Camilo Garcia La Rotta
Montréal, Canada

camilo.garcia.larotta@gmail.com
camilogarcialarotta.github.io
Software Engineer @ GitHub

Subject: go get -u community CFP - lightning talk proposal (10 mins)

Code: https://github.com/CamiloGarciaLaRotta/gifhub
Presentation: https://camilogarcialarotta.github.io/presentations/talks/gifs-in-go

NB: you can press **P** in the slides to view the presenter notes.

Good afternoon,

I learned a couple of hours ago about this virtual conference. I I realize this proposal comes after the defined deadline, but I hope it at least might make it through to the filtering round.

I have been tinkering with Go both professionally and as a hobby for about 2 years. A while back I wrote a small Go CLI application which generates GIFs. The app showcases 3 tasks with Go:

- **HTTP requests**: scraping content from a website
- **Concurrency**: the pipeline concurrency pattern
- **Image manipulation**: generation and manipulation of images

The lightning talk I would like to give focuses on this last point. I find that, out of the 3 concepts, it is the least spoken of. I want to give a small introduction to the primitives available for image manipulation in the standard library. Taking a small detour to understand the drawing model that Go follows, which is based on the Porter-Duff composition paper. Followed by a review of the most common pure Go libraries for image manipulation. Stating their paradigm, their advantages and drawbacks in the context of GIF generation.

I am a recent university graduate and current Software Engineer at GitHub. While I have given lectures and conducted workshops for university students on Git and CI, I have never given a talk to other peer engineers in a conference. I am hoping to break the ice with this presentation at the go get -u community conference!

References:
- https://blog.golang.org/pipelines
- https://blog.golang.org/image-draw
- https://golang.org/pkg/image/gif/
- https://github.com/llgcode/draw2d
- https://github.com/peterhellberg/gfx
- https://github.com/fogleman/gg
- https://keithp.com/~keithp/porterduff/p253-porter.pdf

I attach the speaker notes to get a rough sense of the talk. Note that you can also see them directly in the slides by pressing **P** in the website.

Hi all, my name is Camilo and I've been tinkering in Go for about 2 years.

I consider myself a casual Gopher. Knowing just enough to get by.

I finished university last year, and am currently working at GitHub.

Today I want to share with you my thoughts on the strengths and weaknesses of multiple gif libraries in Go.

If you want to follow along, you can find the slides in the URL at the bottom.

GitHub has a graph which plots the distribution of your contributions by type.

In general, the pattern I saw was the following:
- user begins by just commiting to their personal projects
- user starts using other people's code, so issues and PRs start to pop up
- finally the user maintains or collaborates on a project, leading to more code reviews

So I wrote a program called gifhub to allow me to see this progression for myself.

GifHub follows the pipeline design for concurrency:
- a stage for concurrently scraping the user profile for each year
- some intermediate stages to generate a graph
- and a sink stage to bundle all frames into a GIF

Unfortunately, there were some issues with my initial choice of dependencies.

ImageMagic is not cross platform. Meaning that OSX and Windows users must rely on the containerized version of the CLI.

But more importantly, ImageMagic has an absurd amount of open vulnerabilities.

I decided to release a new version without ImageMagick.

So I went looking for a simple way to generate GIFs in go.

I wanted a pure Go implementation to ensure cross platform portability.

These are the libraries I played with.

All the code I'll be showing in this presentation can be found in GifHub's playground branch.

I am not related in any way to these projects, I simply want to showcase their strengths and weaknesses so that other developers know what tool to use depending on their problem.

A small warning: some of the GIFs contain flashes. If anyone in the audience suffers with epilepsy I am happy to remove those images.

So let's start with the basics. How does the stdlib deal with GIFs?

At the very core a GIF is just a struct with N number of images and delays.

Go has many ways of representing images.

But the **ONLY** one you can use to construct GIFs is the palleted one.

This is important to know from the get-go since manipulation methods tend to vary greatly between image formats.

A palleted image is a rectangle, a set of for each pixel in that rectangle and the palette which maps the index with a color.

Go's rendering model is based on the classic "Compositing Digital Images" paper by Porter and Duff. I have posted a link to a great digest of the paper by the Go blog.

But I'll walk you through a practical example

The Draw method takes a dst image, dst rectangle, a src image and src starting point.

Finally it takes an operation to apply.

I highly encourage everyone who is thinking of doing image manipulation in Go to go through the standard data types before writing any code.

Playing with these primitives is quite fun.

And familiarizing yourselves with how they are instantiated, manipulated and persisted is quite useful.

Generates complex images through a helper struct.

This library follows the HTML canvas standard.

So if you are familiar with the coordinate system top left corner, as well as the API to move the cursor around the canvas, you are good to go.

Being able to feed stdlib images to the context is a very elegant way of integrating with existing code, without the need to do major refactoring.

Unfortunately, the lack of support for palleted images and animations means we can only really use this library to manually construct complex static images.

then manually transform to a GIF. This is a non-negligible overhead.

A library with higher graphical functions such as domain coloring and noise algorithms.

The main difference in usage with draw2d is that gfx does not take stdlib images.

Instead you create images solely by interacting with the gfx API.

Depending on your use case this might be a win or a lose.

Unfortunately, the author is quite explicit about not guaranteeing any breaking changes.

I highly recommend this library if you want quick, powerful image manipulation for a short-termed project.

Unfortunately, for GifHub, the lack of support for fonts is a deal breaker.

Finally we arrive at gg.

It is a mixture of both draw2d's canvas style image manipulation.

with gfx's approach of completely abstracting away the stdlib image types.

While this library does not support animations either, The fact that it is actively used, maintained and has a stable API

made it a good fit for my project.

The final pipeline is
- scrape yearly contributions concurrently
- generate individual png's of each year's graph with gg
- bundle all the png's via stdlib's GIF encoding

Real time means the wall clock time, or time as we humans perceive it.

User time means the time the CPU spent on the given process.