Parallel Ray Tracing of Black Hole Images Using the Schwarzschild Metric

LIAM NADDELL and MARCELO PONCE, Department of Computer and Mathematical Sciences, University of Toronto Scarborough, Canada

Rendering images of black holes by utilizing ray tracing techniques is a common methodology employed in many aspects of scientific and astrophysical visualizations. Similarly, general ray tracing techniques are widely used in areas related to computer graphics. In this work we describe the implementation of a parallel open-source program that can ray trace images in the presence of a black hole geometry. We do this by combining a couple of different techniques usually present in parallel scientific computing, such as, mathematical approximations, utilization of scientific libraries, shared-memory and distributed-memory parallelism.

CCS Concepts: • **Applied computing** → **Physics**; **Astronomy**; • **Computing methodologies** → **Shared memory algorithms**; **Distributed programming languages**.

Additional Key Words and Phrases: Black holes, ray tracing, parallel programming, mathematical approximations, scientific libraries.

ACM Reference Format:
Liam Naddell and Marcelo Ponce. 2025. Parallel Ray Tracing of Black Hole Images Using the Schwarzschild Metric. In *Proceedings of PEARC'25: The Power of Collaboration (PEARC '25).* ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Ray tracing is a foundational technique in computer graphics, which can be used to render photo-realistic scenes. A good introduction to the main techniques and practical implementations can be found in Ref.[17]. In general this implies expensive processes and requires the corresponding appropriate compute time as well as complex and sophisticated algorithms. The applications [13] of which range from generation of high-quality realistic visualizations to rendering of movies or animations, and up to even more exotic images of black holes, such as the ones remarkably done for the black hole residing at the center of the galaxy M87 [2] and the supermassive black hole Sagittarius A* at the center of our own galaxy [6]. In these cases, ray-tracing techniques have been applied to General Relativistic Magneto-HydroDynamics simulations in order to produce the so-called fiducial templated models of the event horizon images. Techniques such as these have allowed us to produce images and media that have captivated the minds of millions.

We should emphasize that there exist a large variety of ray tracer implementations already [3, 8, 19] and even ones with direct astrophysical applications such as the rendering of black hole surroundings [5, 9, 16]. However, our approach seeks to highlight some specific features: i) its open-source implementation; ii) the minimalistic and simplistic implementation strategy,

Authors' Contact Information: Liam Naddell, liam.naddell@mail.utoronto.ca; Marcelo Ponce, m.ponce@utoronto.ca, Department of Computer and Mathematical Sciences, University of Toronto Scarborough, Toronto, ON, Canada.

PEARC '25, Columbus, Ohio

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

i.e. by tackling mostly the actual mathematical problem using specialized libraries to solve the governing differential equation of the problem; iii) the ease of implementation in a fairly efficient and high-performing way by employing standards in shared-memory and distributed-memory paradigms; and iv) the hardware agnostic approach, i.e. not depending on specialized hardware such as GPUs.[14]

In order to model a simple black hole scenario, we can use the Schwarzschild metric [1]. This models black holes which, among other properties, are non-spinning and not charged, meaning that their deformation of the trajectories of light rays do not depend on the angle of approach. It has been well established [11] that the ordinary differential equation (ODE),

$$u'' - u = 3Mu^2 \tag{1}$$

relates the trajectory of a light ray to the mass M of a black hole, from the Schwarzschild metric; where u represents $1/r$, r being the Euclidean distance of a point on the light ray to the black hole center. u is differentiated with respect to ϕ, the azimuthal angle between the point and the black hole origin. See Fig. 1 and its accompanying section for more detail on the geometry of this equation. After providing initial conditions, it is possible to solve Eq.(1) and trace the entire trajectory of the ray with high fidelity.

After being able to compute the trajectory of individual rays, we can use these rays to generate a full image using ray tracing. Ray tracers render a simulated scene by casting light rays from a simulated camera, and computing what objects in the scene would be visible to that ray. Hypothetically, if a particular camera would produce an image of 1920×1080 pixels, we could build the image that camera would record by casting a light ray through each pixel. Because ray tracing's unit of perception is how simulated light rays interact with a simulated scene, it is a natural choice for simulations which involve the bending or distortion of light.

2 Implementation

2.1 Approximating the trajectory of a light ray as a piecewise function

While the fundamental idea of taking a ray tracer and modifying it so that light rays are distorted is sound, the implementation of such an idea is fraught with difficulty. The first major challenge is that ray tracers need linear equations to be able to compute object intersection in a scene. This means we need to find a way to describe our light ray as a piecewise-defined set of linear equations. We found the best way to do this was to compute discrete points on the light ray, and compute intersection using the line segments defined by said points.

2.2 Using GSL for ODE approximation

As for how to compute these segments, the GNU Scientific Library's (GSL) suite [7] of ODE approximation tools found significant utility. GSL allows us to define our equations as C functions, and our particular ray by a set of initial conditions. From there, we can use GSL to compute a series of discrete points on the ray governed by a timestep of our choosing. For initial conditions, we need to describe, in polar coordinates, an initial point on the ray and a single successor point, which was computed using the linear direction, as we are casting rays far from the black hole's influence. As for the *timestep* (as it is commonly known in the GSL documentation), we allowed program users to control the distance between computed points, which we will refer to as ε, i.e. the *discretization* or *step* of the simulation. Lower values of ε result in more accurate trajectories, but also require more computed points. Fig 1 depicts an example of how these paths are computed.

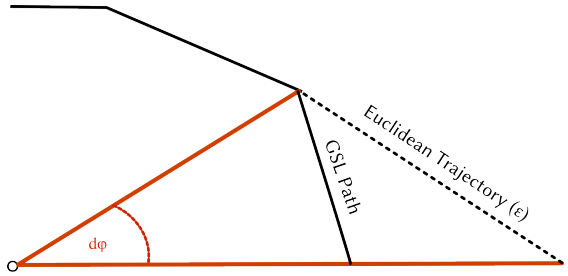

Fig. 1. We use Euclidean geometry to compute $d\phi$, then rely on GSL to stencil the ray's path until the simulation progresses by $d\phi$. The point at the center is the origin of a black hole.

2.3 Domain Decomposition

The essential element of speeding up a problem with Message Passing Interface (MPI) [10] is often to attempt to decompose the problem space into subproblems which can be computed on an individual node. In our case, our problem space is the image we are attempting to render. The most simple solution is to decompose along the scanlines, and render distinct scanlines on distinct nodes. In effect, one would attempt to assign a "band" to each process to render against.

2.4 Use of OpenMP

In the context of a single band, it is quite easy to use OpenMP [4] to accelerate rendering, as the problem can be decomposed along the individual scanlines. This was quite helpful, as MPI was best used to distribute the problem across nodes, and OpenMP was useful for distributing the problem across cores. Additionally, as shown in Sec. 3, OpenMP threads can scale better than MPI processes, meaning we would like to take advantage of the increased performance wherever possible.

2.5 Code Availability

Considering all the elements described in the previous sections, we developed our source code in C++ which is available under a GPL-v2 license, in the following GitHub repository: https://github.com/liamnaddell/BHRaytracer.

3 Results

3.1 A selection of renders

This section includes results of images rendered employing our ray tracer's implementation. The background images for these renders were taken from NASA's Scientific Visualization Studio and the ESA/Nasa Hubble Mission. We present two images – Figs. 2 and 3, it is easy to see the effects of gravitational lensing, as well as the clear Einstein ring.

3.2 Scaling analysis

For this problem, it is possible to consider different *metrics* to analyze the scaling behavior of a given implementation. For instance, one could consider increasing the size of the background image to stress image parsing; or instead increase the number of rendered pixels; or decrease ε yielding to more calls into GSL; or increase samples-per-pixel which controls antialiasing.

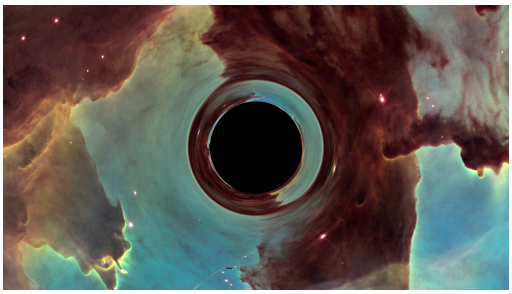

Fig. 2. Image generated from a portion of the Eagle Nebula M16 downloaded from [12].

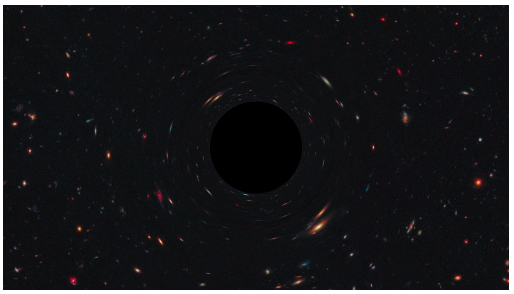

Fig. 3. A second sample image from our ray tracer, in this case from the CANDELS project[18].

When increasing the size of the background image, we estimate that this increases the time spent outside the parallel region, as the background image has to be parsed by each MPI process. Because of this, we consider that this does not present significant interest for scaling analysis. In contrast, the number of rendered pixels, and the number of cores used present useful scaling parameters, since they correspond most directly to the amount of work needing to be done, and the number of workers to do it.

Fig. 4 show a weak scaling analysis where we linearly increase the number of pixels rendered and the single-node cores utilized in tandem. We observe that both OpenMP and MPI start at around 145 seconds, and increase logarithmically before plateauing roughly around 170-185 seconds per run. While ideally, we would observe a consistent time which does not increase, these results show relatively stable performance. We also witness OpenMP slightly outperforming MPI, which is expected, since MPI requires transferring pixel data between processes.

We observe something similar with the strong scaling results in Figs. 6 and 7. These results were run by increasing the number of cores used with a fixed workload. These results show an almost perfect convergence down to the serial fraction with similarly competitive results between OpenMP and MPI.

All the results and analysis presented in this section were obtained running our code on the Niagara supercomputer [15], which is a large homogeneous cluster composed by 2,024 Lenovo SD530 servers each with 40 Intel "Skylake"/"CascadeLake" cores working at approx. 2.4/2.5 GHz. Each result is an average of 5 runs with identical parameters.

4 Discussion

4.1 Educational Value

This ray tracer implementation was originally developed as a final project for an upper year specialized topics course in Computer Science, where the main elements in the course were the discussion of High-Performance Computing techniques. As a rich computational project, it combines elements from multiple disciplines, specifically physics simulations (both through the use of a ray tracer, and through the simulation of gravitational distortion), solutions to differential equations via approximations with GSL, path stenciling, image parsing, parallel and distributed computing with MPI and OpenMP, and domain decomposition. These factors make the project compelling as an educational tool.

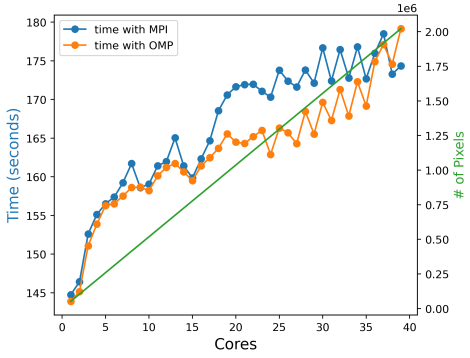

Fig. 4. Weak scaling analysis where MPI process count and image width are increased in tandem

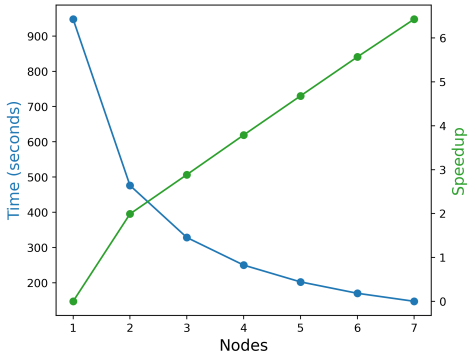

Fig. 5. Multi-node strong scaling analysis with 4 cores per node

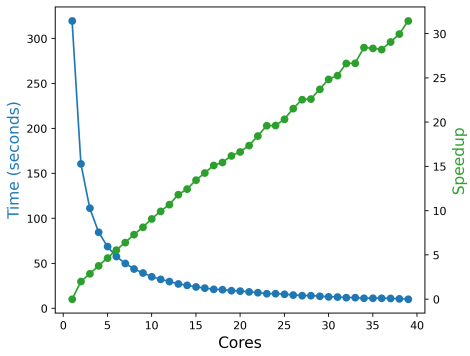

Fig. 6. Strong scaling analysis as MPI process count is increased

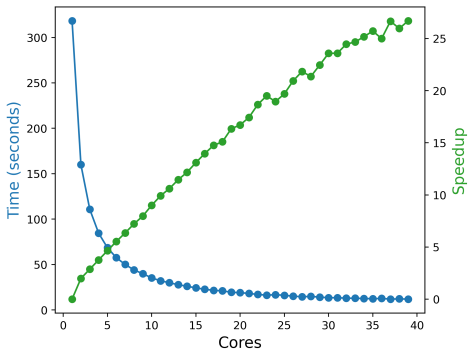

Fig. 7. Strong scaling analysis with OpenMP threads – OpenMP is slightly faster

5 Conclusions

Gravitational lensing is a beautiful phenomenon, both when witnessed from space, and when digitally rendered. This paper explored a relatively simple method for creating a ray traced image which shows gravitational lensing, and how to accelerate the process using shared and distributed memory computing. One may argue that the ray-tracing problem has been solved, however, taking a start-from-scratch approach, can be useful to highlight important and fundamental aspects of the basic techniques and implementations used. Furthermore, an approach such as this offers multiple profound opportunities in the areas of teaching and education, both with regards to general scientific and High-Performance computing techniques.

Acknowledgment

Computations were performed on the Niagara supercomputer and the Teach cluster at the SciNet HPC Consortium. SciNet is funded by Innovation, Science and Economic Development Canada; the Digital Research Alliance of Canada; the Ontario Research Fund: Research Excellence; and the University of Toronto. We are thankful to SciNet for allowing us to use their HPC systems. In particular, to Danny Gruner, Ramses van Zon, Vladimir Slavnic and Norbert Krawiec, for their continuous support and hospitality.

References

[1] 2007. *The Schwarzschild Solution and Black Holes*. Springer New York, New York, NY, 215–263. doi:10.1007/978-0-387-69200-5_10

[2] K et al. Akiyama. 2019. First M87 Event Horizon Telescope Results. I. The Shadow of the Supermassive Black Hole. *The Astrophysical Journal Letters* 875, 1 (April 2019), L1. doi:10.3847/2041-8213/ab0ec7

[3] Carson Brownlee, John Patchett, Li-Ta Lo, David DeMarle, Christopher Mitchell, James Ahrens, and Charles D. Hansen. 2012. A Study of Ray Tracing Large-scale Scientific Data in Two Widely Used Parallel Visualization Applications. In *Eurographics Symposium on Parallel Graphics and Visualization*, Hank Childs, Torsten Kuhlen, and Fabio Marton (Eds.). The Eurographics Association. doi:/10.2312/EGPGV/EGPGV12/051-060

[4] L. Dagum and R. Menon. 1998. OpenMP: an industry standard API for shared-memory programming. *IEEE Computational Science and Engineering* 5, 1 (1998), 46–55. doi:10.1109/99.660313

[5] Ali Can Demiralp, Marcel Krüger, Chu Chao, Torsten W. Kuhlen, and Tim Gerrits. 2022. Astray: A Performance-Portable Geodesic Ray Tracer. In *Vision, Modeling, and Visualization*, Jan Bender, Mario Botsch, and Daniel A. Keim (Eds.). The Eurographics Association. doi:10.2312/vmv.20221208

[6] et al. Event Horizon Telescope Collaboration. 2022. First Sagittarius A* Event Horizon Telescope Results. I. The Shadow of the Supermassive Black Hole in the Center of the Milky Way. *The Astrophysical Journal Letters* 930, 2 (may 2022), L12. doi:10.3847/2041-8213/ac6674

[7] Brian Gough. 2009. *GNU Scientific Library Reference Manual - Third Edition* (3rd ed.). Network Theory Ltd.

[8] Frank Imbens. 2023. Graphical Processing of Geodesic Propagation in Python. arXiv:2310.15183 [gr-qc] https://arxiv.org/abs/2310.15183

[9] Oliver James, Eugénie von Tunzelmann, Paul Franklin, and Kip S Thorne. 2015. Gravitational lensing by spinning black holes in astrophysics, and in the movie Interstellar. *Classical and Quantum Gravity* 32, 6 (feb 2015), 065001. doi:10.1088/0264-9381/32/6/065001

[10] Message Passing Interface Forum. 2023. *MPI: A Message-Passing Interface Standard Version 4.1.* https://www.mpi-forum.org/docs/mpi-4.1/mpi41-report.pdf

[11] Ch.W. Misner, K.S. Thorne, and J.A. Wheeler. 1973. *Gravitation*. W. Freeman.

[12] ESA/NASA Hubble Mission. Accessed: Feb. 2, 2025.. New view of the Pillars of Creation — visible. https://esahubble.org/images/heic1501a/

[13] Jon Peddie. 2019. *Applications of Ray Tracing*. Springer International Publishing, Cham, 91–128. doi:10.1007/978-3-030-17490-3_6

[14] Jon Peddie. 2019. *Ray-Tracing Hardware*. Springer International Publishing, Cham, 129–180. doi:10.1007/978-3-030-17490-3_7

[15] Marcelo Ponce, Ramses van Zon, Scott Northrup, Daniel Gruner, Joseph Chen, Fatih Ertinaz, Alexey Fedoseev, Leslie Groer, Fei Mao, Bruno C. Mundim, Mike Nolta, Jaime Pinto, Marco Saldarriaga, Vladimir Slavnic, Erik Spence, Ching-Hsing Yu, and W. Richard Peltier. 2019. Deploying a Top-100 Supercomputer for Large Parallel Workloads: the Niagara Supercomputer. In *Practice and Experience in Advanced Research Computing 2019: Rise of the Machines (Learning)* (Chicago, IL, USA) *(PEARC '19)*. Association for Computing Machinery, New York, NY, USA, Article 34, 8 pages. doi:10.1145/3332186.3332195

[16] Aniket Sharma, Lia Medeiros, Chi kwan Chan, Goni Halevi, Patrick D. Mullen, James M. Stone, and George N. Wong. 2023. Mahakala: a Python-based Modular Ray-tracing and Radiative Transfer Algorithm for Curved Space-times. arXiv:2304.03804 [astro-ph.HE] https://arxiv.org/abs/2304.03804

[17] Peter Shirley. 2020. *Ray Tracing in One Weekend* (3.0.2 ed.). raytracing.github.io/books/RayTracingInOneWeekend.html

[18] NASA Scientific Visualization Studio. Accessed: Feb. 6, 2025.. CANDELS UDF. https://svs.gsfc.nasa.gov/31020/

[19] I Wald, GP Johnson, J Amstutz, C Brownlee, A Knoll, J Jeffers, J Günther, and P Navratil. 2017. OSPRay - A CPU Ray Tracing Framework for Scientific Visualization. *IEEE Transactions on Visualization and Computer Graphics* 23, 1 (2017), 931–940. doi:10.1109/TVCG.2016.2599041