Writing a Ray Tracer in Go

Tuesday, March 15, 2016

I've been wanting to learn Go for awhile now. I bought a book, read several blogs and tutorials, but I still didn't feel like I was really getting anywhere with the language.

A few weeks ago I went with my a few of my co-workers to the Triangle Golang meetup hosted by my friend Brett, and met some great people. One of the guys demoed an amazing project he had worked on, building a ray tracer in Go.

Seriously, check it out. It blew my mind.

Anyways, this really got me excited again about learning Go and so I decided that I would try writing a *much* less advanced version of a ray tracer myself. I think that picking a semi-small project is the best way for me to actually learn Go, instead of just reading about it.

Over the next several posts, I plan to document my process while learning Go at the same time. This definitely will not be the best Go code ever written. It probably won't be idiomatic, or performant. But, hopefully I'll be able to learn the basics and try out some of the more interesting features of the language.

What is a Ray/Path Tracer

I briefly learned about ray tracers in my computer graphics course in college. I remember grasping the idea, you mimic the way we see objects by shooting 'rays of light' at them and somehow calculate how those rays bounce off, which gives you the color of the object. Or something like that..

I never took the time to actually learn how a ray tracer works, let alone sit down and develop one from scratch. At this point I should probably clarify that what I am actually developing is more accurately called a **path tracer**, not a ray tracer.. but more on that later.

Both ray tracers and path tracers work in the *opposite* way in which we see things in the real world. In reality, light is emitted from light sources (the Sun/lamps/whatever), hits an object and bounces off in many directions. Eventually those rays reach our eyes and we see the objects along with their perceived color (intensity). Modeling reality in software, however, would be extremely inefficient since only the fraction of rays that 'hit' our eye actually matter to us. So, to

Disclaimer: I'm probably glossing over a lot about how light actually works in the real world. But hey, relax.. it's just a blog post.

Ok, so what is the actual difference between a ray tracer and a path tracer? Well from what I understand, what differentiates the two is what happens after a ray intersects with an object in the scene.

Ray tracers fire rays into the scene and when they hit an object, the ray is then split into multiple rays that each go to a different light source in the scene. This means that ray tracers can only compute light that directly hits an object.

Path tracers on the other hand, continuously bounce rays around the scene and integrate all of the values of the intersecting rays and light sources until a certain number of samples (called Sample Per Pixel) is reached.

So in summary, ray tracers take a kind of shortcut and calculate discrete values for each ray as they reach their light source. Path tracers use a more brute-force technique to calculate all possible paths and sum up to a final value per pixel.

Hello, World

Ok, enough theory. Let's talk code.

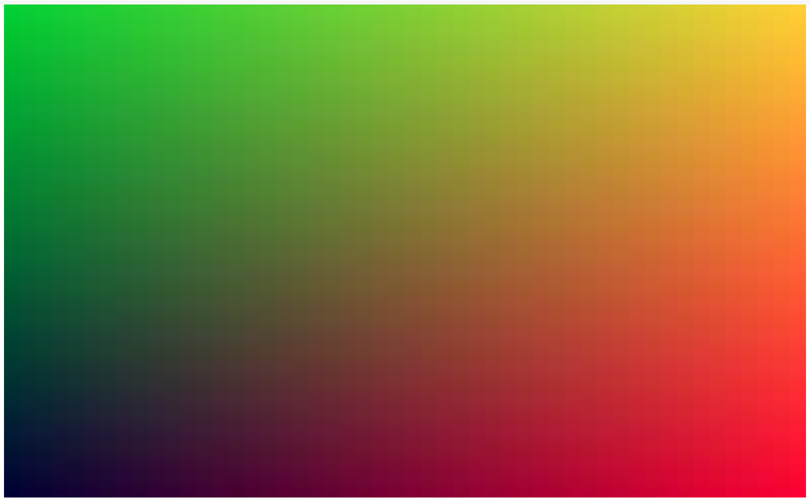
I'm not smart enough to come up with all this on my own, so I bought a book: Ray Tracing in One Weekend on Amazon to guide me through it. Subsequent posts will roughly follow the chapters in the book. I highly recommend getting the book if this kind of stuff interests you (it's only $3).

Anyways, the code in the book is written in C++, which I also haven't done since college.. so it will be an interesting exercise to translate it to Go.

The first chapter in the book has you write some code to output an image. It really doesn't have much to do with path tracing, but it's cool to actually have something show up on the screen after you run your code. The image is a gradient that mixes the values of red, green and blue (RGB) at different intensities from left to right/top to bottom.

The final output looks like this:

mark.phelps/blog



Neat.

The main part of the code that does the work is here:

```go
// writes each pixel with r/g/b values
// from top left to bottom right
for j := ny - 1; j >= 0; j-- {
    for i := 0; i < nx; i++ {
        // red and green values range from
        // 0.0 to 1.0
        v := Vector{X: float64(i) / float64(nx), Y: float64(j) / float64(ny), Z: 0.2}

        // get intensity of colors
        ir := int(color * v.X)
        ig := int(color * v.Y)
        ib := int(color * v.Z)

        _, err = fmt.Fprintf(f, "%d %d %d\n", ir, ig, ib)

        check(err, "Error writing to file: %v\n")
    }
}
```

colors.go hosted with ♡ by GitHub view raw

That's all for this first post. In the next post I'll actually start implementing the path tracer itself.

If you want to follow along in code, check out my repo on Github.

mark.phelps/blog

Like this post? Do me a favor and Tweet