ESILV Année 2

Projet Scientifique Informatique

Auteurs :
M. Victor Lenoble
M. Guillaume Turcas

Encadrant :
François Boisson

Rapport Final du
Mercredi 17 Avril 2019

Table des matières

1 Introduction 2

2 Fonctionnement de l'Équipe 2

3 Le Menu 3

4 Les différentes class 5

5 IntToLitEndian(int taille) 7

6 NoirEtBlanc() 7

7 Negatif() 8

8 Rotation(int demande) 8

9 Miroir(char demande) 9

10 Redim(int redim) 9

11 FonctionAvance(int[,] matriceConv, int decal, int div) 10

12 Pixel[,] Fractal() 11

13 Histogram() 12

14 ImageImage(Pixel[,] image2) 14

15 AfficherImage(Pixel[,] imageAffich, string fichier) 14

16 Conclusion 15

1 Introduction

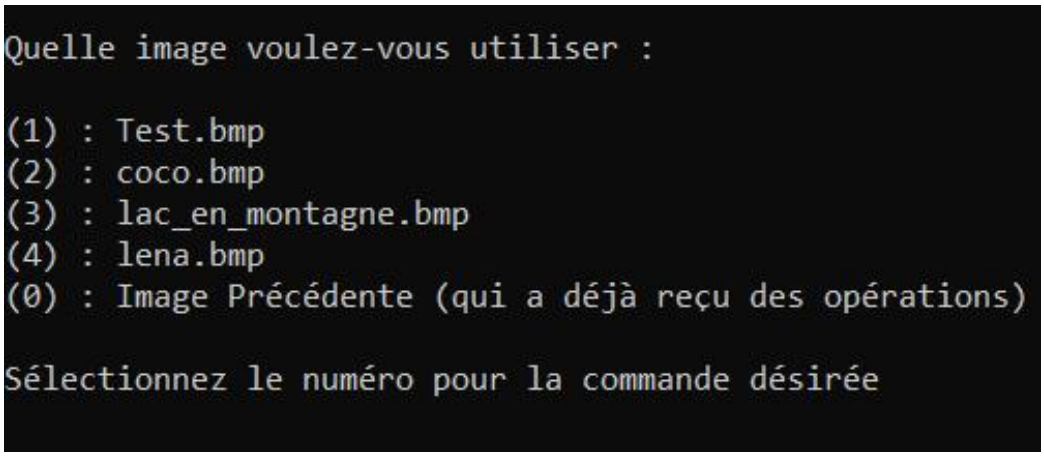
Après avoir choisi tous les deux l'option Informatique pour le semestre 4, et constitué notre équipe, il nous a été fourni un document afin de nous expliquer ce que l'on devait accomplir durant ce semestre. Nous devions réaliser un programme de traitement d'image, avec une très grande autonomie (contrairement au Jeu de La Vie, beaucoup de chose ont dû être trouvées par nous-mêmes) qui nécessitait l'ensemble des connaissances que l'on a acquiert au cours de ces trois derniers semestres, et surtout, beaucoup de rigueur.

2 Fonctionnement de l'Équipe

Le Fonctionnement de l'Équipe était très efficace. Nos personnalités sont assez différentes, mais notre sincère bonne entente a réussi à transformer cela en une réelle force. Ainsi, nous avons réussi à très bien répartir notre travail, de sorte que chacun puisse faire ce qui est dans son domaine de compétence. Après le TD1 qui n'était pas tout de suite évident à comprendre, nous n'avons pas eu l'impression de travailler car nous nous sommes tout de suite vraiment attachés au projet, et travaillons surtout en dehors des séances. En effet, ces dernières ne furent pas forcément des plus productives et nous perdions beaucoup de temps à résoudre des problèmes qui étaient finalement simples. Cependant, après le TD2, nous avons pris une grande avance qui perdura avec à chaque fois un TD d'avance par rapport à ce qui était prévu de faire. C'est pour cela que nous estimons facilement notre temps de travail en dehors des séances à près d'une bonne cinquantaine d'heure à deux.

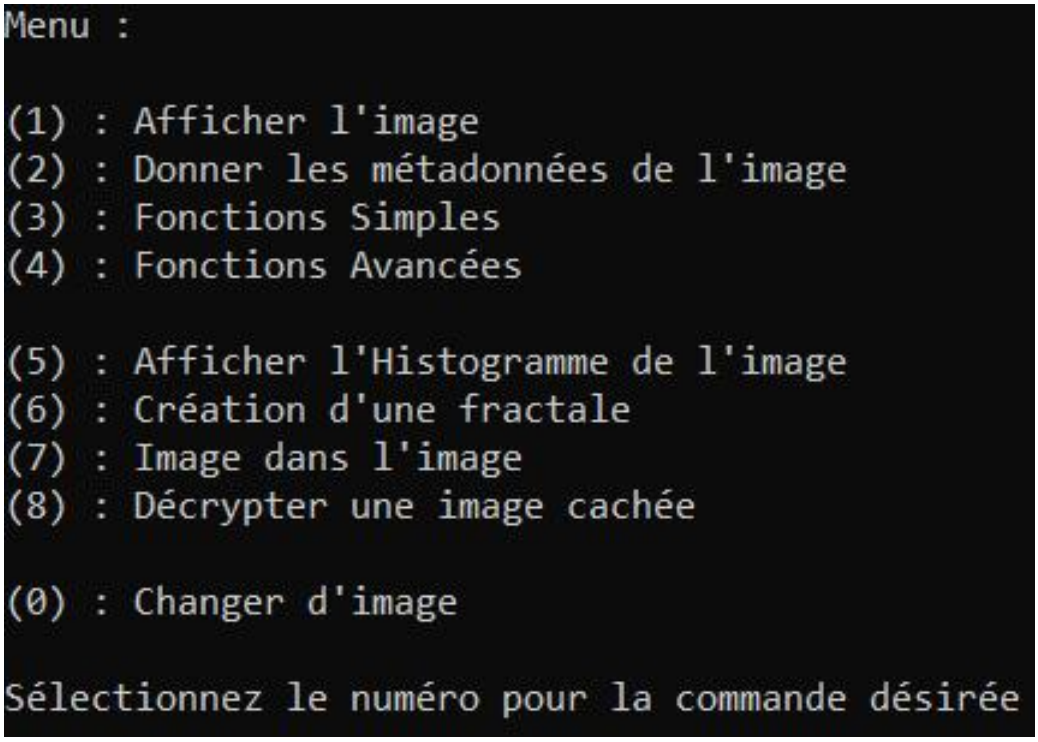
3 Le Menu

Dès le début du lancement du programme, après avoir passé le titre, vous serez confronté à un choix entre plusieurs images en format .bmp. Ces images ont été directement fournies par l'École à travers le DeVinci Online. L'image qui sera générée si vous donnez un chiffre qui n'est pas le bon sera Coco.bmp car c'est l'image qui nous semble la plus intéressante à traiter avec celle du lac, car comporte beaucoup de couleur est possède une taille suffisamment importante pour que cela reste agréable à traiter. Le choix 0, vous donne accès à l'image de sortie directement, qui a été traitée la dernière fois. La particularité de ce choix permet à pouvoir appliquer un grand nombre de filtre à la suite, au lieu qu'elle reprenne à chaque fois sa forme initiale comme c'est le cas avec les autres choix.



Une fois passé cette étape, vous vous retrouverez dans le menu dont nous sommes particulièrement fières. Il est à la fois intuitif et épuré. Il permet d'avoir accès à toutes les fonctionnalités du programme directement en navigant dedans, sans à avoir à toucher au code. Même lorsque l'on voudra changer d'image, il

y'aura une option faite pour ça, et si on appuie sur un mauvais bouton, par exemple on va par erreur dans le sous menu Fonctions Simples alors que l'on désirait aller dans le sous menu Fonctions Avancés, nous avons toujours la possibilité, en appuyant sur 0, à retourner au menu précédent. Cela fait partie des quelques touches personnelles que nous avons fait sur ce programme.



Ce menu, simple d'utilisation, est à la fois un réel confort pour l'utilisateur, mais il l'a été aussi pour nous. En effet, une fois fait, il était beaucoup plus agréable de tester les différentes fonctions de notre programme, sans à avoir à tout modifier dans la classe programme. Il nous a pris du temps, mais cela en valait clairement la peine.

4 Les différentes class

Pour les besoins de ce programme, nous avons créé trois différentes class. La class Program, la class Pixel et la class MyImage.

La class Program permet principalement la création du menu, et à l'appel des différentes fonctions de la class MyImage dont nous allons parler plus en détail dans le futur. À l'intérieur de cette class se trouve néanmoins plusieurs fonction. Bien évidemment la fonction Main qui contrôle tout. La fonction SaisieNombre() qui est indispensable pour se balader dans les menus de manière aussi confortable. La fonction ImageUtilis() qui permet de changer d'image alors que le programme tourne toujours, ou d'en choisir une tout simplement, et la fonction AfficherMatrice(int[,] matrice) qui est utilisée lors de la création de la matrice de convolution personnalisée.
La fonction Main commence par la création de l'image à partir d'un fichier choisi dans la fonction ImageUtilis(). C'est cette image qui sera utilisé tout du long jusqu'à ce que le programme ferme ou que l'utilisateur décide de changer d'image. Ensuite nous avons plusieurs choix. Ou afficher l'image (sera expliqué plus tard), ou afficher les métadonnées de l'image (donne simplement ce qui est trouvé à l'aide du HEADER), ou les fonctions simples (ce qui était demandé au TD2, plus la fonction Négatif qui est de notre création personnelle), ou les fonctions avancés (qui représente les matrices de convolutions (dont la possibilité de faire une matrice de convolution personnalisé avec le décalage et le diviseur, qui là encore, est une création personnelle à notre équipe)), ou alors afficher l'histogramme de l'image, ou encore la création d'une fractale, ou une image dans une image et enfin le décryptage d'une image cachée.

La class Pixel est la class qui est à la fois essentielle, mais aussi la plus simple. Elle ne demande qu'un tableau de trois entiers, et permet d'en créer un Pixel avec le rouge, le vert et le bleue. Puis possède une fonction pour renvoyer ou modifier un pixel, ce qui nous sera très utile par la suite.

La class MyImage est LA class la plus importante de tout notre programme. Elle comporte toutes les fonctions importantes qui sont demandées lors des TDs, et ne comporte au début, qu'une lecture d'un fichier image. Elle lit le code en 8 bits de cette image, lit d'abord son HEADER et lit les informations importantes (la taille, la largeur, la hauteur, en combien de bit l'image est codée, et aussi l'extension de l'image). Une fois le Header lu, c'est à dire les 54 premiers chiffres,

elle va lire tous les pixels un par un, qu'elle va inclure valeur par valeur, dans une matrice RVB. La matrice ImageInit représente l'image qui a été lue par le programme dès le début, et ce sera cette même matrice qui sera utilisée pour toutes les fonctions qui vont impacter sur le changement de l'image.
C'est d'ailleurs cette class qui va comporter toutes les fonctions qui vont suivre.

5 IntToLitEndian(int taille)

Cette fonction très utile, qui prend un nombre quelconque, et le renvoie en format Little Endian. Cette fonction est très utile dans plusieurs situations, surtout dans la fonction AfficherImage.

6 NoirEtBlanc()

Cette fonction, prend l'image de départ, et prend pour chaque pixel, la moyenne entre les trois valeurs, rouge, vert et bleu. Elle change la valeur du Pixel de départ pour la remplacer par cette moyenne. Une fois cette opération effectuée sur chaque Pixel, l'image qui en sort est en noir et blanc. Elle renvoie une matrice de Pixel.

7 Negatif()

Cette fonction, prend l'image de départ, et prend pour chaque pixel, sa valeur, et elle soustrait cette même valeur à 255. Elle change la valeur du Pixel de départ pour la remplacer par cette nouvelle valeur calculée. Une fois cette opération faite sur chaque Pixel, l'image qui en sort est en négatif, donc chaque couleur est inversée dans l'image. Cette fonction fait partie des fonctions supplémentaires qui nous est demandé au TD5. Elle renvoie une matrice de Pixel.

8 Rotation(int demande)

En fonction de la demande initiale faite dans la class Program, on prend chaque pixel de l'image de départ, et on le place dans une nouvelle image de dimension personnalisée en fonction encore une fois de la demande, à un endroit bien précis, afin qu'au final, nous obtenions une image avec une rotation précise (90, 180, 270). Elle renvoie une matrice de Pixel.

9 Miroir(char demande)

En fonction de la demande initiale faite dans la class Program, on prend chaque pixel de l'image de départ, afin de le placer dans une nouvelle image à un endroit bien précis en fonction si vous demandez un miroir horizontal ou vertical. À la fin on obtient une image avec un "filtre" miroir affecté dessus. Elle renvoie une matrice de Pixel.

10 Redim(int redim)

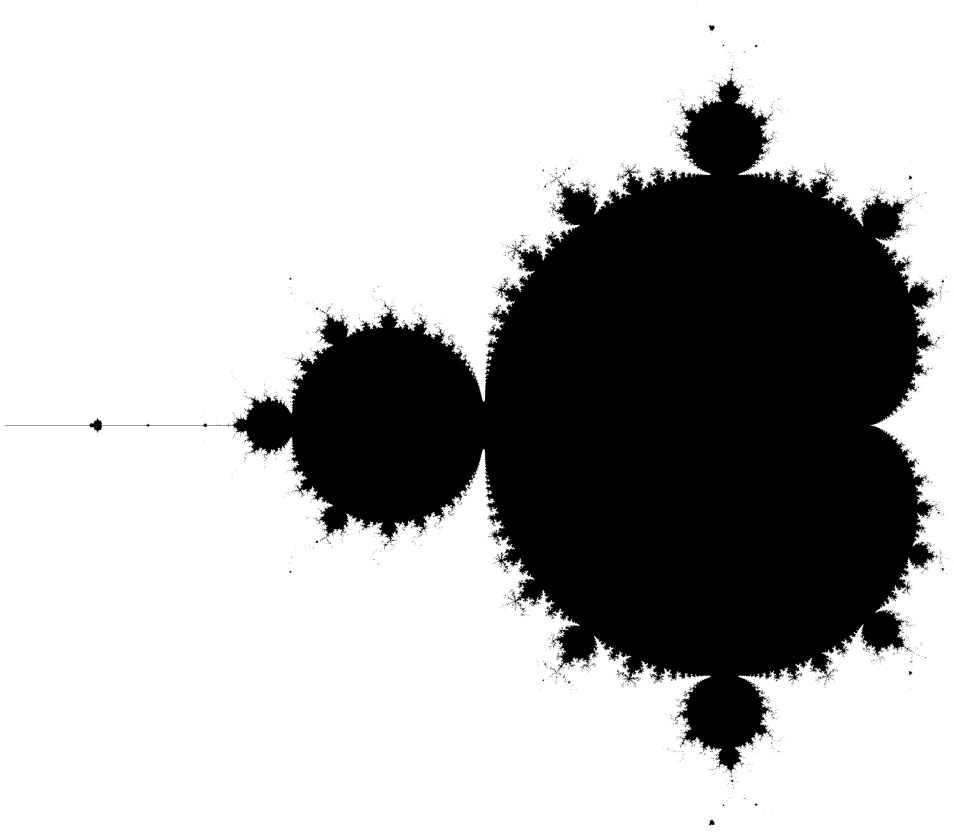
En fonction de la demande initiale faite dans la class Program, on prend l'image de départ, et soit on supprime un pixel sur deux s'il faut la réduire afin de placer ceux qui restent dans une image deux fois plus petite, soit on fait que chaque pixel prenne deux fois plus de place pour les placer dans une image deux fois plus grande. Elle renvoie une matrice de Pixel.

11 FonctionAvance(int[,] matriceConv, int decal, int div)

La fonction FonctionAvance est celle qui permet l'application des matrices de convolution sur une image. Pour se faire, elle va prendre chaque pixel un à un, et va créer une nouvelle matrice. Ceci se faisant par le passage d'une autre fonction, la fonction PixelToConv qui est celle qui permet de faire en sorte qu'un pixel, devienne une matrice 3*3 de pixel en fonction de ceux qui l'entoure, puis cette matrice 3*3 est multipliée avec la matrice de convolution par le biais d'une autre fonction MultiMat3p3, et elle ne garde que le pixel du centre de cette matrice. Ensuite elle lui applique un diviseur et un décalage en fonction de ce qui a été établi dans la class Program. Appliqué à chaque pixel, et nous avons une nouvelle image qui a subi la matrice de convolution. En fonction de la valeur de cette matrice, plusieurs effets peuvent se produire (flou, repoussage, détection de contour...). Cette fonction retourne une matrice de Pixel.

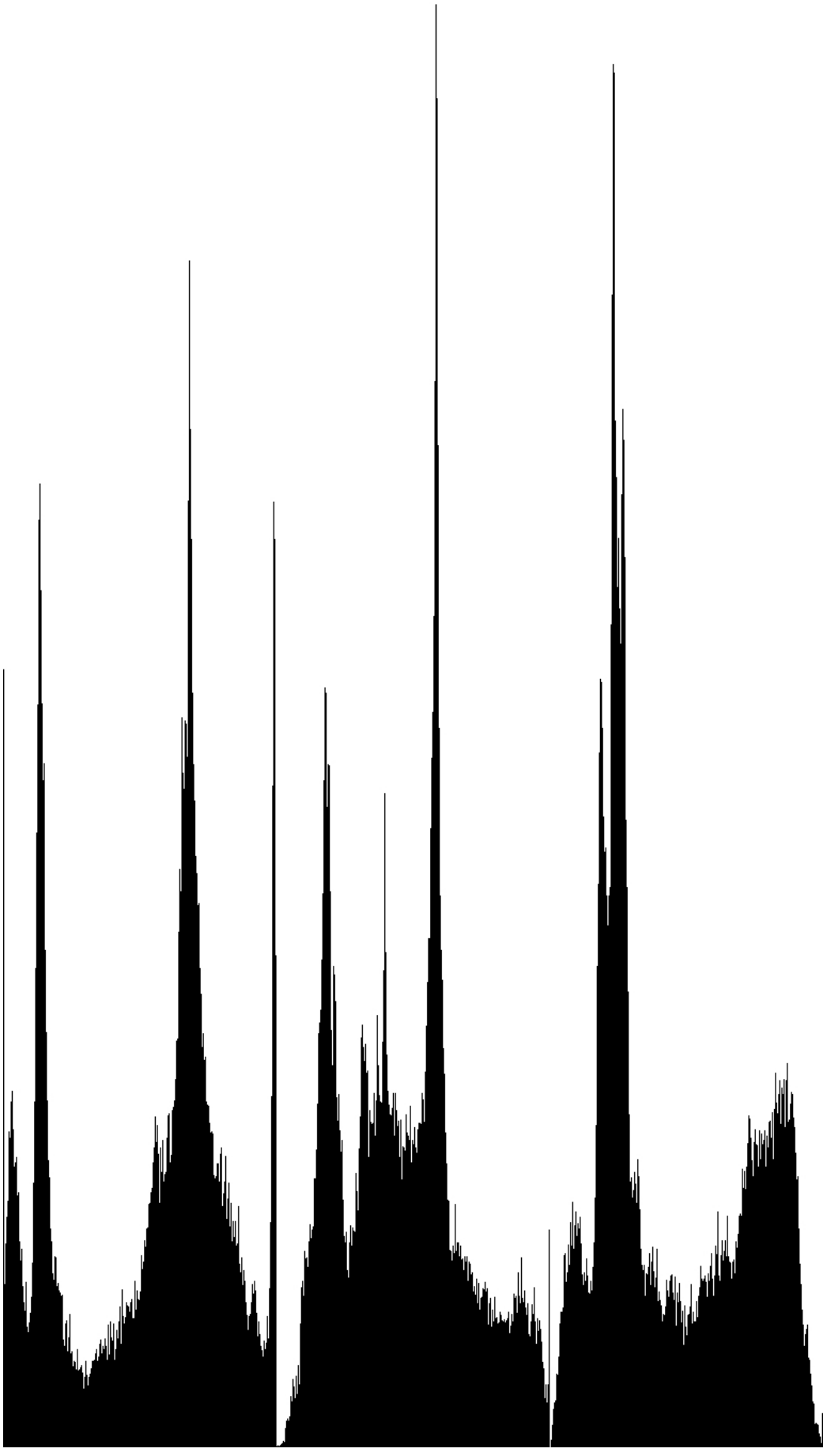
12 Pixel[,] Fractal()

La fonction Fractal permet se dessiner la fractale de Mandelbrot. On commence par définir la taille de l'image pour connaître le nombre de points du plan complexe à étudier. On définit le nombre d'itérations qui décide si la suite (z = z*z +c) créée par les nombres z et c converge en ce point. On utilise les calculs sur partie réelle et partie imaginaire pour ne pas avoir besoin de classe nombre imaginaire. Si la suite dépasse 2 on déduit qu'elle diverge. Au bout d'un nombre d'itérations choisi on considère que la suite converge. La fonction retourne la matrice de pixels (en noir et blanc).



13 Histogram()

Cette fonction, qui à partir de l'image de départ, permet de prendre chaque pixel un à un, récolte chaque valeur de ce pixel (la valeur rouge, la valeur verte et la valeur bleue), et le place dans une matrice qui fait 768 pixels de large, et un max de hauteur (le max représentant la valeur la plus de fois représentée dans la matrice de tableau d'entier (de Pixel) qui forme l'image). Elle compte donc la valeur de chaque sous pixel, et l'incorpore dans une image de taille variable. Pour que cette image reste en .bmp, et que le programme puisse lire le header, nous changeons la valeur max, pour en faire un multiple de quatre supérieur à cette valeur. Et c'est tout. L'image est alors créée de toute pièce par le programme, et renvoie une matrice de Pixel.



14 ImageImage(Pixel[,] image2)

La fonction ImageImage permet de cacher une image (de matrice de Pixel image2) dans l'image avec laquelle on travaille actuellement. Elle commence par réduire le nombre de couleurs en réduisant de 8 bits à 4 bits chaque pixel pour notre image initiale, une division par 16 que l'on applique aussi à la deuxième image sans remultiplier par 16 pour la deuxième image. On insère alors chaque pixel de la deuxième image comme résidu, sur les pixels de la première image.
(Note : la taille finale est fixée sur la première image, si la deuxième image est plus grande alors une partie des data est perdue.)

TrouverImageCachee()

La méthode TrouverImageCachee() permet de réaliser l'étape inverse, par des divisions euclidienne on retrouve pour chaque pixel les résidus potentiels d'une image cachée. En les multipliant par 16 on retrouve une image proche de celle qui a été cachée précédemment, parfois décalée ou tronquée. La méthode retourne une matrice de pixels

15 AfficherImage(Pixel[,] imageAffich, string fichier)

Cette fonction, qui prend au départ la matrice de Pixel que toutes les autres fonctions précédentes ont créé, et le fichier Image de départ, afin de créer une nouvelle image à partir de ça. Pour cela il créé un fichier de taille 54 (pour le Header) + la taille de la matrice de Pixel. Ensuite, il remplit le Header à partir du fichier de départ, puis modifie les différentes valeurs en lien avec la taille, à l'aide, encore une fois, de la matrice de Pixel. Ensuite il donne un par un, chaque valeur de Pixel au fichier, à l'aide notamment, de la fonction LitEndian. Une fois que c'est fini, le programme ouvre le ficher

16 Conclusion

Nous sommes tous les deux très fiers de ce que l'on a accompli durant ce module. Nous avons réussi à faire tout ce qui a été demandé et même plus, avec certains éléments qui nous paraissaient impossible dans un premier temps, et qui se sont révélés comme un succès au final. Nous avons été très efficace, et nous avons beaucoup travaillé et ça a payé. C'est donc un sentiment très positif qui ressort de ce module.