Image format conversions

R (Chandra) Chandrasekhar

2021-03-07 | 2021-03-07

Converting between image formats is something we could be required to do at short notice. This tutorial explores the different tools that are currently available and identifies the most efficient for each task. The `ImageMagick` suite, the `cairo` backend, the `poppler` utilities, `rsvg-convert`, and `CairoSVG` are identified for the specific strengths that make them the tools of choice for different image conversion tasks.

Two varieties of digital images

Digital images come in two broad flavours:

- raster or bitmap graphics, and
- vector graphics.

The former leads to image blockiness or pixellation and loss of definition at high magnifications, as shown in Figure 1, while the latter scales without degradation when magnified, as illustrated in Figure 2.

Figure 1: Raster graphics image of the letter O (150 dpi[1] PNG format).

Raster Graphics

There are dozens of image formats, including these three:

1. Tag(ged) Image File Format (TIFF)

[1] Dots per inch.

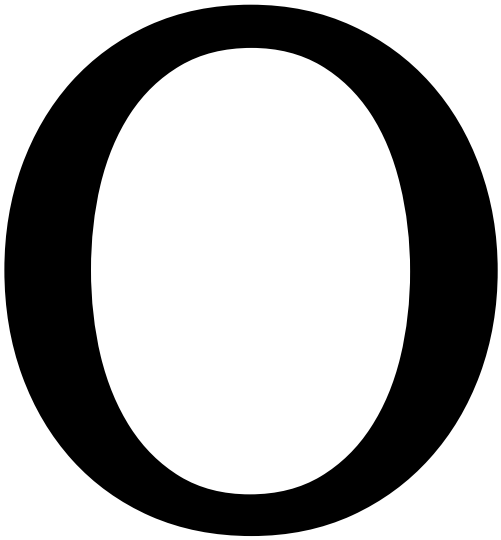

Figure 2: Vector graphics image of the letter O (SVG format).

- lossless compression
- large file sizes
- used in printing and professional graphics
- preferred for archival of scanned photographs

2. Joint Photographic Experts Group (JPEG) format
   - small file sizes
   - lossy compression
   - good quality with fast downloads
   - supported by web browsers
   - preferred for scenes and portraits
   - no transparency
3. Portable Network Graphics (PNG) format
   - lossless compression
   - preferred for text and high definition images
   - supported by most web browsers
   - transparency

All three formats employ raster graphics in which image elements are represented as rectangular arrays of pixels.

Vector Graphics

The two principal vector graphics formats are:

1. Portable Document Format (PDF)
   - preferred for archival quality electronic and printed documents
   - supported by browsers with integrated PDF readers
   - file sizes comparable to raster images
   - machine-readable files
2. Scalable Vector Graphics (SVG) format
   - preferred for scalable graphics on web browsers
   - used in digital image animations and digital art
   - small file sizes
   - human- and machine-readable files

Both these formats yield images which consist of mathematically defined points, lines, curves, and shapes.

Format conversions

For a variety of reasons, it is often necessary to convert from one image format to another. There are four broad possibilities for this:

a. raster to raster;
b. raster to vector;
c. vector to raster; and
d. vector to vector

We consider each of these in turn using platform-neutral open source tools. Since I run GNU/Linux on my desktop, my examples will feature commands from that setup.[2]

Tools for image format conversion

Among the very many tools available, we examine below four that support image format conversion:

1. `ImageMagick`
   - graphics library for image manipulation and display
   - standalone utilities like `convert`, `display`, `identify`, `mogrify`, etc.
   - scripting language
   - pixel-based
   - raster to raster conversions
   - raster to vector conversions
2. cairo
   - vector-based 2D drawing and rendering library
   - multiple output devices/formats
   - used by other programs rather than in standalone mode
3. poppler
   - vector-based PDF rendering library
   - used by several PDF viewers
   - uses cairo as backend
   - standalone utilities like `pdftotext`, `pdftocairo`, and `pdftoppm`
4. Inkscape
   - GUI-based vector graphics editor
   - suitable both for technical illustration and digital art
   - uses SVG as the main format
   - can export to a wide variety of output formats
   - option to use cairo for PDF export

`ImageMagick`: the Swiss Army knife

`ImageMagick` is the name given to a suite of image processing tools originally created in 1987 by John Cristy, then working for Du Pont. In 1990, it was freely released by Du Pont, who transferred copyright to `ImageMagick` Studio LLC who now maintain the project. It is distributed under a derived Apache 2.0 license. The authoritative source code repository shows active development even today, 34 years after the suite was first released [1].

[2] There are many websites that promise conversion online, requiring you to upload the input file and download the output file. These *might be* fraught with security risks. Use them with caution.

`ImageMagick` is so versatile and useful that it may rightfully be called the Swiss Army knife of the image processing world. It comes with several command line utilities, each replete with options. Among these are:

- `convert` which converts from one format to another;
- `display` which displays one or more images;
- `identify` which identifies the type of image and displays its characteristics;
- `mogrify` which transforms an image, modifying its appearance; and
- `montage` which generates an image montage from several images.

The above list is far from exhaustive. The interested reader is referred to the excellent online documentation for further details. The power of `ImageMagick` is enhanced with the magick-script Image Scripting Language. The examples in this blog use the command line versions of invoking `ImageMagick`. If they seem daunting, refer to this explanation [2].

Test images

Two quite different images are used to illustrate the format conversions we perform here. The two test images are:

1. a coloured, text-only test image called `text-only.png`; and
2. a non-text, coloured, graphically rich image called `animals.jpg`.

We will refer to these two images as `text-only` and `animals`, respectively hereafter.

Note that, as explained below, `animals.jpg` is a cropped version of the original image `animals-original.jpg` downloaded from the Web.

Text-only image

The `text-only` image was first generated as a PDF file, `text-only.pdf`, by compiling a LaTeX source file. That file was then converted to various raster formats using the methods discussed later to yield the images `text-only-600-dpi.png` and `text-only-600-dpi.jpg`.

Figure 3: Text-only image in 600 dpi PNG format.

Non-text test image

The non-text image, `animals-original.jpg`, is a colourful, graphically rich image with much detail. It is from a hand-drawn illustration of microscopic marine animals by the German naturalist Ernst Haeckel, scanned as a JPEG, and made available in the public domain.

Pre-processing: Cropping

Cropping is strictly not image format conversion, but is often a necessary pre-processing step in image manipulations. For example, Figure 4 has a whitish, non-monochromatic border around the block print, containing annotations. For our purposes, this border is at best a distraction. It may be removed

[3]These images are in the public domain and covered by the CC0 licence. They are available for download here.

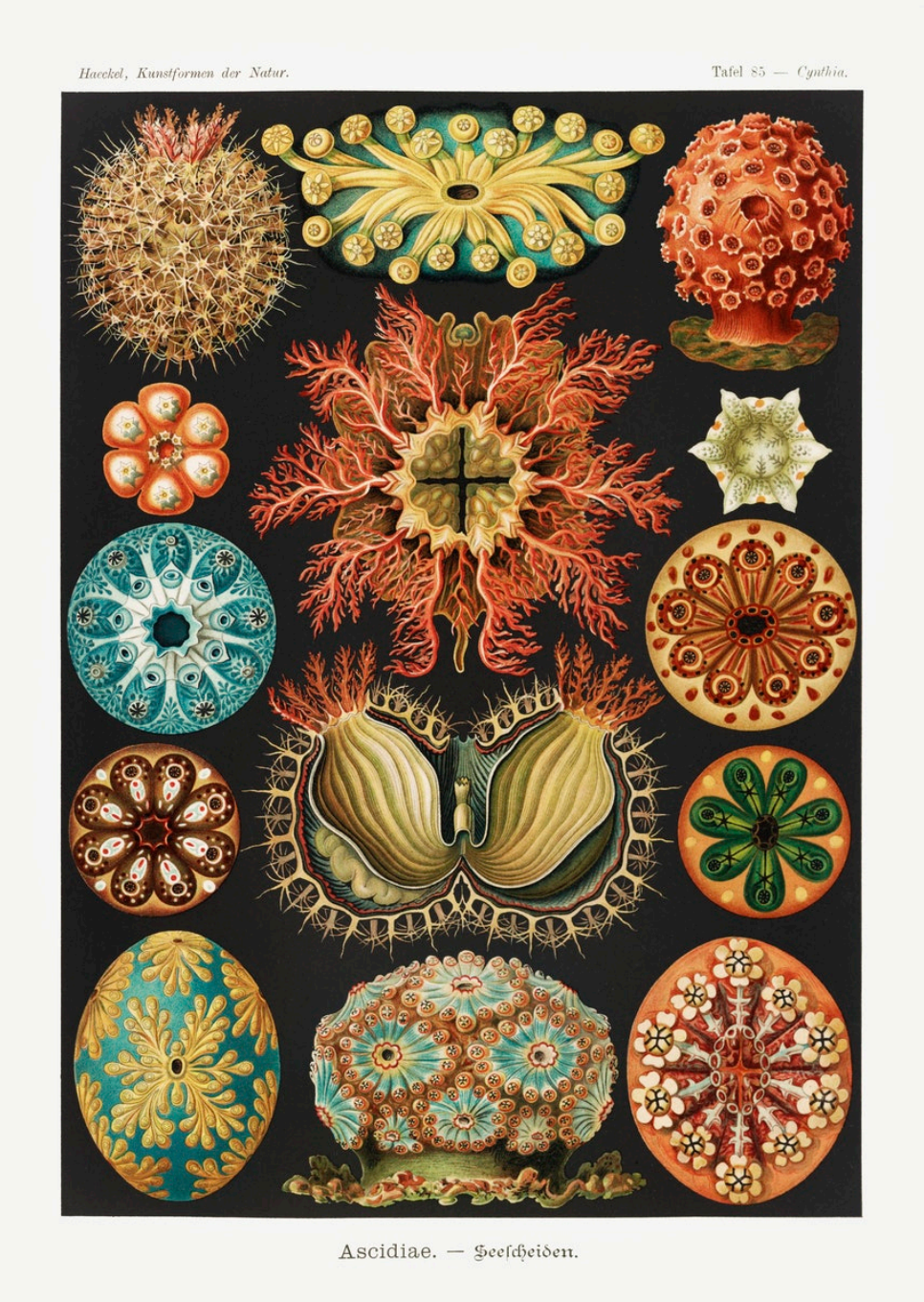

Figure 4: Non-text, graphically rich `animals-original.jpg` image in JPEG format.[3]

altogether by *cropping*, leaving us with only the illustration. We will refer to the resulting cropped image, `animals.jpg`, as `animals`, and use it as the source image in our examples below.

Cropping is usually better done interactively using a GUI (Graphical User Interface), than on the command line. However, the latter, even if a bit tedious, is precisely repeatable.

`ImageMagick`'s `display` utility pops up a GUI when the mouse is over the image and the left mouse button is clicked. We can then drag and fit a window to the *region we wish to keep*, clicking the `Crop` function, and saving the cropped image. The steps are these:

a. left mouse click on the image to reveal the GUI (see Figure 5);
b. `Transform -> Crop`;
c. put the mouse over the top left corner and drag until the bottom right corner to enclose the region of interest;
d. Click again on `Crop`; and
e. `File -> Save` with a different name.

Figure 5: `ImageMagick` interactive GUI.

Alternatively, we may just position the cursor on the top left and bottom right corners of the region we wish to *retain*, noting the co-ordinates in each case. If these coordinates are (x_t, y_t) and (x_b, y_b), respectively, we have $w = x_b - x_t$ and $h = y_b - y_t$. We may then invoke the convert command with crop as the option so:

```
convert -crop 'wxh+x_t+y_t' animals-original.jpg animals.jpg
```

In our case, $(x_t, y_t) = (60, 84)$ and $(x_b, y_b) = (795, 1119)$ giving $w = 735$ and $h = 1035$, leading to

```
convert -crop '735x1035+60+84' animals-original.jpg animals.jpg
```

The resulting cropped image is shown in Figure 6 below.

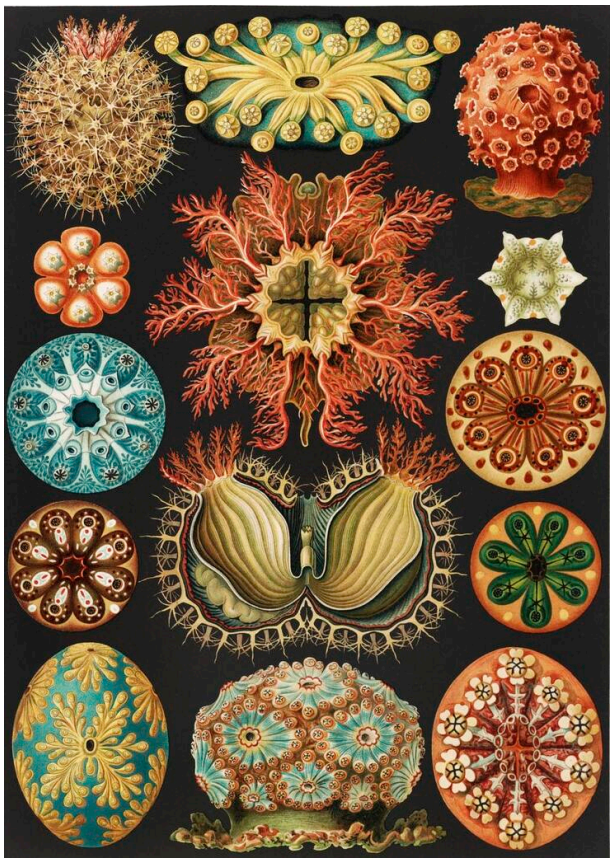

Figure 6: Cropped version of the image in Figure 4. This is the `animals` image.

File sizes

The sizes of the original and cropped files are shown below in human friendly numbers:

```
ls -Xsh animals*.jpg | awk '{print $1 "\t" $2}'
---
200K    animals.jpg
312K    animals-original.jpg
```

As a convention hereafter, when there is a `---` separator between a command and some results, the latter are the results displayed on execution of the command.

As expected, the original file `animals-original.jpg` is larger than the cropped full-size version, `animals.jpg`, and all is well.

Raster to raster conversion

We now perform a sequence of image manipulations, including raster to raster format conversions.

Resizing, format-conversion, and montaging

We may invoke `ImageMagick`'s `convert` function not only to convert from one format to another but also to accomplish cropping (as we have already seen), image-resizing, making the background transparent, and montaging, etc.

Suppose we want to reduce the dimensions of the cropped image to half their original values, and display the full-size and half-size images side by side, we could run the following command:


```
convert animals.jpg -resize 50% animals-halfsize.jpg

# Composite the two images aligning their bottoms
convert +append -gravity south \
animals.jpg animals-halfsize.jpg animals-both.jpg
```

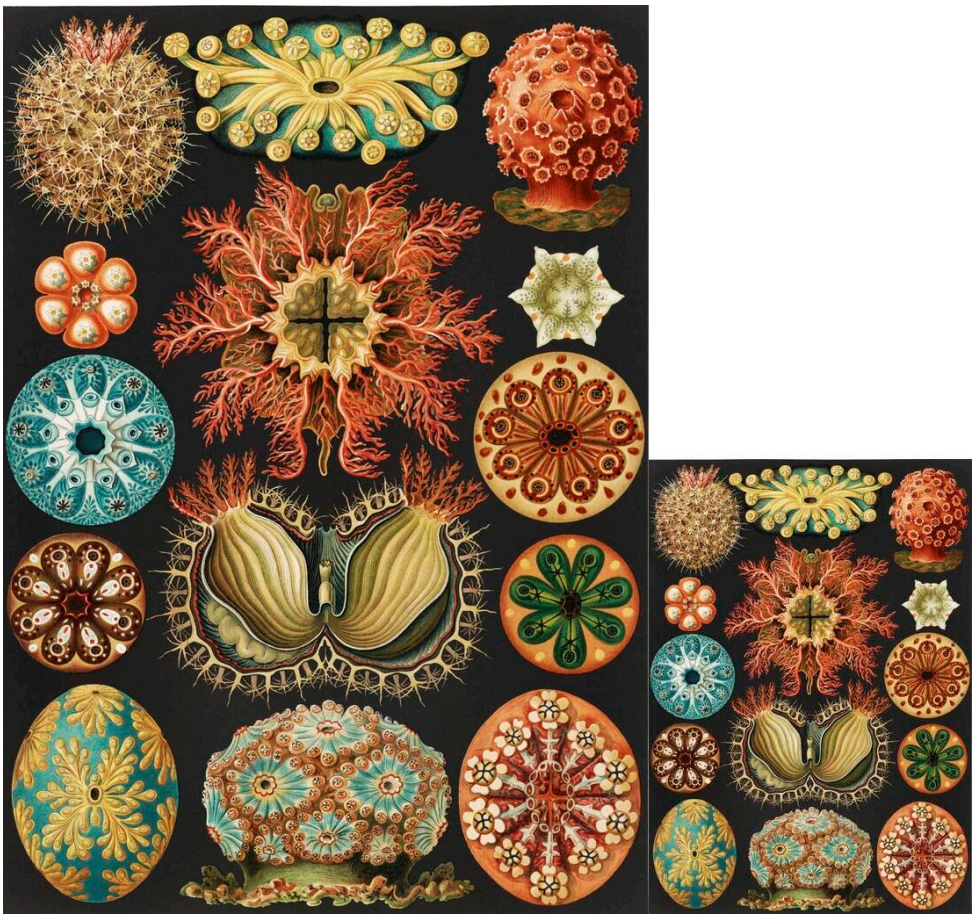

Figure 7: Full-size cropped image on the left and half-sized image on the right.

Background transparency

Notice that there is a coloured white rectangle atop the half-size image on the right in Figure 7. We could remove it by rendering the background transparent. However, because JPEG does not support transparency, through an alpha channel, we have to convert the composite image to the PNG format, which does. This is one, real-life circumstance necessitating raster to raster format conversion.

```
# Non-transparent composite in PNG
convert +append -gravity south \
animals.jpg animals-halfsize.jpg animals-both-non-transparent.png

# Transparent composite in PNG
convert +append -gravity south -background transparent \
animals.jpg animals-halfsize.jpg animals-both-transparent.png
```

File sizes again How do the file sizes of the three composite images compare? How does the non-transparent JPEG compare with the non-transparent PNG? Also, how high a price have we paid for the transparent background?

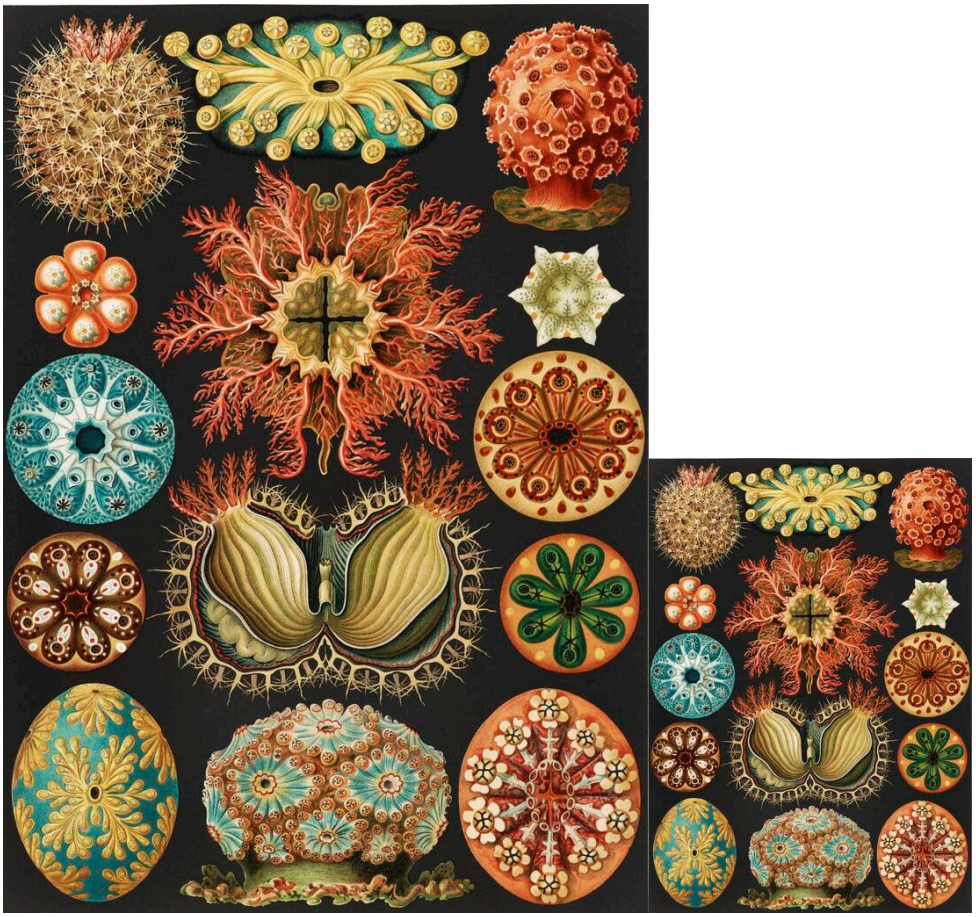

Figure 8: Composite image converted to PNG format with transparent background.

```
ls -Xsh animals-both*| awk '{print $1 "\t" $2}'
---
264K    animals-both.jpg
2.0M    animals-both-non-transparent.png
2.2M    animals-both-transparent.png
```

The PNG composite image is *more than seven times larger* than its JPEG counterpart, *even without transparency*. And transparency makes the PNG file size ten percent larger.

Compression levels and file sizes The image compression level used above is the default compression level in `ImageMagick`. Getting the right combination of image format, image dimensions, image compression, and image quality so that the image loads fast and looks good is quite an art. [3]

To get an idea of the range of file sizes involved, let us try generating a composite image with extremes of the compression level, which can range from 0 to 9.

```
# Least compression
convert -define PNG:compression-level=0 +append -gravity south \
-background transparent \
animals.jpg animals-halfsize.jpg \
animals-both-compressed-0.png

# Most compression
convert -define PNG:compression-level=9 +append -gravity south \
-background transparent \
animals.jpg animals-halfsize.jpg \
animals-both-compressed-9.png
```

The file sizes are:

```
ls -Xsh animals-both* | awk '{print $1 "\t" $2}'
---
264K    animals-both.jpg
4.4M    animals-both-compressed-0.png
2.2M    animals-both-compressed-9.png
2.0M    animals-both-non-transparent.png
2.2M    animals-both-transparent.png
```

It appears that the default compression used by `ImageMagick` gives a file size that is the same as the highest compression level. Indeed, the uncompressed version—with a compression level of zero—gives a file *twice* the size of the uncompressed version and *sixteen times* the size of the JPEG. And we have not even used two other related attributes: filter and strategy [4]. Getting the best tradeoff of image format, image size, file size, loading time, and image quality is still more of an art to be mastered than an algorithm to be applied.

We may conclude from the above that non-textual, detail-rich images are better stored and displayed as JPEGs than PNGs.

Results with the text-only image

For completeness, let us do a simple *no quality loss* conversion from PNG to JPEG for the `text-only` test image, and compare image appearances and file sizes.

```
# Lossless JPEG with a 'quality' of 100
convert -quality 100 text-only-600-dpi-cairo.png \
text-only-600-dpi-cairo.jpg
```



```
# Composite both images into one with a transparent divider
convert text-only-600-dpi-cairo.png text-only-600-dpi-cairo.jpg \
-background transparent -splice 20x0+0+0 +append -chop 20x0+0+0 \
text-only-both-600-dpi-cairo.png

ls -Xsh text*cairo* | awk '{print $1 "\t" $2}'
---
148K    text-only-600-dpi-cairo.jpg
40K     text-only-600-dpi-cairo.png
120K    text-only-both-600-dpi-cairo.png
```

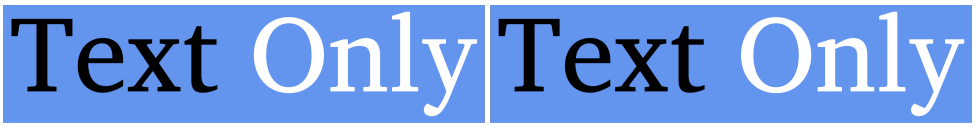

Figure 9: Composite PNG image of the PNG on the left, and JPEG on the right.

Figure 9 does not reveal any degradation in quality after conversion from PNG to JPEG. Note also that the file size of the *composite* PNG image is smaller than the file size of the *single* JPEG image.

PNG versus JPEG

We conclude from the `text-only` images that PNG is better suited for textual images and provides a smaller file size for the same quality.

Conversely, we know from the `animals` images that JPEG is more suited to non-textual detail and yields good quality images at far smaller file sizes than PNG.

Can `cairo` and `poppler` do all this?

Can such processing be done using `cairo` or `poppler`? Not really, in the case of the `animals` image. The *starting point* or *input format* for `cairo` and `poppler` is the PDF format. Our `animals` image is scanned from an illustration and is therefore a JPEG raster image. `ImageMagick`'s forte is the display, manipulation, and processing of raster images; `cairo` and `poppler` start with PDFs and have other goals.

Raster to vector conversions

Let us say that we have a logo, designed and available as a raster image. To use it on the Web, we could, if necessary, reformat it as a JPEG or PNG file. But as we zoom into the page, the raster images will start becoming less sharp and more blocky as shown in Figure 1.

However, if the graphic were in SVG format, supported by most web browsers, the logo would scale without visual degradation as we zoom into the page.

How do we convert a raster image to a vector format like PDF or SVG?

Raster to PDF with `convert`

We could do this using `ImageMagick`'s `convert` utility again. For example,

```
# Convert JPEG image to PDF
convert animals.jpg animals.pdf

ls -Xsh animals.jpg animals.pdf | awk '{print $1 "\t" $2}'
---
200K    animals.jpg
204K    animals.pdf
```


The converted image, animals.pdf, may be viewed from the given link. Web browsers, while they may feature PDF viewers on separate tabs, are still unable to display PDFs as part of a web page. If the converted PDF is magnified by zooming, it will be seen to reveal remarkable detail. And the difference between the JPEG and PDF file sizes is negligible.

What happens, though, if the half-sized image is used to generate the PDF? It is smaller and accordingly embodies less information than the original.

```
convert animals-halfsize.jpg animals-halfsize.pdf

ls -Xsh animals-halfsize.jpg animals-halfsize.pdf | \
awk '{print $1 "\t" $2}'
---
64K     animals-halfsize.jpg
64K     animals-halfsize.pdf
```

Again, no surprises here. The half-sized PDF tracks the file size of the half-sized JPEG.

Raster to SVG with `convert`

Let us try to harness `ImageMagick`'s `convert` again, this time to change from JPEG to SVG:

```
convert animals.jpg animals.svg

ls -Xsh animals.jpg animals.svg | awk '{print $1 "\t" $2}'
---
200K    animals.jpg
436K    animals.svg
```

The SVG file is more than *twice* the size of the original JPEG. The question arises whether there is an alternative route to the SVG that could give us smaller file sizes but comparable fidelity. What if we did not convert from raster to SVG but from raster to PDF and thence to SVG?

Since PDF to SVG conversion is really part of vector to vector conversion, we will revisit this question later.

We will not consider the `text-only` image here as that image originated not as a raster but as a PDF in the first place.

Vector to raster

The `poppler` utilities, with the `cairo` backend are the primary resource here. Because the `text-only` image has been produced natively in PDF, we will use it as the the example for these conversions.

PDF to PNG and JPEG: `poppler` and `cairo`

It was mentioned above that `text-only` was originally generated as a PDF, vector graphics image and subsequently converted to the PNG and JPEG formats. We explain how that was done and also why the `ImageMagick` suite is not used for this purpose.

The `poppler` suite contains utilities to convert from PDF to several raster formats. Two versatile utilities called `pdftocairo` and `pdftoppm` are available for our purpose. One may view their usage by typing the name of the utility prefixed by `man` or suffixed by `-help`, although the former is more exhaustive.

```
pdftocairo -png -r 600 -singlefile text-only.pdf \
text-only-600-dpi-cairo

pdftoppm -png -r 600 -singlefile text-only.pdf \
```



```
text-only-600-dpi-ppm

pdftocairo -jpeg -jpegopt "quality=100" -r 600 \
-singlefile text-only.pdf text-only-600-dpi-cairo

pdftoppm -jpeg -jpegopt "quality=100" -r 600 \
-singlefile text-only.pdf text-only-600-dpi-ppm

convert -units pixelsperinch -density 600 -quality 100 \
text-only-600-dpi-cairo.png text-only-600-dpi-cairo-IM.jpg

convert -units pixelsperinch -density 600 -quality 100 \
text-only-600-dpi-ppm.png text-only-600-dpi-ppm-IM.jpg
```

The value `-r 600` signifies a resolution of 600 pixels per inch (PPI), or alternatively, dots per inch (dpi). The default value is 150 PPI. The value of 600 is suitable for printing on laser printers to give output that will visually rival the original PDF in quality. Note that while raster images have inherent resolutions, PDF images have none: they scale without loss of quality.

The `-singlefile` option is used because we are simply converting a single "page" of PDF rather than a numbered page sequence. In all cases, the destination filename is the "root" of the converted file sequence, which in this case is the filename without any extension.

In addition, the JPEG version may feature lossy compression where quality is traded for file size. Since PNG is lossless, to compare the two formats on an even keel, we specify that the `-quality` of the JPEG should be the maximum of 100.

Both `pdftocairo` and `pdftoppm` are used in the conversions above, with appropriately named filenames.

We could also use `ImageMagick`'s `convert` to convert from PNG to JPEG, and this is done in the last two commands above. Note that this is strictly not a vector to raster conversion but merely raster to raster. See below for why we cannot convert from PDF to raster with `convert`.

The files sizes that result are shown below:

```
ls -Xsh text-only*| awk '{print $1 "\t" $2}'
---
148K    text-only-600-dpi-cairo-IM.jpg
120K    text-only-600-dpi-cairo.jpg
140K    text-only-600-dpi-ppm-IM.jpg
112K    text-only-600-dpi-ppm.jpg
16K     text-only.pdf
40K     text-only-600-dpi-cairo.png
40K     text-only-600-dpi-ppm.png
```

The numbers tell their own story. I would have expected the two sets of raster images output by `pdftocairo` and `pdftoppm` to be roughly equal in size, given their identical options during invocation. Strangely, they are not, at least for the JPEGs. This could be either because of different defaults, or different algorithms, or something else: I simply do not know.

It appears that `pdftoppm` gives marginally smaller file sizes for JPEG than `pdftocairo`. Moreover, when `pdftoppm` is used to convert directly from PDF to JPEG, the file size is smaller than when PNG is used as an intermediate file format and conversion to JPEG is by `ImageMagick`'s `convert`.

One other takeaway is that text-rich images are better rendered in PNG than JPEG. The PDF and PNG image file sizes are of the same order of magnitude, whereas the JPEGS are an order of magnitude larger.

Why is ImageMagick disallowed for PDF to raster?

If you try to convert a PDF to any raster image format, you will get an error:

```
convert text-only.pdf text-only.png
---
convert: attempt to perform an operation not allowed by the security policy `gs' @ error/delegate.c/Ex
convert: no images defined `text-only.png' @ error/convert.c/ConvertImageCommand/3304.
```

The reason why this is now disallowed is [explained in the appendix][Appendix: ImageMagick's security vulnerabilities].

SVG to PNG and JPEG

At present, most converters from SVG to other formats do not support fonts as well as PDF does. Bear this deficit in mind as SVG to raster converters are invoked. Both ImageMagick and Inkscape are able to convert SVG to PNG. Command line invocations for each and their resultng file sizes are detailed below. Note that the cairo backend might be used in many of these utilities.

%%% Clean up below %%%

The Inkscape GUI-based vector graphics editor supports SVG as its native format and allows export of the generated SVG graphics both as PDF and as PNG. Let us use the text-only image as an example and convert it from its original PDF format to an SVG using pdftocairo:

```
pdftocairo -svg text-only.pdf text-only.svg

ls -Xsh text-only.*| awk '{print $1 "\t" $2}'
---
16K     text-only.pdf
12K     text-only.svg
```

We could open up text-only.svg in Inkscape and save it as a PDF, text-only-inkscape.pdf completing the round trip. Or we could use the command line so:

```
inkscape text-only.svg -o text-only-inkscape.pdf

ls -Xsh text-only*.{svg,pdf}| awk '{print $1 "\t" $2}'
---
8.0K    text-only-inkscape.pdf
16K     text-only.pdf
12K     text-only.svg
```

What exactly has been gained or lost in this round trip is a little too recondite to consider here, and covers issues such as PDF version, font embedding or its absence, default borders, etc.

More to the point, though, is how do we convert from SVG to PNG on the command line using inkscape?

What about SVG to PNG? CairoSVG Inkscape SVG to PNG? ImageMagick SVG to PNG?

The cairosvg module offers 4 functions:

```
svg2pdf,
svg2png,
svg2ps, and
svg2svg.
```

CairoSVG is designed to parse well-formed SVG files, and draw them on a Cairo surface. Cairo is then able to export them to PDF, PS, PNG, and even SVG files.

Vector to vector

There are principally two format conversions here:

- PDF to SVG; and
- SVG to PDF.

We look at each in turn below.

PDF to SVG: `pdftocairo` and `pdftoppm`

The `poppler` standalone utilities `pdftocairo` and `pdftoppm` offer a route from PDF to SVG. So, if we had the file `animals.pdf` we could convert it to SVG:

```
# PDF to SVG using pdftocairo
pdftocairo -svg animals.pdf animals-pdftocairo.svg

ls -Xsh animals.pdf animals-pdftocairo.svg | awk '{print $1 "\t" $2}'
---
204K    animals.pdf
268K    animals-pdftocairo.svg
```

Two observations are pertinent:

1. It is preferable to convert from a raster image to PDF using `convert` and then convert that PDF to SVG using `pdftocairo` rather than convert directly from a raster to an SVG using `convert` in a single step. The file sizes that result are very much smaller with the two-stage conversion.
2. `pdftoppm` is not set up to convert *to* SVG, and hence cannot be used.

`pdf2svg`

There is a utility called `pdf2svg` that has been available for some time now. It may be used to accomplish the same PDF to SVG conversion as `pdftocairo`:

```
# PDF to SVG using pdf2svg
pdf2svg animals.pdf animals-pdf2svg.svg

ls -Xsh animals.pdf animals-pdf2svg.svg | awk '{print $1 "\t" $2}'
---
204K    animals.pdf
268K    animals-pdf2svg.svg
```

The file sizes are identical to those from the `pdftocairo` conversion. Drew Barton, the author of `pdf2svg`, has written:

> Note: since this utility was written, the maintainers of Poppler have written a utility that works on the same principle: `pdftocairo`. I recommend that you use their utility since it is better maintained than mine.

So, it appears that `pdftocairo` is sufficient for converting from PDF to SVG.

`rsvg-convert` `pdftocairo`

https://en.wikipedia.org/wiki/Cairo_(graphics)

https://www.cairographics.org/

https://cgit.freedesktop.org/cairo

Inkscape

https://wiki.inkscape.org/wiki/index.php/Tools

pdf2svg

https://cityinthesky.co.uk/opensource/pdf2svg/

https://github.com/dawbarton/pdf2svg

https://inkscape.org/develop/about-svg/

poppler

`pdftoppm`

pdftocairo

Raster to raster

Image to PDF

Still works. No strictures. But the PDF can get grungy. Use a pyramid of resolutions.

Avoiding blurry PDFs

-units pixelsperinch -density 1200 etc., in conversion

Useful when a hgh resolution image is available. In any case: PDF and png/jpg sizes are similar.

Choosing the optimal image resolution for a clear PDF

96dpi for screen 150 dpi default 300 dpi for print [give references]

```
convert -units pixelsperinch -density 300 file.png file.pdf
```

PDF to image not supported

```
#! /bin/magick
convert file.pdf file.png
```

```
convert test.pdf test.png
---
convert: unable to open image 'test.pdf': No such file or directory @ error/blob.c/OpenBlob/3537.
convert: no images defined `test.png' @ error/convert.c/ConvertImageCommand/3304.
```

Security considerations

`ImageMagick` is no more the famed Swiss army knife for conversions from PDFs to images.

Give references to security concerns.

Enter poppler

Vector to raster

It was mentioned above that `text-only` was originally generated as a PDF, vector graphics image and subsequently converted to the PNG and JPEG formats. We explain how that was done and also why the `ImageMagick` suite is not used for this purpose.

poppler and cairo

The poppler suite contains utilities to convert from PDF to several raster formats. Two versatile utilities called pdftocairo and pdftoppm are available for our purpose. One may view their usage by typing the name of the utility prefixed by man or suffixed by -help, although the former is more exhaustive.

```
pdftocairo -png -r 600 -singlefile text-only.pdf \
text-only-600-dpi-cairo

pdftoppm -png -r 600 -singlefile text-only.pdf \
text-only-600-dpi-ppm

pdftocairo -jpeg -jpegopt "quality=100" -r 600 \
-singlefile text-only.pdf text-only-600-dpi-cairo

pdftoppm -jpeg -jpegopt "quality=100" -r 600 \
-singlefile text-only.pdf text-only-600-dpi-ppm

convert -units pixelsperinch -density 600 -quality 100 \
text-only-600-dpi-cairo.png text-only-600-dpi-cairo-IM.jpg

convert -units pixelsperinch -density 600 -quality 100 \
text-only-600-dpi-ppm.png text-only-600-dpi-ppm-IM.jpg
```

The value -r 600 signifies a resolution of 600 pixels per inch (PPI). The default value is 150 PPI. The value of 600 is suitable for printing on laser printers to give output that will visually rival the original PDF in quality. Note that while raster images have inherent resolutions, PDF images have none: they scale without loss of quality.

The -singlefile option is used because we are simply converting a single "page" of PDF rather than a numbered page sequence. In all cases, the destination filename is the "root" of the converted file sequence, which in this case is the filename without any extension.

In addition, the JPEG version may feature lossy compression where quality is traded for file size. Since PNG is lossless, to compare the two formats on an even keel, we specify that the -quality of the JPEG should be the maximum of 100.

Both pdftocairo and pdftoppm are used in the conversions above, with appropriately named filenames. One could also use ImageMagick's convert to convert from PNG to JPEG, and this is done in the last two commands.

The files sizes that result are shown below:

```
ls -Xsh text-only*| awk '{print $1 "\t" $2}'
---
148K    text-only-600-dpi-cairo-IM.jpg
120K    text-only-600-dpi-cairo.jpg
140K    text-only-600-dpi-ppm-IM.jpg
112K    text-only-600-dpi-ppm.jpg
16K     text-only.pdf
40K     text-only-600-dpi-cairo.png
40K     text-only-600-dpi-ppm.png
```

The numbers tell their own story. I would have expected the two sets of raster images output by pdftocairo and pdftoppm to be roughly equal in size, given their identical options during invocation. Strangely, they are not, at least for the JPEGs. This could be either because of different defaults, or different algorithms, or something else: I simply do not know.

It appears that pdftoppm gives marginally smaller file sizes for JPEG than pdftocairo. Moreover, when pdftoppm is used to convert directly from PDF to JPEG, the file size is smaller than when PNG is used as an intermediate file format and conversion to JPEG is by ImageMagick's convert.

One other takeaway is that text-rich images are better rendered in PNG than JPEG. The PDF and PNG image file sizes are of the same order of magnitude, whereas the JPEGS are an order of magnitude larger.

Why is `ImageMagick` disallowed for PDF to raster?

If you try to convert a PDF to any raster image format, you will get an error:

```
convert text-only.pdf text-only.png
---
convert: attempt to perform an operation not allowed by the security policy `gs' @ error/delegate.c/Ex
convert: no images defined `text-only.png' @ error/convert.c/ConvertImageCommand/3304.
```

The reason why this is now disallowed is [explained in the appendix][Appendix: `ImageMagick`'s security vulnerabilities].

SVG to raster

What about SVG to PNG? CairoSVG Inkscape SVG to PNG? `ImageMagick` SVG to PNG?

The cairosvg module offers 4 functions:

```
svg2pdf,
svg2png,
svg2ps, and
svg2svg.
```

CairoSVG is designed to parse well-formed SVG files, and draw them on a Cairo surface. Cairo is then able to export them to PDF, PS, PNG, and even SVG files.

Vector to vector

There are basically two possibilities:

a. PDF to SVG; and
b. SVG to PDF.

Both are possible with the `cairo` library and `poppler` suite as well as other libraries and utilities.

PDF to SVG

pdftocairo -svg pdftoppm -svg? pdf2svg Any others?

SVG to PDF

CairoSVG https://cairosvg.org/: single executabe cairosvg from python-cairosvg: safety rsvg-convert from librsvg Inkscape `ImageMagick` any others?

https://wiki.gnome.org/Projects/LibRsvg

```
pdftoppm -png ernst-heackel-medium.pdf ernst-heackel-medium.png
convert ernst-heackel-medium.jpg ernst-heackel-medium-direct.png
convert ernst-heackel-medium.jpg ernst-heackel-medium-direct.png
```

How to use resize etc.

Summary

`ImageMagick`'s `convert` is the tool of choice for converting from any raster format to another raster format or to PDF or SVG.

When we start out with PDF as the source image format, and the destination format is either a raster format or SVG, the tool of choice is `pdftocairo` from the `poppler` utilities.

When the source format is SVG and the destination format is either PDF or a raster format, the tool of choice is `cairosvg`.

*__tbl:formats?__ summarizes this information, which is current at the time of writing, but could change as image utilities landscape changes with time.

Table 1: Tools for image format conversions. {#tbl:formats}

Conversion Type	Tool
raster to raster	`convert`
raster to PDF	`convert`
raster to SVG	`convert`
PDF to raster	`pdftocairo`
SVG to raster	`cairosvg`
PDF to SVG	`pdftocairo`
SVG to PDF	`cairosvg`

Appendix: Security vulnerabilities in `ImageMagick`

Great power exacts a commensurate price. `ImageMagick`'s great power and ease of use does come at a great price: vulnerability to exploits by malicious remote actors.

`ImageMagick` uses external libraries or *backend tools* which are called via `system()` commands in accordance with *delegated* command strings specified in a configuration file called `policy.xml`.

In April 2016, it was reported that because of insufficient validation of delegated command strings, it was possible for someone to execute malicious code remotely, to the detriment of the unwitting user of `ImageMagick`. This was revealed at a website, interestingly named ImageTragick to attract sufficient attention and remedial action to the discovered bug [5].

In November 2020, another security vulnerability was discovered [6]. It was reported and promptly patched by the `ImageMagick` maintainers [7].

Recent versions of the `ImageMagick` suite, bundled with major distributions, should have correctly configured `policy.xml` files that will block known exploits. Sandboxing is another technique to quarantine the system from possible vulnerabilities. Above all, it is vital to keep system and application software up to date to avail of evolutions in performance and security.

Feedback Please email me your comments and corrections.

A PDF version of this article is available for download here.

References

[1] ImageMagick Studio LLC, 'ImageMagick 7.' [Online]. Available: https://github.com/ImageMagick/ImageMagick. [Accessed: 08-Mar-2021]

[2] 'ImageMagick—Command-line Processing. Anatomy of the Command-line.' [Online]. Available: https://imagemagick.org/script/command-line-processing.php. [Accessed: 12-Mar-2021]

[3] D. Newton, 'Efficient Image Resizing With ImageMagick—Smashing Magazine,' 25-Jun-2015. [Online]. Available: https://www.smashingmagazine.com/2015/06/efficient-image-resizing-with-imagemagick/. [Accessed: 11-Mar-2021]

[4] M. Setchell, 'ImageMagick: Lossless max compression for PNG?—Stack Overflow,' 03-Dec-2014. [Online]. Available: https://stackoverflow.com/questions/27267073/imagemagick-lossless-max-compression-for-png/27269260#27269260. [Accessed: 12-Mar-2021]

[5] —, 'ImageMagick is on fire—CVE-2016–3714,' 12-May-2016. [Online]. Available: https://imagetragick.com/. [Accessed: 08-Mar-2021]

[6] J. Leyden, 'ImageMagick PDF-parsing flaw allowed attacker to execute shell commands via maliciously crafted image,' 23-Nov-2020. [Online]. Available: https://portswigger.net/daily-swig/imagemagick-pdf-parsing-flaw-allowed-attacker-to-execute-shell-commands-via-maliciously-crafted-image. [Accessed: 08-Mar-2021]

[7] A. Inführ, 'ImageMagick - Shell injection via PDF password,' 21-Nov-2020. [Online]. Available: https://insert-script.blogspot.com/2020/11/imagemagick-shell-injection-via-pdf.html. [Accessed: 08-Mar-2021]

