Schwarzschild black hole simulation using Python

Jonathan Peltier[1]

[1]Master PAIP, Université de Lorraine

Abstract

In this article, a non-spinning black hole (Schwarzschild) simulation using python 3.6 is presented. There are already existing simulation using HTML and JavaScript, like the ESA precomputed one [1] or Alessandro Roussel's one with modifiable variables [2]. Photons trajectories are computed on the equator by solving the geodesics equation numerically. Then, the whole black hole is computed from the equatorial trajectories thanks to the spherical symmetry of the solutions.

I. Photon trajectories

The geodesics equation (1) described in [3] is solved using the Schwarzschild metric.

$$\frac{d^2\gamma^\lambda}{dt^2} + \Gamma^\lambda_{\mu\nu}\frac{d\gamma^\mu}{dt}\frac{d\gamma^\nu}{dt} = 0 \tag{1}$$

The symmetry in our problem imply that only equatorial trajectories are needed: $\theta = \frac{\pi}{2}$. After a few steps described in [4], an ordinary differential equation describing the photon trajectory is obtained:

$$\frac{d^2u}{d\phi^2} - \frac{3R_s}{2}u^2 + u = 0 \tag{2}$$

With $u(\phi) = \frac{1}{r(\phi)}$ and $R_s = \frac{2MG}{c^2}$, R_s: Black hole radius, M: black hole mass, G: Gravitational constant, c: light speed in vacuum.

A. Numerical integration

Scipy.integrate.solve_ivp is used with an implicit Runge-Kutta method called Radau imposed by our stiff equation. This method allows to remain accurate where an explicit method would diverge. Before integrate the equation, our second order equation must be transformed into two first order equations. (2) become:

$$\begin{aligned} v_0 &= u[1] \\ v_1 &= \frac{3R_s}{2}u[0]^2 - u[0] \end{aligned} \tag{3}$$

Where $u[0] = u(\phi)$ and $v_0(0)$ represent the initial position, $u[1] = \frac{du}{d\phi}$ and $v_1(0)$ the initial direction.
The backwards ray-tracing method described in [1] is used to know the photon origin. The initial conditions represented on the figure 1 are in our case (see [5]):

$$\begin{aligned} v_0(0) &= \frac{1}{D} \\ v_1(0) &= \frac{1}{D.tan(\alpha)} \end{aligned} \tag{4}$$

Results can be used to plot trajectories or just return the *deviated_angle* which corresponds to the reel pixel position seen for a *seen_angle*. Differents trajectories are represented on the figure 2. Angles are found using relations (5):

$$\begin{aligned} seen\_angle &= \pi - \alpha \\ deviated\_angle &= \phi + asin(\frac{D}{R}\sin(\phi)) \end{aligned} \tag{5}$$

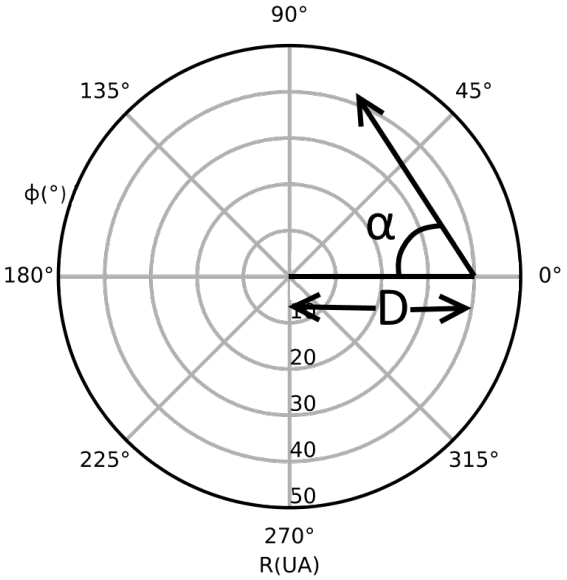

Fig. 1. Initial conditions in polar coordinate with a black hole on the center. D: distance from the black hole, α: angle defining the initial direction.

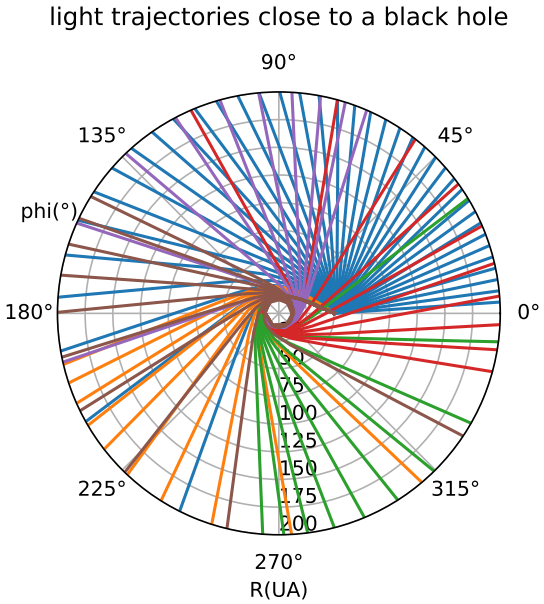

Fig. 2. compute trajectories for D=50, Rs=8. We can see in colors the different precision steps used to cover all possible alpha.

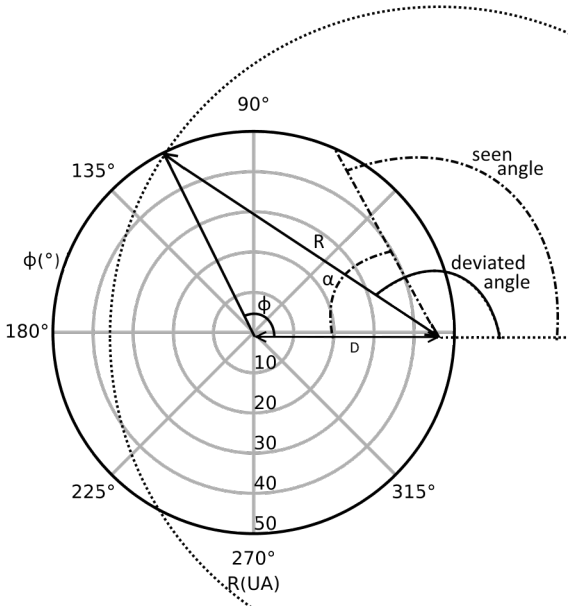

Fig. 3. Relation between seen, deviated angles and α, ϕ. D: distance from the black hole, R: vision radius. α is the initial angle and ϕ the final angle.

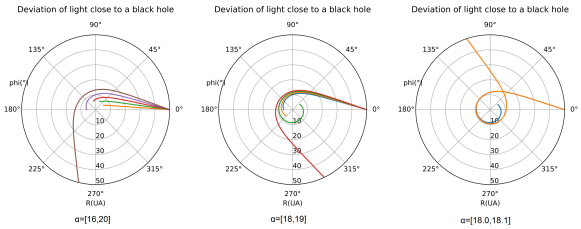

Fig. 4. Illustrate the α_min search. First step: $\alpha \in [16, 20]$. Second step: $\alpha \in [18, 19]$. Third step: $\alpha \in [18.0, 18.1]$.

$\psi = \pi - \phi - asin(\frac{D}{R}\sin\phi)$ see [6]
$deviated\_angle = \pi - \psi$
The angles relations are represented on the figure 3.

II. Interpolation

An interpolation must be performed to avoid calculating trajectories for each pixel. This interpolation must be correct for any situations. Firstly, α_min, for which the photon is not captured by the black hole, must be calculated. Then, this angle is used to choose computed angles properly.

A. Search alpha_min

A root finder could have been used to find the α_min. For example, *scipy.optimize*. But, we built instead our own method. It consist of computing trajectories from a null angle to 180 degrees with a big step and stop when the photon escapes the black hole. Then, start again with the last captured photon with a smaller step. This operation is iterated until a desire precision, fixed by the image size, is reached. Three iterations are shown on the figures 4.

B. Trajectories choice

We decided to compute 40 points between 180 and α_min. Then compute five times, 10 points from

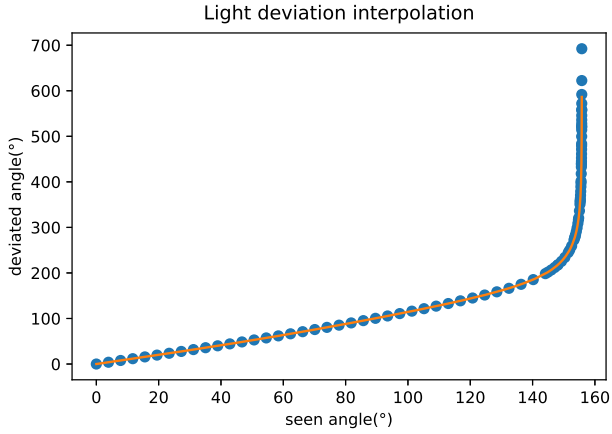

Fig. 5. interpolation for D=50, Rs=8.

$\alpha\_finder = \alpha\_min + \delta$ to α_min as seen on the figure 2. δ is chosen arbitrary to cover a proper area. δ is chosen $= \frac{\alpha\_finder - \alpha\_min}{\frac{points}{3}+1}$. A linear interpolation is used instead of a cubic one because the interpolation processing time is higher for a cubic than a linear one. If a cubic interpolation is used, δ must be chosen carefully to have a sorted array. δ could become in this situation: $\delta = \frac{\alpha\_finder - \alpha\_min}{points+1}$. Once all $seen\_angle$ and $deviated\_angle$ are gather in a list, the interpolation give us all possible trajectories on the equatorial plan. The interpolation $deviated\_angle(seen\_angle)$ is represented on the figure 5.

III. Images creation

The program is made for equirectangular image so, (x, y) coordinates correspond to (ϕ, θ) coordinates. In our situation, the (ϕ_2, θ_2) deviated pixel color, to apply on the (ϕ, θ) seen pixels, must be found. These data are then stored in matrices to skip computation when using the same parameters.

A. Pixel position

The spherical coordinates (ϕ,θ) are transformed into Cartesian coordinates to compute β and rotate the pixels on the equator using a rotation matrix simplified from [7].

$$\beta = -atan(\frac{z}{y}) \tag{6}$$

The new coordinate give use $\phi' = seen\_angle$ and $\theta' = \frac{\pi}{2}$. The interpolation can be used to find the $deviated\_angle = \phi'_2$. Note that in our case, the interpolation is only done with $\phi' \in [0, \pi]$. So, if $\phi' > \pi$, ϕ' become $2\pi - \phi'$ and ϕ'_2 become $2\pi - \phi'_2$ (only valid for a Schwarzschild black hole). Then, ϕ'_2 is transformed into Cartesian coordinate and rotated back by a $-\beta$ angle. Finally, the Cartesian coordinates are transformed into spherical coordinate in order to return $(\phi_2, \theta_2) = (x_2, y_2)$ When inside the black hole, this function returns (-1,-1) to skip the corresponding pixel. The process is shown on the figure 6.

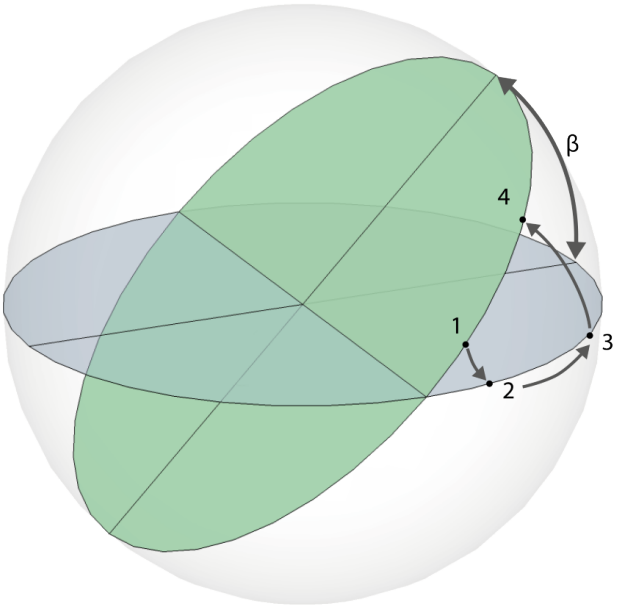

Fig. 6. Schematic illustrating the coordinate operations. 1: ϕ,θ $\Rightarrow$ x,y,z / Compute β / β rotation: x,y,z $\Rightarrow$ x',y',z' 2: x',y',z' $\Rightarrow$ ϕ' / Interpolation : ϕ' $\Rightarrow$ ϕ_2' 3: ϕ_2' $\Rightarrow$ x_2',y_2',z_2' / -β rotation: x_2',y_2',z_2' $\Rightarrow$ x_2,y_2,z_2 4: x_2,y_2,z_2 $\Rightarrow$ ϕ_2,θ_2

Fig. 7. Milky Way without black hole.

B. Matrices

Two matrices are created to store raw black hole deformation and to use it back on other image with the same parameters or to use it on the same image with a offset (to create a moving black hole). One store x_2 position, the other y_2. The columns and rows correspond to the pixel position (x,y). This function takes deviated positions (x_2, y_2) seen in section III-A for every seen pixels (x, y) and store x_2 and y_2 on the $element_{yx}$. If matrices already exist, the program skip the matrices creation and load them instead.

C. Image modification

At the end, loops browse every pixels and attribute the (x_2, y_2) pixel color (R,G,B) information to the (x, y) pixel using matrices. If the $element_{yx}$ return (-1,-1), no pixel is computed. A try/except block is used to color or skip pixels in a "out of interpolation range" representing the missing information induced by images with $FOV_X < 360$ or $FOV_Y < 180$. The images 7 and 8 show the light deformation induced by a R_s=8 UA radius black hole viewed from a distance D=50 UA (the background is considered situated to infinity because of the solve.ivp stops condition malfunction). The image 9 shows the missing information in white as described before.

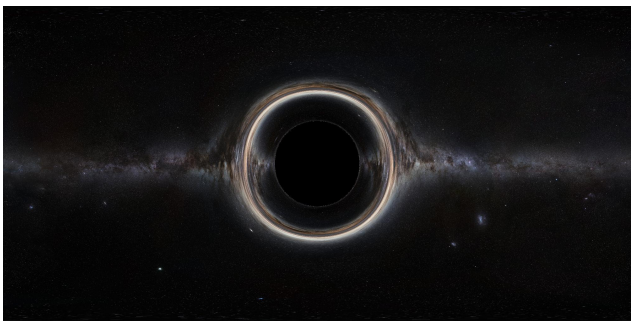

Fig. 8. R_s=8 UA radius black hole view from a distance D=50 UA.

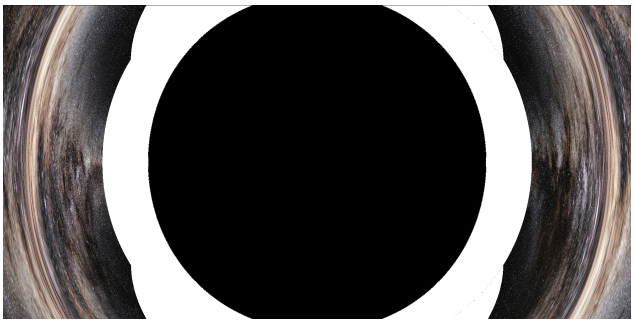

Fig. 9. R_s=8 UA radius black hole view from a distance D=50 UA with a 90 degrees image.

D. X offsets

The offset function is used to simulate a moving black hole or observer. It takes an image, cut in two pieces define by the offset and interposes them to have the new image. Only the x offset is computed because the y offset need complex operations due to the equirectangular images properties. This function is used in a loop to save, on the computer, images with differents offsets. Then, they can be combined using Gimp to obtain a moving picture.

References

[1] ESA. ESA ACT blackhole visualization. https://www.esa.int/gsp/ACT/phy/Projects/Blackholes/WebGL.html. [Accesed on December-2018].

[2] Alessandro Roussel. Black Hole #1. http://www.alessandroroussel.com/creationsList/blackhole/. [Accesed on December-2018].

[3] Wikipedia. Geodesic: Affine geodesics. https://en.wikipedia.org/wiki/Geodesic#Affine_geodesics. [Accesed on December-2018].

[4] Denis Gialis. Geodésiques dans l'espace-temps de Schwarzschild. http://www.astrosurf.com/dg-homepage/pb1_rg.pdf. [Accesed on December-2018].

[5] Alessandro Roussel. Comment simuler un trou noir ? https://www.youtube.com/watch?v=PjWjZFwz3rQ. [Accesed on December-2018].

[6] Wikipedia. Résolution d'un triangle. https://fr.wikipedia.org/wiki/Résolution_d'un_triangle#Un_angle,_le_côté_opposé_et_un_côté_adjacent. [Accesed on December-2018].

[7] stack overflow. Rotation of 3D vector? https://stackoverflow.com/questions/6802577/rotation-of-3d-vector. [Accesed on December-2018].