The Warped Universe: A Voyage into Relativistic Path Tracing

Samuel Joel Jackson
Dept. of ICT and Natural Sciences
Norwegian Uni. of Science and Technology
sam.jackson@ntnu.no

Ben David Normann
Dept. of ICT and Natural Sciences
Norwegian Uni. of Science and Technology
ben.d.normann@ntnu.no

Lars Ivar Hatledal
Dept. of ICT and Natural Sciences
Norwegian Uni. of Science and Technology
laht@ntnu.no

***Abstract*—Path tracing, a technique for simulating the trajectories of light, serves as a practical tool for visualizing light propagation through curved spacetime. In this work, we present an open-source framework for visualizing light paths around astrophysical phenomena such as black holes, leveraging geodesics derived from the Schwarzschild metric to simulate relativistic effects like gravitational lensing and light bending. The rendering engine supports full global illumination and utilizes Monte Carlo integration to produce accurate and physically realistic results. It incorporates optimizations to enhance both rendering performance and precision, achieving a balance between visual fidelity and computational efficiency. The framework can be extended to support more complex spacetimes, such as those described by the Kerr or Reissner-Nordström metrics. This work aims to provide a bridge between theoretical physics and visualization, offering a tool for exploring and communicating complex relativistic phenomena.**

***Index Terms*—path tracing, general relativity, geodesics**

I. Introduction

Physically Based Rendering (PBR) aims to accurately simulate the behavior of light by combining its physical properties with computationally efficient algorithms, striving to create visualizations indistinguishable from real-world imagery.

The fundamental principles of light propagation [1] form the foundation of path tracing and must be adhered to for accurate simulation:

1) Light propagates in straight lines through a medium, barring interactions.
2) All light originates from a defined source, such as the sun or artificial emitters.
3) Upon encountering a surface, light reflects in a manner dictated by the surface's Bidirectional Reflectance Distribution Function (BRDF).
4) Light passing through transparent materials is refracted in accordance with Snell's Law.
5) Light can illuminate surfaces indirectly by scattering off other objects, contributing to global illumination.

The first principle, that light travels in straight lines, is particularly intriguing. While it holds in classical optics, it seems at odds when considered within the framework of General Relativity, where spacetime curvature causes light to bend along geodesics. In order to accurately simulate the path of light along non-euclidean spacetime, we must take the respective metrics into account when computing the ray trajectories.

In this work, we propose an open-source solution aimed at providing visualizations of arbitrary geodesics, a world line that extends beyond the confounds of Euclidean geometry. While similar works exist, our eventual objective is to achieve real-time visualizations without compromising the accuracy of the physical models governed by General Relativity. To this end, we discuss advanced rendering techniques to accelerate the rendering process while maintaining fidelity to the underlying equations.

In Section 2, we review previous work, highlighting key advancements and identifying what we believe to be the current state-of-the-art in cosmological rendering. Section 3 delves into the fundamental equations of light transport and underlying physics of geodesics. Our methods and experimental setup are detailed in Section 4. In Section 5, we present our findings, followed by a discussion in Section 6, where we place our results in the context of prior research and outline potential directions for future work.

II. Related Work

A. Formulations of the relativistic ray tracing algorithm

Multiple methods exist for tracing geodesic paths, with one approach described by Fuerst et al. [2], which utilizes ray tracing to model photon trajectories in curved spacetime. In this method, the photon's worldline, or null geodesic [3] is traced by integrating its position and momentum along the geodesic using a numerical solver, such as the Runge-Kutta method. The ray tracing algorithm takes into account the optical depth for each frequency, which is integrated as the photon moves from the observer to the emitting surface.

B. Relativistic ray tracing in popular science and media

Perhaps one of the most notable uses of geodesic ray tracing in recent times was for the film Interstellar. As detailed in Thorne's book 'The Science of Interstellar' [4], director Christopher Nolan's vision was to create scientifically accurate and visually stunning depictions of astrophysical phenomena. This required strict adherence to the laws of General Relativity while rendering visuals at IMAX quality. The techniques outlined in [5] led to the development of the Double Negative

Gravitational Renderer (DNGR), the first renderer to simulate curved spacetime using light-beams, or bundles of rays, rather than infinitely thin ray paths.

The approach for DNGR, described in [6] and [7] employs advanced ray tracing techniques in Kerr spacetime, utilizing Boyer-Lindquist coordinates and locally non-rotating observers (FIDOs). It maps light rays from the celestial sphere to the camera's local sky by solving the null geodesic equation and accounts for effects like Doppler and gravitational redshift. DNGR enhances accuracy by tracing small ray bundles, calculating their distortion and light intensity shifts, and applying novel spatial and temporal filtering for IMAX-quality visuals.

DNGR was later adapted for use in scientific research, yielding new insights into gravitational lensing patterns near spinning black holes and hypothetical wormholes.

C. GPU accelerated Relativistic ray tracing

Over the past decade, the use of GPUs to accelerate relativistic computations has been well documented, demonstrating significant enhancements in ray tracing performance. Chan et al. developed GRay [8], a massively parallel integrator implemented in CUDA /C++, achieving performance approximately two orders of magnitude faster than CPU-based ray tracers. This advancement enables the efficient exploration of large parameter spaces, facilitating more accurate theoretical predictions of astrophysical phenomena.

Similarly, Pu et al. have proposed Odyssey [9] for producing fast and efficient processing of geodesic computations and general-relativistic radiative transfer (GRRT) calculations in Kerr spacetime. This code utilizes Runge-Kutta integration to attain speeds exceeding 1 nanosecond per photon per iteration, making it a powerful tool for studying the images and spectra of black holes at horizon scales.

D. State of the art

Mandal et al. have developed a methodology for rendering multiple spacetime metrics using full global illumination [11]. Their approach includes solutions for the Schwarzschild and Kerr metrics, as well as the Ellis Wormhole solution. Their renderer is capable of generating a range of phenomena, such as caustics, soft shadows, and secondary ray effects. To the best of our knowledge, this represents the most recent effort to integrate non-linear rendering with Monte Carlo path tracing. Our work aims to replicate the results of this study, providing an open-source framework that documents the steps and processes involved. Our goal is to create a framework that others can build upon and expand in the future.

III. Theory

In this section, we document the theoretical foundation of our path tracer and the equations of motion needed for tracing curved trajectories. The theory and equations presented are derived from prior work, but we include them here for clarity and ease of reference. This discussion is divided into two parts: first, we explore the principles of path tracing, followed by an overview of the geodesic equations relevant to non-Euclidean geometry.

A. The Rendering Equation

The rendering equation, also known as the light transport equation, was first introduced in 1986 [10]. It is expressed as:

$$L_o(\mathbf{x}, \omega_o) = L_e(\mathbf{x}, \omega_o) + \int_\Omega f_r(\mathbf{x}, \omega_i, \omega_o) L_i(\mathbf{x}, \omega_i)(\omega_i \cdot \mathbf{n})\, d\omega_i \quad (1)$$

This equation calculates the total reflected radiance at a point in a scene. For a given point, $\mathbf{x}$, the total outgoing radiance, $L_o(\mathbf{x}, \omega_o)$ in the direction ω_o is the sum of two components:

1) the emitted radiance, $L_e(\mathbf{x}, \omega_o)$, which is the light emitted directly from point $\mathbf{x}$,
2) the reflected radiance, computed as an integral over the surface hemisphere Ω at point $\mathbf{x}$. This accounts for the incident radiance, $L_i(\mathbf{x}, \omega_i)$ which is multiplied by the BRDF of the surface $f_r(\mathbf{x}, \omega_i, \omega_o)$ and the cosine of the angle between the surface normal $\mathbf{n}$ and the incident angle ω_i.

The BRDF encodes the material properties at a point on a given surface. It describes how much of the incident light is reflected in another direction, as well as the frequency of the light and other factors. More generally, the Bidirectional Scattering Distribution Function (BSDF) takes into account the transparency and translucency of the material. In our renderer, we demonstrate three types of surfaces: diffuse, reflective (e.g. mirror), and dielectric (e.g. glass). Each type determines how light interacts with the surface, influencing its appearance and behavior.

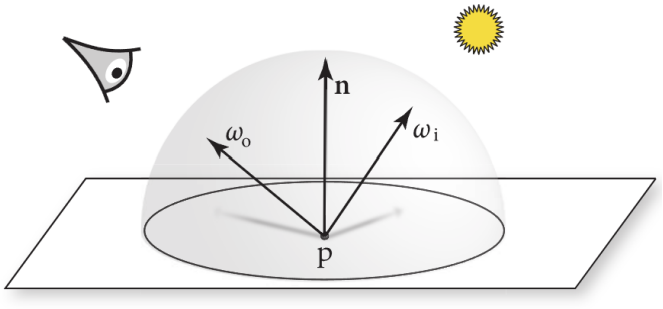

Fig. 1. Incident light from a specific direction interacts with the surface and is reflected toward the observer. Figure accessed from [1]

The rendering equation forms the foundation of our framework, enabling uniform sampling of the scene to achieve an accurate representation of lighting. To solve the equation, we use the Monte Carlo integration method, which employs unbiased sampling techniques to effectively approximate global illumination. This approach captures realistic light transport phenomena, including reflections, refractions, and diffuse scattering, with high precision.

However, this method has a significant drawback: increased computation time. Light rays must scatter multiple times before reaching a light source, and the stochastic nature of

the process often results in wasted computations, as many light paths terminate without ever connecting to a source. Not using enough samples results in a noisy image. To achieve high image quality, a sufficient number of samples per pixel must be used. The time complexity of this process is $O(N)$, where N represents the number of samples taken.

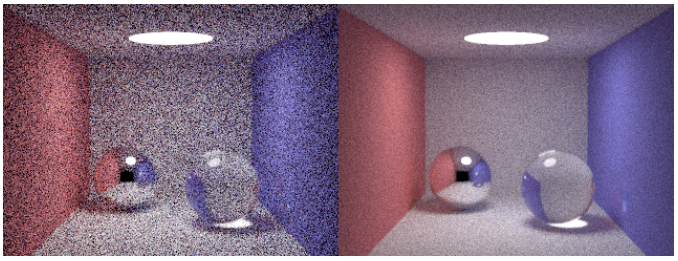

Fig. 2. Two images captured using our renderer: The image on the left was rendered at 60 samples per pixel (spp), taking less than 1 second to generate. In contrast, the image on the right was rendered at 800 spp, requiring 8 seconds to complete.

Instead of increasing the number of samples, alternative approaches can be utilized to improve efficiency while maintaining high-quality results. Techniques such as Multi Importance Sampling (MIS) and Next Event Estimation (NEE) focus on reducing variance by directing computational effort toward the most significant contributions to the lighting equation. These methods prioritize sampling directions and regions that are more likely to capture meaningful light interactions, such as areas of high luminance, key light sources or from the material BRDF.

B. The geodesic equations

Here, we present the underlying equations of motion for our system. For our renderer, we use the Schwarzschild Metric as a simple test case. This metric describes the trajectory of light around a non-rotating, spherically symmetric black hole. Notably, implementing spacetime metrics does not affect the fundamental workings of the rendering equation, only how we calculate the light trajectories. All equations presented below are cited from [11], we have included only the necessary equations, please refer to this paper for a complete derivation.

The geometry surrounding Schwarzschild metric in isotropic cartesian coordinates is given by

$$\mathrm{d}s^2 = -\left(\frac{1-\frac{r_s}{4r}}{1+\frac{r_s}{4r}}\right)^2 \mathrm{d}t^2 + \left(1+\frac{r_s}{4r}\right)^4 (\mathrm{d}x^2+\mathrm{d}y^2+\mathrm{d}z^2) \tag{2}$$

where ds is the spacetime metric, $r_s = 2GM$ is the Schwarzschild radius. We choose natural units for our calculations ($G = c = 1$). r is the radial distance from the centre of the black hole to a point in the scene.

Constructing the geodesic equations of motion requires the use of the Hamiltonian for numerical integration. The positional and momentum components along the geodesic line are given by

$$\frac{\mathrm{d}x^\alpha}{\mathrm{d}\zeta} = \frac{\partial H}{\partial p_\nu} = g^{\alpha\nu}p_\nu \tag{3}$$

$$\frac{\mathrm{d}p_\alpha}{\mathrm{d}\zeta} = -\frac{\partial H}{\partial x^\alpha} = -\frac{1}{2}\frac{\partial g_{\mu\nu}}{\partial x^\alpha}p^\mu p^\nu \tag{4}$$

Within equations 3 and 4, ζ denotes some affine parameter along the geodesic line. $\frac{\mathrm{d}p_\alpha}{\mathrm{d}\zeta}$ denotes the change in the covariant momentum with respect to the affine parameter. $\frac{\mathrm{d}x^\alpha}{\mathrm{d}\zeta}$ is derivative of the metric tensor with respect to the spacetime coordinates, representing how the geometry varies spatially, $\alpha \in \{0,1,2,3\}$.

Our covariant metric tensor, $g^{\alpha\nu}$, is given by eq. 2 for the Schwarzschild metric by the following matrix,

$$g_{\mu\nu} = \begin{bmatrix} \left(-\frac{1-\frac{r_s}{4r}}{1+\frac{r_s}{4r}}\right)^2 & 0 & 0 & 0 \\ 0 & \left(1+\frac{r_s}{4r}\right)^4 & 0 & 0 \\ 0 & 0 & \left(1+\frac{r_s}{4r}\right)^4 & 0 \\ 0 & 0 & 0 & \left(1+\frac{r_s}{4r}\right)^4 \end{bmatrix}$$

Solving for the time components first, using conservation of energy and momentum, eqs. 3 and 4 become,

$$\frac{\mathrm{d}x^0}{\mathrm{d}\zeta} = \left(\frac{1-\frac{r_s}{4r}}{1+\frac{r_s}{4r}}\right)^2 p_0 \tag{5}$$

$$\frac{\mathrm{d}p_0}{\mathrm{d}\zeta} = 0 \tag{6}$$

Substituting $\frac{\mathrm{d}x^0}{\mathrm{d}\zeta}$ using the product rule, the spatial components can be expressed with respect to t (x^0). The equations of motion for the spatial components are given by

$$\frac{\mathrm{d}x^\alpha}{\mathrm{d}t} = \frac{\left(1-\frac{r_s}{4r}\right)^2}{\left(1+\frac{r_s}{4r}\right)^6}p_\alpha \tag{7}$$

$$\frac{\mathrm{d}p_\alpha}{\mathrm{d}t} = -\frac{r_s}{2r^3}\left(\frac{p^2\left(1-\frac{r_s}{4r}\right)^2}{\left(1+\frac{r_s}{4r}\right)^7} + \frac{1}{1-(\frac{r_s}{4r})^2}\right)x_\alpha \tag{8}$$

Here, $p^2 = p_x^2 + p_y^2 + p_z^2$ represents the squared magnitude of the momentum vector. Equations (7) and (8) describe the ray's position and momentum, respectively, at each point along the geodesic. To compute these, we use numerical methods to integrate the differential equations. The initial conditions for the ray are set based on the camera's position and its direction vector. We found that normalizing the momentum helps with the stability of the system over time.

IV. Method

In this iteration, we focus on targeting the CPU to concentrate on the implementation and gain a deeper understanding of the underlying principles of non-linear path tracing. For subsequent iterations, we will switch to the GPU to take advantage of the highly parallelized processing.

The path tracer was developed in Rust, a modern and efficient language that enabled rapid prototyping of our system. To optimize performance, we utilized the Rayon crate for parallel processing, leveraging multiple CPU threads to significantly accelerate rendering times. Our tracer is based upon smallpt, a global illumination renderer in 99 lines of code.

The development was split into two parts, firstly implementing the Monte Carlo integration method to build up our renderer, followed by the geodesic implementation.

A. The Path Tracer Setup

We position the camera in the scene at coordinates $0, 0, 270$ and orient it with a direction vector of approximately $0, 0, -1$, aligning it to look down the z-axis. The canvas dimensions are specified for the rendering, and for testing purposes, we use an area of 320x240 pixels. A ray is defined for each pixel, originating from the camera's position and passing through the center of the corresponding pixel.

We compute the total radiance accumulated for each pixel using Eq. 1. To achieve a detailed and visually accurate image, we define the number of samples per pixel (spp). To improve the sampling distribution, we apply a tent filter, which helps reduce aliasing artifacts and enhances the smoothness of the final image. Our radiance function iteratively calculates the contribution from successive bounces within the scene, using a maximum depth parameter to limit the number of bounces. To balance computational efficiency, Russian Roulette is employed to terminate paths that contribute minimally to the final radiance. For intersection testing, we use the equation for a sphere, and upon a successful intersection, we update the ray direction according to the BRDF of the material at the intersection point.

When sampling, we use a tent filter to eliminate frequencies above the Nyquist limit, thereby reducing aliasing and sampling artifacts in the image. For diffuse materials, we sample the surface hemisphere around the first hit point utilizing a cosine weighted sampling technique for diffuse BRDFs. We assume an ideal specular (mirror) BRDF for reflective materials, and refractive materials are calculated in accordance to Snells Law. Our Ray object is encapsulated within a Geodesic, which extends its functionality by enabling conversions between Cartesian and spherical coordinate systems and providing useful tools for angular momentum calculations.

We present the pseudo-code for our Path Tracer in Algorithm 1,

In order to speed up the computation on the CPU, we use Rust's Rayon crate, which allows the algorithm to use CPU threads in parallel.

Algorithm 1 Path tracing algorithm

```
for each chunk of 100 pixels in data (parallel) do
    Initialize sampler
    for each pixel (p) do
        for sy = 0 → 1, sx = 0 → 1 do
            radiance ← (0, 0, 0)
            for each sample in num_samples do
                (dx, dy) ← tent_filter
                d ← compute_direction
                ray ← initialize_geodesic_ray
                compute radiance
            end for
            output ← output + clamp(radiance) · 0.25
        end for
    end for
end for
```

B. Intersections in Spacetime

In flat (Minkowski) spacetime, scene intersections are computed directly by analyzing the ray's direction and solving for the roots of all objects in the scene. However, in Schwarzschild and other non-linear spacetimes, a ray-marching algorithm is employed to iteratively approach objects before performing intersection tests. This method relies on the assumption that local spacetime on a manifold is approximately flat, reducing to Minkowski spacetime at small scales.

Our solution approximates ray paths using numerical integration of equations (7) and (8), as analytic solutions are not viable in this context. We employ the Runge-Kutta 4 (RK4) algorithm to trace the null geodesic, updating the ray's position and direction at each step. Intersection tests are performed iteratively, considering only those below a designated threshold. The ray advances as close as possible to an object in the scene before recording a successful intersection, with its position approximating the object's surface.

To enhance precision and efficiency, we augment the RK4 algorithm with an adaptive step size (time step), dynamically adjusting the step length based on the ray's proximity to objects. This ensures finer granularity near surfaces and improves computational performance in less complex regions. We outline our approach in Algorithm 2.

We experimented with various step sizes and sigma values, determining that an initial step size of 10 and a sigma of 0.1 provided an optimal balance between computational efficiency and accuracy for our Cornell Box scene.

C. Further Requirements and Expectations

We can now integrate both sections. The path tracer should function as expected in the absence of a gravitational field, for example, by setting the mass to zero. Additionally, it should accurately follow geodesics when reflecting or refracting off surfaces, ensuring that all materials behave consistently with the underlying metric.

Algorithm 2 Tracing the Null Geodesic

function TRACEGEODESIC(geo, t, id, accretion_disk)
  $t \leftarrow \infty$, $step\_size \leftarrow 10$, $sigma \leftarrow 1e-1$
  $max\_iter \leftarrow 300$
  for $iteration \leftarrow 1$ to max_iter **do**
    $hit \leftarrow$ **false**
    for each $sphere$ in $scene$ **do**
      $d \leftarrow sphere$.INTERSECT$(geo.ray)$
      if $0 < d < step\_size$ **then**
        if $d < sigma$ **then**
          $t \leftarrow d$, $id \leftarrow sphere\ index$
          return true
        else
          $hit \leftarrow$ **true**
          $step\_size \leftarrow step\_size \times 0.5$
          break
        end if
      end if
    end for
    if not hit **then**
      $geo \leftarrow$ RK4$(geo, step_size)$
      $step\_size \leftarrow \min(step\_size \times 1.05, 20)$
    end if
  end for
  $t \leftarrow \infty$
  return false
end function

V. RESULTS

To validate our implementation, we propose several tests.

1) Directly visualize the rays near a black hole to ensure they bend correctly.
2) Visualize an accretion disk surrounding the black hole at the center of the scene.
3) Test black holes with varying Schwarzschild radii to examine the effects on light bending with the mass of the black hole.
4) Render the black hole in several locations around the scene, including inside the spheres.

A. Direct ray visualization

Figure 3 illustrates our simulation of rays cast from the camera positioned at (0,-10), uniformly distributed across angles from $-\frac{\pi}{2}$ to $+\frac{\pi}{2}$ along the z-axis. Far from the black hole, the rays propagate in straight lines. However, as they approach the black hole, the curvature of spacetime causes them to bend toward its center. Rays that enter within the Schwarzschild radius (the event horizon) are inevitably drawn into the black hole, unable to escape due to the extreme gravitational pull.

Rays cast just outside the Schwarzschild radius experience significant gravitational influence, bending sharply and appearing to follow paths almost tangential to their initial directions. This effect demonstrates the extreme warping of spacetime near the black hole.

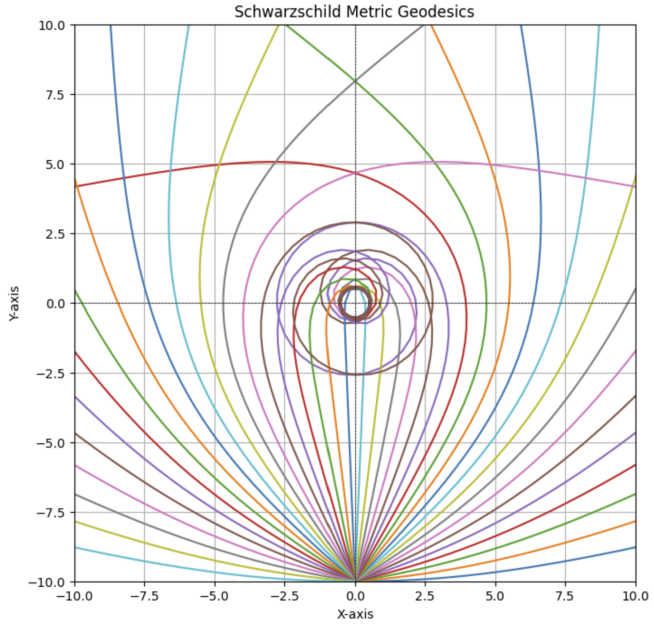

Fig. 3. Rays directed from the camera position and towards a black hole of Schwarzschild radius 5 units.

B. Rendering an Accretion Disk

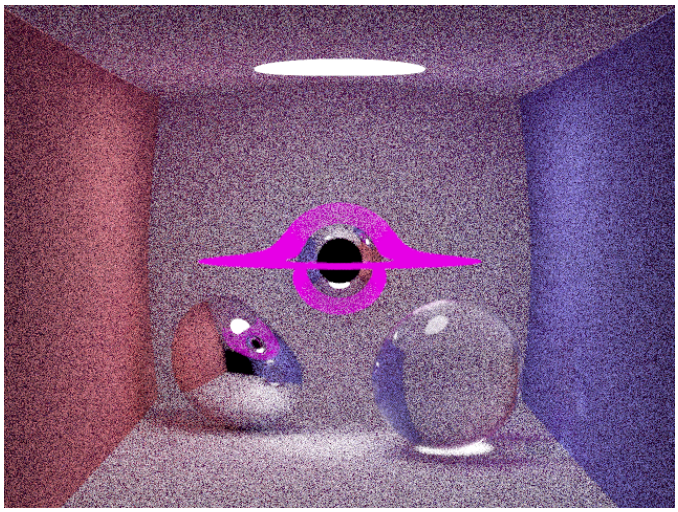

Fig. 4. Our scene, rendering a Schwarzschild black hole with an accretion disk sorrounding it.

An accretion disk is a rotating structure of gas, dust, or other matter spiraling toward a massive object, such as a black hole or neutron star. It forms due to gravitational attraction and angular momentum, with material heating up as it moves inward, emitting radiation from infrared to X-rays. We can demonstrate this phenomenon with our renderer, and use it to further validate our implementation.

The disk in Figure 4 was simulated by selecting rays near the black hole's equatorial plane (y-axis) and within a radial distance of 10 to 30 units from its center. The render appears

to be consistent with features typically found in real images of accretion disks, further validating our renderer.

C. Rendering the scene

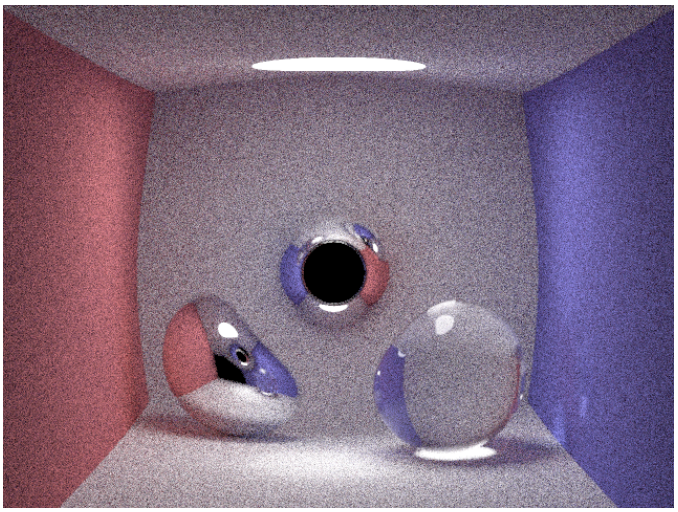

Fig. 5. Our scene, rendering a Schwarzschild black hole within a Cornell Box, featuring a mirror sphere, and a glass sphere. Render Time: 361s

Figure 5 depicts our scene rendered within a Cornell Box under full global illumination. A massive black hole is positioned at the center of the scene, relative to the box's walls. The gravitational field of the black hole perturbs the walls and ceiling, visibly bending light around it. At small angular distances from the black hole, the light bending is sufficient to redirect rays onto the opposite walls, creating a characteristic ring of reflected rays.

The reflective sphere on the right of the image is significantly influenced by the gravitational field, with its surface reflecting the distorted surroundings. Both the reflective and transparent spheres reflect and refract their environment, including the warped projections of the scene, highlighting the effects of global illumination and gravitational lensing.

The lensing effect can also be visualized by changing the mass of the black hole.

As shown in figure 6, larger radii, and hence a larger mass for the black hole lead to more prominent distortions in the surrounding scene. The values for the Schwarzschild radius from top left to bottom right are r = 0,1,2,3,4,5. Notice the lack of any distortions when r=0. This is because the metric breaks down to Minkowski spacetime in the absence of mass.

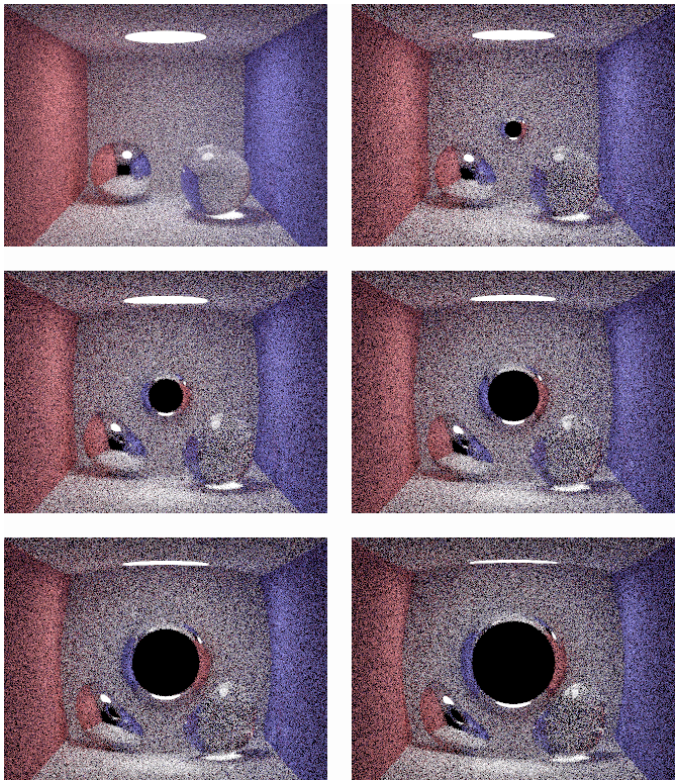

Fig. 6. Our scene, rendering a Schwarzschild black hole with different mass.

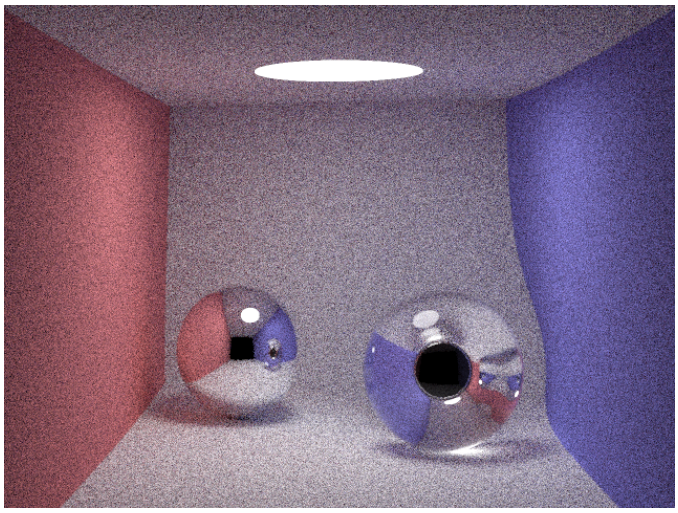

Fig. 7. Center of black hole positioned within the glass sphere.

D. Rendering features of the Schwarzschild Metric

Positioning the center of the black hole at various locations within the scene results in different patterns of distortion, as the gravitational lensing effect reshapes the paths of light rays and subsequently alters the reflections, refractions, and projections observed on the surrounding surfaces and objects.

Figure 7 shows light passing through the transparent glass sphere affected by both the sphere's refractive properties and the intense gravitational field of the black hole. This dual effect causes complex distortions of the scene as the light is refracted and bent by the black hole's gravitational influence.

The reflective inner surface of the glass sphere interacts with the bending of light around the black hole, causing reflections that appear inverted or stretched. These reflections capture the warped walls, ceiling, and other objects in the scene, which are pulled towards the black hole.

The curvature of the glass sphere causes light rays entering the sphere to bend further as they interact with the black

hole's gravitational field. This bending creates caustic patterns (focused regions of light), depending on the alignment of the light source, the glass, and the black hole.

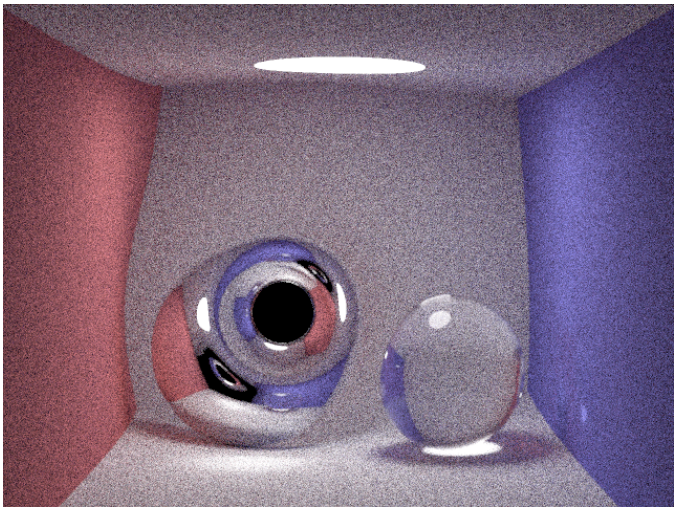

Fig. 8. Center of black hole positioned within the mirror sphere. Render Time: 353s

To produce figure 8, we offset the center of the black hole to the position of the reflective mirror sphere. Due to the black hole's offset position, the reflected distortions create a sense of depth and motion to the sphere. The reflected walls and ceiling seem to stretch and curve around the black hole's location, producing the envelope of material surrounding the black hole.

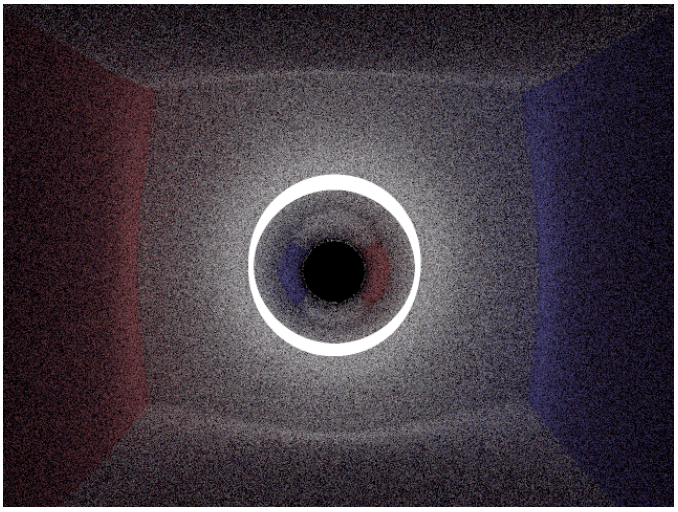

Fig. 9. Center of black hole positioned within the mirror sphere. Render time: 292s

Figure 9 depicts a scene where a black hole is centrally positioned, with a light source of radius 4 units placed directly behind it. The black hole introduces strong gravitational lensing effects, bending light from the source into a bright ring surrounding the central singularity. This phenomenon is known as the Einstein ring, a result of light being curved around the black hole due to its intense gravitational field.

E. Light Source Sampling (Importance Sampling)

Finally, we demonstrate our scene using a type of importance sampling, which directly samples the light source by casting a shadow ray from the point of intersection on a surface. Figure 10 illustrates the resulting importance-sampled scene. A notable error is evident in the rendering: the light is sampled directly using the Euclidean distance from the intersection point, rather than accounting for its true position along the geodesic path. Calculating the exact position would require an analytical solution to the geodesic equations, which is not feasible in this context. As a result, the image appears significantly darker.

Furthermore, a consequence of this type of sampling is the introduction of bright artifacts in the scene, known as fireflies. They are caused by numerical instabilities when solving the rendering equations and are distinguishable from general noise. Unlike noise however, increasing the samples does not reduce these fireflies.

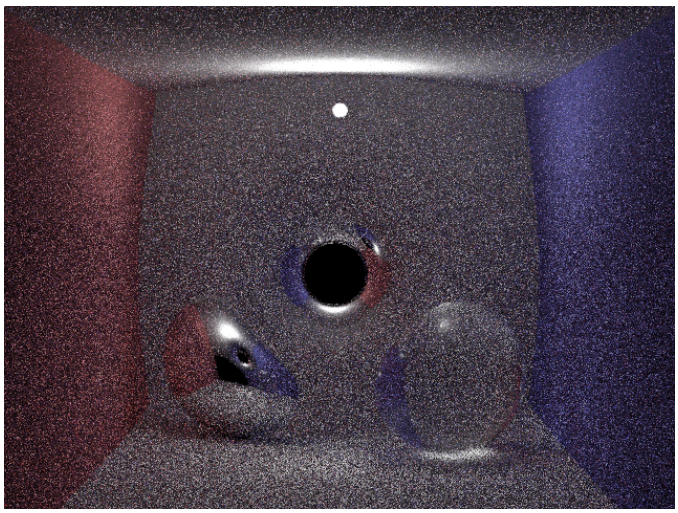

Fig. 10. Importance sampled scene. Render time: 26s

VI. Discussion

The scenes above were rendered using an Intel i7 10875H CPU at a resolution of 640x480 pixels with 200 samples per pixel (spp). Rendering times varied depending on factors like scene complexity and the metric's size and position. For example, the Cornell Box typically required 200–400 seconds to render. Performance could be significantly improved by employing a more dynamic step size adjustment method, such as the Runge-Kutta-Fehlberg (RKF45) algorithm, which adapts the step size automatically based on estimated output error. The resulting renders are visually consistent with those reported in prior studies on this topic.

Despite the observed errors in the importance-sampled render, the rendering time for this image was reduced to under a minute. By lowering the number of samples from 200 to just 8 spp, we were able to maintain comparable visual quality. This considerable speedup highlights the effectiveness of adopting a more stratified sampling approach.

VII. Conclusion

Our renderer demonstrates the capability to simulate space-time geometries, starting with a simple test case based on the Schwarzschild metric. Our objective is to extend this functionality to more complex scenarios, including the Kerr metric, and ultimately support arbitrary geodesic paths. Additionally, we aim to achieve these simulations on timescales approaching real-time performance with user input to control the scene. This undertaking is non-trivial, requiring significant research into computational efficiency and advanced rendering techniques, such as acceleration structures, GPU processing and more efficient sampling methods such as Multi-Importance Sampling (MIS) and Metropolis Light Transport (MLT).

As demonstrated from our importance sampling test, choosing the samples wisely has the potential to significantly speed up the rendering time requiring far fewer samples to be taken. This is a particular area that can be examined further. Whilst importance sampling has limitations in the context of non linear path tracing, MLT or similar methods could provide us with a useful tool. In the case of MLT, which is based upon Markov Chain Monte Carlo (MCMC) technique, the path tracing algorithm is set to converge on paths that contribute most to the final image.

Further optimizations, such as implementing acceleration structures like the Bounding View Hiararchy (BVH), can help identify regions of the scene where ray paths are intersecting, reducing the number of intersection tests needed. This is especially beneficial for more complex scenes composed of thousands or millions of triangles.

Perhaps the most crucial advancement for optimizing our framework for real-time rendering is the integration of CUDA or a graphics library like Vulkan or WebGPU to parallelize computations. Utilizing compute shaders would be a logical next step in advancing our renderer.

Next Steps

We have an almost fully implemented Kerr metric in our repository, but unfortunately, we were unable to finalize it in time for submission. The current implementation is available and should require minimal effort to complete. Moving forward, our goal is to port this implementation to the GPU, which could potentially achieve a performance boost of up to 100x. To accomplish this, we plan to explore using either Unity with compute shaders or a CUDA-based solution. Additionally, we intend to investigate advanced sampling techniques to further optimize computational efficiency. We also explore novel techniques such as Physics Informed Neural Networks (PINN) as an efficient way to approximate the metrics.

Source Code

The source code for this project is available at: https://github.com/skcajs/katerina

References

[1] M. Pharr, W. Jakob, and G. Humphreys, Physically Based Rendering: From Theory to Implementation, 4th ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2023, ISBN 9780262048026.

[2] S. V. Fuerst and K. Wu, "Radiation transfer of emission lines in curved space-time," Astronomy & Astrophysics, vol. 424, no. 3, pp. 733–746, Sep. 2004, DOI: 10.1051/0004-6361:20035814.

[3] R. J. Low, "The geometry of the space of null geodesics," Journal of Mathematical Physics, vol. 30, no. 4, pp. 809–811, Apr. 1989, DOI: 10.1063/1.528401.

[4] K. S. Thorne and C. Nolan, The science of Interstellar. New York: W.W. Norton & Company, 2014.

[5] J. James, O. Oliver, S. Dieckmann, S. Pabst, P.-G. H. Roberts, and K. S. Thorne, Building Interstellar's Black Hole: The Gravitational Renderer. ACM SIGGRAPH 2015 Talks, 2015, n. pag.

[6] O. James, E. von Tunzelmann, P. Franklin, and K. S. Thorne, Gravitational lensing by spinning black holes in astrophysics, and in the movie Interstellar, vol. 32, no. 6, Classical and Quantum Gravity. IOP Publishing, Feb. 2015, pp. 065001. DOI: 10.1088/0264-9381/32/6/065001.

[7] O. James, E. von Tunzelmann, P. Franklin, and K. S. Thorne, "Visualizing Interstellar's Wormhole," arXiv preprint, 2015, n. pag. DOI: 10.1119/1.4916949, Available at arXiv.

[8] C.-K. Chan, D. Psaltis, and F. Özel, "GRay: A massively parallel GPU-based code for ray tracing in relativistic spacetimes," The Astrophysical Journal, vol. 777, no. 1, p. 13, Oct. 2013, DOI: 10.1088/0004-637x/777/1/13.

[9] H.-Y. Pu, K. Yun, Z. Younsi, and S.-J. Yoon, "ODYSSEY: A public GPU-based code for general relativistic radiative transfer in Kerr spacetime," The Astrophysical Journal, vol. 820, no. 2, p. 105, Mar. 2016, DOI: 10.3847/0004-637X/820/2/105.

[10] J. T. Kajiya, "The rendering equation," in Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '86), D. C. Evans and R. J. Athay, Eds., pp. 143–150, 1986, DOI: 10.1145/15922.15902, ISBN: 978-0-89791-196-2.

[11] A. Mandal, K. Ayush, and P. Chaudhuri, "Non-linear Monte Carlo ray tracing for visualizing warped spacetime," in Proceedings of VISIGRAPP, 2021.