/imgproc/imgproc.hpp>
    #include <opencv2/objdetect/objdetect.hpp>
    #include <opencv2/highgui/highgui.hpp>

    // declaration du header pour entre/sortie
    #include "iostream"
    // inclusion de la librairie statique qu'on developpera dans ce cours
    #include "improcfuncs.h"
    using namespace cv;
    using namespace std;
    int main(int argc, char** argv) {
        // on prend en argument, la chaine de charactere
        // entiere que represente le chemin + nom.extension
        // (image.tiff) de l'image
        std::string image_path = argv[1];
        // lecture et allocation d'une image
        cv::Mat img = imread(image_path, IMREAD_COLOR);
        // declaration d'une structure matricielle
        cv::Mat imgray;
        if(img.empty())
        {
            std::cout << "Could not read the image: " << image_path << std::endl;
            return 1;
        }
        // Affichage d'une image en opencv
```

```
28          cv::namedWindow("Display window", 1);
29          cv::imshow("Display window", img);
30          // fonction de conversion d'une image couleur
31          // en niveau de gris
32          cv::cvtColor(img,imgray,cv::COLOR_BGR2GRAY);
33          // Appelation du transform  de log
34          ip::log_transform(imgray);
35          // attente d'une touche de clavier
36          cv::waitKey(0);
37          return 0;
38      }
39
```

1. Nous notons ici que la fonction d'appellation du transformé de `log` est: `ip::log_transform(imgray);` d'où `ip::` est le `namespace` utilisé pour <*image processing*>.

2. La fonction suivante provienne de la bibliothèque d'`opencv` et est utilisée pour une conversion d'image de couleur en RGB. Dans le cours un peu plus tard, on décodera plus en détail le contenu de cette fonction.
`cv::cvtColor(img,imgray,cv::COLOR_BGR2GRAY);` Intuitivement, on en devine que c'est une fonction de conversion d'un espace de couleur à un autre.

3. Pour afficher une image en `opencv`, on utilise les deux lignes suivante:
`cv::namedWindow("On donne un nom", 1);`
`cv::imshow("On donne un nom", img);`

   Cette dernière ligne prend en arguement le même nom déclaré en `cv::namedWindow` et une structure matricielle en deuxième argument. **A noter ici le `string` <*On donne un nom*> doit être exactement pareil dans les deux appels de fonctions, sinon `opencv` nous donne une fenêtre vide!**

4. Autre astuce: si vous pointez votre cursor sur une fonction d'`opencv` et vous appuyez sur la touche F2 de votre clavier, l'éditeur QT vous renvoie vers l'appel de la fonction où vous pouvez voir en détail l'élaboration de la fonction.

Finalement, afin de rendre le `main` qu'on a programmé dans `pointproc.cpp` éxécutable, ces prochaines lignes doit être rajouter (si uniquement ce n'est pas encore fait) dans le fichier `apps/CMakeLists.txt`.

```
1   //# On creer un executable <pointproc> et on ....
```

```
2       //# le relie avec le fichier main pointproc.cpp
3       add_executable(pointproc pointproc.cpp)
4       //# On relie l'executable aux librairies utilisees.
5       target_link_libraries(pointproc
6                             IMAGE_PROC_FNS //#la bibliotheque static
7                             ${OpenCV_LIBS} //#la bibliotheque opencv)
8
```

2.3 Résultats

Dans cette section, nous élaborons les résultats obtenus du processus décrit en haut. Comme vous pouvez constater dans la figure suivante, l'application du transformé de `log` apporte un éclaircissement net de l'image et nous aide (au praticien) à mieux analyzer les différentes parties de cette dernière illustrant la colonne vertébrale d'un patient.

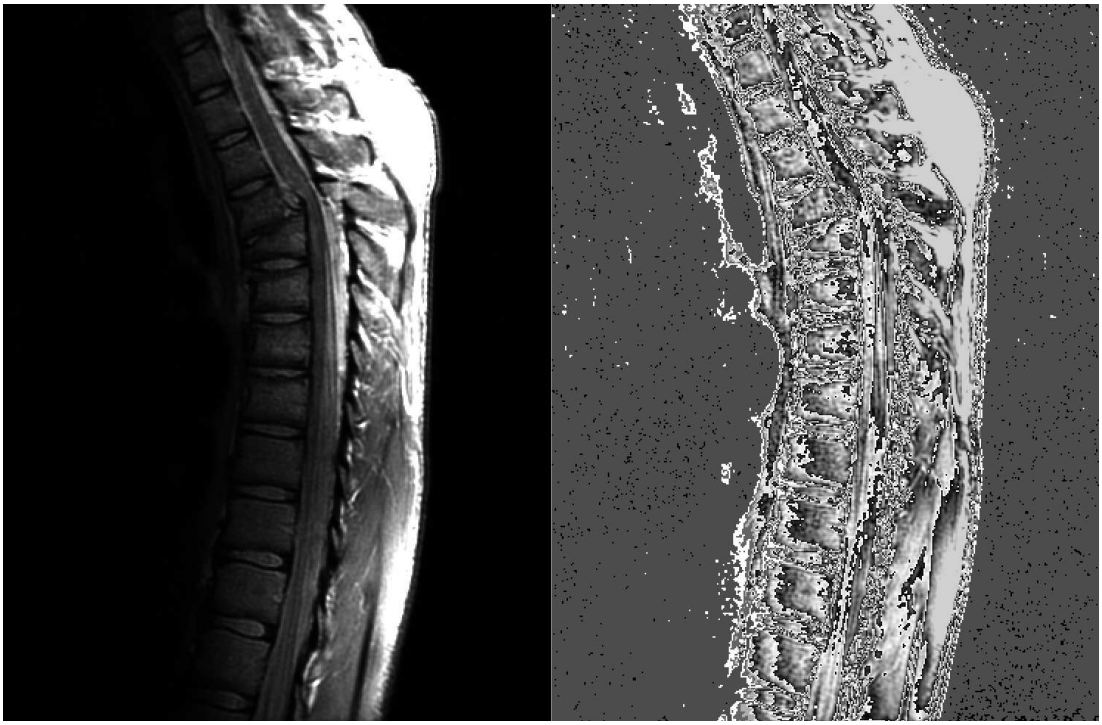


Figure 7: Résultat obtenu du transformé de `log` de l'image `fractured_spine.tiff`

2.4 Exercices

Les exercices sont à rendre au format complet. Le guide de l'élaboration du rapport est celui de la section 2.1 jusqu'au résultat, soit la section 2.3.

1. Implémentez le transformé de l'`exponentiel` en utilisant l'image `aerial_image.tiff`

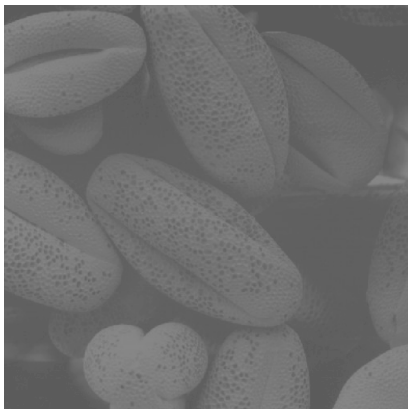



Table 2: (a)`aerial_image.tiff` et (b) `washed_out_pollen_image.tiff`

2. Implémentez le transformé de l'`gamma` en utilisant l'image `fractured_spine_image.tiff`. Utilisez differentes valeurs de gamma pour voir les effets.

3. Implémentez l'extension dynamique (contrast stretching) en utilisant l'image `washed_out_pollen_image.tiff`

Les étapes d'implémentations sont:

1. Déclaration de la fonction, *e.g.* `void exp_transform(...)` dans le header *.h

2. implémentation de la fonction, *e.g.* `void exp_transform(...)` dans *.cpp

3. appel de la fonction dans `pointproc.cpp`

4. compilez, exécutez et lancez dans le terminale

2.5 Egalisation d'histogramme

L'égalisation d'histogramme est une autre méthode d'ajustement du contrast de l'image qui utilise cette fois l'histogramme de cette dernière. Comme on a pu constaté, une image de 8 `bits` consiste de 256 valeurs, L de niveaux gris, pour chaque valeur de L récupérer de l'image on peut ainsi compter son nombre d'occurence. On construit alors l'histogramme de l'image.

Cette approche consiste à appliquer une transformation $T(r)$ sur chaque pixel, et donc d'obtenir une nouvelle image à partir d'une opération indépendante sur chacun des pixels. Cette transformation est construite à partir de l'histogramme cumulé de l'image de départ.

L'égalisation d'histogramme permet de mieux répartir les intensités sur l'ensemble de la plage de valeurs possibles, en "étalant" l'histogramme. Cette approche est intéressante pour les images dont la totalité, ou seulement une partie, est de faible con-

traste, c'est à dire, l'ensemble des pixels sont d'intensité proches. *(source: wikipedia, à référencer)*

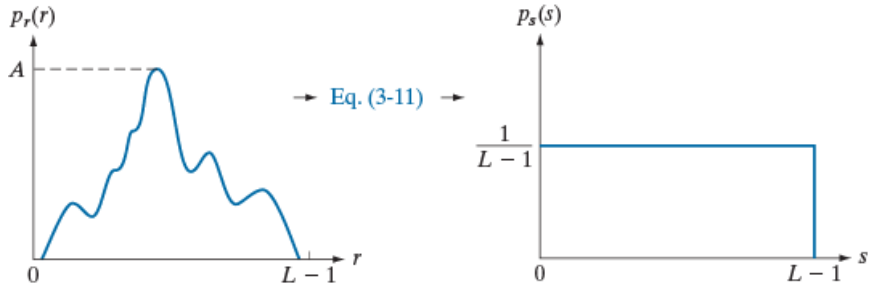


Figure 8: Transformation de la fonction de densité de probabilité du domaine r à s

Pour une image $\boldsymbol{\mathcal{I}}(r)$ (on utilisera la notation r ici pour être concis) en niveau de gris, codée sur L niveaux, on définit n_k, le nombre occurrences du niveau r_k dans l'image de taille M $\times$ N est :

$$p_r(r_k) = p(r = r_k) = \frac{n_k}{MN}, \quad 0 \leq k \leq L, \tag{6}$$

d'où $p_r(r_k)$ est normalisé sur $[0, 1]$. La transformation T qui à chaque valeur r_k de l'image d'origine associe une nouvelle valeur s_k, $s_k = T(r_k)$ est alors définie par:

$$T(r_k) = (L-1)\sum_{j=0}^{k} p_r(r_j), \tag{7}$$

d'où $\sum_{j=0}^{k} p_r(r_j)$ est **l'histogramme cumulé**.

2.5.1 Implémentation du code à compléter

```
1
2       void ip::computeHistogram( cv::Mat &image)
3   {
4       // taille complete en vecteur, M X N
5       int imsize = image.rows * image.cols ;
6       //initialisation du vecteur d'histogramme
7       std::vector<float> hist (256, 0);
8       // initialisation du pointeur    l'image
9       uchar *ptr = image.ptr<uchar>(0);
10      // on rempli les valeurs de l'histogramme
11      for(int i = 0; i< imsize; i++)
12          ++hist[ptr[i]]; //on incremente l'index du histogram
13
14      // on parcourt avec une boucle for et
```

```
15          // on normalise l'histogramme par M X N
16          std::vector<float> normhist (256, 0);
17          //  completez ici
18
19          // compute cummulative distribution function
20          std::vector<float> cfd (256, 0);
21          cfd[0] = normhist[0];
22          // on parcours avec une boucle l'histrogramme et on le cummule dans cfd
23          // remplissez ici
24
25          //Allocation du tableau de correspondence LUT
26          std::vector<float> lut(256,0);
27
28          // On parcourt cfd et on l'etale sur 255, soit lut = 255 X cfd,
29
30          // valeur entiere pour aller recuperer dans l'image
31          int value(0);
32          // on declare une nouvelle image pour voir le resultat
33          Mat imgequalise(image.rows, image.cols, CV_8UC1, Scalar(0));
34
35          // On parcourt l'image et on attribue chaque pixel
36          // au valeur associe dans LUT
37          for(int i = 0 ; i< image.rows; i++)
38              for(int j = 0 ; j< image.cols; j++)
39              {
40                  // A completer ICI
41              }
42          cv::namedWindow("image exp", 1);
43          cv::imshow("image exp", imgequalise );
44
45          show_histogram("hist2",imgequalise);
46
47      }
48
```

3 TP-3 Le Filtrage Spatiale

Le mot "filtrage" est emprunté du domaine fréquentiel où il se réfère à un passage permissive, la modification ou le rejet spécifique des composants fréquentiels d'une image. En occurence,

le filtre passe bas: laisse passer les basses fréquences d'où l'effet est de lisser l'image par une operation de floutage.

le filtre spatiale : modifie une image en remplaçant la valeur de chaque pixel par une fonction des valeurs du pixel et de son entourage. Si l'opération appliquée au pixel est linéaire, on dit que le filtre est spatialement linéaire. Au cas contraire, le filtre est non-linéaire.

3.1 Le mechanisme du filtrage spatiale

Un filtre linéaire spatiale éffectue une opération de produit de somme entre une image $\mathbf{f}$ et un *kernel* $\mathbf{w}$. Le kernel en question est un vecteur de valeurs dont sa taille définie l'opération sur l'entourage et dont les coefficient determine la nature du filtre. Autres mots utilisés pour *kernel* sont; *masque, fenêtre, template*. La figure suivante illustre le mécanisme du filtrage. Mathématiquement, cette opération est connue comme un produit de convolution où une moyenne glissante sur le signale (ou l'image en 2d) symbolise une intégrale sur l'entièreté du domaine du signale. On dit alors que les valeurs d'origine de l'image sont pondérées par celles du filtre en glissant cette dernière sur tout le domaine spatiale de l'image.

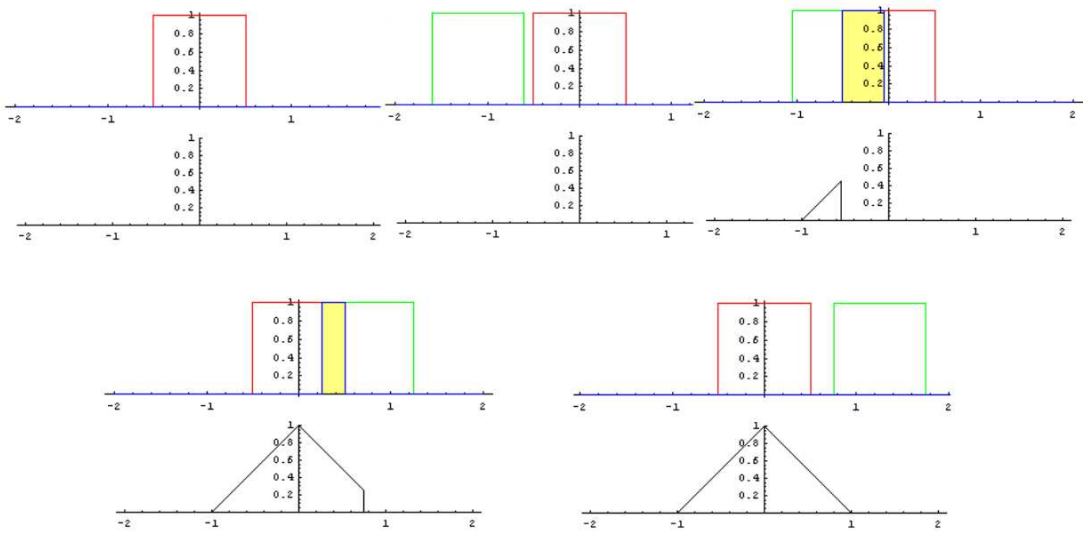


Figure 9: la convolution de $\mathbf{f} \circledast \mathbf{g}$ où l'une des deux fonctions étant parcourues en sens contraire l'une de l'autre afin de garantir la commutativité.

Avant de poursuivre l'élaboration de la convolution, on définit d'abord l'opération de **corrélation** qui est donné par une formulation mathématique (en 1d):

$$\mathbf{g}(\mathbf{x}) = \sum_{s=-a}^{a} \mathbf{w}(\mathbf{s})\mathbf{f}(\mathbf{x}+\mathbf{s}), \tag{8}$$

Ainsi, pour avoir une corrélation complète, le centre du kernel doit correspondre avec la valeur initiale du signale. Si la valeur tombe en dehors du domaine de l'image, un pavage de zeros est donc requis. L'explication est donnée dans la figure suivante:

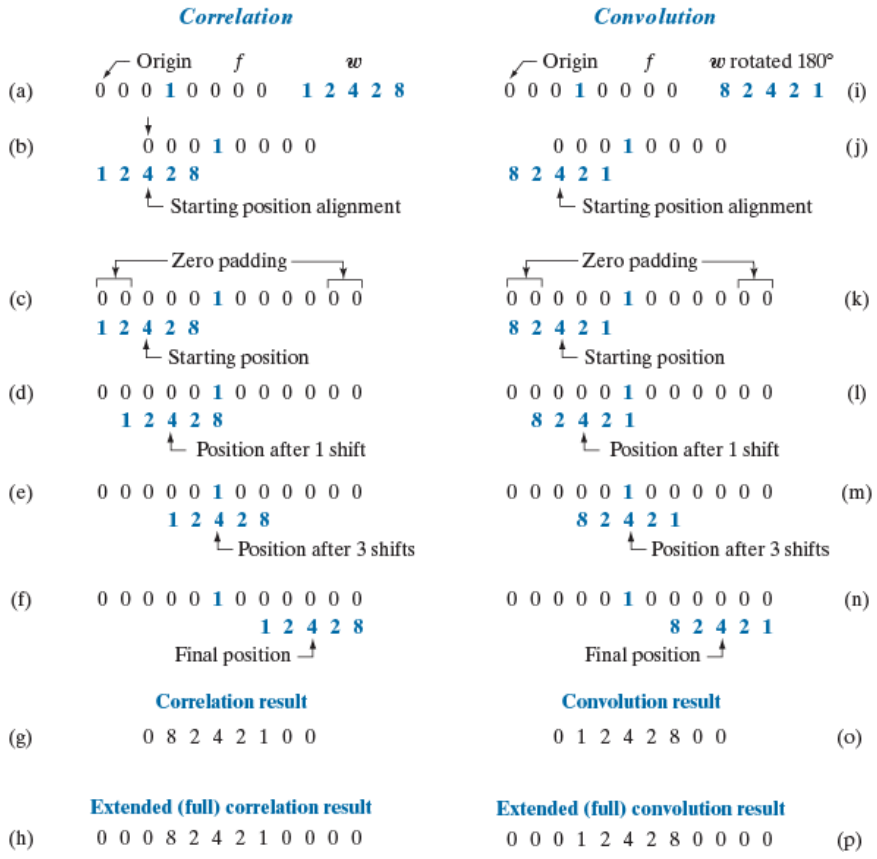


Figure 10: la colonne gauche représente la corrélation tandis que celle de droite représente la convolution.

La corrélation est donc définie en 2d par:

$$(w \circledcirc f)(x,y) = \sum_{s=-a}^{a}\sum_{t=-a}^{a} w(s,y)f(x+s,y+t) \tag{9}$$

La corrélation d'un kernel avec une fonction de Dirac résulte à une rotation de la version du kernel centré sur le dirac. Afin de recouvrir notre signale d'entrée, on doit

donc appliquer une rotation. C'est pour cela, en convolution, on applique d'abord une rotation de 180° avant de glisser le filtre de somme des produits sur l'image.

La convolution est donc donnée par:

$$(w \circledast f)(x,y) = \sum_{s=-a}^{a} \sum_{t=-a}^{a} w(s,y) f(x-s, y-t) \tag{10}$$

d'où le changement de signe "−" aligne les coordonnés de f et w quand l'un des deux subi une rotation de 180°. Pour illustré cette démarche afin d'avoir un oeuil sur l'implémentation, la figure suivante schématise la démarche décrit en haut.

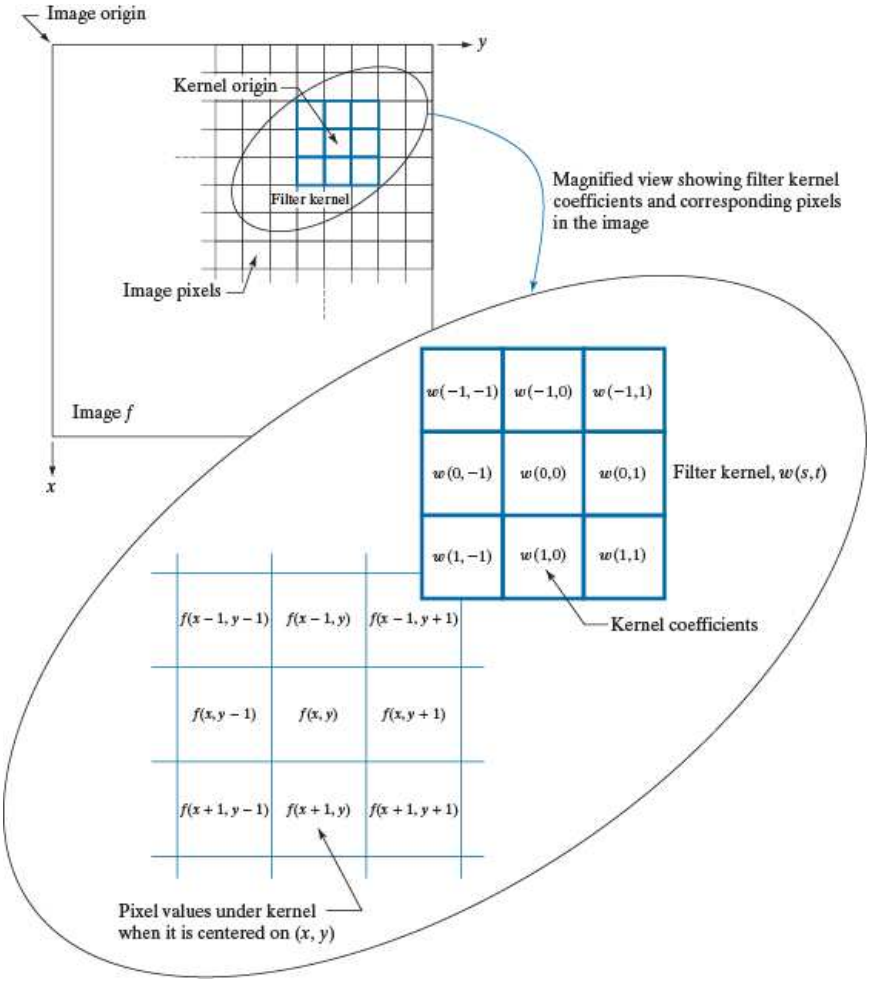


Figure 11: Illustration de la convolution en 2d.

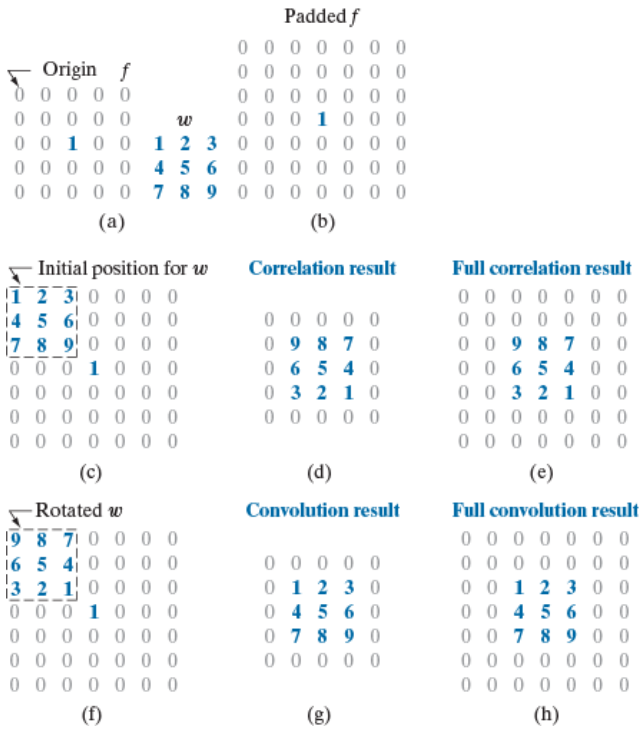


Figure 12: Illustration de la corrélation et la convolution en 2d.

3.2 Comparaison entre un filtre dans le domaine spatiale et fréquentiel

- La convolution dans le domaine spatiale est équivalent de la multiplication dans le domaine fréquentiel et vice-versa.
- Un dirac $\delta \times A$ dans le domaine spatiale est un constant de valeur A dans le domaine fréquentiel et vice-versa.

Un signale (ou image) satisfaisant quelques conditions dites “legères” peut être exprimé comme la somme des sinusoïdes de différents fréquences et amplitudes. Donc, l’apparence d’une image dépend des composants fréquentiels de ces sinusoïdes. Un changement dans la fréquence des composants résulte en un changement dans l’apparence des images.

Les régions de l’image avec une faible variation en niveau de gris (*e.g.* les mûrs) caractérisent des sinusoïdes de basse fréquence. Similairement, les bords et autres transitions fortes sont caractérisés par des hautes fréquence. Donc, en réduisant les composants fréquentiellement fortes dans l’image, cette dernière donne le sens du floutage , soit ”blur”.

On conclut donc que le filtrage linéaire est concerné par l'épreuve de trouver des processus convenable afin de modifier le contenu fréquentiel de l'image pour avoir des résultats satisfaisants, voir des qualités supérieures à celles de l'images d'entré. Donc, dans le domaine spatiale, l'approche de filtrage par convolution est appliquée. Dans le domaine fréquentiel, on applique les filtres multiplicatifs.

3.3 Construction des filtres spatiales

3.3.1 Approche 1

1. Un filtre faisant la moyenne des pixels dans son entourage résulte en floutage de l'image. Cette opération est similaire à un calcule numérique de l'intégration.

2. Un filtre qui calcule la dérivée locale de chaque pixel de l'image résulte en une image plus "affûtée" (sharpen).

3.3.2 Approche 2

1. Échantillonnant une fonction spatiale en 2d dont la forme a des propriétés intéressante. Par exemple, les échantillons d'une gaussienne peut être utilisés pour construire un filtre passe bas de moyenne pondérée. Ces fonctions spatiale en 2D peuvent être généré comme l'inverse du tranformé de Fourier spécifié dans le domaine fréquentiel.

2. En conceptualisant un filtre désiré avec une réponse fréquentielle spécifiée.

3.4 Le lissage spatiale (passe-bas)

Aussi connu comme des filtres de moyennes pondérés, leurs effets sont de réduire des fortes transitions en intensité. De ce fait, comme les bruits aléatoires consistent de ce genre de transition, les filtres de lissage en sont une très bonne application de reduction de détails triviaux dans l'image.

3.4.1 Box Filter Kernels

Dont le résultat de la convolution est l'équivalent d'une moyenne pondérée, le filtre est donné par la matrice suivante:

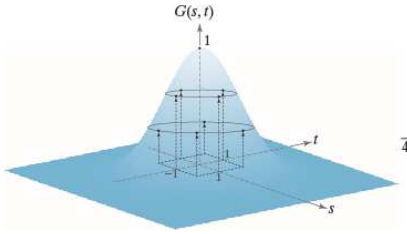
$$B_F = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Le désavantage connu de ce filtre c'est qu'il a tendance à favoriser le lissage autour les directions orthogonaux. Dans des applications comprenant d'images avec niveau de détails conséquent, ou avec des formes géométriques, cela peut être problématique.

3.4.2 Le filtre Gaussien

Le filtre gaussien est donné par l'équation suivante:

$$G(s,t) = \frac{1}{2\pi\sigma^2} e^{-\frac{(s-s_0)^2(t-t_0)^2}{2\sigma^2}} \tag{11}$$



$$\frac{1}{4.8976} \times \begin{bmatrix} 0.3679 & 0.6065 & 0.3679 \\ 0.6065 & 1.0000 & 0.6065 \\ 0.3679 & 0.6065 & 0.3679 \end{bmatrix}$$

Figure 13: Illustration du filtre gaussien en 2d

Les propriétés attractifs du gaussien sont:

1. Le filtre est isotrope, soit, circulairement symétrique. Ce propriété est très désirable puisque nous voulons lisser l'image de façon non biaisé et uniforme dans toutes les directions. On dit donc qu'il est invariant à la rotation.

2. Il a un seul "pic". Cela implique qu'on peut appliquer une moyenne pondérée avec des poinds décroissants pour un pixel centré localement sur trouvant sur la pente de la fonction.

3. Le niveau de lissage est contrôlé par le seule paramètre σ. Un grand σ implique un grand degré de lissage.

4. Le filtre est séparable en s et t, signifiant un gain en tant de calcule de $MN(w_s \times w_t)$ à $MN(w_s + w_t)$. Donc le gain est $\frac{MN(w_s+w_t)}{MN(w_s \times w_t)}$.

5. La réponse fréquentielle du filtre est aussi gaussienne. Par conséquent, les composants de hautes fréquences sont atténuées et donc le filtre est approprié pour le filtrage de bruit.

On note la décomposition du gaussien comme suite:

$$\begin{aligned} G(s,t) &= \frac{1}{2\pi\sigma^2} e^{-\frac{(s^2+t^2)}{2\sigma^2}} \\ &= \frac{1}{2\pi\sigma^2} e^{-\frac{s^2}{2\sigma^2}} \delta(s) \times \frac{1}{2\pi\sigma^2} e^{-\frac{t^2}{2\sigma^2}} \delta(t) \end{aligned} \tag{12}$$

$$G(s,t) = g_s \circledast g_t \tag{13}$$

$$\begin{aligned} \boldsymbol{\mathcal{I}}(x,y) \circledast \mathcal{G}(s,t) &= \boldsymbol{\mathcal{I}}(x,y) \circledast g_s \circledast g_t \\ &= \left[\boldsymbol{\mathcal{I}}(x,y) \circledast g_s\right] \circledast g_t \end{aligned} \tag{14}$$

3.4.3 L'implémentation du filtre gaussien

```
1  void ip::gaussianKernel(cv::Mat &gaussK, const double &sigma_g)
2     {
3         int size = gaussK.rows;
4
5         double x0 = floor(size / 2); // on arrondie la valeur
6         double y0 = floor(size/ 2);  // on arrondie la valeur
7
8         double rx, ry, lg;
9
10        // facteur de normalisation de l'equation
11        lg = (1/(2*sigma_g *sigma_g));
12
13        double norm = 0;
14
15        // double boucle pour parcourrir la matrice 2d
16        for (int y = 0 ; y < size  ; y++)
17            for(int x = 0 ; x < size ; x++)
18            {
19                // on centralise la gaussienne
20                rx = (x-x0) * (x-x0) ;
21                ry = (y-y0) * (y-y0) ;
22
23                // application de la formule gaussienne
24                gaussK.at<float>(x,y) = exp(-(rx  + ry)*lg);
25
26                // la somme de tous les elements de la matrice
27                norm += exp(-(rx  + ry)*lg);
28
29            }
30        // normalisation de la gaussienne centree
31        gaussK /= norm;
32        return;
33    }
34
```

3.4.4 L'implémentation de la convolution

```
1  void ip::convolve(cv::Mat &imgin, cv::Mat &imgout, const cv::Mat &mask)
2 {
```

```
3     CV_Assert(imgin.depth() == CV_8U);  // accept only uchar images
4
5     // on deduit la taille du mask en x et y
6     int halfwx = floor(mask.rows/2);
7     int halfwy = floor(mask.cols/2);
8     float sum = 0;
9
10    // Balayage de l'image en 2d
11    for(int i = halfwx ; i< imgin.rows - halfwx ; i++){
12        for(int j = halfwx ; j< imgin.cols - halfwx; j++)
13        {
14            // Initialisation de la somme
15            sum = 0;
16            // Glissage du masque/filtre dans l'image
17            for (int wx = -halfwx; wx<=halfwx; wx++ )
18                for (int wy = -halfwy; wy<=halfwy; wy++ )
19                {
20                    int u_wx = i + wx;
21                    int v_wy = j + wy;
22
23                    // calcule du produit de somme des valeurs
24                     sum += imgin.at<uchar>(u_wx,v_wy) *
25                             mask.at <float>(wx +halfwx,wy +halfwy);
26          }
27            // si somme plus que 255, valeur max  = 255
28            imgout.at <uchar>(i,j) = (int) (sum > 255)? 255: sum;
29
30        } }// fin balayage de l'image
31    }
32
```

3.5 Exercices

1. Implémentez le filtre moyenneur (box filter) avec differentes tailles de mask et analysez l'impact sur les images `circuitboard-gaussian.tif` et `testpattern1024.tif`.

2. Analysez l'impacte de σ et la taille du mask sur l'image `circuitboard-gaussian.tif`

3. Analysez les résultats du filtre médiane.

4. Testez les exercices d'en haut en rajoutant du bruit gaussien et sel et poivre dans l'image.

5. Completez/ complémentez ce T.P avec les exercises du chapitre 10 de Diane Lingrand (Page 173).

3.6 Détection de bords

Une approximation de dérive de premier - ordre à un point arbitraire en dimension 1-D d'une fonction $f(x)$ est obtenue par l'expression de Taylor de $f(x + \Delta x)$ autour de x soit:

$$\begin{aligned} f(x+\Delta x) &= f(x) + \Delta x \frac{\partial f(x)}{\partial x} + \frac{(\Delta)^2}{2!}\frac{\partial^2 f(x)}{\partial x^2} + \frac{(\Delta)^3}{3!}\frac{\partial^3 f(x)}{\partial x^3} \dots \\ &= \sum_{n=0}^{\infty} \frac{(\Delta x)^n}{n!}\frac{\partial^n f(x)}{\partial x^n} \end{aligned} \tag{15}$$

d'où Δx est la séparation entres plusieurs échantillons de f. Dans notre cas, cette séparation est mesuré en unité pixels. Donc, on a $\Delta x = 1$ pour un échantillon précédent de x et $\Delta x = -1$ est celui du successif. Donc:

$$\begin{aligned} f(x+1) &= f(x) + \frac{\partial f(x)}{\partial x} + \frac{1}{2!}\frac{\partial^2 f(x)}{\partial x^2} + \frac{1}{3!}\frac{\partial^3 f(x)}{\partial x^3} + \dots \\ &= \sum_{n=0}^{\infty} \frac{1}{n!}\frac{\partial^n f(x)}{\partial x^n} \end{aligned} \tag{16}$$

Simultanément,

$$\begin{aligned} f(x-1) &= f(x) - \frac{\partial f(x)}{\partial x} + \frac{1}{2!}\frac{\partial^2 f(x)}{\partial x^2} - \frac{1}{3!}\frac{\partial^3 f(x)}{\partial x^3} + \dots \\ &= \sum_{n=0}^{\infty} \frac{(-1)^n}{n!}\frac{\partial^n f(x)}{\partial x^n} \end{aligned} \tag{17}$$

On en déduit alors le forward difference: différence finie "avancé"

$$\frac{\partial f(x)}{\partial x} = f'(x) = f(x+1) - f(x) \tag{18}$$

d'où les termes linéaires sont conservés,

Backward difference:

$$\frac{\partial f(x)}{\partial x} = f'(x) = f(x) - f(x-1) \tag{19}$$

Central difference:

$$\frac{\partial f(x)}{\partial x} = f'(x) = \frac{f(x+1) - f(x)}{2} \tag{20}$$

Remarques: 3.1. • *Les termes d'ordre grandissants qu'on a pas utilisé représente l'erreur numérique entre une solution exacte et approximation de la dérivée.*

- *En général, plus l'ordre est grand plus notre solution se rapproche de la précision exacte.*
- *Donc l'utilisation des termes de l'ordre grandissant implique un taux de calcule grandissant.*

Par conséquent, il en découle un "stack" de filtres/kernels/masques/noyaux qui vont définir l'application numérique des dérivées sur une image comme suit:

1. Masque convolutionel:
$$G_x = \begin{bmatrix} 0 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad G_y = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$
2. Mask convolutionel (Robert's Cross):
$$G_x = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad G_y = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}$$
3. Central difference:
$$G_x = \begin{bmatrix} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \quad G_y = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$
4. Prewitt:
$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}, \quad G_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$
5. Sobel:
$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Remarques: 3.2.
- *le gradient est sensible au bruit*
- *il produit des bords "non fermés" et donc des étapes supplémentaires de poste traitement sont nécessaires afin de relier les bords*

3.6.1 La dérivée de second ordre

La dérivée de second ordre basée sur la différence centrale et des termes en additionnant (16) et (17),c'est à dire;

$$\frac{d^2f(x)}{dx^2} = f'' = f(x+1) - 2f(x) + f(x-1) \tag{21}$$

En 2-D, la derivée de second ordre est définie par une équation laplacienne donnée par:

$$\bigtriangledown^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \tag{22}$$

On en déduit alors le filtre laplacien comme suit:

1. Masque convolutionel:

$$L_4 = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}, \quad L_8 = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Remarques: 3.3.

- *les bords sont définis comme des points de passages à zéros (zero crossing) du laplacien*
- *le laplacien est très sensible au bruit, c'est à dire, il accentue le bruit.*

3.7 Exercices

1. A implémenter tous les filtres et reproduire les résultats décrit dans le diapo de Yulya Tarabalka en utilisant les images du répertoire `dataset_lignes` fourni.

4 TP-4 La colorimetrie

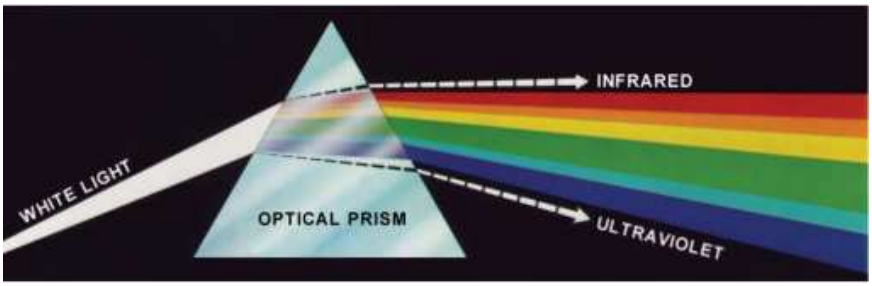


Figure 14: Le spectre de la lumière visible

Le traitement d'images de couleurs est motivé par deux facteurs principaux. Premièrement, la couleur est un descripteur puissant qui simplifie souvent l'identification et l'extraction d'objet dans la scène. Deuxièmement, l'humain peut discerner entre des milliers d'ombres de couleurs comparé de seulement deux douzaines de niveau de gris. Ce dernier est particulièrement intéressant pour l'analyse manuelle des images.

On dit que la lumière est achromatique lorsqu'elle est démunie de la couleur. Son seul attribut est l'intensité. La lumière chromatique étale le spectre électromagnétique de 400 nm à 700 nm. Qualitativement, une source de lumière achromatique est décrit par trois caractéristiques; la radiance, la luminance et l'intensité (brightness).

1. Radiance: c'est l'énergie totale émanant de la source lumineuse mésurée en watts (W).

2. Luminance : c'est une mésure de l'énergie qu'un observateur perçoit de la source lumineuse, mésurée en lumens (lm). Par exemple, la lumière émanant de la source opérant dans l'extrémité la région infra rouge du spectre a une énergie (radiance) significative mais qu'un observateur a du mal à percevoir.

3. L'intensité: est un descripteur subjectif, pratiquement impossible à mésurer. Il englobe la notion achromatique de l'intensité et est un des facteurs clé décrivant la sensation colorimétrique.

Les caractéristiques distinguant généralement une couleur de l'autre sont: la hue (teinte), la saturation et l'intensité.

1. L'intensité: comme introduit précédemment, l'intensité englobe la notion achromatique de la lumière et peut être perçu comme une nuance (assombrissement ou éclaircissement).

2. La teinte: représente la couleur dominante comme perçu par l'observateur. Lorsqu'on décrit un objet comme rouge, orange ou jaune, on se réfère à cet attribut.

3. La saturation: réfère au pureté relative ou la quantité de la lumière blanche mélangée avec la teine. Par exemple, on dit que les couleurs pures du spectre visible sont complètement saturées. Les couleurs comme rose(pink) (rouge + blanc), et lavande (violet + blanc) sont moins saturées avec un degré de saturation étant inversement proportionnel à la quantité de la lumière blanche ajoutée.

   La teinte et la saturation combinées sont connus sous le terme de chromaticité. La quantité de rouge, vert et bleu requise afin d'obtenir une couleur particulière est connue comme les valeurs tristimulus et sont dénotés par X, Y et Z respectivement.

Alors, la couleur est spécifiée par ses coefficients `trichromatique` et est définie par:

$$x = \frac{X}{X+Y+Z} \quad y = \frac{Y}{X+Y+Z} \quad z = \frac{Z}{X+Y+Z} \tag{23}$$

On en déduit alors que

$$x + y + z = 1 \tag{24}$$

L'œil humain B perçoit les couleurs comme une variable combinatoire des couleurs primaires soit 65% de rouge, 33% de vert ,2% de bleu à travers les cones oculaires.

4.1 Les fondamentaux

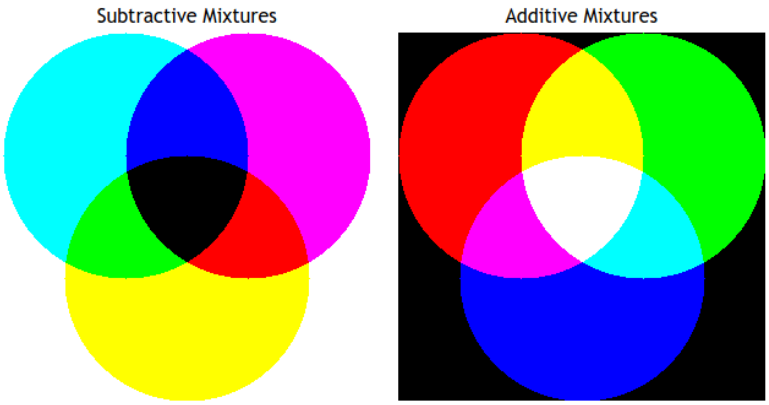


Figure 15: Les mélanges additifs et subtractifs

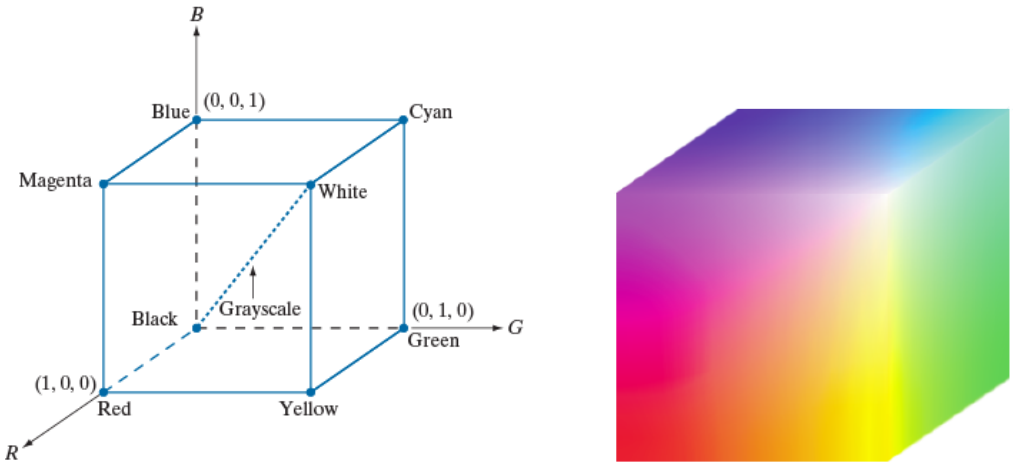


Table 3: (a)l'espace RVB (b) la représentation RVB en 24 `bits`

Les couleurs primaires peuvent être rajoutées afin de produire des couleurs secondaires; magenta (R+B), cyan (G+B) et jaune (R+G).

Une couleur primaire est celle qui absorbe ou soustrait une couleur primaire de la lumière et reflète ou transmet les deux autres. Le Jaune, cyan, magenta sont les pigments des couleurs primaires.

La synthèse soustractive se produit en imprimerie. C'est pourquoi les imprimeurs utilisent les composantes YCM. Si on soustrait la lumière magenta de la lumière blanche (par exemple un filtre), on obtient de la lumière verte. Si on soustrait la lumière cyan, on obtient de la lumière rouge et si on soustrait la lumière jaune, on obtient de la lumière bleue. Si on soustrait à la fois de la lumière magenta, cyan et jaune (par exemple en superposant trois filtres), on n'obtient plus de lumière, donc du noir. [*extrait copié de Diane Lingrand*].

Modèles de couleurs (Espace de couleur)

Le but d'un espace de couleur, c'est de faciliter la spécification de couleurs. En essence, un modèle de couleur est constitué d'une spécification d'un:

1. système de coordonnées
2. sous-espace du système, telle que chaque couleur de modèle est représenté par un point du sous-espace

---

4.1.1 Le modèle RGB

Une image RGB est représentée par $8 \times 3 = 24$ `bits`. Le nombre total de possibilités de couleurs est donné comme suit;

$$\underbrace{2^8}_{8 \text{ bits}} \underbrace{2^8}_{8 \text{ bits}} \underbrace{2^8}_{8 \text{ bits}} = (2^8)^3 = 16,777,216$$

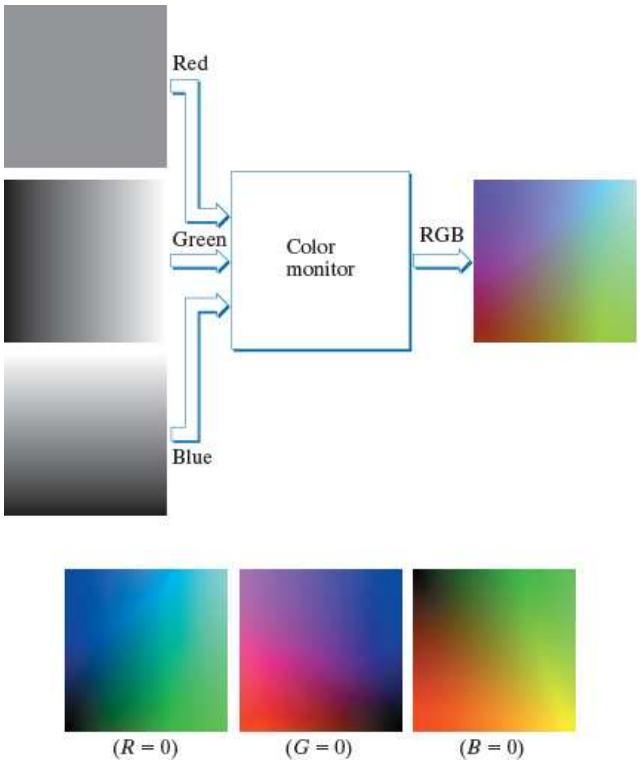


Figure 16: La décomposition des canaux R, V, B pour l'affichage

4.1.2 Le modèle CMY-K

La relation entre le modèle CMY et RGB est donnée par:

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Le modèle $CMYK$ est utilisé dans les imprimantes d'où K est une dose de proportionalité de la couleur noire.

La conversion de CMY à CMYK se fait en mettant:

$$K = min(C, M, Y)$$

Si $K = 1$, nous avons du noir pur, sans contribution de couleur, et donc $C = M = Y = 0$.

Sinon,

$$C = \frac{C - K}{1 - K}$$
$$M = \frac{M - K}{1 - K}$$
$$Y = \frac{Y - K}{1 - K}$$

d'où les valeurs sont normalisées entre [0 1]. Le chemin inverse de CMYK à CMY peut être facilement calculé en inversant les équations décrient en haut.

4.1.3 Le modèle HSI

1. La teinte est un attribue qui décrit la couleur pure (jaune,orange,rouge, ...)
2. Saturation: est le degré de pureté dilué par la lumière blanche
3. Intensité: est un descripteur subjectif étant pratiquement impossible à mesurer. Elle englobe la notion achromatique (sans teinte) et est un facteur clé décrivant la sensation de couleur en terme d'éclaircissement ou de l'assombrissement.

Le modèle HSI découple la composante de l'intensité de l'information colorimétrique qui la transmet. De ce fait ce dernier est un outil utile pour le développement des algorithmes en traitement d'images.

Donc en sommaire, on peut dire le modèle RVB est idéale pour la génération de couleur (acquisition d'image et l'affichage sur un écran) mais son utilité pour la description de couleur est bien limité.

On connait désormais qu'une image en couleur est constituée de 3 images d'intensités en niveau de gris. Donc il est sans surprise qu'on peut extraire la composante d'intensité d'une image RVB.

Si l'on considère l'espace de couleur RVB est qu'on déssine une ligne du vertex (0,0,0) , soit le noir et qu 'on la rejoigne avec le vertex (1,1,1) (le blanc), on a alors la variation en intensité qui s'étend sur cette droite verticale. Donc, si l'on voulait déterminer la composante d'une couleur à un point, on aurait simplement défini un plan perpendiculaire à l'axe d'intensité contenant le point.

L'intersection du plan avec une valeur de l'intensité nous donne un point dont la valeur se positionne entre [0,1].

On en déduit de ce raisonnement que la saturation (pureté) d'une couleur est une fonction incrémentale de la distance de l'axe de l'intensité. En fait la saturation des points sur l'axe de l'intensité est zero, témoignant le fait que tous les points étant sur cet axe sont au niveau de gris.

La teinte peut être aussi déterminée du modèle RVB. Considérant la figure 17 illustrant un plan défini par 3 points (noir, blanc, cyan), le fait que les points blanc

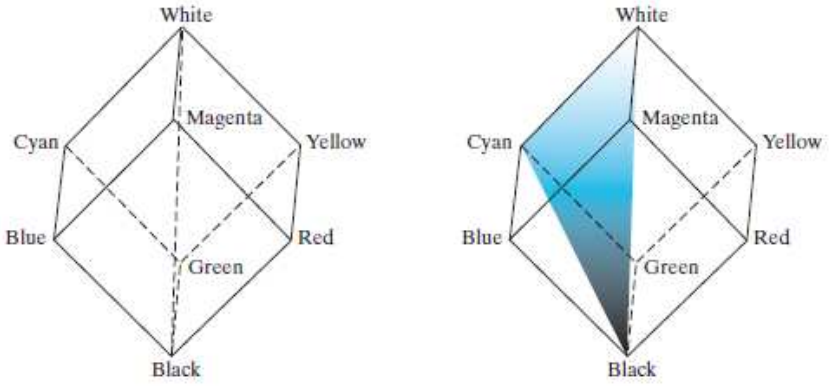


Figure 17: RVB à HSI: le raisonnement

et noir sont contenus dans le plan indique que l'axe de l'intensité est aussi contenue dans le plan. De plus, on observe que tous les points qui sont contenus par le segment du plan défini pat l'axe de l'intensité et les frontière du cube ont les mêmes teintes (cyan and ce cas). On arrive à la même conclusion en se rappelant que toutes les couleurs générées à partir des trois couleurs se trouvant dans une triangle défini par ce dernier.

Si deux de ces points sont noir et blanc, le troisième est un couleur, tous les points sur le triangle auront la même teinte, parceque les composants blanc et noir ne peuvent changer la teinte (l'intensité et la saturation des points sur le triangle seront différents) . En appliquant une rotation du plan nuancé autour de l'axe verticale, on obtient différente teinte. On en conclut donc que la teinte, la saturation et l'intensité peuvent toutes être dérivées du modèle RVB.

Le point clé de cette reorganisation du cube dans la figure 17 et de son espace de couleur correspondant en HSI est que ce dernier est représenté par un axe verticale d'intensité et que le locus des points de couleurs répartissent aux plans perpendiculairement à l'axe.

Comme les plans glissent de haut en bas sur l'axe de d'intensité, les frontières qui sont définies par l'intersection de chaque plan aux facets du cube donne la forme d'un triangle ou un hexagone (figure 18). Cela peut être visualisé rapidement en regardant le cube tout droit sur l'axe du niveau de gris.

Les couleurs primaires sont d'un écart de 120°. Quant aux couleurs secondaires, elles sont de 60° aux couleurs primaires. L'angle entre les secondaires sont aussi 120°.

- L'angle contenu par l'axe rouge désigne le 0° et s'incrémente en anti-horraire défini la teinte (Hue).
- Le vecteur décrivant la distance d'un point aléatoire du centre au point définie la saturation.

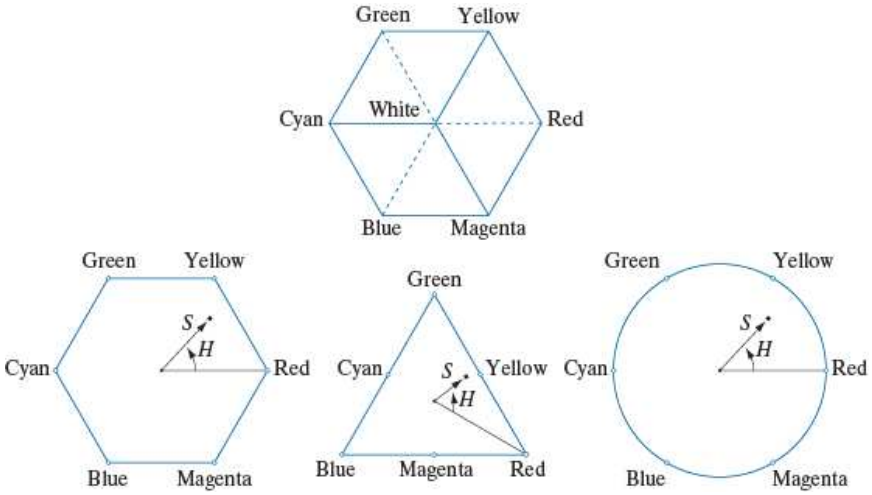


Figure 18: Interprétation du modèle HSI

- L'axe de l'origine émanant de l'hexagone défini l'intensité

Les composantes importantes du modèle de l'espace HSI sont l'axe verticale décrivant l'intensité, La taille du vecteur $\overrightarrow{OS}$ et l'angle entre $\overrightarrow{OS}$ et $\overrightarrow{OR}$. De ce fait, il est d'image de voir le modèle HSI en hexagone, triangle ou cercle. Peut importe la forme choisie, elle peuvent être facilement transformée dans les deux autres espaces par une transformation géométrique.

La conversion mathématique de RVB à HSI est donnée par:

$$H = \begin{cases} \theta \text{ si } B \leq V \\ 360 - \theta \text{ si } B > V \end{cases} \tag{25}$$

avec

$$\theta = \cos^{-1}\left\{ \frac{\frac{1}{2}\big((R-V)+(R-B)\big)}{\left[(R-V)^2+(R-B)(V-B)\right]^{\frac{1}{2}}} \right\} \tag{26}$$

La saturation est donnée par:

$$S = 1 - \frac{3}{R+V+B}\left[\min(R,V,B)\right] \tag{27}$$

Remarques: 4.1. *Il est pratique d'ajouter un ϵ au dénominateur afin d'éviter la division par zéro. Lorsque $R = V = B$, d'où le cas de $\theta = 90°$.*

Et finalement l'intensité est donnée par:

$$I = \frac{1}{3}(R+V+B) \tag{28}$$

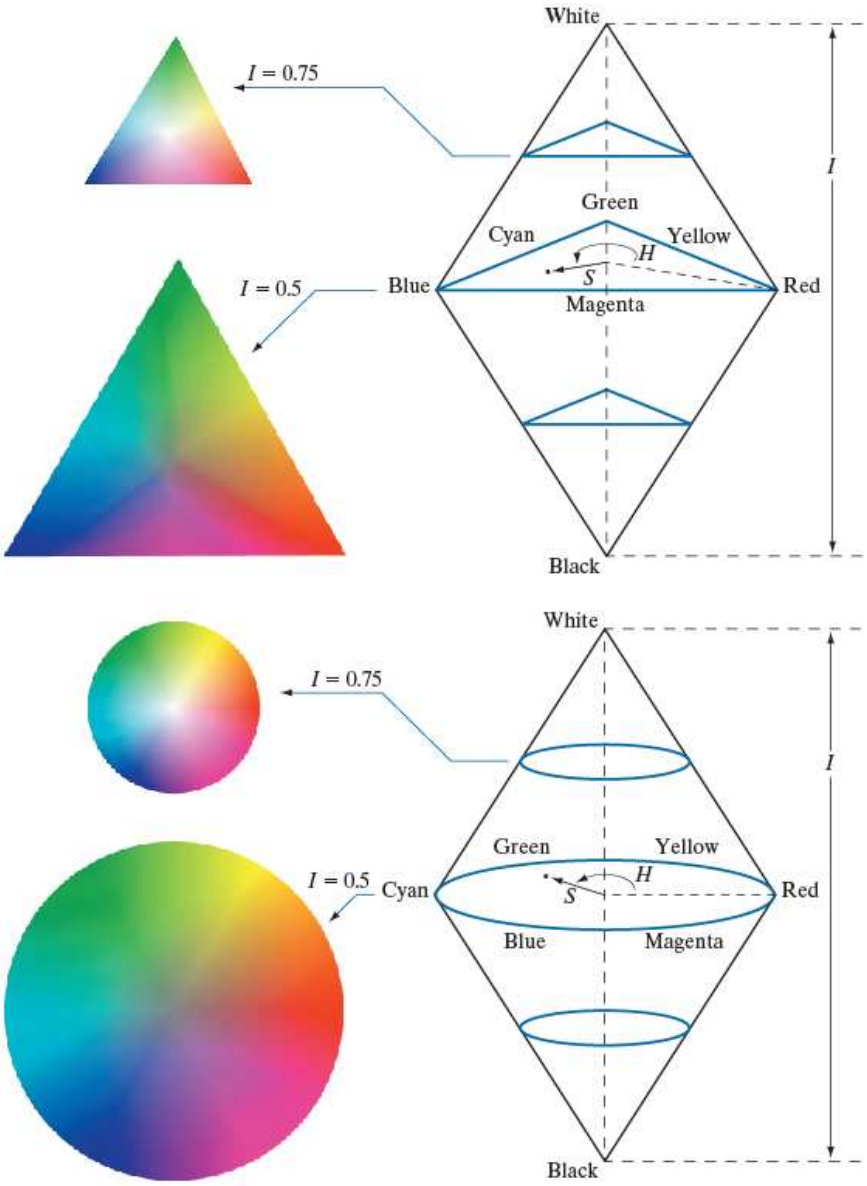


Figure 19: D'autres interprétations du modèle HSI

5 Implémentation algorithmique

Les codes illustrés dans cette section sont à prendre comme pseudocode et non pas comme un acquis "copier/coller" puisqu'il y a des erreurs d'éditions sur LaTeX. Par exemple les brackets ne sont pas inclus mais le coeur des algos y sont bien évidemment!

5.0.1 Conversion d'une image couleur en niveau de gris

```
void ip::convert2gray(cv::Mat &imgin, cv::Mat &imgout){
    if(imgin.type() == CV_8UC3){
        for(int i = 0 ; i< imgin.rows; i++)
            for(int j = 0 ; j< imgin.cols; j++){
                imgout.at<uchar>(i,j) = (imgin.at<cv::Vec3b>(i,j)[0] +
                                         imgin.at<cv::Vec3b>(i,j)[1] +
                                         imgin.at<cv::Vec3b>(i,j)[2])/3;
            }}}

```

5.0.2 Colour slicing

```
void ip::colourslice(cv::Mat &imgin, cv::Mat &imgblue,
                     cv::Mat &imggreen, cv::Mat &imgred)
{

    if(imgin.type() == CV_8UC3){
        for(int i = 0 ; i< imgin.rows; i++)
            for(int j = 0 ; j< imgin.cols; j++){
                //Extracting blue channel
                imgblue.at<cv::Vec3b>(i,j)[0] = imgin.at<cv::Vec3b>(i,j)[0];
                imgblue.at<cv::Vec3b>(i,j)[1]  = 0;
                imgblue.at<cv::Vec3b>(i,j)[2]  = 0;

                //Extracting green channel
                imggreen.at<cv::Vec3b>(i,j)[0] =  0;
                imggreen.at<cv::Vec3b>(i,j)[1]  = imgin.at<cv::Vec3b>(i,j)[1];
                imggreen.at<cv::Vec3b>(i,j)[2]  = 0;

                //Extracting red channel
                imgred.at<cv::Vec3b>(i,j)[0] = 0;
                imgred.at<cv::Vec3b>(i,j)[1] = 0;
                imgred.at<cv::Vec3b>(i,j)[2] = imgin.at<cv::Vec3b>(i,j)[2];

            }}}

```

5.0.3 Conversion RVB à HSI

```
void ip::convert2HSI(cv::Mat &imgin, cv::Mat &imghsi)
{
    float blue, green, red, saturation, hue, intensity,theta, frac1,frac2;
    float eps = 1e-6; //epsilon

    // initialisation du pointeur    l'image

```

```
7
8       if(imgin.type() == CV_8UC3)
9       {
10          for(int i = 0 ; i< imgin.rows; i++){
11              for(int j = 0 ; j< imgin.cols ; j++)
12              {
13                  // extraction du bleu, vert et rouge et on normalize par 255
14                  blue = imgin.at<cv::Vec3b>(i,j)[0]/255.0;
15                  green = imgin.at<cv::Vec3b>(i,j)[1]/255.0;
16                  red = imgin.at<cv::Vec3b>(i,j)[2]/255.0;
17
18                  frac1 = (float)((red - green) + (red - blue)) *0.5;
19                  frac2 = (float) sqrt(pow((red - green),2) +
20                          ((red - blue)*(green - blue)) + eps);
21
22                  //The denominator cannot be ZERO! case of undefined 0/0
23                  if(frac2 == 0){
24                      hue = 0;
25                  }
26                  else{
27                      theta = acos(frac1/frac2);
28                      if(blue <= green)
29                          hue = theta/(M_PI*2); // on normalise par 2*PI
30                      else
31                          hue = (2*M_PI - theta)/(2*M_PI); //on normalise par 2*PI
32                  }
33
34                  // calcule du min (R,V,B)
35                  float minRGB = min(min(red,green),blue);
36
37                  // somme de (R,V,B)
38                  float den = red+green+blue;
39
40                  // The denominator cannot be ZERO!
41                  if(den == 0)
42                      saturation = 0;
43                  else
44                      saturation = 1 - 3*minRGB/den; // calcule de la saturation
45
46                  // calcule de l'intensite
47                  intensity = den/3.0;
48
49                  // concatenation des matrices dans la structure cv::Vec3b
50                  imghsi.at<cv::Vec3b>(i,j)[0] = hue * 255;
51                  imghsi.at<cv::Vec3b>(i,j)[1] = saturation * 255;
52                  imghsi.at<cv::Vec3b>(i,j)[2] = intensity * 255;
53
54              }}}}
55
```

5.0.4 Appel dans le main `colourproc.cpp`

Un nouveau fichier `colourproc.cpp` est à créer avec les dépendences des bibliothèques requises. (à voir dans les exemples précédents).

```
1
2     // exemple de declaration d'une structure matricielle de couleur de 3 canaux
3     cv::Mat imgblue(imgcol.rows, imgcol.cols, CV_8UC3);
4     cv::Mat imggreen(imgcol.rows, imgcol.cols, CV_8UC3);
5     cv::Mat imgred(imgcol.rows, imgcol.cols, CV_8UC3);
6
7     // exemple de colour slice
8     ip::colourslice(imgcol,imgblue, imggreen, imgred);
9
10    // conversion de RVG a nuance de gris
11    cv::Mat imgray(imgcol.rows, imgcol.cols, CV_8UC1, Scalar(0));
12    ip::convert2gray(imgcol,imgray);
13
14    // conversion RGB a HSI
15    cv::Mat imghsi(imgcol.rows, imgcol.cols, CV_8UC3);
16    ip::convert2HSI(imgcol,imghsi);
17    cv::namedWindow("display hsi",WINDOW_AUTOSIZE);
18    cv::imshow("display hsi", imghsi);
19
20    // une variation d'OPENCV
21    Mat cvimghsi;
22    cv::cvtColor(imgcol,cvimghsi,cv::COLOR_BGR2HSV);
23    cv::namedWindow("opencv hsi", 1);
24    cv::imshow("opencv hsi", cvimghsi);
25
```

5.0.5 L'éxécutable à inserer

L'exécutable à insérer dans `apps/CMakeLists.txt`

```
1 add_executable(colourproc colourproc.cpp)
2 target_link_libraries(colourproc
3                 IMAGE_PROC_FNS    #la bibliotheque static
4                 ${OpenCV_LIBS} )
```