

# **Création de mosaïque de photo «Sous contrainte»**

**Et utilisation du logiciel**



# Sommaire

## Introduction

A quoi sert ce TER ?

## Présentation du logiciel **Imagemagick**

Qu'est ce que c'est ?

Comment ça marche ?

Fonctionnalité de Base ?

## Présentation du script

Quelques notions de programmation.

Présentation du script

## Méthode de travail

Quels ont étaient les difficultés rencontrées ?

Pour moi (programmation, expliquer les raisons)

Pour Vincent (Rapport, expliquer les raisons)

Motivation ?

Pourquoi avoir pris ce sujet plutôt qu'un autre.

## Bilan

Points positifs ?

Points négatifs ?

## Conclusion

## Annexes

Programmes

Résultat d'une manipulation

## A quoi sert ce TER ?

*Sujet :*

Ce TER est un prolongement de l'enseignement Unix/Linux (version 2.4). Il doit nous permettre de construire un programme permettant, à partir d'une photo et d'une base de données d'images contenant un nombre conséquent de photos, de construire une mosaïque, c'est-à-dire un assemblage de photos (issues de la base de donnée) reconstituant la photo originale.

Ce sujet va nous permettre de mettre en pratique les quelques notions d'informatiques que nous avons pu voir en cours. Il va également nous donner l'occasion de s'initier à un logiciel d'imagerie : Imagemagick.

Nous présenterons dans un premier temps ce logiciel. Puis nous examinerons les différentes étapes du script dans lequel nous avons utilisés les commandes d'Imagemagick tel que convert, ... Nous expliquerons ensuite les difficultés que Vincent et moi avons rencontrés ainsi que les motivations qui nous ont incités à prendre ce sujet de TER.

Bonne lecture...

# Présentation du logiciel **Imagemagick**

*Qu'est ce que c'est ?*

Imagemagick est un logiciel de traitement d'images. C'est un ensemble de programme qui permet de créer, de modifier, d'afficher ou de convertir des fichiers dans différent formats.

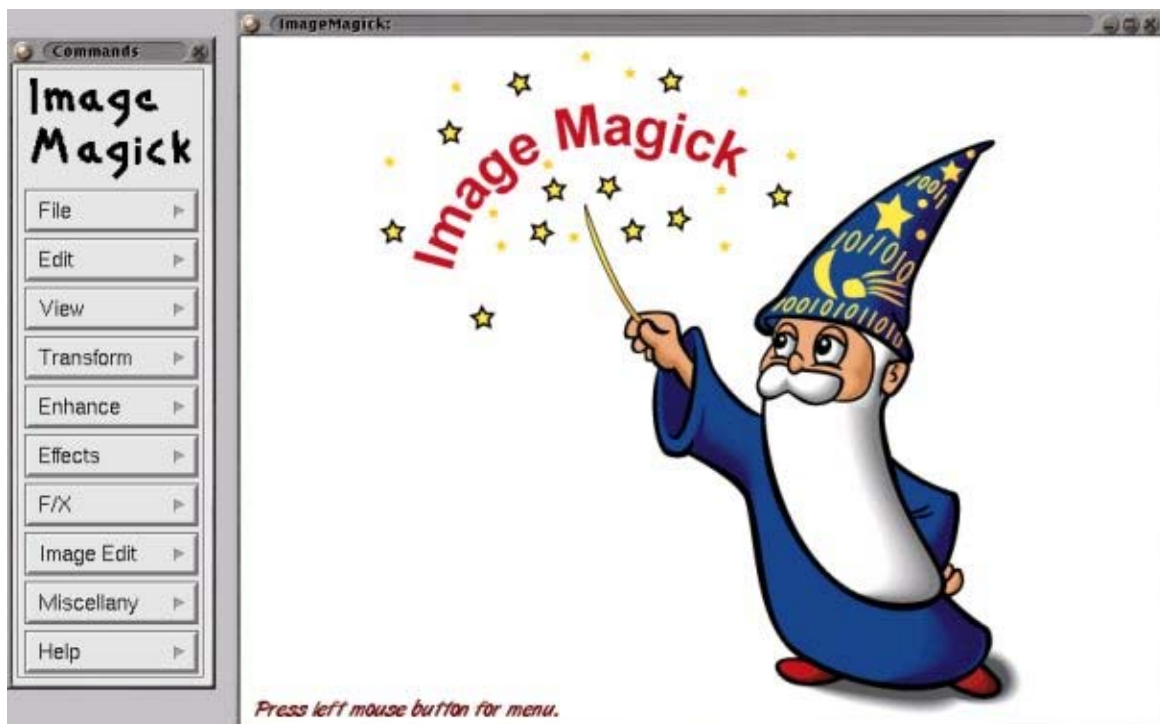
Mais Imagemagick, c'est avant tout un logiciel libre, performant, utilisable sur un grand nombre de plateforme et accessible par n'importe qui.

On peut télécharger gratuitement ce logiciel sur le site officiel :

[www.imagemagick.org](http://www.imagemagick.org)

*Comment ça Marche ?*

La plupart des fonctionnalités de ce logiciel peuvent-être utilisé en ligne de commande (ce que nous faisons dans ce TER). Toutefois, Imagemagick détient une interface graphique facilement utilisable et de très bonne qualité.



## *Que peut-on faire avec Imagemagick ?*

On peut pratiquement tout faire avec ce logiciel. Voici d'ailleurs quelques exemples des possibilités qui nous sont accessibles via Imagemagick :

- Convertir une image d'un format dans un autre (JPEG,...)
- Modifier la couleur
- Rajouter des effets spéciaux
- Créés des animations d'images
- Composer une image par plusieurs autres
- Rajouter du texte
- Décorer une image
- Connaitre les caractéristiques de l'image
- ...

De nombreux exemples sont accessibles à partir du site officiel ainsi qu'à l'adresse suivante :

[www.cit.gu.edu.au/~anthony/graphics/imagick6/](http://www.cit.gu.edu.au/~anthony/graphics/imagick6/)

## *Comment peut-on faire tout cela ?*

Comme nous le disions plus haut, Imagemagick est un ensemble de programmes. Cet ensemble comprend les programmes suivants :



identify

Identification et description du format d'un ou plusieurs fichiers image



convert

Conversion d'un fichier image d'un format dans un autre



display

## Affichage, manipulation et traitements des fichiers image



animate

Affichage d'une séquence d'image



montage

Création d'un ensemble d'images, composé à partir de plusieurs autres images



combine

Création d'une image par combinaison de plusieurs autres images



segment

Segmentation d'une image



...

De plus, chaque commande comprend une multitude d'options.  
Par la suite, nous nous intéresserons aux commandes que nous utiliserons dans le script.

# Présentation du script

*Quelques notions de programmation ...*

## **Les variables :**

Elles associent un nom à une valeur qui peut éventuellement varier au cours du temps.

## **Les structures de contrôle :**

Ils en existent plusieurs, utilisés pour différentes actions.

### *La structure if :*

Dans les scripts, il est parfois utile de faire des tests pour n'exécuter certaines parties uniquement lorsque les conditions nécessaires sont présentes.

Cette structure est de la forme :

```
If instructions
Then
# Instructions 1
Else
# Instructions 2
Fi
```

On peut également comparer une variable à plusieurs valeurs en utilisant des if imbriqués

### *La structure case :*

Pour comparer des variables à plusieurs valeurs, on peut également utiliser la structure case.

```
Case ${variable} in
Valeur1)
# Instructions 1
Valeur2)
#Instructions 2
*)
```

*#Traitement des autres valeurs*

::

Esac

*La structure while :*

Si on a besoin de faire un même traitement plusieurs fois de suite. On utilise cette structure de contrôle appelé boucle.

*While instructions*

Do

*# Traitement à répéter*

Done

*La structure until :*

Similaire à celle au-dessus.

*Until instructions*

Do

*# Traitement à répéter*

Done

*La structure for :*

Dans cette boucle, à la différence des précédentes, on connaît le nombre d'itération.

*For variable in liste-de-valeur*

Do

*# Traitement à répéter*

Done

**Les commandes :**

*La commande echo :*

Permet de rediriger ses paramètres sur la sortie standard.

*La commande # :*

Pratique pour les commentaires.



La commande > et >> :

C'est une commande de redirection qui permet de créer et d'ajouter des données dans un fichier.

Ce petit aperçu devrait permettre de répondre à quelques questions dans la compréhension du script.

## *Etude du script*

On peut identifier cinq étapes dans la création du script :

### *1ère étape : redimensionnement de l'image\_mosaïque*

Elle est constitué d'une **structure de contrôle if**. Le but de cette étape est de redimensionner l'image qui va nous servir de fond pour la mosaïque. On la redimensionne en 1024x768 en utilisant la commande **convert –crop**

### *2ème étape : la fonction compare*

Cette étape sert à la création de la fonction compare. Cette fonction permet de comparer deux images à partir de leurs moyennes de couleur RVB (rouge-vert-bleue) et de choisir l'image la plus appropriée pour composer la mosaïque. Composée d'une **structure de contrôle for et if**. Elle utilise également divers variables (Mean, image1, ...).

### *3ème étape : création d'un répertoire*

L'objectif de cette étape est de créer un répertoire (commande **mkdir**) dans lequel on va stocker les segments de l'image (de la 1ère étape). On se sert de **deux structures de contrôle if**. La première sert à vérifier que le répertoire n'existe pas (afin de ne pas en créer un deuxième). La seconde crée le répertoire. Deux variables sont utilisées : testdir et md.

### *4ème étape : découpage de l'image\_mosaïque*

Cette étape permet de découper l'image\_mosaïque en segment de même taille (32x32) et de les placer dans le répertoire créé dans l'étape 3. Pour commencer, on utilise **deux variables**, sw et sh, qui varient respectivement entre 0 et 992 et entre 0 et 736 et qui sont initialisées à 0. Elles s'incrémentent à chaque boucle de +32. Ces variables vont jouer les rôles de curseur sur l'image\_mosaïque. Puis on utilise **deux boucle while imbriquée** l'une dans l'autre. La première étant conditionnée par la variable sw et la seconde par la variable sh. Les commandes **convert –crop** (qui sert à découper les images) et **identify –verbose** (qui permet d'avoir des informations sur l'image découpé) sont également utilisées.

Petit + : le pourcentage de progression qui permet de mesurer la progression du travail.

#### *5ème étape : recomposition de l'image\_mosaïque avec la base de données*

Comme le titre l'indique, l'objectif de cette étape est de reconstituer l'image\_mosaïque avec d'autres images issues de la base de données afin d'obtenir la mosaïque finale. Pour cela, on réutilise **les deux variables précédentes** (sw et sh) auxquelles s'ajoutent **la variable paths** initialisée à « » et qui contiendra le répertoire créé dans l'étape 3. Comme précédemment, on utilise **deux boucles while** conditionnée de la même manière. Enfin, on utilise la fonction compare (voir étape 2). Les commandes propres à Imagemagick tels que **convert –geometry - composite** ou encore **identify –verbose** sont également utilisées.

Petit + : toutes les commandes qui permet de recomposer l'image\_mosaïque sont redirigé vers un fichier nommé script\_mosaic (voir en annexe). On retrouve également la barre de progression du travail.

## Méthode de travail

Les méthodes de travail de Vincent et de moi sont à peu près les mêmes : Réflexion sur le sujet, recherche d'information à propos du logiciel Imagemagick, recherche d'un algorithme, programmation, test, ...

La différence réside principalement dans la conception de l'algorithme. Vincent programme son script de façon à utiliser la moyenne des trois couleurs (rouge, vert, bleue). Quand à moi, je différencie les trois couleurs et j'étudie leurs moyennes séparément. Bien entendu, le programme devient plus long et surtout plus lent puisque certaines boucles doivent être faites trois fois et l'analyse des images se fait en se basant sur la moyenne de ces trois couleurs.

Enfin, Vincent a également une approche plus professionnelle au niveau de sa programmation puisqu'il insère un programme simple qui permet à l'utilisateur de savoir à quel niveau d'avancement se trouve le programme.

## Quels ont étaient les difficultés rencontrées lors de ce TER ?

*Gaëtan :*

« La difficulté majeur à était la programmation. Comme je l'ai dis dans la partie *Bilan*, j'ai eu l'avantage de commencer seul la programmation. J'ai ainsi commencé à écrire un script en Shell. Pour commencer, je me suis heurté au 'difficultés' du langage. A cela s'ajoute la difficulté d'utiliser un programme quasiment inconnu en ligne de commande. Malgré cela, j'ai réussi à convertir mon image de départ au format qui m'intéressait. J'ai également réussi à segmenter cette dernière et à mettre ces segments dans un répertoire créé précédemment. Finalement, j'ai pu, grâce à la commande `identify -verbose`, trouver les informations qui m'intéressaient et les filtrer avec la commande `Sed`. J'ai également écrit des lignes de commandes qui, en théorie, devaient me choisir une image optimale par rapport à des images se trouvant dans une base de données. Le problème est que cela ne fonctionnait pas. Je n'ai également pas réussi à utiliser certaine commande en langage Shell tel que la valeur absolue (voir le script en annexe). Enfin, bien que je bloquais sur le choix optimal de l'image, j'ai tout de même réussi à trouver la commande pour recomposer l'image\_mosaïque.

Un second problème à était celui du manque de temps, bien que le script, je m'en rends compte, ne soit pas d'une difficulté hors du commun. »

*Vincent :*

« Les difficultés majeures ont étés la découpe de l'image original et la comparaison avec les autres images.

Avant la découpe de l'image originale, il a fallu d'abord mesurer l'image. Ensuite, sa taille n'étant pas celle que nous attendions, il a fallu la remettre à la taille voulu. Pour cela, il faut former l'image avec une commande `Imagemagick`. Apres on découpe l'image original a 32x32 avec deux boucle sous Shell.

La deuxième difficulté a été de comparer les images issues de la segmentation de l'image-mosaïque avec la base de données des images.

Pour arriver à ce résultat, j'ai eu besoin d'obtenir des informations détaillées à propos de l'image. Ensuite, j'ai construit une fonction avec des commandes issues d'Imagemagick pour comparer des informations et choisir une image similaire. »

## Motivation

*Gaëtan :*

« Sur l'ensemble des TER, j'ai hésité sur deux sujets, tous deux en informatique. Le premier étant celui-ci, le second étant un TER sur la Cryptographie : 'la factorisation d'un entier par la méthode du crible quadratique'. Ce dernier me permettait d'utiliser un logiciel de programmation Scilab que je maîtrise assez bien. De plus la cryptographie est une application qui m'intéresserait, non pas dans ma vie professionnelle, mais en tant qu'utilisateur. Finalement mon choix s'est arrêté sur le sujet 'la création d'une mosaïque' pour plusieurs raisons. Premièrement, étant donné que nous avons une matière Unix/Linux ce semestre, j'ai pensé que ce TER aurait l'avantage de me faire progresser plus rapidement. Deuxièmement, puisque la programmation m'intéresse, et en particulier sur toutes les plateformes et dans tous les langages, je désirai accumuler un maximum d'expérience sous Unix. J'aimerais, à la fin de cette formation, maîtriser divers langages et programmes tels que SAS, Shell, Scilab, C, PHP et en particulier SQL et Java. Etant déjà à l'aise sur certains (tel que Scilab) et plus réticent sur Unix/Linux, voilà pourquoi j'ai choisi ce TER. Troisième et dernier point, l'intérêt de ce TER était d'avoir un aspect pratique (ce que je recherche dans cette formation professionnelle) et pas seulement théorique. Ce sujet nous permet de construire quelque chose que l'on peut voir... »

*Vincent :*

« Ce sujet à été très intéressant pour moi. Il requiert au programmeur de bonne connaissance des commandes d'Unix/Linux et du langage informatique. Ce cours étant très important ce semestre, ainsi que l'année prochaine, ce sujet m'a permis de pouvoir utiliser des logiciels sur Unix/Linux et également d'avoir plus d'information à propos de Linux.

## Bilan

Gaëtan :

« Pour commencer, Vincent et moi n'avons pas été immédiatement en groupe. On a chacun travaillé de notre côté pendant un mois. Cela m'a permis de travailler le programme en solo et de créer mon propre script (**voir en annexe : second script**). Bien que j'ai échoué sur la création du programme, j'en ai retiré divers points positifs :

Pour commencer, j'ai fait mes propres recherches sur l'utilisation du logiciel *Imagemagick*. Ensuite, j'ai également installé une version de linux (Suze) sur mon pc, ce qui m'a permis de me familiariser avec cette nouvelle interface. Finalement, j'ai créé un algorithme et j'ai essayé de le programmer en Shell ce qui m'a permis de m'améliorer dans l'utilisation du Shell ainsi que l'utilisation des scripts.

Si, avec Vincent, nous avons dès le départ créé un groupe, étant donné que Vincent à un niveau bien supérieur en programmation sous Shell (par sa formation précédente), il aurait créé son programme (**voir en annexe : premier script**) et je n'aurais pas eu les moyens de l'aider. L'avantage est que j'ai pu comparer son programme au mien. J'ai également pu lui apporter beaucoup plus d'aide lors de la formation de notre groupe.

Le seul regret que j'ai ne se porte pas sur le TER mais plutôt sur la manière dont il a été mis en place. Je trouve qu'il est dommage que les sujets de TER aient été donnés en février et non pas en Octobre. Cela permettrait aux étudiants de le commencer plus tôt. Enfin, je regrette qu'il n'y ait pas eu de TER en rapport avec les entreprises. »



Vincent :

« Au début du TER, j'ai analysé le sujet et écrit un cahier de charge dans lequel j'ai noté toutes mes idées afin de choisir la meilleur façon de réaliser mon projet. Ensuite, j'ai recherché des informations a propos du logiciel Imagemagick version 6.0 et je l'ai étudié durant quelques semaines. Par la suite, j'ai crée un algorithme et programmer sous linux Redhat avec le Shell durant un mois. Pendant ce temps, étant donné Gaëtan faisait le même sujet que moi, nous avons partagé quelques idées.

Lorsque le programme à était achevé, j'ai commencé à faire des tests sur mon ordinateur. Finalement, j'ai amélioré les algorithmes jusqu'à ce que tous les programmes fonctionnent bien et réalise l'objet désiré.

# ANNEXES

## *Premier script (Vincent)*

```
#!/bin/bash
#check image size/if input image is not 1024x768 then exit programme
imagesize="$( identify -format "%wx%h" $1)"
if [ $imagesize != "1024x768" ]; then
echo "please format image to 1024x768"
#info="$(identify $1)"
#echo $info#|sed -e "/^[1-9]../p"
#format image to 1024x768
convert -crop 1024x768+0+0 $1 $1
exit
fi

#fonc compare two images ,return similar filename /compare sw sh path
filename
function compare()
{
image1="$1/$2"
returnfile=image1
returnvalue=1000

#echo "compare"

for image2 in `ls images`
do
Mean=$(convert $image1 images/$image2 -compose difference -composite -fx
'(r+g+b)/3' miff:-|identify -verbose -|sed -n '/^.*Mean:
*/{s//scale=2//;s/(.*)//;s/$/*100\//32768*10//;p;q;}'|bc)
Mean=$(echo $Mean|sed -e 's/\..*//g')

if [ $Mean -lt $returnvalue ]; then
returnfile=$image2
returnvalue=$Mean
fi
done

echo $returnfile
return

}

#creator a directory
testdir="$(ls -d "Repertoire_$1")"
if [ -n $testdir ]; then
rm -fr "Repertoire_$1"
#rmdir "Repertoire_$1"
echo "Repertoire suprimé"
fi

md="$(mkdir "Repertoire_$1")"
if [ -z $md ]; then
echo "cree un repertoire : Repertoire_$1"
else
echo "error"
fi
```

```

#split a image
sw=0
sh=0
paths=""
while [ $sw -le 992 ]
do

while [ $sh -le 736 ]
do
paths="Repertoire_$1/$sw#$sh"
convert -crop 32x32+$sw+$sh $1 $paths
#identify -verbose "Repertoire_$1/$1_$sw#$sh">"Repertoire_$1/$sw#$sh.ide"

sh=`expr $sh + 32`
echo -n "."
done
sh=0
sw=`expr $sw + 32`

progresse="$(echo "($sw/1024)*100"|bc -l|sed -e 's/\^*0000*//g'|cut -c1-6)"
echo $progresse%"

done

echo ".....100% decoupe image success"


echo "en train analyse base de donne de image....."
echo "recompose l'image mosaïque....."
#convert -size 1024x768 xc:white mosaic_$1
touch script_mosaic
echo "convert -size 1024x768 xc:white ">script_mosaic
echo "create success"
sw=0
sh=0
paths=""
while [ $sw -le 992 ]
do

while [ $sh -le 736 ]
do
paths="Repertoire_$1"

#echo "$sw#$sh"
#compare $sw $sh $paths "$sw#$sh"
getfile="$(compare $paths "$sw#$sh")"
#echo "filenameget : $getfile"

echo "images/$getfile -geometry 32x32+$sw+$sh -composite ">>script_mosaic
#convert mosaic_$1 $getfile -geometry 32x32+$sw+$sh -composite mosaic_$1
#identify -verbose "Repertoire_$1/$1_$sw#$sh">"Repertoire_$1/$sw#$sh.ide"

sh=`expr $sh + 32`
echo -n "."
done
sh=0

```

```

sw=`expr $sw + 32`

progresse="$(echo "($sw/1024)*100"|bc -l|sed -e 's/\^*0000*//g'|cut -c1-6)"
echo $progresse%"

done
echo "mosaic_test.jpg">>script_mosaic
echo "compose images to new file"
scripts=$(cat script_mosaic)
`$scripts`
echo "finish"
#convert 08.jpg 06.jpg -compose difference -composite -fx '(r+g+b)/3'
miff:-|identify -#verbose -|sed -n '/^.*Mean:
*/{s//scale=2;/;s/(.*)//;s/$/*100\//32768;/p;q;}'|bc

```

## Second script (Gaëtan)

```

#!/Desktop/

#cd Ter\ affiche\ film7

#on cree un fond sur lequel collé les images selectionnées a la fin:
#convert -size 60x80 xc:skyblue composite.gif

#On initialise des variables contenant les moyennes des couleurs des images
de la base
TR=$(identify -verbose images\1\)\.jpg|sed -e '17,17!d' -e
'/[[[:digit:]]/!d' -e 's/^.*Mean:[:space:]]*// ' -e 's/[:space:]](.*$'// -e
's/\./')
echo ${TR}

#on pose une variable contenant le nom de l'image optimale
imop=$(identify -verbose images\1\)\.jpg|sed -e '1,1!d' -e 's/ JPEG.*$//')
echo ${imop}

#On converti l'image à découpé au format voulu puis on la découpe
convert -geometry 60x80 images\15\).jpg im_seg
convert -crop 30x40 im_seg im #les images de la base de données sont 30x40

#on identifie les moyennes de rouge, de vert et de bleu pour chaque image
for image in im-{0} #,1,2,3}
do
echo ${image}
rm=$(identify -verbose ${image}|sed -e '17,17!d' -e '/[[[:digit:]]/!d' -e
's/^.*Mean:[:space:]]*// ' -e 's/[:space:]](.*$'// -e 's/\./')
#gm=$(identify -verbose ${image}|sed -e '22,22!d' -e '/[[[:digit:]]/!d' -e
's/^.*Mean:[:space:]]*// ' -e 's/[:space:]](.*$'// -e 's/\./')
#bm=$(identify -verbose ${base}|sed -e '27,27!d' -e '/[[[:digit:]]/!d' -e
's/^.*Mean:[:space:]]*// ' -e 's/[:space:]](.*$'// -e 's/\./')
echo la moyenne de rouge est ${rm} #, de vert est ${gm} et celle de bleu
est ${bm}

for base in *.jpg

```

```

do
    echo ${base}
    RM=$(identify -verbose ${base}|sed -e '17,17!d' -e '/[[[:digit:]]/!d'
-e 's/^. *Mean:[[:space:]]*///' -e 's/[[:space:]](. *$'// -e 's/\././')
    #GM=$(identify -verbose ${base}|sed -e '22,22!d' -e '/[[[:digit:]]/!d'
-e 's/^. *Mean:[[:space:]]*///' -e 's/[[:space:]](. *$'// -e 's/\././')
    #BM=$(identify -verbose ${base}|sed -e '27,27!d' -e '/[[[:digit:]]/!d'
-e 's/^. *Mean:[[:space:]]*///' -e 's/[[:space:]](. *$'// -e 's/\././')
    echo la moyenne de rouge est ${RM} #, de vert est ${GM} et celle de
bleu est ${BM}

```

# On rend toutes les différences entres les moyennes positives (on s'occupe  
# ici seulement de la moyenne de rouge)

```

    A=$(( ${RM} - ${rm} ))
    if [ ${A} < 0 ]
    then A= $((-1*${A}))
    fi
    echo ${A}

    B=$(( ${TR} - ${rm} ))
    if [ ${B} >= 0 ]
    then echo ${B}
    else B= $((-1*${B})) && echo ${B}
    fi

    if [ ${A} <= ${B} ]
    then TR=${RM} && imop=
    else TR=${TR}
    fi

```

# Comment attribuer la bonne image, imop, à la moyenne optimale de rouge  
#correspondant.

#on place l'image optimale ( découverte à ce stade de la programmation)  
#pour reactualiser à chaque boucle la position, on doit poser deux  
#variables qui feront curseur

```

    lon=0
    lar=0
    While [ $lon -le 80 ]
    Do
        While [ $lar -le 60 ]
        Do

            #composite -geometry 40x30+$lon+$lar ${imop} -composite
            $lar = '$lar + 30'

        Done

        $lar = 0
        $lon = '$lon + 40'

    Done

```

Done

done

## *Resultat du premier script*

