Topics Geometry and Topology – MATH 599
Visualization of Kerr black holes

Renaud Raquépas, Erick Schulz

April 29, 2017

1 Introduction

Black holes are one of the most intriguing aspects of Einstein's theory of general relativity. In popular culture, it has been the subject of many misconceptions. Recently, the Hollywood production *Interstellar* has however delivered to general audiences high quality depictions of black holes based on serious science [JvTFT15].

For this project, we are interested in understanding how physically (relatively) accurate visual representations of rotating black holes are obtained. Concretely, we create in `MATLAB` a ray tracer for a Kerr geometry describing the surroundings of a rotating black hole.

We start by briefly describing the geometry of Kerr space-time. Then, we present a toy model for vision and ray tracing before moving on to numerical implementation.

2 The Kerr metric

The Kerr metric is an axisymmetric solution of Einstein's field equations which describes the geometry of spacetime near a rotating electrically neutral rotating mass [Ker63]. It is a generalization of the Schwarzschild metric, and a special case of the Kerr–Newman metric.

2.1 The metric and Boyer–Lindquist coordinates

The Kerr metric is usually expressed in so-called Boyer–Lindquist coordinates (t, r, θ, ϕ). These are related to Euclidian coordinates through

$$\begin{aligned} t &= t, \\ x &= \sqrt{r^2 + a^2}\sin\theta\cos\phi, \\ y &= \sqrt{r^2 + a^2}\sin\theta\sin\phi, \\ z &= r\cos\theta. \end{aligned}$$

The Kerr metric is the non-degenerate symmetric 2-tensor given, for r large enough, by

$$-\frac{\Delta}{\Sigma}(\mathrm{d}t - a\sin^2\theta\,\mathrm{d}\phi)^2 + \frac{\sin^2\theta}{\Sigma}((r^2+a^2)\,\mathrm{d}\phi - a\,\mathrm{d}t)^2 + \frac{\Sigma}{\Delta}\,\mathrm{d}r^2 + \Sigma\,\mathrm{d}\theta^2, \tag{1}$$

with

$$\begin{aligned} \Delta(r) &:= r^2 - Rr + a^2, \\ \Sigma(r,\theta) &:= r^2 + a^2\cos^2\theta, \end{aligned}$$

where $R = 2M$ and $a = JM^{-1}$ are parameters determined by the mass M and angular momentum J of the black hole. We work in Plank units $G = 1$, $c = 1$. Note that this metric is invariant under the transformation $(R, a; t, r, \theta, \phi) \mapsto (R, -a; -t, r, \theta, \phi)$. Also note that we recover the Schwarzschild metric when $a = 0$ and that this Kerr metric is approximately flat for $r \gg a, R$. Finally, we remark that metric coefficients diverge as $\Delta(r) \to 0$, *i.e.* as $r \to \frac{1}{2}(R \pm \sqrt{R^2 - 4a^2}) = M \pm \sqrt{M^2 - a^2}$.

In the basis $(\partial_t, \partial_r, \partial_\theta, \partial_\phi)$ for tangent spaces, the metric has matrix form

$$g = \begin{pmatrix} -\left(1 - \frac{Rr}{\Sigma(r,\theta)}\right) & 0 & 0 & -\frac{Rra\sin^2(\theta)}{\Sigma(r,\theta)} \\ 0 & \frac{\Sigma(r,\theta)}{\Delta(r)} & 0 & 0 \\ 0 & 0 & \Sigma(r,\theta) & 0 \\ -\frac{Rra\sin^2(\theta)}{\Sigma(r,\theta)} & 0 & 0 & \frac{\sin^2(\theta)\left(\left(a^2+r^2\right)^2 - a^2\Delta(r)\sin^2(\theta)\right)}{\Sigma(r,\theta)} \end{pmatrix}.$$

It will so be useful to have its matrix inverse:

$$g^{-1} = \begin{pmatrix} -\frac{\frac{rR\sin^2(\theta)a^2}{\Sigma(r,\theta)} + a^2 + r^2}{\Delta(r)} & 0 & 0 & -\frac{Rar}{\Sigma(r,\theta)\Delta(r)} \\ 0 & \frac{\Delta(r)}{\Sigma(r,\theta)} & 0 & 0 \\ 0 & 0 & \frac{1}{\Sigma(r,\theta)} & 0 \\ -\frac{Rar}{\Sigma(r,\theta)\Delta(r)} & 0 & 0 & \frac{\Delta(r) - a^2\sin^2(\theta)}{\Sigma(r,\theta)\Delta(r)\sin^2(\theta)} \end{pmatrix}.$$

2.2 Geodesic equations

As will become clear in the next section, visualizing a rotating black hole will necessitate solving — numerically at least — the geodesic equations for the metric connection associated to the Kerr metric. We use a Hamiltonian approach to this problem.

Consider the energy functional

$$\begin{aligned} E(\gamma) &= \frac{1}{2}\int_a^b g(\dot\gamma(s), \dot\gamma(s))\,\mathrm{d}s \\ &= \frac{1}{2}\int_a^b g_{\mu\nu}\dot\gamma^\mu(s)\dot\gamma^\nu(s)\,\mathrm{d}s \end{aligned}$$

for γ a curve parametrized by $s \in [a, b]$. Extremizing this functional using the Euler–Lagrange equation

$$\frac{1}{2}\frac{\partial}{\partial\gamma^\mu} g(\dot\gamma(s), \dot\gamma(s)) = \frac{1}{2}\frac{\mathrm{d}}{\mathrm{d}s}\frac{\partial}{\partial\dot\gamma^\mu} g(\dot\gamma(s), \dot\gamma(s))$$

yields

$$0 = g_{\mu\nu}\frac{\mathrm{d}}{\mathrm{d}s}\dot\gamma^\mu + \frac{1}{2}\left(\frac{\partial g_{\mu\nu}}{\partial\gamma^\lambda} + \frac{\partial g_{\nu\lambda}}{\partial\gamma^\mu} - \frac{\partial g_{\mu\lambda}}{\partial\gamma^\mu}\right)\dot\gamma^\mu\dot\gamma^\lambda,$$

i.e. the (lowered) geodesic equations. This means that geodesic are encapsulated in a least action principle for the Lagrangean $\mathcal{L}(\gamma, \dot\gamma) = \frac{1}{2}g(\dot\gamma, \dot\gamma)$ and hence can be treated using Hamilton's canonical equations

$$\begin{cases} \dot x^\mu &= \dfrac{\partial H(x,p)}{\partial p_\mu} \\ \dot p_\mu &= -\dfrac{\partial H(x,p)}{\partial x^\mu} \end{cases}$$

for the Hamiltonian

$$\begin{aligned} H(x,p) &= -\mathcal{L} + \dot\gamma^\mu\frac{\partial\mathcal{L}}{\partial\dot\gamma^\mu} \\ &= \tfrac{1}{2}g^{\mu\nu}p_\mu p_\nu, \end{aligned}$$

where $x^\mu = \gamma^\mu$ and $p_\mu = g_{\mu\lambda}\dot{\gamma}^\lambda$. In practice, we will solve this system of first order ODEs using a Runge–Kutta-4 method.

We also take advantage of conserved quantities arising from Noether's theorem. These conserved quantities are the angular momentum arising from the Killing vector field ∂_ϕ

$$L = -\frac{Ra}{r}\dot{t} + \big(r^2 + a^2 + \frac{Ra^2}{r}\big)\dot{\phi}$$

and the energy arising from the Killing vector field ∂_t

$$E = \big(1 - \frac{r}{R}\big)\dot{t} + \frac{Ra}{r}\dot{\phi}.$$

2.3 The Runge–Kutta-4 method

The Runge–Kutta-4 method is applicable to ODEs of the form

$$\dot{X}(\lambda) = F(\lambda, x(\lambda))$$

It is based on the fact that the solution X to such a system satisfies the approximation

$$X(\lambda + h) = X(\lambda) + \frac{h}{6}\left(f_1(\lambda, X(\lambda)) + f_2(\lambda, X(\lambda)) + f_3(\lambda, X(\lambda)) + f_4(\lambda, X(\lambda))\right) + O(h^5) \tag{2}$$

where

$$\begin{aligned}
f_1(\lambda, X(\lambda)) &= F(\lambda, x(\lambda)),\\
f_2(\lambda, X(\lambda)) &= F(\lambda + \tfrac{1}{2}h, X(\lambda) + \tfrac{1}{2}hs_1(\lambda, X)),\\
f_3(\lambda, X(\lambda)) &= F(\lambda + \tfrac{1}{2}h, X(\lambda) + \tfrac{1}{2}hs_2(\lambda, X)),\\
f_4(\lambda, X(\lambda)) &= F(\lambda + h, X(\lambda) + hs_3(\lambda, X)).
\end{aligned}$$

We do not prove this well-known approximation, which is obtained through a clever Taylor expansion.

We use this Runge–Kutta to integrate the geodesic equations where

$$X = (x^0, \dots, x^3, p_0, \dots, p_3)$$

and

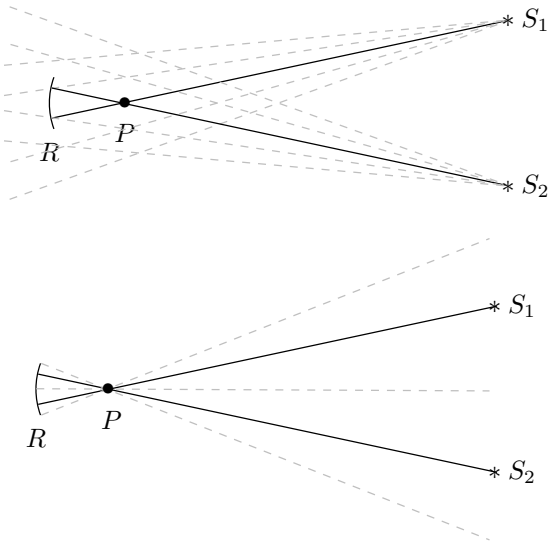
$$F = (\tfrac{\partial H}{\partial p_0}, \dots, \tfrac{\partial H}{\partial p_3}, -\tfrac{\partial H}{\partial x^0}, \dots, -\tfrac{\partial H}{\partial x^3}).$$

We use a step size h proportional to $\Delta(r)$, because some coefficients in the equation blow up as $\Delta(r)^{-1}$ near the horizon.

3 Ray tracer

3.1 Toy model for ray tracing

To understand better how to generate images of Kerr black holes, we pause and take a look at a toy model for vision in flat space-time. Consider an eye (or camera) modelled by a point pupil P and a retina (or screen) R, and light sources, say a star S_i. The sources S_i emits light in all directions, in straight line, and (only) the rays that pass through the pupil P are then recorded on the retina R, which we think of as an indexed array of pixels. The color of the source is recorded at the corresponding pixel of the retina.



It turns out that it is computationally more convenient to think of this problem in reverse and see what sources light rays leaving each pixel of the retina: if the emitted ray hits no source, the pixel gets no input (black) and if it hits source S_i, then it gets the color of this source.

This toy model evacuates notions of focus and many other details but is good enough for our purposes. In our set up the sources will be placed on a celestial sphere of large radius centred at the black hole. The main difficulty becomes that because of the curvature of the Kerr metric, light does *not* travel in straight lines. For each ray leaving the retina, we must solve non-trivial geodesic equations to obtain the source it hits. We will also have to deal with horizons and add an accretion disk.

3.2 Our setup

For our ray tracer, the light sources (stars) will be placed on a distant surface of constant $R_{\text{cel}} \gg R > a$. We call this surface the *celestial sphere*, even though it is only approximately sphere (in the limit $a/R_{\text{cel}} \ll 1$). The camera will is placed at $(0, r(0), \theta(0), \phi(0))$ with small $\theta(0) > 0$ and with $R_{\text{cel}} > r(0) \gg R$. It is no loss of generality to set $\phi(0) \equiv 0 \pmod{2\pi}$.

3.3 Initial conditions for the geodesic equations

For each pixel, we will need, in addition to the initial position $(0, r(0), \theta(0), \phi(0))$, initial data for the conjugate momenta $(p_t(0), p_r(0), p_\theta(0), p_\phi(0))$. Those are determined by the position of the retina and the requirement that the initial tangent vector is (approximately) light-like.

The screen is described by physical height h_P, width w_P and distance d_P from the observer $(0, r(0), \theta(0), 0)$, which is in Minkowski coordinates $(0, \sqrt{r(0)^2 + a^2}\sin\theta(0), 0, r(0)\cos\theta(0))$. A pixel (i, j) of the screen corresponds, in cartesian cordinates to a position $(0, \sqrt{r(0)^2 + a^2}\sin\theta(0) + d_P, w(i,j), r(0)\cos\theta(0), h(i,j))$ where

$$w(i,j) = -\frac{w_P}{2} + (j-1)\frac{w_P}{J-1},$$
$$h(i,j) = \frac{h_P}{2} - (i-1)\frac{h_P}{I-1},$$

and I and J are the number of vertical and horizontal pixels respectively (the resolution of the image).

The momenta are determined by the initial coordinate velocities:

$$\begin{aligned}
\dot{t}(0) &= 1,\\
\dot{x}(0) &= \frac{d_P}{\sqrt{d_P^2+h^2+w^2}},\\
\dot{y}(0) &= \frac{w}{\sqrt{d_P^2+h^2+w^2}},\\
\dot{z}(0) &= \frac{h}{\sqrt{d_P^2+h^2+w^2}}.
\end{aligned}$$

Note that this vector would be light-like in flat space-time and is therefore approximately light-like in the regime $r(0) \gg R, a$. Pushed in Boyer–Lindquist coordinates, using the Jacobian of the coordinates transformations,

$$\begin{aligned}
\dot{t}(0) &= 1,\\
\dot{r}(0) &= \frac{2\left(r(0)\sqrt{a^2+r(0)^2}d_P\sin\theta(0) + h\left(a^2+r(0)^2\right)\cos\theta(0)\right)}{\sqrt{d_P^2+h^2+w^2}\left(a^2\cos(2\theta(0))+a^2+2r(0)^2\right)},\\
\dot{\theta}(0) &= \frac{2\left(\sqrt{a^2+r(0)^2}d_P\cos\theta(0) - hr(0)\sin\theta(0)\right)}{\sqrt{d_P^2+h^2+w^2}\left(a^2\cos(2\theta(0))+a^2+2r(0)^2\right)},\\
\dot{\phi}(0) &= \frac{w\csc\theta(0)}{\sqrt{\left(a^2+r(0)^2\right)\left(d_P^2+h^2+w^2\right)}}.
\end{aligned}$$

The corresponding momenta $p_\mu(0) = g_{\mu\nu}\dot{x}^\nu(0)$ are thus

$$\begin{aligned}
p_t(0) &= -1 - \frac{ar(0)Rw\sin\theta(0))}{\Sigma(r(0),\theta(0))\sqrt{\left(a^2+r(0)^2\right)\left(d_P^2+h^2+w^2\right)}} + \frac{r(0)R}{\Sigma(r(0),\theta(0))},\\
p_r(0) &= \frac{2\Sigma(r(0),\theta(0))\left(r(0)\sqrt{a^2+r(0)^2}d_P\sin\theta(0) + h\left(a^2+r(0)^2\right)\cos\theta(0)\right)}{\Delta(r(0))\sqrt{d_P^2+h^2+w^2}\left(a^2\cos(2\theta(0))+a^2+2r(0)^2\right)},\\
p_\theta(0) &= \frac{2\Sigma(r(0),\theta(0))\left(\sqrt{a^2+r(0)^2}d_P\cos\theta(0) - hr(0)\sin\theta(0)\right)}{\sqrt{d_P^2+h^2+w^2}\left(a^2\cos(2\theta(0))+a^2+2r(0)^2\right)},\\
p_\phi(0) &= \frac{\sin\theta(0)\left(\frac{\left(\left(a^2+r(0)^2\right)^2 - a^2\sin^2\theta(0)\Delta(r(0))\right)w}{\sqrt{\left(a^2+r(0)^2\right)\left(d_P^2+h^2+w^2\right)}} - ar(0)R\sin(\theta)\right)}{\Sigma(r(0),\theta(0))}.
\end{aligned}$$

4 Beyond the math: graphics and the code

4.1 Generating a uniform celestial sphere

We want a celestial sphere (in the approximation $a \ll R_{\text{cel}}$) that is uniformly filled with stars. If we were to take a standard planar image of stars from the web, or generate stars uniformly at random on a plane (with 2:1 image ration because the angular coordinates are $(\theta, \phi) \in (0, \pi) \times [0, 2\pi)$), once wrapped, stars would accumulate with abnormally high density around the poles $\theta = 0$ and $\theta = \pi$.

However, people have worked on ray tracers for such systems already and have already deformed planar images appropriately to provide uniform skies once wrapped. We will use their images, see for example Figure 1

Figure 1: An example of a properly deformed image. REF.

4.2 Case analysis

Solving the geodesic equations with initial conditions corresponding to the pixel (i, j), we distinguish three main cases:

1. the ray exiting the camera leaves hits the celestial sphere far from the black hole;
2. the ray enters the event horizon of the black hole.

In the first case, hitting the celesting sphere means that a given step of the RK4 results in a point outside the celestial sphere. A choice should then be made to keep the coordinates of the point closest to the sphere. These coordinates are stored, and they are later used to generate the corespondence between the RGB value of each pixel given an image. In the second step, we approach the black hole with adaptive steps to prevent blow up. If we a ray steps close within a margin error to the black hole, we consider that it enters the event horizon. A boolean is kept to indicate the the RGB triple associated to that pixel should indicate pure black.

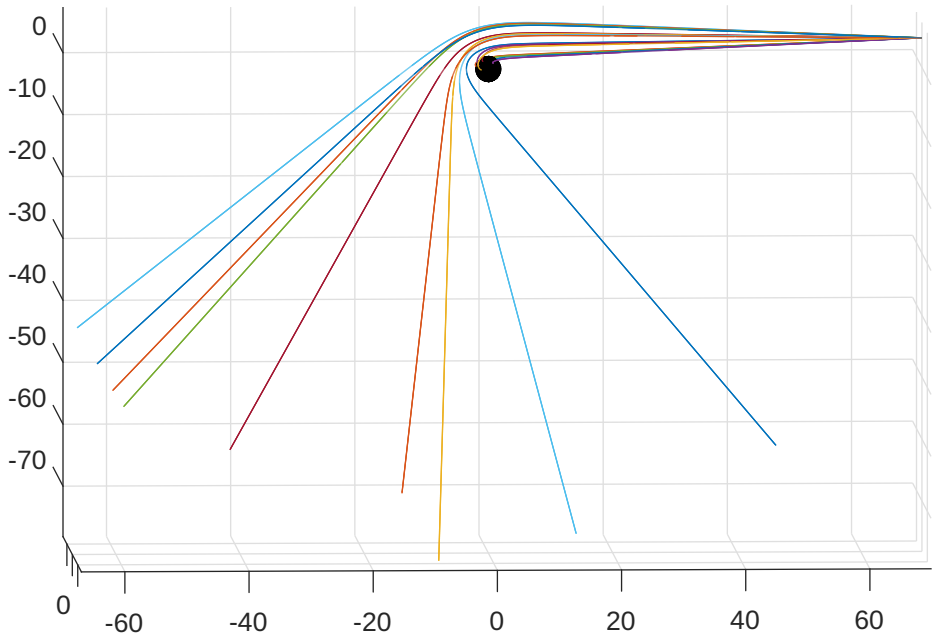


Figure 2: The rays emanate from the camera on the right of the image: some enter the event horizon (black sphere); others continue their curved trajectory until they hit the celestial sphere.

4.3 Adding an accretion disk

Without going into the physics of accretion disks (see for example [AF13]), we wish to visualize how the radiation from matter spinning in the equatorial plane of the Kerr metric would deformed on an image of a rotating black hole. We do this by adding a third case in our ray tracer:

3. the ray hits the equatorial plane $\theta = \frac{\pi}{2}$ for a radius in an interval $[r_{\text{disk min}}, r_{\text{disk max}}]$ defining the accretion disk.

This last situation is treated independently of the two cases presented in the last section, so that the user can choose whether to generate the accretion disk or not. The proceedure is in that case similar.

4.4 Coordinate problems

Note that the Boyer–Lindquist coordinates are ill-defined at $\theta = 0$ and $\theta = \pi$, which corresponds to the z-axis of Euclidean coordinates. For this reason, the geodesic equations in Boyer–Lindquist are not well-behaved near this axis, causing the solution behave wildly.

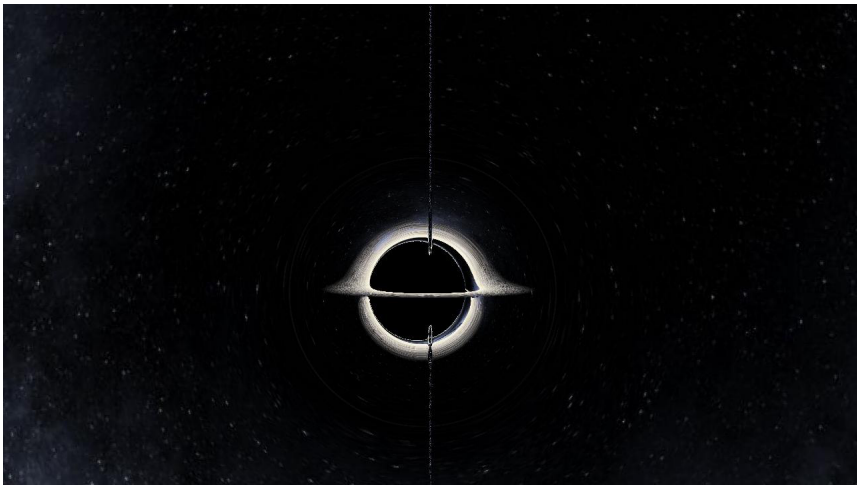


Figure 3: The rays passing near the z-axis behave wildly, causing a serious defect at the center of the image.

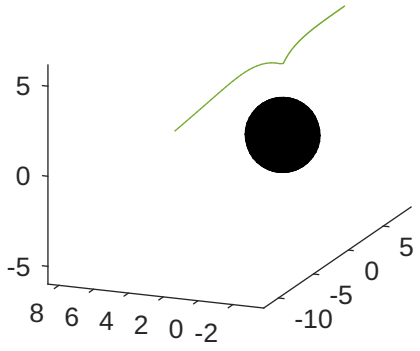


Figure 4: Trajectory of a ray passing near the z-axis.

We get around this problem by sneakily removing a certain number of columns of pixels in the middle of the image... More on this in Section 5.

5 Possible improvements

Although we are satisfied (given the time that was available) with the images generated, there are a few things that we have not taken into account and that would improve the realism of the depiction. First, as we mentioned earlier, the problem with the Boyer–Lindquist coordinates in the vicinity of at $\theta = 0$ and $\theta = \pi$ should be taken care of by a proper change of coordinates. Also, we have not considered the fact that the light rays emitted by the stars should undergo gravitational red-shift. Indeed, we should adjust the color of the pixels according to this effet. Moreover, our treatment of what happens near — but not quite inside — the event horizon is rather crude.

References

[AF13] Marek A. Abramowicz and P. Chris Fragile, *Foundations of black hole accretion disk theory*, Living Reviews in Relativity **16** (2013), no. 1, 1.

[JvTFT15] Oliver James, Eugénie von Tunzelmann, Paul Franklin, and Kip S Thorne, *Gravitational lensing by spinning black holes in astrophysics, and in the movie interstellar*, Classical and Quantum Gravity **32** (2015), no. 6, 065001.

[Ker63] Roy P. Kerr, *Gravitational field of a spinning mass as an example of algebraically special metrics*, Phys. Rev. Lett. **11** (1963), 237–238.

A Code for ray tracing

```
%% RAY TRACING IN KERR SPACETIME %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This program generates the spherical coordinates associated to the      %
% endpoints of geodesics segments seny from an observational window into  %
% a celestial sphere whose center holds a Kerr blackhole                   %
%                                                                          %
% FEATURES:                                                                %
%                                                                          %
% - Generates coordinates for two images to allow for comparison:          %
%        1. Kerr spacetime                                                 &
%        2. Euclidean space                                                &
%                                                                          &
% - Coordinates for an ACCRETION are also obtained independently.          &
%                                                                          &
% - Can be used to plot the rays around the blackhole independently using  &
%   the matlab plot3 function:                                             %
%        Type 1 : plots the rays individually in matlab plot3              %
%        Type 0 : generate jpeg image by ray tracing                       %
         plot_type = 1;                                                    %
%                                                                          %
% - Image to be wrapped onto the celestial sphere                          %
        %img_scene = imread('bgedit.jpg');                                 %
%                                                                          %
% - Choice of image resolution (resolution_height x resolution_width):     %
         resolution_height = 4; % pixels                                   %
         resolution_width  = 4; % pixels                                   %
%                                                                          %
% - Boyer-Lindquist coordinate sytem                                       %
%                                                                          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
tStart = tic;

% Metric related data and eqs.
G = 1; % gravitational constant
M = 1; % mass of blackhole
a = 0.6; % Kerr (spin) parameter
R = 2 * G * M;
radius_celestial_sphere = 80;
aDiskMin = 2*R;
aDiskMax = 5*R;

% Plot black sphere for the black hole if 'plot_type  = 1' is chosen
% and hold on the the plot
if plot_type == 1
    [X,Y,Z] = sphere;
    colormap([0,0,0])
    surf(R*X,R*Y,R*Z)
    axis equal
    hold on
end

% Some physical formulas (used to lighten notation)
Sigma = @(r,theta) r^2 + a^2 * cos(theta)^2;
drSigma = @(r) 2 * r;
dthetaSigma = @(theta) - 2* a^2 * cos(theta) * sin(theta);
Delta = @(r) r^2 - R * r + a^2;
drDelta = @(r) 2 * r - R;

% Physical dimensions of observational window
window_height = 0.00001;
window_width = (resolution_width/resolution_height)*window_height;
distance_from_window = -2*0.000007;

% Initialize coordinates matrices for image mapping
% coords_no_aDisk(i,j,1): theta
% coords_no_aDisk(i,j,2): phi
% coords_no_aDisk(i,j,3): 1 if stopped at R
%                         0 if stopped at radius_celestial_sphere
coords_no_aDisk = zeros(resolution_height,resolution_width,3);
% coords_aDisk(i,j,1): radius
% coords_aDisk(i,j,2): phi
% coords_aDisk(i,j,3): 0 if did not hit accretion disk
%                      1 if pass through accretion disk
coords_aDisk = zeros(resolution_height,resolution_width,3);

stepsize = 0.1; % stepsize for Runge-Kutta 4

hbar = parfor_progressbar(resolution_width,'Please wait...'); %create the progress bar (resolution_width);
for j = 1:resolution_width
    hbar.iterate(1); % update progress by one iteration
    for i = 1:resolution_height

       % Pixels location on observational window (we ommit the vertical
       % singularity at pi/2 by introducing a jump: singularity_hack)
       %singularity_hack = 0.01;
%        if j < resolution_width/2 - singularity_hack*resolution_width
%             h = window_height/2 - (i-1) * window_height/(resolution_height-1);
```

```
%             w = -window_width/2 + (j-1) * window_width/(resolution_width -1);
%         else
%             h = window_height/2 - (i-1) * window_height/(resolution_height -1);
%             w = -window_width/2 + (singularity_hack*window_width) + (j-1) * window_width/(resolution_width -1);
%         end
%
        h = window_height/2 - (i-1) * window_height/(resolution_height -1);
        w = -window_width/2 + (j-1) * window_width/(resolution_width -1);

        % Initializing initial conditions
        r = 70;
        theta = pi/2 - pi/46; % offset the blackhole to see with of aDisk
        phi = 0;
        t_dot = 1;
        phi_dot = (csc(theta) * w ) / sqrt( ( a^2 + r^2 ) * ( distance_from_window^2 + w^2 + h^2 )) ;
        p_r = 2 * Sigma(r,theta) * ( h * ( a^2 + r^2 ) * cos(theta) + r * sqrt( a^2 +r^2 ) * sin(theta) * distance_from_window )...
                    / ( sqrt( distance_from_window^2 + h^2 + w^2 ) * ( a^2 + 2 * r^2 + a^2 * cos(2*theta) ) * Delta(r) );
        p_theta = 2 * Sigma(r,theta) * ( -h * r * sin(theta) + sqrt( a^2 + r^2 ) * cos(theta) * distance_from_window )...
                    / ( sqrt( distance_from_window^2 + h^2 + w^2 ) * ( a^2 + 2 * r^2 + a^2 * cos(2*theta) ) );

        % Conserved quantities (instantiated with I.C.)
        E = ( 1 - R/r ) * t_dot + ( R * a * phi_dot )/r;
        L = - ( R * a )/r * t_dot + ( r^2 + a^2 + ( R * a^2 )/r ) * phi_dot;

        % Geodesic equations (system of first order ODEs)
        % input : x = [r; theta; phi; pr; ptheta]
        % ouput: dx = [dr; dtheta; dphi; dpr; dptheta] "x dot"
        f = @(lambda, x)  [( x(4) * Delta(x(1)) ) / Sigma(x(1),x(2)) ;
                           x(5) / Sigma(x(1),x(2));
                           ( a * ( -a * L + x(1) * R * E )+ L * csc(x(2))^2 * Delta(x(1)) )/( Delta(x(1)) * Sigma(x(1),x(2)) );
                           - ( 1 / ( 2 * Delta(x(1))^2 * Sigma(x(1),x(2))^2) ) * (Sigma(x(1),x(2)) * (-E * Delta(x(1)) *...
                           ( a * R * ( -2 * L + a * E * sin(x(2))^2 )+2 * x(1) * E * Sigma(x(1),x(2)) )...
                                    + (  a * ( a * L^2 - 2 * L * x(1) * R * E + a * x(1) * R * E^2 * sin(x(2))^2) + x(4)^2 *...
                                    Delta(x(1))^2 + ( a^2 + x(1)^2 ) * E^2 * Sigma(x(1),x(2)) ) * drDelta(x(1)) )...
                                    + Delta(x(1)) * ( a * ( L * ( a * L - 2 * x(1) * R * E )+ a * x(1) * R * E^2 * sin(x(2))^2 )...
                                    - Delta(x(1)) * ( x(5)^2 + L^2 * csc(x(2))^2 + x(4)^2 * Delta(x(1)) ) ) * drSigma(x(1)) );
                           -( 1 / ( 2 * Delta(x(1)) * Sigma(x(1),x(2))^2) ) * ( -2 * sin(x(2)) * ( a^2 * x(1) * R * E^2 * cos(x(2))...
                           + L^2 * cot(x(2)) * csc(x(2))^3 * Delta(x(1)) ) * Sigma(x(1),x(2))...
                                    + ( a * (L * (a * L - 2 * x(1) * R * E ) + a * x(1) * R * E^2 * sin(x(2))^2 ) - Delta(x(1))...
                                    * ( x(5)^2 + L^2 * csc(x(2))^2 + x(4)^2 * Delta(x(1)) ) ) * dthetaSigma(x(2)) )];

        % Solving for the geodesics using Runga-Kutta 4
        % RK4 parameters
        x_0 = [ r ; theta ; phi ; p_r ; p_theta ];
        curve = x_0';

        switch plot_type
            case 1
                k = 1;
                % Curves are computed within the celestial sphere where they exist
                while (R < curve(k,1)) && (curve(k,1) < radius_celestial_sphere) && (k<20000)

                    % Clean coordintates values
                    curve(k,2) = mod(curve(k,2),2*pi);
                    curve(k,3) = mod(curve(k,3),2*pi);
                    if curve(k,2) > pi
                        curve(k,2) = 2*pi - curve(k,2);
                        curve(k,3) = mod(pi + curve(k,3),2*pi);
                    end

                    k = k+1;

                    % Use Runge-Kutta 4 step to evolve geodesic curve
                    % Some adaptativity is obtained by multiplying stepsize
                    % by Delta, which scale inversly to the coordinates
                    % singularity near the event horizon
                    curve(k,:) = rk4step(f, 0, curve(k-1,:)' , min([stepsize*Delta(curve(k-1,1));stepsize]))';
                end

                % Transform to euclidean coordinates and plot3
                [n,m] = size(curve);
                A = a*ones(n,1);
                Boyer2Cart = @(r,theta,phi) [sqrt(r.^2 + A.^2).*sin(theta).*cos(phi),...
                                             sqrt(r.^2 + A.^2).*sin(theta).*sin(phi),...
                                             r.*cos(theta)];
                cart = Boyer2Cart(curve(:,1),curve(:,2),curve(:,3));
                plot3(cart(:,1),cart(:,2),cart(:,3));

            case 0

                k = 1; % keeping count of steps
                passed_through_aDisk = 0;

                % Curves steps are computed within the celestial sphere where they exist
                while (1.2*R < curve(1)) && (curve(1) < radius_celestial_sphere) && (k<20000)

                    % Use Runge-Kutta to take a temporary step foward
                    temp_curve_step = rk4step(f, 0, curve' , min([stepsize*Delta(curve(1));stepsize]))';
```

```
                    if passed_through_aDisk == 0
                        % Check if the temporary step go through the accretion disk
                        if (temp_curve_step(2)-pi/2)*(curve(2)-pi/2) < 0
                            if ( aDiskMin < temp_curve_step(1) ) && ( temp_curve_step(1) < aDiskMax )
                                coords_aDisk(i,j,:) = [temp_curve_step(1); temp_curve_step(3);  1];
                                passed_through_aDisk = 1;
                            end
                        end
                    end

                    % Update curve with the temporary step
                    curve = temp_curve_step;

                    % Clean coordinates values
                    curve(2) = mod(curve(2),2*pi);
                    curve(3) = mod(curve(3),2*pi);
                    if curve(2) > pi
                        curve(2) = 2*pi - curve(2);
                        curve(3) = mod(pi + curve(3),2*pi);
                    end

                    k = k+1;
                end

                % Save coordinates according to cases
                if (1.2*R < curve(1))
                    coords_no_aDisk(i,j,:) = [curve(2); curve(3); 0];
                else
                    coords_no_aDisk(i,j,:) = [curve(2); curve(3); 1];
                end
        end

    end
end
close(hbar);

if plot_type == 1
    hold off
end

% Save coordinates as maltab file
save img_Map_no_BH coords_no_aDisk coords_aDisk aDiskMin aDiskMax;

%% Generating final images from different celestial scenes

image_type_with_disk = 1;
boolean_aDisk = 1;

load('img_Map_no_BH');

[resolution_height,resolution_width, ~] = size(coords_no_aDisk);

accretion_disk_scene = imread('adisk_skewed.png');
[accretion_scene_height_res, accretion_scene_width_res, ~] = size(accretion_disk_scene);

k = 1;

    switch k
        case 1
            celestial_scene = imread('GriddedGal.jpg');
        case 2
            celestial_scene = imread('InterstellarWormhole_Fig6b.jpg');
        case 3
            celestial_scene = imread('InterstellarWormhole_Fig10.jpg');
    end

    % Resolution data related to scene
    [celestial_scene_height_res, celestial_scene_width_res, ~] = size(celestial_scene);


    IMG = zeros(resolution_height,resolution_width,3, 'uint8');

    % Generate image
    for j = 1:resolution_width
        for i = 1:resolution_height

            if image_type_with_disk == 1
                if boolean_aDisk == 1
                    if coords_aDisk(i,j,3) == 1

                        % Clean coordinates values
                        coords_aDisk(i,j,2) = mod(coords_aDisk(i,j,2),2*pi);
                        if coords_aDisk(i,j,2) > pi
                            coords_aDisk(i,j,2) = 2*pi - coords_aDisk(i,j,2);
                        end

                        %coords_aDisk(i,j,2)
```

```
                        %round(coords_aDisk(i,j,2)*((accretion_scene_width_res-1)/(2*pi))+1)

                        if coords_no_aDisk(i,j,3) == 1
                            IMG(i,j,:) = accretion_disk_scene(round((coords_aDisk(i,j,1)-aDiskMin)*((accretion_scene_height_res-1)/(aDiskMax - aDiskMin))+1),...
                                round(coords_aDisk(i,j,2)*((accretion_scene_width_res-1)/(2*pi))+1),:);
                        else

                            IMG(i,j,:) = max(accretion_disk_scene(round((coords_aDisk(i,j,1)-aDiskMin)*((accretion_scene_height_res-1)/(aDiskMax - aDiskMin))+1),...
                                round(coords_aDisk(i,j,2)*((accretion_scene_width_res-1)/(2*pi))+1),:),...
                                celestial_scene(round(coords_no_aDisk(i,j,1)*((celestial_scene_height_res-1)/pi)+1),...
                                round(coords_no_aDisk(i,j,2)*((celestial_scene_width_res-1)/(2*pi))+1),:));

                        end
                        boolean_aDisk = 0;
                    end
                end
            end

            if boolean_aDisk == 1
                if coords_no_aDisk(i,j,3) == 1
                    IMG(i,j,1) = 0;
                    IMG(i,j,2) = 0;
                    IMG(i,j,3) = 0;

                else
                    IMG(i,j,:) = celestial_scene(round(coords_no_aDisk(i,j,1)*((celestial_scene_height_res-1)/pi)+1),...
                        round(coords_no_aDisk(i,j,2)*((celestial_scene_width_res-1)/(2*pi))+1),:);
                end
            end

            boolean_aDisk = 1;
        end
    end

    IMG = [IMG(:,1:(resolution_width/2-3),:) IMG(:,(resolution_width/2-3),:) max(IMG(:,(resolution_width/2-3),:),IMG(:,resolution_width/2+5,:))...
        IMG(:,resolution_width/2+5,:) IMG(:,(resolution_width/2+5):resolution_width,:)];

    switch k
        case 1
            imwrite(IMG,'grid','jpg')
        case 2
            imwrite(IMG,'test2_InterstellarWormhole_Fig6_BH_disk.jpg','jpg')
        case 3
            imwrite(IMG,'test2_InterstellarWormhole_Fig10_BH_disk.jpg','jpg')
    end

%Final computations time
tEnd = toc(tStart);
fprintf('%d minutes and %f seconds\n',floor(tEnd/60),rem(tEnd,60));
```