

EISTI

PROJET : MANIPULATION D'IMAGES

ALGORITHMIQUE ET PROGRAMMATION PROCÉDURALE

Rapport de Projet

David RIGAUX

Soner CAVUSOGLU

17 janvier 2016



Table des matières

1	Introduction	3
2	Définitions	3
2.1	PPM	3
2.2	Pixel	4
3	Étude du projet	4
4	UNITS	4
4.1	ES.pas	5
4.1.1	Les Records	5
4.1.2	Chargement	6
4.1.3	Sauvegarde	7
4.2	menuText.pas	7
4.2.1	Menu Visuel	7
4.2.2	Entrée de paramètres	17
4.3	manipulationBasiques.pas	17
4.3.1	Conversion en niveaux de gris	17
4.3.2	Binarisation de l'image	18
4.3.3	Zoom *2	19
4.3.4	ZoomSur2	21
4.3.5	Affichage de l'histogramme	22
4.3.6	Recadrage dynamique	23
4.3.7	Renforcement de contraste	23
4.3.8	Floutage	24
4.3.9	Érosion	25
4.3.10	Dilatation	26
4.3.11	Segmentation	27
4.4	create.pas	28
5	Limite du programme	30
5.1	Fonctions qui modifient l'image	31
5.1.1	Renforcement de contraste	31
5.2	Fonctions non fonctionnelles	31
5.2.1	Convolution	31

5.2.2	Opérateur de chanda	31
6	Bilan Personnels	31
6.1	David RIGAUX	31
6.2	Soner CAVUSOGLU	32
7	Conclusion	33

1 Introduction

L'objet de notre projet en informatique est de réaliser un programme pouvant manipuler des images. Pour réaliser ce projet, nous avions un temps imparti d'un mois et demi, et le langage de programmation devait être le Pascal. Les images que nous devons traiter sont de type *.ppm*. Ce type d'image est relativement utile, car il est composé entièrement de chiffres et il peut être codé. Nous avons structuré notre projet en plusieurs UNITS, tel que *manipulationBasiques.pas*, *ES.pas*, *menuText.pas* et *create.pas* afin que notre programme puisse être plus lisible et structuré. En effet chaque Unit a son rôle spécifique.

2 Définitions

2.1 PPM

Le portable pixmap file format (PPM) est un format de fichier utilisé pour les échanges. Il a été défini et est utilisé par le projet *NetPBM*. Le format de fichier est utilisé pour des images couleur. Chaque pixel est codé par trois valeurs (rouge, vert et bleu). Le codage d'une image en PPM est le suivant :

1. P3 (*Numéro magique*)
2. En commentaire le nom du fichier (*Optionnel*)
3. largeur hauteur (*Dimensions*)
4. MaxVal (*Valeur maximum des couleurs*)

```
1 P3
2 #david.ppm
3 10 10
4 1
5 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1
6 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1
7 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1
8 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1
9 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
11 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1
12 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1
13 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1
14 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1
```

FIGURE 1 – Une image PPM codée sur 1 couleur

2.2 Pixel

Le pixel (souvent abrégé : px) est l'unité de base permettant de mesurer la définition d'une image numérique matricielle. Son nom provient de la locution anglaise picture element, qui signifie « élément d'image ». Un pixel RGB (*Red, Green et Blue*) est un pixel ayant pour chaque couleur primitive 256 couleurs (allant de 0 à 255). Ce qui nous fait un total de 16 777 216 couleurs possibles pour un tel pixel ($256*256*256$). Nous travaillons le plus souvent avec ce type de pixel.

3 Étude du projet

Avant de commencer quoi que ce soit à coder, il nous fallait tout d'abord comprendre le projet. Nous avons alors lu plusieurs fois l'énoncé en prenant des notes et essayant de se visualiser ce qu'il nous était exactement demandé et à faire des schémas. De plus nous avons fait beaucoup de recherche sur Internet pour comprendre certains mots de vocabulaire que nous ne connaissons pas et également des recherches sur certaines structure en pascal qui pourrait nous être utile.

Il nous est demandé de prendre une image en entrée, la chargé et ensuite lui appliquer plusieurs *effets/modifications*. Et finalement il faut qu'on enregistre cette image sous, c'est-à-dire créer un nouveau fichier et sauvegarder dedans. Le nombre total de fonctions de modification est douze. Cependant, il y a également un certain nombre de fonctions qui ne sont pas visibles par l'utilisateur, mais qui effectuent certaines opérations en arrière-plan. La consigne était de CODER pour que l'utilisateur puisse modifier une image depuis la console sans lancer le programme même. C'est-à-dire l'utilisateur peut écrire dans la console, par exemple :

```
./image -i toto.ppm -o tata.ppm
```

Dernièrement, l'énoncé demande de créer une fonction qui permet de créer une croix, avec la hauteur, la largeur et l'épaisseur de la croix entrée par l'utilisateur.

4 UNITS

Comme dit précédemment dans l'introduction, notre programme comporte quatre UNITS. Nous allons maintenant vous parler de leur fonc-

tionnement ainsi que de leur élaboration.

4.1 ES.pas

Ce Unit s'occupe de tout ce qui est E/S d'image. Plus précisément, il s'occupe du chargement et de la sauvegarde des images. Par conséquent seul dans ce Unit nous verrons du code pour lire ou écrire dans un fichier. Nous avons pris cette décision, car c'était plus judicieux et moins gourmand en ressource. Nous utilisons 3 balises dans ce Unit :

```
{$i-}  
{$i+}  
{$mode objfpc}
```

Les deux premières balises disent au compilateur comment traiter les situations d'erreurs d'E/S : lever une exception ou stocker le code d'erreur dans la variable IOResult. La dernière sert à activer le mode ObjetPascal.

Nous avons déclaré le record dans ce Unit également parce que toutes les images passent par ce UNIT, car elles entrent et sortent par ces lignes de codes. À travers ce Unit, nous avons ajouté des vérifications permettant de vérifier si le fichier existe, quant-il s'agit d'un chargement, et si l'extension du fichier entré bien *.ppm*.

4.1.1 Les Records

Nous allons maintenant vous expliquer les structures des différents RECORD qui sont présents dans ce programme :

4.1.1.1 Pixel

Le record Pixel est défini ainsi :

```
TYPE pixel = RECORD  
  r, g, b: INTEGER;  
END;
```

En déclarant un record pixel avec R,G,B ceci nous permet de faire par la suite un tableau avec 3 cases dans une case, ce qui facilite énormément le codage du programme.

4.1.1.2 limage

Le RECORD Image est sûrement le Record le plus important du programme, car toutes les manipulations des images passent par ce type de RECORD. Il est défini comme ceci :

```
TYPE Image = RECORD
    magic : STRING;
    commentaire : STRING;
    width : INTEGER;
    height : INTEGER;
    max : byte;
    body : ARRAY OF ARRAY OF PIXEL;
END;
```

Je vais vous expliquer chaque partie du RECORD :

- *magic* : Il s'agit dans notre cas de programme "P3"
- *commentaire* : Si dans une image on trouve un commentaire, le commentaire va être stocké dans ce STRING.
- *width* : La largeur de l'image
- *height* : La hauteur de l'image
- *max* : La valeur maximum de codage de couleur
- *body* : Un tableau dynamique multidimensionnel de Pixel

4.1.2 Chargement

Pour le chargement, nous avons commencé par ouvrir le fichier en question en lecture seule, puis en stockant les paramètres de l'image dans le record (magic, commentaire, width, height et max). Pour stocker la largeur et la hauteur de l'image, il nous fallait lire la ligne avec les dimensions. Nous utilisons ensuite un *length* pour calculer la longueur de la chaîne de caractère et finalement nous lisons jusqu'à ce qu'on rencontre un *espace*, à ce moment-là on aura la largeur de l'image. Pour finir, nous lisons la chaîne jusqu'à la fin pour avoir la hauteur. La valeur Max se lit simplement avec un READLN dans le fichier. Maintenant que nous savons les dimensions de l'image, nous pouvons initialiser le tableau de Pixels. Sachant que l'ordre de codage d'un pixel commence par le rouge, puis le vert et finalement le bleu. Dans la boucle FOR, qui nous permet de parcourir toutes les lignes

restantes du fichier, nous avons fait en sorte de lire selon l'ordre de codage de pixel. Ainsi pour chaque pixel nous avons ses valeurs de rouge, de vert et de bleue. Ce qui est important de savoir c'est que dès qu'on a "récolté" toutes les informations dans le record nous fermons directement le fichier. La variable dans laquelle nous stockons ces informations est globale à tout le programme ce qui nous permet d'y avoir accès dans tout le programme. Pour avoir une fonction de sauvegarde "*puissante*", nous avons utilisé une exception qui essaye un certain nombre d'instructions et si une ne marche pas elle saute directement à la partie *except*. Dans ce programme, il est codé pour que cela tourne en rond jusqu'à ce que l'utilisateur entre un nom de fichier cohérent.

4.1.3 Sauvegarde

Dans notre programme la sauvegarde a été faite en ouvrant le fichier (ou en créant un fichier) en mode écriture et à écrire le contenu du record à la suite. Après toute modification appliquée à l'image, l'utilisateur à la possibilité d'enregistrer son image. Il a ensuite 4 choix possibles, soit d'enregistrer, donc écraser l'image précédente, soit d'enregistrer sous et est ensuite demandé d'entrer le nom de fichier qu'il veut créer. Nous utilisons également une exception, pour le même but que dans le chargement.

4.2 menuText.pas

Dans ce Unit, tout le menu du programme est défini avec les différents menus (Création ou modification, entrée du nom de fichier à modifier, manipulations d'images, sauvegarde, ainsi que toute la partie création de la croix). Ce unit est en quelque sorte tout le corps du programme, donc il est primordial au bon fonctionnement du programme et à l'utilisation de l'utilisateur. À travers tout le programme, l'utilisateur a la possibilité de quitter le programme avec une option dédiée à cet effet.

4.2.1 Menu Visuel

Comme dit précédemment, le menu est primordial pour son utilisation. Dès le lancement du programme, l'utilisateur se trouve face à un écran avec une bannière écrite en gros "IMAGE". Et le choix de modifier ou de créer une image.

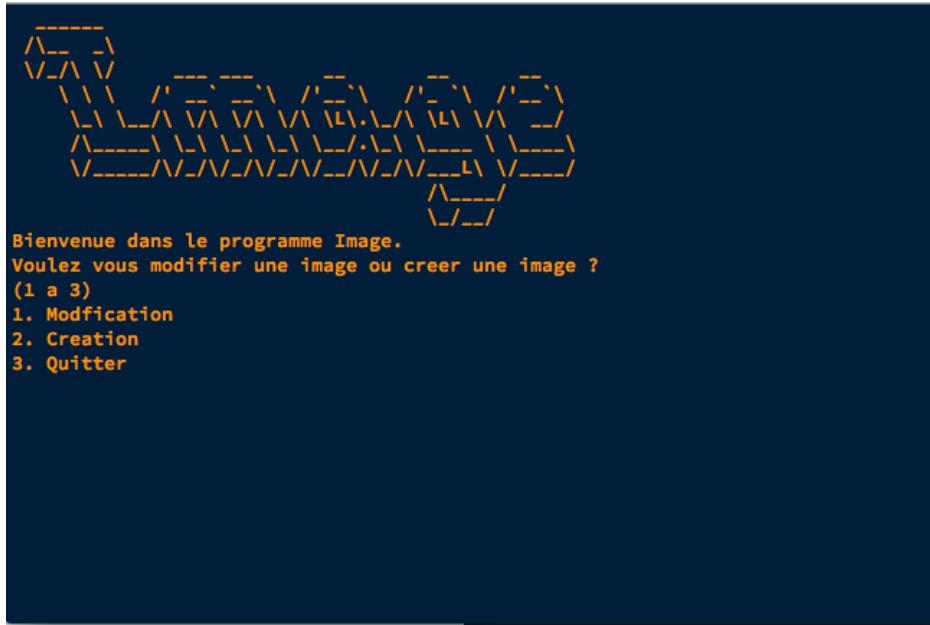


FIGURE 2 – Ouverture du programme

À chaque fois, qu'on appelle un menu on a ajouté la commande *ClrScr* ce qui permet d'effacer le contenu sur l'écran, et nous réaffichons toujours la bannière. Cela permet de donner un effet de changement de fenêtre qui est assez agréable et par-dessus tout ne perd pas l'utilisateur parmi toutes les lignes afficher sur la console. L'entrée que l'utilisateur fait est en fait un STRING. Il était préférable de choisir des strings, car même s' il entre une chaîne de caractères quelconque le programme ne sortira pas avec une ligne d'erreur. S'il s'agissait d'un INTEGER, il suffisait à l'utilisateur de faire une faute de frappe pour qu'il perd tout le travaille accomplie pour le moment, qui peut être frustrant. Si l'usager entre autres que 1, 2 ou 3, le programme va afficher que son entrée est invalide et le laissera entrer encore une fois le choix qu'il veut. Il s'agit donc d'un menu où il est impossible d'en ressortir une erreur.

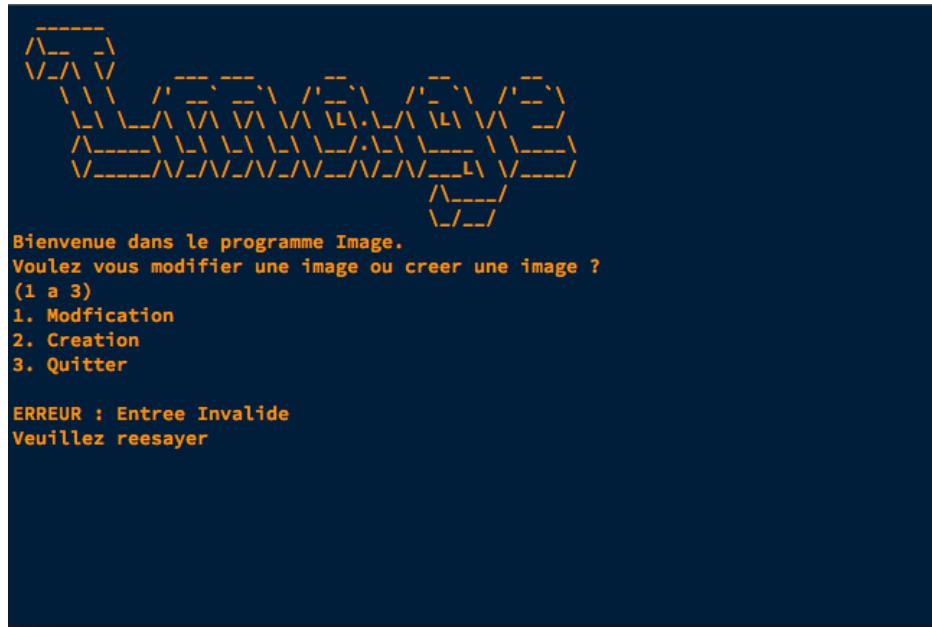


FIGURE 3 – Entrée Invalidé

4.2.1.1 1. *Modification*

Dans le cas où l'utilisateur entre le choix "1", il devra ensuite entrer le nom du fichier qu'il veut modifier, qu'on va donc charger. Ici aussi, nous avons une vérification que tout d'abord son entrée soit correcte c'est-à-dire que le nom du fichier finit bien par *.ppm*. pour effectuer cette vérification nous prenons la longueur du string qu'il vient de rentrer et avec un FOR length DOWNT0 length-4 on fait un ajoute les lettres et nous vérifions qu'il s'agit bien de "*mpp.*". Si le nom du fichier n'est correct, il affiche une erreur,sinon le programme va ensuite utiliser la fonction chargement du Unit *ES.pas* et vérifier qu'il s'agit d'un fichier existant. S'il n'existe pas, il affiche une erreur. À cette étape, il y a également la possibilité de taper "retour" ou "RETOUR" (insensible à la case) pour revenir en arrière c'est-à-dire au menu avec le choix entre la création et la modification.



FIGURE 4 – Entrée nom du fichier pour modification

Après avoir entré un nom de fichier correct et que le programme l’ait chargé sans rencontrer une erreur l’utilisateur aura ensuite le choix parmi toutes les manipulations mises à sa disposition. Le menu qui suit à un total de 15 choix, inclus l’option de retourner en arrière, sauvegarder et quitter le programme. Le menu affiche toutes les options avec des numéros devant, pour l’utiliser il ne suffit que d’entrer le numéro de l’option que vous voulez appliquer. Avant le menu, il est marqué le nom du fichier que le programme a chargé. L’entrée du choix passe ici aussi par une vérification comme tous les autres menus du programme. Quand l’utilisateur a entré le choix fait, il est marqué avant le menu l’opération qui vient de se faire et il peut continuer à faire tout autre choix l’un après l’autre.



FIGURE 5 – Menu modification

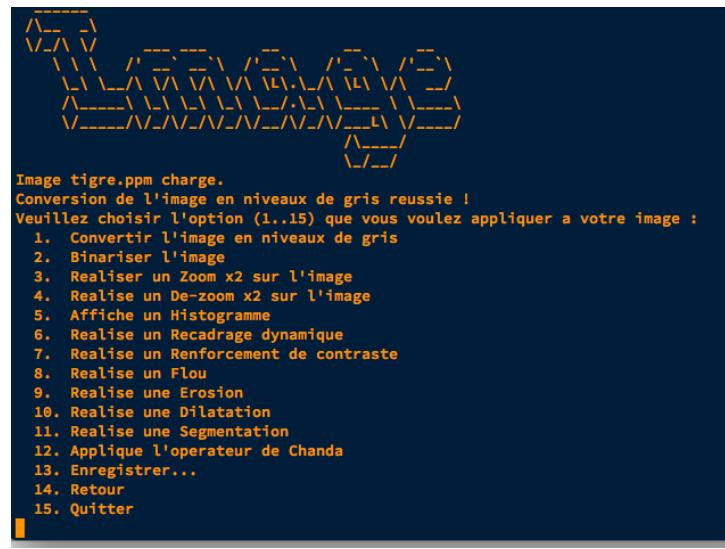


FIGURE 6 – Effet appliqué

L'option retour est présente si jamais vous vous êtes trompé d'image à charger, au lieu de tout recommencer les manipulations il ne suffit de taper

de "14" pour faire un retour et entrer un nouveau nom de fichier. Le retour est faisable grâce au fait que la variable de l'image chargée est globale donc on écrase tout simplement ensuite avec la nouvelle image chargée. Toute modification n'est appliquée à l'image même que quand nous passons par l'étape de sauvegarde.

En choisissant cette option un nouveau menu s'affiche avec les options possibles pour l'enregistrement (enregistrer, enregistrer sous, retour ou quitter).

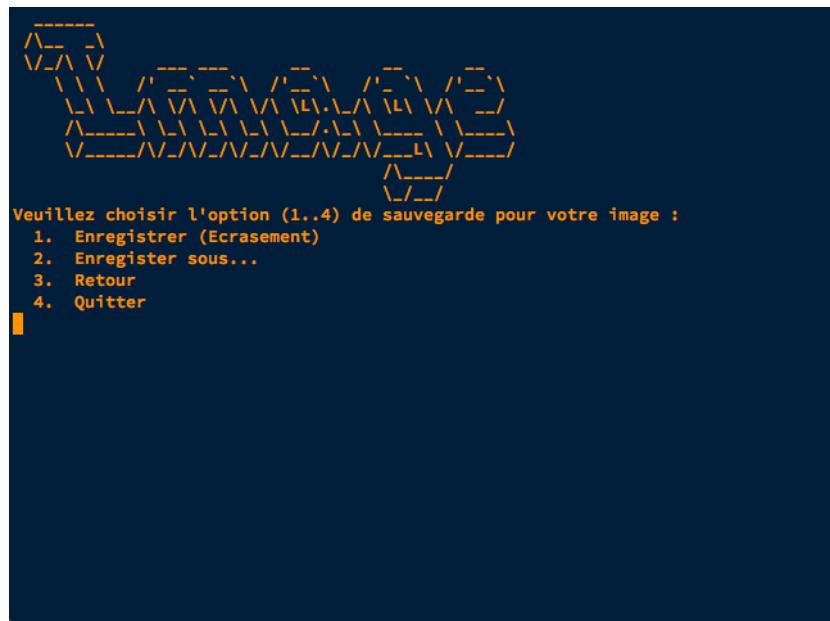


FIGURE 7 – Menu Enregistrement

Si vous choisissez ici l'option de Enregistrer tout simplement, l'image d'origine sera écrasée avec l'image modifiée et ensuite quittera le programme avec un écran de sortie. Si cependant l'utilisateur choisit d'enregistrer sous alors dans ce cas il devra ensuite entrer le nom du fichier qu'il veut créer. Le programme vérifie une fois de plus qu'il s'agit bien d'un nom de fichier valide avec l'extension *.ppm* et si tout est bon, il renvoie le nom du fichier dans la fonction d'enregistrement sous qui créer alors ce fichier et écrit dans ce fichier, puis quitte le programme avec l'écran de sortie.

FIGURE 8 – Enregistrement sous

FIGURE 9 – Écran de Sortie

4.2.1.2 2. *Création*

En choisissant cette option, le programme affiche aussitôt un écran où l'utilisateur doit entrer le nom du fichier qu'il veut créer. Le code vérifie comme dans le reste du code que l'extension du fichier est correcte et ainsi commence à demander des informations à l'utilisateur sur l'image qu'il veut créer. Il demande tout d'abord la largeur de l'image qu'il veut en pixels. Cette valeur sera vérifiée par une fonction dans le Unit *create.pas* qu'on évoquera dans la suite du rapport. La même action est appliquée pour la hauteur de l'image, et finalement il est demandé d'entrer l'épaisseur de la croix souhaitée.

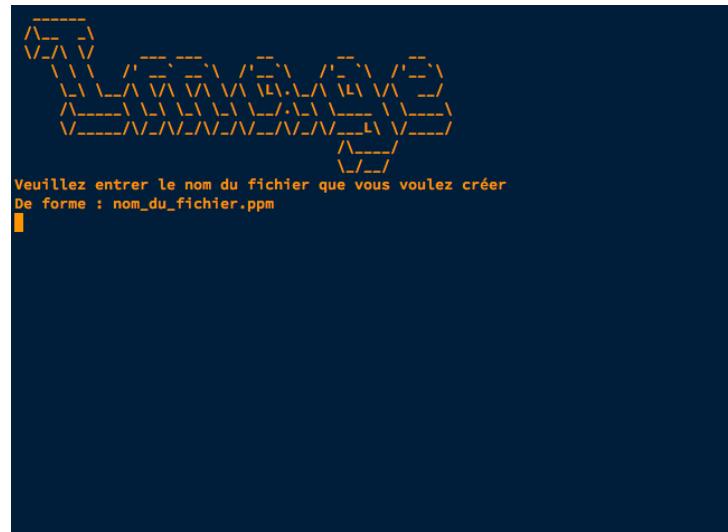


FIGURE 10 – Entrée nom du fichier à créer

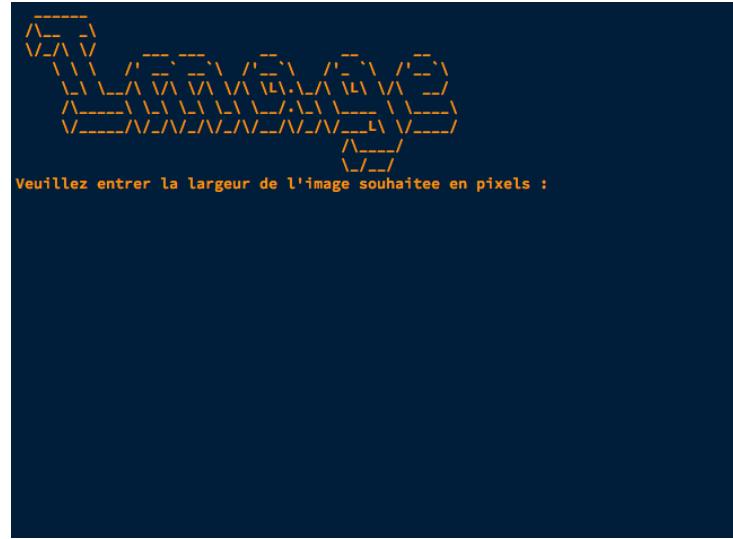


FIGURE 11 – Entrée de largeur

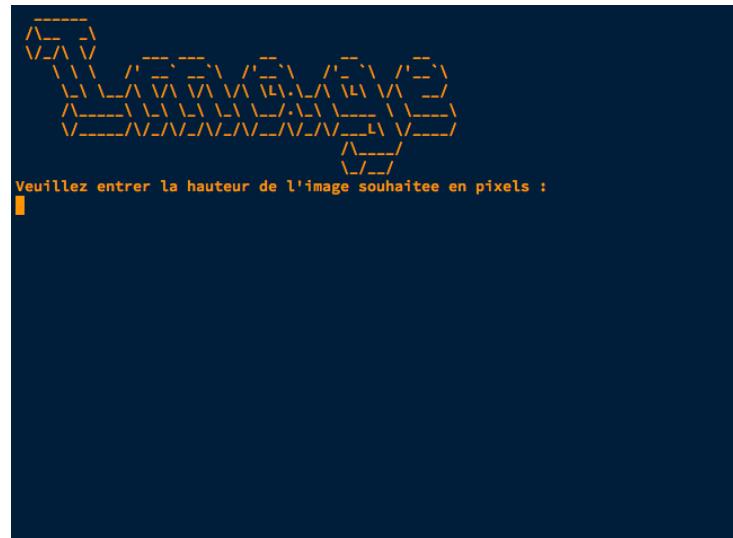


FIGURE 12 – Entrée de hauteur

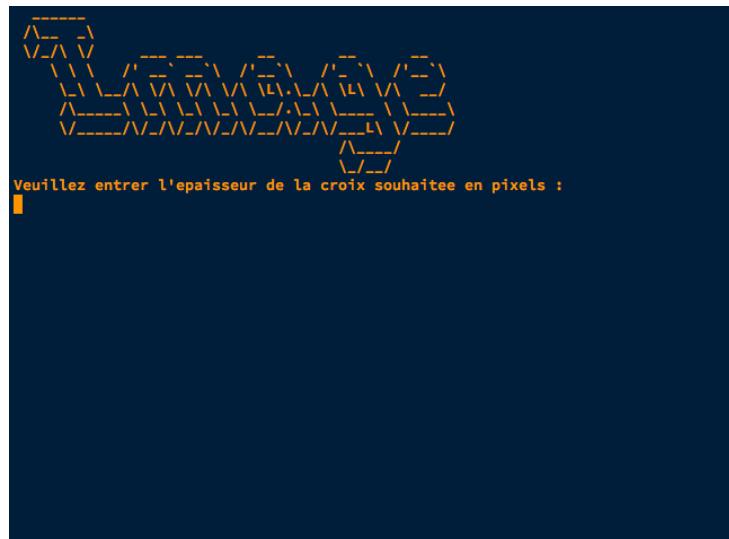


FIGURE 13 – Entrée de l'épaisseur

Finalement, nous avons un écran qui nous dit si l'image a été créée ou pas et les dimensions de cette image en question. De plus, certaines options sont présentes, par exemple, accéder directement à la modification d'image, un retour à l'accueil ou la possibilité de quitter le programme.

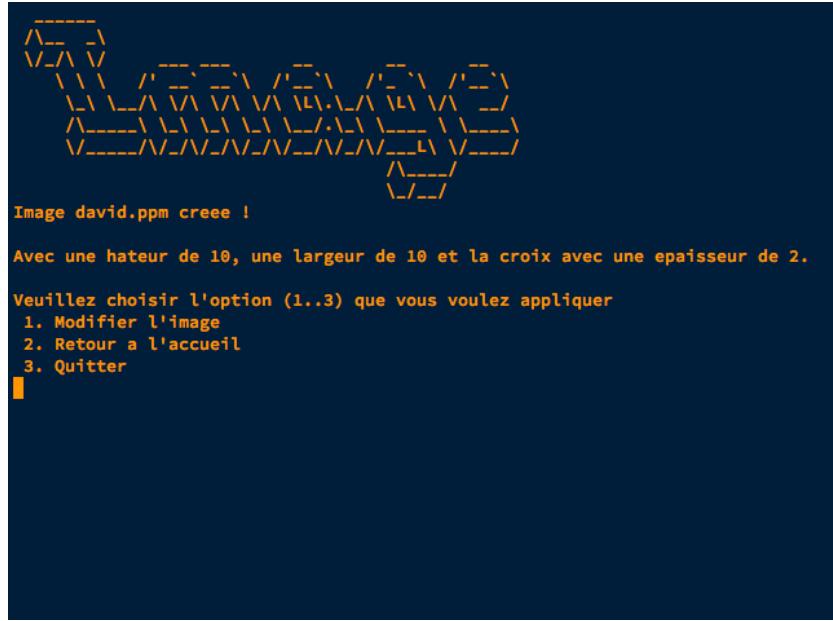


FIGURE 14 – Sortie de création

Le reste se passe dans le unit *create.pas*

4.2.2 Entrée de paramètres

4.3 manipulationBasiques.pas

Dans cet Unit on retrouve toutes les différentes fonctions permettant de manipuler notre image. Cependant, la majorité des opérations traillent sur des images en niveaux de gris ou binarisées. C'est pourquoi nous avons décidé de vous les présenter en premier. De plus, toutes nos fonctions recopient les dimensions de l'image initiale. À partir de ces dimensions, nous construisons des tableaux. Généralement, les lignes et les colonnes correspondent aux dimensions de notre image, sauf pour la fonction zoom et dézoomé. À partir de ces tableaux (qui se retrouvent dans toutes nos fonctions), nous faisons nos calculs.

4.3.1 Conversion en niveaux de gris

Le but de cette fonction est de convertir notre image initiale (en couleur) , en une image en niveaux de gris.



(a) Image non modifiée

(b) Image modifiée

FIGURE 15 – Conversion en niveaux de gris

Notre fonction *convertisVersGris* applique pour chaque pixel un calcul simple :

$$Gris = 0,3 * Rouge + 0,59 * Vert + 0,11 * Bleu$$

Une fois ce calcul fait, nous gardons les entiers, et nous les égalisons aux 3 sous pixels rouges , vert , bleu. Cela nous permet d'obtenir un niveau de gris pour chaque pixel , et du coup, en l'appliquant sur toute l'image nous avons une image transformée en niveau de gris.

4.3.2 Binarisation de l'image

Le but de cette fonction est de convertir notre image initiale (en couleur) , en une image binarisée, c'est-à-dire en une image en noir et blanc.

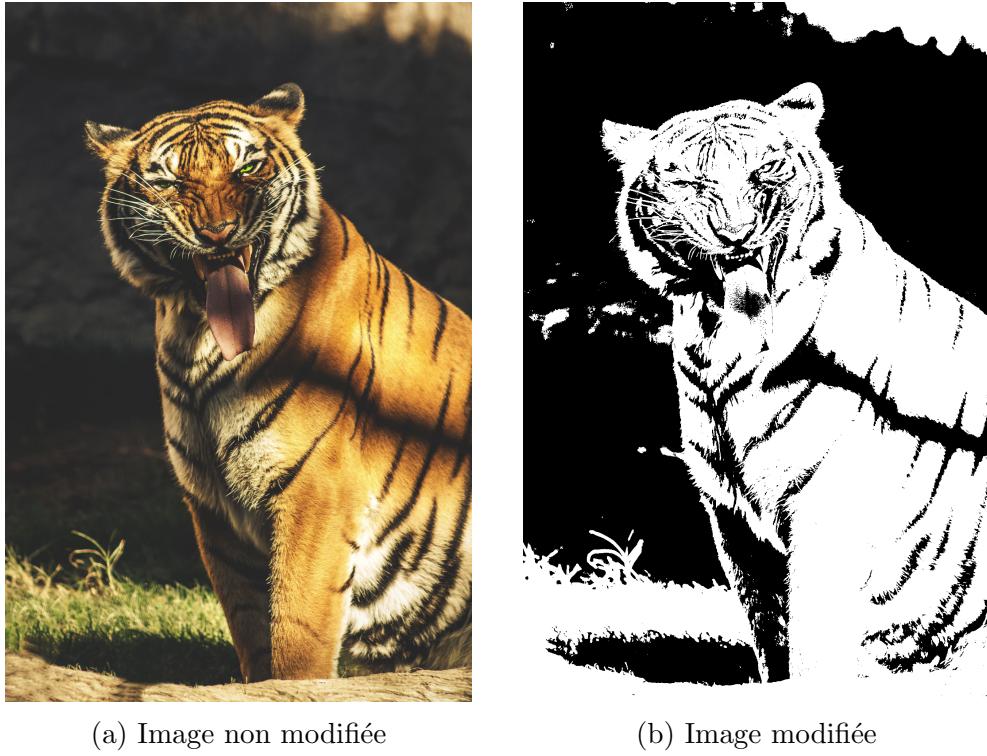


FIGURE 16 – Binarisation de l'image

Pour notre fonction *binariseImage* on applique notre fonction histogramme à notre image. La fonction histogramme nous donne le nombre de fois où chacune des 256 couleurs est utilisée. Ensuite, nous appliquons la fonction *calculSeuil* qui nous calcul le seuil. Et ce seuil est égal à la valeur médiane, c'est-à-dire que la moitié des pixels seront en dessous de ce seuil et l'autre moitié au-dessus. Une fois qu'on a ce seuil, tous les effectifs (pixels) en dessous de ce seuil prennent la valeur 0 et donc deviennent noirs, et inversement, l'autre moitié prend la valeur 255 et deviennent blanches.

4.3.3 Zoom *2

Le but de cette fonction est de réaliser un zoom fois 2 à notre image initiale, c'est-à-dire de la voir agrandir.

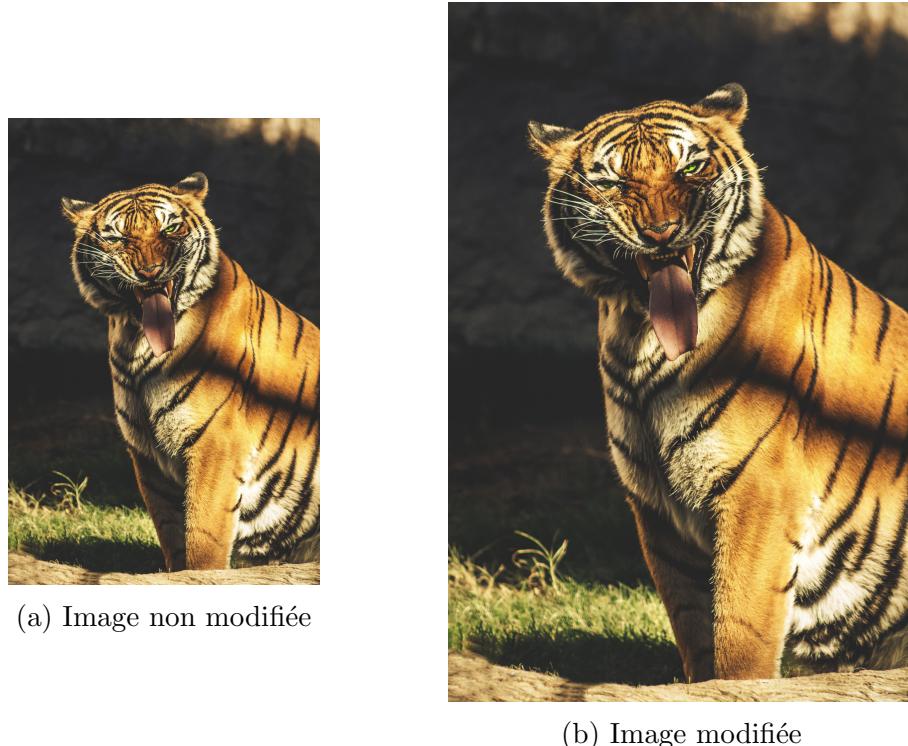


FIGURE 17 – Zoome *2

P3	P3
1280 1919	2560 3838
255	255

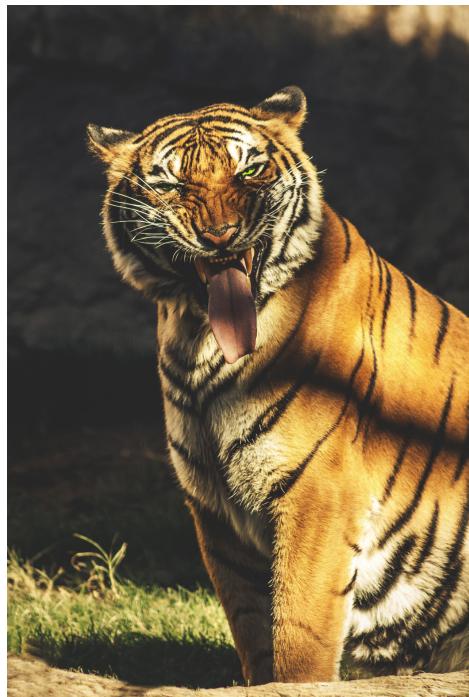
(a) Image non modifiée
(b) Image modifiée

FIGURE 18 – Dimensions des images

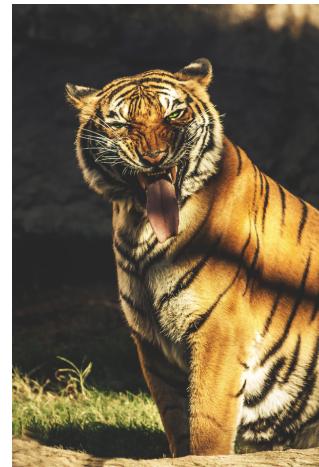
Notre fonction *zoomex2* créer une image avec des dimensions doubles de l'image initiale et l'agrandit.

4.3.4 ZoomeSur2

Le but de cette fonction est de réaliser un zoom sur 2 à notre image initiale, c'est-à-dire de la voir rétrécir.



(a) Image non modifiée



(b) Image modifiée

FIGURE 19 – ZoomeSur2

P3
1280 1919
255

(a) Image non modifiée

P3
640 959
255

(b) Image modifiée

FIGURE 20 – Dimensions des images

Notre fonction *zoomerSur2* créer une image avec des dimensions doubles de l'image initiale et le rétrécit.

4.3.5 Affichage de l'histogramme

Notre fonction *calculHist* nous classe dans l'ordre (de 0 à 255) le nombre de fois où chaque couleur est présente dans l'image. Pour cela, on initialise la valeur des effectifs (*lumi[lu] := 0*) pour chaque couleur et dès qu'une couleur est rencontrée , elle prend sa valeur +1. (*lumi[pe] := lumi[pe] + 1 ;*). Cette fonction est très utile, notamment dans le calcul du seuil. En effet, on cherche, pour le seuil, la valeur médiane du tableau. Or les effectifs sont rangés dans l'ordre, grâce à notre histogramme.

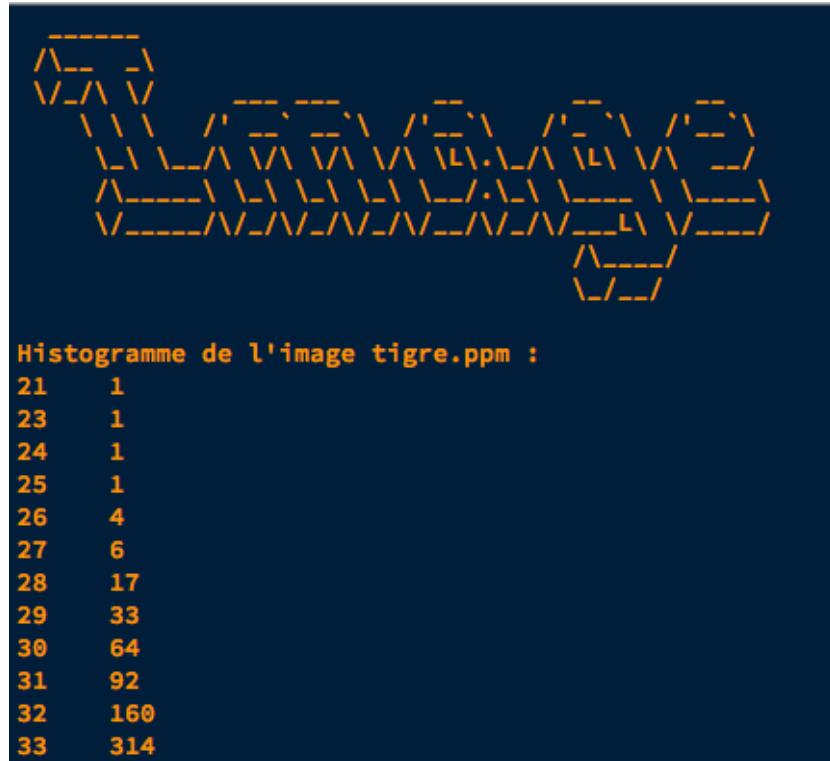


FIGURE 21 – Exemple de l'histogramme de l'image Tigre.png

4.3.6 Recadrage dynamique

Le but de cette fonction est de mieux percevoir une image. En effet si l'image est trop sombre, et bien son étendue de luminance va augmenter et l'on percevra donc mieux l'image.



(a) Image non modifiée



(b) Image modifiée

FIGURE 22 – Recadrage Dynamique

Pour cette fonction, on utilise d'abord une image en niveau de gris, ensuite on cherche grâce à la procédure «maximin» quelle est la luminance la plus élevée et la plus basse, à l'aide de notre histogramme. Une fois que l'on a notre niveau de gris le plus élevé et le plus faible, on applique à chacun des pixels la formule suivante pour calculer sa nouvelle luminance. « $(255/(maxi-mini))*(img.body[l,d].r-mini)$ »

4.3.7 Renforcement de contraste

Avant de vous expliquer l'intérêt de cette fonction, je tiens à préciser qu'elle ne marche pas correctement et nous ne comprenons pas pourquoi puisqu'en utilisant le même principe pour la fonction floue, elle marche. Le but de cette fonction est de renforcer le contraste de l'image en utilisant la matrice de convolution suivante.



FIGURE 23 – Renforcement de contraste

Notre fonction change, dans tout le tableau, la valeur pour chacun des 3 sous pixels par la valeur qui leur sont attribués par la matrice. C'est-à-dire qu'on multiplie par 5 la valeur du pixel en question : «(img.body[i,j].r » et on lui soustrait la somme du pixel se situant à gauche, en haut , en bas et à droite «(img.body[i,j-1].r + img.body[i-1,j].r + img.body[i+1,j].r + img.body[i,j+1].r) ;»

4.3.8 Floutage

Le but de cette fonction est de flouter une image. Pour flouter une image il suffit de prendre la moyenne des pixels au voisinage d'un pixel, y compris, et cela pour chacun des 3 sous pixels.



FIGURE 24 – Floutage

Notre fonction change, dans tout le tableau, la valeur pour chacun des 3 sous pixels, par la valeur moyenne du sous-pixel au voisinage de ce sous-pixel. C'est-à-dire qu'on additionne toutes les valeurs d'un pixel et de son voisinage pour la même couleur, et qu'on divise par 9 pour obtenir la moyenne. Voici le code pour le rouge par exemple : «`red := (img.body[i,j].r + img.body[i-1,j-1].r + img.body[i,j-1].r + img.body[i+1,j-1].r + img.body[i-1,j].r + img.body[i+1,j].r + img.body[i-1,j+1].r + img.body[i,j+1].r + img.body[i+1,j+1].r) div 9;`». (Nous ne voyons pas beaucoup le Floutage ici, je vous invite alors d'aller voir dans le dossier *Images PNG* ou *Images PPM* pour voir le vrai résultat).

4.3.9 Érosion

Le but de cette fonction est d'éroder l'image. C'est-à-dire que les pixels se situant au voisinage des pixels noirs uniquement deviennent noirs à leurs tours.

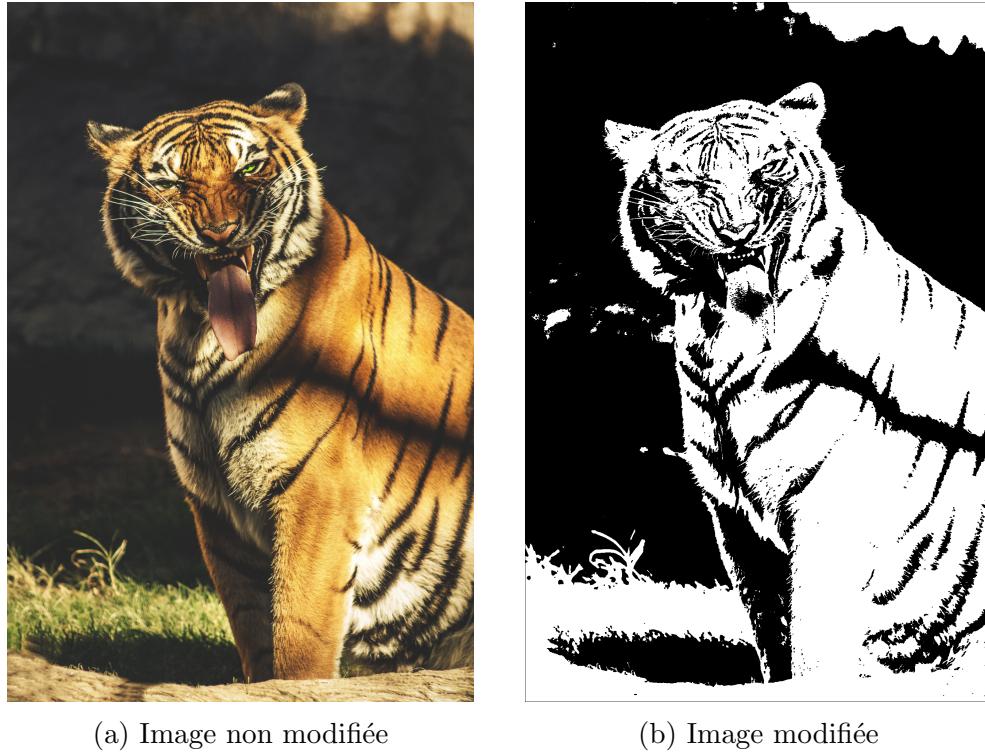


FIGURE 25 – Érosion

Pour cette fonction, nous binarisons tout d'abord l'image. Ensuite comme d'habitude, nous avons recopié les dimensions de l'image dans un tableau. Et en parcourant tout notre tableau, hormis les bords, dès que le pixel était noir, tous les pixels à son voisinage devenaient noirs également. Pour les autres pixels (blanc, car l'image est binarisée) nous ne les modifions pas, nous les recopions tel quelle.

4.3.10 Dilatation

Le but de cette fonction est de dilater l'image. C'est-à-dire que les pixels se situant au voisinage des pixels blancs uniquement deviennent blancs à leurs tours. C'est l'inverse de l'érosion.



(a) Image non modifiée



(b) Image modifiée

FIGURE 26 – Dilatation

Pour cette fonction, nous binarisons tout d'abord l'image. Ensuite comme d'habitude, nous avons recopié les dimensions de l'image dans un tableau. Et en parcourant tout notre tableau, hormis les bords, dès que le pixel était blanc, tous les pixels à son voisinage devenaient blancs également. Pour les autres pixels (noirs, car l'image est binarisée) nous ne les modifions pas, nous les recopions tel quelle.

4.3.11 Segmentation

Le but de cette fonction est de segmenter l'image. C'est-à-dire que si les pixels se situant au voisinage de notre pixel, y compris, sont majoritairement blancs, les pixels à son voisinage deviennent blancs à leurs tours. Et si les pixels se situant au voisinage de notre pixel, y compris, sont majoritairement noirs, les pixels à son voisinage deviennent noirs à leurs tours. La segmentation est donc la résultante de l'érosion et de la dilatation.

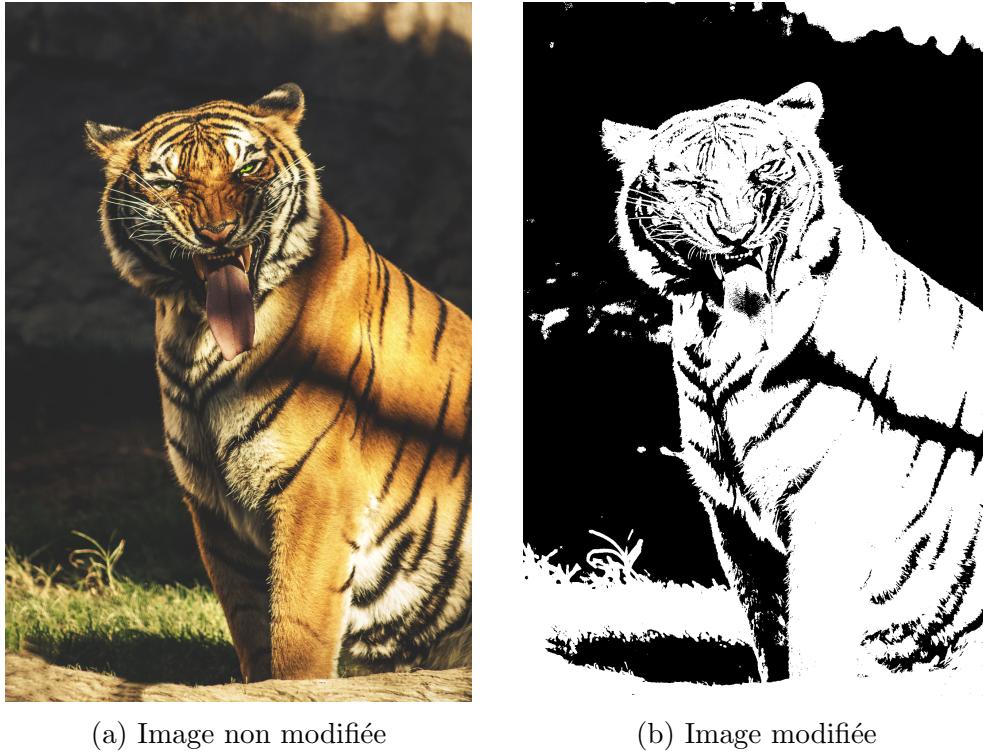


FIGURE 27 – Segmentation

Pour cette fonction, nous binarisons tout d'abord l'image. Ensuite comme d'habitude, nous avons recopié les dimensions de l'image dans un tableau. Nous avons créé la fonction MAXI, qui est là pour déterminer si autour d'un pixel , en le comptant lui-même, quelle couleur entre le noir et le blanc est le plus présent. Et en parcourant tout notre tableau, hormis les bords, on appelle notre fonction MAXI. Si MAXI détermine que le blanc est plus présent que le noir, tous les pixels à son voisinage deviennent blancs également. Si MAXI détermine que le noir est plus présent que le blanc, tous les pixels à son voisinage deviennent noirs également.

4.4 create.pas

Ce unit s'occupe de toutes les opérations de création d'images. Il est plus ou moins complexe, car nous n'avons pas procédé de la manière la plus simple. Nous allons maintenant expliquer comment les fonctions marchent. Tout d'abord comme dit précédemment, l'utilisateur entre les dimensions

de l'image et l'épaisseur de la croix. Le code vérifie tout d'abord si ce que *menuText.pas* a renvoyé est correct et est bien que des chiffres en appliquant un *TryStrToInt*. Ceci fait que si l'on peut le convertir en INTEGER, alors il le fera et renverra un booléen, mais si ceci n'est pas possible alors cela renvoie un faux. Et tant qu'on ne peut pas le convertir en INTEGER alors le programme va continuer à demander à l'utilisateur d'entrer les dimensions demandées. Pour ensuite créer la croix, nous avons fait :

$$\left[\frac{\text{largeur}}{2} - \frac{\text{epaisseur}}{2}; \frac{\text{largeur}}{2} + \frac{\text{epaisseur}}{2} \right] \quad (\text{largeur de croix})$$

$$\left[\frac{\text{hauteur}}{2} - \frac{\text{epaisseur}}{2}; \frac{\text{hauteur}}{2} + \frac{\text{epaisseur}}{2} \right] \quad (\text{hauteur de croix})$$

Cela nous permet d'avoir l'intervalle dans lequel il faut appliquer du noir et ça à l'aide de boucles *FOR ... TO ... DO*. Quand une fraction ne tombait pas sur un nombre exact c'est à faire à un reste avec une division euclidienne nous avons pris que la partie entière du nombre.

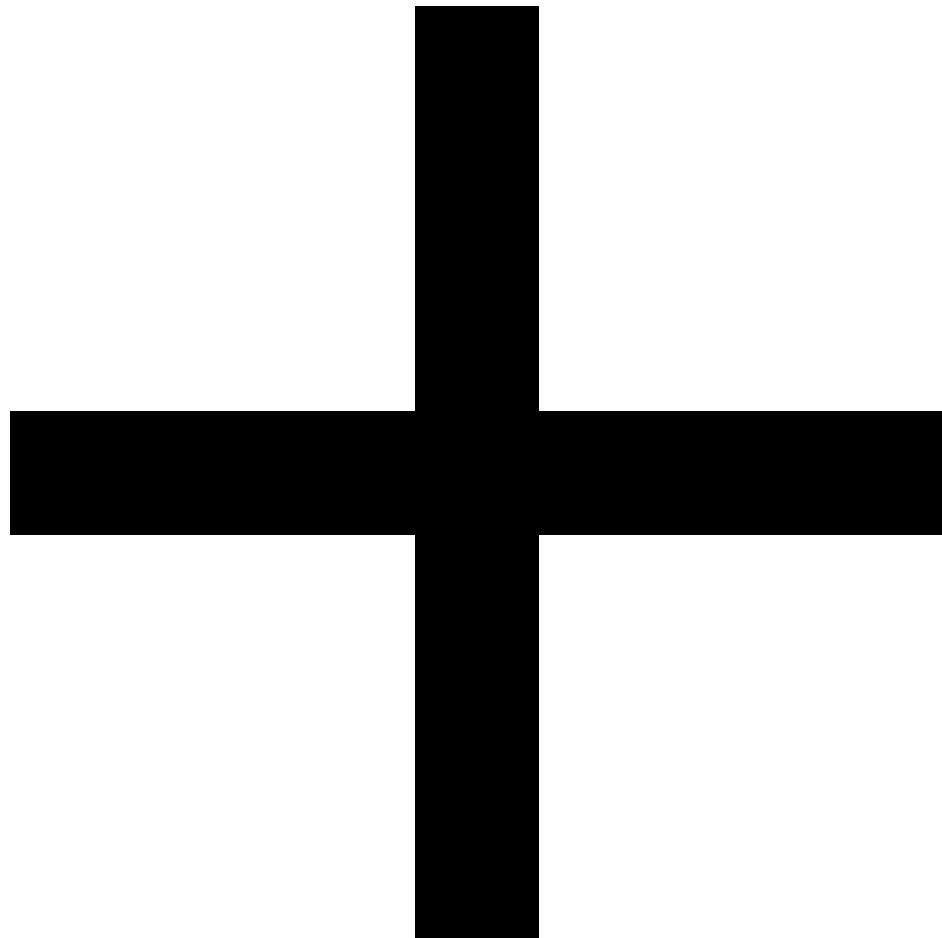


FIGURE 28 – Croix de 300*300px avec une épaisseur de 40px

5 Limite du programme

Malheureusement le projet comporte quelques défauts ; il n'est pas fini.

5.1 Fonctions qui modifient l'image

5.1.1 Renforcement de contraste

Notre fonction ne nous donne pas l'image attendu après l'application.

5.2 Fonctions non fonctionnelles

Vous pouvez trouver les deux fonctions suivantes, à la fin de l'Unit «ManipulationBasique.pas»

5.2.1 Convolution

Ne l'utilisant pas dans une autre fonction de notre programme, nous ne savons pas si elle est opérationnelle ou non.

5.2.2 Opérateur de chanda

Nous avons travaillé sur cette fonction avec le groupe d'Antoine Denis. Mais elle n'est pas opérationnelle.

6 Bilan Personels

6.1 David RIGAUX

Dès que j'ai eu l'énoncé du projet, je me suis directement lancé dans l'étude de celui-ci. À la première lecture, j'ai vu le nombre de pages, je me suis dit que ça allait être très dur, et ça a été le cas. Au début de la programmation, je ne savais pas dans quelle direction aller, par où commencer. J'ai donc pris la décision de commencer par faire la gestion des paramètres à l'aide de la commande *paramstr*. J'ai trouvé ça extrêmement intéressant, car cela fonctionnait vraiment comme un vrai programme qu'on télécharge d'internet, cela a était une immense motivation pour moi. Après avoir compris comment cela marche je me suis lancé dans la création du menu Visuel. Il était demandé dans l'énoncé de faire un menu qui tourne en boucle jusqu'à ce que l'utilisateur sort du programme, cela voulait également dire qu'aucune erreur ne devait survenir pendant qu'il fonctionnait. J'ai donc réfléchis longuement comment je pouvais faire pour qu'il marche sans problème. Je suis

enfin tombé sur le raisonnement qu'un STRING peut contenir des chiffres et des lettres, mais qu'un INTEGER que des chiffres, donc pour éviter toute erreur pendant l'utilisation j'ai décidé de n'utiliser que des strings pour toutes les entrées de l'utilisateur et ensuite les convertir, s'il le faut en INTEGER. Il m'est venu à l'idée de faire un beau menu qui soit agréable à voir. Je suis donc allé sur un site qui m'écrivait un texte en grand comme on peut le voir sur les menus. Ce projet m'a permis de reprendre toutes les notions que nous avons apprises en classe depuis le début de l'année et a contribué à l'approfondissement du sujet. La manipulation d'image a toujours un de mes intérêts, c'est pour cela que faire ce programme à toujours était intéressant. Cependant, j'ai rencontré énormément de difficultés pendant le codage, car j'ai rencontré énormément d'erreur de compilation et d'erreur pendant l'exécution du programme. Le codage du lancement du programme par la commande *./image* avec des paramètres après, n'as pas été aboutit, car je n'ai pas réussi à le faire. J'ai eu du mal à associer l'ordre des paramètres dans le programme, à part *-h* pour afficher le menu d'aide que j'ai réussi à coder. J'ai appris à mieux trouver les erreurs en utilisant des marqueurs de position dans le programme pour pouvoir trouver l'erreur et la corriger. Je me suis personnellement occupé de la création du menu, de l'entrée et de la sortie et également la création de l'image.

6.2 Soner CAVUSOGLU

Lorsque l'on m'a donné l'énoncé du projet, j'ai lu la première page, la seconde, puis j'ai lu en diagonale les autres pour m'imprégner l'idée générale du projet. Une fois arrivé au bout je me suis demandé par où commencer ? Voyant que le travail était conséquent, je me suis mis à réécrire le projet. C'est-à-dire que mes premières séances se résument à comprendre le sujet et à le développer. Pourquoi le développer me direz-vous. Eh bien en le lisant plusieurs fois, j'ai rencontré des termes dont je connais vaguement le principe, et certains pas du tout. En le développant, je comprenais de mieux en mieux ce qui nous était demandé et j'en savais déjà plus qu'au départ ce que la majorité des fonctions requérait. Lorsque j'eu fini, David avait bien entamé la création du Menu. Je me suis donc mis à coder les premières fonctions telles que converties en niveaux de gris et j'ai continué à m'occuper des manipulations basiques de l'image. Cependant lors du codage j'ai rapidement rencontré des problèmes auxquels je n'y avais pas pensé tel que pour l'érosion : lorsque l'on appliquait la fonction au bord, on sortait des dimensions. Ainsi il fallait

appliquer la fonction à partir des dimensions - 1pixel, comme s'il y avait un bord. Il en valait de même pour la dilatation et la segmentation. D'autres problèmes me sont survenus dû à une compréhension incomplète de l'énoncé. Heureusement qu'il y'avait de l'entraide au sein de notre classe, et même en dehors. On pouvait réfléchir à plusieurs et confronter nos idées. Et par la suite demander à Mr Bart laquelle de nos propositions était la bonne. Cela était très utile puisqu'il nous permettait de ne pas nous lancer dans de mauvais chemins et de développer des fonctions en trop, « inutile ». Un dernier problème nous a suivis du début jusqu'à la fin, c'était la retranscription de mon code avec les variables de David, il suffisait d'une seule erreur, et l'on était perdu plus d'une heure à trouver la solution. Il reste malheureusement des limites à notre programme : le projet n'est pas fini. Nous avons une fonction qui modifie l'image « Renforcement de contraste » ; mais elle ne nous donne pas l'image attendue après l'application. Nous avons également deux fonctions non fonctionnelles que vous trouverez, en commentaire, à la fin de l'Unit « ManipulationBasique.pas ». Il s'agit de la « Convolution » et de « l'Opérateur de Chanda ». En effet, la fonction convolution n'est pas utilisée dans une autre fonction de notre programme (bien qu'elle pourrait être utile au renforcement de contraste) nous ne savons pas si elle est opérationnelle ou non. Et pour la fonction Opérateur de Chanda, nous avons travaillé avec le groupe d'Antoine Denis ensemble, mais elle n'est pas opérationnelle. Malgré que le projet ne soit pas fini, je suis très content d'avoir pu travailler sur ce projet intéressant. Pour ma part, chaque problème rencontré m'a donné de la motivation pour aller aussi loin. En effet, la résolution d'un petit problème m'encourage à aller plus loin. Et celle d'un grand problème permet de travailler à plusieurs sur la résolution de celui-ci. Le travail entre équipes est vraiment quelque chose de remarquable et qui me fait plaisir : on peut avancer tout en s'amusant. La réalisation de ce projet m'a permis de revoir la quasi-totalité des notions en informatique et d'en découvrir davantage sur les traitements de l'image.

7 Conclusion

À première vu on pense que le projet est impossible, mais avec le dur travail on se rend compte d'énormément de choses sur le monde de la programmation. Avec ce projet, nous avons appris à travailler en équipe, à écouter la personne avec qui vous travaillez, mais également de faire confiance

en cette personne pour son travaille. Malgré un démarrage lent, dû à des difficultés de répartition de travail, mais également ne sachant pas par quelle partie commencer, nous avons réussi à faire le projet ,même si incomplet, avec un résultat satisfaisant. Nous sommes ravis d'avoir réalisé ce projet. Nous avons surpassé ensemble les difficultés rencontrées et en avons appris davantage sur le traitement de l'image.