**What is rendering**

A figure in svg format is stored as xml data containing a vector representation of a drawing, such as a sequence of lines, shapes, text, with their relative position, size, color, attributes, etc. The benefit of svg is that it can be resized without loss of quality. And because it is just xml, the shapes and text can be manipulated using standard xml/css tools, such as a browser or the xml2 package.

For an image to be displayed on screen, printed in a document, or loaded in editing software, it has to be *rendered* into a bitmap. A bitmap is a fixed a array of w × h pixels with color values. Bitmap formats such as png, jpeg, or tiff all store the same pixel data, using different compression methods.

The rsvg package renders svg into a bitmap image with the format and size of your choice, directly in R, and without loss of quality:

# Example SVG image

svgdata <- '

I love SVG!

'

# Render with rsvg into png

writeLines(svgdata, 'image.svg')

rsvg::rsvg\_png('image.svg', 'image.png', width = 800)

Instead of rendering to a png/jpeg file, you can also render the svg into raw bitmap data (called raw vectors in R), which you can read with for example magick or any other imaging tool:

# Or: convert into raw bitmap data

bitmap <- rsvg\_raw('image.svg', width = 600)

str(bitmap)

##> raw [1:4, 1:600, 1:600]

# Read the bitmap in magick

image <- magick::image\_read(bitmap)

I love SVG!

In magick, you can easily do all sorts of post-processing and conversion of the bitmap image. The magick package has a convenient wrapper function read\_image\_svg that does exactly this: it uses rsvg to render the image and then reads the bitmap data as a magick image.

**Using SVG for R graphics**

The best way to create svg files from graphics in R is using the svglite package. Try running the code below and then have a look at mtcars.svg in a text editor.

library(svglite)

library(ggplot2)

# SVG sizes are in inches, not pixels

res <- 144

svglite("mtcars.svg", width = 1080/res, height = 720/res)

ggplot(mtcars, aes(mpg, disp, colour = hp)) + geom\_point() + geom\_smooth()

dev.off()

Again we can use rsvg directly or via magick to convert this to a bitmap image:

# Render the svg into a png image with rsvg via magick

img <- magick::image\_read\_svg("mtcars.svg", width = 1080)

magick::image\_write(img, 'mtcars.png')

This generates a png image of with 1080x720px, without loss of quality.

**Using CSS for R graphics?**

One feature in librsvg that has improved a lot from servo is support for CSS. As can be seen in the example above, svg allows for specifying global styling via CSS rules. In the browser, CSS and JavaScript can also be used to add interactivity and animation to SVG.

With the latest version of librsvg it is now also possible to specify the CSS stylesheet from an external file, rather than inlining it in the svg itself. For example you can have a fig.svg file like this:

viewBox="0 0 1200 250" xmlns="<http://www.w3.org/2000/svg>">

cx="200" cy="125" r="120" />

x="140" y="40" transform="rotate(30 10,20)">Separate CSS!

And a separate style.css file like this:

circle {

fill: gold;

stroke: maroon;

stroke-width: 12px;

}

text {

fill: navy;

font-size: 2em;

font-family: "Times, Serif"

}

Which you would render in R like this to get the same figure as above.

rsvg\_png('fig.svg', css = 'style.css', file = 'output.png')

So is this useful? Maybe, I’m not sure. The R graphics system is pretty old, it currently doesn’t have any notion of separating style from layout like we do in modern webpages. It could be useful to think about which styling properties of graphics could be decoupled from the figure structure.

D3’s vocabulary of graphical marks comes directly from web standards: HTML, SVG, and CSS. For example, you can create SVG elements using D3 and style them with external stylesheets. You can use composite filter effects, dashed strokes and clipping. If browser vendors introduce new features tomorrow, you’ll be able to use them immediately—no toolkit update required. And, if you decide in the future to use a toolkit other than D3, you can take your knowledge of standards with you!

Maybe not everything generalizes directly to R, but some aspects do. One could imagine it would be useful to specify fonts and color palettes in the rendering phase, rather than hardcoding these in the graphic. Or that the same svg file would work in dark-mode, or with accessibility styling. For this to work, the graphics device would have to add support for tagging shapes and textboxes with a class or id, such that these can be selected using xpath, css or javascript.

I think that if we can untangle these things in the graphics device, it may be possible to produce R graphics as objects that can both be rendered into bitmaps for printing, but at the same time allow for interactivity and animation in the browser.