RAPPORT DE PROJET D'ALGORITHMIQUE – IG3

Script de traitement d'images au format PGM en langage Python

Réalisé par

Joris MAILLET

Valmiro RIBEIRO DA SILVA

Année universitaire 2013-2014

Sommaire

Introduction 1
Contexte 1
Présentation 1
1 Cahier des charges 2
1.1 Lancement du script 2
1.2 Fonctions publiques 2
1.2.1 Spécifications fonctionnelles 2
1.2.2 Propriétés 3
1.3 Fonctions utilitaires privées 3
1.3.1 Spécifications fonctionnelles 3
2 Rapport technique 5
2.1 Démarche et choix de l'algorithme 5
2.1.1 Structures de données 5
2.1.2 Choix lors du traitement d'image 5
2.1.3 Interface utilisateur 5
2.2 Complexité 6
2.3 Description logique 6
2.3.1 Contenu des classes 6
2.3.2 Liste des fonctions 6
3 Rapport d'activité 20
3.1 Répartition des tâches 20
3.2 Difficultés rencontrées 20
4 Manuel d'utilisation 21
4.1 Lancement du script 21
4.2 Utilisation des fonctions 21
4.3 Paramètres obligatoires et optionnels 25
Annexe – Représentation physique I

Table des figures

Figure 1 – Canada.pgm 21
Figure 2 – Canada négatif 22
Figure 3 – Canada pivot 90 22
Figure 4 – Canada pivot moins 90 23
Figure 5 – Canada miroir vertical 23
Figure 6 – Canada miroir horizontal 24
Figure 7 – Canada redimensionné à 25% 24
Figure 8 – Canada contrastes automatiques 24
Figure 9 – Canada niveau de gris manuels 25

Introduction

Contexte

Le projet python intervient dans le cadre d'un TP d'algorithmique, lors du cinquième semestre du cycle d'ingénieur "Informatique et Gestion". Ce projet nous permet de nous familiariser avec un langage interprété que nous ne connaissions pas : Python. Celui-ci permet de fabriquer facilement des scripts à destination de l'utilisateur, ce que nous avons fait lors du développement de notre programme.

Présentation

Le projet consiste tout d'abord en la lecture et écriture d'images au format PGM. Ce format est reconnu dans la plupart des outils d'imagerie sous Unix, notamment la variante PGM raw que nous avons utilisé. Le projet inclus également un ensemble de fonctionnalités permettant leurs traitements : redimensionnements, rotations, gestion des contrastes.

Au final, l'utilisateur doit pouvoir disposer de l'ensemble des fonctions qu'il attend, et les utiliser de manière simple et efficace. Ce rapport de projet comporte les parties suivantes : un cahier des charges spécifiant l'ensemble des fonctionnalités disponibles. Ensuite, un rapport technique précisant : la démarche de l'algorithme, les choix des structures de données, et la complexité des algorithmes. Un rapport d'activité sera présent, où nous expliquerons notre répartition des tâches et nos difficultés rencontrées. Enfin, un manuel d'utilisation permettra à l'utilisateur de bien prendre en main le programme pour effectuer correctement le traitement des images PGM raw qu'il souhaite.

1 Cahier des charges

1.1 Lancement du script

L'utilisateur pourra lancer le script de traitement des images par le biais de l'invite de commandes, en lançant la commande "python" suivie du nom du script, puis l'ensemble des paramètres désirés. Il devra avoir à disposition une image au format PGM qui sera ensuite modifiée par le script.

1.2 Fonctions publiques

1.2.1 Spécifications fonctionnelles

L'ensemble des fonctionnalités disponibles sont résumées ci-dessous :

- Inverser les couleurs de l'image

fonction negatifPGMraw : image : Image → image : Image

- Pivoter une image de 90 degrés dans le sens des aiguilles d'une montre

fonction pivot90PGMraw : image : Image → image : Image

- Pivoter une image de 90 degrés dans le sens inverse des aiguilles d'une montre

fonction pivotMoins90PGMraw : image : Image → image : Image

- Retourner verticalement l'image

fonction miroirVerticalPGMraw : image : Image → image : Image

- Retourner horizontalement l'image

fonction miroirHorizontalPGMraw : image : Image → image : Image

- Modifier la taille d'une image

fonction scalePGMraw : image : Image, pourcentage : int → image : Image

L'appel de cette fonction renverra une erreur si l'argument obligatoire pour cette fonction n'est pas un entier strictement positif, ou s'il n'existe pas.

- Contraster automatiquement une image : optimise les contrastes de l'image de façon à avoir des niveaux de gris allant de 0 à 255

fonction contrasteAutoPGMraw : image : Image → image : Image

- Modifier manuellement les niveaux de gris d'une image

fonction niveauDeGrisPGMraw : image : Image, grisMin : int, grisMax : int → image : Image

L'appel de cette fonction renverra une erreur si les deux arguments obligatoires pour cette fonction ne sont pas des entiers, avec le premier entier inférieur ou égal au deuxième entier, ou s'ils n'existent pas.

Chaque fonction sera utilisable par le biais d'arguments et de paramètres. Par exemple, l'argument -ndg suivit des valeurs 20 et 200 exécutera la fonction niveauDeGrisPGMraw avec les valeurs 20 et 200. Le résultat de cette fonction est visible sur la figure N° ? du manuel d'utilisation.

Par ailleurs, un argument sera obligatoire : le nom du fichier à traiter. Il devra être inscrit à la fin de la ligne entrée par l'utilisateur. S'il n'est pas entré, le lancement du script provoquera une erreur. Il est possible de rajouter après le fichier source le nom du fichier de destination qui sera créé. Les deux fichiers doivent donc comporter l'extension ".pgm" lorsqu'ils sont écrits dans l'invite de commande.

1.2.2 Propriétés

- negatifPGMraw(negatifPGMraw(Image)) == Image

- pivot90PGMraw(pivotMoins90PGMraw(Image)) == Image

- miroirVerticalPGMraw(miroirVeticalPGMraw(Image)) == Image

- miroirHorizontalPGMraw(miroirHorizontalPGMraw(Image)) == Image

- scalePGMraw(Image,100) == Image

1.3 Fonctions utilitaires privées

D'autres fonctions sont présentes, mais sont utilisées uniquement par les fonctions de l'utilisateur. Ce sont des fonctions utilitaires qui permettent le bon déroulement des fonctions publiques.

1.3.1 Spécifications fonctionnelles

- Récupérer la valeur du pixel d'une image

fonction get_pixel : image : Image, i : int, j : int → pixel : int

- Renvoyer une nouvelle position d'un pixel, en fonction de la différence de largeur ou hauteur entre deux images et de la position initiale du pixel

fonction map : aux : int, w1 : int, w2 : int → res : int

- Lire un fichier au format PGMraw :

fonction lirePGMraw : chemin : String → image : Image

Erreur si le fichier ciblé par « chemin » n'est pas valide, ou n'existe pas.

- Ecrire un fichier au format PGMraw

fonction ecrirePGMraw : image : Image, chemin : String → Vide

- Vérifier si un fichier est au format PGMraw

fonction estPGMrawValide : path : String → estValide : booleen

Erreur si le fichier n'existe pas.

- Récupérer la dimension d'une ligne

fonction recupererDimension : ligne : String → dim : Dimension

Erreur si la ligne des dimensions ne comporte pas uniquement deux entiers strictement positifs.

- Récupérer la valeur entière d'une couleur depuis un caractère codé en ASCII

fonction recupererCouleur : couleurFichier : ASCII → couleur : int

- Ecrire un caractère codé en ASCII dans un fichier depuis une valeur entière d'une couleur

fonction ecrireCouleur : fichier : Fichier , couleur : int → Vide

Si la couleur passée n'est pas comprise entre 0 et 255, la fonction écrira la couleur « 0 ».

- Vérifier si une ligne est un commentaire, c'est à dire si elle commence par le caractère '#'

fonction estCommentaire : ligne : String → estCommentaire : booleen

- Récupérer une ligne depuis un fichier

fonction recupererLigne : fichier : Fichier, nblignes : int, sauterCommentaires : boolean → ligne : String

S'il n'y a plus de ligne à lire, la fonction renverra une chaine de caractères vide.

2 Rapport technique

2.1 Démarche et choix de l'algorithme

2.1.1 Structures de données

La première discussion s'est portée sur le choix de la structure de données de l'image. Puisque le nombre de pixels est connu, il paraît logique de stocker leurs valeurs dans un tableau. On dispose ainsi d'un accès séquentiel pour chaque pixel ce qui améliore grandement le temps d'accès aux variables, lors de la lecture et de l'écriture des phases de traitements.

Il aurait été possible de stocker l'ensemble des pixels de l'image dans un tableau à une dimension, puisque les couleurs du format PGM raw sont rangés dans une même ligne. Cependant, il est plus simple de travailler sur un tableau à deux dimensions, car l'on peut mieux se représenter l'image, et les modifications de chaque pixel est plus aisée. Le choix final s'est donc porté naturellement sur un tableau à deux dimensions comme structure de données de nos pixels de l'image.

A cela, nous avons introduit un type Dimension, qui permet de stocker sous forme d'entiers la largeur et la hauteur de l'image.

De ce fait, le type abstrait Image est composé :

- D'un tableau à deux dimensions contenant chaque pixel de l'image

- Un type Dimension comprenant la largeur et la hauteur de l'image

Une classe FileCorruptError est également présente. Elle hérite de la classe Exception, ce qui signifie qu'elle est utilisée pour lever des exceptions en cas d'erreur de fichier corrompu. Elle est notamment présente dans les fonctions qui gèrent le contrôle des fichiers, c'est-à-dire les fonctions estPGMrawValide, lirePGMraw, et recupererDimension.

2.1.2 Choix lors du traitement d'image

A chaque traitement, une nouvelle image est renvoyée. Nous avons fait ce choix lors de la spécification fonctionnelle pour que l'utilisateur puisse modifier la nouvelle image, mais également conserver la précédente s'il souhaite effectuer d'autres traitements. Avec la configuration actuelle du script de traitement, il n'est pas possible de modifier deux images lors d'une même exécution de script. Cependant, en vue d'une éventuelle mise à jour, avec notamment une interface demandant à chaque fois l'image que l'utilisateur souhaite modifier, cet aspect de la conception des algorithmes sera déjà opérationnel.

2.1.3 Interface utilisateur

L'interface utilisateur reprend les mêmes concepts d'un script de base, avec un passage d'arguments tels que "-help" ou "-ndg 0 255". Ceci permet de lancer des traitements

de manière très rapide, car il est possible en une ligne d'écrire l'ensemble des traitements mis à disposition à l'utilisateur. Chaque argument est lu l'un à la suite de l'autre, donc l'utilisateur va pouvoir enchaîner les arguments de sorte qu'il atteigne son objectif rapidement et simplement.

2.2 Complexité

Puisque le choix de notre structure de données s'est porté sur un tableau à deux dimensions, la plupart de nos fonctions ont une complexité de l'ordre de n^2. Plus précisément, de l'ordre de largeur*hauteur de l'image. On peut également dire que cette complexité correspond au nombre de pixels.

La complexité de chacun des algorithmes est présente à l'intérieur des fonctions, dans la description logique ci-dessous.

2.3 Description logique

2.3.1 Contenu des classes

Classe FileCorruptError :

messageError : String

Classe Dimension :

hauteur : int

largeur : int

Classe Image :

img : int[][]

dim : Dimension

2.3.2 Liste des fonctions

fonction negatifPGMraw (image : Image) : Image

#Donnée : image, l'image originale

#Résultat : une copie de l'image en négatif

int[][] tab = [image.dim.largeur][image.dim.hauteur]

image2 = Image(tab, Dimension(image.dim.hauteur, image.dim.largeur))

```
        pour i de 0 à image.dim.hauteur-1 faire
            pour j de 0 à image.dim.largeur-1 faire
                image2.img[i][j]=255-image.img[i][j]
            fin pour
        fin pour
        retourner image2
    fin
```

fonction pivot90PGMraw(image : Image) : Image

#Donnée : image, l'image originale

#Résultat : une copie de l'image pivoté de 90 degrés dans le sens des aiguilles d'une montre

```
        int[][] tab = [image.dim.largeur][image.dim.hauteur]
        image2 = Image(tab, Dimension(image.dim.largeur, image.dim.hauteur))
        pour i de 0 à image2.dim.hauteur-1
            pour j de 0 à image2.dim.largeur-1
                image2.img[i][j] = image.img[image.dim.hauteur-j-1][i]
            fin pour
        fin pour
        retourner image2
    fin
```

fonction pivotMoins90PGMraw(image : Image) : Image

#Donnée : image, l'image originale

#Résultat : une copie de l'image pivotée de 90 degrés dans le sens inverse des aiguilles d'une montre

```
        tab = [image.dim.largeur][image.dim.hauteur]
        image2 = Image(tab, Dimension(image.dim.largeur, image.dim.hauteur))
        pour i de 0 à image2.dim.hauteur-1
                pour j de 0 à image2.dim.largeur-1
                        image2.img[i][j] = image.img[j][i]
                fin pour
        fin pour
        retourner image2
fin
```

fonction miroirHorizontalPGMraw(image : Image) : Image

```
#Donnée : image, l'image originale
#Résultat : une copie de l'image retournée horizontalement
        int[][] tab = [image.dim.largeur][image.dim.hauteur]
        image2 = Image(tab, Dimension(image.dim.largeur, image.dim.hauteur))
        pour i de 0 à image.dim.hauteur-1
                pour j de 0 à image.dim.largeur-1
                        image2.img[i][j] = image.img[i][image.dim.largeur-j-1]
                fin pour
        fin pour
        retourner image2
fin
```

fonction miroirVerticalPGMraw(image : Image) : Image

```
#Donnée : image, l'image originale
```

#Résultat : une copie de l’image retournée verticalement

tab = [image.dim.largeur][image.dim.hauteur]

image2 = Image(tab, Dimension(image.dim.largeur, image.dim.hauteur))

pour i de 0 à image.dim.hauteur-1

pour j de 0 à image.dim.largeur-1

image2.img[i][j] = image.img[image.dim.hauteur-i-1][j]

fin pour

fin pour

retourner image2

fin

fonction niveauDeGrisPGMraw(image : Image, grisMin : int, grisMax : int) : Image

#Données : image, l’image originale, grisMin : le niveau de gris minimal de l’image à remettre à zéro, grisMax : le niveau de gris maximal à remettre à 255

#Résultat : une copie de l’image avec le nouveau niveau de gris spécifié

tab = [image.dim.largeur][image.dim.hauteur]

image2 = Image(tab, Dimension(image.dim.largeur, image.dim.hauteur))

si(grisMin > grisMax) alors

afficher(La valeur minimale du niveau de gris doit être inferieure à la valeur maximale. L'image ne sera pas modifiée")

pour i de 0 à image.dim.hauteur

pour j de 0 à image.dim.largeur

image2.img[i][j] = image.img[i][j]

fin pour

fin pour

```
            retourner image2
        fin si
        float moy = (float)((grisMin+grisMax)/2.0)
        float distMax = (float)(255-grisMax)
        float distMin = (foat)(0+grisMin)
        pour i de 0 à image.dim.hauteur
            pour j de 0 à image.dim.largeur
                si image.img[i][j]>=moy :
                    coefficient = ( (float)(image.img[i][j] - moy) / (float)(grisMax - moy) )
                    difference =  (int)(coefficient * distMax)
                sinon :
                    coefficient = ( (float)(moy-image.img[i][j]) / (float)(moy-grisMin) )
                    difference = (int)(-coefficient * distMin)
                fin si
                image2.img[i][j] = image.img[i][j] + difference
                si (image2.img[i][j] < 0):
                    image2.img[i][j] = 0
                fin si
                si (image2.img[i][j] > 255):
                    image2.img[i][j] = 255
                fin si
            fin pour
        fin pour
        retourner image2
```

fin

fonction contrasteAutoPGMraw(image : Image) : Image

#Donnée : image, l'image originale

#Résultat : une copie de l'image avec un niveau de gris optimisé

```
int maxi = 0
int mini = 255
pour i de 0 à image.dim.hauteur-1 faire
    pour j de 0 à image.dim.largeur faire
        si image.img[i][j]>maxi alors
            maxi = image.img[i][j]
        fin si
        si image.img[i][j]<mini alors
            mini = image.img[i][j]
        fin si
    fin pour
fin pour
retourner niveauDeGrisPGMraw(image,mini,maxi)
```

fin

fonction map(aux : int, w1 : int, w2 : int) : float

#Données : aux : la position (x ou y) de base d'un pixel, w1 : une longueur (largeur ou hauteur) originale, w2 : une longeur (largeur ou hauteur) de destination

#Résultat : res de type float, la position (x ou y) finale d'un pixel

```
float res = float((aux * w1)/ float(w2))
retourner res
```

fin

fonction get_pixel(image : Image , i : int, j : int) : int

#Donnée : image, l'image, i : la position x du pixel, j : la position y du pixel

#Résultat : le pixel correspondant au (x,y) spécifié

si (i < 0) alors

i = 0

si (i >= image.dim.hauteur) alors

i = image.dim.hauteur - 1

si (j < 0) alors

j = 0

si (j >= image.dim.largeur) alors

j = image.dim.largeur - 1

retourner image.img[i][j]

fin

Fonction scalePGMraw(image : Image, percentage : float) : Image

#Donnée : image, l'image originale, percentage : le pourcentage de redimensionnement

#Résultat : une copie de l'image redimensionnée

crée une nouvelle matrice et une nouvelle dimension pour la nouvelle image

tab = [image.dim.largeur*percentage][image.dim.hauteur*percentage]

dim = Dimension((int (image.dim.hauteur * percentage)),(int (image.dim.largeur * percentage)))

crée la nouvelle image

image2 = Image(tab,dim)

[p1,...,p4] sont les pixels qu'on regarde pour créer le nouveau pixel, p5, pour la nouvelle image

int p1 = 0; p2 = 0; p3 = 0; p4 = 0; p5 = 0;

ii et jj sont variables pour prendre des pixels de l'image donnée et les mettre sur la nouvelle image

int ii = 0; jj = 0

#Invariant: à la n-eme itération, toutes les lignes < i de l'image2 auront les nouveaux pixels

pour i de 0 à image2.dim.hauteur -1 faire

Invariant: à la n-eme itération dans la ligne courante, toutes les positions < j de l'image2 auront les nouveaux pixels

pour j de 0 à image2.dim.largeur - 1

#prendre des pixels des lignes et des colonnes pour créer la nouvelle image

ii = map(i, image.dim.hauteur, image2.dim.hauteur)

jj = map(j, image.dim.largeur, image2.dim.largeur)

#prendre 4 pixels de la image donnée pour faire l'interpolation

p1 = get_pixel(image, ii, jj)

p2 = get_pixel(image, ii + 1, jj)

p3 = get_pixel(image, ii, jj + 1)

p4 = get_pixel(image, ii + 1, jj + 1)

#l'interpolation. le nouveau pixel aura la moyenne des autres pixels

p5 = (p1 + p2 + p3 + p4) / 4

#ajouter le nouveau pixel a la nouvelle image

```
                    image2.img[i][j] = p5
            fin pour
        fin pour
        retourner image2
fin
```

fonction estPGMrawValide (path: String) : booleen

```
#Donnée : path, le chemin d'accès vers le fichier
#Résultat : estValide, un booléen qui indique si le fichier est au format PGM
        fichier = ouvrir(path, accès en lecture)

        #on récupère la 1ère ligne valide
        ligneSuivante = recupererLigne(fichier, 1, vrai)
        si (ligneSuivante != "P5\n")
            retourner faux
        fin si

        #on recupere la 2eme ligne valide
        ligneSuivante = recupererLigne(fichier, 1, vrai)
        aux = recupererDimension (ligneSuivante)
        si (aux.largeur == -1)
            retourner Faux
        fin si

        #on recupere la 3ème ligne
```

```
ligneSuivante = recupererLigne(fichier, 1, vrai)
si (ligneSuivante != “255\n”)
        retouner Faux
fin si

ligneSuivante = recupererLigne(fichier, 1, faux)
cpt = 0
#en python on aura besoin de savoir si on compte ou pas le caractère du fin de ligne
tant que (longueur(lignesuivante > 0))
        cpt = cpt + longueur(ligneSuivante)
        ligneSuivante = recupererLigne(fichier, 1, faux)
fin tant que
si cpt != aux.hauteur*aux.largeur
        retourner faux
retourner vrai
fin
```

fonction lirePGMraw (chemin : String) : Image

```
#Donnée : chemin, le chemin d’accès au fichier
#Résultat : image, le type abstrait contenant les données du fichier PGM
si(non estPGMrawValide(chemin)) alors
        ERREUR
fichier = ouvrir(chemin,acces en lecture)
ligne = 0
ligneSuivante = recupererLigne(fichier,2,Vrai)
```

```
        dimension = recupererDimension(ligneSuivante)
        img = [dimension.hauteur][dimension.largeur]
        ligneSuivante = recupererLigne(fichier,1,Faux)
        pour i de 0 à dimension.largeur*dimension.hauteur faire
            img[i / dimension.largeur][i % dimension.largeur] = recupererCouleur(fichier.lireProchainCaractere())
        fin pour
        fermer(fichier)
        retourner Image(img,dimension)
    fin
```

fonction ecrirePGMraw (image : Image, chemin : String)

```
    #Données : image : l'image à ecrire, chemin : le chemin pour écrire l'image
    #Résultat : Vide
        fichier = ouvrir(chemin,acces en ecriture)
        ecrire(fichier,
                "P5\n"+
                image.dim.hauteur+" "+image.dim.largeur+"\n"+
                "255\n")
        pour i de 0 à image.dim.hauteur-1
            pour j de 0 à image.dim.largeur-1
                ecrireCouleur(fichier,image[i][j])
            fin pour
        fin pour
        afficher("Le fichier " + chemin + "a bien été enregistre")
    fin
```

fonction recupererDimension (ligne : String) : Dimension

#Donnée : ligne, la ligne des dimensions de l'image

#Résultat : dimension, les dimensions de l'image

```
compteurCaractere = 0
tant que(compteurCaractere < longueur(ligne) et ligne[compteurCaractere] !=''):
    compteurCaractere = compteurCaractere + 1
fin tant que
si (compteurCaractere == longueur(ligne)) alors
    ERREUR
fin si
largeur = (int)(ligne[0 : compteurCaractere])
hauteur = (int)(ligne[compteurCaractere : longueur(ligne)])
si(largeur <= 0 ou hauteur <= 0) alors
    ERREUR
fin si
retourner Dimension(hauteur,largeur)
fin
```

fonction estCommentaire(ligne : String) : boolean

#Donnée : ligne, la ligne à vérifier

#Résultat : un booléen qui vaut vrai si la ligne est un commentaire, faux sinon

```
retourner (ligne[0] == '#')
```

fonction recupererLigne(fichier : Fichier, nblignes : int, sauterCommentaires : boolean) : String

#Données : fichier, le fichier sur lequel on doit récupérer la ligne, nblignes : le nombre de ligne (moins 1) à passer pour récupérer la bonne ligne, sauterCommentaires : s'il vaut vrai, on ne comptera pas la ligne des commentaires comme une vraie ligne.

#Résultat : la ligne à récupérer depuis le fichier

```
ligne = 0
finFichier = faux
tant que (finFichier == faux et ligne<nblignes)
    ligneSuivante = prochaineLigne(fichier)
    si longueur(ligneSuivante == 0)
        finFichier = vrai
    sinon si (non (sauterCommentaires et estCommentaire(ligneSuivante)))
        ligne = ligne + 1
    fin si
fin tant que
retourner ligneSuivante
```

fin

fonction recupererCouleur (couleurFichier) : int

#Donnée : couleursFichier, une couleur codée en ASCII

#Résultat : la couleur convertie en valeur entière

```
retourner (int)(prochainCaractere(fichier))
```

fonction ecrireCouleur (fichier : fichier, couleur : int)

#Donnée : fichier, le fichier dans lequel il faut écrire la couleur, couleur : la valeure entière de la couleur

#Résultat : Vide, mais la couleur est convertie en code ASCII et écrite dans le fichier

```
    si(couleur < 0 ou couleur > 255) alors
        ecrireEnFin(fichier,(char)0)
    sinon
        ecrireEnFin(fichier,(char)couleur)
    fin si
fin
```

3 Rapport d'activité

3.1 Répartition des tâches

La répartition des tâches a été différente en fonction de l'avancée du projet. Au début du projet, le travail s'est fait à deux pour définir l'ensemble des spécifications fonctionnelles.

Ensuite, au cours du projet, nous avons tous les deux ciblé les méthodes les plus compliquées dont nous pourrions étudier les solutions possibles de manière individuelle. A partir de ces méthodes compliquées, nous avons travaillé dessus pour mettre en commun de manière fréquente nos idées sur des solutions algorithmiques possibles. La décision de l'algorithme final, quant à elle, s'est faite à deux.

Nous avons répartit les tâches de manière équitable au niveau de la programmation, une fois la partie algorithmique terminée, pour le reste de la durée du projet.

3.2 Difficultés rencontrées

La première difficulté s'est posé sur la fonction de redimensionnement d'une image. Pour résoudre ce problème, nous avons tout d'abord créé les fonctions augmenter taille et réduire taille. Leur conception était simple, et permettait de répondre partiellement aux attentes de l'utilisateur. A force de recherches, nous avons trouvé une solution baée sur une interpolation linéaire, qui a permis de redimensionner efficacement une image PGM.

La deuxième difficulté principale s'est posé sur la fonction « Niveau de gris » Pour résoudre ce problème, il a d'abord fallut dessiner des schémas avec des histogrammes d'images, pour se visualiser la manière dont la fonction allait résoudre le traitement d'image. Les premières ébauches de fonction et d'interpolations ont émergées, et, une fois l'algorithme mis en place et vérifié, l'interpolation linéaire a été intégrée au code et s'est révélée efficace après une série de tests.

4 Manuel d'utilisation

4.1 Lancement du script

Le script de traitement d'images se lance à l'aide de l'invite de commande. Il suffit de taper « python » puis le nom du script. Si aucun argument n'est passé en paramètre, ou si l'argument –h ou –help est passé en paramètre, une aide s'affichera, vous expliquant la manière de procéder pour exécuter ce script, et l'ensemble des fonctions disponibles.

4.2 Utilisation des fonctions

Les fonctions utilisables sont résumées lors de l'affichage de l'aide du script, mais sont également décrites plus en détail ci-dessous. Chaque fonction sera sujette à un exemple, dont l'image originale appelée « Canada.pgm » se trouve ci-dessous :

Figure 1 – Canada.pgm

- Inverser les couleurs de l'image

Ajouter l'option –n

Le résultat de la fonction donne cette image :

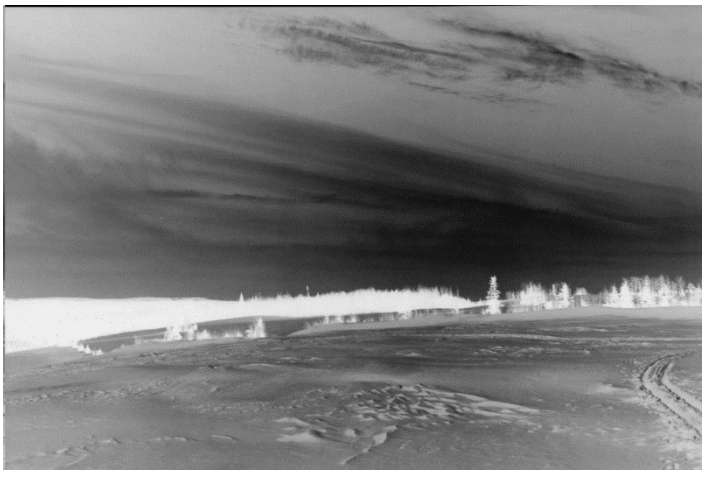

Figure 2 – Canada négatif

- Pivoter une image de 90 degrés dans le sens des aiguilles d'une montre

Ajouter l'option -p

Le résultat de la fonction donne cette image :

Figure 3 – Canada pivot 90

- Pivoter une image de 90 degrés dans le sens inverse des aiguilles d'une montre

Ajouter l'option -pi

Le résultat de la fonction donne cette image :

Figure 4 – Canada pivot moins 90

- Retourner verticalement l'image

Ajouter l'option -mv

Le résultat de la fonction donne cette image :

Figure 5 – Canada miroir vertical

- Retourner horizontalement l'image

Ajouter l'option -mh

Le résultat de la fonction donne cette image :

Figure 6 – Canada miroir *horizontal*

- Modifier la taille d'une image

Ajouter l'option –sc suivit du pourcentage de redimensionnement (40, 100, 125.5, etc…)

L'appel de cette fonction renverra une erreur si l'argument obligatoire pour cette fonction n'est pas un entier strictement positif.

Le résultat de la fonction, avec le paramètre « 25 » donne cette image :

Figure 7 – Canada redimensionné à 25%

- Contraster automatiquement une image :

Ajouter l'option -autoc

Le résultat de la fonction donne cette image :

Figure 8 – Canada contrastes automatiques

- Modifier manuellement les niveaux de gris d'une image

Ajouter l'option –ndg suivit du niveau de gris minimal suivit du niveau de gris maximal à appliquer

L'appel de cette fonction renverra une erreur si les deux arguments obligatoires pour cette fonction ne sont pas des entiers, avec le premier entier inférieur ou égal au deuxième entier.

Le résultat de la fonction, avec le paramètre « 20 200 » donne cette image :

Figure 9 – Canada niveau de gris manuels

4.3 Paramètres obligatoires et optionnels

Il est indispensable de passer à la fin de la ligne du script le nom de l'image source à modifier. Cet argument correspond au chemin d'accès relatif à l'image. Si l'image est au même endroit que le script, alors le chemin source correspondra uniquement au nom de l'image. Ce nom d'image doit comporter son extension, c'est-à-dire « .pgm », sinon le lancement du script provoquera une erreur.

De manière optionnelle, il est possible de rajouter le chemin de destination de l'image, pour conserver une copie de l'image source. Dans ce cas-là, le chemin de destination doit être passé en paramètre à la fin de la ligne de script, c'est-à-dire après le chemin de l'image source. De la même manière que l'image source, si l'image de destination doit se trouver dans le même répertoire que le script, alors le chemin de destination passé en paramètre correspondra uniquement au nom du fichier, avec l'extension « .pgm ».

Conclusion

Ce projet nous a apporté de bonnes connaissances pratiques et techniques. En effet, l'utilisation du langage python nous a permis d'apprendre le fonctionnement d'un langage non typé, permettant d'effectuer des scripts à destination d'un utilisateur. De plus, ce TP nous a apporté de la rigueur concernant le bon déroulement d'un projet, en tenant compte de la complexité des algorithmes, et d'autres paramètres tels que des invariants pour prouver qu'une fonction effectue bien le résultat attendu.

Si nous devions améliorer notre projet, nous aurions proposé une nouvelle interface pour l'utilisateur. A chaque traitement, celui-ci aurait pu choisir son image source et image de destination. Ainsi, il aurait pu effectuer plusieurs traitements en parallèle, sans devoir relancer le script à chaque changement d'image.