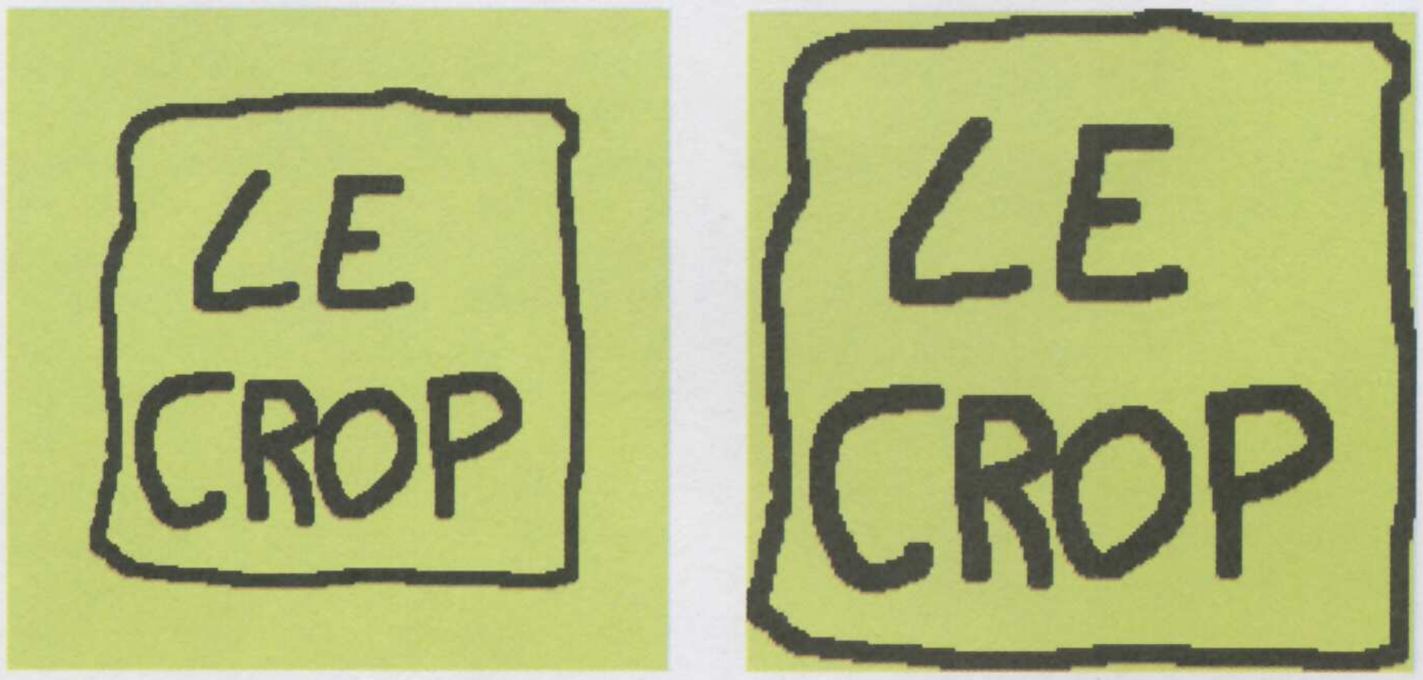
Traitement par lot et opérations non interactives avec ImageMagick

Certains logiciels permettent de réaliser de la retouche d'images, simplement et efficacement. Mais lorsqu'il devient nécessaire de retoucher un grand nombre d'images, des solutions plus adaptées s'imposent.



Elimination du contour grâce à convert.

Les interfaces graphiques bien pensées sont en effet idéales pour réaliser le traitement d'une seule image, ou éventuellement d'un petit nombre. En revanche, il s'avère fastidieux d'effectuer par ce moyen des opérations répétitives sur un grand nombre de fichiers.

Les méthodes explicitées ici sont issues d'un cas concret, en l'occurrence l'archivage d'un grand nombre d'images en provenance d'un scanner, pour lesquelles il convenait de supprimer les marges inutiles. La première solution, rapidement abandonnée, fut d'ouvrir et de modifier les images une à une. La seconde envisagée consista en l'utilisation des scripts *Photoshop*, mais d'une part, ceux-ci se montrent délicats à réaliser, et d'autre part, aussi intéressant soit-il, *Photoshop* n'incarne pas le produit le plus économique pour résoudre ce genre de problème. Enfin, et surtout, même si la version *5* sait désormais lire les fichiers TIFF compressés dans le mode CCITT Group IV, il ne permet en aucun cas d'enregistrer dans ce mode, ce qui le rend inutilisable dans le cas présent.

La solution à ce problème réside dans *ImageMagick* et ses différents utilitaires. En survolant quelque peu le manuel de la commande convert, il s'avère que ce programme a effectivement quelque chose de 'Magick'.

Voici un premier script rapidement écrit :

```
#!/bin/bash
for fic in *.tif
do
echo -n Cropping $fic ...
            /usr/X11R6/bin/convert
-crop 0x0 $fic cropped_$fic
            echo OK
done
```

Au cœur de celui-ci se trouve la fameuse commande convert, avec pour principal paramètre, l'action à effectuer ; ici, la suppression des bordures ne contient que la couleur de fond sur la plus grande largeur à chaque fois. Autrement dit, on enlève le cadre blanc, qui n'offre aucun intérêt.

En outre, convert est un programme pourvu de bon sens puisque, sauvant par défaut dans le même format que celui de l'image source, il opte automatiquement pour la méthode de compression la plus appropriée. Ainsi, avec une image fournie en noir et blanc et compressée en LZW, le résultat a été quant à lui compressé en CCITT Group IV, ce qui se révèle assurément plus économique.

Bien entendu, le mieux étant l'ennemi du bien, comme la gestion de cet algorithme de compression ne se montre pas très répandue dans les programmes manipulant le format TIFF, son emploi peut desservir l'utilisateur. MAIS il existe une parade qui se limite malheureusement à interdire purement et simplement toute compression avec l'option +compress, qu'il suffirait de placer avant l'option -crop.

Incidemment, et comme si cela ne suffisait pas, convert règle un problème apparaissant avec certains autres logiciels. Il s'agit de ceux qui ont la mauvaise idée de coder le paramètre 'RowsPerStrip' des images sauvées au format TIFF avec la valeur 'Infinite'. Or, si cela est autorisé, il existe au moins un programme du commerce, spécialisé dans la consultation d'archives au format TIFF, qui n'aime pas du tout cette valeur, et qui figure bien évidemment parmi les plus fréquemment utilisés. La bonne nouvelle vient de ce que convert rectifie automatiquement le tir en donnant à ce paramètre la valeur souhaitée.

Mogrify

Les traitements de groupes de fichiers peuvent se voir réalisés à l'aide de la commande mogrify, comme dans l'exemple suivant :

```
mogrify -crop 0x0 image_{1,2}.tiff
```

La différence majeure avec convert résulte dans le remplacement du contenu des fichiers d'origine par le résultat du traitement dont ils font l'objet.

Le principal avantage avec cette commande réside dans l'inutilité de scripts comme celui présenté plus haut, ce qui implique une prise en main plus simple et plus rapide pour tous ceux qui ne souhaitent pas se plonger dans les méandres des shells.

Toutefois il subsiste quand même certaines petites (ou grandes) prérogatives des scripts, surtout avec des programmes aussi peu loquaces (ou au contraire trop verbeux). L'une d'entre elles consiste à pouvoir afficher la progression de la tâche demandée, tout particulièrement si elle est longue. Il s'avère toujours rassurant de savoir qu'un blocage n'a pas eu lieu lorsque l'on se voit informé,

GIMP vs. Photoshop!

Loin de les mettre en opposition comme pourrait le laisser croire le titre de cet encadré, une différence de taille mérite d'être soulignée, car elle pourrait inciter certains utilisateurs de Photoshop à se tourner vers son outsider. Elle concerne la limitation arbitraire qu'impose ce programme, avec des images dont les dimensions ne peuvent excéder très exactement 30 000 pixels de côté. Et cela reste vrai dans la version 5 qui vient de sortir.

Un tel chiffre peut paraître démesurément grand et hors de portée mais quand vous êtes amenés à utiliser des scanners à tambour format A0 avec une finesse pouvant atteindre les 1200 dpi, vous mesurez rapidement que les limites évoquées ne tiennent plus du tout du fantasme.

GIMP, et vous pouvez faire le test, accepte parfaitement de créer des images supérieures à 65000 pixels. Ce qui n'a rien de délirant, puisque des formats comme le TIFF acceptent, depuis la nuit des temps, de telles valeurs.

fichier après fichier, que le traitement continue.
Le premier exemple évoqué plus haut avec mogrify est intéressant, car il va nous permettre de bien comprendre la différence de comportement qu'offre ce programme par rapport à convert.
Une commande comme

```
convert -crop 0x0 image_{1,2}.tiff
```

est traduite par le shell en

```
convert -crop 0x0 image_1.tiff image_2.tiff
```

ce qui signifie dans la syntaxe comprise par convert "opère le traitement demandé sur le contenu du fichier image_1.tiff et enregistre le résultat dans le fichier image_2.tiff ".
À l'opposé, les mêmes paramètres fournis à mogrify dans la commande

```
mogrify -crop 0x0 image_{1,2}.tiff
```

sont évidemment traduits de la même manière par le shell en

```
mogrify -crop 0x0 image_1.tiff image_2.tiff
```

MAIS se trouvent transformés par ce programme en "opère un traitement sur le contenu du fichier image_1.tiff et enregistre le résultat dans le fichier image_1.tiff, PUIS, opère le traitement demandé sur le contenu du fichier image_2.tiff et enregistre le résultat dans ce fichier image_2.tiff ".
Ce qui, vous en conviendrez, ne revient pas du tout au même.

Pour finir

Remarque en passant, à la suite de tentatives avortées, des fichiers temporaires engendrés par convert ou mogrify peuvent polluer votre répertoire /tmp, portant des noms préfixés avec 'magicka', et se terminant par un numéro qui correspond de toute évidence au PID, auquel était associé le programme en cours d'exécution avant de se voir sauvagement interrompu. En d'autres termes, si ces fichiers se multiplient et que la fréquence de nettoyage de ce dossier ne se révèle pas assez importante, vous risquez rapidement de ne plus avoir de place sur votre filesystem.
Conseil : avant de vous lancer avec d'imposants fichiers, faites des tests à l'aide des documents de taille très raisonnable, de façon à ne pas être surpris ensuite par les durées des différents traitements et d'éviter ainsi de vous ronger les ongles devant un ordinateur en apparence inerte.

Pour conclure, le rôle de convert ne se cantonne pas aux nettoyages d'images ou aux découpages ; il inclut aussi la création d'animations, au format GIF par exemple, ou la réalisation de planches contact, en utilisant plusieurs images comme sources. Il est aussi capable d'extraire une image isolée ou une séquence au sein d'une animation.
Le principal reproche que l'on pourrait faire à ces outils (convert ou mogrify), outre qu'ils ne préparent toujours pas le café, provient de leur mutisme, que bien des espions soumis à la torture leurs envieraient.
S'il existe des programmes qui vous racontent leur vie pour un oui ou pour un non, le moins qu'on puisse dire pour convert et mogrify, c'est qu'ils ne permettent pas de comprendre leur fonctionnement ni la syntaxe qu'ils attendent par une suite d'essais et d'échecs.
La réalité s'avère un peu plus nuancée, car chacun de ces programmes dispose d'une option verbose. Mais, d'une part, elle n'apporte aucune aide en cas de syntaxe erronée et d'autre part, elle illustre parfaitement le proverbe "le mieux est l'ennemi du bien", car elle affiche les caractéristiques détaillées de chaque document, à l'image de ce que peut afficher la commande tiffinfo lorsqu'il s'agit d'un fichier au format TIFF.
De surcroît, l'option verbose provoque l'affichage d'informations, non pas sur la sortie standard mais sur celle des erreurs, ce qui ne facilite pas le filtrage des messages dans l'optique de se focaliser aux seuls représentatifs de la progression des traitements demandés.
Vous pourrez trouver *ImageMagick* à l'adresse :
ftp://ftp.wizards.dupont.com/pub/ImageMagick/ImageMagick.X.tar.gz (où X correspond à la numérotation de la dernière version disponible).

Yannick Cadin
Yannick@kommando.com

Application du traitement par lots à un script CGI

Après avoir abordé le mois dernier le traitement automatique d'images, nous allons aujourd'hui utiliser ces fonctionnalités pour permettre à un internaute de consulter une carte de grand format.

La vignette de visualisation.

Qui dit grand format, dit image de taille volumineuse, impossible à télécharger, même sur un réseau 100 Mbits, d'autant que les navigateurs n'acceptent d'afficher que des images dont les dimensions n'excèdent pas, en règle générale, les 8000 pixels de côté (nous sommes loin des 16 000 pixels autorisés dans une image GIF). D'autre part, réduire l'image dans le but de la transmettre la rendrait illisible, donc inutilisable. Dans cette situation, *ImageMagick* va s'avérer d'un grand secours.
Un programme comme *convert* offre quelques avantages déterminants dans la réalisation de ce type d'application. Il peut notamment s'exécuter en tâche de fond sans requérir aucune interface graphique, ce qui implique une belle économie de ressources en terme de charge CPU. En outre, la présence de traitements directement exploitables dispense d'une quelconque programmation et de connaissances intrinsèques des formats graphiques ou de leur manipulation. En conséquence, il permet d'effectuer un prototypage aisé, et pourquoi pas, une exploitation en conditions réelles si les temps de réponse s'avèrent satisfaisants.

Etape 1 : convertir l'image originale pour obtenir un fichier au format RGB
En admettant que l'image originale porte le nom 'carte.tif', la commande idoine sera ;

```
convert carte.tif rgb:carte.rgb
```

Cette étape se révèle indispensable, car *convert* n'autorise le découpage de portion d'image que si le fichier source est codé dans le format RGB. A noter tout de même qu'une image RGB non compressée de 65536 pixels de côté occupe 24 Go (et pas seulement 12, car chaque composante rouge, verte et bleue de chaque point est codée sur 2 octets).

Etape 2 : faire une vignette (une représentation réduite) qui servira de point de départ à l'internaute
Prenez connaissance des dimensions de votre image d'origine (à l'aide de tiffinfo par exemple, si elle se trouve au format TIFF) et déterminez, uniquement sur la base de votre bon sens, les dimensions que devra avoir la vignette. Lancez ensuite une commande du genre :

```
convert -geometry
30%x30% carte.tif
gif:carte.gif
```

ou, si vous préférez spécifier les valeurs en pixels (notre modèle fait 685 par 925 pixels) :

```
convert -geometry
205x277 carte.tif
gif:carte.gif
```

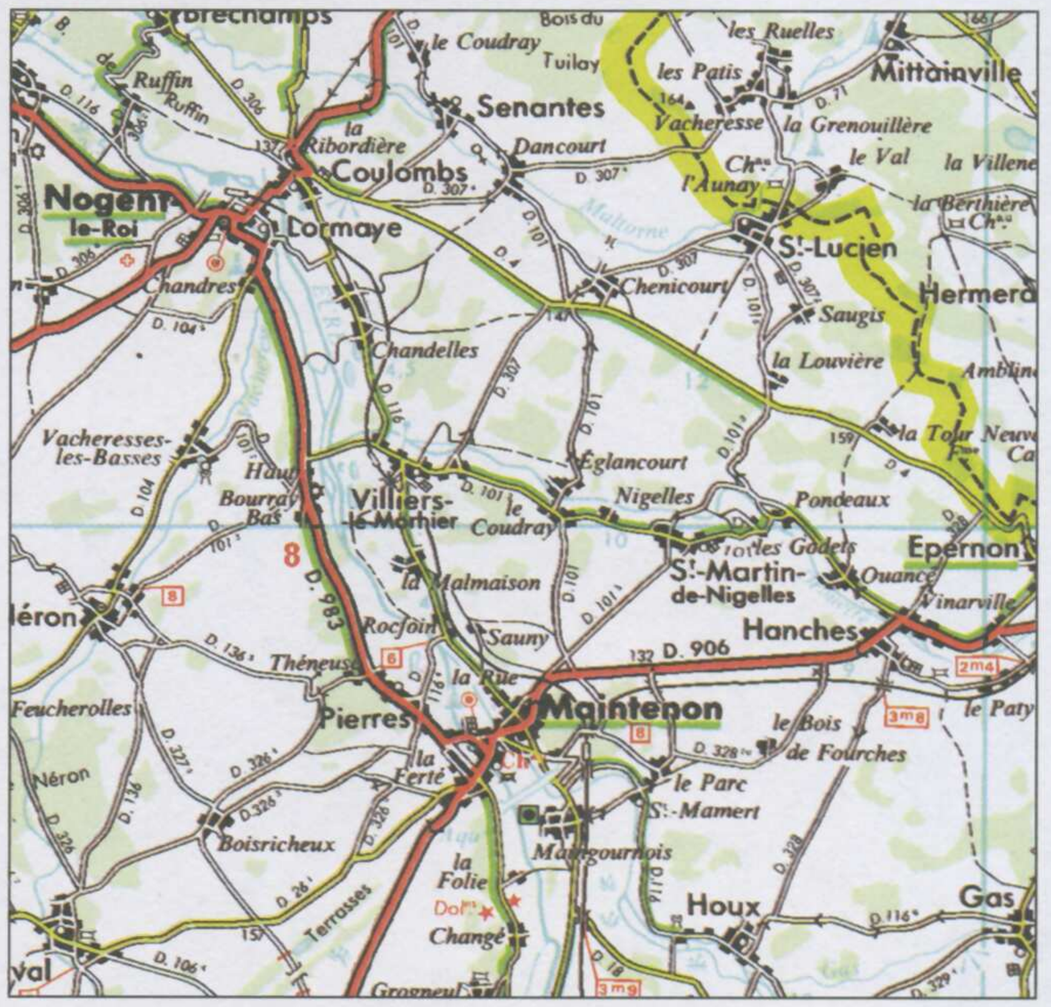
Bien entendu, les deux premières phases peuvent parfaitement être réalisées avec n'importe quel programme, interactif ou non, permettant des manipulations équivalentes.
A l'inverse, on peut vouloir calculer ces valeurs si l'on souhaite traiter automatiquement de la sorte plusieurs cartes de dimensions différentes. Il demeure relativement facile de le faire à l'aide de langages tels que Perl ou TCL en filtrant les informations rapportées par la commande tiffinfo ; l'opération se montre plus délicate à effectuer avec un shell traditionnel, mais n'a pour autant rien d'impossible. Voici comment procéder :
Récupération de la largeur et de la hauteur :

```
[prompt_shell]% largeur=`tiffinfo
carte.tif |grep 'Image Width' |awk
'{ print $3 }'`
[prompt_shell]% hauteur=`tiffinfo
carte.tif |grep 'Image Width' |awk
'{ print $6 }'`
```

La présente syntaxe fonctionne en particulier pour le shell Bash ; en ce qui concerne les autres shells, il faudra éventuellement adapter.
D'autre part, les références au format TIFF pour l'image originale étant omniprésentes dans ce qui précède, il peut s'avérer préférable de convertir dès le départ son image dans ce format à l'aide de la commande :

```
convert
carte.EXTENSION_FORMAT_QUELCONQUE
tiff:carte.tif
```

Ensuite, on choisit la largeur et la hauteur que devra avoir chaque vignette. En toute logique, on choisira une largeur constante, ce qui semble relativement approprié dans



Notre carte aux dimensions réelles.

le cas d'une exploitation avec une interface Web. Prenons donc 250 pixels de large. Le calcul de la hauteur correspondante ressemblera alors à :

```
[prompt_shell]%
hauteur_apres_reduction=`expr
250000 / $largeur \* $hauteur /
1000`
```

avec pas moins de quatre remarques cette fois. S'il peut sembler ridicule d'utiliser la valeur 250000 pour, au final, diviser le tout par 1000, il faut bien garder à l'esprit que la commande expr travaille essentiellement sur des valeurs entières et que les calculs avec des nombres arrondis donnent des résultats par trop approximatifs. De surcroît, il serait assez hasardeux d'essayer d'introduire des parenthèses dans une expression soumise à expr, car si celle-ci ne bronche pas, les résultats se révèlent pour le moins déroutants.
Le second point porte sur la présence d'une barre de fraction inverse juste devant l'astérisque, ceci dans le but d'empêcher le shell de substituer à cette astérisque la liste des fichiers contenus dans le répertoire courant.
En troisième lieu, prenez soin d'insérer au moins un espace après chaque opérande et opérateur, car la syntaxe d'expr l'exige.
Enfin, les apostrophes sont ici de type 'inverse'.
En conclusion, après ces différentes commandes, la création de la vignette ressemblera à :

```
convert -geometry
250x$hauteur_apres_reduction
carte.tif gif:carte.gif
```

Il est possible que la vignette fasse un pixel de moins en largeur que la taille demandée, car les résultats fournis par expr se voient systématiquement arrondis par défaut et parce que convert conserve les proportions de l'image.

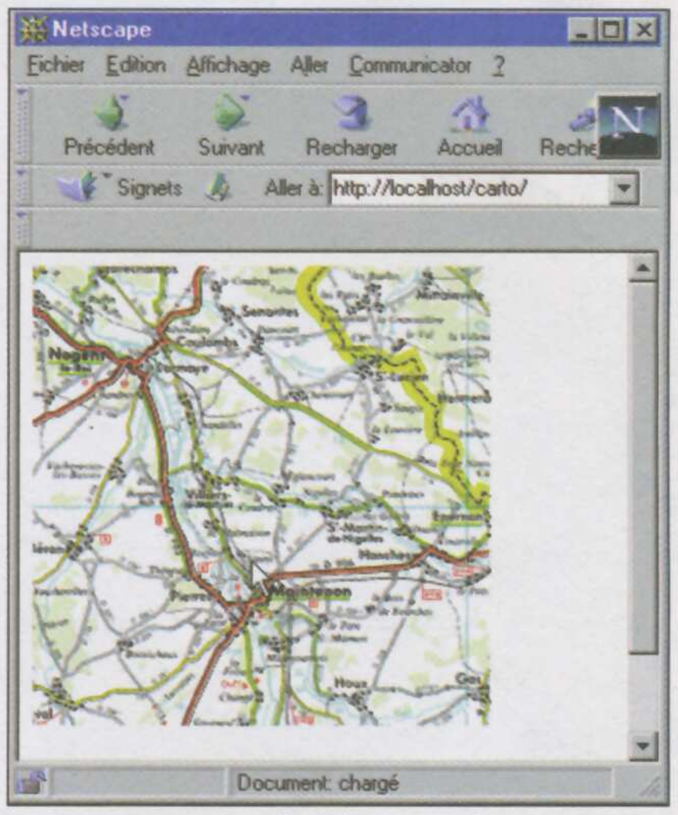
Étape 3 : rédaction du script

Le script affichera une portion découpée depuis l'image originale au format RGB, en fonction des coordonnées du point sélectionné par l'utilisateur, sur la représentation réduite (la vignette), comme s'il s'agissait d'un zoom.
Le fichier HTML qui va présenter la vignette à l'utilisateur devra comporter les éléments suivants :

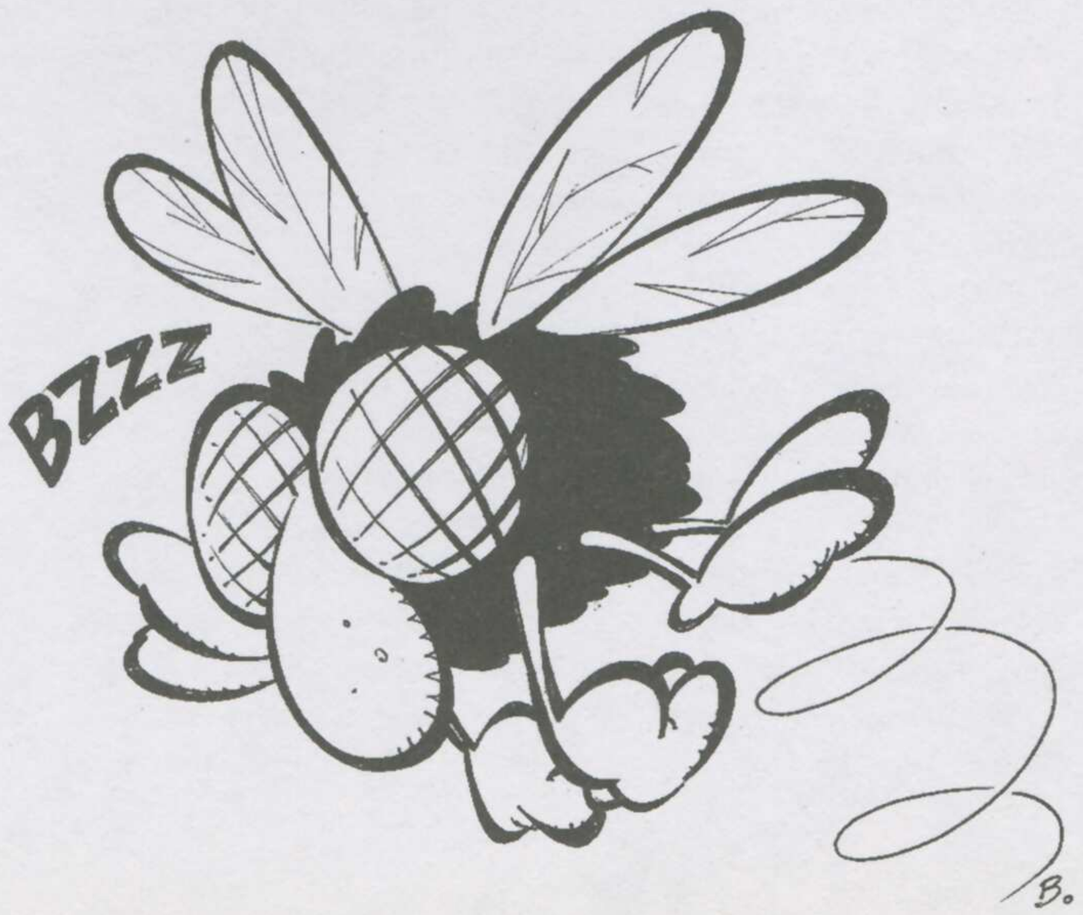
```
<FORM ACTION="/cgi-bin/zoom"
METHOD="GET">
<INPUT TYPE="image" SRC="carte.gif"
NAME="point_selectionne" BORDER="0">
</FORM>
```

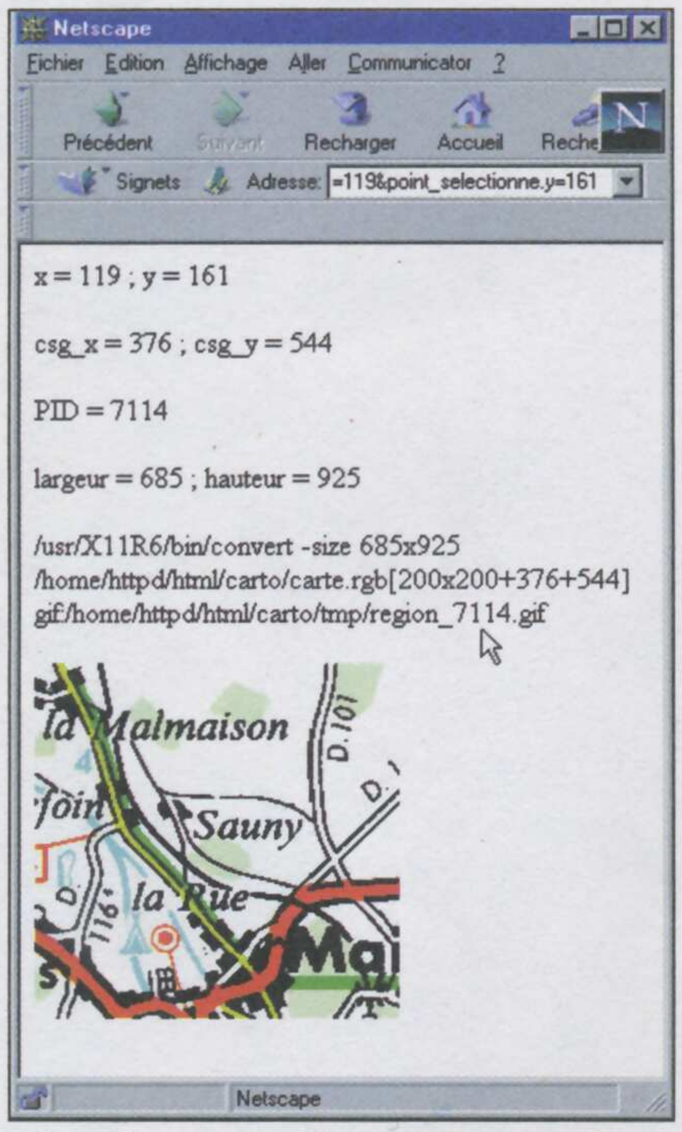
Ces trois directives sont pleinement suffisantes pour expérimenter notre petite application.
Il faut copier le fichier carte.gif dans le même répertoire que le fichier HTML ci-dessus, en l'occurrence le dossier /home/httpd/html/carto, créé pour l'occasion.
Ci-après se trouve le contenu du fichier nommé 'zoom' dans le répertoire des scripts CGI (tel que défini dans le fichier de configuration de votre serveur Web, /home/httpd/cgi-bin par défaut pour apache 1.2.6 sur Linux Red Hat 5.1).
Première version :

```
1 #!/bin/bash
2
3 largeur_vignette='250'
4
5 largeur=`/usr/bin/tiffinfo
/home/httpd/html/carto/carte.tif
|grep 'Image Width' |awk '{ print
$3 }'`
6 hauteur=`/usr/bin/tiffinfo
/home/httpd/html/carto/carte.tif
|grep 'Image Width' |awk '{ print
$6 }'`
7
8 facteur=`expr $largeur \* 1000 /
$largeur_vignette`
9
10 x=`echo $QUERY_STRING | sed -e
's/point_selectionne\.x=//;s/&point
_selectionne\.y=[0-9]*//'`
11 y=`echo $QUERY_STRING | sed -e
's/point_selectionne\.x=[0-
9]*&point_selectionne\.y=//'`
12
13 csg_x=`expr $x \* $facteur /
1000 - 100`
14 csg_y=`expr $y \* $facteur /
1000 - 100`
15
16 /usr/X11R6/bin/convert -size
${largeur}x$hauteur
/home/httpd/html/carto/carte.rgb[200x
200+$csg_x+$csg_y]
gif:/home/httpd/html/carto/tmp/regi
on_$$.gif
17
18
19 echo Content-type: text/html
20 echo
21
22 echo "x = $x ; y = $y<P>"
23 echo "csg_x = $csg_x ; csg_y =
$csg_y<P>"
24
25 echo "PID = $$<P>"
26
27 echo "largeur = $largeur ; hauteur =
$hauteur<P>"
28
29 echo "/usr/X11R6/bin/convert -size
${largeur}x$hauteur
/home/httpd/html/carto/carte.rgb[200x
200+$csg_x+$csg_y]
gif:/home/httpd/html/carto/tmp/regi
on_$$.gif<P>"
30
31 echo '<IMG
```



Premier script de l'étape 3 (la vignette dans un navigateur SANS menu gamma).





Second script de l'étape 3 (le zoom dans un navigateur SANS légende et AVEC 'debug').

```
SRC="/carto/tmp/region_'$$'.gif">'
```

Note : les numéros en regard de chaque ligne sont uniquement présents pour des raisons de commodité et ne doivent pas apparaître dans votre script.
Ligne 3, la largeur de la vignette étant totalement arbitraire, il nous faut bien la définir explicitement. Ligne 8, le point que va sélectionner l'internaute se verra exprimé par rapport aux dimensions de la vignette ; or, il nous faut des valeurs données par rapport aux dimensions de l'image originale pour pouvoir découper une région à l'intérieur. La variable 'facteur' va nous permettre de les calculer. Lignes 10 et 11, à l'aide de sed, on extrait de la variable d'environnement QUERY_STRING, créée par le serveur Web pour le script CGI, les valeurs x et y du point sélectionné (en fonction de la directive INPUT de notre document HTML, la variable QUERY_STRING correspond à une chaîne, qui ressemblera à :
point_selectionne.x=123&point_selectionne.y=456). Lignes 13 et 14, on calcule les coordonnées du coin supérieur gauche de la région que l'on va découper en choisissant arbitrairement une portion carrée de 200 pixels de côté.
Ligne 16, on procède au découpage à proprement parler. Trois choses sont à noter. En premier lieu, intéressons-nous à l'utilisation du chemin d'accès complet pour chercher le programme *convert*. Celle-ci donne le moyen de pallier les aléas de la valeur de la variable d'environnement PATH communiquée au script (remarquez au passage le manque cruel de rigueur, puisque les commandes sed et expr ne bénéficient pas du même traitement ; quant à echo, il s'agit d'une commande interne du shell qui ne peut donc pas se voir attribuée de chemin d'accès). Petit rappel, la commande 'which' peut vous permettre de connaître le chemin d'accès d'un programme, simplement en lui fournissant comme paramètre le nom de celui-ci, comme dans 'which convert'. D'autre part, concentrons-nous sur l'utilisation de la variable spéciale $$. Il s'agit d'un vieux truc assez simple et suffisamment efficace lorsque l'on veut créer des fichiers temporaires sur un système multitâche/multi-utilisateurs. De fait, on ne saurait se permettre de choisir un nom de fichier figé, car plusieurs personnes peuvent exécuter le même programme (ce script en l'occurrence) presque simultanément ; il en résulte que le fichier temporaire est uniquement valide pour le dernier des utilisateurs à l'avoir sollicité. Un identificateur unique est le PID (Process IDentification), qui se trouve remis à 0 lors uniquement du reboot. L'accès à cette valeur dans un shell s'opère par le biais de la variable $$, ce qui explique qu'on l'utilise dans le cas présent.
Enfin, le troisième point important consiste dans l'usage des accolades comme délimiteurs du nom de variable 'largeur'. Leur présence s'explique par la lettre x suivant directement le nom de la variable. Comme la syntaxe de convert n'autorise pas d'espace, les accolades permettent de faire comprendre au shell que la variable en question est bien 'largeur' et non pas 'largeurx'.
Afin de ne pas tout mélanger, les fichiers temporaires sont entreposés dans un sous-répertoire créé spécialement et baptisé 'tmp' dans notre dossier 'carto'. Il ne faut pas oublier de donner le droit d'écriture à tout le monde pour le répertoire 'tmp' car, par défaut, l'utilisateur associé à l'exécution d'un script CGI est le pseudo-utilisateur 'nobody' et, s'il n'a pas le droit d'écrire dans 'tmp', aucune image temporaire ne pourra se créer.
Aux lignes 19 et 20 correspond l'envoi au navigateur Web de l'en-tête du document HTML que nous sommes en train de créer dynamiquement. Le deuxième echo sans paramètre se révèle extrêmement important : il provoque l'envoi d'un second retour chariot, qui est identifié par le navigateur Web comme le séparateur entre la zone des en-têtes arrivée en premier et la zone des données qui va suivre. Lignes 22 à 29 se déploie l'affichage de toutes les variables nécessaires à ce script, utile uniquement pendant la phase d'élaboration. Sa présence ici sert à montrer qu'il s'agit encore de la méthode la plus simple et la plus pratique de vérifier la cohérence des données tout au long de la conception. Ainsi, la façon dont le shell interprète les paramètres communiqués à une commande, pour mettre en évidence des pièges comme celui qui impose de faire usage d'accolades, va même jusqu'à s'afficher. La ligne 31 se consacre à la directive HTML pour l'affichage de la portion d'image découpée ; elle nécessite seulement deux remarques. D'une part, il convient de bien faire attention aux chemins d'accès mis en œuvre ; en effet, nous sommes contraints de mélanger des références propres au système de fichiers du système d'exploitation et d'autres spécifiques au classement interne du serveur Web (de plus, il faut prendre garde à l'emploi d'un chemin d'accès relatif, le répertoire où s'exécute le script n'ayant rien à voir avec celui où se trouve le fichier HTML qui y fait référence). D'autre part, focalisons-nous sur l'emploi qui est fait des apostrophes comme délimiteurs. La difficulté ici consiste à éviter l'interprétation des signes supérieur et inférieur ; il y aurait possibilité d'utiliser des guillemets ou des apostrophes mais, puisque l'on veut aussi incorporer des guillemets dans la directive HTML, on utilise alors les apostrophes comme délimiteurs. Mais, comme les apostrophes, à l'inverse des guillemets, annulent l'évaluation des variables introduites par le caractère $, on place celles-ci en dehors des guillemets. En résumé, la règle à retenir est que les apostrophes évitent l'interprétation de caractères jugés spéciaux par le shell, sauf pour les variables avec $ comme préfixe. Les apostrophes vont plus loin en inhibant aussi l'interprétation de ce caractère $. Pour les programmeurs Perl, cela s'avère d'autant plus facile à retenir que ces règles demeurent identiques dans ce langage.
Les deux principaux défauts de cette première version sont l'absence de traitement approprié pour les découpages débordant de l'image originale et l'utilisation de fichiers temporaires.
En effet, si le morceau à découper déborde de l'image source, le résultat n'est pas automatiquement tronqué. Un débordement en largeur crée une bande latérale sur la nouvelle image générée. En hauteur, le programme *convert* ne rend tout simplement pas la main.

Étape 3b : amélioration du script

Ce nouveau script a pour but de recentrer l'image découpée en fonction des limites de l'image originale, de façon à éviter des bordures latérales disgracieuses et des blocages,

Etape 3e (zoom dans un navigateur AVEC légende et SANS 'debug').

pour les découpes susceptibles de dépasser en haut ou en bas. Par conséquent, il est nécessaire d'introduire des calculs supplémentaires, qui vont nécessiter l'utilisation d'un langage comme Perl, plus à même d'effectuer ces calculs que la commande 'expr'. Vous trouverez ci-dessous la traduction du script en Perl, avec une légère amélioration consistant donc à recadrer la portion découpée, lignes 21 et 22, afin qu'elle soit toujours dans les strictes limites de l'image originale, et pour s'assurer de ne pas provoquer de blocage.

```
1 #!/usr/bin/perl
2
3 $largeur_vignette = 250;
4
5 open (TIFFINFO, '/usr/bin/tiffinfo
/home/httpd/html/carto/carte.tif |');
6 while (<TIFFINFO>)
7 {
8         if (/Image Width: (\d+)
Image Length: (\d+)/)
9    {
10      $largeur = $1;
11      $hauteur = $2;
12      last;
13   }
14 }
15 close (TIFFINFO);
16
17 $facteur = $largeur /
$largeur_vignette;
18
19 ($x, $y) = $ENV{'QUERY_STRING'}
=~
/point_selectionne\.x=(\d+)&point_
selectionne\.y=(\d+)/;
20
21 $csg_x = min (max (0, int ($x *
$facteur - 100)), $largeur - 200);
22 $csg_y = min (max (0, int ($y *
$facteur - 100)), $hauteur - 200);;
23
24 system ("/usr/X11R6/bin/convert
-size ${largeur}x$hauteur
/home/httpd/html/carto/carte.rgb[200x
200+$csg_x+$csg_y]
gif:/home/httpd/html/carto/tmp/regi
on_$$.gif");
25
26 print <<"FIN_HTML";
27 Content-type: text/html
28
29 x = $x ; y = $y<P>
30 csg_x = $csg_x ; csg_y =
$csg_y<P>
31
32 PID = $$<P>
33
34 largeur = $largeur ; hauteur =
$hauteur<P>
35
36 /usr/X11R6/bin/convert -size
${largeur}x$hauteur
/home/httpd/html/carto/carte.rgb[200x
200+$csg_x+$csg_y]
gif:/home/httpd/html/carto/tmp/regi
on_$$.gif<P>
37
38 <IMG
SRC="/carto/tmp/region_$$.gif">
39 FIN_HTML
40
41 sub min
42 {
43      local ($v1, $v2) = @_;
44
45      return ($v1 < $v2) ? $v1 : $v2;
46 }
47
48 sub max
49 {
50      local ($v1, $v2) = @_;
51
52      return ($v1 > $v2) ? $v1 : $v2;
53 }
```

Etape 3c : élimination des fichiers temporaires

Le but de ce nouveau script est de remplacer les fichiers temporaires par une utilisation judicieuse de la sortie standard avec *convert*, car les fichiers temporaires se montrent particulièrement gênants dans le cas du Web, vu qu'ils ne peuvent déterminer avec exactitude la durée de vie qu'il convient de leur accorder.

Deux solution existent. Soit l'on affiche exclusivement la portion découpée, et dans ce cas, il s'agit d'un simple script qui envoie l'en-tête approprié et conclut son traitement par l'exécution de *convert*, lequel envoie, à son tour, son résultat sur la sortie standard. Soit l'on désire incorporer la portion choisie dans un document HTML et il faut alors scinder la tâche en deux scripts qui vont travailler en alternance ; le premier servira à élaborer dynamiquement le document HTML avec une directive d'insertion d'image faisant référence au second, celui-ci ayant pour rôle le découpage de la portion d'image et son envoi sur la sortie standard avec l'en-tête adéquat.

Passage à la pratique. Premier cas :

```
1 #!/usr/bin/perl
2
3 $largeur_vignette = 250;
4
5 open (TIFFINFO, '/usr/bin/tiffinfo
/home/httpd/html/carto/carte.tif |');
6 while (<TIFFINFO>)
7 {
8         if (/Image Width: (\d+)
Image Length: (\d+)/)
9    {
10      $largeur = $1;
11      $hauteur = $2;
12      last;
13   }
14 }
15 close (TIFFINFO);
16
17 $facteur = $largeur /
$largeur_vignette;
18
19 ($x, $y) = $ENV{'QUERY_STRING'} =~
/point_selectionne\.x=(\d+)&point_s
electionne\.y=(\d+)/;
20
21 $csg_x = min (max (0, int ($x *
$facteur - 100)), $largeur - 200);
22 $csg_y = min (max (0, int ($y *
$facteur - 100)), $hauteur - 200);;
23
24 $| = 1;
25
26 print "Content-type:
image/gif\n\n";
27
28 system ("/usr/X11R6/bin/convert
-size ${largeur}x$hauteur
/home/httpd/html/carto/carte.rgb[200x
200+$csg_x+$csg_y] gif:-");
29
30 sub min
31 {
32      local ($v1, $v2) = @_;
33
34      return ($v1 < $v2) ? $v1 : $v2;
```

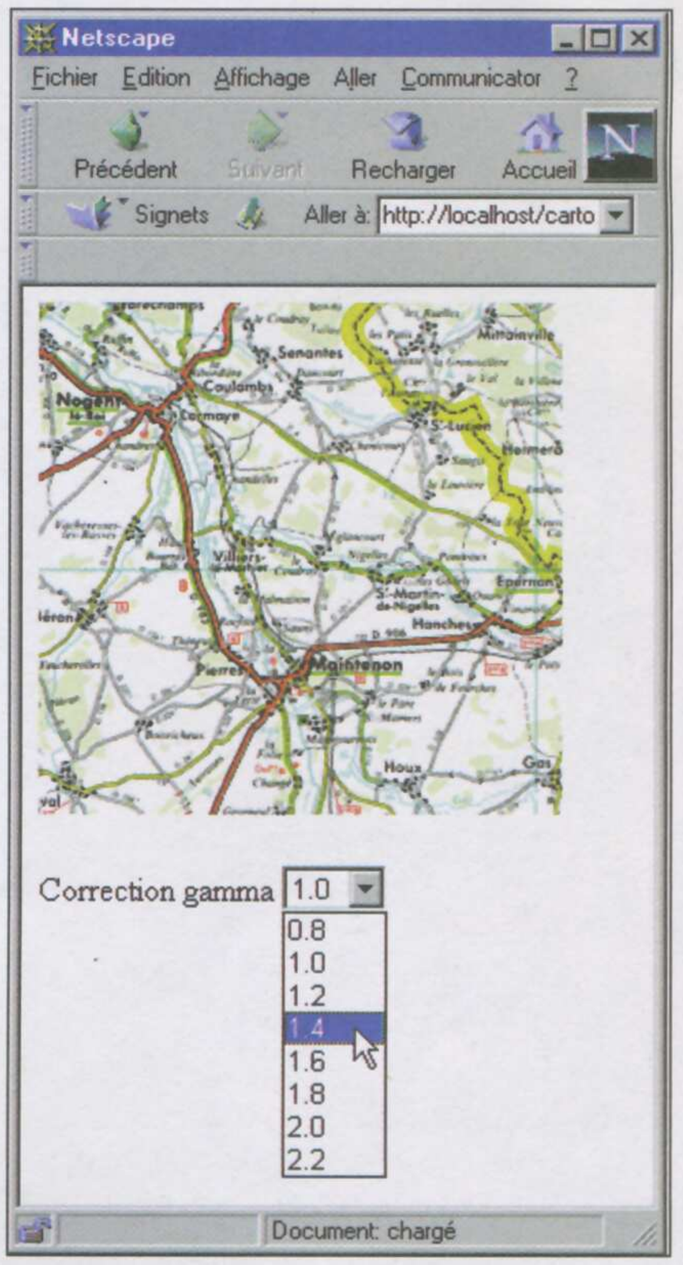
```
35 }
36
37 sub max
38 {
39      local ($v1, $v2) = @_;
40
41      return ($v1 > $v2) ? $v1 : $v2;
42 }
```

Vous remarquerez les deux principales innovations de ce script, le type du document créé, 'image/gif', remplaçant 'text/html', et le nom du fichier destination dans les paramètres de *convert* qui laisse place au tiret. Ce caractère symbolise sous UNIX, pour bon nombre de programmes - et *convert* n'échappe pas à cette règle - l'entrée ou la sortie standard. L'image ainsi créée ne transite plus sur le disque mais est directement expédiée au navigateur Web de l'internaute.
Petite variante : si votre image d'origine n'est pas une carte utilisant peu de couleurs mais une photo, le format GIF ne s'avère pas forcément le plus approprié. Au contraire, le JPEG est recommandé. Cela tombe à pic, car non seulement vous retrouverez toutes vos couleurs mais, pompon sur le gâteau, l'image élaborée nécessitera un nombre d'octets bien moins important que son équivalent GIF – environ trois fois moins - en conservant une bonne qualité. Enfin, et cela est aussi intéressant pour une image peu colorée, dans le contexte présent, *convert* demandera environ cinq fois moins de temps pour créer une représentation au format JPEG que son équivalent au format GIF. Pour parvenir à ce résultat, il suffit d'apporter trois modifications mineures aux lignes 26 et 28 pour obtenir ce qui suit.

```
26 print "Content-type:
image/jpeg\n\n";
27
28 system ("/usr/X11R6/bin/convert
-quality 90 -size ${largeur}x$hau-
teur
/home/httpd/html/carto/carte.rgb[20
0x200+$csg_x+$csg_y] jpeg:-");
```

Ligne 26, on remplace 'gif' par 'jpeg'. Ligne 28, on fait de même et on ajoute le niveau de qualité de l'image à produire (75 par défaut si cette option est absente).
Le second cas consiste à scinder la tâche en deux : un premier document au format HTML est retourné au navigateur, qui fait référence à une image n'existant pas encore ; en effet, on se contente simplement dans cette référence d'invoquer le script chargé de l'élaborer. Le premier script est donc remanié pour donner ce qui suit.

```
1 #!/usr/bin/perl
2
3 $largeur_vignette = 250;
4
5 open (TIFFINFO, '/usr/bin/tiffin-
fo /home/httpd/html/carto/carte.tif
|');
6 while (<TIFFINFO>)
7 {
8         if (/Image Width: (\d+)
Image Length: (\d+)/)
9  {
10    $largeur = $1;
11    $hauteur = $2;
12    last;
13  }
14 }
15 close (TIFFINFO);
16
17 $facteur = $largeur /
$largeur_vignette;
18
19 ($x, $y) = $ENV{'QUERY_STRING'}
=~
/point_selectionne\.x=(\d+)&point_
selectionne\.y=(\d+)/;
20
21 $csg_x = min (max (0, int ($x *
$facteur - 100)), $largeur - 200);
22 $csg_y = min (max (0, int ($y *
$facteur - 100)), $hauteur - 200);;
23
24
25 print <<"FIN_HTML";
26 Content-type: text/html
27
28 Document HTML g&eacute;n&eacu-
te;r&eacute; dynamiquement faisant
r&eacute;f&eacute;rence &agrave;
une
29 image qui va &ecirc;tre &agrave;
son tour g&eacute;n&eacute;r&eacu-
te;e dynamiquement<P>
30
31 <IMG SRC="/cgi-bin/
image_seule.pl?$csg_x.$csg_y.$lar-
geur.$hauteur"><P>
32
33 SANS laisser de trace sur le
disque du serveur.
34
35 FIN_HTML
36
37 sub min
38 {
39 local ($v1, $v2) = @_;
40
41 return ($v1 < $v2) ? $v1 : $v2;
42 }
43
44 sub max
45 {
46 local ($v1, $v2) = @_;
47
```



Premier script de l'étape 3f (vignette dans un navigateur AVEC menu gamma).

Fichier qui va servir de légende pour l'étape 3.

```
48 return ($v1 > $v2) ? $v1 : $v2;
49 }
```

On y retrouve une structure générale présentée précédemment avec quelques retouches. L'appel à *convert* a disparu, et la directive 'IMG' fait désormais référence à un second script plutôt qu'a une image statique.
Ce fameux script, 'image_seule.pl', est explicité ci-dessous.

```
1 #!/usr/bin/perl
2
3 ($csg_x, $csg_y, $largeur,
$hauteur) = split (/\./,
$ENV{'QUERY_STRING'});
4
5 $| = 1;
6
7 print "Content-type:
image/jpeg\n\n";
8
9 system ("/usr/X11R6/bin/convert
-quality 90 -size ${largeur}x
$hauteur
/home/httpd/html/carto/carte.rgb[20
```

```
0x200+$csg_x+$csg_y] jpeg:-");
```

Point très important à respecter : la présence de la ligne '$| = 1;', ici, et dans les scripts précédents. La variable spéciale $| en Perl gère le comportement des entrées/sorties quant à l'utilisation des buffers. Par défaut, les entrées/sorties sont 'bufferisées' (anglicisme barbare mais... faute de trouver mieux), essentiellement pour des questions de performances. Dans le cas présent, l'affichage de l'en-tête 'Content-type...' ne suffit pas à remplir à lui seul le buffer ; le contenu de celui-ci ne se trouve donc pas 'transmis' (de plus, la fin du script n'étant pas atteinte, ce buffer n'a pas non plus de raison de se vider - 'flush' en anglais -) et l'on passe alors sur la commande system qui invoque *convert*. Celle-ci envoie l'image, le script récupère la main, se termine et ses buffers se vident ; l'en-tête est alors, seulement à ce moment, envoyé... Un peu trop tard !

Étape 3d : utilisation d'une version compressée de l'image RGB

Si l'image au format RGB est vraiment trop volumineuse, il reste possible de la compresser, au détriment, on s'en doute, des performances. En effet, *convert* reconnaît et traite automatiquement les fichiers compressés à l'aide de compress ou de gzip, grâce à la présence des suffixes caractéristiques que ces commandes ajoutent aux noms des fichiers.
Deux minuscules manipulations suffisent. La première consiste bien sûr à compresser l'image RGB.

```
gzip -9 carte.rgb
```

La seconde se résume à remplacer dans le script CGI le nom 'carte.rgb' par 'carte.rgb.gz'. C'est tout !

Étape 3e : ajout d'une image superposée

Cette image, qui se superpose à celle découpée, peut contenir, par exemple, une légende, une règle des distances pour avoir une idée de l'échelle, le logo de la société offrant le service (en guise de copyright) ou une sorte d'étiquette arborant en gros le mot 'évaluation', pour montrer à l'internaute qu'il n'accède qu'à une version de démonstration.
Dans cette optique, il est nécessaire de créer une nouvelle image (avec Gimp par exemple) de 100 pixels de large sur 20 de haut pour les besoins de ce petit amusement. Trois horreurs gribouillées plus tard, et nous avons un fichier 'legende.tif'. Il ne reste plus qu'à apporter une ultime modification au dernier script pour que l'appel à *convert* devienne :

```
system ("/usr/X11R6/bin/convert
-draw 'image 90,170
/home/httpd/html/carto/legende.tif' -
quality 90 -size ${largeur}x$hauteur
/home/httpd/html/carto/carte.rgb.gz
[200x200+$csg_x+$csg_y] jpeg:-");
```

Vous remarquerez la présence effective de l'extension '.gz' derrière le nom de l'image RGB et la syntaxe permettant d'incruster notre légende sur la portion de carte présentée à l'internaute, placée dans son coin inférieur droit, à 10 pixels de distance des bords.
Ne subsiste qu'un seul petit regret : à moins d'avoir omis une possibilité lors des tests, la gestion de la transparence de l'image collée (si elle existe) fait défaut.

Étape 3f : gestion libre de la correction gamma

Poussons nos investigations un petit peu plus loin pour épater notre internaute en lui offrant la possibilité de modifier à sa guise la correction gamma de la portion d'image qui lui sera présentée. Il faut pour cela modifier nos trois fichiers. Dans le document HTML de présentation de la vignette, on ajoute quelques directives pour arriver au résultat suivant :

```
<FORM ACTION="/cgi-bin/doc_html.pl"
METHOD="GET"><INPUT TYPE="image"
SRC="carte.gif" NAME="point_
selectionne" BORDER="0">
<P>Correction gamma <SELECT
NAME="gamma">
<OPTION>0.8<OPTION
SELECTED>1.0<OPTION>1.2<OPTION>1.4
<OPTION>1.6<OPTION>1.8<OPTION>2.0<O
PTION>2.2</SELECT>
</FORM>
```

En fait, on se contente de proposer un menu déroulant avec quelques valeurs prédéfinies dans l'intervalle autorisé par l'option -gamma de *convert*.
Dans le script principal, les lignes 19, 20 et 31 sont remplacées par :

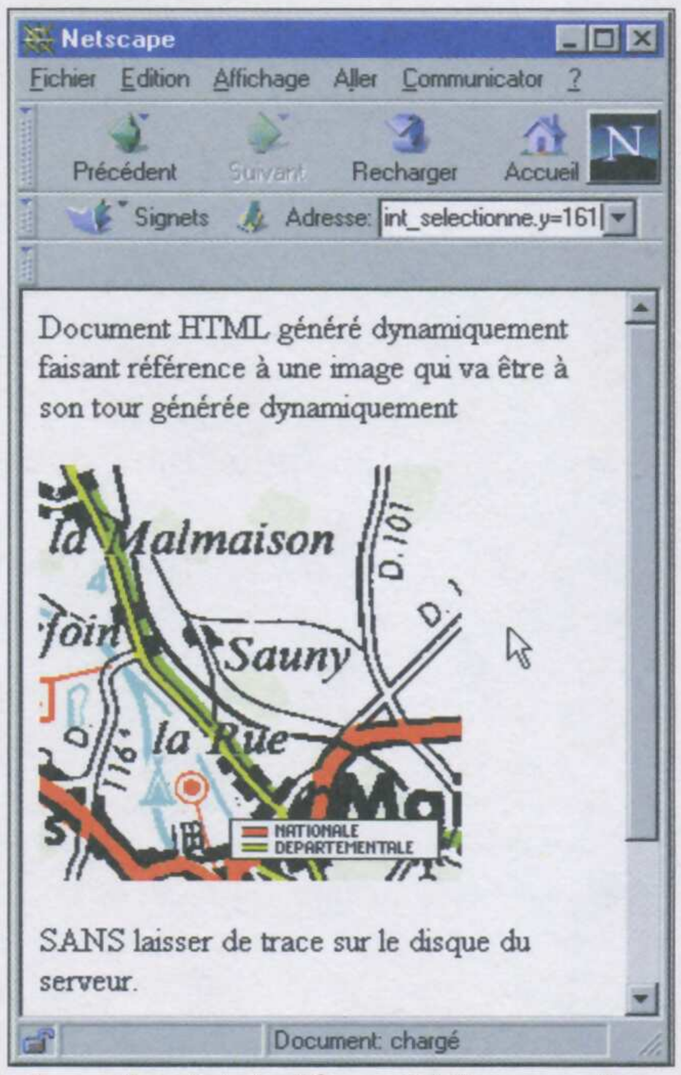
```
19 ($gamma, $x, $y) =
$ENV{'QUERY_STRING'} =~
/gamma=(\d\.\d)&point_selectionne\.
x=(\d+)&point_selectionne\.y=(\d+)/;
20 $gamma *= 10;
31 <IMG SRC="/cgi-bin/
image_seule.pl?$csg_x.$csg_y.$lar-
geur.$hauteur.$gamma"><P>
```

Enfin, le second et dernier script (image_seule.pl) ressemblera à :

```
#!/usr/bin/perl
($csg_x, $csg_y, $largeur, $hauteur,
$gamma) = split (/\./,
$ENV{'QUERY_STRING'});
$gamma /= 10;
$| = 1;
print "Content-type:
image/jpeg\n\n";
system ("/usr/X11R6/bin/convert
-draw 'image 90,170
/home/httpd/html/carto/legende.tif'
-quality 90 -gamma $gamma -size
${largeur}x$hauteur
/home/httpd/html/carto/carte.rgb.gz
[200x200+$csg_x+$csg_y] jpeg:-");
```

Ceci clôt ce survol de quelques-unes des très nombreuses possibilités de *convert*.
Passons maintenant aux réflexions de comptoir. Bien évidemment, il y a moyen d'améliorer encore les temps de réponse, en commençant par éviter d'invoquer systématiquement le programme *tiffinfo* alors que l'on se trouve en mesure de stocker les informations nécessaires dans un fichier à part, pouvant s'appeler carte.rgb.size par exemple. Il est maintenant de votre ressort d'optimiser ces quelques exemples.
On pourrait aussi, afin d'éviter la multiplication des scripts qui concourent à un même but, les regrouper en un seul avec une détermination automatique du traitement à accomplir par une simple analyse du contenu de la variable d'environnement QUERY_STRING. Un unique script pourrait alors, tour à tour, engendrer le document HTML ou envoyer l'image invoquée par ce document, suivant la nature des paramètres détectés dans QUERY_STRING.

Yannick Cadin
Yannick@kommando.com



Second script de l'étape 3f (le zoom dans un navigateur AVEC légende et AVEC correction gamma).

Traitement d'images compressées

Dans le précédent article, nous utilisions convert (du lot d'utilitaires ImageMagick) pour extraire une petite portion d'une image volumineuse. Nous allons ici améliorer encore ce processus, pour présenter d'autres commandes intéressantes.

Mais surtout, nous nous affranchirons d'un petit problème que l'on rencontre avec les commandes ImageMagick, à savoir le chargement intégral des images avant leur exploitation. Bon nombre d'utilitaires graphiques présentent le même inconvénient.
Autrement dit, pourquoi charger une image pouvant faire plusieurs milliards d'octets alors que seule l'extraction d'une infime partie nous intéresse ? Là encore, cette contrainte sera le prétexte à la mise en oeuvre d'une bibliothèque bien sympathique.

Utiliser des images compressées

On commence en donnant une alternative à l'emploi de convert pour ce qui concerne l'extraction, avec la commande fbext tirée du lot de commandes FBM, Fuzzy Pixmap Manipulation (ftp://nl.cs.cmu.edu/usr/mlm/ftp/fbm1.2.tar.Z).
Le bénéfice ici consiste à pouvoir utiliser en entrée des formats graphiques compressés, ce que n'autorise pas convert dans le cadre d'une extraction, si ce n'est par l'artifice consistant à compresser le fichier RGB avec la commande gzip, ce qui s'avère assez peu exaltant.
Avant de compiler, il peut être intéressant de modifier le Makefile de FBM, afin que les variables BIN et MAN pointent respectivement sur /usr/local/bin et /usr/local/man (les créer au besoin ainsi que /usr/local/man/manl).
Les utilitaires fournis sont susceptibles de reconnaître les formats les plus usités, à condition toutefois de disposer des bibliothèques correspondantes et d'en fournir les chemins d'accès via le fichier Makefile (pour les TIFF et JPEG en particulier).
Afin de montrer à quoi ressemblerait éventuellement un appel à fbext dans notre script du mois dernier (image_seule.pl), nous nous contentons du format GIF pour le codage de l'image source. En revanche, fbext ne connaissant que la version 87a du format GIF, il faut au préalable convertir notre image dans ce format, comme indiqué ci-dessous :

```
convert carte.tif gif87:carte.gif87
```

En outre, dans le script image_seule.pl, il suffit de modifier la ligne principale pour qu'elle ressemble à ce qui suit :

```
system ("/usr/local/bin/fbext -G $csg_x
$csg_y 200 200 <carte.gif87 |
/usr/X11R6/bin/convert -draw 'image 90,170
legende.tif' -quality 90 -gamma $gamma
gif87:- jpeg:-");
```

Quelques remarques s'imposent à présent. L'option -G non documentée dans le 'man'uel de fbext correspond à l'indication de format GIF en sortie. Cette option ainsi que celles des autres formats connus sont visibles par 'man fbm'. Le format en entrée se trouve déterminé automatiquement et peut donc différer du résultat.
Il est évident que pour des images de plus de 16 000 pixels de côté, il faudra recourir au format TIFF en entrée et donc interfacer les commandes FBM avec la libtiff.
Enfin, comme on peut le constater, l'intérêt global demeure faible, car il faut tout de même faire appel à convert pour l'incrustation d'une légende sous forme de vignette graphique. Cela permet toutefois de présenter FBM, ainsi que la syntaxe requise pour demander à convert de lire sur l'entrée standard dans un format donné.

On n'est jamais mieux servi...

Cette petite transition n'était là que pour présenter FBM, un des nombreux outils de manipulation des graphiques bitmap. Il n'en reste pas moins qu'il n'est pas forcément très adapté à des extractions au sein de très gros fichiers, si lui aussi charge l'intégralité de ces fichiers avant que d'aller piocher dedans.
Aussi, à défaut de trouver la perle rare (qui pourtant existe, nous n'en doutons pas), nous pouvons la développer nous-mêmes. Le cahier des charges est relativement simple, avec un support de grandes images (jusqu'à 4 Go en l'occurrence), une absence de réelle limitation en nombre de pixels en largeur et hauteur et un maximum de 256 couleurs (puisque prévu a priori pour des cartes)... Deux points importants sont également à prendre en compte : le stockage d'un index pour connaître l'adresse de chaque début de ligne dans le fichier car, et là réside le deuxième point et l'aspect le plus intéressant, nous allons compacter chaque ligne à l'aide d'une fonction de la librairie zlib.
En clair, notre document pourra faire jusqu'à 4 Go, MAIS 4 Go compressés.
Vous retrouvez les sources auto-documentés des deux programmes de conversion vers, ET depuis, notre format propriétaire.
Et même si ce n'est pas encore la panacée, vous percevez rapidement que nous ne nous préoccupons que des lignes dont une partie des points va se retrouver dans la portion extraite, grâce à ce fameux index.
De retour dans notre script image_seule.pl, la commande principale devient désormais :

```
system ("/usr/local/bin/cpii2rgb carte.cpii
$csg_x $csg_y 200 200 |
/usr/X11R6/bin/convert -draw 'image 90,170
legende.tif' -quality 90 -gamma $gamma -size
200x200 rgb:- jpeg:-");
```

Vous remarquerez que l'on s'appuie toujours sur convert pour élaborer le document dans son format définitif. Cela nous évite de développer les routines d'écriture en GIF, JPEG ou autre PNG. On se contente d'envoyer du RGB, ce qui est simpliste, et convert se charge du travail délicat. Il permet aussi et surtout de traiter l'incrustation de notre vignette graphique, ainsi que la correction gamma.
Au préalable, il aura bien évidemment fallu convertir notre image originale dans notre nouveau format (le CPII) à l'aide de la commande suivante :

```
/usr/local/bin/rgb2cpii carte.rgb 1000 1000
carte.cpii
```

Concernant les sources, vous passerez rapidement sur la piètre qualité de leur rédaction pour ne vous focaliser que sur certains détails spécifiques à l'emploi des fonctions compress2 et uncompress de zlib.
Le point probablement le plus important à prendre en considération est la double fonction de la variable contenant la taille du buffer de destination. En effet, si après l'appel aux fonctions compress2 ou uncompress, elle contient la longueur

résultant du traitement, elle sert aussi à fournir en entrée à ces mêmes fonctions la largeur du buffer dans lequel elles peuvent évoluer. Il est donc indispensable de toujours y placer la valeur appropriée AVANT chaque appel.

La grande diversité de librairies ou d'API graphiques spécialisées (libtiff, API de GIMP, librairies d'ImageMagick, libpng, libjpeg, libungif, etc.) fournies en standard ou téléchargeables depuis Internet offre beaucoup de perspectives pour n'importe quel développement spécifique.

Yannick Cadin-yannick@kommando.com

Voir toujours plus grand, toujours plus vite

Nous reprochions à Photoshop, dans le précédent article, de s'auto-limiter à 30 000 pixels de côté. Si ces dimensions s'avèrent suffisantes pour bon nombre d'utilisateurs, il est des professions pour lesquelles ces valeurs rendent le logiciel simplement inexploitable. Pour eux, il existe désormais un programme appelé ViewMan (View Manager), édité par la société israélienne Scan Group (dont les produits sont distribués en France par Levona à Montreuil) et capable d'afficher à une vitesse fabuleuse des documents TIFF compressés, pesant plusieurs centaines de millions d'octets (jusqu'à 4 Go), et ce, sans nécessiter de mémoire vive ni de disque dur très important.

Outre la visualisation, ce programme permet l'impression sur traceur HP, en envoyant directement les images au format natif de ces matériels, le HP-RTL.

Spécialisé dans la retouche post-numérisation, il offre des fonctions de nettoyage et de séparation des couleurs en couches distinctes.

```
cpii2rgb.c
/* Decompression depuis image au format CPII */

/* La compilation ressemble a :
  gcc -O3 -DHAVE_UNISTD_H -DUSE_MMAP -o cpii2rgb cpii2rgb.c -lz
  strip cpii2rgb
*/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <zlib.h>

#define LARGEUR_MIN  1
#define HAUTEUR_MIN  1

#define min(a,b) ((a) < (b) ? (a) : (b))

void developpeCouleurs(char *table);

void afficheUtilisation(const char *application) {
fprintf(stderr,"Usage : %s image_compressee x y largeur hauteur
[image_RGB]\n", application);
}

int main(int argc, char *argv[]) {
if ((argc != 6) && (argc != 7)) {
  afficheUtilisation(argv[0]);
} else {
  long          xExtraction = atoi(argv[2]), yExtraction =
atoi(argv[3]), largeurExtraction = atoi(argv[4]), hauteurExtraction =
atoi(argv[5]);
  if (largeurExtraction >= LARGEUR_MIN && hauteurExtraction >= HAU-
TEUR_MIN) {
    static const char *maVersion = ZLIB_VERSION;
    FILE * source, *destination;

    if (zlibVersion()[0] != maVersion[0]) {
      fprintf(stderr, "Version de librairie zlib inutilisable\n");
      exit(1);
    } else {
      if (strcmp(zlibVersion(), ZLIB_VERSION) != 0)
        fprintf(stderr,"Attention : versions différentes de librairies
zlib\n");
    }
if (argc == 7) {
  if (!(destination = fopen(argv[6], "wb"))) {
    fprintf(stderr, "création %s impossible\n", argv[6]);
    exit(1);
    }
  } else {
    destination = stdout;
  }
  if (source = fopen(argv[1], "rb")) {
    char signature[8];
    long largeur, hauteur,  longueurMaximale;
    char couleurs[256][6];

    if ((fread(signature, sizeof(signature), 1, source) == 1) &&
    (fread(&largeur, sizeof(long), 1, source) == 1) && (fread
(&hauteur,sizeof(long),1, source)== 1) &&
    (fread(&longueurMaximale, sizeof(long), 1, source) == 1) &&
    (fread(couleurs, sizeof(long), 3, source) == 3) && (fread(cou-
leurs, sizeof(long),256,
source)== 256)) {
      unsigned long     *tableLignes;
      Byte * donneesCompressees, *donneesDecompressees;

      if ((xExtraction >= largeur) || (yExtraction >= hauteur)) {
        fprintf(stderr,"zone à extraire en dehors des limites de
l'image\n");
        exit(1);
      }
      largeurExtraction = min(largeurExtraction, largeur -
xExtraction);
      hauteurExtraction = min(hauteurExtraction, hauteur -
yExtraction);
      developpeCouleurs((char *) couleurs);
      tableLignes = (unsigned long *) malloc((hauteurExtraction +
1) *sizeof(long));
      donneesCompressees = (Byte * ) malloc(longueurMaximale *
sizeof(Byte));
      donneesDecompressees = (Byte * ) malloc(largeur *
sizeof(Byte));
      if (tableLignes == Z_NULL || donneesCompressees == Z_NULL ||
donneesDecompressees == Z_NULL) {
        fprintf(stderr, "mémoire insuffisante\n");
        exit(1);
      }
      if (fseek(source, sizeof(signature) + (6 + 256 +
```

```
yExtraction)*sizeof(long), SEEK_SET) == -1) {
            fprintf(stderr,"erreur lors du déplacement dans
l'index\n");
            exit(1);
          }
          if (fread(tableLignes, sizeof(long), hauteurExtraction +
1,source) == (hauteurExtraction + 1)) {
            if (fseek(source, sizeof(signature) + (6 + 256 + hauteur +
1) * sizeof(long)+ tableLignes[0], SEEK_SET) == -1) {
              fprintf(stderr,"erreur lors du déplacement dans l'in-
dex\n");
              exit(1);
            } else {
              long ligne;
              for (ligne = 0; ligne < hauteurExtraction; ligne++) {
                long longueurCompressee = tableLignes[ligne + 1] -
tableLignes[ligne],        longueurDecompressee = largeur,
numeroPixel;
                int erreur;

                if (fread(donneesCompressees,
sizeof(Byte),longueurCompressee, source) != longueurCompressee) {
                  fprintf(stderr, "données manquantes\n");
                  exit(1);
                }
                if (((erreur =
uncompress(donneesDecompressees,&longueurDecompressee,
donneesCompressees,longueurCompressee)) != Z_OK) ||
(longueurDecompressee != largeur)) {
                  fprintf(stderr,"une erreur de décompression est
intervenue ligne %d (code %d)\n",ligne + yExtraction, erreur);
                  exit(1);
                }
                for (numeroPixel = xExtraction; numeroPixel <
(xExtraction+ largeurExtraction); numeroPixel++) {
                  if
(fwrite(&couleurs[donneesDecompressees[numeroPixel]],sizeof(char), 6,
destination) != 6) {
                    fprintf(stderr, "écriture impossible\n");
                    exit(1);
                  }
                }
              }
            } else {
              fprintf(stderr,"lecture de l'index des lignes à extraire
impossible\n");
            }
          } else {
            fprintf(stderr, "lecture de l'entête impossible\n");
          }
          fclose(source);
          if (destination != stdout)
            fclose(destination);
        } else
          fprintf(stderr, "ouverture %s impossible\n", argv[1]);
      } else
        fprintf(stderr,"largeur ou hauteur trop faible pour justifier
une extraction (%d pixels au minimum)\n",LARGEUR_MIN);
    }
exit(0);
return 0;
}

void developpeCouleurs(char *table) {
int indice;
long *couleurs = (long *) table;
for (indice = 255; indice >= 0; indice--) {
  long couleur = couleurs[indice];
  char rouge = (couleur >> 16) &255, vert = (couleur >> 8) &255, bleu
= couleur &255;
  table[indice * 6 + 5] = table[indice * 6 + 4] = bleu;
  table[indice * 6 + 3] = table[indice * 6 + 2] = vert;
  table[indice * 6 + 1] = table[indice * 6 + 0] = rouge;
  }
}
```

```
rgb2cpii.c

/* Ameliorations futures :
   grouper les differents elements de l'entete dans une structure
(plus simple a manipuler pour les entrees/sorties) */

/* Compression depuis une image au format RGB */

/* Structure du fichier genere :
 signature sur 8 octets (identifiant sur 4 caracteres "CPII", numero
de version sur 3 caracteres 000-999 et caractere nul)
 largeur sur 4 octets
 hauteur sur 4 octets
 longueur compressee maximale sur 4 octets
 zone libre pour usage futur sur 12 octets
 palette sur 4 x 256 = 1024 octets (fixe) ; 4e octet de chaque couleur
(celui de poids fort) reserve pour usage futur (pour l'instant a 0xff)
 index de toutes les lignes sur (hauteur + 1) x 4 octets (la derniere
valeur permet de calculer la taille compressee de la derniere ligne de
l'image)
 chaque ligne est ensuite codee sur largeur x 1 octet (index dans la
palette) puis compressee
*/

/* La compilation ressemble a :
  gcc -O3 -DHAVE_UNISTD_H -DUSE_MMAP -o rgb2cpii rgb2cpii.c -lz
  strip rgb2cpii
*/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <zlib.h>

#define LARGEUR_MIN   500
#define HAUTEUR_MIN   500

char numeroEncre (long *palette, char rouge, char vert, char bleu);


void afficheUtilisation (const char *application){
printf ("Usage : %s image_RGB largeur hauteur image_compressee\n",
application);
}
```

```
int main(int argc, char *argv[]){
if (argc != 5)
  afficheUtilisation (argv[0]);
else {
  char signature[8] = "CPII001";  /* Compressed Paletted Indexed Image
*/
  long largeur = atoi (argv[2]), hauteur = atoi (argv[3]);

  if (largeur >= LARGEUR_MIN && hauteur >= HAUTEUR_MIN) {
    static const char *maVersion = ZLIB_VERSION;
    unsigned long palette[256], *tableLignes;
    Byte *donneesCompressees, *donneesDecompressees;
    FILE *source, *destination;

    if (zlibVersion ()[0] != maVersion[0]) {
      fprintf (stderr, "Version de librairie zlib inutilisable\n");
      exit (1);
    } else {
      if (strcmp (zlibVersion (), ZLIB_VERSION) != 0)
        fprintf (stderr, "Attention : versions différentes de librai-
ries zlib\n");
    }
    tableLignes = (unsigned long *) malloc ((hauteur + 1) * sizeof
(long));
    donneesCompressees = (Byte * ) malloc ((long) (largeur * 1.1 + 12)
* sizeof (Byte));
    donneesDecompressees = (Byte * ) malloc (largeur * sizeof (Byte));
    if (tableLignes == Z_NULL || donneesCompressees == Z_NULL ||
donneesDecompressees == Z_NULL) {
      fprintf (stderr, "mémoire insuffisante\n");
      exit (1);
    }
    if ((source = fopen (argv[1], "rb")) && (destination = fopen
(argv[4], "wb"))) {
      long ligne,
      longueurMaximale = 0;
      unsigned long deplacement = 0;
      if (fseek (destination, sizeof (signature) + (6 + 256 + hauteur
+ 1) * sizeof (long), SEEK_SET) == -1) {
        fprintf (stderr, "réservation de la zone d'entête
impossible\n");
        exit (1);
      }
      memset (palette, 0, 256 * sizeof (long));
      for (ligne = 0; ligne < hauteur; ligne++) {
        long numeroPixel, longueurCompressee = (long) largeur *1.1 +
12;
        int erreur;
        for (numeroPixel = 0; numeroPixel < largeur; numeroPixel++) {
          char pixel[6];
          if (fread (pixel, sizeof (pixel), 1, source) == 1)
            donneesDecompressees[numeroPixel] = numeroEncre (palette,
pixel[0],pixel[2], pixel[4]);
          else {
            fprintf (stderr, "données manquantes\n");
            exit (1);
          }
        }

        if ((erreur = compress2 (donneesCompressees,
&longueurCompressee, donneesDecompressees, largeur, Z_BEST_COMPRES-
SION)) != Z_OK) {
          fprintf (stderr, "une erreur de compression est intervenue
ligne %d (code %d)\n", ligne, erreur);
          exit (1);
          }
        if (fwrite (donneesCompressees, sizeof (char),
longueurCompressee, destination) != longueurCompressee) {
          fprintf (stderr, "écriture impossible\n");
          exit (1);
        }
        if (longueurCompressee > longueurMaximale)
          longueurMaximale = longueurCompressee;
        tableLignes[ligne] = deplacement;
        deplacement += longueurCompressee;
      }
      tableLignes[ligne] = deplacement;
      fclose (source);
      rewind (destination);
      if ((fwrite (signature, sizeof (signature), 1, destination) ==
1) && (fwrite (&hauteur, sizeof (long),1, destination) == 1) && (fwri-
te (&largeur, sizeof (long), 1, destination) == 1) && (fwrite
(&longueurMaximale,sizeof (long), 1, destination) == 1) && (fwrite
(palette, sizeof (long), 3, destination) == 3) && (fwrite (palette,
sizeof (long), 256, destination) == 256) && (fwrite
(tableLignes,sizeof (long), hauteur + 1, destination) == (hauteur +
1))) {
        fclose (destination);
        printf ("Image compressée !\n");
      } else{
        fclose (destination);
        fprintf (stderr, "écriture entête impossible\n");
        unlink (argv[4]);
      }
      free (tableLignes);
      free (donneesCompressees);
      free (donneesDecompressees);
    } else
      fprintf (stderr, "ouverture %s ou création %s impossible\n",
argv[1], argv[4]);
  } else
    fprintf (stderr, "largeur ou hauteur trop faible pour justifier
une compression (%d minimum)\n",LARGEUR_MIN);
  }
exit(0);
return 0;
}

char numeroEncre (long *palette, char rouge, char vert, char bleu) {
long couleur = (255 << 24) | (rouge << 16) | (vert << 8) | bleu,
couleurTestee;
int indice;
for (indice = 0; indice < 256; indice++) {
  if (couleurTestee = palette[indice]) {
    if (couleur == couleurTestee)
      return (char) indice;
  } else
    break;
}
if (indice == 256) {
  static int excesCouleursAffiche = 0;
  if (!excesCouleursAffiche) {
    printf ("Il y a plus de 256 couleurs différentes dans l'image\n");
```