A LA DÉCOUVERTE DES IMAGES COULEUR.

1• OBJECTIF DE CE TRAVAIL.

Le travail qui vous est proposé ici consiste à manipuler des images couleurs et à découvrir les liens qui unissent les espaces de représentation RVB et HSV.

2• IMAGE COULEUR.

Une image couleur est composée de trois images de luminance dans des bandes de fréquences données. Vous pouvez préparer votre travail en essayant de recueillir des informations sur les différentes représentation des images.
Le principe des capteurs et des représentations des images couleurs est directement relié à la nature physiologique de l'œil (figure 1).

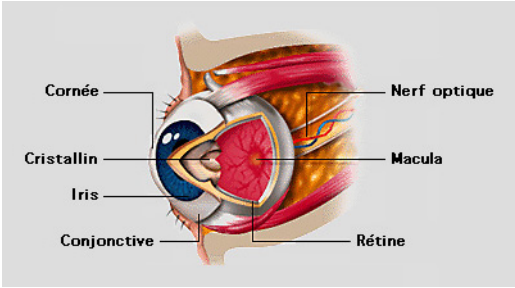

Figure 1 : l'œil.

La rétine de l'œil est composée de deux types de capteurs : les bâtonnets et les cônes. Les bâtonnets sont sensibles à l'intensité lumineuse dans la bande de fréquence dite *visible* (environ de 400 à 700 nm, figure 2) tandis que les cônes ont une sensibilité dont l'étendue fréquentielle est plus réduite (figures 3 et 4).

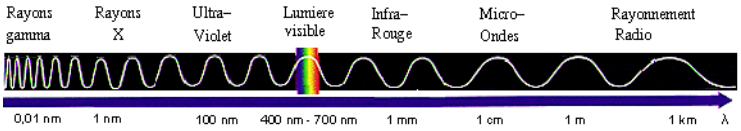

Figure 2 : lumière visible.

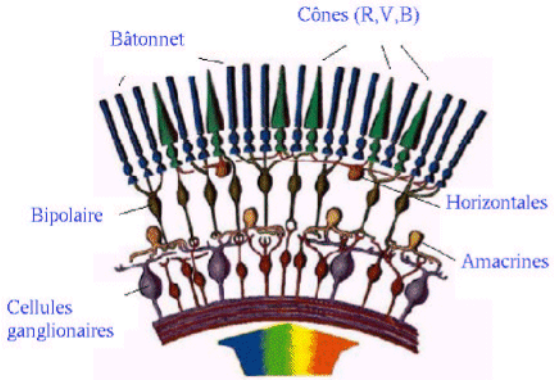

Figure 3 : capteurs de lumière.

Il existe trois types de cônes que nous appellerons les cônes R (rouge), V (vert) et B

(bleu). Les cônes bleus sont sensibles à des longueurs d'ondes autour de 450 nm, les cônes verts autour de 550 nm et les cônes rouges autour de 600 (figure 4).

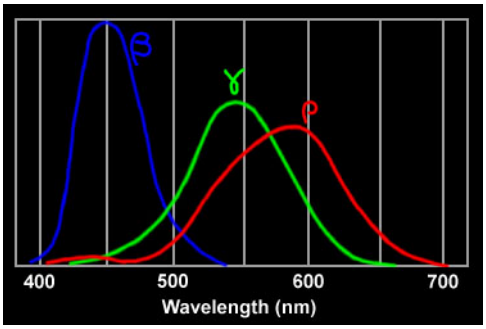

Figure 4 : sensibilité des cônes.

Lorsque l'oeil perçoit une longueur d'onde de 490 nm (ce qui est le cyan), les cônes B et G sont excité tandis que les cônes R sont au repos. Ce différentiel d'excitation est interprété par le cerveau comme étant la couleur cyan. Donc pour recréer cette illusion, il suffit de disposer d'un système projetant sur l'œil des ondes lumineuses bleues (450 nm) et de vertes (550 nm) dans des proportions équivalentes. C'est le principe de fonctionnement des écrans couleur.

A l'inverse, l'information nécessaire pour créer cette illusion peut être condensée en une proportion d'intensité lumineuse dans le rouge, le vert et le bleu. C'est le principe de la représentation RVB des images numériques couleurs.

3• Espace capteur / espace perceptuel.

Lorsqu'il s'agit de représenter la perception que l'on a d'une image couleur, l'espace RVB peut sembler inadéquat. On parle en effet des couleurs et non pas de leur interprétation en terme de RVB (seuls les imprimeurs le font et encore dans l'espace CJM). C'est pourquoi il est souvent utile, en traitement d'image, de passer de la représentation RVB à une représentation plus adéquate avec notre représentation interne de la perception des images couleurs. Un des espaces les plus utilisés est l'espace Luminance-Teinte-Saturation (en anglais Hue Saturation Value).

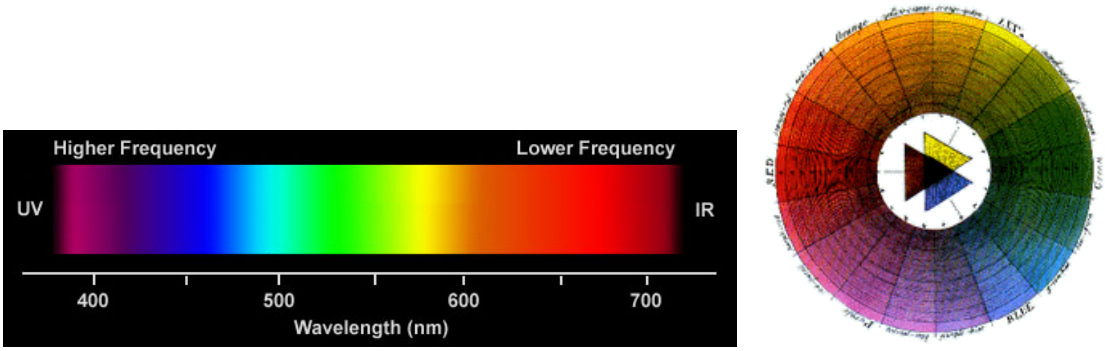

Figure 5 : spectre des couleurs et cercle chromatique.

- la luminance correspond à l'excitation des bâtonnets, c'est à dire à l'énergie moyenne qu'ils reçoivent. Elle est équivalente à la moyenne des énergies reçues pas chaque cône (pour l'œil) ou chaque photo-récepteur (pour le capteur d'images),

- la couleur, c'est la longueur d'onde émise par l'objet imagé, sa représentation est souvent cyclique et coïncide avec celle de l'arc-en-ciel (figure 5),
- la saturation, c'est l'aptitude à distinguer la couleur, c'est la *pureté* de la couleur (figure 6).

Figure 6 : saturation croissante du rouge.

4• MANIPULATIONS.

Vous disposez de l'image d'une illustration de Bilibine (peintre russe du début XX^{ème}). Cette image est très colorée et présente des aplats de différentes couleurs qui vont nous permettre de bien appréhender l'espace perceptuel.
Les différentes manipulations que nous allons effectuer sur cette image peuvent être résumées sur le schémas suivant (figure 7).

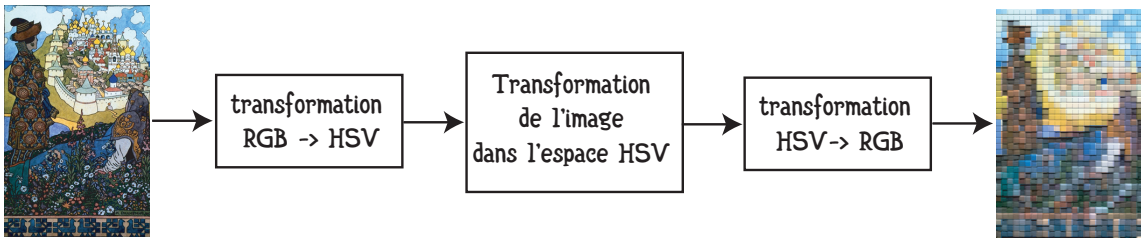

Figure 7 : chaîne de traitement dans l'espace HSV.

Ces manipulations vont consister à transformer l'image couleur RVB en une image HSV, d'effectuer une opération de transformation perceptuelle de l'image, puis de revenir à l'espace RVB pour afficher l'image résultat.

4.1• Manipulation d'une image.

Pour charger une image il suffit d'utiliser la fonction **imread** :

```
MonImage = imread('bilibin.jpg') ;
```

L'image est alors disponible sous la forme d'un tableau codé sur 8 bits.
Pour afficher une image :

```
figure(6) ; image(NouvelleImage) ; drawnow ;
```

Il est important de noter qu'on ne peut faire de calculs sur matlab que si les valeurs sont représentées en virgules flottante (sur 32 bits ou plus). Pour convertir l'image en format "virgule flottante" il faut faire :

```
MonImageDouble = double(MonImage) ;
```

L'image RVB est formée de Nlin lignes, Ncol colonnes et Nplan plans (ici 3 plans). Ces informations sont récupérables facilement grâce à :

```
[Nlin, Ncol, Nplan] = size(MonImage) ;
```

ATTENTION : on ne peut afficher une image que si elle est codée sur 8 bits !

4.2• Transformation RVB -> HSV.

C'est une transformation non linéaire, dont voici l'expression :

$$V = \max(R, G, B)$$

$$S = \frac{V - \min(R, G, B)}{V}$$

$$H = \frac{1}{6}\begin{cases} \dfrac{G - B}{V - \min(R, G, B)}, \text{ si } V = R \\ 2 + \dfrac{B - R}{V - \min(R, G, B)}, \text{ si } V = G \\ 4 + \dfrac{R - G}{V - \min(R, G, B)}, \text{ si } V = B \end{cases}$$

Elle est déjà présente sur matlab sous le nom : **rgb2hsv**.

4.3• Transformation HSV -> RGB.

Cette transformation est complexe. Elle est présente sur matlab sous le nom : **hsv2rgb**. Voici son algorithme en C :

```
if ( S == 0 )                  //HSL values = From 0 to 1
{
 R = L ;                     //RGB results = From 0 to 255
 G = L ;
 B = L ;
}
else
{
 if ( L < 0.5 ) var_2 = L * ( 1 + S ) ;
 else           var_2 = ( L +  S ) - ( S * L ) ;
 var_1 = 2 * L - var_2 ;
 R = Hue_2_RGB( var_1, var_2, H + ( 1 / 3 ) )
 G = Hue_2_RGB( var_1, var_2, H )
 B = Hue_2_RGB( var_1, var_2, H - ( 1 / 3 ) )
}
Hue_2_RGB( v1, v2, vH )          //Function Hue_2_RGB
{
 if ( vH < 0 ) vH += 1 ;
 if ( vH > 1 ) vH -= 1 ;
 if ( ( 6 * vH ) < 1 ) return ( v1 + ( v2 - v1 ) * 6 * vH ) ;
 if ( ( 2 * vH ) < 1 ) return ( v2 ) ;
 if ( ( 3 * vH ) < 2 ) return ( v1 + ( v2 - v1 ) * ( ( 2 / 3 ) - vH ) * 6 );
 return ( v1 ) ;
}
```

Elle suppose que les valeurs RGB comme HSV sont codées entre 0 et 1. Sinon, il faut faire la conversion [0, 255] <-> [0, 1]. Sous matlab, les valeurs RGB sont codées en [0,255] tandis que la transformation RGB->HSV code en [0,1].
Dans un premier temps, regardez les trois plans HSV indépendamment et essayez com-

prendre ces images. Pour extraire un plan particulier il faut écrire :

```
MonImageHSV = rgb2hsv(MonImage) ;
MaTeinte = MonImageHSV(:,:,1) ; % par exemple
```

4.4• Rotation de la table des couleurs.

On vous propose de faire tourner la table des couleurs. Pour cela, il vous suffit de remplacer, pour chaque pixel, la valeur de teinte par une valeur supérieure. Par exemple, vous pouvez diviser l'espace des teintes en 100 valeurs et remplacer, pour chaque pixel, H par H+deltaH ; en faisant attention à bien obtenir la rotation des couleurs c'est à dire que si H+deltaH est supérieur à 1 il faut lui retrancher 1 (figure 9).

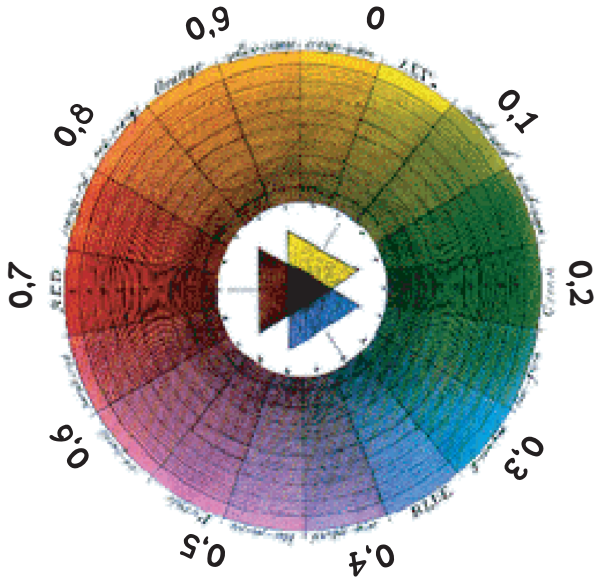

Figure 8 : espace des teintes.

Je vous propose donc de faire une boucle avec un incrément de 0.05, de créer des images pour lesquelles vous faites tourner la représentation colorée et d'afficher ces images les unes à la suite des autres. Vous devez obtenir un film dans lequel les couleurs semblent migrer dans l'image.

4.5• Modification de l'illumination et de la saturation.

Faites varier l'illumination d'abord, puis la saturation par pallier de 0.1.

Exemple pour l'illumination :

```
for deltaV=-1:0.1:1
    Luminance(:,:) = ImageHSV(:,:,2) + deltaS ;
    Luminance = min(Luminance,1) ; Luminance = max(Luminance,0) ;
    ... % recomposition de l'image et affichage
end
```

Constatez les modifications.

4.6• Segmentation en fonction de la couleur.

Récupérez la valeur de la teinte dans une zone (utilisez pour cela la fonction **ginput**) puis mettez à 0 tous les pixels de l'image qui ont une valeur de teinte éloignée de plus de 0.1 de la valeur sélectionnée. Regardez ce qui se passe si vous changez le rayon du voisinage dans l'espace H.

4.7• Segmentation en fonction de la saturation.

Mettez à 0 tous les pixels dont la saturation est inférieure à 0.8.

4.8• Segmentation dans l'espace HSV et RVB.

Considerez les espaces RVB et HSV comme des espaces à 3 dimensions. Définissez une distance de voisinage et mettez a 0 tous les pixels dont la valeur dans l'espace RVB ou HSV est éloignée (au sens de cette distance) de votre valeur de référence. Voyez si vous réussissez facilement à réaliser le même type de segmentation les deux cas. Essayez par exemple de segmenter uniquement les casques des guerriers de l'image bilibinWar.jpg.

4.9• Egalisation d'histogramme sur les images couleurs.

Egalisez l'histogramme de la composante valeur (V) de l'image couleur. Affichez l'image. Faîtes parallement une égalisation de chaque composante de l'image RVB. Comparez les résultats.

4.10• Croissance de région dans l'image couleur.

Il s'agit de l'algorithme qui porte le surnom de *baguette magique* dans les logiciels de retouche photo. Il consiste à créer une région connexe de pixels ayant des *couleurs* proches. On vous propose de le programmer. Il se déroule en sept étapes :

1) sélection du point de référence, définition de la tolérance (delta), création de l'image binaire des points déjà visités.

```
PointDeDepart = ginput(1) ;
PointDeDepart = round(PointDeDepart) ;
lin = PointDeDepart(2) ; col = PointDeDepart(1) ;
PointDeDepart(1) = lin ; PointDeDepart(2) = col ;
delta = 10 ;
PointDejaVisite = zeros(Nlin, Ncol) ;
```

2) ajout du point dans une liste (vide au départ),

```
ListeDePoint(1:2,1) = PointDeDepart(1:2) ;
PointCourant = PointDeDepart ;
NombreDePoints = 1 ;
PointDejaVisite(PointCourant) = 1 ;
numero = 1 ;
```

3) début de l'algorithme

4) pour chaque point de la liste, regarder un des points du voisinage

```
for u=-1:1
 for v=-1:1
  PointVisite(1) = PointCourant(1) + u ;
  PointVisite(2) = PointCourant(2) + v ;
```

5) s'il a déjà été visité passer au point suivant

```
  if(PointDejaVisite(PointVisite) < 1 )
   PointDejaVisite(PointVisite) = 1 ;
```

6) si la distance dans l'espace (RGB ou HSV) est inférieurs à la tolérance (delta) ajouter le point a la liste

```
   ListeDePoint(1:2,numero) = PointVisite(1:2) ;
   numero = numero + 1 ;
```

7) si la liste est terminée, sortir de l'algorithme.

Mettez alors à 0 tous les points qui ne sont pas sur la liste pour voir l'effet de votre croissance de région. Attention aux bords de l'image !!!