Représenter les images

prévision : 4/12 et 11/12

1 Le principe de base du codage d'une image

Pour coder une image, on dessine un quadrillage sur l'image.
On attribue à chaque cellule du quadrillage un code numérique correspondant à sa couleur (ou à sa couleur dominante ou à une "moyenne" de ses couleurs).
Les couleurs étant codées numériquement (comme toute information pour un ordinateur), on obtient ainsi une grille de nombres représentant l'image.
Pour vous convaincre de cette "structure de grille", télécharger une photo (au format jpg par exemple), l'ouvrir avec GIMP et zoomer jusqu'à voir apparaître une succession de petits carrés : chaque carré correspond à un pixel, monochrome.
Un peu de lecture :
– principe de base pour le codage d'une image : http://blog.kleinproject.org/?p=719&lang=fr#more-719
– plus général : http://interstices.info/jcms/c_16351/vision-artificielle-et-traitement-dimages

2 Création d'un fichier image

Exercice 1

1. Exécuter le programme suivant enonces/A/cree.py

Python

```python
# -*- coding: utf-8 -*-

largeur=70
hauteur=70
n=largeur*hauteur

#ouverture d'un fichier texte en mode écriture :
f=open("coucou.txt","w")

#on écrit P1 sur la ligne 1 du fichier :
f.write("P1")
# on passe à la ligne :
f.write("\n")
# on écrit la valeur de largeur, espace, la valeur de hauteur :
f.write(str(largeur)+" "+str(hauteur))


# pour chaque ligne :
for j in range(hauteur):
        # passage à la ligne :
    f.write("\n")
    # pour chaque colonne :
    for i in range(largeur):
        # si j est pair
        # on inscrit 0
        if j%2==0:f.write("0" )
        # sinon on inscrit 1
        else: f.write("1")

# on ferme le fichier:
f.close()
```

2. Observer le contenu du fichier texte ainsi créé (avec un éditeur de texte). Faire le lien avec le code python.
3. Consulter les propriétés du fichier (clic droit). Quel est le type de ce fichier ?
4. Effacer l'extension txt du fichier coucou.txt. Consulter les propriétés du fichier. Quel est maintenant son type ?
5. Ouvrir le fichier avec un visionneur d'images.
6. Chercher sur le web la définition du format pbm (portable bit map) et expliquer pourquoi, sans l'extension txt, l'OS a identifié une image. On pourra consulter la page http://fr.wikipedia.org/wiki/Portable_pixmap.
7. Quelle est la largeur en pixels de l'image ? Quelle est sa hauteur en pixels ? Combien faut-il de valeurs numériques pour coder un pixel dans ce format d'image ?
8. Modifier le code du programme python pour que l'image produite ait une largeur de 10 pixels et une hauteur de 30 pixels. Observer l'image obtenue et son code.
9. Sur le fichier définissant l'image de largeur 10 et hauteur 30, changer quelques lignes de 0 par une ligne de 1 ("à la main"). Observer l'effet de cette modification.
10. Modifier le programme python afin que l'image obtenue ne soit pas une alternance de lignes noires et lignes blanches mais une alternance de colonnes noires et colonnes blanches. On pourra également remplacer l'extension txt par l'extension pbm afin d'obtenir directement un fichier image.
11. Modifier le programme python afin d'obtenir l'image d'un carré.
12. Modifier le programme python afin d'obtenir l'image d'un cercle.

Une résolution

1.

2.
3. type texte.
4. type image PBM.
5.
6. Lorsqu'on a oté l'extension txt, ce sont les premières lignes du fichier qui sont utilisées pour la reconnaissance du type. Le début de notre fichier correspond à la définition d'un fichier PBM ASCII.
7. largeur et hauteur sont celles indiquées par le code du programme (70 pixels en largeur, 70 en hauteur). Dans ce format, un pixel est codé par un seul chiffre : 0 ou 1, correspondant à blanc ou noir.
8. Il suffit de modifier largeur et hauteur dans le programme python. L'objectif est de faire comprendre le rôle de ces instructions. `corriges/A/rectangle.py`
9. L'objectif est de bien comprendre la correspondance codage – pixel de ce format.
10. Modification du texte du programme pour obtenir alternances en colonnes plutôt qu'en lignes `corriges/A/colonnes.py` :

Python

```
# -*- coding: utf-8 -*-

largeur=70
hauteur=70
n=largeur*hauteur

#ouverture d'un fichier texte en mode écriture :
f=open("coucou.pbm","w")

#on écrit P1 sur la ligne 1 du fichier :
f.write("P1")
# on passe à la ligne :
f.write("\n")
# on écrit la valeur de l, espace, la valeur de h :
f.write(str(largeur)+" "+str(hauteur))

for j in range(hauteur):
    f.write("\n")
    for i in range(largeur):
        if i%2==0:f.write("0" )
        else: f.write("1")

# on ferme le fichier:
f.close()
```

11. Un code possible (carré) `corriges/A/carre.py` :

Python

```python
# -*- coding: utf-8 -*-

largeur=70
hauteur=70
n=largeur*hauteur

#ouverture d'un fichier texte en mode écriture :
f=open("coucou.pbm","w")

#on écrit P1 sur la ligne 1 du fichier :
f.write("P1")
# on passe à la ligne :
f.write("\n")
# on écrit la valeur de l, espace, la valeur de h :
f.write(str(largeur)+" "+str(hauteur))


# côté /2
r=20//2

for j in range(hauteur):
    f.write("\n")
    for i in range(largeur):
        if max(abs(i-largeur//2),abs(j-hauteur//2))==r:f.write("1" )
        else: f.write("0")

# on ferme le fichier:
f.close()
```

12. Le problème de la discrétisation est mis en évidence par les discontinuités observées sur la figure obtenue.
    Un code possible (cercle) `corriges/A/cercle.py` :

Python

```
# -*- coding: utf-8 -*-

largeur=70
hauteur=70
n=largeur*hauteur

#ouverture d'un fichier texte en mode écriture :
f=open("coucou.pbm","w")

#on écrit P1 sur la ligne 1 du fichier :
f.write("P1")
# on passe à la ligne :
f.write("\n")
# on écrit la valeur de l, espace, la valeur de h :
f.write(str(largeur)+" "+str(hauteur))


# carre du rayon :
r=10**2

for j in range(hauteur):
    f.write("\n")
    for i in range(largeur):
        if (i-largeur//2)**2+(j-hauteur//2)**2 in range(r-10,r+11):f.write("1" )
        else: f.write("0")

# on ferme le fichier:
f.close()
```

Choix d'un intervalle autour de r^2 pour "lisser". □

Exercice 2

1. Dans le dossier enonces/B, vous trouverez une photo de l'entrée du lycée de la Plaine de l'Ain.

Avec le logiciel GIMP, enregistrer cette photo au format pbm ascii (utiliser "enregistrer sous" et choisir le format adapté). Ouvrir ensuite le fichier avec un éditeur de texte (modifier éventuellement l'extension en txt).

2. En consultant les propriétés du fichier image (ou du fichier txt), lire la taille du fichier (en octets). Expliquer la taille de ce fichier à l'aide des dimensions (en pixels) de l'image.
3. Une ligne du fichier pbm ne doit pas dépasser 70 caractères. Lorsque la largeur de l'image est supérieure à 70, quelle information du fichier texte ascii permet au logiciel lisant l'image de "savoir" où s'arrête le code de chaque rangée horizontale de pixels de l'image ?
4. Reprendre la photo de départ. Avec le logiciel GIMP, la transformer en pbm binaire.

5. Comparer la taille du fichier pbm ascii avec celle du fichier pbm binaire. Expliquer.
6. Pour visualiser le code du fichier pbm codé en binaire, utiliser un éditeur hexadécimal (par exemple le logiciel GHex). Vous lisez par exemple 3F dans la partie correspondant au codage des pixels. Que signifie ce 3F ?

Une résolution

1. Exemple dans le dossier corriges/B.
2. Mon image fait environ 88 ko. La largeur de l'image, lue dans le fichier txt, est de 340 pour une hauteur de 255. Ce qui fait $340 \times 255 = 86700$ octets (puisque chaque pixel est codé par un caractère ascii, ce qui occupe un octet). Si on ajoute les octets de l'entête et ceux des changements de ligne, on trouve les 88 ko annoncés.
3. La largeur est indiquée dans l'entête, le logiciel lit donc 'largeur' caractères à la suite plutôt que ligne par ligne. On pourrait revenir ici aux choix faits dans les premiers fichiers créés en ne mettant qu'un seul caractère par ligne par exemple pour constater que "ça marche".
4.
5. En ascii, un caractère est codé par un octet, soit 8 bits. Comme les caractères utilisés sont des 0 et des 1, un seul bit devrait suffire : c'est ce qu'il se passe dans le format pbm codé en binaire. La taille est donc divisée par 8.
6. Les bits sont regroupés en paquet de 8 pour former un octet (cf définition du format pbm binaire à la page http://fr.wikipedia.org/wiki/Portable_pixmap). 3F a pour écriture binaire 00111111, il s'agit donc du codage de 8 pixels (2 blancs, suivis de 6 noirs). □

3 Les formats d'images

Il existe de nombreux formats d'images : pbm, pgm, ppm, png, jpeg, gif, ps, pict, tiff ... Vous pourrez vous faire une idée des définitions de ces formats en cherchant sur le web.
Un point de départ possible (qui ne concerne pas que les images) :
http://fr.wikipedia.org/wiki/Liste_d%27extensions_de_fichiers
Des caractéristiques distinguant les différents formats d'images :
– Noir et blanc, en niveaux de gris, en couleurs.
– Vectorielle ou bitmap.
– Image compressée ou non.
– Formats publics ou non. Certains concepteurs gardent leur format secret afin de contraindre les utilisateurs à utiliser les logiciels du concepteur pour traiter ces images. D'autres formats ont au contraire une définition publique.
– Formats propriétaires ou libres. Pour certains formats, des droits doivent être payés aux concepteurs, pour d'autres non.
Le format pbm est par exemple libre, public, non compressé, bitmap, noir et blanc.

Exercice 3
Donner (recherche web) les caractéristiques des formats gif et png.
gif : Graphics Interchange Formats.
png : Portable Network Graphics.

Une résolution
Pour le png, consulter http://ptaff.ca/png/ ou encore http://www.w3.org/Graphics/PNG/ ou http://www.libpng.org/pub/png/.
Pour le gif : http://fr.wikipedia.org/wiki/Graphics_Interchange_Format. □

4 Images en niveaux de gris

On a vu précédemment le format pbm qui est un format codant les images en noir et blanc.
Certains formats, comme le format pgm (portable greymap), utilisent des nuances de gris.

Rappelons la structure d'un fichier PBM :
- P1 est inscrit sur la ligne 1 (code ascii) ou P4 (code binaire).
- En ligne 2 : la largeur de l'image suivie d'un caractère d'espacement, suivie de la hauteur de l'image (en pixels).
- Sur les lignes suivantes : la liste des couleurs des pixels (ligne par ligne, de haut en bas et de droite à gauche), chaque ligne compte au plus 70 caractères (on peut donc inscrire entre 1 et 70 caractères par ligne de codage ascii, une ligne ascii ne correspond pas à une ligne de pixels de l'image, une ligne de pixels correspond à "largeur" caractères pris à la suite).
- On peut ajouter des commentaires (ligne commençant par le symbole #).

Le format PGM (codé en ascii) est construit sur le même modèle :
- P2 est inscrit sur la ligne 1 (P5 pour le pgm binaire).
- En ligne 2 :la largeur de l'image suivie d'un espace, suivie de la hauteur de l'image (en pixels).
- En ligne 3, la valeur maximale utilisée pour estimer les niveaux de gris, par exemple 255. 255 correspond alors au blanc, 0 au noir et les valeurs entre 1 et 254 à différents gris du plus foncé au plus clair.
- Sur les lignes suivantes : la liste des couleurs des pixels (c'est à dire des entiers entre 0 et la valeur indiquée en ligne 3) séparés par des espaces ou des retours à la ligne, ligne par ligne, de haut en bas et de droite à gauche, chaque ligne compte au plus 70 caractères.
- On peut ajouter des commentaires (ligne commençant par le symbole #).

Exercice 4

1. Reprendre la photo du lycée précédemment transformée. Avec le logiciel GIMP, l'enregistrer sous le format pgm ascii. Ouvrir le fichier avec un éditeur de textes.
2. Quelle est la taille (en ko) de cette image ? Justifier (approximativement) cette taille en tenant compte de la dimension en pixels.

Une résolution

1.
2. Mon image a un poids de 309 ko environ pour 340 sur 255 pixels. Chaque pixel est codé par un entier entre 1 et 3 chiffres suivi d'un caractère espace ou passage à la ligne. Soit entre 2 octets et 4 octets par pixel. $340 \times 255 \times 4 = 346800$ (soit 346 ko). □

Exercice 5

Écrire un programme python :
- input : un fichier pgm ascii.
- output : un fichier pgm ascii aux teintes de gris inversées. Si la valeur maximale utilisée est 255, chaque valeur x sera donc remplacée par $255 - x$.

On donne ci-dessous quelques éléments techniques. Compléter.

Python

```
# ouverture du fichier source en mode lecture (dans le même dossier que le .py) :
fsource = open("NomDuFichierSource.pgm","r")
# ouverture d'un nouveau fichier (en mode écriture)
# qui contiendra au final le code de l'image aux gris inversés :
fbut=open("NomDuFichierBut.pgm","w")
# on crée une liste des lignes du fichier initial :
L=fsource.readlines()
# on recopie dans le fichier but les lignes de l'entête (cas où l'entête occupe 4 lignes :
# P2, commentaire, dimensions en pixels, valeur max correspondant au blanc)
for i in range(4):
    fbut.write(L[i])

 # inversion des codes correspondant aux teintes de gris :
 ...

# fermeture des fichiers :
fsource.close()
fbut.close()
```

L'effet attendu pourra être visualisé avec les fichiers du dossier enonces/C.

Une résolution

cf dossier corriges/C.

Le programme python suivant est utilisé avec un fichier pgm codé en ascii comportant 4 lignes d'entête , chaque pixel est ensuite codé par un entier entre 0 et 255, un entier par ligne.

Python

```
# -*- coding: utf-8 -*-

def inversion(valeurlue):
    return str(255-int(valeurlue))

# ouverture du fichier source :
fsource = open("AmberLycASCII.pgm","r")
# ouverture d'un nouveau fichier
# qui contiendra au final le code de l'image aux gris inversés :
fbut=open("AmberInverse.pgm","w")
# on crée une liste des lignes du fichier initial :
L=fsource.readlines()
# on recopie dans le fichier but les lignes de l'entête :
for i in range(4):
    fbut.write(L[i])
# toutes les lignes suivantes contiennent un nombre à inverser :
for i in range(4,len(L)):
    fbut.write(inversion(L[i]))
    fbut.write("\n")
# fermeture des fichiers :
fsource.close()
fbut.close()
```

5 Un format avec couleurs

5.1 Le principe de la formation des couleurs sur un écran.

Notre oeil contient des cellules (les cônes) sensibles à la couleur (longueur d'onde) de la lumière qu'ils reçoivent. Ces cônes sont de trois sortes, de maximum de sensibilité respectivement dans le rouge, le vert et le bleu. Quelle que soit la lumière percue, l'oeil ne communique au cerveau que l'intensité de la réaction des cônes. Deux lumières distinctes mais stimulant les trois types de cônes de la même façon nous seront donc indiscernables (par exemple, le mélange de lumière rouge et verte peut donner à l'oeil la même sensation qu'une lumière jaune).
Sur l'écran d'un ordinateur, chaque pixel est composé de trois sources de lumière (rouge, verte, bleue) et en faisant varier l'intensité de chacune d'elles, on peut simuler un grand nombre de couleurs.
Si, comme certains animaux, nous avions quatre type de cônes, nos écrans devraient être conçus avec quatre sources de lumière.

5.2 Le format ppm

Le format ppm (portable pixel map) est un format (parmi beacoup d'autres) concernant les images couleurs.
Dans ce format, on associe à chaque pixel trois nombres entre 0 et 255 correspondant à l'intensité des couleurs rouge, vert, bleu (RGB).
Consulter la page `http://fr.wikipedia.org/wiki/Portable_pixmap` pour la définition du format ppm.

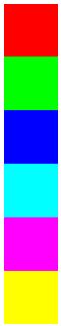
Exercice 6
Lire, exécuter et comprendre le fichier `enonces/F/couleurs.py` dont le code est rappelé ci-dessous.

Python

```python
# -*- coding: utf-8 -*-


largeur=100
hauteur=600
n=largeur*hauteur

#ouverture d'un fichier en mode écriture :
f=open("couleurs.ppm","w")

#on écrit P3 sur la ligne 1 du fichier :
f.write("P3")
# on passe à la ligne :
f.write("\n")
# on écrit la valeur de largeur, espace, la valeur de hauteur :
f.write(str(largeur)+" "+str(hauteur))
f.write("\n")
# valeur max pour l'intensité des couleurs :
f.write("255")
f.write("\n")


for j in range(hauteur//6):
        for i in range(largeur):
                f.write("255 0  0")
                f.write("\n")

for j in range(hauteur//6):
        for i in range(largeur):
                f.write("0 255  0")
                f.write("\n")

for j in range(hauteur//6):
        for i in range(largeur):
                f.write("0 0 255")
                f.write("\n")

for j in range(hauteur//6):
        for i in range(largeur):
                f.write("0 255 255")
                f.write("\n")

for j in range(hauteur//6):
        for i in range(largeur):
                f.write("255 0 255")
                f.write("\n")

for j in range(hauteur//6):
        for i in range(largeur):
                f.write("255 255 0")
                f.write("\n")

# on ferme le fichier:
f.close()
```

Une résolution

L'objectif est de comprendre ou de réviser le principe d'un codage RVB.
L'image obtenue est :



□

Exercice 7

Quelle sera l'image créée par le programme python ci-dessous ?

Python

```python
# -*- coding: utf-8 -*-

largeur=50
hauteur=256
n=largeur*hauteur

#ouverture d'un fichier en mode écriture :
f=open("coucou.ppm","w")

#on écrit P3 sur la ligne 1 du fichier :
f.write("P3")
# on passe à la ligne :
f.write("\n")
# on écrit la valeur de largeur, espace, la valeur de hauteur :
f.write(str(largeur)+" "+str(hauteur))
f.write("\n")
# valeur max pour l'intensité des couleurs :
f.write("255")
f.write("\n")

for j in range(hauteur):
    for i in range(largeur):
        f.write(str(j)+" 0"+" 0")
        f.write("\n")


# on ferme le fichier:
f.close()
```

Fichier enonces/D/D.py

Une résolution

Dégradé de rouge : en haut (ligne 1), le rouge est à 0 (noir) et en bas le rouge est à son intensité max (255).
cf corriges/D/degrade.py

□

Exercice 8

On veut créer l'image d'un carré noir plein de largeur 50 pixels et hauteur 50 pixels.

1. Donner la taille (en octets) du fichier obtenu si l'on définit l'image au format pbm ascii.
2. Donner la taille (en octets) du fichier obtenu si l'on définit l'image au format ppm ascii.

Une résolution

En négligeant le poids de l'entête :

1. $50 \times 50 = 2500$ caractères 0 ou 1. Si l'on change de ligne tous les 50 caractères, on ajoute une cinquantaine de caractères "retour à la ligne". Soit environ 2550 octets ($\approx 2{,}6$ ko).
2. Chaque pixel est maintenant codé par "0 0 0 ", soit 6 caractères (compter les espaces, ils sont nécessaires puisque les nombres peuvent être de plus d'un chiffre). $6 \times 50 \times 50 = 15000$. On passe à environ 15 ko pour la même image. Soit une multiplication du poids par environ 5,8. On comprend ici l'intérêt de choisir des formats adaptés au fichier manipulé. On comprend aussi qu'il existe divers formats de compression permettant de réduire les poids. □

Exercice 9

Dans le dossier enonces/E, vous trouverez une image de fleur au format ppm ascii.
La première ligne contient le code P3 correspondant à ce format.
La ligne 2 contient les dimensions largeur et hauteur en pixels de l'image : 800 688
La ligne 3 contient 255, valeur maximale utilisée pour coder les couleurs.
Chacune des lignes qui suit contient un nombre entre 0 et 255.

1. Quelle est la signification des nombres des lignes 4, 5 et 6 ?
2. Caractériser les numéros de ligne dont le numéro correspondant à la composante "vert" d'un pixel.
3. Expliquer comment calculer le nombre de lignes de ce fichier à partir des indications précédentes.
4. Ci-dessous un extrait de programme dont la boucle principale a été effacée. A vous de la réécrire.

Python

```python
# -*- coding: utf-8 -*-

# ouverture du fichier source en lecture:
fsource = open("rose.ppm","r")
# ouverture d'un nouveau fichier
# qui contiendra au final le code de l'image aux rouges et verts intervertis :
fbut=open("verte.ppm","w")
# on crée une liste des lignes du fichier initial :
L=fsource.readlines()

# on recopie dans le fichier but les lignes de l'entête :
for i in range(3):
    fbut.write(L[i])

# boucle à écrire intervertissant les composantes : les valeurs correspondants au rouge
    deviendront les valeurs pour le vert
# les valeurs du vert deviennent les valeurs du bleu et les valeurs du bleu deviennent
    celles du rouge.


# fermeture des fichiers :
fsource.close()
fbut.close()
```

Une résolution

cf dossier corriges/E.

1. Les lignes 4, 5, 6 contiennent les composantes rouge, vert, bleu du premier pixel (en haut à gauche).
2. Le rouge est sur les lignes de numéro multiple de 3, le vert sur les lignes dont le numéro a pour reste 1 modulo 3 et le bleu sur les lignes de reste 2.
3. 3 lignes d'entête + $800 \times 688 \times 3 = 1\,651\,203$ lignes.
4. La rose verte à herbe bleue `corriges/E/rose.py`

Python

```python
# -*- coding: utf-8 -*-

# ouverture du fichier source en lecture:
fsource = open("rose.ppm","r")
# ouverture d'un nouveau fichier
# qui contiendra au final le code de l'image aux rouges et verts intervertis :
fbut=open("verte.ppm","w")
# on crée une liste des lignes du fichier initial :
L=fsource.readlines()

# on recopie dans le fichier but les lignes de l'entête :
for i in range(3):
    fbut.write(L[i])

for i in range(3,len(L),3):
    a,b,c=L[i],L[i+1],L[i+2]
    fbut.write(c)
    fbut.write("\n")
    fbut.write(a)
    fbut.write("\n")
    fbut.write(b)
    fbut.write("\n")
# fermeture des fichiers :
fsource.close()
fbut.close()
```

□