



## **RAPPORT DE PROJET**

# Traitement d'image : *reconnaissance de forme par apprentissage*

Dans le cadre de l'U.E  
**3E103**

Proposée en  
**LICENCE L3 EEA**

ANNÉE  
**2015-2016**

Par  
**Charles PAULAS VICTOR - Kamel MELLAH - Houria  
KEZADRI - Meriem SAIDI - Ebenezer AFARI**

# Sommaire

- I. [Problèmes et résolutions](#)
  - A. [Introduction](#)
  - B. [Apprentissage](#)
    - 1. [Acquisition des images et traitements préliminaires](#)
    - 2. [Moment géométrique](#)
    - 3. [Moment de Legendre](#)
    - 4. [Calcul de distance](#)
    - 5. [Résultats](#)
- II. [Structures et bibliothèques propre au projet](#)
  - A. [Administration de la base de donnée](#)
  - B. [Rôle des bibliothèques](#)
- III. [Bugs connus](#)
- IV. [Améliorations du projet](#)
  - A. [Fonction "Glisser-Déposer"](#)
  - B. [Affichage des Images références](#)
  - C. [Apprentissage en continue](#)
  - D. [Images en couleur](#)
- V. [Bibliographie](#)
- VI. [Annexes](#)

## **I. Problèmes et résolutions**

### **A. Introduction**

Durant ce projet, nous avons décidé de travailler sur la reconnaissance de forme. Le sujet étant vaste, nous nous sommes fixés l'objectif suivant : concevoir un programme en C afin de reconnaître des caractères (dans le sens typographique du terme).

Ainsi, nous avons cherché à créer un programme, dans lequel l'utilisateur proposerait le chemin vers une image, au format bitmap, dotée d'un fond noir et d'un caractère blanc et la machine répondrait par la lettre correspondante à l'image.

La stratégie adoptée pour cette réalisation s'appuie sur une méthode d'apprentissage en deux phases.

Premièrement, nous concevons un programme capable d'analyser certains aspects d'une image, ici les formes. Cette analyse est sauvegardée en mémoire (sous forme de fichier texte ou binaire). Or, il suffit d'associer à une analyse le nom d'une lettre, et ainsi donner du sens à la donnée afin de l'exploiter dans notre stratégie. Nous appellerons cette phase, l'apprentissage. Après avoir répété l'opération sur plusieurs images (par exemple, autant de fois que de lettre dans l'alphabet, soit 26 fois), nous obtenons un programme qui est doté d'un "savoir" concernant les caractères. Nous appellerons ce "savoir" aussi base de données. Comme toute base de données, une indexation efficace des données est nécessaire pour permettre une recherche efficace dans le "savoir".

La seconde phase consiste à tester notre programme sur des images qu'il n'a jamais traité durant sa phase d'apprentissage, comme on pourrait demander à un étudiant de proposer un algorithme durant un examen. Tel un bon élève, le programme doit nous proposer la lettre associée à l'image proposée durant le test, en s'appuyant sur son "savoir".

### **B. Apprentissage**

#### **1. Acquisition des images et traitements préliminaires**

##### **a) Bitmap**

Le Bitmap (.bmp) est un format d'image, tout comme le .jpg ou le .png. Sous ce format, les images sont stockées sous forme de matrice de pixels. Sachant que l'intensité d'un pixel est comprise entre 0 et 255, on peut établir trois matrices RVB (Rouge, Vert, Bleu) associées. Cela permet ainsi une exploitation plus facile. Tout au long de ce projet, l'utilisateur, devra obligatoirement fournir une image de type .bmp. Notre traitement de la problématique ne prend en charge que ce format.

Nous avons à notre disposition dans ce projet une bibliothèque, « MyLibBmp », qui permet de traiter beaucoup plus facilement ces types d'image.

## b) Image binaire

Comme nous l'avons indiqué précédemment, notre « savoir » ou base de données sera exclusivement composée d'images de caractères blancs sous fond noir. Sous cette considération, les matrices des composants R, V, et B sont identiques.

La fonction `readBmpImage`, intégrée à la bibliothèque `MyLibBmp.h` permet de récupérer des données associées à l'image. Des données telles que la taille de l'image, les coordonnées x et y, et aussi les composante RVB ont pu être ainsi récupéré. Puisque les matrices R, V, ou B ont des valeurs qui sont soit 0 ou 255, nous avons dû les normaliser en 0 et 1. L'algorithme ci-dessous a permis de réaliser cette opération:

```
For i=0 à X
    For j=0 à Y
        Si ImBmp[i][j] = 255
            ImBmp[i][j] = 1
        End if
        Sinon
            ImBmp[i][j] = 0
        End sinon
    End for
End for
```

Pour éviter d'avoir des dimensions d'images gigantesque, nous avons dû redimensionner les images de la base de données. Les nouvelles abscisses et ordonnées sont toutes inférieures ou égales à 40 pixels. L'image affichée après compilation et exécution du programme, est identique, à l'image de base, sauf que celle-ci est une matrice de 0 et de 1.

Pour des raisons d'efficacité et d'optimisation des ressources de la machines, nous ne pouvons nous permettre de travailler avec des matrices 2D composées de 0 et 1. Nous verrons dans la partie suivante l'une des solutions envisageables.

## 2. Moment géométrique

Une image possède plusieurs caractéristiques: couleurs, formes, textures ... Ici, nous voulons seulement travailler avec les formes de l'image, par conséquent nous avons besoin d'extraire de l'image les informations concernant la forme. Ainsi, nous allégeons la quantité de données à traiter pour la suite (gain d'efficacité).

Pour ce faire, nous utilisons des descripteurs mathématiques et plus particulièrement les moments (géométrique puis de Legendre).

### a) Calcul des moments

L'expression mathématique des moments géométriques s'écrit ainsi:

$$M_{p,q} = \iint_{\Omega} x^p y^q f(x,y) dx dy.$$

Le moment géométrique (qu'on appellera MG) se caractérise par les valeurs P et Q dont la somme nous donne l'ordre du MG. L'ordre 0 représente l'aire de la forme. Lorsqu'on lui associe les calculs MG d'ordre 1, on peut déterminer le centre de gravité de la forme.

Les coordonnées de ce point s'écrivent ainsi:

$$x_c = \frac{m_{1,0}}{m_{0,0}} \quad \text{et} \quad y_c = \frac{m_{0,1}}{m_{0,0}}$$

Nous travaillons ici avec des images représentables par des objets mathématiques tels que des distributions bidimensionnelles matricielles d'intensité (nommé f(x,y) dans la formule et codé entre 0 et 1, noir ou blanc), par conséquent nous pouvons traduire la double intégrales en une double sommes. Le résultat de la traduction se présente ainsi:

$$m_{p,q} = \sum_{p=0}^m \sum_{q=0}^n x^p y^q f(x,y)$$

Cette double sommes est plus simple à implémenter que la double intégrales. Le code en C, issu de cette formule, est la suivante:

```
// double sommes est implémentables par deux boucles for imbriquées
int calcul_moment(int **image, int dimX, int dimY, int p, int q){
    int x,y,intgr=0,inty;
    for(x=0; x<dimX; x++){ // parcours les lignes de la matrice image
        {
            inty=0;
            for(y=0; y<dimY; y++) //parcours les colonnes de la matrice image
            {
                inty+=puissance(x,p)*puissance(y,q)*image[x][y]; /* calcule par "accumulation"
de la somme à chaque pixel (ou element de la matrice image) */
            }
            intgr+=inty; // calcul de la deuxieme somme qui vient "encapsuler" la première
        }
    }
    return intgr; // retourne le résultat de la double somme sur le support entier
}
```

Le résultat obtenu de cette double somme est un scalaire qu'il faudra ensuite stocker dans une matrice triangulaire dont les dimensions dépendent de l'ordre du moments géométrique voulu.

## b) Conception d'une matrice triangulaire avec les moments

Pour manipuler les moments géométriques des images dont on dispose, nous les plaçons dans une matrice dont la taille dépend de l'ordre maximal choisi  $p+q$ . L'ordre maximal correspond à la valeur maximale de la quantité  $p+q$  pour laquelle on calcule le moment géométrique d'une image. Au-delà de cet ordre, on ne calcule plus le moment de l'image. La matrice ainsi construite sera triangulaire.

Pour réaliser cette matrice, on a créé une fonction "matricemoment". Cette fonction doit retourner une matrice de type `float**` (une matrice de type `int**` provoque de l'overflow). Les arguments de la fonction sont : l'image elle-même, ses dimensions et l'ordre choisi (donné par  $p$  et  $q$ ). Pour éviter toute erreur de segmentation, on fixe une variable  $M$  valant deux fois l'ordre maximal. On alloue de la mémoire pour la matrice avec une taille  $M*M$ . Puis on parcourt la matrice de sorte à la remplir. Si les indices de la case qu'on souhaite remplir est plus grand que l'ordre maximal, on y met 0, sinon on y met le moment géométrique calculé pour ce couple  $(p,q)$ .

```
float** matricemoment(int** image,int dimX,int dimY,int p, int q){
    int i,j;
    float M=2*(p+q);
    float** mat=calloc(M, sizeof(float*));
    for(i=0;i<M;i++){
        mat[i]=calloc(M, sizeof(float));
    }
    for(i=0;i<M;i++){
        for(j=0;j<M;j++){
            if((i+j)>(p+q)){
                mat[i][j]=0;
            }else{
                mat[i][j]=calcul_moment(image,dimX,dimY,i,j);
            }
        }
    }
    return mat;
}
```

Concernant la principale difficulté rencontrée, elle fut de déterminer la taille de la matrice. Au départ nous avons envisager de parcourir les abscisses et les ordonnées de la matrice à remplir de 0 à  $(p+q)-1$ . Lors de l'allocation de mémoire pour la matrice à remplir, nous avons donc mis une taille de  $(p+q)*(p+q)$ . En procédant ainsi nous avons eu des erreurs de segmentation, et l'apparition dans la matrice de valeurs aberrantes. Pour parer à ce problème on a utilisé une taille égale à deux fois l'ordre maximal.

## 3. Moment de Legendre

Les moments de Legendre permettent de travailler sur des formes non centrées, ayant subies des transformations géométrique telles que des rotations, des translations ou des homothéties (réduction ou agrandissement de la forme). Cela veut dire que

l'utilisateur gagne en liberté par rapport au moment de géométrie concernant le choix de l'image à proposer à notre programme.

#### a) Polynôme de Legendre

Le calcul des polynômes de Legendre pour des valeurs comprises entre -1 et 1 permet d'avoir une base orthogonale. Ainsi, cela nous permettra de pallier aux problèmes de redondance d'information qui pourrait survenir après le calcul des moments géométrique.

La formule de base du polynôme de Legendre, est tel que représenté ci-dessous :

$$P_n(x) = \frac{1}{2^n n!} \frac{d^n}{dx^n} ((x^2 - 1)^n)$$

Son implémentation en langage C étant fastidieux, nous avons donc utilisé un polynôme associé au celui de Legendre. C'est la récurrence de Bonnet :

$$(n + 1)P_{n+1}(x) = (2n + 1)xP_n(x) - nP_{n-1}(x).$$

Nous avons écrit une fonction Pol\_Legendre, permettant de calculer récurrence de Bonnet. Cette fonction prend en paramètre un **ordre n** et **l'intervalle x** compris entre -1 et 1.

L'algorithme suivant nous a permis d'effectuer le calcul avec succès.

```
Variables :  
  i : Entier  
  P[n] : tableau 1D de réels  
  P0 à 1  
  P1 à x  
  Pour i=1 à n  
    Pi+1 = (2*i+1)*x*Pi - i*Pi-1  
    Pi+1 = Pi+1/(i+1)  
  End for  
Retourner Pn
```

P<sub>n</sub> ici est la dernière case du tableau P, et le polynôme de Legendre que nous souhaitons calculer.

#### b) Calcul des moments

Notre représentation des formes est fondée sur les moments de Legendre de leur fonction caractéristique, rassemblés dans un vecteur appelé descripteur.

$$\lambda_{p,q} = C_{pq} \iint P_p(x) P_q(y) dx dy$$

(p+q) est appelé l'ordre du moment et (C<sub>pq</sub>) est une constante de normalisation. Cette approche régions, permet un calcul simple et rapide. Et le choix d'une base orthogonale permet d'obtenir une description hiérarchique des formes, dont la précision augmente avec l'ordre N, en pratique, pour choisir l'ordre N, on effectue une étude de l'erreur quadratique moyenne entre la fonction caractéristique de la forme de référence et sa reconstruction.

Actuellement, l'ordre N jusqu'au quel nous choisissons de calculer les moments de la forme de référence est déterminé par inspection visuelle à partir d'expériences de reconstruction, les polynômes de Legendre constituant une base orthogonale de projection, la formule de reconstruction est immédiate :

$$I(x, y) = \sum_{p,q}^{p+q \leq N} \lambda_{p,q} P_p(x) P_q(y)$$

Nous allons utiliser la formule suivante pour calculer plus facilement ces moments :

$$\lambda_{mn} = \frac{(2m+1)(2n+1)}{4} \sum_x \sum_y P_m(x) P_n(y) P_{xy}$$

Dans notre cas, m et n sont p et q respectivement, et P<sub>xy</sub> représente l'image associé.

Pour pouvoir calculer les moments de Legendre, il faut d'abord introduire deux fonctions affine, du type f(x) = ax + b et f(y) = ay + b, où ax = 2/(x-1), ay = 2/(y-1), et b = -1 ; Notons que x et y ici sont les coordonnées de l'image. Pour tout x et y, f(x) et f(y) sont compris entre -1 et 1. Cette linéarisation des coordonnées x et y nous est particulièrement utile dans la mesure où, pour le calcul de polynômes de Legendre exploitable dans notre application, il faut que x et y sont compris entre 0 et 1. En d'autres termes, f(x) et f(y) deviennent p et q, en considérant la formule ci-dessus.

L'étape de la linéarisation terminée, nous pouvons calculer maintenant les moments de Legendre.

Dans un premier temps, Nous allons stocker dans une matrice 2D les moments de tous les pixels composant l'image. On observe que par rapport à la diagonale, tous les éléments d'une part de la diagonale, à son opposé sur l'autre part de la diagonale. Le calcul du moment total serait donc nul si on considère la matrice complète. Pour cette raison, nous ne considérons que la moitié de la matrice par rapport à la diagonale. Concrètement, la matrice obtenue sera une matrice triangulaire.



L'algorithme pour calculer les moments de Legendre est le suivant :

```
for p = 0 à N // N ici est l'ordre souhaiter
    For q = 0 à N-p
        Cpq = (2p+1)(2q+1)/4
        Legendre[p][q] = Cpq * Pp(x)*Pq(y)*image[p][q];
        Moment = moment + Legendre[p][q]
    End for
End for
```

$P_p$  et  $P_q$  sont les polynômes de Legendre d'ordre  $p$  et  $q$  en tout points de  $x$  et  $y$  compris entre -1 et 1. La matrice 2D ainsi obtenue, est de forme triangulaire.

L'information contenue dans une matrice de similarité se trouve principalement dans les blocs à forte valeur de similarité, ce qui indique une répétition dans la structure de l'objet, à un niveau local, les diagonales de similarité, plus particulièrement, dévoilent une « non similarité » deux régions d'une suite de valeurs différentes. La matrice de forme triangulaire se construit en fait selon un principe cognitif de rétention mémorielle de l'information. La confrontation de durées et de distances nécessite un axe dédié non pas au moment, mais à la différence entre moments.

#### 4. Calcul de distance

La reconnaissance de forme à partir d'une image de référence implique d'établir un moyen de comparer la forme qu'on souhaite identifier avec la forme déjà référencée. Pour établir cette comparaison, nous passons par le calcul de distance. Le calcul de distance consiste à calculer la différence terme à terme entre deux matrices de moments. L'image de référence la plus proche de l'image qu'on souhaite identifier est celle dont la matrice de moment a la plus petite distance de toutes les distances calculées.

Pour calculer la distance entre deux matrices, nous avons implémenter une fonction "diffmom" qui prend en argument les deux matrice ainsi que l'ordre choisit de manière à disposer de la taille de la matrice. La fonction "diffmom" calcul la différence terme à terme entre les deux matrices puis renvoi la distance maximale calculée de manière à connaître par la suite quelle matrice de moment est "la moins éloignée" de la matrice de moment qu'on souhaite identifiée.

Pour trouver l'image la plus proche, nous avons créer la fonction "compare" qui prend en argument :

- Un pointeur sur la structure de l'image qu'on cherche à identifier. On dispose donc de toutes ses caractéristiques.
- La liste chaînée, qui est la base de donnée.
- Le nombre d'images référencées. Étant donné que la base de donnée est établie à l'avance et qu'elle n'évolue pas dans le programme, on connaît leur nombre.

La fonction "compare" parcourt la liste chaînée, et utilise la fonction "diffmom" qui renvoi la distance maximale entre deux matrices, de sorte à calculer la distance maximale entre chaque matrice de moment référencée et la matrice de moment de l'image qu'on souhaite reconnaître. Chaque maximum est placé dans un tableau. Puis on recherche la

plus petite distance de tous les maxima. On récupère alors l'indice de la case correspondant à ce minimum. L'indice donne aussi l'emplacement de l'image correspondante dans la base de donnée. Ainsi, on peut associer l'image entrée par l'utilisateur à l'image référencée qui lui est la moins distante. La fonction renvoi le caractère reconnu.

## 5. Résultats

D'une part il n'y a pas d'attributs universels et d'autre part le choix des descripteurs dépend fortement de la base d'images à utiliser et des connaissances à priori qu'on peut avoir sur la base, et qu'elle que soit la mesure de distance utilisée cette distance et en fonction des caractéristiques choisies.

L'utilisation des moments géométrique permet de représenté les propriétés spatiales de la distribution des pixels, une méthode facile à calculer et à implémenter, très sensible au bruit et aux déformations, les résultats ne sont pas normalisées avec un temps de calcul très long.

En revanche l'utilisation des moments de Legendre donne des résultats normalisées, un bas ordre et suffisant pour représenter la forme globale de l'image à reconstruire.

## II. Structures et bibliothèques du projet

### A. Administration de la base de donnée

La constitution d'une base de donnée d'images de référence est indispensable. Elle permet de disposer facilement d'une série d'images analysées et indexées. Nous avons choisis de créer cette base de donnée sous forme d'une liste simplement chaînée. La structure de donnée utilisée fut la suivante :

```
struct image {
    unsigned int dimx ;// dimension x
    unsigned int dimy ;// dimension y
    unsigned char** img ;// distribution bidimensionnelle de l'intensité
    float** momg; // matrice de moments géométrique
    double** momleg;// matrice de moment de legendre
    char lettre; // le nom de la lettre (information sémantique de l'objet)
    image* isuiv ; // pointeur du l'element suivant de la chaîne
};
```

Les champs à renseigner sont donc les propriétés de l'image : ses dimensions, sa matrice de moments géométriques et de Legendre, l'image elle-même et la description de l'image (pour l'image d'un A, la description sera le caractère 'A').

Pour manipuler la liste chaînée des images de référence, nous avons écrit trois fonctions:

- La première "newlm" permet de créer une structure pour une image donnée en argument, lui allouer de la mémoire et d'initialiser la structure. La fonction renvoi un pointeur sur la structure nouvellement créé.
- La seconde "entete" permet d'insérer en tête de liste une nouvelle image (structure) dans la base de donnée (liste chaînée). Nous nous sommes suffit à cette fonction pour insérer une image dans notre base de donnée car nous n'avons trié les images référencées.
- La troisième "freeimage" permet de libérer la mémoire précédemment allouée à la structure "image".

## B. Rôle des bibliothèques

Nous utilisons dans ce projet principalement quatre bibliothèques (mis à part la bibliothèque standard stdio.h):

- myLibBmp permet de manipuler des fichiers image au format .bmp (cette bibliothèque a été fournis par le professeur).
- bin\_image permet fait figure d'interface d'acquisition pour les images. Elle propose aussi des outils de binarisation de l'image. Cette bibliothèque s'appuie sur myLibBmp.
- imagerec1\_2 nommé en référence à "Image recognition" regroupe toutes les fonctions concernant les moments géométriques. Cette bibliothèque s'appuie sur myLibBmp.
- Legendre propose des outils de calcul spécifiques aux polynôme de Legendre et au moment de Legendre. Cette bibliothèque s'appuie sur bin\_image

## III. Bugs connus

En sortie de la fonction 'binarisation\_image', on s'était rendu compte que la matrice d'image binaire était inexploitable, car à chaque fois, l'application code bloc cessait de fonctionner.

Nous avons émis les hypothèses suivantes dans l'espoir de résoudre le problème :

- Faille au niveau de l'allocation d'espace de la matrice 2D. L'allocation étant faite directement dans la fonction, nous avons décidé d'allouer l'espace mémoire directement dans la fonction 'main' (bien sûr, nous avons supprimé les allocations à l'intérieur de la fonction). Cela fut sans succès.
- Mauvaise libération de l'espace mémoire. Tout comme pour la phase d'allocation, la libération de la mémoire était directement gérer par la fonction 'binarisation\_image', et nous re-effectué le processus dans la fonction main, sans

succès apparent aussi. Jusqu'à ce jour, c'est un bug que nous n'avons pas pu résoudre.

- La bibliothèque proposait pour l'interprétation des images bitmap fournit une matrice de l'image 2D dans le type "char". Or pour appliquer nos méthodes numériques, il nous faut des données du types "int" ou "double". Pour cela, nous avons conçu la fonction Char2Int. Malheureusement, cette fonction présente des bugs que nous avons pas eu le temps de corriger entraînant l'incapacité de faire fonctionner notre programme générale.

## **IV. Améliorations du projet**

### **A. Fonction "Glisser-Déposer"**

Pour une meilleur interaction Homme/Machine, on pourrait créer une interface graphique qui permettrait de faire un "glisser déposer" de l'image bitmap dans une zone de la fenêtre de l'interface, conduisant à une réponse immédiate de la machine. Ainsi l'utilisateur n'a pas besoin de spécifier l'emplacement de l'image.

### **B. Affichage des Images références**

Notre programme, après avoir effectué la détection, pourrait présenter, en plus de la lettre correspondante, l'image de référence de la base de données.

### **C. Apprentissage en continue**

Si le résultat retourné par le programme est correct (avec une validation humaine), le programme pourrait rajouter le résultat de ce test au sein de ça base de donnée pour gagner en agilité lors d'un nouveau test.

### **D. Images en couleur**

Notre programme permet de travailler avec des images noir et blanc. Mais si l'utilisateur nous propose des images provenant d'une affiche publicitaire par exemple, notre programme devrait être capable de travailler avec le formalisme des images en couleur. En effet, les images bitmap couleurs sont la superposition de trois couches (Rouge, Vert et Bleu : RGB). Nous devons donc ajouter à notre programme des fonctions de recomposition des couches RGB et répéter trois fois nos calculs pour chaque images.

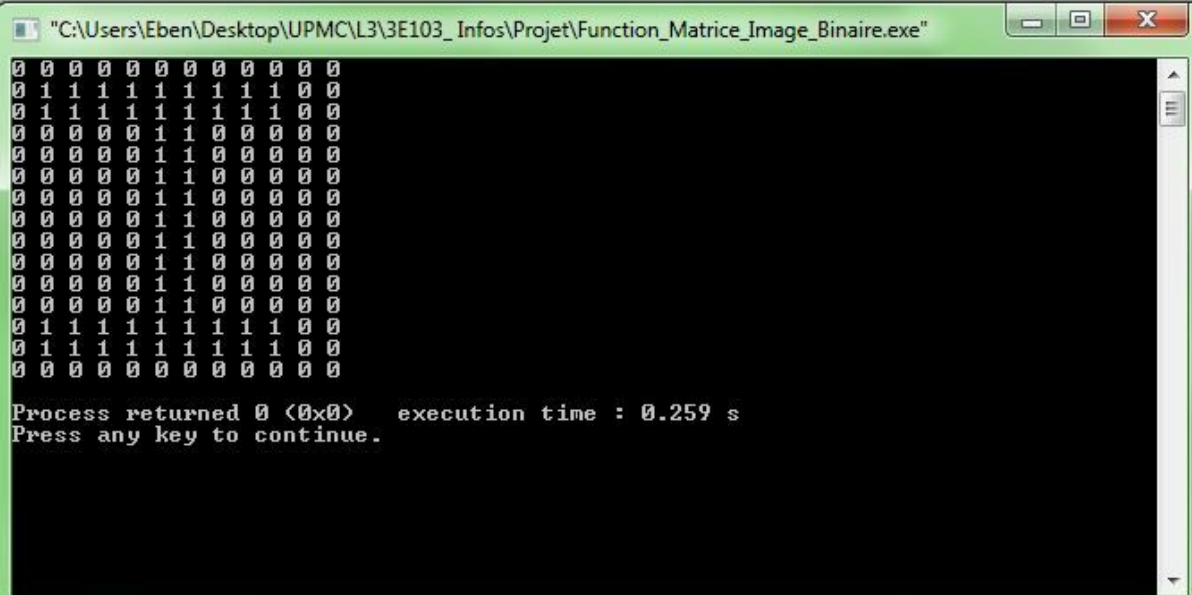
## **V. Bibliographie**

- Recherche d'image par le contenu par Saïda Bedouhene  
<http://www.ummto.dz/IMG/pdf/these-2.pdf>

- Descripteur de formes :

<http://perso.univ-lr.fr/kkimchen/fichiersPDF/rapportThese2.pdf>

## Annexes



"C:\Users\Eben\Desktop\UPMC\L3\3E103\_Infos\Projet\Function\_Matrice\_Image\_Binaire.exe"

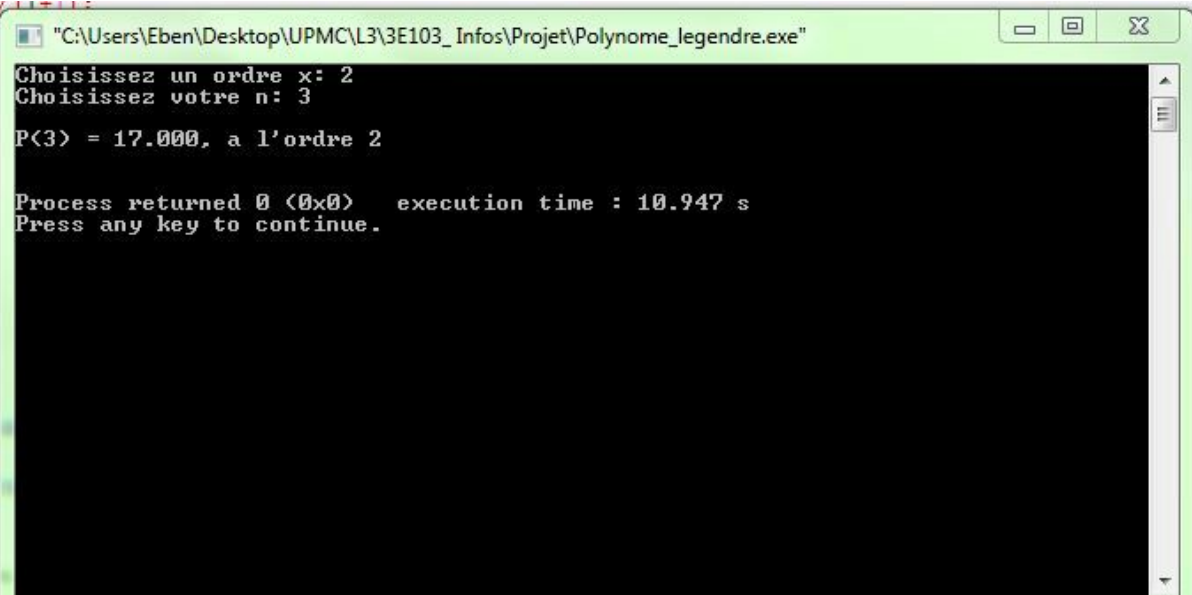
```

0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 1 1 1 1 1 1 0 0 0
0 1 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 1 1 0 0 0 0 0
0 0 0 0 0 1 1 0 0 0 0 0
0 0 0 0 0 1 1 0 0 0 0 0
0 0 0 0 0 1 1 0 0 0 0 0
0 0 0 0 0 1 1 0 0 0 0 0
0 0 0 0 0 1 1 0 0 0 0 0
0 0 0 0 0 1 1 0 0 0 0 0
0 0 0 0 0 1 1 0 0 0 0 0
0 0 0 0 0 1 1 0 0 0 0 0
0 0 0 0 0 1 1 0 0 0 0 0
0 0 0 0 0 1 1 0 0 0 0 0
0 1 1 1 1 1 1 1 1 0 0 0
0 1 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0

```

Process returned 0 (0x0) execution time : 0.259 s  
Press any key to continue.

Matrice binaire de la lettre "I" après execution de la fonction "binarisation\_image"



"C:\Users\Eben\Desktop\UPMC\L3\3E103\_Infos\Projet\Polynome\_legendre.exe"

```

Choisissez un ordre x: 2
Choisissez votre n: 3
P<3> = 17.000, a l'ordre 2

```

Process returned 0 (0x0) execution time : 10.947 s  
Press any key to continue.

Polynôme de Legendre: (exemple de Calcul)  $P_3$  pour  $X=2$

```
C:\Users\Eben\Desktop\Proj_Binarisation\Proj_Binarisation\test\bin\Debug\test.exe
1.00 -0.27 -0.39 0.36 0.12 -0.35
1.00 -0.09 -0.49 0.13 0.34
1.00 0.09 -0.49 -0.13
1.00 0.27 -0.39
1.00 0.45
1.00

0.00 0.00 1.00 1.00 2.00 2.00 3.00 3.00 4.00 4.00
0.00 1.40 0.91 0.20 0.02 0.77 1.45 1.35 0.40
1.00 1.36 0.17 0.40 1.87 1.20 -0.21 0.54
1.00 1.30 0.02 2.10 1.22 -0.28 1.70
2.00 0.70 0.96 2.40 -0.40 1.90
2.00 0.21 2.39 0.95 0.81
3.00 -0.12 3.66 -0.61
3.00 0.00 3.50
4.00 0.78
4.00

Moment total = 75.029 et elements total Matrice = 180
Process returned 0 (0x0)   execution time : 0.035 s
Press any key to continue.
```

Moments de Legendre, pour une matrice binaire composée que de "1"