Visual Realizations of Curved Spacetimes

Bill Varcho

May 2, 2017

1 Introduction

In 2014, director Christopher Nolan brought his newest blockbuster, Interstellar, to the big screen. Just like Nolan's previous movies, Interstellar featured an engaging plot, dynamic characters, and a sense of familiarity, even in the strange environments it takes place in. However this sci-fi flick was unique from its predecessors, particularly where it matters most; the details. Interstellar, was the first ever blockbuster to use the principles and physics of Einstein's General Relativity, as both a plot device, and as an influence for the visual imagery. In fact, Nolan and his visual effects team (an English studio named Double Negative), worked together with physicist Kip Thorne to produce the first every visually realistic images of black holes that Hollywood has ever seen.

In this project I attempt to understand the team's work, and describe their solutions in an interesting manner. Additionally, because the physics described seemed 'relatively' straightforward, I also implemented the solution in C++ and CUDA for part of my project as well.

2 Specifying the Geometry

In the movie, two gravitational phenemenon were taken special care in rendering; both a wormhole, and a rotating black hole. The physics for generating light rays for both is quite similar, and (unsurprisingly) high intertwined with the metric. Because of this similarity I chose to combine both of these phenomenon into something new that I could render for this project: a wormhole with an accretion disk.

2.1 Wormhole Metric

2.1.1 Ellis Metric

show grapher picture

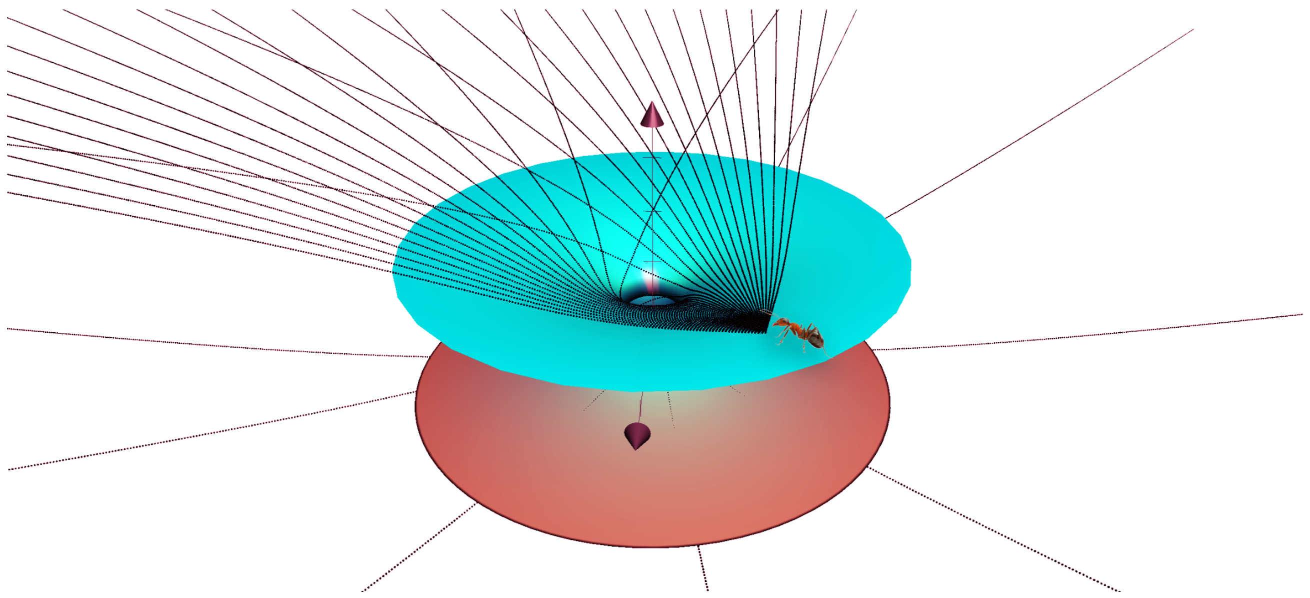

Figure 1: Embedding diagram for the Ellis Wormhole, and light-path geodesics. These light paths are what the 'ant' on this surface sees. In particular notice how near the throat of the wormhole, many of the light paths are highly curved, resulting in a warping of the image the ant sees.

2.2 Accretion Disk

Finally, to create a unique aspect for this project I decided to add an accretion disk to the wormhole metric. To do this I followed a similar procedure to the one the author's used in the papers, which checked whether or not a light ray intersected the disk at any point, and if so rendered it. Furthermore, the warping of this disk in spacetime, creates some interesting and novel visual effects which are fun to realize.

3 Mathematization of Light Rays

Most famously, the paths that light takes in curved spacetime is a geodesic, and thus a solution to the 'geodesic equation', namely

$$\frac{d^2x}{d\zeta^2} + \Gamma^{\alpha}{}_{\mu\nu}\frac{dx^{\mu}}{d\zeta}\frac{dx^{\nu}}{d\zeta} = 0$$

Unfortunately, as the author's note, this equation does not play very nicely with numerical work, and thus another formulation is required. This problem is avoided via Hamilton-Jacobi Theory. By defining a 'super-Hamiltonian' **H**, which is dependent on a particles' position, momentum, and the metric, we can use the theory to set up a system of ODE's describing the motion of the path. First we start by defining

$$\mathbf{H}(x^{\alpha}, p_{\beta}) = \frac{1}{2}g^{\mu\nu}(x^{\alpha})p_{\mu}p_{\nu}$$

Then we can derive the following equations, describing the particles path...

$$\begin{aligned}\frac{dx^{\alpha}}{d\zeta} &= \frac{d\mathbf{H}}{dp_{\alpha}} = g^{\alpha\nu}p_{\nu}\\ \frac{dp_{\alpha}}{d\zeta} &= -\frac{d\mathbf{H}}{dx^{\alpha}} = -\frac{1}{2}\frac{g^{\mu\nu}}{dx^{\alpha}}p_{\mu}p_{\nu}\end{aligned}$$

where ζ is parameterizes the curve taken by the particle.

Now, using the general wormhole metric provided by the interstellar team above, we can conclude that the super-Hamiltonian takes the form

$$\mathbf{H} = \frac{1}{2}(-p_t^2 + p_l^2 + \frac{p_{\theta}^2}{r(l)^2} + \frac{p_{\phi}^2}{r(l)^2\sin^2\theta})$$

Next, the paper provides details on the symmetries of this super-Hamiltonian, and makes the key observation that there are several conserved quantities along the rays path. The first quantity conserved is for the momentum component $b = p_{\phi}$, the second is the time component $p_t = -1$ and the third (which is much less obvious) is known as the impact parameter, and is given then label B^2 (see the paper for more details). Using the constancy of the time momentum component, we can then assert that $\zeta = t$, which yields the following set of ODEs, which completely describe light particle motion in our general wormhole metric.

$$\begin{aligned}\frac{dl}{dt} &= p_l\\ \frac{d\theta}{dt} &= \frac{p_{\theta}}{r^2}\\ \frac{d\phi}{dt} &= \frac{b}{r^2\sin^2\theta}\\ \frac{dp_l}{dt} &= B^2\frac{\frac{dr}{dl}}{r^3}\\ \frac{dp_{\theta}}{dt} &= \frac{b^2\cos\theta}{r^2\sin^3\theta}\end{aligned}$$

4 Computing the Light Paths

Now that we have mathematized the movement of light through time, we now need some sort of method that will enable us to integrate these equations. Unfortunately the complexity of these equations, finding exact analytical solutions can be rather difficult. To alleviate this strain we can use (as the authors suggest) numerical integration.

4.1 Numerical Integration

4.1.1 Euler's Method

As you may know, there are no shortage of numerical techniques for solving systems of ODEs, but because I wanted to implement the code for this project myself, I decided to start with the simplest techniques, and work up from there. The first solution I tried was using Euler's Method. This simple idea tries to predict the next point on a solution trajectory of a differential equation, by modeling it as a linear function of the position of the point and the derivative at the location. Specifically, for some vector field $f(x, y)$, we can approximate the next y-value y_{n+1}, by specifying a step-size Δ and then evaluating

$$\boxed{y_{n+1} = y_n + \Delta f(x_n, y_n)}$$
$$x_{n+1} = x_n + \Delta$$

Although the above is in a nice form, it may not be immediately intuitive to how this method can be applied to our equations above. Therefore, I have explicitly described the translated case for our scenario below.

$$\begin{aligned} l_{n+1} &= l_n + \Delta \frac{dl}{dt} \\ \theta_{n+1} &= \theta_n + \Delta \frac{d\theta}{dt} \\ \phi_{n+1} &= \phi_n + \Delta \frac{d\phi}{dt} \\ p_{l,n+1} &= p_{l,n} + \Delta \frac{dp_l}{dt} \\ p_{\theta,n+1} &= p_{\theta,n} + \Delta \frac{dp_\theta}{dt} \end{aligned}$$

A nice aspect of this approach is its simplicity and ease of implementation for the programmer. This allowed me to quickly prototype my implementation, and also have a 'ground truth' when it came time to implement more complex methods. However because of the linear approach of the method, in order to keep the error under control, small step sizes are needed, otherwise visual artifacts begin to appear (especially near the poles).

4.1.2 Runge-Kutta 4^{th} Order Method

The next method I explored was the Runge-Kutta 4th order method. This technique is an extension of Euler's method, but attempts to eliminate more of the error, by creating a higher-order approximation. To do this, 4 different approximations (the k_is) are computed and then averaged together to get a better approximation for the slope. The canonical description of this method is given by the following computation.

$$\begin{aligned} k_1 &= f(x_n, y_n) \\ k_2 &= f(x_n + \frac{\Delta}{2}, y_n + \frac{\Delta}{2} k_1) \\ k_3 &= f(x_n + \frac{\Delta}{2}, y_n + \frac{\Delta}{2} k_2) \\ k_4 &= f(x_n + \Delta, y_n + \Delta k_3) \end{aligned}$$
$$\boxed{y_{n+1} = y_n + \frac{\Delta}{6}(k_1 + 2k_2 + 2k_3 + k_4)}$$
$$x_{n+1} = x_n + \Delta$$

In a similar style to the above I will include what the math specifically looks like for computing one time-step of the RK4 method on our system of coupled ODEs. One thing, that is very important to note is how each of the k_i's are dependent on the previous ones, and thus all most be computed in the proper order. The sequence of computations I have presented below would be a proper way to correctly implement this method.

$$
\begin{aligned}
k_{l,1} &= \frac{dl}{dt}(p_{l,n}) \\
k_{\theta,1} &= \frac{d\theta}{dt}(l_n, p_{\theta,n}) \\
k_{\phi,1} &= \frac{d\phi}{dt}(l_n, \theta_n, b) \\
k_{p_l,1} &= \frac{dp_l}{dt}(l_n, B^2) \\
k_{p_\theta,1} &= \frac{dp_\theta}{dt}(l_n, \theta_n, b) \\
k_{l,2} &= \frac{dl}{dt}(p_{l,n} + \frac{\Delta}{2}k_{p_l,1}) \\
k_{\theta,2} &= \frac{d\theta}{dt}(l_n + \frac{\Delta}{2}k_{l,1}, p_{\theta,n} + \frac{\Delta}{2}k_{p_\theta,1}) \\
k_{\phi,2} &= \frac{d\phi}{dt}(l_n + \frac{\Delta}{2}k_{l,1}, \theta_n + \frac{\Delta}{2}k_{\theta,1}, b) \\
k_{p_l,2} &= \frac{dp_l}{dt}(l_n + \frac{\Delta}{2}k_{l,1}, B^2) \\
k_{p_\theta,2} &= \frac{dp_\theta}{dt}(l_n + \frac{\Delta}{2}k_{l,1}, \theta_n + \frac{\Delta}{2}k_{\theta,1}, b) \\
k_{l,3} &= \frac{dl}{dt}(p_{l,n} + \frac{\Delta}{2}k_{p_l,2}) \\
k_{\theta,3} &= \frac{d\theta}{dt}(l_n + \frac{\Delta}{2}k_{l,2}, p_{\theta,n} + \frac{\Delta}{2}k_{p_\theta,2}) \\
k_{\phi,3} &= \frac{d\phi}{dt}(l_n + \frac{\Delta}{2}k_{l,2}, \theta_n + \frac{\Delta}{2}k_{\theta,2}, b) \\
k_{p_l,3} &= \frac{dp_l}{dt}(l_n + \frac{\Delta}{2}k_{l,2}, B^2) \\
k_{p_\theta,3} &= \frac{dp_\theta}{dt}(l_n + \frac{\Delta}{2}k_{l,2}, \theta_n + \frac{\Delta}{2}k_{\theta,2}, b) \\
k_{l,4} &= \frac{dl}{dt}(p_{l,n} + \Delta k_{p_l,3}) \\
k_{\theta,4} &= \frac{d\theta}{dt}(l_n + \Delta k_{l,3}, p_{\theta,n} + \Delta k_{p_\theta,3}) \\
k_{\phi,4} &= \frac{d\phi}{dt}(l_n + \Delta k_{l,3}, \theta_n + \Delta k_{\theta,3}, b) \\
k_{p_l,4} &= \frac{dp_l}{dt}(l_n + \Delta k_{l,3}, B^2) \\
k_{p_\theta,4} &= \frac{dp_\theta}{dt}(l_n + \Delta k_{l,3}, \theta_n + \Delta k_{\theta,3}, b)
\end{aligned}
$$

$$
\begin{aligned}
l_{n+1} &= l_n + \frac{\Delta}{6}(k_{l,1} + 2k_{l,2} + 2k_{l,3} + k_{l,4}) \\
\theta_{n+1} &= \theta_n + \frac{\Delta}{6}(k_{\theta,1} + 2k_{\theta,2} + 2k_{\theta,3} + k_{\theta,4}) \\
\phi_{n+1} &= \phi_n + \frac{\Delta}{6}(k_{\phi,1} + 2k_{\phi,2} + 2k_{\phi,3} + k_{\phi,4}) \\
p_{l,n+1} &= p_{l,n} + \frac{\Delta}{6}(k_{p_l,1} + 2k_{p_l,2} + 2k_{p_l,3} + k_{p_l,4}) \\
p_{\theta,n+1} &= p_{\theta,n} + \frac{\Delta}{6}(k_{p_\theta,1} + 2k_{p_\theta,2} + 2k_{p_\theta,3} + k_{p_\theta,4})
\end{aligned}
$$

As can be seen from the above set of computations, this method required many more calculations at each time step, thus slowing down the speed of the simulation. However, since the approximations, were more accurate, slightly larger timesteps could be taken, and certain light paths were as susceptible to the numerical instablity that was often seen using Euler's method.

4.1.3 Runge-Kutta-Fehlberg Method

Finally, I want to talk about the last integration method I used; RKF45. This technique, is similar to RK4, but computes both a 5^{th} and 6^{th} order approximation at each time-step. These two approximations are then compared, and based on their difference the optimal time-step is computed. Although, quite effective, this method is also somewhat numerically verbose, and thus I have decided to remove it from this write up. However the flavor is similar in nature to the previous, and if the reader if interested further, the can inquire my code.

In practice, the RKF45 method is very useful, particularly in how it handles it's adaptive time-step. This allowed the program to make much larger steps for light paths, while still remaining numerically intact for most of the really complex paths. This had a measurable impact on the speed and efficiency of the implementation. Unfortunately, though, in the most difficult areas to render, the time-step would often have to be quite small to avoid innacuracies, which could prevent certain portions of the regions from rendering completely.

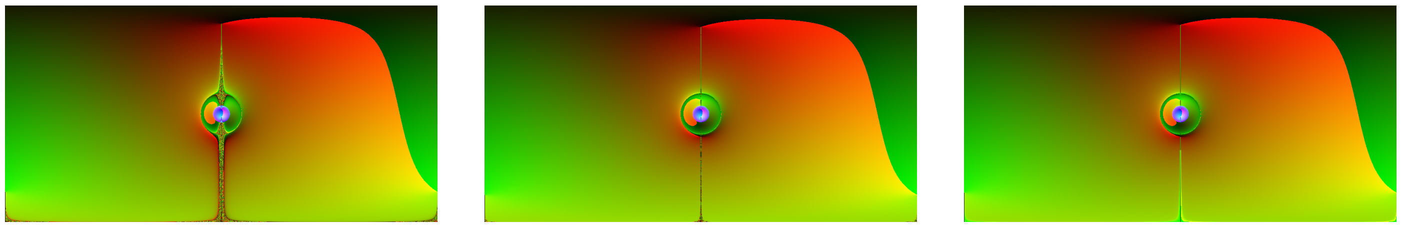
(a) Euler (b) Runge-Kutta (c) Runge-Kutta-Fehlberg

Figure 2: Integration using the 3 proposed methods. In these images the red channel encodes the value of ϕ that the incoming light is coming from, while the green channel holds the value of θ. It is worth pointing out the numerical inaccuracies that occurred when using Euler's method. These errors were reduced using RK4, and then even further via RKF45.

4.2 CUDA

With modern computers, numerically solving a single light path is not a problem at all, and can be done in a mere fraction of a second. However, in the attempt to render a fully sized image, the computational cost becomes much more noticeably. Specifically, in my implementation, I wished to produce an image of size 1024x512 pixels. Therefore, over 500k light paths would have to be computed. On my laptop this task was quite daunting, and took almost 45 minutes to render. In a pedagogical sense, this time factor was quite limiting, because it because unwieldy to debug images, and explore the parameter space of the wormhole. Additionally, since all of the paths that needed to be computed were not interdependent, I wanted to find a better solution.

To overcome this, I decided to modify my code to take advantage of the underlying architecture of my laptop. To do this I decided to write my code as a CUDA kernel. CUDA is a parallel programming interface, designed by NVIDIA, and allowed my code to run on the Graphics card of my computer. This allowed me to compute large groups of paths at the same time massively speeding up my code. By writing the numerical solver as a 'kernel' (or CUDA specific code) I was then able to render the same image in under 8 minutes.

5 Results

[1] Below are a couple of the images rendered with my implementation of the software. The first image was generated before implementing the code with CUDA (thus forcing the resolution to be lower). The second image however, was able to be rendered by CUDA quite efficiently.

[1]If you would like to see more of the images I rendered, just shoot me an email. To keep this project shorter I ommitted most of them.

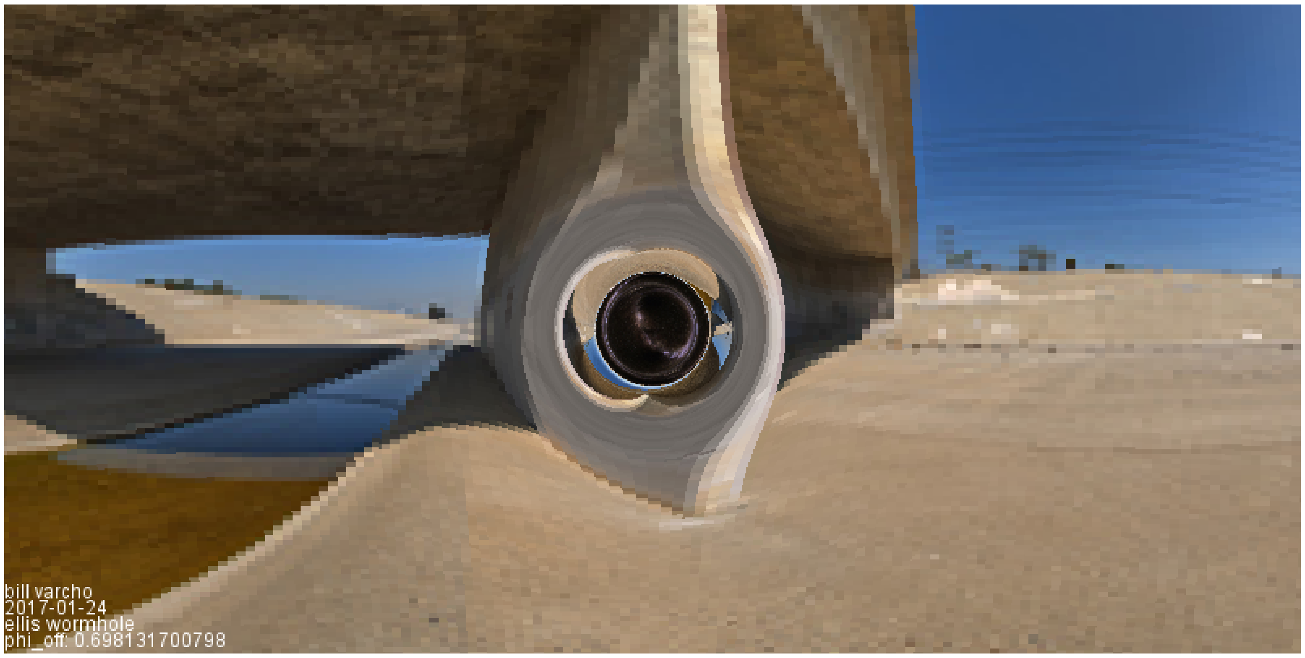

Figure 3: Final rendered image of a wormhole (without an accretion disk) and textures mapped. The warping of the column behind the wormhole is quite noticeable!

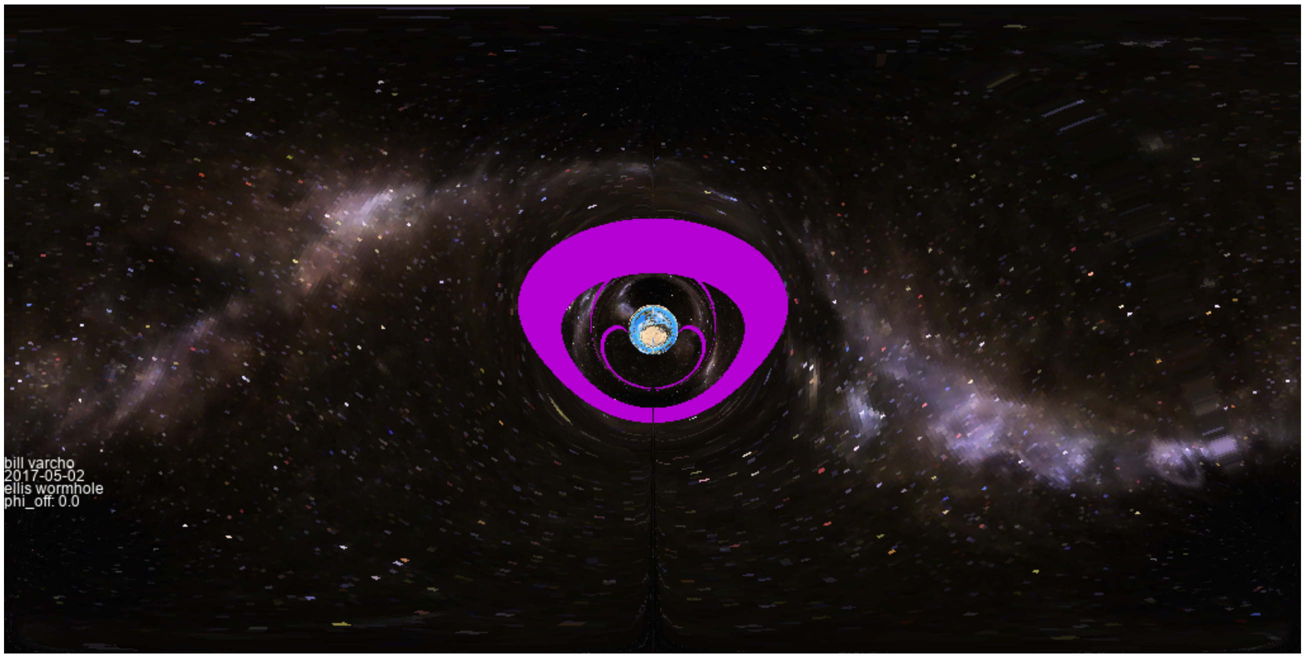

Figure 4: Rendered Image with (a purple) accretion disk, viewed from slightly below. The disk is warped, (and can be seen multiple times), due to the lensing affect.

6 Conclusion

As I movie buff myself, I was quite surprised when I first encountered these papers by the Interstellar team. Oftentimes Hollywood leaves a could shoulder to the real science underlying certain movies, particularly when they involve outer space and science fiction. Because of this, these articles were quite refreshing. Having never scene Einstein's theory used as a mechanism for entertainment, I became really excited by the imagination of the authors. In the movie, the general public was given their first glimpse of what a real black hole would look like, shattering any previous misconceptions.

Furthermore, by taking on the challenge to implement their ideas in code myself, I truly believe that I was able to get a strong grasp of their method, as well as improve my programming maturity. The computational complexity of this task forced me to embrace new tools (such as CUDA), which I believe will be quite valuable for any numerical work I do in the future. Additionally, I have uploaded all of my code on Github (an online website for sharing source code of software) and it is available here.

Finally, I wanted to conclude with by thanking you for taking the time to teach this two-semester course in General Relativity. To be completely honest, the entire reason I took the course was because I was inspired by Interstellar, and wanted to learn more. I could not have anticipated the variety and intrigue of the mathematical methods used in the description of curved spacetime. As a computer scientists, gravitational phenomenom, will probably not make its way into my every day work, but the mathematical methods, and ways of thinking used in the creation of this theory will definitely be useful. Because of this, I am quite thankful.

References

[1] Oliver James, Eugenie von Tunzelmann, Paul Franklin, Kip S. Thorne. *Visualizing Interstellar's Wormhole.* `https://arxiv.org/pdf/1502.03809.pdf`

[2] Oliver James, Eugenie von Tunzelmann, Paul Franklin, Kip S. Thorne. *Gravitational Lensing by Spinning Black Holes in Astrophysics, and in the Movie Interstellar.* `https://arxiv.org/pdf/1502.03808.pdf`

[3] Press W, Teukolsky S, Vetterling W and Flannery B. *Numerical Recipes in C (2nd ed.)* Art of Scientific Computing (Cambridge: Cambridge University Press), 1993