Floating images processing (for fun and profit ?)

tTh

17 juillet 2023

Une image flottante ?

Back in a far past part of history.

Mais de quoi parle-t-on exactement ?

Traditionnellement, les valeurs des pixels dans les images informatiques sont mémorisées sur 8 bits, un octet, soit 256 valeurs différentes. Ceci dit, on trouve parfois des images codées sur 16 bits par composante, mais c'est loin d'être le cas général. J'ai donc souhaité aller plus loin, et coder chaque canal de chaque pixel en virgule flottante sur 32bits, le type `float` du langage C. Ce qui correspond à la norme IEEE 754-1985.

Attention ! tout le code que nous allons voir ensemble est en perpétuelle évolution [1], et sa fiablité (surtout sur certains aspects mathématiques) reste à démontrer. Mais le service après-vente est assez réactif. Du moins pour ceux qui suivent le canal `#tetalab` sur le réseau IRC de *libera.chat*.

Attention ! ce document commence par une bonne rafale de technique parfois *hardue* [2], avec des pointeurs dedans, mais vous êtes déja au courant. Le pointeur, c'est bien, c'est comme le doigt sur les écrans de zombiephones, ça sert juste à dire « C'est là ! », c'est donc gravement utile, mais parfois, le doigt glisse. Et là, tout peut arriver. Vous avez parfaitement le droit de sauter directement à la page 15 pour quelque chose de plus concret.

MiniFaq

En avril 2021, pendant la fin d'un monde, on m'a demandé de préciser certains points. Je vais tenter de répondre à ces FAQ :« Fréquentes et Absurdes Questions » avec des réponses absurdes et précises.

c'est quoi une image d'un point de vue physique ?

Une représentation approximative d'un monde physique invisible. L'infinie bio-complexité de notre système oculaire et ses interactions avec notre multivac neuronal n'est devenue quantifiable qu'avec l'arrivée des premiers calculateurs numériques quand certains ont songé à passer du flou mental à la grille physique.

Les détails techniques sur cette représentation sont en page 7, avec des pointeurs dedans. Ah, les pointeurs, la pire chose du monde, mais pourquoi s'en passer...

quelques rappels de comment on acquiert et numérise une image

Avant de se l'approprier, il vaut la rendre réelle dans sa vision intérieur. Il faut l'imaginer comme des lignes de code interposées entre la grille physique dont on vient de parler, et une

1. voir page 19 pour les détails
2. hard + ardue = private joke Usenet

autre grille, bien réelle, des pixels sur notre écran Trinitron de 44cm. Et puisque on en parle, je me permet de vous conseiller avec **vigueur** la webcam de Playstation (aka PsEye). Moins de 10 Euros dans les machin-trocs, et un grain d'image vraiment présent.

`https://fr.wikipedia.org/wiki/PlayStation_Eye`

définir les notions de base : canal, pixel, codage couleurs

Ah vous en voulez de la technique ? Oké, on y va. En fait, on va tripoter pleins de nombres, plus ou moins rangés dans des champs de pixels, pour concrétiser l'existence perceptuelle de ces grains de lumière chromatisés.

quelques belles images / schémas polychromes en intro pour illustrer tout ça

Tour cela est dans le grand Ternet mondial, je pourrais même vous donner l'url si vous me promettez de ne pas la glisser dans le capitalisme de surveillance.

Table des matières

1 Théorie **4**

2 Premier exemple **4**

3 Installation **5**

4 Utilisation coté codeur **6**

5 Les outils **15**

6 Debug **18**

7 TODO **19**

8 Exemples pour yusers **19**

9 Video for Linux **23**

10 À l'extérieur **24**

11 Le flou temporel **25**

12 Expérimentations **27**

13 Et pour la suite ? **27**

1 Théorie

Pour le moment, seule la quête de l'empirisme absolu a été visée. Les justifications mathématiques attendront le retour du schmod777. Ceci dit, rien ne nous empêche d'aller consulter Wikipedia, afin de mieux connaitre ces nombres flottants que nous allons utiliser :

> An IEEE 754 32-bit base-2 floating-point variable has a maximum value of $(2 - 2^{-23}) \times 2^{127} \approx 3.4028235 \times 10^{38}$. All integers with 7 or fewer decimal digits, and any 2^n for a whole number $-149 \leq n \leq 127$, can be converted exactly into an IEEE 754 single-precision floating-point value.
>
> In the IEEE 754-2008 standard, the 32-bit base-2 format is officially referred to as binary32 ; it was called single in IEEE 754-1985.

https ://dl.acm.org/doi/pdf/10.1145/103162.103163

1.1 Dynamique

Dynamique, précision et *macheps* ? Peu de gens connaissent la fourbitude des calculs en virgule flottante avec les ordinateurs. Moi-même compris. Il est évident qu'une étude théorique doit être effectuée afin d'éviter les potentiels inconvénients.

Ceci dit, le standard *ieee754* nous indique qu'il y a 23 bits pour la mantisse, ce qui nous propose déja plus de huit millions de valeurs. Considérons un cas typique : le cumul de N images ayant un niveau maximum Vm.

1.2 Pixel négatif ?

Il est très difficle d'imaginer une lumière négative. Sauf peut-être si nous songeons à des coefficients d'absorption, ou un canal *alpha* qui inverserait les valeurs ? Un domaine dont l'exploration peut confiner au mysticisme de part la multitude des traitement possibles : valeur absolue, *clamping*, report sur les autres composantes. Votre imagination est la limite.

2 Premier exemple

FloatImg a débuté sous la forme de quelques fonctions basiques en C, gérant la structure des données d'image en mémoire et sur disque. Ça a été imaginé de façon presque empirique, mais nous sommes tous là pour améliorer les choses, dans la mesure de nos moyens. Nous allons donc directement rentrer au cœur du problème, en écrivant quelques lignes de code montrant le fonctionnement général de la chose.

2.1 L'idée

Pour commencer par quelque chose de simple, nous allons créer une image RGB complètement noire, puis l'enregistrer dans un fichier `.fimg`, un format complètement inconnu, puisque je viens de l'inventer à l'instant même [3].

Tout d'abord, nous devons déclarer et garnir quelques variables pour gérer la machinerie interne.

3. Enfin, non, il y a déja longtemps, avant la grande pandémie.

```
int             width = 640, height = 480;
char            *fname = "quux.fimg";
FloatImg        fimg;
```

Ensuite, nous enchainerons trois étapes : la création de l'image en mémoire centrale, l'initialisation des valeurs de chaque pixel à 0.0 (une valeur que certains associent au noir complet, et d'autres à une impossibilité quantique), et pour conclure, l'enregistrement de cette image dans un fichier [4] binaire.

```
memset(fimg, 0, sizeof(FloatImg));
foo = fimg_create(&fimg, width, height, FIMG_TYPE_RGB);
if (foo) {
        fprintf(stderr, "create floatimg -> %d\n", foo);
        exit(1);
        }
fimg_clear(&fimg);
foo = fimg_dump_to_file(&fimg, fname, 0);
if (foo) {
        fprintf(stderr, "dump fimg -> %d\n", foo);
        exit(1);
        }
```

Une fois ce code enrobé dans un `main()`, compilé puis exécuté, nous pouvons entrevoir, grâce au logiciel `fimgstats` (décrit en page 16), le résultat sous forme de chiffres divers, et/ou inutiles :

```
$ ./fimgstats quux.img
----------- numbers from 'quux.img' :
  640    480   3  0x7f3718c4f010 0x7f3718d7b010 0x7f3718ea7010
surface      307200
mean values:
      R         0.000000
      G         0.000000
      B         0.000000
      A         0.000000
max value       0.000000
```

2.2 Action ?

Nous avons donc sous la main un ensemble d'outils mécaniques qui ne demande qu'à faire des trucs futiles et des images qui clignotent. Avec un bon script bash, il y a déja de quoi faire. La suite vers la page 6.

Vous trouverez dans le répertoire `tools/` d'autres exemples de mise en œuvre de quelques fonctions disponibles sous formes d'outils en ligne de commande, lesquels outils sont approximativement décrits en page 15.

3 Installation

Sauf indications contraires, ces instructions se réfèrent à une distribution Debian récente (amd64 et x86), mais ça marche quasiment pareil avec Fedora 64, et probablement Raspbian, modulo les éventuels soucis de boutisme. *Attention, ça va devenir un peu gore...*

4. Au format ésotérique, mais très véloce.

3.1 Prérequis

Vous devez, en dehors des outils classiques (gcc, Awk, make...), avoir quelques bibliothèques installées : libv4l2, libpnglite, libtiff, libnetpbm [5], libz [6], libcurses, libcfitsio... éventuellement avec le -dev correspondant, qui contient, entre autres, les fichiers `.h` associés et probablement d'autres choses.

Il est même quasiment certain que Bash soit indispensable, tout comme GNU/make. Une connaissance de base de l'utilisation du shell et de l'écriture de Makefile's sera un plus.

Il faut aussi savoir où trouver le code. Il est dans le Git du Tetalab [7]. Mais comme je ne comprend rien à Git, c'est pas la peine de m'envoyer des trucs genre *pull-request* auquels je ne comprend rien.

3.2 Compilation

La première chose à faire est d'aller regarder le contenu du fichier `Global.makefile` à la racine du projet. Il contient des informations pertinentes pour la suite des choses. Ensuite, un script `build.sh` permet de construire approximativement le bouzin. Il est loin d'être parfait [8]. Dans chacun des répertoires à traiter, ce script devrait trouver un Makefile et un fichier `t.c` qui est le source de la cible par défaut du make.

Pour le moment, la procédure d'installation est un peu rude, pour ne pas dire clairement sommaire. Si le résultat de l'étape compilation vous semble correct, vous pouvez copier les deux fichiers `floatimg.h` et `libfloatimg.a` dans un emplacement approprié, par exemple `/usr/local/include` et `/usr/local/lib`.

Le script `install.sh`, à la racine du projet, est censé faciliter un peu la chose. Il prend également en compte la copie des divers binaires du dossier `tools/` (cf page 15) dans le répertoire prévu à cet effet : `/usr/local/bin`.

Il reste enfin quelques exemples d'utilisation des outils de la ligne de commande depuis un shell dans le répertoire `scripts/`. Ils sont décrits plus en détail page 20, et c'est à vous de les adapter à votre *usecase*. Faites-en ce que vous voulez.

4 Utilisation coté codeur

Classiquement, il y a un fichier `.h` à inclure dans chacun de vos codes source, `floatimg.h`, généralement logé dans `/usr/local/include` et contenant un certain nombre de définition de structures, de macros, de constantes [9] et les prototypes des fonctions utilisables par vos logiciels.

Au niveau du code source, ces fonctions sont très approximativement classées en deux catégories : `lib/` et `funcs/`. La première contient les choses qui sont relativement figées, et la seconde celles qui risquent de bouger. Cette classification est en fait indécement arbitraire. D'autant plus qu'il y a aussi un répertoire nommé « experimental ».

4.1 Structures, macros...

Les pixels flottants d'une image résidant en mémoire centrale sont décrits par un ensemble de données (certains appelent ça des *metadatas*) regroupées dans une jolie structure que nous allons

5. package libnetpbm10-dev
6. package zlib1g-dev
7. `https://git.tetalab.org/tTh/FloatImg`
8. Il doit être possible de faire un Makefile récursif, mais...
9. À l'ancienne, via le pré-processeur

examiner dès maintenant.

```
/*      in memory descriptor */
typedef struct {
        unsigned long   magic;
        int             width;
        int             height;
        int             type;
        float           fval;
        int             count;
        float           *R, *G, *B, *A;
        int             reserved;
        } FloatImg;
```

Le premier champ, `magic`, servira un de ces jours à robustifier l'ensemble du machin. Les deux suivants sont *obvious*. Le troisième est le type d'image : pour le moment, il y en a un certain nombre qui sont définis [10] : gris, rgb et rgba/rgbz. Les constantes adéquates sont déclarées dans `floatimg.h` :

```
#define FIMG_TYPE_GRAY              1
#define FIMG_TYPE_RGB               3
#define FIMG_TYPE_RGBA              4
#define FIMG_TYPE_RGBZ              99
```

Un peu plus loin, nous avons les pointeurs vers les différents *pixmaps* de l'image. En principe l'organisation interne de ces zones est improbable, puisque elle dérive d'idées approximatives. C'est cette utilisation constructive de larache qui fait que seuls les champs documentés de cette structure ne sont pas explosifs.

Mais revenons aux choses sérieuses... Les deux champs suivants (fval et count) sont à la disposition du *yuser* qui peut jouer avec à loisir pour faire, par exemple, ce genre de chose : imaginons un périphérique de capture qui nous fournisse des images en gris sur 4 bits (et linéaire). Et que nous voulions cumuler quelques images...

Le champ *count* sera mis à 0 et le champ *fval* sera initialisé à 15.0 (ce qui est la valeur maximale que peut renvoyer ce capteur). Ensuite, dans la boucle capture/cumul, *count* sera incrémenté à chaque passe, et nous aurons donc, en finale, toutes les informations nécessaires pour exploiter au mieux la dynamique de notre image dans les étapes ultérieures, puisque la valeur maximale théorique est égale à $fval * count$.

La fonction `fimg_printhead(FloatImg *h)` affiche sommairement le contenu de ce descripteur, et `fimg_describe(FloatImg *head, char *txt)` propose un affichage plus détaillé. Ça peut parfois aider.

Une bonne partie des fonctions que nous allons voir est indéterministe. Ce qui veut dire, en langage de tous les soirs, que ça risque de ne pas être la même chose dans l'avenir. Mais après tout, ce n'est encore qu'un concept en devenir, n'est-il pas ?

4.2 Les fondations

La première chose que nous devons absolument voir est la gestion dynamique de la mémoire qui sera occupée par tous ces pixels flottants, ce qui est un sujet parfois délicat [11] parce que les pixels sont précieux et doivent être bien rangés. Cette gestion est donc faite, à la base, par ces deux fonctions :

10. et plus ou moins bien gérés...
11. GC or not GC ?

```
int fimg_create(FloatImg *fimg, int w, int h, int type);
int fimg_destroy(FloatImg *fimg);
```

L'appelant doit lui-même gérer le descripteur d'image (une structure C décrite plus haut) en le considérant comme un type semi-opaque dont la forme peut varier, mais qu'il convient de mettre à zéro avant le moindre usage [12]. Certains membres de cette structure sont documentés dans ce document, et les autres sont dangereux à toucher. Les types d'images actuellement gérés sont les trois grands classiques : niveau de gris, rouge-vert-bleu et rgb avec un canal alpha, et expliquées quelques lignes plus haut.

Comme vous allez le voir plus loin, il y a plein de fonctions qui prennent en argument deux images : la source et la destination. Dans la plupart des cas, ces deux images doivent être compatibles, c'est à dire même type et mêmes dimensions.

```
/* return 0 if pictures are compatible */
int fimg_images_not_compatible(FloatImg *a, FloatImg *b);
```

C'est bien beau d'être enfin une image résidente en mémoire centrale, mais pouvoir aussi exister à long terme en étant stocké dans la matrice est tout aussi pertinent. Il y a deux opérations qui supportent le reste des transits ram/ps. Le format de ces fichiers est décrit page 11.

```
int fimg_dump_to_file(FloatImg *fimg, char *fname, int notused);
int fimg_load_from_dump(char *fname, FloatImg *where);
```

Recharger une image depuis un fichier nécessite que celle-ci et l'image de destination en mémoire ait précisément les mêmes caractéristiques (taille, type...), donc l'image en mémoire centrale doit être pré-allouée. On peut connaitre ces valeurs en appelant `int fimg_fileinfos(char *fname, int datas[3])`.

Si tout s'est bien passé (valeur retournée égale à 0), on va trouver la largeur dans `datas[0]`, la hauteur dans `datas[1]` et le type dans `datas[2]` [13].

Je sais aussi que certains d'entre vous aiment la facilité, aussi je vais vous révéler l'existence d'un nouveau truc bien plus simple, une fonction qui enchaine ces deux actions (allocation, puis lecture), et s'utilise comme ça :

```
FloatImg       head;
memset(&head, 0, sizeof(FloatImg));
foo = fimg_create_from_dump("lena.fimg", &head);
```

Si la valeur retournée est différente de 0, c'est que quelque chose s'est probablement mal passé. Certains messages sont parfois explicites. Mais parfois non. Quand aux valeurs retournées en cas d'erreur, c'est le désordre intégral [14].

4.3 Dessiner

Bon, vous avez une image latente, et vous souhaitez dessiner dessus (ou dedans ?) avec vos encres flottantes ? Il y a des fonctions pour ça, par exemple :

```
int fimg_plot_rgb(FloatImg *head, int x, int y, float r, float g, float b);
```

Les paramètres sont explicites, mais leur validité doit être sévèrement controlée par l'appelant. Il y a une fonction soeur, `fimg_add_rgb`, qui ajoute du rgb à un pixel, laquelle fonction a d'ailleurs

12. `man 3 memset`
13. La fonction `fimg_type_is_valid(int type)` peut vous aider...
14. Un vrai désastre, même...

été à la base de la seconde génération de la photographie en cumul.

Inversement, la fonction `fimg_get_rgb(FloatImg *head, int x, int y, float *rgb)` permet de lire les valeurs des trois canaux d'un pixel donné. Là aussi, il n'y a aucun contrôle sur la validité des valeurs x et y de la demande.

Quand au canal *alpha*, il est pour le moment superbement ignoré. Ceci dit, on vient de me faire remarquer qu'il peut être utilisable aussi pour faire du *z-buffer*, une technique classique dans la génération d'images en trois dimensions....

4.4 Contraste

Certaines opérations d'ajustement du contraste d'une image semblent cohérentes avec la notion d'image flottante. Certaines d'entre elles, les plus simples, sont disponibles. Les autres sont à imaginer.

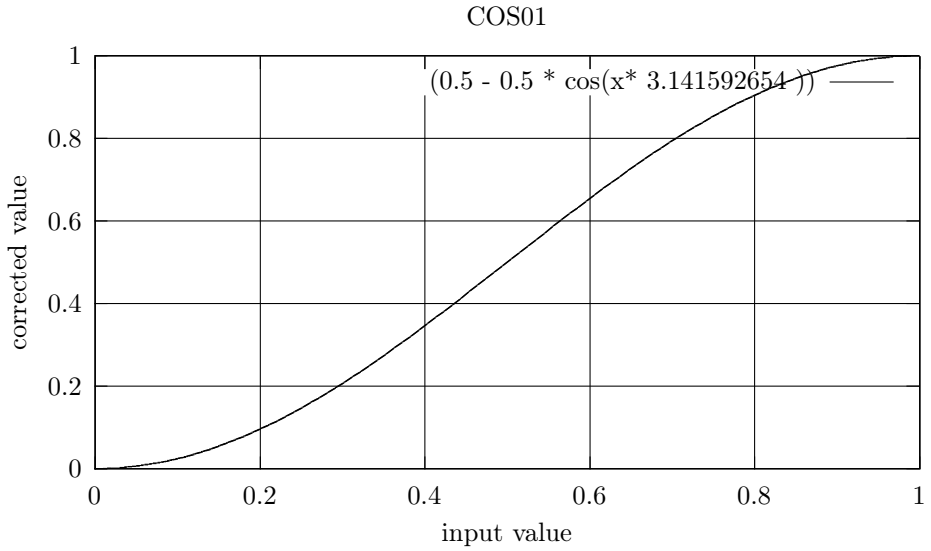

FIGURE 1 – Correcteur cos01

Ils prennent chacun trois paramètres, d'abord les images source et destination (`* FloatImg`), ensuite le troisième qui est un nombre en double précision donnant la valeur maximale *supposée* de l'image source, valeur qui peut être déterminée de plusieurs manières.

```
/* source in lib/contrast.c */
int fimg_square_root(FloatImg *s, FloatImg *d, double maxval);
int fimg_power_2(FloatImg *s, FloatImg *d, double maxval);
int fimg_cos_01(FloatImg *s, FloatImg *d, double maxval);
int fimg_cos_010(FloatImg *s, FloatImg *d, double maxval);
```

Si vous souhaitez rajouter votre propre méthode de modification de contraste, il y a quelques explication en page 14. Mais rien ne vous oblige à le faire. Sauf vos envies.

Rappelons qu'il est possible pour un logiciel applicatif comme `grabvidseq` (cf page 23) de renseigner deux champs du descripteur d'image avec des données pertinentes. Ces deux champs sont *fval* et *count*. Dans ce cas particulier, le premier indique la valeur maximale du capteur, et le second sert à compter le nombre de capture [15] effectuées.

15. Et c'est bien géré aussi dans l'upscaling.

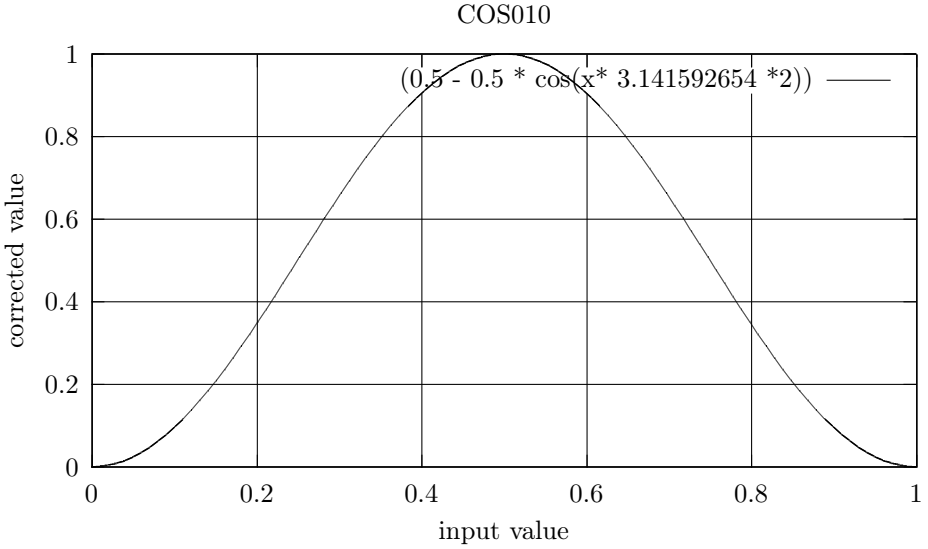

FIGURE 2 – Correcteur cos010

La fonction `fimg_normalize(FloatImg *fi, double maxima, int notused);` tente de gérer ce cas d'utilisation. Son ajout au captureur d'images floues sera probablement le bienvenue. Je me suis bien rendu compte à l'usage [16] en situation festive qu'il manquait des données dans la chaine de traitement.

L'autre façon de procéder est d'explorer notre image à la recherche de la valeur maximale. La fonction `float fimg_get_maxvalue(&fimg)` est prévue pour ça de façon sommaire. C'est actuellement la méthode utilisée par l'outil qui sert à faire les modifications de contraste (page 16). On pourra aussi envisager d'utiliser `fimg_get_minmax_rgb(FloatImg *head, float mmvals[6])`, qui permet un contrôle bien plus fin des dérives.

La prochaine étape consistera à trouver une façon de faire une égalisation par histogramme qui respecte, dans toute sa futilité, le concept de pixel flottant. *Et c'est pas gagné...*

4.5 Géométrie

Très prochainement, le retour du blitter. Et pour attendre, un truc improbable, voire même inutile, en fait l'inverse de l'upscaling.

```
int fimg_halfsize_0(FloatImg *src, FloatImg *dst, int notused);
int fimg_halfsize_1(FloatImg *src, FloatImg *dst, int notused);
```

Attention lors de l'appel, le descripteur `dst` ne doit pas contenir d'image, et doit être effacé avec un bon `memset(&result, 0, sizeof(FloatImg));` bien senti.

La première propose un résultat très moyen : il n'y a pas d'interpolation, alors que la seconde semble bien mieux.

```
int fimg_extract_0(FloatImg *src, FloatImg *dst, int x, int y);
```

Contrairement à la fonction précédente, celle-ci demande absolument une image de destination initialisée aux dimensions (largeur et hauteur) désirées.

16. Une histoire de *fonderie*, un logiciel censé faire des films flous à partir d'images floues

```
int fimg_rotate_90(FloatImg *src, FloatImg *dst, int notused);
```

Rotation de 90 degrés dans le sens horlogique [17] d'une image RGB. L'image de destination peut être soit vierge, soit pré-allouée aux bonnes dimensions (échange W et H).

4.6 Format du fichier FIMG

D'un design très empirique, c'est certainement à revoir pour l'avenir. La force du *legacy* va-t-elle dominer le monde ? Il faudrait normaliser l'endianess et le packing dans les structs [18], et surtout l'ajout de données sur la prise de vue, du genre type de capteur, date et heure, réglages divers...

```
typedef struct {
        char            magic[8];
        int32_t         w, h, t;
        } FimgFileHead;
```

Le champ `magic[8]` doit contenir une valeur magique : les quatre premiers octets doivent contenir les quatre caractères `'FIMG'`, et les quatre derniers doivent être à 0, sauf que, voir plus bas, le cinquième vas vous étonner. Le champ `t` (le type de l'image) doit avoir les trois octets de poids fort à 0 [19].

Ensuite, nous aurons (dans le cas courant) : 1, 2 ou 4 blocs de WxH pixels sous forme de Float32. La première ligne lue est la ligne du haut de l'image. Les valeurs négatives sont tolérées.

Vous trouverez les constantes de type pertinentes dans le fichier `floatimg.h`, et quelques informations (non-)essentielles qui ne vous serviront probablement à rien.

4.7 Métadonnées

Attention, ce n'est pas encore une version définitive, beaucoup de choses restent à préciser sur le contenu de cette structure, mais l'essentiel est déja là. On reconnait un fichier avec metadata quand l'octet `magic[4]` du premier header est égal à la lettre `'a'`.

```
typedef struct {
        char            magic[8];
        struct timeval  timestamp;
        int32_t         count;
        float           fval;
        char            idcam[32];
        int32_t         origin;                 // enum ?
        } FimgMetaData;
```

Voyons maintenant chacun des champs de cette structure, en prenant bien en compte qu'à ce moment [20], tout n'est pas figé. Ceci dit, nous allons aussi retrouver de vieilles connaissances.

to be continued

17. ou trigonométrique,le code et la doc ne semblent pas d'accord.
18. Directives du compilateur ?
19. Pourquoi ? Je n'en sais rien.
20. 4 avril 2022

4.8 Exportation & Importation

Notre format de fichier étant totalement inconnu [21], il nous faut bien exporter nos images en quelque chose de plus connu. Bien entendu, c'est toujours affaire de compromis entre précision de valeurs et taille des fichiers.

Et dans le sens inverse, il serait bien de savoir importer le monde extérieur dans nos sombres caves à pixel. Il faut quand même reconnaitre que c'est un peu la jungle dans les formats de fichiers d'image, ce qui explique le retard du développement dans ce domaine...

4.8.1 Vers PNM

Nous avons ici 16 bits par composante, soit 65536 valeurs différentes, ce qui est bien au-delà de ce que peuvent percevoir vos yeux. Hélas, c'est au prix d'une taille énorme sur les fichiers. D'un autre coté, l'utilisation du codage ASCII (alors qu'on pourrait mettre du binaire, plus compact) y est pour quelque chose.

```
int fimg_save_as_pnm(FloatImg *head, char *fname, int flags);
```

Le bit `0` du paramètre `flags` mis à `1` demande à la fonction de faire la mise à l'échelle avec le couple *fvalue/count* décrit plus haut dans cette doc. Et si il est à zéro, c'est la fonction de recherche de valeur maximale (cf page 9) qui est utilisée.

Le bit `1` permettra bientôt de demander l'enregistrement de métadonnées pertinentes, telle que l'epochtime de l'enregistrement.

Les autres bits ne sont pas utilisés et doivent être à zéro.

4.8.2 Vers PNG

Actuellement, on ne peut enregistrer qu'en mode RGB uniquement, avec 8 bits par composante, mais on a quand même une bonne compression, ça compense.

J'utilise *libpnglite* avec qui j'ai un peu de mal à suivre. Mais je me soigne. Le mode 16 bits va bientôt arriver. On peut aussi songer à l'export des metadonnées, pour celles qui sont compatibles avec le standard PNG.

```
int fimg_save_as_png(FloatImg *src, char *outname, int flags);
```

Tous les flags doivent être à zéro. Sinon, ça foire parfois. Et en fait (mars 2021) je ne suis pas très content de `pnglite`, donc un de ces jours, je prendrais ~~cinq~~ quelques jours pour régler ce souci en passant à la bibliothèque canonique.

4.8.3 Vers TIFF

Le format canonique de la PAO du siècle dernier. Il permet de gérer une foultitude de formats numériques. C'est aussi un format classique proposé par les gros scanners corporates.

```
int fimg_write_as_tiff(FloatImg *src, char *outname, int flags);
```

Tous les flags doivent être à zéro. Pour le moment. Un premier jet pas forcément parfait.

21. Du monde extérieur, vous l'aurez compris.

4.8.4 Vers FITS

Ce format est essentiellement utilisé pour stocker des images d'astronomie, donc il peut aussi servir pour des images floues. Cette partie est basée sur la bibliothèque `cfitsio` de la NASA.

```
int fimg_save_R_as_fits(FloatImg *src, char *outname, int flags);
int fimg_save_G_as_fits(FloatImg *src, char *outname, int flags);
int fimg_save_B_as_fits(FloatImg *src, char *outname, int flags);
```

Bizarrement, l'image est stockée *upside-down* et je ne sais pas encore comment régler ce petit détail.

Tous les flags doivent être à zéro.

4.9 Utilitaires

Commençons par quelques petits trucs pour nous faciliter la vie dans des domaines annexes, tels que l'interprétation d'arguments dans la ligne de commande ou un fichier de configuration.

```
int parse_WxH(char *str, int *pw, int *ph)
int parse_double(char *str, double *dptr)
```

La fonction `int format_from_extension(char *fname)` examine un nom de fichier tel que `lena.xxx`, et retourne, si la partie `xxx` est connue, un éventuel nombre positif, dont les valeurs sont déclarées dans floatimg.h le valeureux. Les extensions actuellement connues sont : fimg, png, pnm, pgm, fits et tiff. Le bmp est plus ou moins prévu...

To be continued...

4.10 Effets

Quelques routines qui servent futilement à *brotcher* les images en tripotant les flux visuels chromatiques.

```
int fimg_killcolors_a(FloatImg *fimg, float fval);
int fimg_killcolors_b(FloatImg *fimg, float fval);
int fimg_colors_mixer_a(FloatImg *fimg, float fval);
```

4.11 Glitches

Un *glitch* peut-il être classé dans la catégorie des effets spéciaux ou non ? Hmha, non. un fx est paramétrable et surtout répétitif. Un glitch est quasiment souvent un phénomène aléatoire et tout aussi paramétrable.

J'ai commencé à étudier ces objets étranges quand j'ai commencé à travailler sur l'interpolator à l'automne 2020. Hélas, j'ai vite réalisé que c'était assez délicat. Pour ce genre de *usecase*, le numérique est pitoyable si on le compare au (hélas défunt) Betamax.

4.12 Filtrages

Pour commencer, il faut que je réfléchisse au traitement des bordures des images. Ensuite que je débuggue ces deux fonctions :

```
int fimg_lissage_2x2(FloatImg *img);
int fimg_killborders(FloatImg *img);
```

Bon, oké, ça marche ? Passons à l'étape suivante. La convolution avec une matrice 3x3, c'est possible. Et pas trop compliqué à faire. Bon, il reste le souci avec les bordures, souci qui ne peut être que temporaire, mais ésotérique à fixer.

Passons maintenant aux choses sérieuses, et définissons la description d'un filtre 3x3.

```
typedef struct {
        float                   matrix[9];
        float                   mult;
        float                   offset;
        } FimgFilter3x3;
```

L'usage des champs `mult` et `offset` n'est pas clairement défini. Le prototype de la fonction de filtrage non plus, mais assez simple quand même. Source et destination ne peuvent désigner la même image, et le champ `matrix` du filtre doit contenir des valeurs cohérentes.

```
int fimg_filter_3x3(FloatImg *src, FloatImg *dst, FimgFilter3x3 *filtr)
```

Comme dans la plupart des cas, la gestion des valeurs négatives de pixel est laissé au hasard, qui fait souvent du portnawak. Quoique, il doit bien exister quelques solutions de contournement : clamping ou shift ?

To be continued...

4.13 Exemple de fonction

Nous allons maintenant écrire une fonction intégrable dans le répertoire `funcs/`. Elle aura donc accès aux *internals*[22] de FloatImg, une chose qui est en principe interdit aux programmes *enduser*. Soyez prudents.

Cette fonction va faire quelque chose à partir d'une image source et d'une valeur, et écrire le résultat dans une image de destination. Pour simplifier les choses, nous n'allons traiter que les images de type FIMG_TYPE_RGB, de loin le plus répandu par les temps qui courent.

```
int fimg_example(FloatImg *s, FloatImg *d, float value)
{
int     size, index;

if ( s->type!=FIMG_TYPE_RGB || d->type!=FIMG_TYPE_RGB) {
        perror("fimg_example");
        return -1;
        }

size = s->width * s->height;
for (idx=0; idx<size; idx++) {
        d->R[idx] = s->R[idx] - value;
        d->G[idx] = s->G[idx] - value;
        d->B[idx] = s->B[idx] - value;
        }

return 0;
}
```

Je vous laisse imaginer les dégats que peut faire cette fonction en utilisation réelle. Mieux,

22. que je peux décider de changer n'importe quand

je vous propose d'essayer par vous-même, vous allez voir, c'est assez créatif. En particulier tout le reste du code qui suppose qu'un pixel ne peut **pas** être négatif va peut-être exploser de rire. Vous pouvez aussi remarquer qu'il n'y a pas de controle de cohérence sur les dimensions des deux images, malgré l'existence de fonctions prévues à cet effet, mais il fallait rester simple...

5 Les outils

3615mavie : sur des projets comme celui-ci, qui travaillent in-fine sur des objets que l'on peut considérer comme « physiques », il est important de passer à une utilisation normale [23] et construire des trucs qui mettent en action le code primitif pour partir sur des bases saines.

Ces machins ont en commun quelques options bien pratiques : `-h` pour avoir un résumé des options disponibles, `-v` qui augmente la puissance de bavardage, et `-K nn.nn` pour un paramètre flottant. Dans un avenir incertain, il existera des pages de man.

5.1 mkfimg

Propose la création d'un fichier contenant une image de « teinte » constante (ou pas). Cette notion de teinte est assez inconsistante pour le moment, mais ça n'est pas si grave que ça.

```
tth@debian:~/Devel/FloatImg/tools$ ./mkfimg -h
Usage:  mkfimg [options]  quux.fimg width height
        -k N.N  give a float parameter
        -t type         howto make the pic
                        black, drand48...
        -v      increase verbosity
```

Il y a deux syntaxes possibles pour les dimensions de l'image générée : deux nombres séparés ou la notation `WIDTHxHEIGHT`.

La plupart des types d'image générée prennent un paramètre flottant qui devra être donné avec l'option `-k F.F` avec une valeur par défaut à 1.0, ce qui n'est pas toujours une bonne valeur, ymmv.

black/gray/grey : efface avec 0.0 (black) ou avec la valeur `-k` (niveau de gris).

drand48 : beaucoup de bruit dans chacun des canaux.

hdeg/vdeg : dégradé du noir au blanc (relatif à `-k`).

5.2 png2fimg

Grosse panne à réparer.

```
tth@debian:~/TMP/floatimg$ png2fimg A.png foo.fimg
error in 'fimg_create_from_png' : read png -> -1 File error
png2fimg : err -1, abort.
```

Il faut peut-être envisager le passage à `libpng`, la bibliothèque actuellement utilisée (`pnglite`) ayant tendance à borker en lecture sur certains fichiers pourtant corrects.

23. Il y a une vie en dehors de git.

5.3 fimgstats

Affichage de quelques valeurs calculées à partir du contenu d'un fichier `.fimg`.

```
usage : fimgstats [options] file.fimg
        -c      make a machinable csv
        -v      increase verbosity
```

À vrai dire, je ne sais pas encore quelles métriques seront utiles en première approche, alors commençont par le plus simple, les valeurs moyennes de chaque composante.

Puis nous rajouterons [24] le calcul de la variance. Les compétences de `schmod777` sont attendues au dd2.

5.4 fimgfx

Ce programme, *en cours de création*, applique un effet spécial à une image. À l'heure actuelle [25], nous avons déja quelques ajustements basiques de contraste, qui ne tiennent pas vraiment compte du contenu de l'image.

```
tth@daubian:~/Devel/FloatImg/tools$ ./fimgfx -v -h
--- fimg special effects ---
        cos01 cos010 pow2 sqrt gray0 cmixa xper desat
```

Certaines de ces opérations ont besoin d'un paramètre flottant. Celui-ci peut être fixé avec l'option `-k`. Une liste détaillée des opérations possibles sera lisible avec le sélecteur `-L`.

Ajustements de contraste : cos01 cos010 pow2 sqrt

Distorsions chromatiques : gray0

Déformations géométriques : r90

Et pour les aventureux, la commande `xper` (abréviation de *expérimental*) permet de tester la plus récente tentative de friture du moment. D'autre part un set bien plus complet d'effets àlc est disponible dans les logiciels de flou temporel, décrits plus loin dans ce document.

5.5 fimgops

Quelques opérations diverses entre deux images, qui doivent être de la même taille, et uniquement du type *RGB*. Certaines de ces opérations peuvent avoir un effet étrange sur vos images, par exemple si un pixel se retrouve avec une valeur négative.

```
usage:
        fimgops [options] A.fimg B.fimg operator D.fimg
operators:
        add             1
        sub             2
        mix             3
        mul             4
        mini            5
        maxi            6
options:
```

24. Les patchs sont les bienvenus
25. janvier 2019, vers 13 :37

```
        -g              convert output to gray
        -k N.N          set float value
        -v              increase verbosity
        -X              explosive action
```

Pour des operateurs paramétrable (comme `mix`), le paramêtre flottant doit être fourni en utilisant l'option `-k`. La véracité mathématique n'est pas garantie. Et n'oubliez pas que les valeurs négatives peuvent être la cause de *glitches* de qualitay.

La liste des opérations est susceptible d'être agrémenté de quelques possibilités bien féroces [26].

5.6 fimg2png, fimg2pnm, fimg2tiff, fimg2fits

Quelques petits proggies pour exporter notre format secret vers des choses plus directement utilisables. À condition que le code soit écrit et documenté.

D'un autre coté, écrire un greffon d'import/export pour Gimp ou ImageMagick ou Krita ne devrait pas être trop difficile. Des volontaires ?

D'ailleurs, pourquoi N logiciels indépendants alors q'un seul devrait être nécessaire ?, ce qui me conduit à envisager un *exporter* universel, mais dont l'ergonomie et les paramètres possibles doivent être réfléchis avec vigueur et nonchalance.

5.7 fimg2text

Voici donc quelque chose qui retourne vers la bonne vieille tradition Unix et permet de traiter des images flottantes avec des outils classiques comme l'injustement méconnu `Awk`. Dans le comportement par défaut (le seul actuellement) cet outil sort, pour chaque pixel, cinq valeurs :

- Les coordonnées entières x, y du pixel
- Les valeurs flottantes des composantes RGB

On peut même espérer qu'un jour, il soit possible de faire la transformation inverse. Quelle joie de générer des images flottantes avec un programme écrit en *gw-basic* :)

```
tth@fubar:~/Devel/FloatImg/doc$ fimg2text -h
usage:
        fimg2text [options] foo.fimg > bar.csv
options:
        -v              increase verbosity
        -n 3.14         normalize picture
        -s N            steps on x & y
```

Vous voulez un petit exemple ?

```
tth@fubar:~/Devel/FloatImg/doc$ mkfimg -t drand48 quux.fimg 4 2
tth@fubar:~/Devel/FloatImg/doc$ fimg2text -s 1 quux.fimg
0 0 142.518127 84.036987 20.688946
1 0 164.273315 68.397079 138.501587
2 0 45.887970 25.905518 93.683243
3 0 53.292942 222.000000 111.711548
0 1 82.289795 113.054855 180.530014
1 1 42.051765 114.543625 4.249123
2 1 71.826775 96.219109 63.222294
3 1 160.535675 136.104919 204.020691
tth@fubar:~/Devel/FloatImg/doc$
```

26. Stay tuned, flim at 11.

Et maintenant, vous voulez un gros exemple ? Bah, ça doit demander l'utilisation de Povray.

5.8 fimg2gray

Nous avons vu dans ce document que chaque image flottante pouvait avoir plusieurs plans de réalité. Il ne faut en négliger aucun, voire même tenter de les faire se rencontrer, se combiner et s'influencer mutuellement.

Il faut quand même deviner que pour passer de l'espace RGB à une abstraction linéaire monodimensionnelle, il existe une foultitude de méthodes, toutes plus légitimes que les autres. Et face à l'incertitude du choix, j'ai reporté l'écriture de ce logiciel aux calendes grecques, voire même plus tard, après la grande pandémie...

5.9 cumulfimgs

Cet outil accumule une quantité d'images flottantes (de même taille et de même type) afin d'obtenir un flou temporel de meilleure qualité. Aucune mise à l'échelle n'etant effectuée, les pixels de sortie peuvent atteindre des valeurs considérables [27]

```
tth@delirium:~/Devel/FloatImg/tools$ ./cumulfimgs -h
usage :
        cumulfimgs a.fimg b.fimg c-fimg ...
cumulator options :
        -v      increase verbosity
        -o      name of output file
        -g      convert to gray level
```

Le nom par défaut du fichier résultant est `out.fimg`. L'exportation « multiformat » est enfin fonctionnelle, dans la mesure des formats connus.

6 Debug

Et quand plus rien ne fonctionne normalement ?

Ayant une attirance marquée pour la méthode de développement dite de *larache*, que je pratique intensément, j'ai quand même besoin de m'équiper de quelques garde-fous. Avec l'expérience, j'ai découvert quelques méthodes bien utile quand on est à court de poudre verte.

6.1 À la compilation

Première de cordée, la méthode `DEBUG_LEVEL`. Elle me sert essentiellement à tracer les appels de fonctions, et les paramètres qu'elles reçoivent. Ça fait vraiment partie de la création des *boilerplates*. C'est une valeur numérique entière définie dans chaque `Makefile` qui sera ensuite utilisée par le préprocesseur afin de rajouter ce genre de code dans l'exécutable :

```
int fimg_export_picture(FloatImg *pic, char *fname, int flags)
{
int filetype, foo;
#if DEBUG_LEVEL
fprintf(stderr, ">>> %s ( %p '%s' 0x%X )\n", __func__, pic, fname, flags);
#endif
...
```

27. Faut-il prévoir une gestion des overflows ?

Le principe est simple, n'est-il pas, alors pourquoi s'en priver ? Bien entendu, la même constante peut être utilisée plus agressivement avec une condition comme `#if DEBUG_LEVEL > 2` permettant de générer encore plus de messages traçants.

Ensuite, pour les cas les plus graves, qui nécessiteront l'utilisation du dévermineur `gdb`, il y a la directive `MUST_ABORT` qui est bien plus violente, mais parfois bien pratique. Voici le cas classique d'utilisation :

```
foo = fimg_strange_function(&picture, 13.37);
if (foo) {
        fprintf(stderr, "epic fail %d in %s\n", foo, __func__);
#if MUST_ABORT
        abort();
#endif
...
```

À condition d'avoir bien réglé votre ulimit pour la génération d'un coredump, vous aurez sous la main un fichier `core` qui vous permettra de, par exemple, remonter la pile d'appel avec la commande `back` de gdb. Mais pour le moment, juste une infime partie du code est instrumentée avec ce dispositif.

6.2 À l'exécution

De l'utilisation des nombres magiques dans la détection des structures corrompues par un pointeur ayant perdu le nord...

De l'utilisation des variables d'environnement pour transformer des warnings en erreurs fatales...

7 TODO

Il reste plein de choses à faire pour que ce soit vraiment utilisable, surtout dans un contexte artistique à grande porosité. C'est par ces frottements de techniques ayant du sens que les choses seront acquises, pour le pire, le meilleur et la techno-futilité du monde futur..

- ✷ Import/export au format TIFF à continuer.
- ✷ Remplacer le « fait-maison » par LIBNETPNM. *[en cours]*.
- ✷ Compléter les traitements mathémathiques (eg le gamma).
- ✷ Formaliser les codes d'erreur. **Urgent**.
- ✷ Faire une passe complète de Valgrind.
- ✷ Intégrer la fonderie, l'interpolator et le singlepass.
- ✷ Vérifier le gestion des images mono-canal et de la transparence.

8 Exemples pour yusers

Nous allons *essayer d'improviser* un exemple presque réel, avec un peu de rache dedans, et beaucoup de simplification. Ce qui est autorisé dans les exemples, mais dans la vrai vie, il ne faut jamais négliger le traitement des éventuelles erreurs.

Nous savons générer une image contenant des pixels aux valeurs probablement aléatoires, avec la commande `mkfimg`, qui utilise le `drand48` de POSIX. Maintenant, posons-nous une question de statisticien : que se passe-t-il si nous faisons la somme de plusieurs centaines de ces images ?

```
#!/bin/bash
ACCU="quux.fimg"
TMPF="tmp.fimg"
DIMS="320 240"
mkfimg $ACCU $DIMS
for i in {0..1000}
do
        mkfimg -t drand48 ${TMPF} ${DIMS}
        fname=$( printf "xx%04d.pnm" $i )
        fimgops $ACCU $TMPF add $ACCU
        fimg2pnm -v -g $ACCU $fname
done
convert -delay 10 xx*.pnm foo.gif
```

Voilà, si les choses se passent mal, vous allez découvrir que votre `drand48` n'est pas si "drand" que ça. Et ce n'est pas à moi d'en tirer les conclusions...

8.1 Scripts

Le script bash `scripts/shoot.sh` est un front-end encore un peu rudimentaire vers le programme de capture d'image décrit page 23. Il utilise deux fichiers dans le répertoire de travail : *reglages* et *compteur*. Le premier est, en fait, un bout de shell affectant quelques variables, ou plutôt, les surchargeant.

Voici un exemple de réglage :

```
OPTIONS="${OPTIONS} -v -c pow2 "
SHOW="yes"
NBRE=1000
PERIOD=0
OFORMAT="p_%04d.png"
```

La première ligne demande, en plus des options par défaut, plus de bavardage, et un changement de contraste. La seconde demande l'affichage de la photo. Les deux suivantes demandent la capture de 1000 images à la cadence méga-blast. La dernière est moins simple : `man sprintf` pour comprendre.

Quand au second fichier, il contient un compteur (stocké en ascii) qui est incrémenté après chaque capture réussie. Et ce compteur est utilisable par la variable `OFORMAT` que nous avons vue quelques lignes plus haut.

```
#!/bin/bash


# change this to point to your installed binary
#
GVS=${HOME}/Devel/FloatImg/v4l2/grabvidseq

#       ------------------------------------
#       set some default values
DEV=/dev/video2
SZ=640x480
NBRE=320
PERIOD=0.0
```

```
COUNT=compteur
OPTIONS="␣-v␣"
SHOW="no"
# output format can be of those types:
#       .pnm .fimg or .png
OFORMAT="P_%04d.pnm"

#               ces parametres peuvent etre surcharges avec
#               un fichier nomme "reglages" dans le repertoire
#               de travail.

#       ------------------------------------
#       overide parameters from $PWD
if [ -r ./reglages ]
then
        source ./reglages
fi

#       ------------------------------------
#       get the current picture number
if [ -r $COUNT ]
then
        numero=$( head -1 $COUNT )
else
        numero=1
fi

#       ------------------------------------
#       make the output filename
if [ 1 -eq $# ]
then
        outfile="$1"
else
        outfile=$( printf ${OFORMAT} $numero )
fi

#       ------------------------------------
#       grab and display the fancy picture
$GVS -d $DEV -n $NBRE -p $PERIOD $OPTIONS -s $SZ -o $outfile

if [ ${SHOW} == "yes" ]
then
        display $outfile &
fi

#       ------------------------------------
#       increment and save the picture number
numero=$(( numero + 1 ))
echo $numero > $COUNT
```

8.2 Fonderie

Ce projet externe [28] est destiné à la confection de films flous à partir de photos floues. Le script `scripts/echomix.sh` est une première expérimentation en bash, utilisant deux outils en CLI, le premier pouvant salement brotcher une image, et le second capable de mélanger harmonieusement

28. ... pour le moment, j'ai des soucis sur l'architecture du **pipdeprod** à adopter...

deux images, la balance est équilibrée.

Il s'agit donc d'un petit programme écrit en Bash, un langage dont la connaissance est, pour moi, indispensable à qui veut faire des images kitchies. Mais ne vous inquiétez pas, c'est en fait assez simple à comprendre. Et comprendre, c'est apprendre.

Voici donc le script, décomposé et expliqué :

```
#!/bin/bash

SRCDIR="Fist"
DSTDIR="Pong"
FTMP="/dev/shm/tmp.fimg"
FDST="/dev/shm/foo.fimg"

# count the nomber of picz in the source directory
NBRE=$(ls -1 ${SRCDIR}/*.fimg | wc -l)
# compute the echo picz offset
OFFS=$(( NBRE / 4 ))
```

Dans ce préliminaire logiciel, nous avons nommé le répertoire SRCDIR contenant les captures d'image au format fimg, le répertoire DSTDIR dans lequel seront rangées les images calculées, et l'emplacement de deux fichiers de travail, placés dans ce qui peut être vu comme un *ramdisk* et qui accélère un peu les opérations.

Les quelques lignes suivantes, qui semble bien magiques, ne sont en fait que de la magie Unix. Elles nous permettent d'avoir NBRE, le nombre d'images à traiter, et OFFS, un décalage dépendant du nombre d'image. Muni de toutes ces informations, nous pouvons rentrer dans le lard du sujet, la boucle qui travaille.

```
#                                                MAIN LOOP
for idx in $(seq 0 $NBRE)
do
        # build the two input filenames ...
        #
        imgA=$(printf "$SRCDIR/%04d.fimg" $idx)
        vb=$(( $(( idx + OFFS )) % NBRE))
        imgB=$(printf "$SRCDIR/%04d.fimg" $vb)

        #        ... and the output filename
        #
        dst=$(printf "%s/%05d.png" ${DSTDIR} $idx)
```

Dans cette première partie de la boucle nous avons construit plusieurs noms de fichier à partir du rang de la boucle en cours d'exécution, des deux valeurs NBRE et OFFS calculées en préambule.

```
        # trying to autocompute the mixing coefficient
        #
        compute=" s(${idx} / 16) "
        K=$(echo $compute | bc -l)
        printf " %25s  =>  %8.3f\n" "$compute" $K
```

Cette seconde partie sert à calculer avec la commande `bc` [29] un coefficient variable en fonction du temps : $sin(idx/16)$ afin d'avoir une oscillation du coefficient entre -1.0 et 1.0, deux valeurs probablement glitchantes.

29. `bc`, c'est vraiment un truc à découvrir.

```
        #       do the hard floating computation
        #
        fimgfx   -v cos010 ${imgB} ${FTMP}
        fimgops  -k ${K} ${FTMP} ${imgA} mix ${FDST}
```

Étape suivante, étape cruciale : le brassage d'une multitude de pixels flottants.

Tout d'abord, nous faisons subir à l'image-echo (`imgB`, définie au début du script) un distorsion chromatique de type *cos010*, le résultat étant écrit dans un fichier temporaire. Ensuite, nous mixons l'image primaire et son echo en utilisant le rapport de mixage calculé quelques lignes plus haut.

```
        #       write the output as PNG for video encoding
        #
        fimg2png ${FDST} ${dst}
done
```

Et en fin de boucle, nous convertissons le résultat de nos savants calculs au format PNG, et nous écrivons le fichier dans le répertoire de destination fixé au début. C'est le moment de passer la main à ffmpeg.

C'est juste une POC, et une implémentation bien plus complète écrite en **C** est déja en chantier, avec une complexité prévue à un niveau assez réjouissant.

9 Video for Linux

Donc, maintenant, nous savons un peu tripoter ces images flottantes. Et nous devons nous poser une question fondamentale [30] sur la provenance de ces données prétendant être des images.

En fait, notre désir secret (enfin, surtout le mien) est la découverte des choses cachées du monde qui nous entoure. Nous voulons des images du **réel** et pour cela, l'outil le plus commun, le plus répandu, est la webcam. L'universelle webcam. Et l'incontournable v4l2.

9.1 grabvidseq

Un logiciel en évolution (vraiment trop) lente, qui permet déja la capture d'images en *longue pose* selon la méthode du cumul, et devrait bientôt retrouver sa capacité à enregistrer des séquences d'images. (*workaround* : écrire une boucle en shell)

```
tth@debian:~/Devel/FloatImg/v4l2$ ./grabvidseq -h
options :
        -c quux         contrast adjustement
        -d /dev/?       select video device
        -g              convert to gray
        -n NNN          how many frames ?
        -O ./           set Output dir
        -o bla          set output filename
        -p NN.N         delay between frames
        -r 90           mode portrait
        -s WxH          size of capture
        -u              try upscaling...
        -v              increase verbosity
        -X arg          Xperiment option
```

30. primitive ?

La plupart de ces options ont un usage quasi-évident. L'option `-s` doit correspondre à une des résolutions possibles de votre capteur. Le type du fichier en sortie (option `-o`) est déterminé par l'extension du nom. Actuellement seulement `.fimg`, `.pnm`, `.fits`, `.tiff` et `.png` sont reconnus.

La conversion en gris (option `-g`) mérite un peu plus de travail, et une paramétrisation plus facile. L'ajustement de contraste (option`-c`) est vaguement expliqué page 9.

L'option `-X` me permet d'intégrer des *fritures* expérimentales dans le binaire, et ne doit donc pas être utilisée dans des scripts si on a des visions à long (ou même moyen) terme.

9.1.1 Upscaling

La fonction que j'ai appelée *upscaling* est un petit hack qui permet de doubler artificiellement la résolution de l'image, en profitant du fait que l'on est capable de prendre N images en rafale.

Pour être rigoureux dans la prise de vue, ce N doit être un multiple de 4, surtout si le nombre de capture est faible. N'hésitez pas à faire des essais, le résultat est parfois aléatoire, surtout avec une caméra qui bouge.

Là, il manque un schéma...

9.2 video-infos

Que contient, que peut faire mon périphérique *àlc* ? Quelles sont ses possibilités de réglage ?

```
tth@debian:~/Devel/FloatImg$ v4l2/video-infos -h
Options :
        -d              select the video device
        -K nnn          set the K parameter
        -l              list video devices
        -T bla          add a title
        -v              increase verbosity
```

Je me sens obligé d'avouer qu'il reste quelques points mystérieux dans l'API de V4L2, et donc, que ce que raconte ce logiciel doit être pris avec des pincettes. En particulier des choses essentielles comme la liste des résolutions disponibles.

9.3 nc-camcontrol

Ajustement *Brightness Contrast Saturation Hue...*

Probablement pilotable au joystick.

10 À l'extérieur

Il existe une foultitude de logiciels (composants ou end-yuser) et il est souvent nécessaire de pouvoir communiquer facilement avec eux. Nous avons déja quelques possibilité d'exportation, mais passer par cette étape intermédiaire est parfois délicat à gérer dans un *pipedeprod* autrement bien huilé.

10.1 ImageMagick

Pour afficher notre format .fimg exotique avec `display`, un des éléments du package ImageMagick, vous devez mettre ce bout de XML dans le fichier `$HOME/.magick/delegates.xml` :

```
<?xml version="1.0" encoding="UTF-8"?>
<delegatemap>
    <delegate decode="fimg" command="fimg2png '%i' '%o'"/>
</delegatemap>
```

C'est juste un hack rapide, qui ne fonctionne pas très bien avec d'autres commande de IM, comme `identify`, qui a tendance à raconter un peu n'importe quoi, puisqu'elle se base sur le résultat de la conversion. Je compte donc sur le bouquin de *Brunus* pour avancer...

10.2 Gimp

Mmmmm... Ça semble un peu plus compliqué. La documentation à ce sujet me semble ésotérique. D'un autre coté, il faut faire ça en **C**, ce qui ne peut pas être totalement négatif. Sauf pour les riiRistes.

10.3 ffmpeg

Un petit aide-mémoire pour encoder vos superbes timelapses :

```
ffmpeg  -nostdin                                \
        -y -r 30 -f image2 -i stereo/S%04d.png  \
        -c:v libx264                            \
        -pix_fmt yuv420p                        \
        -tune film                              \
        -metadata artist='---[ tTh ]---'        \
        -metadata title='awesome video'         \
        stereo.mp4
```

Il existe environ un gazillion d'options pour encoder avec ffmpeg. Bon courage pour toutes les explorer, voire même juste les comprendre.

10.4 Et encore ?

Il y a d'autres logiciels pour lesquels écrire une fonction d'importation serait bien : *Geeqie*, un visualiseur d'image fort pratique, ou *Krita* qui semble avoir les faveurs de dessinateurs de talent. Ce qui nous conduit à une question importante : quels sont les logiciels qui gèrent le chargement d'image par un système de *plugin*, parce que recompiler tout Gimp juste pour ça, c'est un peu overkill.

11 Le flou temporel

Et si nous jouions sur l'axe du temps ?

Nous avons déja plus ou moins la magie du cumul sur la prise de vue d'**image** en enchainant plusieurs captures d'image accumulées en une seule. Maintenant, voyons ce que l'on peut faire à partir de plusieurs images. On peut d'abord penser faire une moyenne (ou la somme, en fait) de toutes ces images. Mais ce n'est qu'une façon déguisée de faire du cumul. C'est à ce moment que nous changeons l'axe de vue du défi.

Par ailleurs, il m'a semblé pertinent d'inclure dans le projet une foultitude d'effets spéciaux. Qui manquent cruellement de possibilités de paramétrage. Mais peuvent être enchainés dans ce

que j'appelle des *filter-chains* : ces effets seront appliqués dans l'ordre aux image traitées soit en entrée soit en sortie. Je pense qu'un schéma [31] s'impose.

11.1 `./fonderie` : *moving average*

Basé sur la moyenne mobile, avec une structure de fifo particulièrement mal conçue. Qui, en fait, n'est pas vraiment un fifo, mais plutôt un buffer circulaire. Pour chaque image source, elle est lue, passe dans la chaine de filtre d'entrée et est insérée dans ce buffer. Ensuite on calcule la moyenne des images contenues dans ce buffer. Et pour conclure, cette image moyennée passe au travers de la chaine de filtre de sortie, avant d'être enregistrée en PNG dans le répertoire de destination.

```
*** fonderie compiled Jan 23 2023, 10:43:56, pid 23366
options:
        -E      input:filter:chain
        -F      output:filter:chain
        -I      input glob pattern
        -L      list available filters
        -O      output directory
        -T      fifo size
        -v      increase verbosity
```

Euh, c'est quoi un '*input glob pattern*' ?

C'est ce qui vous permet de sélectionner un peu les images que vous voulez traiter, exactement comme vous pourriez le faire avec un shell classique. Par exemple pour choisir une image sur dix, le glob-pattern sera '`frames/????0.fimg`', et vous trouverez les explications détaillées dans le manuel : `glob(7)`, et `glob(3)` pour la fonction utilisée.

11.2 Interpolator

Pour le moment, juste des calculs pas si simple que ça. Je pense qu'il faudra se lancer dans des calculs splinesques pour améliorer les choses dans la création des images intermédiaires. Voyons d'abord le principe actuel.

Nous avons une série de centaines, voire de milliers, de photos. En parcourant cette liste, nous allons en prélever une sur N, et entre celle-ci et la précédente prélevée, nous allons calculer par interpolation **linéaire** $N-1$ images intermédiaires, et les intercaler entre nos deux sélections pour générer le flux de sortie, ce qui devrait nous donner un ralenti de bon aloi.

```
usage:
        interpolator [options] <inglob> <outdir> <nbsteep>
options:
        -E i:bla:k      input  filter chain
        -F name:j       output filter chain
        -n              make negative
        -S nn           mysterious sort
        -L              list available filters
        -v              increase verbosity
```

L'option « mysterious sort » mérite bien son nom. Cette friture a été écrite il y a bien longtemps pour un projet précis, et les résultats ont été décevants. Je pense qu'il est toxique de l'utiliser pour le moment [32].

31. omg, apprendre un nouveau truc en LaTeX, quel challenge étourdissant !
32. Ou le repenser différent ?

11.3 Singlepass

Ce programme `singlepass` prétend vous permettre de tester tous les filtres disponibles, et dont on peut avoir la liste avec l'option `-L`[33]

```
usage:
        -F      define:the:filter:chain
        -g      input glob pattern
        -L      list available filters
        -O      /output/directory   (default ./p8)
        -r N    repetiiing factor
        -v      spit more messages
```

Il n'y a pas de moyenne mobile, pas d'interpolation, mais un facteur de répétition qui permet de dupliquer N fois une image dans le flux de sortie. Si vous globez `frames/????[02468]`, vous prenez une image sur deux, alors un facteur de répétition à 2 conservera la 'vitesse' de la séquence, mais avec une petite saccade régulière de bon aloi *:)*

11.4 Déviance dans le flou ?

Là, nous tombons dans de la *troiD* de haut niveau, avec plein de maths compliquées à l'intérieur. Et surtout quelque chose qui n'est encore qu'une idée abstraite, mais il y aura du zbuffer dedans.

12 Expérimentations

Il m'arrive parfois d'avoir de vagues idées d'image, et donc de logiciel, qui tournent dans un coin de la tête. Parfois je songe à une fonction qui me serait bien utile, mais j'ai des doutes sur son API[34] qui soit à la fois simple et complète. Je fais donc des essais. Parfois j'imagine confusément un algorithme tordu et sans but précis. Je le *runne* et je le *re-runne* un gazillion de fois dans mon cerveau processique. Quel va être son facteur O ? Je fais donc des essais.

13 Et pour la suite ?

En fait, je fait de la photo par la méthode du « cumul » depuis plusieurs années. Une webcam, un Linux, et ça *juste marche*. Sauf que c'est quand même un peu galère à déplacer, il faut avoir un shell pour déclencher, c'est pas facile à utiliser en mode portnawak...

L'idée est donc de construire un appareil autonome, basé sur un Raspi et une webcam USB, pilotable par LIRC, alimenté par une (grosse) batterie et permettant d'aller faire des images au bord d'un lac ou dans la campagne de l'Ariège. Et, comme le dit la sagesse populaire : « fimg at 11 ».

33. liste hélas peu machinable.
34. Application Programming Interface

Index

.fimg, 4, 16, 17
/dev/shm/, 22
égalisation, 10
3d, 27

algorithme, 27
alpha, 9
api, 27
ascii, 12
Awk, 17

bash, 20, 22
bc, 22
Betamax, 13
blitter, 10
Brunus, 25
bug, 13, 15

C, 23
concept, 10
contraste, 9
cumul, 7, 9, 18, 23, 27
cumulfimgs, 18

dd2, 16
Debian, 5
Debug, 18
DEBUG_LEVEL, 18
drand48, 13, 20
dynamique, 4

exemple, 4, 14, 19
experimental, 27
export, 12

FAQ, 1
Fedora, 5
ffmpeg, 23, 25
fifo, 26
film, 21
filtrage, 13
fimg2fits, 17
fimg2gray, 18
fimg2png, 17
fimg2pnm, 17
fimg2tiff, 17
fimg_add_rgb, 8
fimgfx, 16
fimgops, 16
fimgstats, 16
FITS, 13
float, 1
FloatImg desc, 7
fonderie, 21
format, 11

géométrie, 10
gamma, 19
gdb, 19
geeqie, 25
Gimp, 17, 25
glitch, 13
grabvidseq, 23

histogramme, 10

identify, 25
ieee754, 4
ImageMagick, 17, 24
interpolator, 13, 26

joystick, 24

kitchy, 22
krita, 17, 25

lib/, 7
libpng, 15
Linux, 27
LIRC, 27

make, 6
Makefile, 18
man, 15
metadata, 11, 12
mkfimg, 15

octet, 1

PAO, 12
photographie, 9
plugin, 25
PNG, 12, 26
png2fimg, 15
PNM, 12
pnm, 19
POC, 23
POSIX, 20
Povray, 18
printf, 20
procrastination, 18

rache, 19

ramdisk, 22
Raspbian, 5
RGB, 4, 18
rgba, 7
rgbz, 7, 9
rotation, 11

scripts, 20
sfx, 13
shell, 6

théorie, 4
TIFF, 12, 19
timelapse, 25
timestamp, 11
TODO, 19
tools/, 5

Unix, 17, 22
upscaling, 24
USB, 27

v4l2, 23
valgrind, 1, 19
vaporware, 12
variance, 16
video-infos, 24

webcam, 23, 27

XML, 24
XXX, 13–16, 19

ymmv, 15

z-buffer, 9
zbuffer, 27