Solène Mary-Vallée
Baptiste Ory

Programme de traitement d'images miniGimp

/Projet C IMAC 1

Programmation et algorithmique
/Vincent Nozick

/Sommaire

/Introduction

/Présentation du projet

Le projet qui nous a été confié pour ce cours de « Programmation et algorithmique » est la réalisation d'un **programme de traitement d'images en langage C**.

Ce programme devait pouvoir modifier des images avec différents **effets choisis par l'utilisateur** et afficher des **histogrammes** (représentation de la distribution des couleurs de l'image), un peu à la manière de ce que peut faire un logiciel de retouche photo avec une interface comme *Adobe Photoshop* ou *Gimp*.

Il nous a été proposé d'utiliser le **format d'image PPM** (*Portable pixmap*) pour les images : un format binaire en couleur. De plus, les effets devaient être appliqués à l'image avec des **LUTs** (*look-up tables*) : des tableaux de correspondance de couleur qui permettent de modifier indirectement les images.

/Fonctionnalités générales

Voici ce que nous sommes finalement arrivés à implémenter pour notre programme (fonctionnalités demandées dans la consigne ou non) :

Fonctionnalité	Demandée	Description
Manipulation des images	X	Chargement, création et sauvegarde des images
IHM	X	Interactions homme-machine : récupérer les options et arguments voulus par l'utilisateur
LUTs	X	Effets pour modifier l'image indirectement grâce à un tableau de correspondance de couleur
Histogrammes	X (dans des images en option)	Création d'histogrammes pour l'image d'arrivée et l'image de départ dans des images, pour voir la répartition des couleurs
Historique		Création ou mise à jour d'un fichier texte pour garder une trace des effets appliqués aux images
Changements sur l'image		Effets qui, à l'inverse des LUTs, modifient directement l'image (dont des matrices de convolution)

Les LUTs et changements sur l'image nous permettent de proposer 13 effets différents (avec paramètres ou non) :

Effets	Type	Description	Paramètre
ADDLUM	LUT	Augmente la luminosité	X
DIMLUM	LUT	Diminue la luminosité	X

ADDCON	LUT	Augmente le contraste	X
DIMCON	LUT	Diminue le contraste	X
INVERT	LUT	Inverse les couleurs	
SEPIA	Changement sur l'image	Effet sépia	X
BLACKANDWHITE	Changement sur l'image	Effet noir et blanc	
BLUR	Changement / matrice	Effet flou	
REINEDGES	Changement / matrice	Effet renforcement des bords	
DETECTEDGES	Changement / matrice	Effet détection des bords	
BUMP	Changement / matrice	Effet repoussage	
HIGHPASS	Changement / matrice	Passe-haut	
LOWPASS	Changement / matrice	Passe-bas	

/Mise en contexte

Le **programme exécutable** (depuis la racine du dossier du projet) est : bin/minigimp.

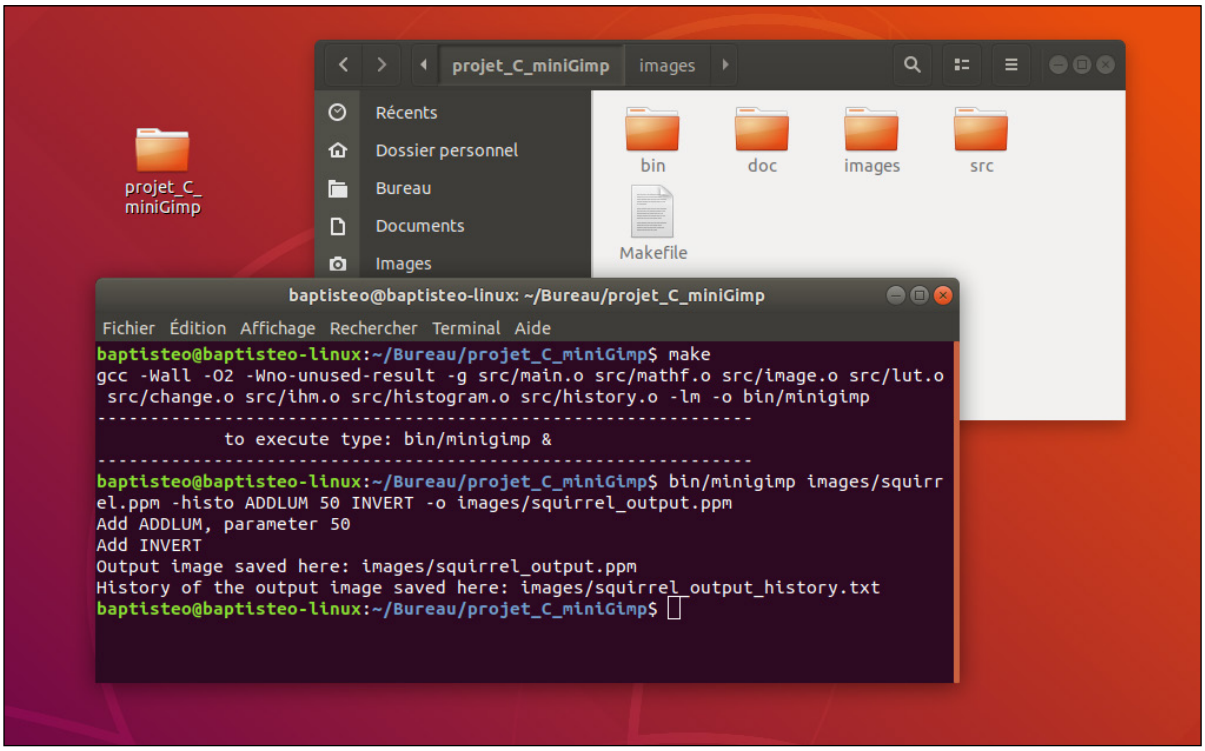
La **commande d'usage** pour se servir du programme et modifier une image est :

bin/minigimp image_input.ppm [-h] [-histo] [<code_effect>[_<param1>]*]* [-o image_output.ppm]

-h permet de générer des **histogrammes** pour l'image à éditer et l'image éditée
-histo permet de générer ou mettre à jour l'**historique** pour l'image éditée

Ainsi, un utilisateur qui veut modifier l'image « squirrel.ppm » (dans le répertoire « images ») en ajoutant de la luminosité (paramètre 50), en inversant les couleurs et en ne créant que l'historique tapera cette ligne de commande à partir du dossier du projet :

bin/minigimp images/squirrel.ppm -histo ADDLUM 50 INVERT -o images/squirrel_output.ppm



Mise en contexte du fonctionnement du programme sur Linux (Ubuntu)

/Fonctionnalités générales

/IHM

L'**IHM** (interactions homme-machine) **récupère les options et arguments voulus par l'utilisateur** pour pouvoir les utiliser dans le programme : image de départ, image d'arrivée, effets et paramètres, présence des histogrammes et présence de l'historique. On le trouve dans le fichier « **ihm.c** ».

La principale difficulté pour la création de cet IHM fut de prendre en compte **toutes les possibilités d'arguments entrés par l'utilisateur** (certains utilisateurs peuvent être très vicieux...) et de pouvoir ainsi s'adapter aux différentes situations pour **stopper le programme le moins souvent possible**, ou **orienter l'utilisateur sur la marche à suivre dans le cas contraire**.

Chaque argument entré est donc **testé puis stocké s'il correspond à une entité reconnue**. Ainsi, si par exemple l'utilisateur entre des effets qui ne sont pas reconnus, ceux-ci ne seront pas pris en compte. De plus, un **nom de sortie par défaut** est donné si celui-ci n'est pas renseigné par l'utilisateur. Tout cela permet d'arrêter le programme seulement si l'utilisateur renseigne une image PPM d'entrée inconnue.

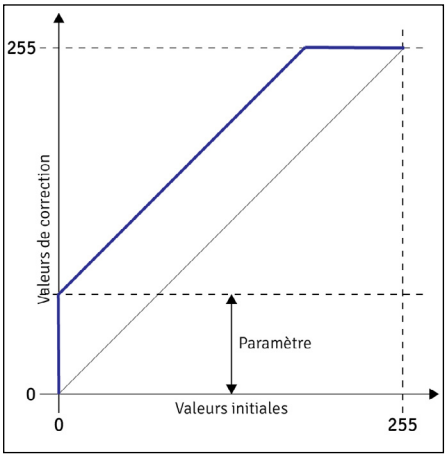
/LUT

Les **LUTs modifient les images indirectement grâce à un tableau de correspondance des couleurs.** On les trouve dans le fichier « **lut.c** ». Chaque valeur du codage des couleurs (de 0 à 255 : du plus foncé au plus clair) est modifiée dans le tableau LUT **selon l'effet LUT choisi**. Une fois que chaque LUT a été appliquée au tableau, les valeurs qui dépassent les nombres admis par le codage des couleurs sont **rognées** : les valeurs en dessous de 0 prennent la valeur 0 et les valeurs au-dessus de 255 prennent la valeur 255.

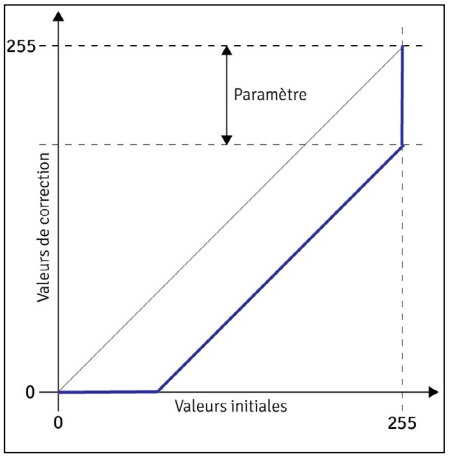
Enfin, chaque composante de chaque pixel de l'image à éditer (rouge, vert et bleu) est **convertie pour correspondre à sa nouvelle valeur dans le tableau LUT** : les valeurs de 160 vont passer à 170 si la 160e composante du tableau de LUT est maintenant 170.

Mire de référence : « mire.ppm »

ADDLUM et **DIMLUM** ont un algorithme très simple : ils **ajoutent ou enlèvent un paramètre donné par l'utilisateur à chaque valeur du tableau**. Cela forme ces courbes (après rognage) :

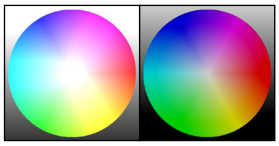


Courbe ADDLUM

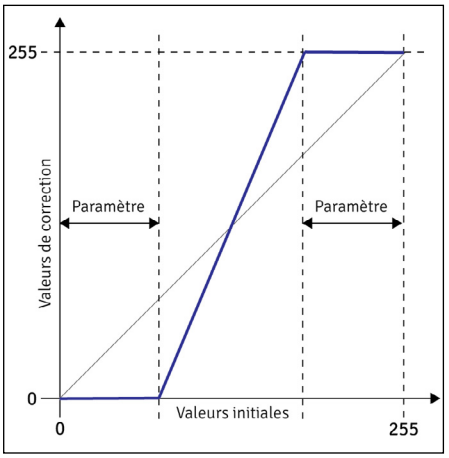


Courbe DIMLUM

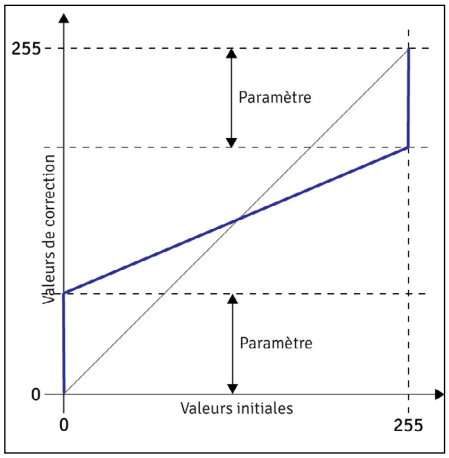
Rendus des effets



ADDCON et **DIMCON** ont un algorithme plus complexe : grâce à une **fonction affine, chaque valeur du tableau prend une valeur plus ou moins haute selon qu'elle est plus foncée ou plus claire**, avec la possibilité de rendre la **pente plus ou moins abrupte en faisant varier un paramètre**. Par contre, si le paramètre est supérieur à 128, il est **ramené à 128** pour ne pas inverser les courbes. Cela forme ces courbes (après rognage) :

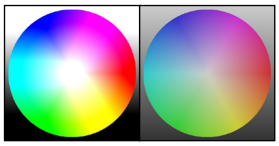


Courbe ADDCON



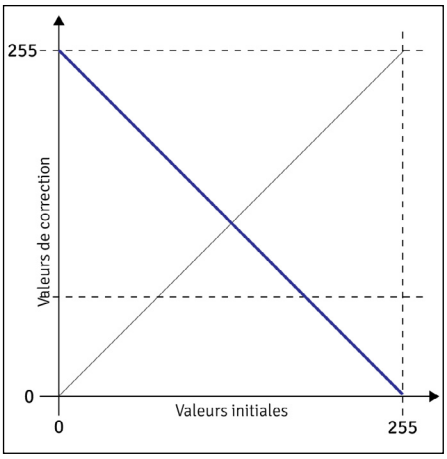
Courbe DIMCON

Rendus des effets



C'est cet algorithme qui nous a fait le plus réfléchir. Nous avons aussi eu de la difficulté à traduire la fonction affine en langage C. Le résultat final est satisfaisant, mais il y a malheureusement une **perte de données** (visible sur l'histogramme), car il peut y avoir plusieurs fois les mêmes valeurs dans le tableau LUT ou des valeurs qui n'apparaissent pas. Une solution serait peut-être d'arrondir les courbes.

INVERT a le fonctionnement le plus simple : **chaque valeur du tableau prendre sa valeur opposée en soustrayant chaque valeur à 255**. Celle forme cette courbe :



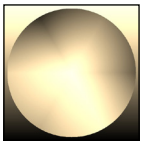
Courbe INVERT

Rendu de l'effet

L'effet **SEPIA** était demandé, mais nous ne sommes **pas parvenus à la faire avec le principe des LUTs**. Il est donc codé dans le programme avec les effets « changements sur l'image » (« **change.c** »). En effet, **chaque nouvelle valeur d'un pixel de couleur d'un triplet est déterminée en fonction des valeurs du triplet**. Ce n'est plus seulement à une valeur de départ qu'on applique une nouvelle valeur pour chaque couleur (rouge, vert ou bleu), mais à un triplet de valeurs. Nous avons trouvé les fonctions suivantes à appliquer à ce triplet (r, g, b) :

```
r = r*0.393+g*0.769+b*0.189+param
g = r*0.349+g*0.686+b*0.168+param
b = r*0.272+g*0.534+b*0.131
```

Rendu de l'effet



Les valeurs sont bornées entre 0 et 255 en appliquant successivement les fonctions min et max (codées dans le fichier « **mathf.c** ») à chaque résultat, de la même manière qu'on rognait le tableau de LUT. Nous avons ajouté le paramètre entré par l'utilisateur (param) afin de pouvoir rendre l'image plus ou moins jaune.

/Histogramme

L'**histogramme** est un graphique montrant le nombre de pixels pour chaque valeur (entre 0 et 255 donc) présente dans l'image. Sa création est codée dans le fichier « **histogram.c** ». Il y a deux étapes dans la création de nos histogrammes : **sonder l'image pixel par pixel en incrémentant un tableau**, puis **tracer l'histogramme dans une image à partir de ce tableau**.

La fonction d'**incrémentation du tableau** fut simple à coder (si l'on n'oublie pas d'initialiser toutes les cases du tableau à 0). On peut à partir de cette fonction décider de **remplir le tableau par rapport au rouge, vert ou bleu, ou à la luminance** (luminance par défaut).

La fonction de **création d'image histogramme à partir du tableau** nous a apporté plus de difficultés. Premièrement, il a fallu réfléchir à **comment construire l'image affichant l'histogramme**. Nous avons finalement créé l'image ligne par ligne de la façon suivante : si la nième valeur du tableau est supérieure ou égale au numéro de la ligne, la zone prévue pour le pic de cette valeur sur cette ligne est colorée en noir, sinon en blanc. Deuxièmement, **la taille de l'histogramme** posait problème. Au début, nous avions fixé la largeur à 256 pixels (un pixel pour chaque valeur possible) et la hauteur au maximum des valeurs du tableau (déterminé à l'aide d'une simple fonction « maxInArray » codée dans le fichier « mathf.c »). Cependant, cette méthode donnait un histogramme tout en longueur pas très lisible. Et, en augmentant simplement la largeur (qui reste un multiple de 256 pour des raisons pratiques), l'image devenait trop lourde et la hauteur fluctuait trop en fonction de l'image d'entrée. Nous avons donc choisi de **fixer la hauteur et la largeur de notre histogramme et de réguler toutes les valeurs dans le tableau par rapport à cette hauteur et au maximum des valeurs du tableau** (les maximums sont égaux à la hauteur choisie et que les autres valeurs sont proportionnelles).

Une amélioration de cette fonctionnalité serait de pouvoir indiquer l'**échelle sur l'image histogramme**.

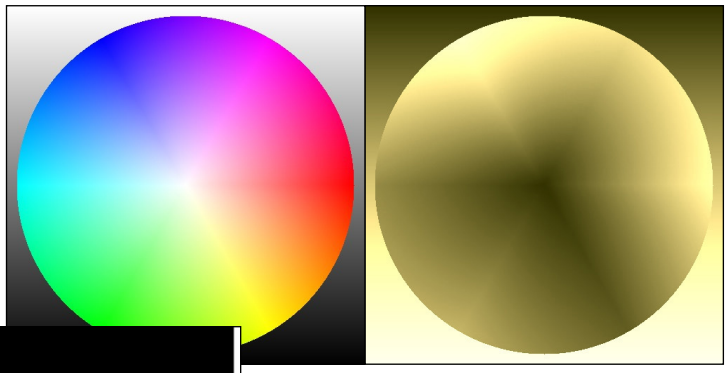
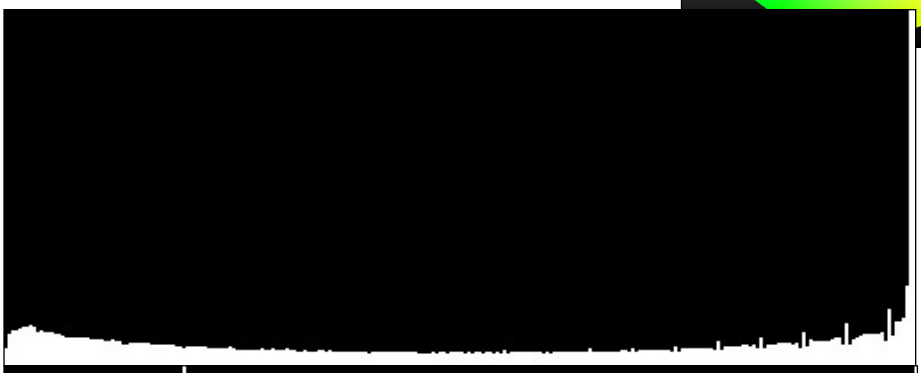
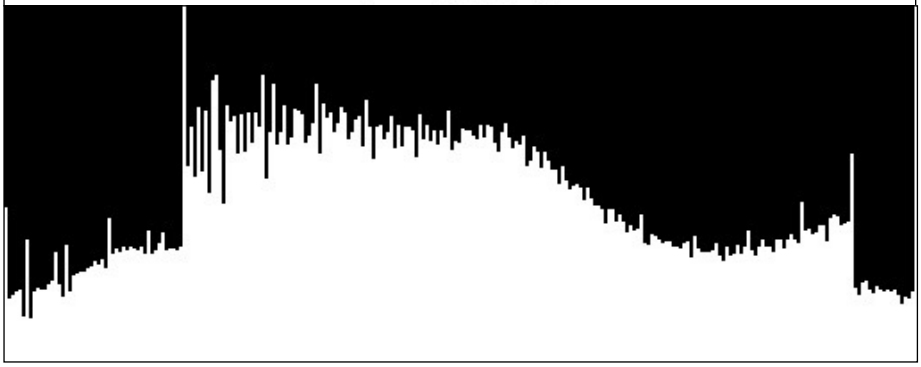


Image « mire.ppm » à éditer et éditée avec plusieurs effets



Histogramme de l'image à éditer



Histogramme de l'image éditée avec plusieurs effets

/Fonctionnalités supplémentaires

/Historique

L'**historique** permet de garder une trace des **effets appliqués aux images dans un fichier texte**, de la même manière qu'on écrit dans les images PPM.

Chaque modification (un effet appliqué) de l'image est **sauvegardée dans un tableau au moment de son application**. Puis, ce tableau de chaînes de caractères est **envoyé à la partie historique pour écriture dans un fichier texte**. Ce fichier texte est **nommé par rapport à l'image de sortie**.

La principale difficulté fut de prendre en compte le fait que l'image d'entrée puisse déjà avoir un historique d'effets appliqué. Dans ce cas, on récupère les lignes de l'historique et on les place en premier dans le nouvel historique.

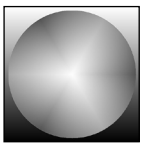
/Changements sur l'image

Les « **changements sur l'image** » modifient directement les pixels de l'image. Ils sont codés dans le fichier « **change.c** ».

BLACKANDWHITE est une fonction d'effet simple qui, pour passer l'image en noir et blanc, applique à chaque partie d'un triplet (r, g, b) du pixel la moyenne de ce triplet :

```
Pour i allant de 0 à largeur*hauteur*3, i avance de 3 en 3
    val = (r*g*b)/3
    (r, g, b) = (val, val, val)
```

Rendu de l'effet

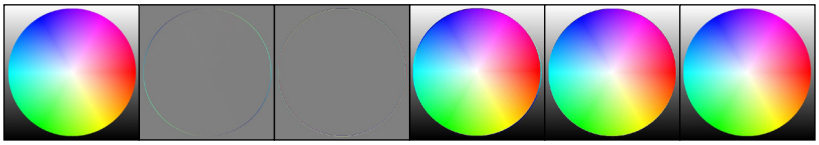


Nous avons ensuite décidé d'ajouter d'autres effets en utilisant des **matrices de convolution**. Le principe est le suivant : pour chaque pixel de l'image, on multiplie chacun de ses pixels voisins (de même couleur : même composante du triplet) et ce pixel par la valeur correspondante dans une matrice prédéfinie. Puis, on fait la somme des valeurs qui devient la nouvelle valeur de ce pixel.

La fonction ne fut pas si compliquée à coder en soi, mais il y eut quelques subtilités à prendre en considération. Tout d'abord, **on ne peut pas modifier directement l'image** sinon les calculs sont faussés pour les pixels suivants. Il faut donc en créer une nouvelle dans laquelle seront ajoutées au fur et à mesure les valeurs calculées. Ensuite, il y a des **cas particuliers pour les pixels situés sur les bords de l'image** (car ils n'ont pas autant de voisins) : comme il est plus facile de ne pas traiter ces cas-là, nous avons pris une marge d'un pixel (pour les pixels sur la ligne horizontale de l'image haute et basse). Et finalement, pour avoir un résultat cohérent, il a aussi fallu **diviser ou appliquer un décalage à la somme des valeurs, puis rogner les valeurs** qui ne sont pas admises dans le codage des couleurs.

Une fois cette fonction créée, il est possible d'appliquer différents filtres en changeant simplement la matrice envoyée à notre fonction. Nous proposons les effets **BLUR, REINEDGES, DETECTEDGES, BUMP, HIGHPASS** et **LOWPASS** dans notre programme grâce à ces matrices de convolution.

Rendus des effets (dans l'ordre énoncé ci-dessus)



/Visuels

/LUTs et changements sur l'image

Image originale

Image avec ADDLUM paramètre 50

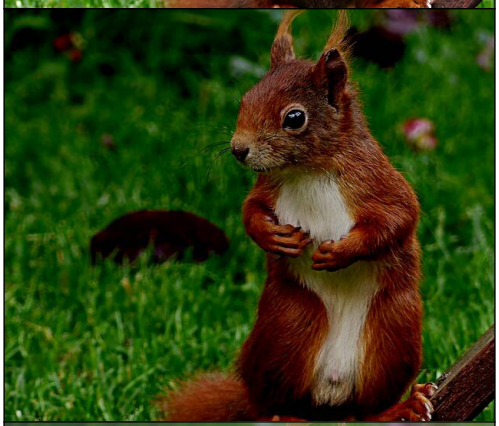


Image avec DIMLUM paramètre 50

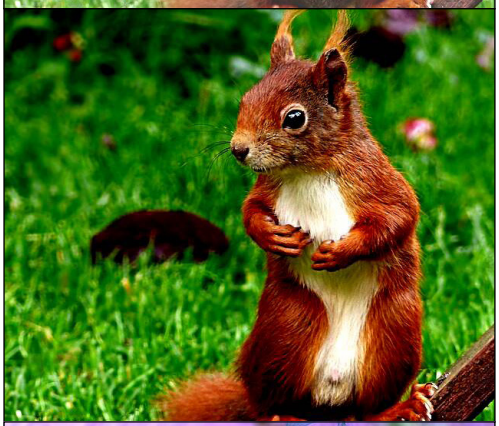


Image avec ADDCON paramètre 50

Image avec DIMCON paramètre 50

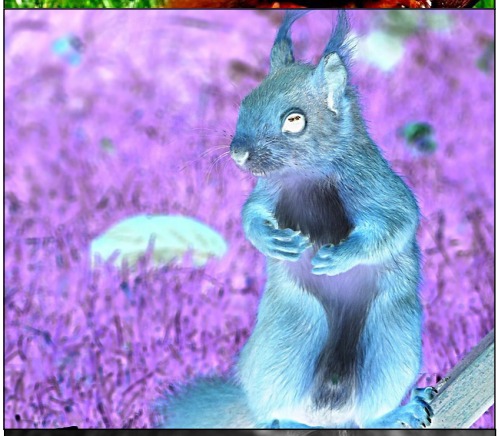


Image avec INVERT

Image avec SEPIA paramètre 50

Image avec BLACKAND WHITE

Image avec BLUR

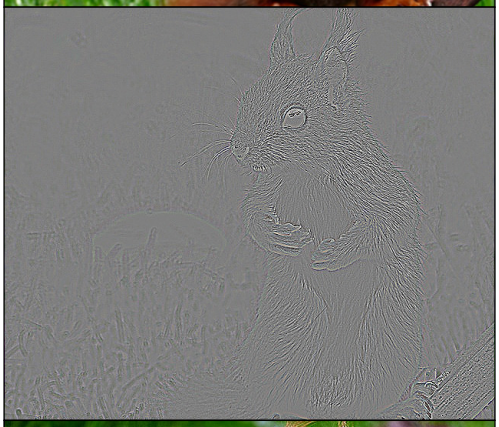


Image avec DETECTEDGES

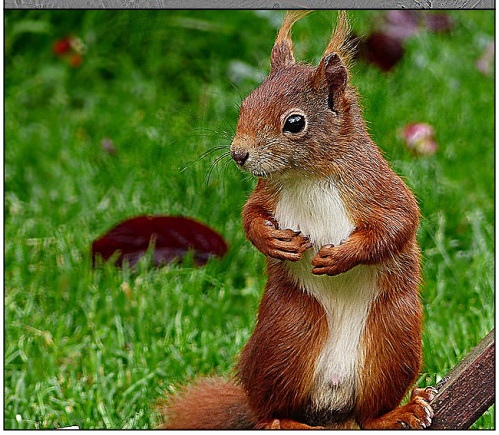


Image avec HIGHPASS

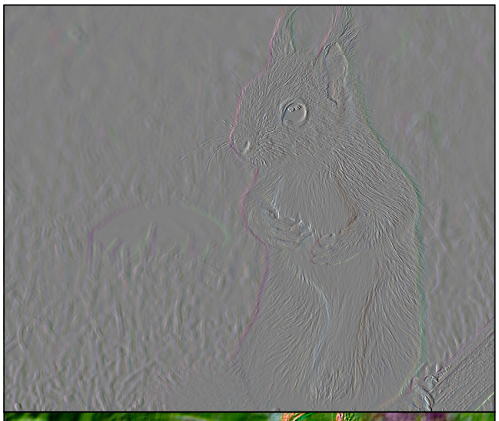
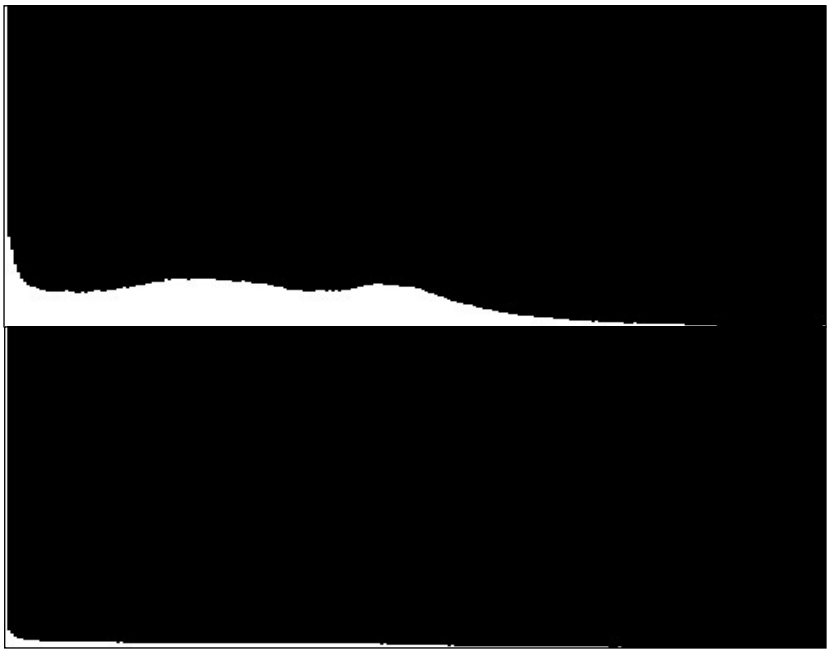


Image avec REINEDGES

Image avec BUMP

Image avec LOWPASS

/Histogrammes



Histogramme de l'image originale et histogramme de l'image éditée

Image éditée avec DIMLUM (paramètre 10) + ADDCON (paramètre 25) + BUMP + LOWPASS

/Sources

Équipe de documentation de GIMP. Matrice de convolution : GNU Image Manipulation Program [en ligne]. Disponible sur : https://docs.gimp.org/2.6/fr/plug-in-convmatrix.html

IDS. Traitement d'images avec les tables LUT (« lookup tables ») : IDS Imaging Development Systems GmbH [en ligne]. 2014. Disponible sur : https://fr.ids-imaging.com/tech-tipps-detail/fr_techtip-lookup-tables.html

LEGRAND, Frédéric. Filtrage d'une image par convolution : Frédéric Legrand, Informatique appliquée aux sciences physiques [en ligne]. 01/04/2014. Disponible sur : http://www.f-legrand.fr/scidoc/docimg/image/filtrage/convolution/convolution.html

MÜLLER, Didier. Chapitre 4, Traitement d'images : Nymphomath [en ligne]. 06/2018. Disponible sur : https://www.apprendre-en-ligne.net/info/images/images.pdf

MSB software. 4, Traitement d'images : Astroart 2.0 [en ligne]. 15/02/2002 (modif.). Disponible sur : http://xmcvs.free.fr/astroart/Chapitre4.pdf

NOZICK, Vincent. Histogrammes et Lookup-tables : Université Paris-Est Marne-la-Vallée [en ligne]. Disponible sur : http://igm.univ-mlv.fr/~vnozick/teaching/slides/M1_ti/03%20histogramme_LUT.pdf?fbclid=IwAR2-EpCmDnVEhb17B-pvtkm4l1r-7VYWanywLDDEFjO9Grg-ByjIc-p6bM0I

PERRET, Benjamin. Traitement d'histogramme : Benjamin Perret, Traitement et analyse d'images [en ligne]. 2017. Disponible sur : https://perso.esiee.fr/~perretb/I5FM/TAI/histogramme/index.html

PILLOU, Jean-François. Traitement d'images : Comment Ça Marche [en ligne]. 16/12/2016 (modif.). Disponible sur : https://www.commentcamarche.net/contents/1216-traitement-d-images

Université Rennes 2, LP CIAN. Histogrammes : Université Rennes 2 [en ligne]. Disponible sur : https://www.sites.univ-rennes2.fr/arts-spectacle/cian/image_numFlash/pdf/chap5_cours51.pdf

Lien github du projet : https://github.com/BaptisteOry/projet_C_miniGimp.git