At rOpenSci we are developing on a suite of packages that expose powerful graphics and imaging libraries in R. Our latest addition is [av](https://github.com/ropensci/av#readme) – a new package for working with audio/video based on the [FFmpeg](https://www.ffmpeg.org/) AV libraries. This ambitious new project will become the video counterpart of the [magick](https://cloud.r-project.org/web/packages/magick/vignettes/intro.html) package which we use for working with images.

install.packages("av")

av::av\_demo()

The package can be installed directly from CRAN and includes a test function av\_demo() which generates a demo video from random histograms.

**Why AV in R?**

One popular application is animating graphics by combining a sequence of graphics into a video. The [animation](https://cran.r-project.org/web/packages/animation/index.html) and [gganimate](https://github.com/thomasp85/gganimate) packages have many great examples. However up till now these packages would have to shell out to external software (such as the ffmpeg command line program) to generate the video. This process is inefficient and error prone and requires that the correct version of the external software is installed on the user/server machines, which is often not the case.

The av package takes away this technical burden. It uses the same libraries as FFmpeg, however because we interface directly to the C API, there is no need to shell out or install utilities. Everything we need is linked into the R package, which means that if the package is installed, it always works.

FFmpeg provides a full-featured video editing library, and now that the core package is in place, we can take things a step further. For example you can already enhance an animation with an audio track (to narrate what is going on or show off your karaoke skills) or apply one of the 100+ built-in [video filters](https://ffmpeg.org/ffmpeg-filters.html#Video-Filters). In future versions we also want to add things like screen capturing and reading raw video frames and audio samples for analysis in R.

**Create video from images**

The av\_encode\_video() function converts a set of images (png, jpeg, etc) into a video file with custom container format, codec, fps, and filters. The video format is determined from the file extension (mp4, mkv, flv, gif). Av supports all popular codecs and muxers (codecs compress the raw audio/video and a muxer is the container format which interleaves one or more audio and video streams into a file).

# Create some PNG images

png("input%03d.png", width = 1280, height = 720, res = 108)

for(i in 1:10){

print(ggplot2::qplot(rnorm(100)))

}

dev.off()

png\_files <- sprintf("input%03d.png", 1:10)

av::av\_encode\_video(png\_files, 'output.mp4', framerate = 1)

utils::browseURL('output.mp4')

Your browser does not support the video tag.

**Create video from graphics**

The example above illustrates how to generate encode a set of png images into a HD video. For generating a video from the R graphics, av includes av\_capture\_graphics() – a convenient wrapper which automatically opens and closes the graphics device and then encodes the video:

library(gapminder)

library(ggplot2)

makeplot <- function(){

datalist <- split(gapminder, gapminder$year)

lapply(datalist, function(data){

p <- ggplot(data, aes(gdpPercap, lifeExp, size = pop, color = continent)) +

scale\_size("population", limits = range(gapminder$pop)) + geom\_point() + ylim(20, 90) +

scale\_x\_log10(limits = range(gapminder$gdpPercap)) + ggtitle(data$year) + theme\_classic()

print(p)

})

}

# Play 1 plot per sec

video\_file <- file.path(tempdir(), 'output.mp4')

av::av\_capture\_graphics(makeplot(), video\_file, 1280, 720, res = 144)

utils::browseURL(video\_file)

Your browser does not support the video tag.

**Using av with gganimate**

The latest version of the super cool gganimate package already has built-in support for rendering video with av by using the av\_renderer() output function. It is not yet on CRAN so you need to install from GitHub:

# Install gganimate

devtools::install\_github("thomasp85/gganimate", "thomasp85/transformr")

Try this example animation:

library(gganimate)

# Create the gganimate plot

p <- ggplot(airquality, aes(Day, Temp)) +

geom\_line(size = 2, colour = 'steelblue') +

transition\_states(Month, 4, 1) +

shadow\_mark(size = 1, colour = 'grey')

# Render and show the video

df <- animate(p, renderer = av\_renderer('animation.mp4'),

width = 1280, height = 720, res = 104, fps = 25)

utils::browseURL('animation.mp4')

Your browser does not support the video tag.

Check the [gganimate repo](https://github.com/thomasp85/gganimate) for many cool examples.

**Filters**

AV also allows for adding a custom [video filter chain](https://ffmpeg.org/ffmpeg-filters.html#Video-Filters). For example this will use the same animation as above, then negate the colors, and apply an orange fade-in effect to the first 15 frames.

# Continue on the example above

myrenderer <- av\_renderer('animation.mp4',

vfilter = 'negate=1, fade=in:0:15:color=orange')

df <- animate(p, renderer = myrenderer,

width = 1280, height = 720, res = 104, fps = 25)

utils::browseURL('animation.mp4')

Your browser does not support the video tag.

Note that filters can affect the number of frames and final framerate of the video. Below the same example as we had earlier, however now we added a vfilter which increases the framerate of the video from 1 to 10 by interpolating the intermediate frames, which results in a smoother transition between the frames:

library(gapminder)

library(ggplot2)

makeplot <- function(){

datalist <- split(gapminder, gapminder$year)

lapply(datalist, function(data){

p <- ggplot(data, aes(gdpPercap, lifeExp, size = pop, color = continent)) +

scale\_size("population", limits = range(gapminder$pop)) + geom\_point() + ylim(20, 90) +

scale\_x\_log10(limits = range(gapminder$gdpPercap)) + ggtitle(data$year) + theme\_classic()

print(p)

})

}

# Play 1 input plot per sec, interpolate frames until 10 FPS

av::av\_capture\_graphics(makeplot(), 'smooth.mp4', 1280, 720, res = 104,

vfilter = 'framerate=fps=10')

utils::browseURL('smooth.mp4')