Animagifage (`gif89a`)

tTh

13 mai 2015

1 Introduction

Replaçons d'abord le format de fichier d'image GIF [1] dans son contexte historique. Nous sommes au milieu des années 80, l'épave du Titanic vient d'être retrouvée, le tunnel sous la Manche n'existe pas encore, les meilleurs écrans disponibles sont en 800 x 600, avec 256 couleurs à choisir parmi 262144 [2], Marcia Baila fait danser les gens et les connections au réseau Internet *grand public* ne dépassent pas 28.8 kb/s.

C'est à partir de ces deux contraintes que les opérateurs d'un grand réseau commercial (différent de l'Internet tel qu'on le connait maintenant), CompuServe, a inventé en 1987 le format GIF (dans sa variante `gif87a`), dont le taux de compression, très bon pour l'époque, favorisait une rapide transmission des images de chatons.

Pour ça, il a (hélas ou tant mieux, on ne sait pas trop) fallu faire un choix, une grande concession : limiter le nombre de couleurs à 256, pas une de plus, contrairement aux images de type « photo », où chacune des composantes RGB est codée elle-même sur 256 niveaux, ce qui nous emmène vite à une quasi-infinité de teintes.

En 1989, le besoin de diffuser des *lolcats* qui clignotent se fait vivement ressentir, et apparaît donc la variante `gif89a`, qui supporte *enfin* les animations. C'est, en quelques semaines, un succès mondial. Nous aurons peut-être le temps de voir plus en détail comment c'est implémenté.

2 La philosophie gif89a

C'est une pratique artistique à double facette, un peu (mais pas trop) comme le miroir d'Alice. Deux aspects opposés, deux mondes radicalement

1. Graphics Interchange Format
2. 2**(3*6)

différents, et pourtant complémentaires, deux univers que tout sépare, et qui pourtant sont nécessaires l'un à l'autre, et réciproquement inclus.

La première étape est l'écriture d'un **script**, un petit programme en quelque sorte, qui va enchaîner diverses opérations, et ce, de multiples fois sur une ou plusieurs images. C'est une partie assez technique, surtout pour les néophites en programmation, mais ne vous inquiétez pas, un script simple est compréhensible en quelques minutes, et son utilisation devient vite instinctive. Bien, voilà, votre script fonctionne, et il est temps de passer à ...

... la deuxième étape, maintenant que votre petit esclave bash ronronne sans accroc, qui consiste à lui faire manger une ou plusieurs images. Et c'est là que votre fibre artistique devra se mettre en route dans le cadre des restrictions techniques mises en place au paragraphe précédent. C'est à ce moment que vous allez (re-)découvrir l'importance capitale de choses comme la lumière, le cadrage, la profondeur de champ, et surtout la balance des couleurs.

3 Aspects techniques

Oui, vous allez apprendre à programmer dans un langage étrange, et truffé de pièges sournois . Non, ce n'est pas compliqué, et nous allons plutôt procéder pas à pas selon une méthode éprouvée : l'expérimentation active [3] sur un sujet bien précis.

Ensuite, et je suis désolé de vous l'annoncer si tardivement, tout ce que je vais vous expliquer ne fonctionnera qu'avec un système d'exploitation décent, c'est à dire en gros du Linux. Songez donc à avoir une machine supportable.

Machine dans laquelle vous aurez disponible un certain nombre de logiciels indispensables, voire nécessaires, ou seulement complémentaires.

- l'interpréteur de script shell `bash`.
- le package `ImageMagick`.
- un éditeur de texte décent.
- une chaîne de compilation `Gcc` [4].
- un utilitaire de capture, genre `vgrabbj`.
- un interpréteur `awk`, parce que c'est fun.

Il existe plein d'autres logiciels plus classiques, comme Gimp ou Krita, ou carrément exotiques comme xpaint, potrace, netpbm ou GMIc. Et comme le dit la sagesse populaire, il y a toujours plusieurs manières de faire la même chose.

3. aka *ShotGun Programming*
4. Facultatif

4 Premiers exemples

4.1 À partir d'une seule image.

Il nous suffira de faire varier un (ou plusieurs) paramêtre du ou des traitements. Nous allons rester simple, sans toutefois aller jusqu'au simplisme. Faire clignoter le ying et le yang, le sombre et le clair, le jour et la nuit. Bref, alterner une image et son négatif.

```
SOURCE=image.png
convert -negate $SOURCE foo.png
convert -delay 100 $SOURCE foo.png yingyang.gif
```

Et ce petit début d'animagifage s'utilise comme ça, depuis votre xterm favori :

```
$ bash premier.sh
$ animate yingyang.gif
```

Et voilà, ouéééé ! Vous êtes devenus des animagifeurs bashistes. Maintenant, nous allons rendre l'utilisation de ce script plus simple afin qu'il puisse être ré-utilisé sans avoir à modifier à la main les noms de fichier ; bref, que vous puissiez taper au prompt quelque chose comme ceci :

```
$ ./fabrique.sh image.png yingyang.gif
```

Fascinant, n'est-il pas ? Donc petite plongée dans les arcanes de bash, la ligne de commande et les paramètres positionnels. Démonstration par l'exemple :

```
$ cat params.sh
#!/bin/bash
echo $# $1 $2 $3
$ chmod u+x params.sh
$ ./params.sh
0
$ ./params.sh foo
1 foo
$ ./params.sh foo bar
2 foo bar
```

Oké, je crois que c'est clair pour tout le monde, et bien suffisant pour le moment. Mais il existe quelques subtilités [5] que nous découvrirons plus tard. Mettons vite en œuvre cette recette.

5. man 1 getopt

Voici le premier script modifié :

```
#!/bin/bash
SOURCE=$1
convert -negate $SOURCE foo.png
convert -delay 100 $SOURCE foo.png $2
```

Et tout devient plus facile, puisque, ô joie, les scripts shells peuvent s'enchaîner facilement, à l'aide, ô surprise, d'un script shell. Pouvoir traiter plusieurs image à la file est un plus non négligeable pour votre pipdeprod.

4.2 À partir d'une séquence.

On pourra aussi faire varier un (ou plusieurs) paramètre d'un des traitements sur une série d'images. Mais pour commencer, nous allons essayer de rester dans un esprit assez rudimentaire. Une simple boucle qui va négativer une séquence d'image, puis, en une seule gif, les lier toutes.

Voici le prérequis de base : avoir N images numérotées séquentiellement d'une façon numérique, et non pas dans le classement du dictionnaire. C'est à dire des numéros cadrés fixes avec les zéros de gauche : `08 09 10 11` à la place de `8 9 10 11`. C'est comme ça et pas autrement. Désolé.

La commande POSIX `printf` sera là pour nous aider à gérer une partie de ce cas de figure en formatant correctement nos noms de fichiers. Voyons tout ça sur un fragment de script qui permet de capturer cette fameuse séquence d'image :

```
#!/bin/bash

NOMBRE=60
DEVICE=/dev/video1
GRABOPT="-S␣-o␣png␣-i␣qsif␣-d␣$DEVICE"

for frame in $(seq 1 $NOMBRE)
do
        ecran=$(printf "/tmp/S%03d.png" $frame)
        vgrabbj $GRABOPT > $ecran
        sleep $ATTENTE
done
```

Après l'exécution de cette séquence de prise de vues, nous allons trouver dans notre `/tmp` 60 images allant de `S001.png` à `S060.png`, et c'est sur cette séquence que nous allons répéter nos traitement.

```
#!/bin/bash
```

```
NOMBRE=60
ATTENTE=1

for frame in $(seq 1 $NOMBRE)
do
        ecran=$(printf "/tmp/S%03d.png" $frame)
        convert -resize 800x100 $ecran $ecran
done
convert -delay 10 /tmp/S*.png kikoo.gif
```

5 Prises de vue

C'est la seconde facette de l'art des GIFs animées : la prise de vue. Cadrage, éclairage, arrière-plan, tout est important, comme en photographie classique. Mais nous avons ici une contrainte supplémentaire : l'obligatoire réduction du nombre de couleur.

C'est à cause de cette contrainte qu'il faut essayer d'éviter une trop grand diversité de teintes dans la scène, afin que les algorithmes de réduction du nombre de couleur puissent bien fonctionner. Par exemple, si vous n'avez que du vert, le *quantizer* pourra, dans cette petite palette de 256 couleurs, en construire assez pour que vos délicats dégradés de vert ne soient pas trop dégradés.

6 La partie gore

Les formats de fichier bitmaps[6]. Une belle angoisse. Il en existe un gazillion, chacun ayant sa dose d'avantages et son avalanche de défauts.

7 Conclusion

Voilà, j'espère que vous vous êtes bien amusés en apprenant à faire ces belles images qui clignotent.

6. carte de bits