Wolfgang Kindeldei

Imager - eine universelle Bildverarbeitungs-Bibliothek

Motivation

Immer wieder ergibt sich die Problemstellung, Thumbnails zu erstellen, Bilder mit Wasserzeichen, einem Logo oder Text zu versehen oder gar Graphiken von Grund auf selbst zu erstellen. Greift man zum Klassiker ImageMagick? Oder gibt es Alternativen? Mein persönlicher Favorit ist `Imager`, eine schnelle und universelle Bibliothek zur Bildbearbeitung, die auf diverse in C geschriebene Bibliotheken zurückgreift.

Um `Imager` schmackhaft zu machen, möchte ich mit einem kurzen Benchmark beginnen. Eine JPEG-Datei aus einer 16 Megapixel Kamera soll zu einem Thumbnail mit einer Breite von 150 Pixeln konvertiert werden. Alle erzeugten JPEG Dateien wurden mit jeweils der maximal möglichen Qualität erzeugt, damit auch Unterschiede in puncto Qualität vergleichbar zu machen.

Angetreten sind ImageMagick als Kommandozeile (Convert), `Image::Magick`, `GD`, `Imager` und `Image::Epeg` (siehe Listing 1).

Schön, `Imager` ist etwa doppelt so schnell wie ImageMagick, unabhängig davon, ob die Kommandozeilen- oder Perl-Version von ImageMagick benutzt wird. Absoluter Gewinner ist `Image::Epeg`, das nochmal um Faktor 4 schneller ist. Leider kann letzteres Modul lediglich JPEG Dateien lesen und erzeugen. Außerdem kennt diese Bibliothek keinerlei Bildmanipulations-Befehle, scheidet daher bei vielen Anforderungen aus. Die Qualität der erzeugten Bilder ist ebenfalls die schlechteste der vier Test-Kandidaten.

Dass ImageMagick langsamer ist als Imager, ist ebenfalls erklärbar. Imager beherrscht keinerlei Farbmanagement, kann also nicht mit Farbprofilen umgehen und kann Metadaten aus Bildern weder lesen noch schreiben. ImageMagick kann neben vielen weiteren Dingen damit umgehen, was der Ausführungszeit leider nicht gerade entgegen kommt. Auch entschädigt die längere Laufzeit mit geringfügig schärferen Ergebnissen.

Installation

Etwas Vorsicht ist beim Installieren von Imager geboten, denn Imager sucht nach zahlreichen C-Bibliotheken und bindet alles Verfügbare ein. Je nach Zustand des Systems, auf dem Imager compiliert wird, fällt der Funktionsumfang daher unter Umständen ganz anders aus. Fehlermeldungen gibt es nicht, denn dieses Verhalten ist zunächst so beabsichtigt, um möglichst flexibel zu sein.

Wer Transparenz möchte, kann sich zum Beispiel so behelfen, allerdings ist das natürlich kein Weg für automatisierbare Installationen.

```
          Rate Convert  Magick      GD  Imager    Epeg
Convert 1.14/s      --     -6%    -25%    -52%    -91%
Magick  1.22/s      7%      --    -19%    -48%    -90%
GD      1.52/s     32%     24%      --    -36%    -88%
Imager  2.37/s    107%     94%     57%      --    -81%
Epeg    12.3/s    974%    905%    711%    418%      --
```

Listing 1

```
$ cpanm --interactive Imager
  ...
Libraries found:
  FT2
  JPEG
  PNG
  T1
  TIFF
Libraries *not* found:
  GIF
  Win32
OK
Building and testing Imager-0.97 ... OK
Successfully reinstalled Imager-0.97
1 distribution installed
```

In diesem Fall kann Imager auf einem *nix System nicht mit GIF Dateien umgehen, da die dafür benötigte Bibliothek nicht gefunden wurde. Die zusätzliche Distribution `Imager::File::GIF` wird in diesem Fall einfach nicht mit installiert.

Wenn ein Projekt bestimmte Dateiformate benötigt, kann die zusätzliche Installation der diversen zusätzlichen Distributionen `Imager::File::*` oder `Imager::Font::*` vornehmen. Laufen die Tests der Installation erfolgreich durch, ist alles im grünen Bereich. Fehlschlagende Tests lassen dann relativ schnell Rückschlüsse auf fehlende oder inkompatible C-Bibliotheken zu.

```
$ cpanm Imager::File::GIF
--> Working on Imager::File::GIF
Fetching .../authors/id/T/TO/TONYC/
   Imager-File-GIF-0.88.tar.gz ... OK
Configuring Imager-File-GIF-0.88 ... N/A
! Configure failed for Imager-File-GIF-0.88.
  See .cpanm/build.log for details.
```

Funktionsumfang

Um einen Teil der Möglichkeiten von Imager kennen zu lernen, schauen wir uns ein Beispiel-Programm an. Aus einem vorgegebenen Bild wird eine neue Graphik erzeugt, die ein Abspiel-Symbol in der Mitte sowie ein Logo in einer Ecke besitzt. Eine Copyright Information in einer weiteren Ecke wird ebenfalls mit gezeichnet. Um erweiterbar zu sein und den Programmcode leichter nachvollziehbar zu halten, erzeugen wir eine einfache Klasse (Thumbnail), in der die verwendeten Operationen sowie ein Imager-Objekt gekapselt sind. Die Erzeugung unserer Graphik sieht dann so aus:

```
Thumbnail
  ->load($source_file)
  ->scale(THUMBNAIL_WIDTH, THUMBNAIL_HEIGHT)
  ->add_play_icon(PLAY_ICON_RADIUS)
  ->add_logo($logo_file)
  ->add_copyright
  ->save($thumbnail_file);
```

Die einzelnen Operationen sind relativ schnell abgehandelt und hoffentlich weitgehend selbstsprechend. Die von uns eingesetzte Klasse besitzt lediglich ein Attribut (image), in dem das Imager-Bild im jeweils aktuellen Zustand gehalten wird.

```
package Thumbnail;
use Moose;
use Imager;

has image => (
    is  => 'rw',
      isa => 'Imager',
);
```

Bild Datei lesen

```
sub load {
    my ($class, $file) = @_;

    my $self = $class->new;

    $self->image(
       Imager->new(file => $file));

    return $self;
}
```

Skalieren und in korrekte Größe bringen

Die Skalierung erfordert zwei Arbeitsschritte, denn die reine Skalierung behält das Seitenverhältnis des Bildes bei. Wir müssen also das skalierte Bild noch zuschneiden.

```
sub scale {
    my ($self, $width, $height) = @_;

    # skalieren -- erzeugtes Bild
    # ist zu groß
    my $image = $self->image->scale(
        xpixels => $width,
        ypixels => $height,
        type    => 'max',
        qtype   => 'mixing',
    );

    # zuschneiden -- Mitte ist Fixpunkt
    my $left =
     int(($image->getwidth  - $width)  / 2);
    my $top  =
     int(($image->getheight - $height) / 2);
    $image = $image->crop(
        left   => $left,
        right  => $left + $width,
        top    => $top,
        bottom => $top + $height,
    );

    $self->image($image);

    return $self;
}
```

Zeichnen eines einfachen Abspiel-Pfeils

```
sub add_play_icon {
    my ($self, $radius) = @_;

    my $center_x =
      int($self->image->getwidth  / 2);
    my $center_y =
      int($self->image->getheight / 2);

    # große schwarze Fläche
    $self->image->circle(
        color => '#000000',
        r     => $radius + 2,
        x     => $center_x,
        y     => $center_y,
        aa    => 1,
    );

    # etwas kleinere weiße Fläche
    $self->image->circle(
        color => '#ffffff',
        r     => $radius,
        x     => $center_x,
        y     => $center_y,
        aa    => 1,
    );

    # schwarzes Dreieck
    my $delta = ($radius - 2) / sqrt(2);
    $self->image->polygon(
        color  => '#000000',
        points => [
            [ $center_x - $delta,
                $center_y - $delta],
            [ $center_x + $radius - 2,
                $center_y ],
            [ $center_x - $delta,
                $center_y + $delta],
        ],
        aa     => 1,
    );

    return $self;
}
```

Hinzufügen eines Logos aus anderer Bild-Datei

```
sub add_logo {
    my ($self, $file) = @_;

    my $logo = Imager->new(file => $file);
    $self->image->paste(
        left => $self->image->getwidth
              - $logo->getwidth  - 1,
        top  => $self->image->getheight
              - $logo->getheight - 1,
        src  => $logo,
    );

    return $self;
}
```

Text zeichnen

```
sub add_copyright {
    my $self = shift;

    my $text = 'WKI';
    my $font = Imager::Font->new(
        file => "$FindBin::Bin/Twister.ttf",
    );

    $self->image->string(
        string => $text,
        x      => 5,
        y      => $self->image->
                    getheight - 10,
        color  => '#ffffff',
        font   => $font,
        size   => 20,
        aa     => 1,
    );

    return $self;
}
```

speichern des erzeugten Bildes

```
sub save {
    my ($self, $file) = @_;

    $self->image->write(
        file => $file,
    );

    return $self;
}
```

Zugabe

Imager kann noch viel mehr. Dank der brillanten Dokumentation sind die notwendigen Methoden und deren Argumente auch schnell herauszufinden. Nicht unerwähnt bleiben sollten die Eigenschaften dennoch:

• Füll-Funktionen
gezeichnete Flächen lassen sich nicht nur mit soliden Farben füllen, sondern auch mit Mustern, Schraffuren, Verläufen oder gekachelten Bildern.

• Transformationen
neben der gezeigten Skalierung stehen auch Streckungen, Rotationen und Spiegelungen zur Verfügung.

• Farb-Manipulationen
Diverse Operationen bieten Farb-Operationen, Helligkeits- und Kontrastveränderungen sowie diverse arithmetische Veränderungen beim Zusammenfügen verschiedener Einzelteile.

• Transparenzen
Imager kann wahlweise einen Alpha-Kanel mitführen, der beim Erzeugen von zum Beispiel PNG- oder GIF-Bildern transparente Bilder produzieren kann.

• Anti-Aliasing
fast jeder Zeichenbefehl beherrscht die Option `aa`, mit der die Kanten der gezeichneten Objekte geglättet werden, was optisch ansprechendere Ergebnisse zur Folge hat.

• Filter
Fast schon Photoshop-Niveau hat die Liste der zur Verfügung stehenden Filter Funktionen. Damit stehen eine ganze Reihe von Verwandlungs- und Verfremdungs-Befehle zur Auswahl.

Letzte Zugabe

Zahlreiche CPAN Autoren haben Imager ebenfalls in ihr Herz geschlossen und diverse Distributionen auf Imager aufgebaut. Hier eine kleine Auswahl:

• Imager::Montage
Erlaubt die Erzeugung einer Komposition aus einer Menge von Einzelbildern.

• Imager::Graph
Erzeugt Balken-, Torten- und Liniengraphiken mit sehr viel Variations-Möglichkeiten.

• Imager::Simple
Wie der Name vermuten lässt, erleichtert dieses Modul einige Arbeiten rund um Imager.

• Imager::Heatmap
Erzeugt Heatmaps aus statistischen Daten.

• Imager::QRCode
Erleichtert die Erzeugung von QR Codes.

Fazit

Wer auf Farbmanagement verzichten kann und auf bestimmte Dateiformate (z.B. EPS oder PDF) keinen Wert legt, hat mit Imager eine schnelle und vielseitige Bildverarbeitungs-Bibliothek, die ohne Zicken auch automatisiert installierbar ist und mit umfangreicher Dokumentation geliefert wird. Der Funktionsumfang ist ebenso überzeugend wie die zahlreiche zusätzlichen auf CPAN befindlichen auf Imager aufbauenden Distributionen.