Phase de développement :

Nous avons cherché dans la phase de développement a suivre les étapes qu’on a décrit dans les schémas bloc de l’étude préliminaire. Pour cela, on a procédé étape par étape, pour etre sur de notre code et comprendre l’ensemble des fonctions, point à point.

Après avoir chargé une image dans scilab grâce a notre fonction, on a pu travailler sur la matrice. Cette fonction permet de convertir l’image en matrice. Il y a également une option qui permet de convertir une image couleur en image en nuances de gris. C’est ce que nous avons utilsé : Nous n’avons travailler qu’avec des images en nuances de gris.

|  |
| --- |
| function **matriceImage**=chargerImage(**path**, **isRGB**)  if **isRGB** == 0 then  **matriceImage** = double(imread(**path**));  else  **matriceImage** = double(rgb2gray(imread(**path**)));  end  endfunction |

Une fois notre image sous forme matricielle on a pu chercher a appliquer notre première étape, le passage d’une image en image fréquentielle. Pour cela, on réalise dans un premier temps une transformée de fourier.

On utilise la fonction de scilab fft. Cette fonction permet de récupérer une matrice de nombres complexes. Ces nombres contiennent donc les informations de modules et phases de notre image. Dans notre cas, on va chercher a appliquer notre tatouage a notre module, et laisser notre phase inchangée, pour pouvoir reconstituer l’image à la fin. Pour cela, nous avons créé une fonction qui permet de calculer la phase et le module :

|  |
| --- |
| ImageFreq = fft(image);  [module,phase] = complexe(ImageFreq);  function [**mod**, **phase**]=complexe(**matrice**)  [N,M] = size(**matrice**)  for x = 1:N  for y = 1:M  **mod**(x,y) = abs(**matrice**(x,y))  **phase**(x,y) = atan(imag(**matrice**(x,y)),real(**matrice**(x,y)))  end  end    endfunction |

Maintenant que nous avons le module de notre image, nous pouvons créer un tatouage pour chercher a l’appliquer. Nous avons choisi un tatouage très basique, qui nous permet de tester nos différentes fonctions. Ce tatouage comporte une symétrie centrale :

|  |
| --- |
| tatouage = [ 1,1,1,1,1,1,1;  1,0,1,1,1,0,1;  1,1,1,1,1,1,1;  0,1,1,1,1,1,0;  1,1,1,1,1,1,1;  1,0,1,1,1,0,1;  1,1,1,1,1,1,1  ] |

La première chose à faire est de dimensionner notre tatouage à la taille de notre image. Le rendu n’est pas forcement très esthétique, il s’agit de 6 gros rectangles noirs. L’objectif qu’on s’est fixé est de mettre en place nos fonctions, sans regarder le côté « tatouage utile ». On utilise pour cela la fonction de scilab imresize. On lui donne en paramètres notre tatouage, un coefficient alpha et comment on veut l’agrandir. Dans notre cas, alpha correspond au ratio entre le nombre de lignes du tatouage et le nombre de lignes de notre image. Vu qu’on travaille avec des images carrées, le ratio est le même pour les colonnes. On choisit un agrandissement « area » pour garder les proportions.

|  |
| --- |
| [N,M] = size(image);  [R,C] = size(tatouage);  alpha = N/R;  img\_tatouage = imresize(tatouage,alpha,'area'); |

Maintenant que notre image et notre tatouage font la même taille, nous pouvons appliquer un fftshift a notre module et le multiplier point à point par notre tatouage. Il faut ensuite appliquer un second fftshift, pour recentrer correctement les valeurs.

|  |
| --- |
| image\_marquee = fftshift(module);  image\_marquee = image\_marquee .\* img\_tatouage;  image\_marquee = fftshift(image\_marquee); |

La dernière étape consiste à reformer notre image à partir du « module tatoué » et de la phase de l’image initiale.

On applique pour cela la formule de trigonométrie qui lie le module et la phase grâce a une exponentielle.

Il ne reste ensuite qu’a faire une transformée inverse de Fourier pour reconstituer notre image pour pouvoir la visualiser.

|  |
| --- |
| cplx = image\_marquee .\* exp(%i .\* phase);  image\_marquee = fft(cplx,1); |

Tout au long du code, il est possible de visualiser « l’image fréquentielle » pour vérifier que les différentes étapes sont correctes. On peut aussi grâce à ça vérifier que le tatouage est bien présent dans notre image. On applique la fonction calcule module pour recalculer le module. On utilise également la fonction « afficheImage » pour afficher les images de façon classique lorsqu’elles ne sont pas en fréquence

|  |
| --- |
| function **mod**=calculmodule(**matrice**)  [N,M] = size(**matrice**)  for i=1:N  for j=1:M  **mod**(i,j) = log(**matrice**(i,j) +1)  end  end  endfunction  function afficherImage(**matriceImage**)  imshow(uint8(**matriceImage**));  endfunction |

La dernière partie que l’on a mis en place consiste à calculer le PSNR, comme expliqué dans la partie précédente, pour connaitre la robustesse de notre tatouage :

|  |
| --- |
| function **res**=PSNR(**image**, **image\_res**)  **res** = 10\*log10( (255^2)/(mean((**image\_res**-**image**).^2)) );  endfunction |

Vous trouverez dans la partie suivante l’ensemble de nos résultats obtenus et tests effectués.

CONCLUSION

Ce projet, qui consiste à tatouer une image, nous a permis d’approfondir nos connaissances au niveau du langage Scilab et nous a également permis de mieux comprendre le fonctionnement du traitement d’une image de façon générale.

On a apprécié la mise en application d’un tatouage sur une image. Cette application nous montre bien que même avec un tatouage qui semble très visible a l’œil, il peut passer inaperçu après des traitements. Cela nous a permis de bien comprendre les enjeux des water mark, qui permettent d’identifier le propriétaire d’une image. Cela pose aussi des questions de transmissions d’informations de façon discrète, comme par des procédés de stéganographie.

On a également pu se rendre compte du temps de traitement que ce procédé demande. Aussi, pour marquer des images a grande échelle, ou pour marquer un grand nombre d’images, il faudrait des ordinateurs bien plus performants.

Ce projet était très intéressant dans le cadre de nos études, mais aussi de nos futurs emplois.