The Astrophysical Journal, 777:13 (9pp), 2013 November 1 doi:10.1088/0004-637X/777/1/13

GRay: A MASSIVELY PARALLEL GPU-BASED CODE FOR RAY TRACING IN RELATIVISTIC SPACETIMES

Chi-kwan Chan[1,2], Dimitrios Psaltis[1,3], and Feryal Özel[1,3,4]
[1] Department of Astronomy, University of Arizona, 933 N. Cherry Ave., Tucson, AZ 85721, USA
[2] NORDITA, KTH Royal Institute of Technology and Stockholm University, Roslagstullsbacken 23, SE-106 91 Stockholm, Sweden
[3] Institute for Theory and Computation, Harvard-Smithsonian Center for Astrophysics, 60 Garden St., Cambridge, MA 02138, USA
[4] Radcliffe Institute for Advanced Study, Harvard University, 8 Garden St., Cambridge, MA 02138, USA
Received 2013 March 20; accepted 2013 June 4; published 2013 October 9

ABSTRACT

We introduce GRay, a massively parallel integrator designed to trace the trajectories of billions of photons in a curved spacetime. This graphics-processing-unit (GPU)-based integrator employs the *stream processing* paradigm, is implemented in CUDA C/C++, and runs on nVidia graphics cards. The peak performance of GRay using single-precision floating-point arithmetic on a single GPU exceeds 300 GFLOP (or 1 ns per photon per time step). For a realistic problem, where the peak performance cannot be reached, GRay is two orders of magnitude faster than existing central-processing-unit-based ray-tracing codes. This performance enhancement allows more effective searches of large parameter spaces when comparing theoretical predictions of images, spectra, and light curves from the vicinities of compact objects to observations. GRay can also perform on-the-fly ray tracing within general relativistic magnetohydrodynamic algorithms that simulate accretion flows around compact objects. Making use of this algorithm, we calculate the properties of the shadows of Kerr black holes and the photon rings that surround them. We also provide accurate fitting formulae of their dependencies on black hole spin and observer inclination, which can be used to interpret upcoming observations of the black holes at the center of the Milky Way, as well as M87, with the Event Horizon Telescope.

Key words: gravitation – methods: numerical – radiative transfer

Online-only material: color figures

1. INTRODUCTION

The propagation of photons in the curved spacetimes around black holes and neutron stars determines the appearance of these compact objects to an observer at infinity as well as the thermodynamic properties of the accretion flows around them. This strong-field lensing imprints characteristic signatures of the spacetimes on the emerging radiation, which have been exploited in various attempts to infer the properties of the compact objects themselves.

As an example, special and general relativistic effects broaden fluorescence lines that originate in the accretion disks and give them the characteristic, asymmetric, and double-peaked profiles that have been used in inferring black hole spins in active galactic nuclei and in galactic sources (see Miller 2007 for a review). In recent years, this approach has provided strong evidence for rapid spins in black holes such as MCG 6-30-15 (Brenneman & Reynolds 2006) and 1H 0707−495 (Fabian et al. 2009) and is expected to mature even further with upcoming observations with Astro-H (Takahashi et al. 2012).

A similar application of strong-field lensing is encountered in modeling the images from the accretion flows around the black holes in the center of the Milky Way and M87 (e.g., Broderick et al. 2009; Dexter et al. 2009). In the near future, such lensing models will be crucial for interpreting imaging observations of these two sources with the Event Horizon Telescope (Doeleman et al. 2009).

Finally, strong-field lensing around a spinning neutron star determines the pulse profile generated from a hot spot on its surface. The pulsation amplitude in such a light curve depends sensitively on the compactness of the neutron star (Pechenick et al. 1983). For this reason, comparing model to observed pulsation light curves has led to coarse measurements of the neutron-star properties in rotation-powered (e.g., Bogdanov et al. 2007) and accretion-powered millisecond pulsars (Leahy et al. 2008) and bursters (e.g., Weinberg et al. 2001; Muno et al. 2002). This technique shapes the key science goals of two proposed X-ray missions, ESA's LOFT (Feroci et al. 2012) and NASA's NICER (Arzoumanian et al. 2009).

The general ray-tracing problem in a relativistic spacetime has been addressed by several research groups to date (e.g., Cunningham 1975; Pechenick et al. 1983; Laor 1991; Speith et al. 1995; Miller & Lamb 1998; Braje & Romani 2002; Dovčiak et al. 2004; Broderick 2006; Cadeau et al. 2007; Dexter & Agol 2009; Dolence et al. 2009; Psaltis & Johannsen 2012; Bauböck et al. 2012) following two general approaches. In one approach, which is only applicable to the Kerr spacetime of spinning black holes, several integrals of motion are used to reduce the order of the differential equations. In the other approach, which can be used both in the case of black holes and neutron stars, the second-order geodesic equations are integrated.

The Kerr metric is of Petrov-type D and, therefore, the Carter constant Q provides a third integral of motion along the trajectories of photons, making the first approach possible (see discussion in Johannsen & Psaltis 2010). Introducing a deviation from the Kerr metric, however, either in order to model neutron-star spacetimes, which can have different multipole moments, or to test the no-hair theorem of black holes, does not necessarily preserve its Petrov-type D and the Carter constant is no longer conserved along geodesics (actually, no such Killing tensor exists in spacetimes that are not of Petrov-type D). As a result, ray tracing in a non-Kerr metric requires integrating the second-order differential equations for individual geodesics. Our current algorithm based on Psaltis & Johannsen (2012), as well as those of Broderick (2006), Cadeau et al. (2007), and Dolence et al.

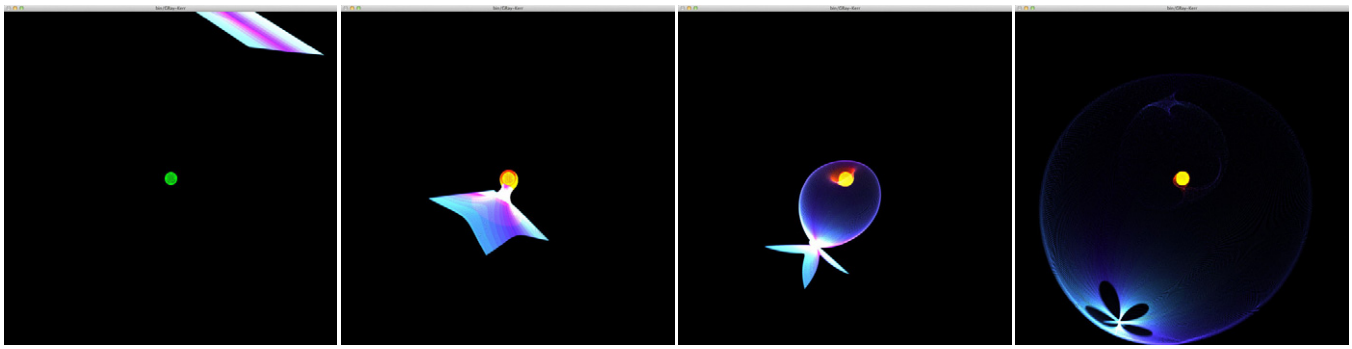

Figure 1. Four successive screen shots of GRay in its interactive mode, which allows users to visualize the photon positions in a Cartesian reference frame and to adjust the viewing angles in real time. We use a three-channel color scheme (RGB) to encode different properties of the photons. The red and blue color channels represent the k_t component of the photon momenta—the photons are redder for stronger and bluer for weaker gravitational redshift. The green color denotes the impact parameter $b_{\rm impact}$. For this particular calculation, we set up a plane-parallel grid of photons originating at a large distance from a spin 0.999 black hole, which is denoted by the green sphere in the first panel. This grid of photons is deformed as they pass near the black hole horizon in the second panel because of gravitational time dilation. Some of these photons are deflected by large angles in the strong field of the black hole and escape in a nearly isotropic distribution that forms the bubble shown in the third panel. Finally, the caustics in the black hole spacetime form the star shaped structure near the horizon in the fourth panel. In the same panel, the photons that are trapped near the event horizon form the yellow sphere near the center of the image. The photon data always reside on the graphics card memory and the visualization is done by CUDA-OpenGL interoperability (see Section 2). While the geodesic equations are integrated by CUDA, the coordinate transformation and particle rendering of the same data are both done by the OpenGL shader.

(A color version of this figure is available in the online journal.)

(2009), follow the latter approach, making them applicable to a wider range of astrophysical settings.

Although the latter algorithms are not limited by assumptions regarding the spacetimes of the compact objects and reach efficiencies of $\simeq 10^4$ geodesic integrations per second, they are still not at the level of efficiency necessary for the applications discussed earlier. For example, in order to simulate the X-ray characteristics of the accretion flow around a black hole, we need to calculate images and spectra from the innermost $\simeq 100$ gravitational radii around the black hole. In order to capture fine details (such as those introduced by rays that graze the photon orbit and can affect the detailed images and iron-line profiles; see, e.g., Johannsen & Psaltis 2010; Beckwith & Done 2005), we need to resolve the image plane with a grid spacing of $\leqslant 0.1$ gravitational radii. As a result, for a single image, we need to trace at least 10^6 geodesics. Even at the current best rate of $\simeq 10^4$ geodesic integrations per second, a single monoenergetic image at a single instant in time will require ~ 100 s on a fast workstation. This is prohibitively slow, if, for example, we aim to simulate time-variable emission from a numerical simulation or perform large parameter studies of black hole spins, accretion rates, and observer inclinations, when fitting line profiles to data.

A potential resolution to this bottleneck is calculating a large library of geodesics, storing them on the disk, and using them with an appropriate interpolation routine either in numerical simulations or when fitting data. To estimate the requirements for this approach, we consider, for the sake of the argument, a rather coarse grid on the image plane, spanning $100M$ in each direction, with a resolution of $1M$. In principle, we can refine this grid only for those impact parameters that correspond to geodesics that graze the photon orbit. In order, e.g., to integrate the radiative transfer equation for each one of these 10^4 geodesics that reach the image plane, we need to store enough information to reproduce the trajectory without recalculating it. Assuming a coarse resolution again, we may choose to store ~ 100 points per geodesic within the inner 100 gravitational radii. Along each point, we will need to store at least three components of the photon four-momentum (since we can always calculate the fourth component by the requirement that the photon traces a null geodesic). For single-precision storage (i.e., 4 bytes per number), we will need to store $4 \times 3 \times 100 \times 10^4 = 12$ MB of information per image. If we want now to use a rather coarse grid of ~ 30 values in black hole spin and ~ 30 values in the inclination of the observer, we need to make use of a $30 \times 30 \times 12$ MB $= 11$ GB database. Such a database can only be stored in a hard disk. At an average latency time of $\simeq 1$ ms for current disks, the efficiency of this approach cannot exceed $\sim 10^3$ geodesics per second (given that a typical disk sector has a size of at most 4 KB and can handle the data of no more than a few geodesics). This is actually comparable to and, in fact, lower than the efficiency one would achieve by calculating the geodesics in the first place. Note also that this estimate was performed for a very coarse grid.

The good news is that ray tracing in vacuum is a trivially parallelizable algorithm, as individual rays follow independent paths in the spacetime. Our goal in this paper is to present a new, massively parallel algorithm that exploits the recent advances in state-of-the-art graphics processing unit (GPU) platforms designed specifically to handle a large number of parallel threads for ray tracing in general computer visualization (see Figure 1).

Our algorithm is based on the ray-tracing approach of Psaltis & Johannsen (2012) and Bauböck et al. (2012), employs nVidia's proprietary Compute Unified Device Architecture (CUDA) framework, and is implemented in CUDA C/C++. We briefly describe the implementation and list the benchmark results in the next section. As an application, we take advantage of the speed of the code and compute the shadows of black holes of different spins at different inclinations in Section 4. Finally, we discuss future applications of the code such as ray tracing on the fly with general relativistic magnetohydrodynamic (MHD) models of accretion flows in Section 5.

2. IMPLEMENTATION, NUMERICAL SCHEME, AND FEATURES

GPUs were originally developed to handle computationally intensive graphics applications. They provide hardware accelerated rendering in computer graphics, computer-aided design, video games, etc. Indeed, modern GPUs are

optimized specifically with ray tracing in mind (albeit for what we would call flat Euclidean spaces). However, they have recently found extensive use in scientific computing, known as General-Purpose (computing on) Graphics Processing Units (GPGPU, see http://gpgpu.org), as they provide a low-cost, massively parallel platform for computations that do not have large memory needs. These two attributes make GPU technology optimal for the solution of ray tracing in curved spacetimes.

GPUs achieve their high performance by adopting the *stream processing* paradigm, which is one kind of single instruction, multiple data architectures. There are hundreds[5] of *stream processors* on a single chip. These stream processors are designed to perform relatively simple computation in parallel. On the other hand, the on-chip support of caching (fast memory and their automatic management) and branching (conditional code execution, i.e., if-else statements) is primitive.[6] The developers are responsible for ensuring efficient memory access.

This architecture allows most of the transistors to be devoted to performance arithmetic and yields an impressive peak performance. In addition, GPUs hide memory latency by fast switching between computing threads—developers are encouraged to oversubscribe the physical stream processors in order to keep the GPU busy. This is a very different design compared to general purpose multicore central processing units (CPUs), which uses multiple instruction, multiple data architecture and have intelligent cache management and branch predictor to maximize the performance.

Although Open Computing Language (OpenCL) is the industrial open standard of GPGPU programming, we choose CUDA C/C++ to implement this publicly available version of `GRay` because of the availability of good textbooks (e.g., Kirk & Hwu 2010; Sanders & Kandrot 2010) and the easier learning curve. In CUDA terminology, the *host* (i.e., the CPU and the main memory) sends a parallel task to the *device* (i.e., the GPU and the graphics card memory) by launching a computing *kernel*. The kernel runs concurrently on the device involving many lightweight computing *threads*. Because of hardware limitation, threads are organized in *blocks* (of threads) and *grids* (of blocks). Threads within a block can communicate with each other by using a small amount of fast, on-chip *shared memory*; while threads across different blocks can only communicate by accessing a slow, on-card *global memory*.

Because geodesics do not interact with each other, in `GRay`, we simply put each geodesic into a CUDA thread. The states of the photons are stored as an array of structure, which, unfortunately, is not optimal for the GPU to access. In order to maximize the bandwidth, we employ an *in-block data transpose* by using the shared memory.[7] We fix the block size, i.e., the number of threads within a block, $n_{\rm block}$, to 64, which is larger than the number of physical stream processors in a multiprocessor. This oversubscription keeps the GPU busy by allowing a stream processor to work on a thread while waiting for the data for another thread to arrive.[8] The grid size, i.e., the number of blocks within a grid, $n_{\rm grid}$, is computed by the idiomatic formula (see, e.g., Kirk & Hwu 2010; Sanders & Kandrot 2010, or sample codes provided by the CUDA software development kit):

$$n_{\rm grid} = \lfloor (n-1)/n_{\rm block} \rfloor + 1, \tag{1}$$

where n is the total number of photons and $\lfloor \cdot \rfloor$ is the floor function. The above formula ensures that $n_{\rm grid} n_{\rm block} \geqslant n$ so there are enough threads to integrate all the photons.

We employ a standard fourth-order Runge–Kutta scheme presented in Psaltis & Johannsen (2012) to integrate Equations (9)–(12). To avoid the coordinate singularity of the Kerr metric at the event horizon $r_{\rm bh} \equiv 1 + \sqrt{1-a^2}$, we set the step size as

$$\Delta\lambda' \equiv \min\left(\frac{\Delta}{|d\ln r/d\lambda'| + |d\theta/d\lambda'| + |d\phi/d\lambda'|}, \frac{r - r_{\rm bh}}{2|dr/d\lambda'|} \right), \tag{2}$$

and stop integrating the photon trajectory at $r_{\rm bh} + \delta$ to avoid it crossing the horizon at $r_{\rm bh}$. Both $\Delta \sim 1/32$ and $\delta \sim 10^{-6}$ are user-provided parameters. In addition, we use the remapping

$$\theta, \phi, k_\theta \mapsto \begin{cases} 2\pi - \theta, \phi + \pi, -k_\theta & \text{if } \theta > \pi \\ -\theta, \phi - \pi, -k_\theta & \text{if } \theta < 0 \end{cases} \tag{3}$$

to enforce θ to stay in the domain $[0, \pi]$.

The scheme described above can accurately integrate almost all geodesics. However, it breaks down for some of the geodesics that pass through the poles at $\theta = 0$ or π. To illustrate how the scheme breaks down, we choose the special initial conditions $r_0 \cos\theta_0 = 1000M$, $r_0 \sin\theta_0 = 4.833605M$, and $\phi_0 = 0$ for which the photon trajectory passes both the south and north poles of a spin 0.99 black hole.[9] In each panel of Figure 2, we plot the result of tracing the above ray with blue dotted lines.

In the left panel of Figure 2, the gray circle marks the location of the event horizon for the spin 0.99 black hole. The vertical black line is the pole. The green dashed and the red solid lines are the numerical trajectories of the photons with the same initial conditions but with different treatments of the coordinate singularity at the pole, as we will describe below. All three trajectories go around the south pole without any apparent problem and wind back to the north pole. While the red and green trajectories go through the north pole, circulate around the black hole a couple times, and eventually hit the event horizon, the blue trajectory is kicked back to infinity due to a numerical error.

The central panel of Figure 2 is a 100× magnification of the region where the trajectories intersect with the north pole. It shows that the blue trajectory fails to step correctly across the pole. To pinpoint this numerical difficulty, we overplot all the Runge–Kutta sub- and full-steps by open and filled circles, respectively. The two overlapping open blue circles land very close to the pole.

The right panel offers a further 1000× magnification of the same region. It is now clear that the two nearly overlapping open

[5] For example, there are 16 multiprocessors on nVidia Tesla M2090. Each multiprocessor is made up by 32 cores. Hence, there are a total of 512 stream processors on a single GPU.

[6] On the current generation of GPUs, all executed branches within a CUDA block are run in series. This primitive branching ability is another reason that we prefer simple numerical integration over semi-analytical methods in a GPU-based code. Semi-analytical methods usually require branching between computationally intensive functions. Unless preconditioning (e.g., sorting according to branching criterion) is applied, the effective number of operations is summed over all branchings.

[7] `GRay` integrates each geodesic for many steps in a single data load. The in-block transpose, therefore, only improves `GRay`'s performance in the interactive mode, where we limit the number of steps to trade for response time.

[8] Because ray tracing in the Kerr spacetime is computationally intensive, this oversubscription does not play a crucial role in `GRay`'s performance.

[9] Because of cylindrical symmetry, the value of ϕ_0 is not important in this setup. Indeed, for $\phi_0 = 0°, 1°, 2°, \ldots, 359°$, the same pole problem is always encountered.

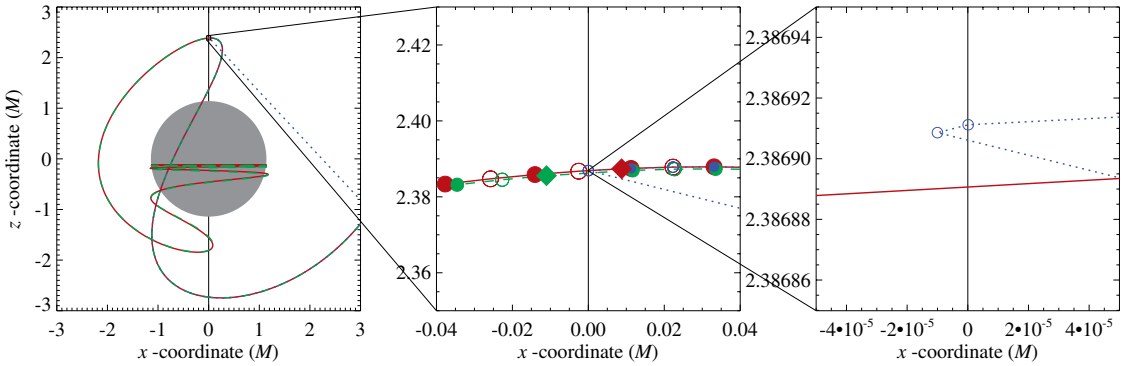

Figure 2. Numerical difficulties occur when the substeps of a fourth-order Runge–Kutta update are evaluated very close to, or on the two different sides, of the poles. In the left panel, the gray circle marks the location of the event horizon for a spin 0.99 black hole. The vertical black line is the coordinate pole. The red solid, green dashed, and blue dotted lines are numerical trajectories of the photons with the same initial conditions (see the text) but with different treatments of the coordinate singularity at the pole. All three trajectories go around the south pole without any apparent problems and wind back to the north pole. However, while the red and green trajectories go through the north pole and eventually hit the event horizon, the blue trajectory is kicked back to infinity. The central panel zooms in by a factor of 100 to show that the blue trajectory fails to step across the pole. To understand this numerical problem, we overplot all the sub- and full-steps by open and filled circles, respectively. The two overlapping open blue circles are evaluated very close to the pole. The right panel zooms further in by another factor of 1000. It is now clear that the two open blue circles are located on different sides of the pole. The low-order truncation errors in the fourth-order Runge–Kutta scheme, instead of canceling, are enhanced. The green trajectory avoids this numerical problem by falling back to a first-order forward Euler scheme, marked by the green diamond in the central panel, whenever the geodesic moves across the pole. This first-order treatment has a larger truncation error because of the first-order stepping—this is visible in the central panel; and it may fail if a full step (filled circle) gets too close to the pole. To reduce the truncation error and make the integrator more robust, the red trajectory uses the forward Euler scheme with a smaller time step, which is marked by the red diamond in the central panel. The subsequent steps are all shifted to avoid the pole. This final treatment is what we employ in the production scheme.

(A color version of this figure is available in the online journal.)

blue circles actually sit on opposite sides of the pole. This is a problem for the fourth-order Runge–Kutta scheme, in which the solution is assumed to be smooth and can be Taylor expanded. In this scheme, the low-order truncation errors are normally canceled by a clever combination of the substeps. Evaluating the geodesic equation in the different substeps on the two sides of the pole, however, introduces an inconsistency in the scheme and enhances the low-order truncation errors.

The green trajectory in Figure 2 shows the result of an improved scheme, which solves the inconsistency by falling back to a first-order forward Euler step whenever a geodesic moves across the pole. The low-order step is marked by the green diamond in the central panel. This treatment mends the numerical difficulty and allows the photon to pass through the pole. Unfortunately, the low-order stepping results in a larger truncation error in the numerical solution. The small but visible offset between the green trajectory and the other two trajectories in the central panel is indeed caused by the low-order step at the south pole. Even worse, this treatment may fail when a full step (i.e., the filled circles) gets too close to the pole.

To reduce the truncation error of the low-order step and make the integrator more robust, in the production scheme of `GRay`, we follow Psaltis & Johannsen (2012) to monitor the quantity

$$\xi \equiv \left[g_{rr}\left(\frac{dr}{d\lambda'}\right)^2 + g_{\phi\phi}\left(\frac{d\phi}{d\lambda'}\right)^2 + g_{\theta\theta}\left(\frac{d\theta}{d\lambda'}\right)^2 + 2g_{t\phi}\left(\frac{dt}{d\lambda'}\right)\left(\frac{d\phi}{d\lambda'}\right)\right] \Bigg/ \left[g_{tt}\left(\frac{dt}{d\lambda'}\right)^2 \right], \quad (4)$$

which should always remain equal to -1. If $|\xi + 1| > \epsilon$, for some small parameter $\epsilon \sim 10^{-3}$ in the numerical scheme, we *re-integrate* the inaccurate step by falling back to the first-order forward Euler scheme with a *smaller* time step $\Delta\lambda'/9$. This step size is chosen so that (1) the absolute numerical error of the solution does not increase substantially because of this single low-order step and (2) the pole is not encountered even if the Euler scheme is continuously applied. This first-order step is marked by the red diamond in the central panel of Figure 2. The subsequent steps, as shown in the figure by the red circles, are all shifted toward the left and skip the pole.

We find this final pole treatment extremely robust and use it for all our production calculations. For the 3×10^8 trajectories that we will integrate in Section 4, none of them fails at the pole as long as we fix $\Delta = 10^{-6}$. The rest of the implementation of the algorithm, the initial conditions, and the setup of the rays on the image plane proceed as in Psaltis & Johannsen (2012).

In addition to performing the computation of ray tracing, `GRay` takes advantage of the programmable graphics pipeline to perform real time data visualization. It can be compiled in an interactive mode by enabling OpenGL. The OpenGL frame buffer is allocated on the graphics card, which is then mapped to CUDA for ray tracing. This technique is called *CUDA-OpenGL interoperability*—there is no need to transfer the data between the host and the device. Because the data reside on the graphics card and are accessible to OpenGL, we use the OpenGL Shading Language (see http://www.opengl.org) to perform coordinate transformation and sprite drawing. A screen shot of this built-in real time visualization is provided in Figure 1.

3. BENCHMARKS

The theoretical peak performance of a high-end GPU is always about an order of magnitude faster than the peak performance of a high-end multicore CPU (Kirk & Hwu 2010; Sanders & Kandrot 2010). However, because of the fundamental difference in the hardware design, their real world performances depend on the nature of the problem and the implementation of the algorithms. In order to compare different aspects of the implementation of the ray-tracing algorithm, we perform two different benchmarks on three codes in this section.

1. `Geokerr` is a well-established, publicly available code written in `FORTRAN`. The code uses a semi-analytical approach to solve for null geodesics in Kerr spacetimes, which leads to accurate solutions even with arbitrarily large

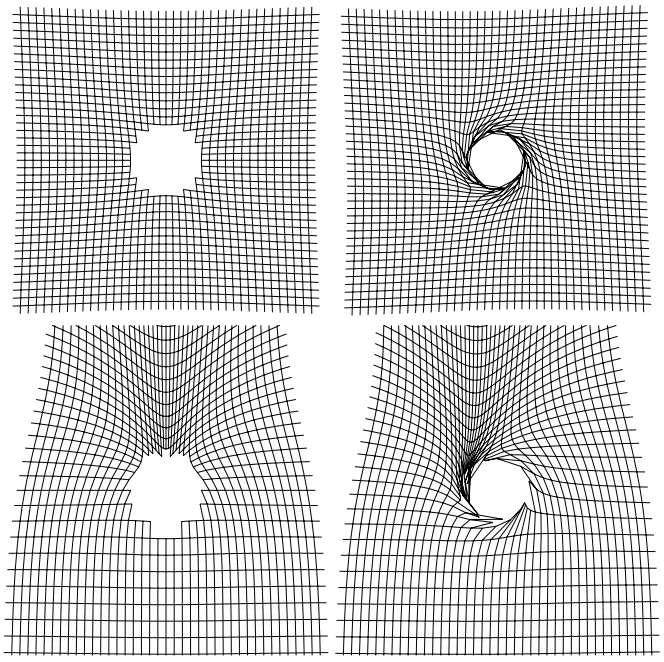

Figure 3. Projections of a uniform Cartesian grid in the image plane to the equatorial plane of spin 0 (left column) and 0.95 (right column) black holes. The images in the top and bottom rows have inclination angles 0° and 60°, respectively. They are plotted in a way to match Figure 2 of Schnittman & Bertschinger (2004) and Figure 3 of Dexter & Agol (2009); i.e., the horizontal and vertical axes correspond to the $-\beta_0$- and $-\alpha_0$-directions. The configuration in the lower right panel with parameters $a = 0.95$ and $i = 60°$ is the representative ray-tracing problem we use in Figure 4 for the comparative benchmarks.

time steps.[10] The details of the algorithm are documented in Dexter & Agol (2009).

2. `Ray` is an algorithm that uses a standard fourth-order Runge–Kutta scheme to integrate the geodesic equations in spacetimes with arbitrary quadrupole moments. It is written in C and runs efficiently on CPUs. The code has been used to test the no-hair theorem and generate profiles and spectra from spinning neutron stars (Psaltis & Johannsen 2012; Bauböck et al. 2012).
3. `GRay`, the open source GPU code we describe in this paper, is based on `Ray`'s algorithm. It is written in CUDA C/C++ and runs efficiently on most nVidia GPUs. The source code is published under the GNU General Public License Version 3 and is available at https://github.com/chanchikwan/gray.

For the first benchmark, we compute the projection of a uniform Cartesian grid in the image plane onto the equatorial plane of a spinning black hole. This problem was carried out in Schnittman & Bertschinger (2004) and then used as a test case in Dexter & Agol (2009). We reproduce the published results in Figure 3, using `GRay` and initializing the image plane at $r = 1000M$. The left and right columns show the projections for two black holes with spins 0 and 0.95, respectively; in each case, the top and bottom rows represent observer inclinations of 0° and 60°, respectively.

The case shown in the lower right panel of Figure 3 with parameters $a = 0.95$ and $i = 60°$ is a representative problem, which we will use as a benchmark. We use the three algorithms `Geokerr`, `Ray`, and `GRay` and calculate the projection using a grid of n geodesics for each method. In Figure 4, we plot the

[10] Note, however, that substeps are needed if there are turning points in the null geodesics.

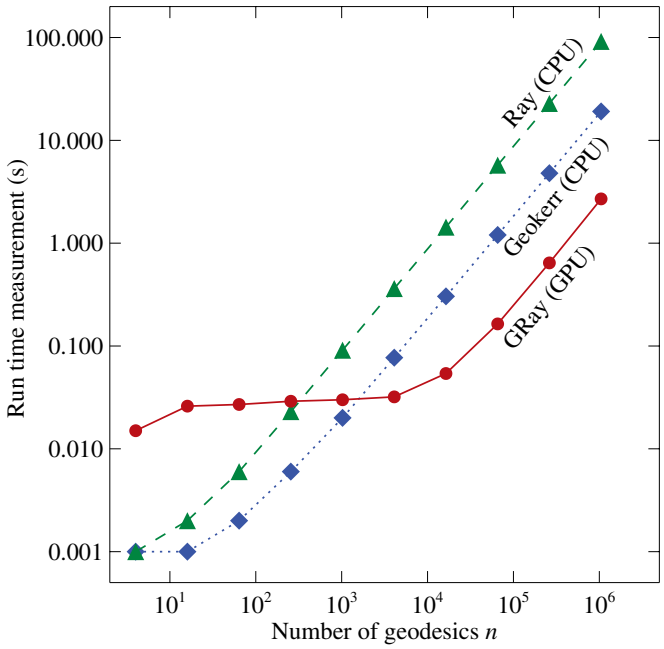

Figure 4. Results of the grid projection benchmark for the configuration shown in the lower right panel of Figure 3. The run times of three different algorithms, `Geokerr` (blue diamonds), `Ray` (green triangles), and `GRay` (red circles) in double precision, are plotted against the number of geodesics traced for each image. The asymptotic linear dependence seen for all three algorithms demonstrates explicitly that the ray-tracing problem is highly parallelizable. For a small number of geodesics, the performance of `GRay` flattens to a constant value (approximately 20 ms for the configuration used) because of the time required for launching the CUDA kernel.

(A color version of this figure is available in the online journal.)

run time on a single processor of each calculation as a function of the number of geodesics traced.

We can draw a few interesting conclusions from this simple benchmark. The performance of all algorithms scales linearly for almost all problems, signifying the fact that ray tracing is a highly parallelizable problem. For a small number of rays, the performance of `GRay` flattens at about 20 ms for the configuration used, because of the time required for launching the CUDA Kernel. This is independent of the number of geodesics but, of course, depends on the specific hardware, drivers, and operating system used. For a MacBook Pro running OS X, this time is of the order of a few tens of milliseconds, while the launching time may be as large as 0.5 s for some Linux configurations.

For calculations with a large number of geodesics, which is the regime that motivated our work, `GRay` is faster than both `Geokerr` and `Ray` by one to two orders of magnitude. It is important to emphasize that the performance of `GRay` exceeds that of the other algorithms even in this benchmark that is designed in a way that favors the semi-analytical approach of `Geokerr`. This is true because we are only interested in the intersection of the ray with the equatorial plane, which `Geokerr` can achieve with a very small number of steps per ray. In more general radiative transfer problems, however, we have to divide each ray in small steps in all methods in order to integrate accurately the radiative transfer equation through black hole accretion flows. This requirement puts the Runge–Kutta integrators at a larger advantage compared to semi-analytic approaches.

In order to assess the performance of `GRay` in this second situation, we setup a benchmark to measure the average time

Table 1
Benchmark Results of GRay in Comparison to Other General Relativistic Ray Tracing Codes

Processor	Geokerr[a]	Ray	GRay
nVidia Tesla M2090[b]	...	...	1.15
nVidia GeForce GT 650M[b]	...	...	3.27
nVidia Tesla M2090	...	...	7.15
nVidia GeForce GT 650M	...	...	71.87
Intel Core i7-3720QM 2.60GHz[c]	23000	356.67	...
Intel Xeon E5520 2.27GHz[c]	43800	692.68	...

Notes. We focus only at the performance of the geodesic integrators. The numbers listed in the above table have unit of *nanosecond per time step per photon*. Hence, the smaller number indicates higher performance.
[a] Geokerr computes the geodesic semi-analytically and hence can take arbitrary long time steps unless there is a turning point in the geodesic.
[b] Single-precision floating arithmetic is used.
[c] Both Geokerr and Ray are serial codes. Hence, only one CPU core is used in these measurements.

that the integrators require to take a single step in the integration of a photon path. We list the results of this benchmark for the three algorithms in Table 1, where the numbers have unit of *nanoseconds per time step per photon*, such that the smaller number indicates higher performance. In this benchmark, the benefit of the GPU integrator becomes clearly visible as GRay is 50 times faster than Ray and more than a factor of 1000 faster than Geokerr.

4. PROPERTIES OF PHOTON RINGS AROUND KERR BLACK HOLES

Being a massively parallel algorithm, GRay is an ideal tool to study black hole images that involve integrating billions of photon trajectories. In general, the details of black hole images depend on the time-dependent properties of the turbulent accretion flows (see also Section 5 for a detailed discussion). In all cases, however, for optically thin accretion flows such as the one expected around Sgr A* at millimeter wavelengths, the projection of the circular photon orbit produces a bright ring on the image plane that stands out against the background (see Luminet 1979; Beckwith & Done 2005; Johannsen & Psaltis 2010). As pointed out by Johannsen & Psaltis (2010), the shape of this so-called *photon ring* that surrounds the black hole shadow is a general relativistic effect and is insensitive to the complicated astrophysics of the accretion flows. Careful matching of the theoretical predictions of the photon ring with observations, therefore, provides an unmistakable way to measure the black hole mass and even to test the no-hair theorem (Johannsen & Psaltis 2010; Johannsen et al. 2012).

We performed a systematic calculation of the photon rings around Kerr black holes of different spins a and observer inclinations i. We choose 16 values of spin according to the relation

$$a_j = 1 - 10^{-j/5}, \text{ where } j = 0, 1, \ldots, 15 \tag{5}$$

so that $1 - a$ is evenly spaced in log scale, and 19 values of inclination $i = 0, 5, 10, \ldots, 90$. For each configuration, we set up the image plane at $r = 1000M$ and define its center at the intersection of this plane with a radial line emerging out of the black hole. We define $(\mathcal{R}, \vartheta)$ to be the local polar coordinate of the image plane. We set up a grid of 6000×181 rays in the polar domain $(1.5, 7.5) \times [0, \pi]$ and integrate them toward the black hole. Hence, there are $16 \times 19 \times 6000 \times 181 \approx 3 \times 10^8$ geodesics in this parameter study.[11]

We plot the outlines of the photon rings in Figures 5 and 6. As discussed in Johannsen & Psaltis (2010), we find that the size of the photon ring depends very weakly on the spin of the black hole and the inclination of the observer. Moreover, the ring retains a highly circular shape at even high spins, as significant asymmetries appear only at $a \gtrsim 0.99$. In Johannsen & Psaltis (2010), we attributed this to the cancellation of the ellipsoidal geometry of the Kerr spacetime by the frame-dragging effects on the propagation of photons, which appears to be exact at the quadrupole order.

In order to quantify the magnitude of the effects discussed above, we follow Johannsen & Psaltis (2010) to define the horizontal displacement of the ring from the geometric center of the spacetime as

$$D \equiv \frac{|\alpha_{0,\min} + \alpha_{0,\max}|}{2}, \tag{6}$$

the average radius of the ring as

$$\langle R \rangle \equiv \frac{1}{2\pi} \int_0^{2\pi} R d\vartheta, \tag{7}$$

where $R \equiv [(\alpha_0 - D)^2 + \beta_0^2]^{1/2}$ and $\theta \equiv \tan^{-1}(\beta_0/\alpha_0)$, and the asymmetry parameter as

$$A \equiv 2 \left[\frac{1}{2\pi} \int_0^{2\pi} (R - \langle R \rangle)^2 \, d\vartheta \right]^{1/2}. \tag{8}$$

In the above equations, the coordinates α_0 and β_0 are understood to be measured on a two-dimensional Cartesian coordinate system on the image plane, i.e., they are related to $\mathcal{R}$ and ϑ by the coordinate transformation:

$$\alpha_0 = \mathcal{R} \cos\vartheta, \tag{9}$$

$$\beta_0 = \mathcal{R} \sin\vartheta. \tag{10}$$

In Figure 7, we plot these ring quantities as functions of the observer inclination i at different black hole spins a.

In order to facilitate the comparison of theoretical models to upcoming observations of black hole shadows with the Event Horizon Telescope, we have obtained simple analytic fits to the dependence of the average radius and asymmetry of the photon ring for black holes with different spins and observer inclinations. In particular, we find

$$\langle R \rangle \simeq R_0 + R_1 \cos(2.14i - 22.^\circ 9) \tag{11}$$

with

$$R_0 = (5.2 - 0.209a + 0.445a^2 - 0.567a^3)M$$
$$R_1 = \left[0.24 - \frac{3.3}{(a - 0.9017)^2 + 0.059} \right] \times 10^{-3} M \tag{12}$$

and

$$A \simeq A_0 \sin^n i \ , \tag{13}$$

[11] We use git to manage the source code of GRay. For reproducibility, the setup of this parameter study is available in the source repository with commit id 4e20d4c0. See also the commit message for running the study.

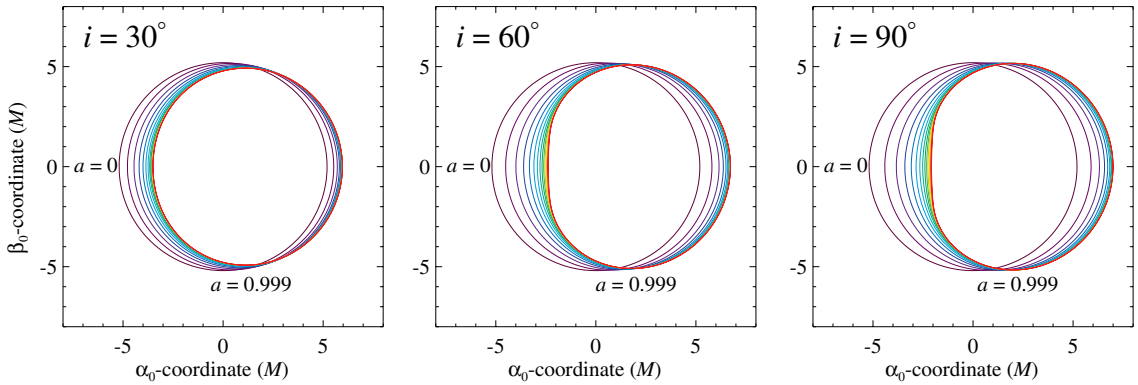

Figure 5. Photon rings around Kerr black holes with different spins a and observer inclinations i. The left, central, and right panels show the photon rings for $i = 30°$, 60°, and 90°, respectively. In each panel, different colors represent different spins—from black being $a = 0$ to red being $a = 0.999$. For each inclination, the size of the photon ring depends very weakly on the black hole spin. Moreover, the photon ring retains its nearly circular shape even at high black hole spins; a significant distortion appears only for $a \gtrsim 0.99$ and at large inclination angles.

(A color version of this figure is available in the online journal.)

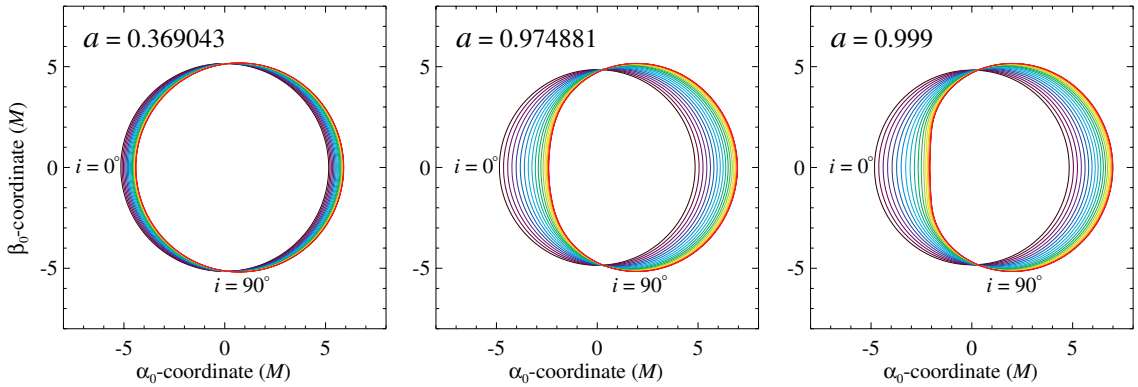

Figure 6. Photon rings around Kerr black holes with different spins a and observer inclinations i. The left, central, and right panels plot the photon rings for $a = 0.369043$, 0.974881, and 0.999, respectively. In each panel, different colors represent different inclinations—going from black for $i = 0°$, to blue for $i = 5°$, 10°, ..., to red for $i = 90°$. The photon rings become asymmetric only for $a \gtrsim 0.99$ and at large inclination angles.

(A color version of this figure is available in the online journal.)

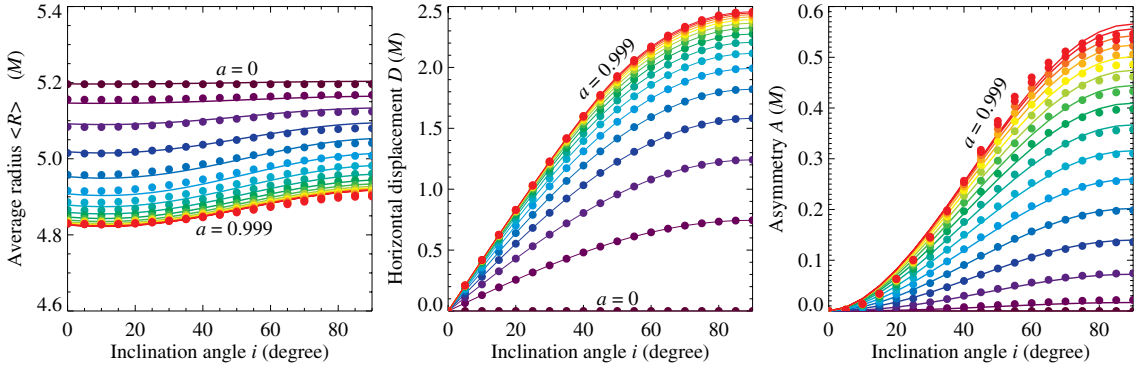

Figure 7. Photon-ring properties for Kerr black holes with different spins a and different inclinations i. The left, central, and right panels show the average radius $\langle R \rangle$, the horizontal displacement D, and the asymmetry parameter A of the rings, respectively. In each panel, the horizontal axis is the inclination i and different colors represent different spins—from black for $a = 0$ to red for $a = 0.999$. In the leftmost and rightmost panels, the solid curves show the analytic fits discussed in the text.

(A color version of this figure is available in the online journal.)

with

$$A_0 = (0.332a^3 + 0.176a^{21.7} + 0.0756a^{195})M$$
$$n = 1.55(1-a)^{-0.022} + 1.3(1-a)^{0.98}. \quad (14)$$

In all relations, the arguments of the trigonometric functions are in degrees. The above empirical relations are shown as solid curves in the leftmost and rightmost panels of Figure 7.

In Figure 8, we provide a different representation of the above results, by plotting contours of constant average radius $\langle R \rangle$ and asymmetry parameter A on the parameter space of black hole spin a and observer inclination i. The Event Horizon Telescope (Doeleman et al. 2009) aims to perform imaging observations of the inner accretion flows around the black holes in the center of the Milky Way and of M87, in order to measure these two parameters of the black hole shadows. (The displacement D cannot be readily measured, since there is very little indication of the geometric center of the spacetime that can be obtained from the images.) The spin of the black hole and the inclination of the observer can be independently determined based on where

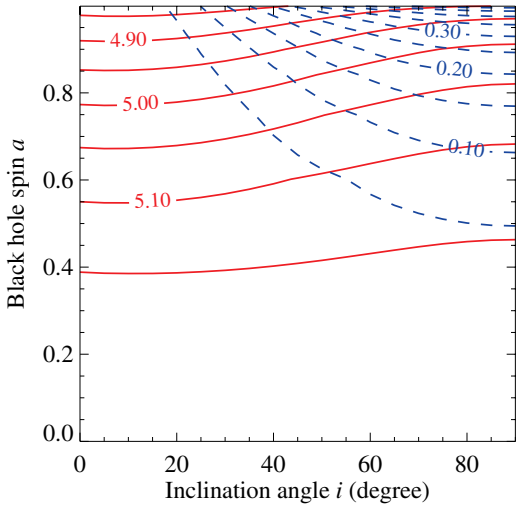

Figure 8. Contours of constant photon-ring properties around Kerr black holes, which can be inferred with imaging observations of their accretion flows. The contours of constant average radius of the ring, $\langle R \rangle$, and asymmetry parameter, A, are plotted as the solid red and dashed blue curves, respectively. If the two contour lines correspond to an observed photon-ring radius and asymmetry for a black hole cross, then both the spin of the black hole and the inclination of the observer can be independently inferred. On the other hand, if the two contour lines do not intersect, this will indicate a violation of the no-hair theorem (Johannsen & Psaltis 2010).

(A color version of this figure is available in the online journal.)

the two contour lines of the observed radius and the asymmetry for the black hole cross. If the two contour lines do not cross, then the no-hair theorem is violated (Johannsen & Psaltis 2010).

5. DISCUSSIONS

In this paper, we presented our implementation of the massively parallel ray-tracing algorithm `GRay` for GPU architecture. We demonstrated that its performance is about two orders of magnitude faster than equivalent CPU ray-tracing codes. Running this algorithm on an nVidia Tesla M2090 card, we are able to compute a 1024×1024 pixel image in about a few seconds (see Figure 4). At the same time, we can achieve time steps per photon as small as 1 ns on the same GPU card (see Table 1). Bearing in mind that communication is almost always slower than computation in high-performance computing (e.g., the host-device bandwidth, through PCI Express, is at least an order of magnitude faster than the hard disk bandwidth, through SATA) also leads us to conclude that using GPUs to perform the computation of geodesics when needed in an algorithm is a more efficient approach to solving this problem compared to tabulating precomputed results in a database.

Our initial goal is to use `GRay` to make significant advances in modeling and interpreting observational data. Nevertheless, `GRay` will also be extremely useful in performing three-dimensional, MHD calculations in full general relativity aiming to achieve ab initio simulations of MHD processes in the vicinity of black hole horizons (see, e.g., De Villiers & Hawley 2003; Gammie et al. 2003; Mizuno et al. 2006; Giacomazzo & Rezzolla 2007; Del Zanna et al. 2007; Cerdá-Durán et al. 2008; Zink 2011). Besides being very important for improving our understanding of accretion flows, MHD simulations have been instrumental in interpreting observations of Sgr A* and its unusual flares (see, e.g., Chan et al. 2009; Mościbrodzka et al. 2009; Dodds-Eden et al. 2010; Dolence et al. 2012).

Comparing the results of numerical simulations to observations requires, at the very least, using the calculated time-dependent thermodynamic and hydrodynamic properties of the MHD flows to predict light curves, spectra, and images. At the same time, the propagation of radiation within the MHD flow contributes to its heating and cooling. In addition, radiation forces determine even the dynamics of near Eddington accretion flows. Calculating the propagation of radiation within the accretion flow and to an observer at infinity in a time-dependent manner is very time consuming. It has been taken into account only in limited simulations and under various simplifying assumptions (see, e.g., De Villiers 2008). In fact, only a handful of numerical algorithms have been used to date for calculations of observed quantities post facto, based on *snapshots* of MHD simulations (Dexter & Agol 2009; Dolence et al. 2009). This "fast light" approximation breaks down close to the black hole because of the speed of the plasma there is comparable to the speed of light. In order to overcome the storage requirement of frequent data dump, `GRay` may be integrated into a general relativistic MHD code to perform ray tracing on the fly.

When the radiative transfer equation needs to be solved along the photon rays, heavy branching may be required if the relevant absorption and emission coefficients are calculated on the fly using the primitive MHD variables of the simulation. This will introduce a heavy burden on the algorithm and significantly reduce the efficiency of using a GPU architecture. On the other hand, if instead of repeatedly calculating the absorption and emission coefficients, one simply reads them off a precomputed table (as is done in stellar evolution codes), then the efficiency of this method is determined by the communication bandwidth between the memory and the GPU core.

The current state-of-the-art GRMHD simulations of accretion disks have resolution of order $256 \times 128 \times 64$, which take about 100 MB of storage for each snapshot. Taking Tesla M2090, the GPU we used for our production runs, as an example, the maximum memory bandwidth is 177 GB s^{-1} and the peak performance is 1332 single-precision GFLOPS. On the one hand, the GPU can, in principle, read in the quantities for 1770 snapshots per second at its maximum bandwidth. On the other hand, there are about 396 floating-point operations per full time step in the Kerr module. The GPU can perform at most 3.4 billion time steps per second—about 3482 steps per second for a 1024×1024 image (our benchmark shows 1/4 of the peak performance, which is very efficient). Therefore, GPU ray tracing of GRMHD simulations is a well-balanced problem between memory access and computation even taking into account data access. Therefore, bandwidth limitations will not adversely affect the integration of our GPU ray-tracing code with hydrodynamic or MHD algorithms.

This work was supported in part by the NSF grant AST-1108753, NSF CAREER award AST-0746549, and Chandra Theory grant TM2-13002X. F.Ö. gratefully acknowledges support from the Radcliffe Institute for Advanced Study at Harvard University.

REFERENCES

Arzoumanian, Z., Bogdanov, S., Cordes, J., et al. 2009, in The Astronomy and Astrophysics Decadal Survey, Vol. 2010, New Worlds, New Horizons in Astronomy and Astrophysics, ed. C. Gruber (Washington, DC: The National Academies Press), 6
Bauböck, M., Psaltis, D., Özel, F., & Johannsen, T. 2012, ApJ, 753, 175
Beckwith, K., & Done, C. 2005, MNRAS, 359, 1217
Bogdanov, S., Rybicki, G. B., & Grindlay, J. E. 2007, ApJ, 670, 668

Braje, T. M., & Romani, R. W. 2002, ApJ, 580, 1043
Brenneman, L. W., & Reynolds, C. S. 2006, ApJ, 652, 1028
Broderick, A. E. 2006, MNRAS, 366, L10
Broderick, A. E., Fish, V. L., Doeleman, S. S., & Loeb, A. 2009, ApJ, 697, 45
Cadeau, C., Morsink, S. M., Leahy, D., & Campbell, S. S. 2007, ApJ, 654, 458
Cerdá-Durán, P., Font, J. A., Antón, L., & Müller, E. 2008, A&A, 492, 937
Chan, C.-k., Liu, S., Fryer, C. L., et al. 2009, ApJ, 701, 521
Cunningham, C. T. 1975, PhRvD, 12, 323
Del Zanna, L., Zanotti, O., Bucciantini, N., & Londrillo, P. 2007, A&A, 473, 11
De Villiers, J.-P. 2008, arXiv:0802.0848
De Villiers, J.-P., & Hawley, J. F. 2003, ApJ, 589, 458
Dexter, J., & Agol, E. 2009, ApJ, 696, 1616
Dexter, J., Agol, E., & Fragile, P. C. 2009, ApJL, 703, L142
Dodds-Eden, K., Sharma, P., Quataert, E., et al. 2010, ApJ, 725, 450
Doeleman, S., Agol, E., Backer, D., et al. 2009, in The Astronomy and Astrophysics Decadal Survey, Vol. 2010, New Worlds, New Horizons in Astronomy and Astrophysics, ed. C. Gruber (Washington, DC: The National Academies Press), 68
Dolence, J. C., Gammie, C. F., Mościbrodzka, M., & Leung, P. K. 2009, ApJS, 184, 387
Dolence, J. C., Gammie, C. F., Shiokawa, H., & Noble, S. C. 2012, ApJL, 746, L10
Dovčiak, M., Karas, V., Martocchia, A., Matt, G., & Yaqoob, T. 2004, in RAGtime 4/5: Workshops on Black Holes and Neutron Stars, ed. S. Hledík & Z. Stuchlík (Czech Republic: Silesian University in Opava), 33
Fabian, A. C., Zoghbi, A., Ross, R. R., et al. 2009, Natur, 459, 540
Feroci, M., den Herder, J. W., Bozzo, E., et al. 2012, Proc. SPIE, 8443, 84432D
Gammie, C. F., McKinney, J. C., & Tóth, G. 2003, ApJ, 589, 444
Giacomazzo, B., & Rezzolla, L. 2007, CQGra, 24, 235
Johannsen, T., & Psaltis, D. 2010, ApJ, 718, 446
Johannsen, T., Psaltis, D., Gillessen, S., et al. 2012, ApJ, 758, 30
Kirk, D. B., & Hwu, W.-m. W. 2010, Programming Massively Parallel Processors: A Hands-on Approach (1st ed.; San Francisco, CA: Morgan Kaufmann Publishers Inc.)
Laor, A. 1991, ApJ, 376, 90
Leahy, D. A., Morsink, S. M., & Cadeau, C. 2008, ApJ, 672, 1119
Luminet, J.-P. 1979, A&A, 75, 228
Miller, J. M. 2007, ARA&A, 45, 441
Miller, M. C., & Lamb, F. K. 1998, ApJL, 499, L37
Mizuno, Y., Nishikawa, K.-I., Koide, S., Hardee, P., & Fishman, G. J. 2006, arXiv:astro-ph/0609004
Mościbrodzka, M., Gammie, C. F., Dolence, J. C., Shiokawa, H., & Leung, P. K. 2009, ApJ, 706, 497
Muno, M. P., Özel, F., & Chakrabarty, D. 2002, ApJ, 581, 550
Pechenick, K. R., Ftaclas, C., & Cohen, J. M. 1983, ApJ, 274, 846
Psaltis, D., & Johannsen, T. 2012, ApJ, 745, 1
Sanders, J., & Kandrot, E. 2010, CUDA by Example: An Introduction to General-Purpose GPU Programming (1st ed.; Boston, MA: Addison-Wesley Professional)
Schnittman, J. D., & Bertschinger, E. 2004, ApJ, 606, 1098
Speith, R., Riffert, H., & Ruder, H. 1995, CoPhC, 88, 109
Takahashi, T., Mitsuda, K., Kelley, R., et al. 2012, Proc. SPIE, 8443, 84431Z
Weinberg, N., Miller, M. C., & Lamb, D. Q. 2001, ApJ, 546, 1098
Zink, B. 2011, arXiv:1102.5202