Gravispy: gravitational simulations in open source software*

Spence Norwood, Kellen O'Keefe, and Calvin Ross

University of Texas at Dallas, Physics Department

Abstract

Gravispy is designed to visualize the effects of gravitational lensing. The goals of the project is to calculate a massive object's effect on spacetime using the Schwarzschild metric and modeling the resulting warped perspective in three dimensions. By using the Gauss-Kronrod quadrature formula for integration and the Brent-Dekker method for finding the roots of equations, the null-geodesics representing the trajectories of light are calculated numerically at radii near the event horizon of a black hole. These values are then used to create a mapping between the light of the unperturbed state supplied by the user and the final lensed state. With this map, a resultant image is produced and is projected onto a sphere using OpenGL to provide a panoramic view of the environment. The results of this project have been generally successful. Gravitational lensing has been correctly modeled—with the exception of a small shearing of the output image—and the lensing effect is viewable in a three dimensional model as intended. Initial goals were to make the simulation dynamic, but currently only a static simulation is supported. Future improvements to this project could be made by implementing this functionality, and by increasing the physical accuracy of the simulation by modeling the effect of blueshift and the existence of hidden images.

* https://github.com/cjayross/gravispy

I. INTRODUCTION

Gravitational lensing is the process by which the curvature of spacetime induced by the presence of a very massive and dense object distorts the straight line paths traveled by light. This results in a warped perspective of background light sources that lie behind said object with respect to the observer.

The goal of our project is to model this process visually. By which we mean to incorporate the lensing equations that have been solved by cosmologists such as [1, Perlick] and [2, Frittelli, et al.] into OpenGL so that we can create interactive, graphical visualizations of the distortions in (ideally) real-time.

Although we managed to accomplish the numerical methods and calculations for the lensing equations, we have not yet been able to finish it's incorporation into OpenGL. However, we have managed to create Gravispy's first simulation by applying our calculated distortions onto an image supplied by the user as discussed in Section IV.

II. MOTIVATION

As documented by [1, Perlick], the process of analyzing the exact lensing of spherical spacetimes amounts to representing light that travels backwards in time from the point of their observation. What this means is that observed light is modeled to originate from the observer themself and propagates out to the source from which it was created. This is exactly the assumptions made in the raytracing algorithms applied in computer graphics. As such, the prospect of modeling the phenomena of gravitational lensing is an enticing one, however, most existing implementations of these lensing models are much more tailored toward scientific representations rather than a more intuitive, layman representation of such a beautiful phenomena.

In terms of the course, the necessary problems that are presented with this task happen to also mirror the very subjects covered in class, including root finding and numerical integration. And yet, this happens to be true in such a way that is not too overbearing for a semester project but certainly challenging enough to require us to expand our understanding of the subjects involved in order to accomplish our goal.

III. METHODS

III.1. Background Theory

To approach the problem, the first step was to begin with establishing the parameters that will be used to characterize both the parameters of the input and the format to describe the output. This was done by establishing two coordinate systems: the *observer's sky* and the *source sky*, where each are charted via spherical coordinates.

Physically, the observer's sky is defined as the past lightcone at the location of the observer, meaning that the light that is observed at this point describes the set of null-geodesics that have been emitted from the source sky that cross with the observer. Therefore, describing the lensing due to the curvature of spacetime amounts to solving the null-geodesic equation using the observer as the initial condition:

$$\frac{d^2x^\mu}{d\lambda^2} + \Gamma^\mu_{\rho\sigma}\frac{dx^\rho}{d\lambda}\frac{dx^\sigma}{d\lambda} = 0, \tag{1}$$

$$g_{\mu\nu}\frac{dx^\mu}{d\lambda}\frac{dx^\nu}{d\lambda} = 0. \tag{2}$$

Where x^μ characterizes the path of the light through spacetime, λ is an arbitrary choice of an affine parameter, $g_{\mu\nu}$ is the spacetime metric, and $\Gamma^\mu_{\rho\sigma}$ are the Christoffel symbols.

Now, to characterize the problem, our goal is to map angles, (θ, ϕ), on the observer's sky to the output angles (ψ, γ) on the source sky. A mathematical simplification for spherically symmetric lenses is to define a new angle, Θ, that is defined as the angle between the point (θ, ϕ) and the optical axis that is drawn from the center of the lens to the observer.[1] This reduces the problem to finding how Θ maps to Φ, the angle between the source (ψ, γ) and the same optical axis used to define Θ. These parameters are illustrated in Fig.(1).

Assumptions can be made to simplify the problem. In our current implementation we have three variants that are used to calculate the lensing of a Schwarzschild spacetime. The simplest case is the thin lens (or Newtonian) approximation of the geodesic by modeling the path as two lines that join at the impact parameter, the point that the light is closest to the lens' center. A diagram of this process can be viewed in Fig.(2). The relationship between the two trajectories is given by the deflection angle, α, which is inversely proportional to the impact parameter r_P:

$$\alpha = \frac{4M}{r_P}. \tag{3}$$

Beyond the thin lens approximation, we implement two exact lensing equations, one that applies to all spherically symmetric and static metrics (as is the Schwarzschild metric) while the second is an algebraically simplified variant of the general lens equation that is tailored for the Schwarzschild metric specifically.

Suppose that you have a spherically symmetric and static spacetime metric in the form:

$$g = A(r)^2\big(S(r)^2\,dr^2 + R(r)^2(d\theta^2 + \sin^2\theta\,d\phi^2) - dt^2\big), \tag{4}$$

then as a function of $R(r)$, $S(r)$, and Θ, the exact lens equation is[1]

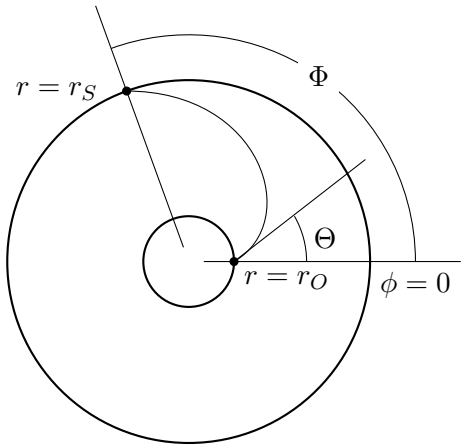
$$\Phi = \frac{|\cos\Theta|}{\cos\Theta}\int_{r_O}^{r_S} \frac{R(r_O)S(r)\sin\Theta\,dr}{R(r)\sqrt{R(r)^2 - R(r_O)^2\sin^2\Theta}}. \tag{5}$$

This integral however needs to be evaluated piecewise in the case that the time derivate, $\dot{r}$, changes sign.

III.2. Numerical Approach

Computationally, the primary limitation of the problem is governed by the accuracy and time complexity of evaluating Eq.(5) numerically. For this, we used `scipy.integrate.quad`

Figure 1. Illustration of the process we intend to describe (inspired by a figure used in [1]). The center of the two concentric circles defines the center of the lens. r_O and r_S are the radial distance from the lens' center two the observer and the collection of sources respectively. The curve between the two radii represents the path that was traveled by a light ray that was emitted at the point at the angular position Φ relative to the optical axis and is absorbed by the observer at the angle Θ.



which uses a 21-point Gauss-Kronrod quadrature formula implemented from the Fortran library QUADPACK to perform the integration.

Gauss-Kronrod quadrature is an extension of Gaussian quadrature which evaluates the integration of a function $f(x)$ that is well approximated by polynomials up to an order of n by

$$\int_a^b f(x)\,dx \approx \frac{b-a}{2}\sum_{i=1}^{n}\omega_i f(\frac{b-a}{2}x_i + \frac{a+b}{2}). \tag{6}$$

The set x_i here are referred to as *nodes* and ω_i as the *weights* and are both determined by an orthogonal polynomial of degree n. Particularly, x_i corresponds the to i-th root of said polynomial. Unfortunately however, going into more detail than this would extend far beyond the scope of this report.

For the Schwarzschild metric, we are able to reduce Eq.(5) by making the coordinate transform[2]

$$l = \frac{1}{r\sqrt{2}} \tag{7}$$

and defining the impact parameter, l_P, as the smallest, positive root of

$$l_O^2(1-2\sqrt{2}Ml_O) - l_P^2(1-2\sqrt{2}Ml_P)\sin^2\Theta = 0. \tag{8}$$

With these definitions, Eq.(5) reduces to

$$|\Phi| = \int_{l_S}^{l_O} \frac{dl}{\sqrt{l_P^2(1-2\sqrt{2}Ml_P) - l^2(1-2\sqrt{2}Ml)}} \tag{9}$$

$$+\,2\int_{l_O}^{l_P} \frac{dl}{\sqrt{l_P^2(1-2\sqrt{2}Ml_P) - l^2(1-2\sqrt{2}Ml)}}. \tag{10}$$

Where the sign of Φ is determined by the sign of $\cos\Theta$.

Note that in Eq.(10), the factor of 2 in front of the second integral corresponds to the fact that the light travels first down toward the lens' center before launching back out and returning to the sphere at r_O. Thus, the integration is taken twice, once as r_O descends to the impact parameter r_P, and then again as it reascends.

Eq.(8) will always have exactly two positive roots in the case when

$$\sin\Theta > 3\sqrt{2}Ml_O\sqrt{3(1-2\sqrt{2}Ml_O)}. \tag{11}$$

Thus, to find the impact parameter our program uses `scipy.optimize.brentq` to find the roots of Eq.(8).

The function `brentq` utilizes the Brent-Dekker method of finding the roots of a function $f(x)$. It works by combining the bisection method discussed in class and the secant method, which works like a finite-difference approximation of Newton's method. The algorithm works by choosing two values a_0 and b_0 such that $f(a_0)$ and $f(b_0)$ have different signs. From there, two values, s and m, are calculated by

$$s = \begin{cases} b_k - \frac{b_k - b_{k-1}}{f(b_k) - f(b_{k-1})} f(b_k), & \text{if } f(b_k) \neq f(b_{k-1}) \\ m, & \text{otherwise} \end{cases} \tag{12}$$

$$m = \frac{a_k + b_k}{2}. \tag{13}$$

Afterward, the next iteration, b_{k+1}, is then set to s, if s is in between m and b_k, and it is set to m otherwise. However, in order to guarantee convergence, there are additional checks that decide whether or not the secant method should be used for the next iteration. Assuming s satisfies the previously stated condition and ϵ is the desired tolerance, the secant method is allowed to be used in the next iteration if:

Bisection Method Used Last	Secant Method Used Last
$\lvert s - b_k \rvert < \frac{1}{2} \lvert b_k - b_{k-1} \rvert$	$\lvert s - b_k \rvert < \frac{1}{2} \lvert b_{k-1} - b_{k-2} \rvert$
and $\lvert \epsilon \rvert < \lvert b_k - b_{k-1} \rvert$	**and** $\lvert \epsilon \rvert < \lvert b_{k-1} - b_{k-2} \rvert$

III.3. Implementation

With a formula for Φ, implementing the lensing function is done by using the function to build a mapping between the source sphere and the observer's sphere. We elected to build a dictionary object of the lens since this allows us to save it for repeated use, which would certainly be needed if we intend to incorporate dynamic scenes in the future. For example, if we wanted to create the illusion of the observer orbiting the lens, instead of recalculating the lens each frame, we could roll the image being used as the backdrop along the x-axis and then reapply the same lens map calculated earlier.

To build the lens map itself, we had to use spherical geometry to represent a flat, 2D image as the surface of a sphere. To start, we needed to define a coordinate mapping between pixels of the inputed 2D image and the hypothetical sphere. For this we used our methods `pix2sph` and `sph2pix`:

```
import numpy as np

def wrap(angles):
    # wrap angles to the interval [0,2*pi]
    angles = np.asarray(angles)
    return np.mod(angles+2*np.pi, 2*np.pi)

def unwrap(angles):
    # wrap angles to the interval [-pi,pi]
    angles = np.asarray(angles)
    return np.mod(angles+np.pi, 2*np.pi)-np.pi

def sph2pix(theta, phi, res):
    x = np.rint((res[0]-1)*wrap(phi)/2/np.pi)
    y = np.rint((res[1]-1)*(1-np.sin(theta))/2)
    return np.array([x, y]).astype(int)

def pix2sph(x, y, res):
    # phi \in [-pi, pi]
    # theta \in [-pi/2, pi/2]
    phi = unwrap(2*np.pi*x/(res[0]-1))
    theta = np.arccos(2*y/(res[1]-1)-1)-np.pi/2
    return np.array([theta, phi])
```

With the above code, the lensing map can be constructed by defining Θ by the spherical analog to the Pythagorean theorem:

$$\cos\Theta = \cos\theta\cos\phi. \tag{14}$$

Once the angle Φ is deduced from Θ, the angles (ψ, γ) can then be retrieved via the law of sines:

$$\frac{\sin\psi}{\sin\Phi} = \frac{\sin\theta}{\sin\Theta}, \quad \frac{\sin\gamma}{\sin\Phi} = \frac{\sin\phi}{\sin\Theta}. \tag{15}$$

In total, the full `generate_lens_map` method is given as follows (note that Θ and Φ are named `alpha` and `beta` respectively).

```
import numpy as np
import itertools as it
from PIL import Image
from ..geom import pix2sph, sph2pix, wrap, unwrap

def generate_lens_map(lens, res, args=(), prec=3):
    # it.product produces the cartesian product of
```

```
    # the ranges [0, width) and [0, height)
    coords = list(it.product(*map(np.arange, res)))
    x, y = np.asarray(coords).astype(int).T
    theta, phi = pix2sph(x, y, res)

    arccos2 = lambda a: np.sign(unwrap(a))*np.arccos(a)
    arcsin2 = lambda a, k: np.pi*k + (-1)**k*np.arcsin(a)

    # consider alphas to be equal if they are the same up
    # to 3 decimals (by default). this reduces the amount of
    # calls to lens from possibly millions to only about 3000
    alpha = np.round(arccos2(np.cos(theta)*np.cos(phi)), prec)
    # compress alpha
    alphaz = np.unique(alpha)
    betaz = np.fromiter(lens(alphaz, *args), np.float64)
    # expand betaz
    beta_map = dict(zip(alphaz, betaz))
    beta = np.fromiter(map(beta_map.__getitem__, alpha), np.float64)

    # we will intentionally fail invalid calculations,
    # they will be filtered out afterward.
    # as such, we don't need to be warned that they occurred.
    errstate = np.seterr(all='ignore')

    sigma = np.sin(beta)/np.sin(alpha)
    mu, nu = map(lambda a: sigma*np.sin(a), [theta, phi])
    # this choice of k's needs to be scrutinized
    k1, k2 = map(lambda a: np.abs(unwrap(a))>np.pi/2, [theta, phi])
    psi, gamma = map(arcsin2, [mu, nu], [k1,k2])

    # cut out invalid results
    idxs = np.logical_not(np.isnan(psi) | np.isnan(gamma))
    keys = zip(x[idxs], y[idxs])
    values = zip(*sph2pix(psi[idxs], gamma[idxs], res))
    np.seterr(**errstate)

    return dict(zip(keys, values))
```

With the generated lens map, the process of applying the map to an image is described in Fig.(3). However, the primary issue with `generate_lens_map` is the tearing in the image that occurs at the lines $\phi \in \{-\pi/2, \pi/2\}$. We unfortunately haven't yet found out the cause of this.

Figure 2. The thin lens approach is the simplest approximation of the lensing in a Schwarzschild spacetime and is depicted below. α is the deflection due to the lensing and is very analogous to the refraction due to a typical glass lens.

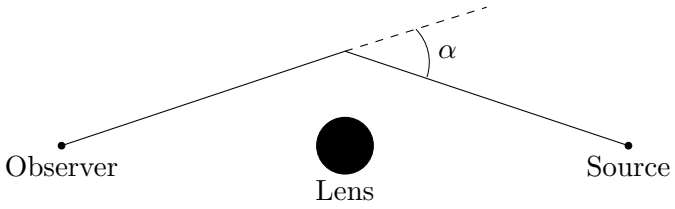
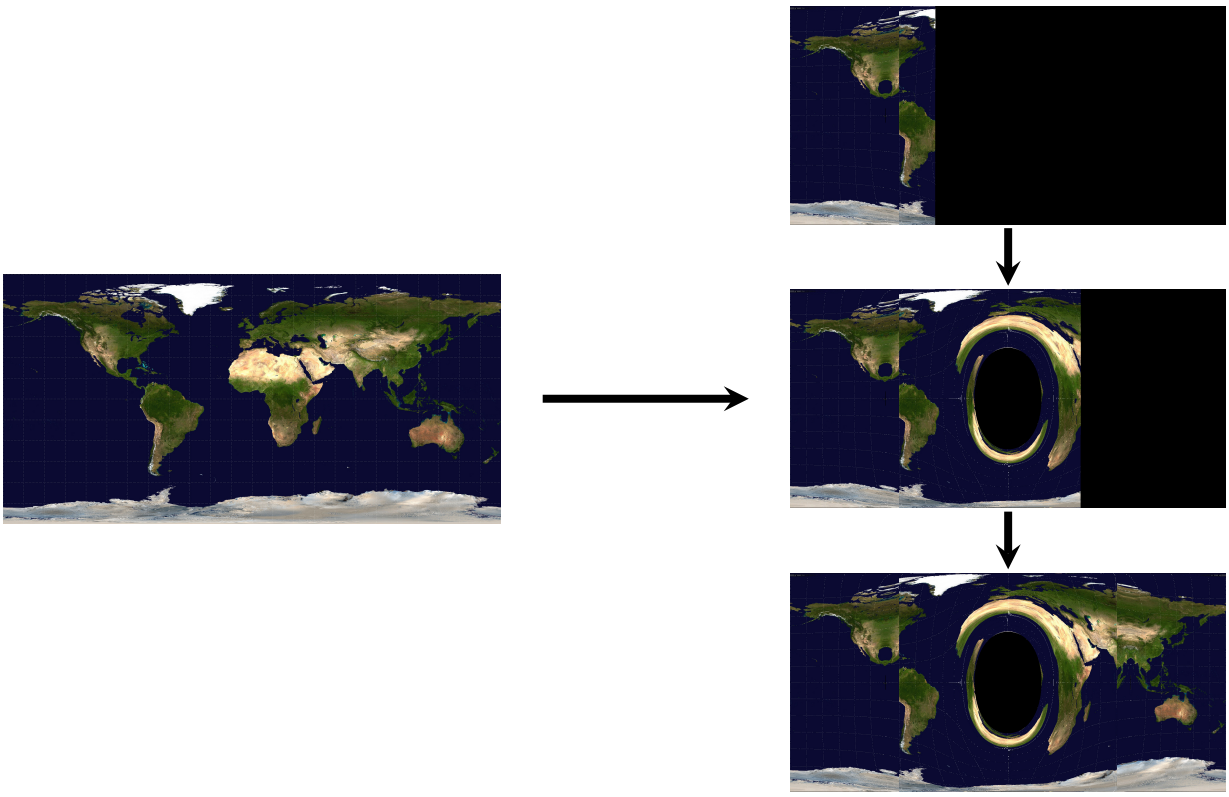


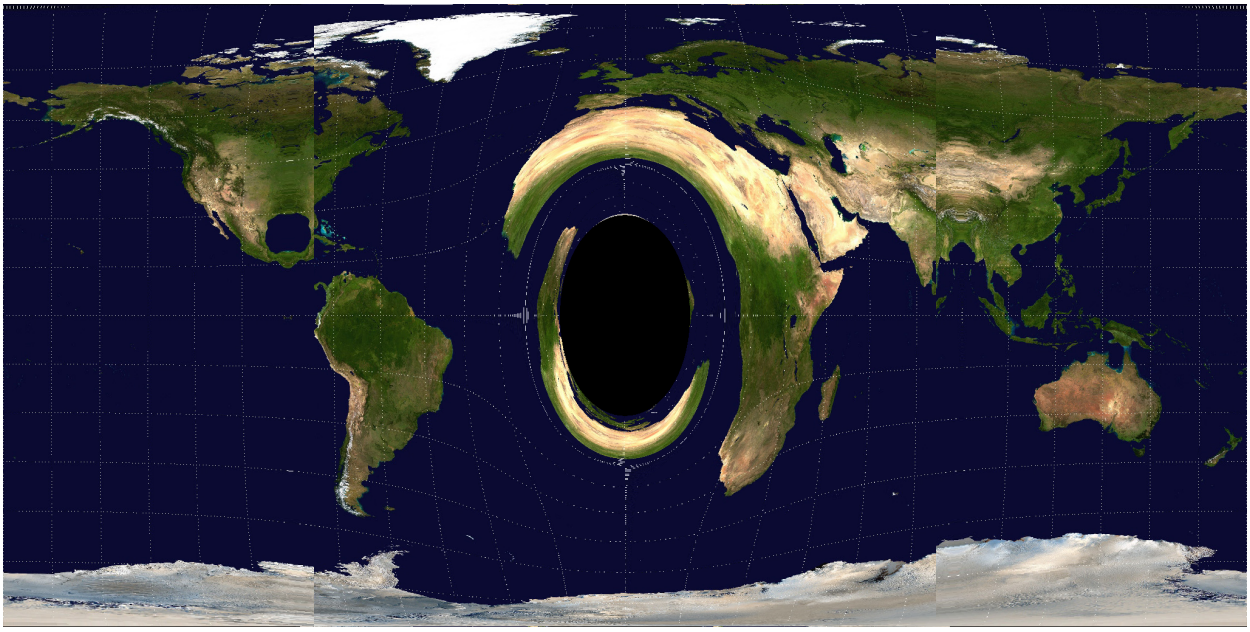
Figure 3. Illustration marking intermediary steps during the image forming process. First, a new image is generated as a default, black canvas of the same resolution as the input image. Then during execution, the algorithm used by our `apply_lensing` method iterates through each pixel of the newly generated image and uses the passed lensing map to determine which pixel from the input should be placed at that location.



IV. RESULTS

Our final product thus far has been able to not only achieve the majority of the desired lensing calculations, but we were able to also implement these effects onto real images.

Figure 4. Final result of applying the explicit Schwarzschild lensing map on a sample image using a Schwarzschild space-time defined by $M = 1.0$ and $r_O = 10M$. In this example, one can easily see the problematic image tearing at the $\pi/2$ and $-\pi/2$ meridians.



Examples of these visualisations can be seen in Fig.(4) and (5). Both of these examples demonstrate the current robustness of gravitational lensing even for observers very near the event horizon of a black hole.

In Fig.(6), we demonstrate the varying effectiveness of the different lensing techniques that are available in our program. Another aspect of Fig.(6) to note is the difference in the shapes of the graphs formed by both variants of the exact lensing equation. We expected the only difference between the two methods would be the computation time, but the fact that they result in different graphs suggests that the `scipy.integrate.quad` method evaluates differently between the two approaches and discovering the cause of this may be a difficult venture.

V. DISCUSSION AND CONCLUSION

As brought up in Subsection III.3, there is an issue with our current methods that results in visible tearing at the $\pm\pi/2$ meridians of the image. However, since this image is intended to be used to texture a sphere in the final OpenGL implementation and that the observer in the simulation—who will be sitting at the sphere's center—will not have a field of view

Figure 5. Second example of a generated Schwarzschild lensing map using the same space-time as in Fig.(4) but at a radius of $r_O = 2.5M$, which is significantly closer to the event horizon. The accuracy of this result hasn't yet been scrutinized, however the overall converging of the observer's sky to the point at $\phi = 0$ is to be expected.

larger than π, this problem can perhaps be hidden from the end user. Nonetheless, this bug may be due to mistakes we have yet to see in our underlying geometry that may make our lensing less robust or may introduce more bugs latter on.

Aside from this, further updates intend to also implement gravitational blueshift of light that occurs when the observer is significantly closer to the singularity than the sources. Another, much more significant problem that our code overlooks is that of the existence of hidden images. Hidden images represent the degeneracies of the lensing map, meaning that a single point on the observer's sky maps to more than one location on the source sky. The result of this is going to be an amplification of the luminosity at specific locations on the observer's sky and is the phenomena that most accentuates the presence of Einstein rings.

---

[1] V. Perlick, Phys. Rev. D **69** (2004), https://arxiv.org/pdf/gr-qc/0307072.pdf.

[2] S. Frittelli, T. P. Kling, and E. T. Newman, Phys. Rev. D **61** (2000), https://arxiv.org/pdf/gr-qc/0001037.pdf.

Figure 6. Comparison between the three levels of generality in the Schwarzschild lensing equation, with the resultant angles shifted vertically by a factor of π for clarity. The first graph accentuates the differences between the methods near the singularity, while the latter graph shows how the errors of the first decrease as the radius of the observer increases.