Visualisierungsinstitut der Universität Stuttgart (VISUS)
Allmandring 19
70569 Stuttgart

Bachelorarbeit

General-Relativistic Polygon Rendering

Felix Winterhalter

Course of Study:	Informatik
Examiner:	Prof. Dr. Daniel Weiskopf
Supervisor:	Dr. Thomas Müller, Dipl.-Inf. Christoph Schulz
Commenced:	May 25, 2020
Completed:	November 25, 2020

Abstract

General relativistic visualization addresses optical effects in the realm of general relativity, such as bending of light around massive objects like black holes and stars, affecting the appearance of objects in their proximity. In this thesis, we propose a method for real-time relativistic visualization of triangle meshes in the proximity of black holes, based on polygon rendering and a precomputed table containing information about the distortion and apparent locations of points to a static observer. The visualization process is based on standard polygon rendering through the OpenGL pipeline, but we apply a subdivision to heavily distorted triangles to achieve more accurate results (approximating the curving of straight lines in proximity of a black hole). We consider a static observer with a freely rotating camera, as well as movable objects (considered 'quasistatic', i.e. movement itself does not impact appearance, only its location in space). Some results are compared with renders from the ray tracing software GeoVis and we provide additional example visualizations of different objects and scenarios, as well as benchmarks for both the visualization as well as the precomputation-step of our method.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

FPS Frames per Second. 40

GR General Relativity. 17

SR Special Relativity. 17

TCS Tessellation Control Shader. 33

TES Tessellation Evaluation Shader. 34

1 Introduction

1.1 Task Description and Scope

Goal of this thesis is to develop and evaluate a novel method for real-time visualization of distortion effects in the vicinity of a Schwarzschild black hole based on “classic” polygon-rendering. This method should be used to transform a triangle mesh accordingly, and subdivide it where necessary. In developing the method, only static Schwarzschild-spacetime is considered, and its symmetry is taken advantage of where possible to increase efficiency in terms of performance and memory. Object movement from interaction is considered ‘quasistatic’, which means visualizations appear as if the object was standing still at any given time (in general relativity, movement heavily affects visual effects). The observer is static in all cases. Algorithms are implemented in C++ with the OpenGL graphics specification.

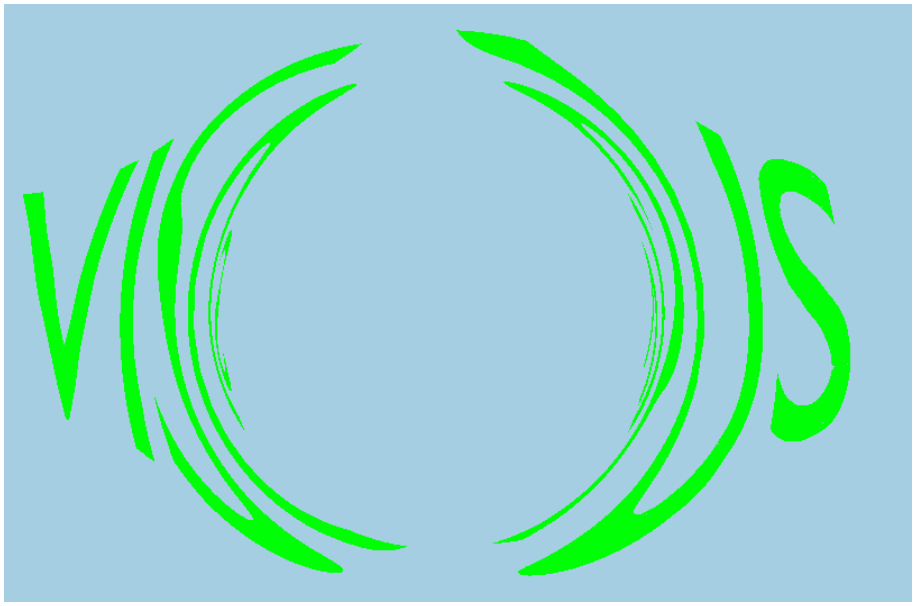

Figure 1.1: Deformed VISUS-lettering

1.2 Time Management

The task can be split into three more or less separate parts:

- Getting familiar with literature, the current state of research and the available tools

- Developing a program to precompute lookup-textures used in step 3
- Developing the actual visualization making use of the lookup-textures generated in step 2

Each step is estimated to take roughly 2 months to complete.

2 Fundamentals

2.1 Physical Fundamentals

This section is divided into 3 parts: A short introduction to Special Relativity (SR), an introduction to General Relativity (GR) and how it relates to SR, and lastly a more formal part in which relevant equations and mathematical elements are presented.

2.1.1 Special Relativity

Prior to Einstein's proposal of SR in 1905 it was assumed that light (like sound) could be described as a wave phenomenon [BMW16, Chapter 2.1] and thus relies on a carrier medium, the postulated "luminiferous aether". There existed multiple different theories about the exact properties and behavior of said aether, like the (partial) aether drag hypothesis, which assumes that the aether is in some way tied to matter and moves with it in space, and on the opposite the hypothesis of a stationary aether encompassing all matter. Towards the end of the 19th century, attempts were made to determine precisely specific parameters in said theories.

Hypothesis of a stationary aether

The "Michelson-Morley-Experiment" (1881/1887) for example had the purpose of determining the velocity of the earth relative to the aether, but its results turned out to be incompatible with a stationary aether: The basic idea was that the speed of light on earth should depend on its direction "inside" the aether and the postulated "aether wind" dragging or slowing it, similar to how a paper ball thrown into the wind will travel slower when thrown against the wind direction, and faster when thrown in the same direction. The experiment consisted of a light source, a beam splitter splitting the emitted beam into two perpendicular parts which then travel the same distances to a mirror and into a telescope where the split beams interfere (Figure 2.1). The entire setup is then slowly rotated while observing the effects in the telescope. Were the hypothesis about a stationary aether true, one would expect the interference pattern in the telescope to change periodically while rotating, since the different angles to the proposed aether would lead to different travel times and travel speeds in the split beams. Michelson and Morley were however unable to observe any change in the interference pattern, thus making the hypothesis of a stationary aether indefensible.

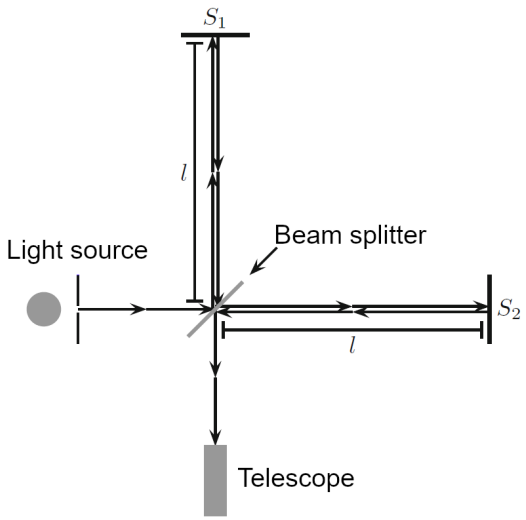

Figure 2.1: Schematic drawing of the Michelson-Morley-experiment (taken from [BMW16])

Aether drag hypothesis

The aether drag hypothesis postulates a "full" attachment of the aether to matter, but is incompatible with observed stellar aberration, a phenomenon where a moving observer observes apparent motion of celestial objects relative to their original positions depending on the speed and direction of the observer. In a world where the aether drag hypothesis is true, the expected offset would depend on the direction and velocity of the observed object as well as the observer, however this is not observable in experiments, as it seems the magnitude of the effect is only determined by the observer's velocity.

Partial aether drag hypothesis

One experiment on the aether drag hypothesis, the "Fizeau-experiment" (1851), was carried out to determine the effect of moving water on the speed of light. According to the hypothesis, any moving matter should drag some aether with it, thus affecting the speed of light depending on its direction, and determining the magnitude of this effect was the goal of the experiment. The setup again consists of a light source and a beam splitter, this time redirecting the two beams through a pipe in opposite directions, again reuniting them in a telescope (Figure 2.2). Water then flows through the pipe and one observes interference in the telescope Again, were the aether drag hypothesis true, one would expect different speeds and different travel times for the beams and thus interference depending on the speed at which water flows through the pipe, but the observations made do not match the predictions in case of "full" aether drag. The results match predictions made in the partial aether drag hypothesis, but this would already contradict the results of the Michelson-Morley-experiment.

Fundamentals of Special Relativity

The basic postulates in Special Relativity as used by Einstein are (Chapter 7.1 in [Gol80]):

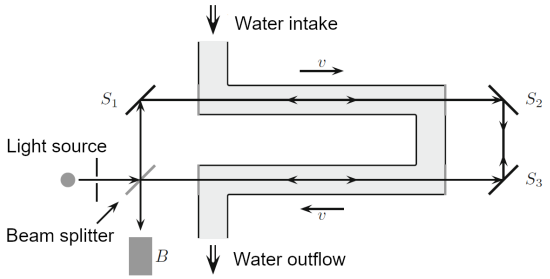

Figure 2.2: Schematic drawing of the Fizeau-experiment (taken from [BMW16])

- "Principle of Relativity":The laws of physics are the same to all inertial (= not experiencing acceleration) observers.
- "Principle of Invariant Light Speed":The speed of light is the same to all inertial observers.

This means that all inertial frames of reference are equivalent in terms of formulating physical laws (more particularly it is impossible to distinguish one inertial frame of reference with an absolute velocity of 0 (Chapter 2.4 in [BMW16])).

2.1.2 General Relativity

The equivalence Principle

"A little reflection will show that the law of the equality of the inertial and gravitational mass is equivalent to the assertion that the acceleration imparted to a body by a gravitational field is independent of the nature of the body. For Newton's equation of motion in a gravitational field, written out in full, it is:
(Inertial mass) · (Acceleration) = (Intensity of the gravitational field) · (Gravitational mass)
It is only when there is numerical equality between the inertial and gravitational mass that the acceleration is independent of the nature of the body." - (P. 59, [EA03])

The equivalence Principle at the base of the theory of GR states that gravitational mass (measured by reaction to gravitation) and inertial mass (its resistance to acceleration) are equivalent. This implies for example that an observer in a laboratory can not determine whether he is experiencing gravity in a "stationary" laboratory or acceleration in space (without gravity). The equivalence Principle alone is already enough to predict curved light beams close to massive objects: Consider a constantly accelerating laboratory with a laser pointer for example, where the laser is pointed horizontally at a detector on the opposite side of the room (2.4). From an outsider's perspective, the light beam in the accelerating laboratory will be a straight line from left to right - however in the time it took the beam to reach the detector, the laboratory has moved a certain distance, hence curving the ray from the perspective of an observer inside the room. The equivalence principle now implies that the accelerating laboratory is equivalent to a laboratory in a gravitational field, hence predicting the same bent beam of light there. The bending of light around massive objects was first observed by Arthur Stanley Eddington during a solar eclipse in 1919: the moon covering the sun made it possible to observe stars close to its position in the sky that would have usually been

overpowered by the sun's brightness. During the solar eclipse, they appeared to be further away from the sun than usual - an effect known as 'gravitational lensing', causing the light beams from the stars to be slightly bent when passing the sun, thus changing their apparent positions in the sky (2.3).

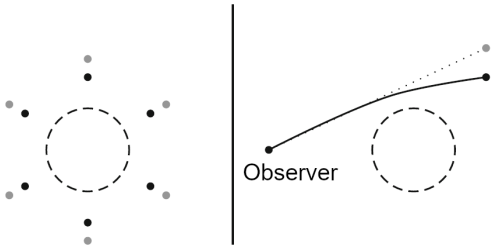

Figure 2.3: Stars during a solar eclipse appearing further away from the sun (grey) than they actually are (black) (taken from [BMW16])

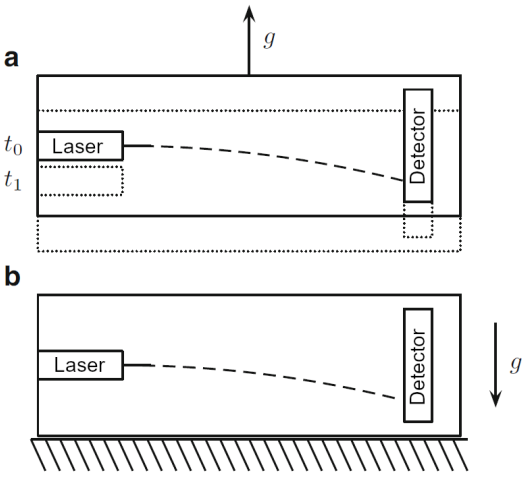

Figure 2.4: A constantly accelerating laboratory (a) is equivalent to a laboratory in a corresponding gravitational field (b) (taken from [BMW16])

2.1.3 (Schwarzschild) Black Holes

Gravitation on the surface of spherical objects increases with their density (1.4.2 in [BMW16]). The "Schwarzschild radius" r_s (2.1) is a measurement for an object's gravitational impact (the bigger an object's r_s the more noticeable its gravitational pull).

$$r_s = 2\frac{GM}{c^2} \tag{2.1}$$

where G is the Gravitational constant, c the speed of light, and M the mass of the object. Given the Schwarzschild radius, we can calculate the escape velocity v_e needed to 'escape' the object starting from a radius R using 2.2.

$$v_e = c\sqrt{\frac{r_s}{R}} \tag{2.2}$$

(2.1, 2.2 taken from [BMW16], Chapter 1.4). When an object's actual radius r is smaller than its Schwarzschild radius r_s the escape velocity grows beyond the speed of light, and not even light will be able to escape from the surface of such an object, rendering it black. In this thesis only Schwarzschild black holes (non-rotating charge-free black holes) will be discussed, the speed of light c will be normalized to 1 for readability. For Schwarzschild black holes, the surface of a sphere with radius r_s forms its 'event horizon', a surface below which no light (or matter) will be able to escape.

2.1.4 Mathematical Formulation

We focus on the mathematical formulation with respect to so-called "null geodesics", paths of photons in Schwarzschild spacetime. In GR, our commonly used three-dimensional space is extended to a four-dimensional "spacetime", by including time as an additional dimension. In this framework, mass and energy "curve" spacetime (just like a heavy object would deform a flexible surface it rests on) and affect the paths of light traveling through it. To analyze trajectories in spacetime, we need a way of calculating the distance between two points (a 'metric') - one that also considers the time-component (as opposed to our usual three-dimensional space, where we generally don't consider time when calculating distances). In the case of point-symmetric mass distribution (like it is the case with a Schwarzschild black hole), the metric that satisfies Einstein's field equations is the **Schwarzschild metric**:

$$ds^2 = -\left(1 - \frac{r_s}{r}\right) dt^2 + \frac{dr^2}{1 - r_s/r} + r^2(d\theta^2 + \sin^2(\theta) d\phi^2) \tag{2.3}$$

In a spacetime with this metric, the paths of light rays ('null-geodesics', see Fig. 2.5 and 2.6) around a Schwarzschild black hole are described with a set of differential equations (2.4 - 2.7), where λ is used to parametrize a curve and (t, r, θ, ϕ) are spherical coordinates with a time component t. This system of second order differential equations can be transformed into a system of first order, which will be done in Chapter 3.

$$\frac{d^2t}{d\lambda^2} = -\frac{r_s}{r(r - r_s)} \frac{dr}{d\lambda} \frac{dt}{d\lambda} \tag{2.4}$$

$$\frac{d^2r}{d\lambda^2} = \frac{r_s}{2r(r - r_s)} \left(\frac{dr}{d\lambda}\right)^2 - \frac{r_s}{2r^3}(r - r_s) \left(\frac{dt}{d\lambda}\right)^2 + (r - r_s) \left[\left(\frac{d\theta}{d\lambda}\right)^2 + \sin^2\theta \left(\frac{d\phi}{d\lambda}\right)^2\right] \tag{2.5}$$

$$\frac{d^2\theta}{d\lambda^2} = \sin\theta \cos\theta \left(\frac{d\phi}{d\lambda}\right)^2 - \frac{2}{r} \frac{d\theta}{d\lambda} \frac{dr}{d\lambda} \tag{2.6}$$

$$\frac{d^2\phi}{d\lambda^2} = -2\frac{\cos\theta}{\sin\theta} \frac{d\theta}{d\lambda} \frac{d\phi}{d\lambda} - \frac{2}{r} \frac{d\phi}{d\lambda} \frac{dr}{d\lambda} \tag{2.7}$$

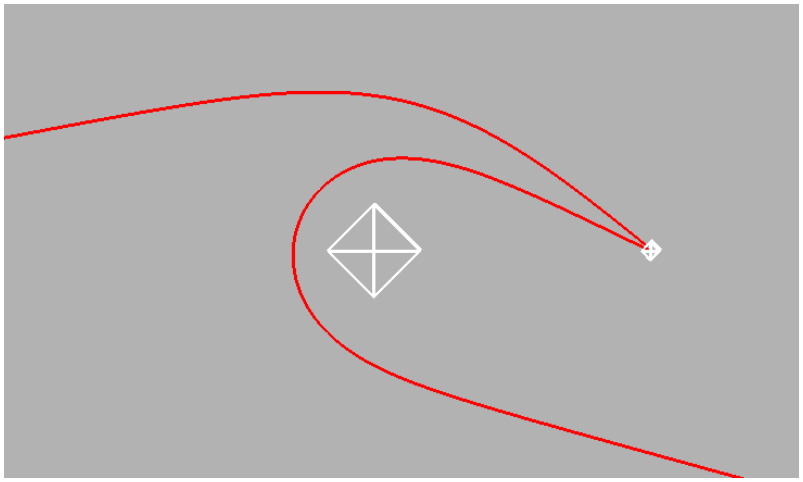

Figure 2.5: Examples for light trajectories in Schwarzschild spacetime

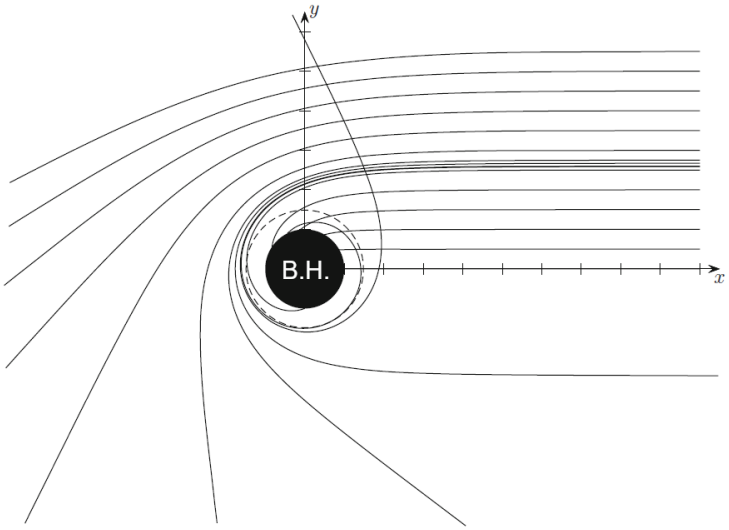

Figure 2.6: Examples for light trajectories in Schwarzschild spacetime

2.2 Thematic Introduction

Relativistic effects are relatively hard to grasp, especially for laymen, so accurate visualization can be used to effectively explain its predicted phenomena as well as provide students or other not-so-laymen with a better understanding of the consequences of SR and GR without having them rely on a deep mathematical background. Generating accurate visual representations of black holes and their surroundings especially has been a research topic investigated from multiple angles: The standard approach for high-quality images in special general relativity is four-dimensional ray tracing, used in [Mül14], and a parallelized approach of this is explored in [CPÖ13]. Other research points include visualization of a black hole's distant surroundings ([MW10]) with focus on real-time performance enabling interactive applications [VE20]. There are also attempts to combine polygon rendering with a local (more performant) variant of ray tracing [MGW10]. The here presented

approach will attempt to provide a highly performant variant of polygon-rendering, deploying a precomputed lookup-table and a mesh subdivision procedure with the purpose of achieving frame rates enabling interaction in real-time.

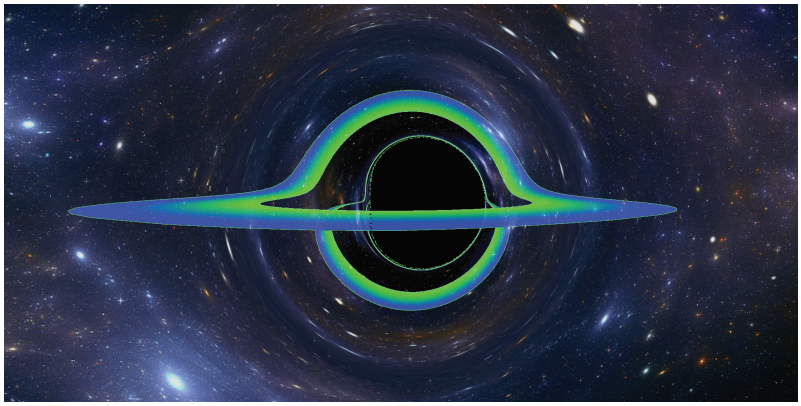

Figure 2.7: Visualization of an accretion disk around a Kerr black hole from [VE20]

3 Outlining the new Method

The method is split into two parts:

- Computation of a two-dimensional lookup-table which stores light travel times and viewing angles for a set of points in a plane containing the observer and the black hole itself,
- using said lookup-table to apply the distortions to a mesh of vertices and generate a visualization

3.1 Functionality

3.1.1 Calculating the lookup-table

To calculate this table, we specify multiple parameters: the observer's distance to the black hole (in multiples of its Schwarzschild radius), a maximum radius for the sample values (we sample points between the event horizon and the maximum radius), resolutions for angular and distance sampling (specifying how many sample points we generate in each direction) and a step size for numerical integration of the equations 2.4 -2.7. We then iterate through the different angular and radial positions specified (restricting angles to $[0, \pi]$ due to symmetry), calculate viewing distance and angles, and store them in pairs in a table, such that each point corresponds to 4 values (2 angles and 2 distances) (See 3.1). The relevant parameters and calculated values are then written to a file for further usage.

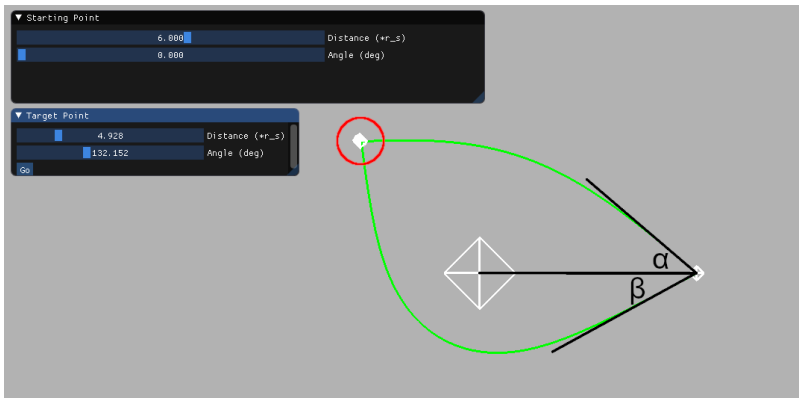

Figure 3.1: The table contains the angles α, β in which the target point (circled in red) appears for the observer on the right, as well as the light travel times along the green paths.

3.1.2 Table visualizations

In this section, we look at some visual representations of the data stored in the lookup-textures.

Figure 3.2: Lookup-texture of viewing angles for obsDist = $5r_s$, maxDist = $20r_s$

The horizontal coordinates in Fig 3.2 correspond to radial distances from the black hole, the vertical components to the angle in the plane containing black hole, observer and vertex. In this visualization, a viewing angle of 0° is colored black, an angle of 180° is colored white. The black hole in this case 'sits' at the left edge of the texture (where the radial component is $1r_s$) and the observer's position corresponds to the point in the bottom left where, going from left to right, the pixels change color from black to white (the viewing angle 'flips' here as the vertex moves past the observer away from the black hole). Another interesting aspect would be contour lines of these textures as seen in Fig. 3.3: These lines correspond to all the points that end up in the same apparent stellar position for the observer.

We can do the same thing with the texture component storing the viewing distance (Fig. 3.4): here the viewing distance increases (pixels turn from black to white) as we move away from the observer at the bottom edge of the texture. Contour lines of this texture correspond to points that appear in the same distance to the observer (Fig. 3.5)

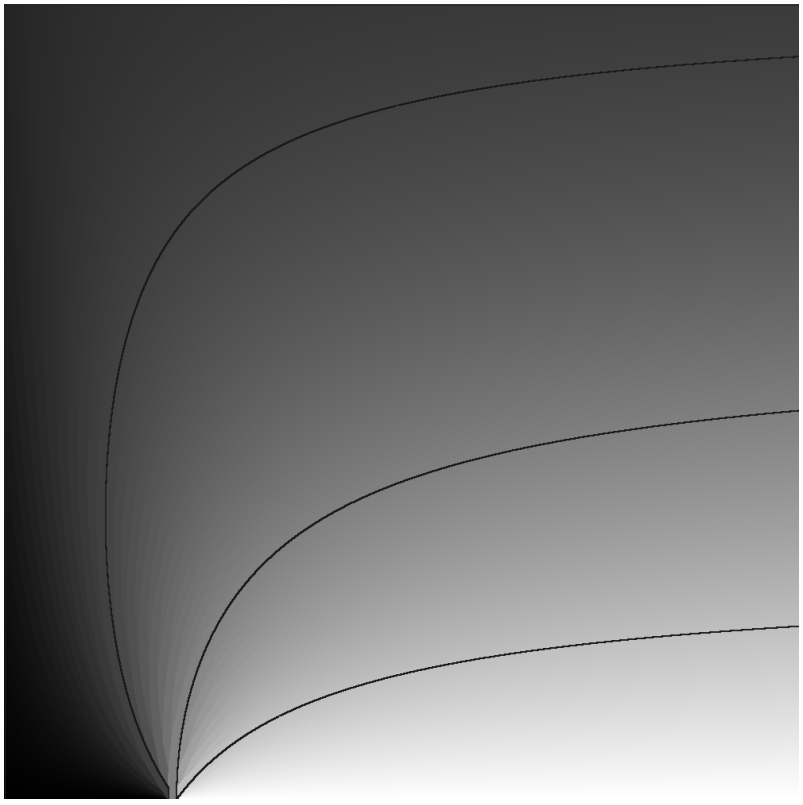

Figure 3.3: Contour lines for the angles 45°, 90°, 135° (left to right)

3.1.3 Utilizing the lookup-table for visualization

Using the precomputed table, we can calculate and apply an offset vector for each vertex, and calculate their apparent positions in this way. To do this, we need to compute each vertex's distance to the black hole, and its angular coordinate inside the plane containing the black hole, the observer and the vertex itself, and use those to lookup the appropriate viewing angle and distance in the previously computed table.

3.1.4 Subdivision

So far, this approach has the very obvious drawback of very bad visual results in case of a low triangle count in the model being processed. To solve this problem, we subdivide the mesh where its distortion is "high". Where the distortion is low, for example when far away from the black hole in the observer's field of view, no subdivision is needed, as straight edges also appear as (almost) straight lines after distortion. The level of subdivision scales increasingly with the amount of distortion applied to a triangle's edge. (See 3.7)

Figure 3.4: Lookup-texture of viewing distances for obsDist = $5r_s$, maxDist = $20r_s$

3.2 Implementation

The methods described are implemented in C++ using the OpenGL graphics specification.

3.2.1 Table calculation

The algorithms used to calculate the viewing angles and distances are outlined in 3.1 and 3.2. We deploy a kind of 'shooting method' starting out with rays at angles 0 and π (pointing straight at the center at the black hole and straight away from it), then iteratively calculating a new ray in the middle between those rays until we find a ray that comes sufficiently close to our target vector. The $calculateRay$-function (numerically) solves a (first-order) variation of the equations 2.4 - 2.7 with the classical fourth-order Runge-Kutta method (Chapter 235 in [But03]). We obtain this variation by introducing new variables for our first-order derivatives in these equations and substituting accordingly (which introduces additional initial conditions):

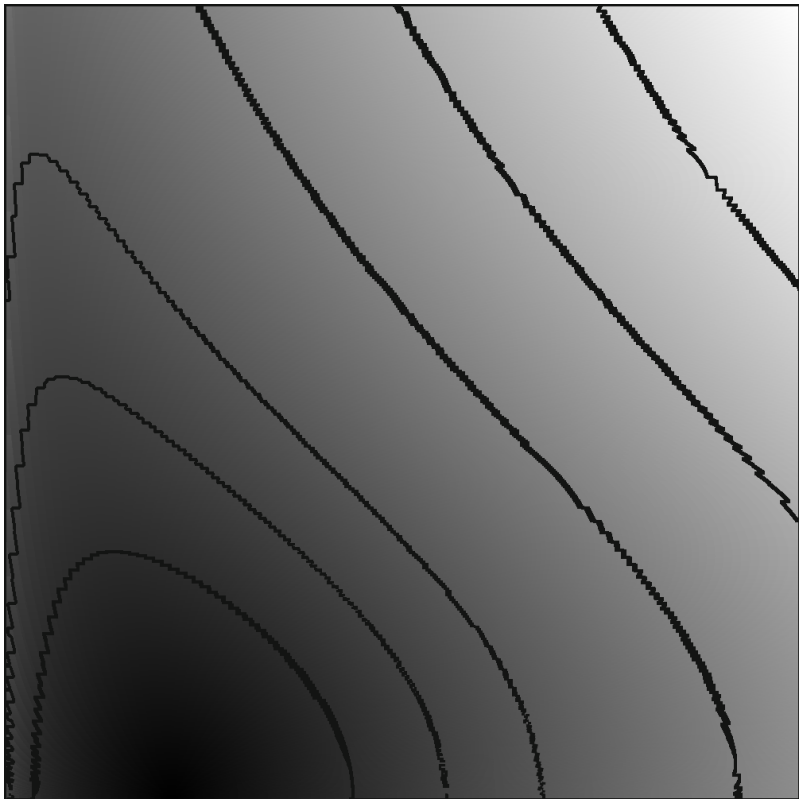

Figure 3.5: Contour lines for viewing distance

$$
\begin{aligned}
r_1 &:= r \\
r_2 &:= \frac{dr}{d\lambda} \\
t_1 &:= t \\
t_2 &:= \frac{dt}{d\lambda} \\
\theta_1 &:= \theta \\
\theta_2 &:= \frac{d\theta}{d\lambda} \\
\phi_1 &:= \phi \\
\phi_2 &:= \frac{d\phi}{d\lambda}
\end{aligned}
$$

These substitutions lead to the following system of equations:

$$\frac{dt_2}{d\lambda} = -\frac{r_s}{r_1 - r_s} r_2 t_2 \tag{3.1}$$

$$\frac{dr_2}{d\lambda} = \frac{r_s}{2r_1(r_1 - r_s)}(r_2)^2 - \frac{r_s}{2(r_1)^3}(r_1 - r_s)(t_2)^2 + (r_1 - r_s)[(\theta_2)^2 + \sin^2(\theta_1)(\phi_2)^2] \tag{3.2}$$

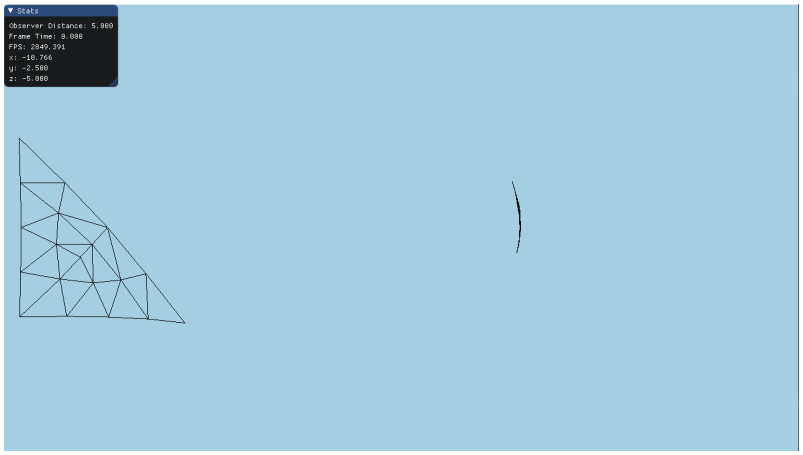

Figure 3.6: The triangle mesh is far away from the black hole, no subdivision is applied to the left instance

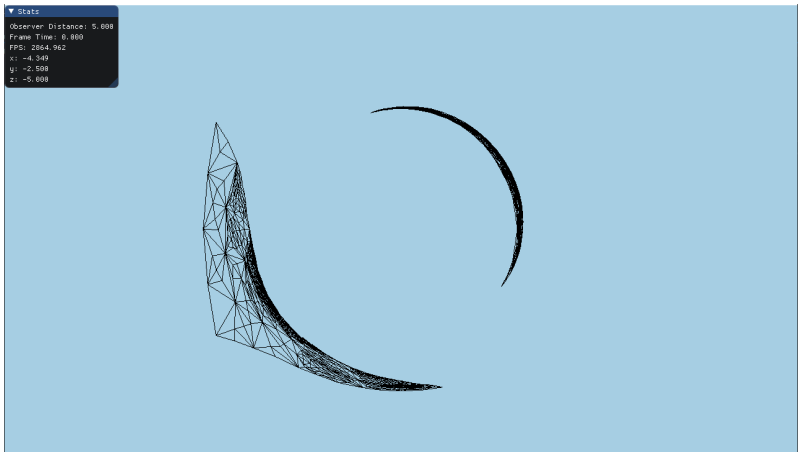

Figure 3.7: As the triangles in the mesh approach the black hole, the level of subdivision increases to accomodate the increasing amounts of distortion

$$\frac{d\theta_2}{d\lambda} = \sin\theta_1 \cos\theta_2 (\phi_2)^2 - \frac{2}{r_1}\theta_2 r_2 \tag{3.3}$$

$$\frac{d\phi_2}{d\lambda} = -2\frac{\cos\theta_1}{\sin\theta_1}\theta_2\phi_2 - \frac{2}{r_1}\phi_2 r_2 \tag{3.4}$$

$$\frac{dt_1}{d\lambda} = t_2 \tag{3.5}$$

$$\frac{dr_1}{d\lambda} = r_2 \tag{3.6}$$

$$\frac{d\theta_1}{d\lambda} = \theta_2 \tag{3.7}$$

$$\frac{d\theta_1}{d\lambda} = \theta_2 \tag{3.8}$$

With the aforementioned initial conditions for $t_{1/2}, r_{1/2}, \theta_{1/2}, \phi_{1/2}$. In our case, the initial conditions for the position are the coordinates of the observer:

$$(t_1(0), r_1(0), \theta_1(0), \phi_1(0)) = (0, obsDist, \frac{\pi}{2}, 0) \tag{3.9}$$

Some optimizations are made here: the $calculateRay$-method only continues integrating until it the ray passes the $target$-vector's angular coordinates, since we are only interested in the 'first-order' images , not those involving multiple rotations of the ray around the black hole. This also means that the method we use to check if the target is sufficiently close to a ray is only required to check the end of the ray (since this is where the closest points would be). This speeds up computation quite a bit and makes it less dependent on step size (since the amount of points calculated in a ray does not impact the runtime of the method used to check this).

The generated table is then stored to disk together with the relevant parameters:

angularRes	distanceRes	observerDistance	maxDistance	results[][]
int	int	float	float	4 · distanceRes · angularRes · float

the results[][] -vector (r) is stored as follows:

r[0][0]	r[0][1]	...	r[0][distanceRes-1]	r[1][0]	...	r[angularRes-1][distanceRes-1]

where every entry consists of four numbers with type $float$.

Algorithm 3.1 Table generation

procedure TABLE($stepSize, observerDistance, maxDistance, distanceRes, angularRes$)
  $startVec \leftarrow (0, observerDistance, \frac{\pi}{2}, 0)^\top$
  for $d = 0; d < distanceRes; d++$ **do**
    $currentDistance \leftarrow 1 + d \cdot \frac{maxDistance-1}{distanceRes-1}$
    for $a = 0; a < angularRes; a++$ **do**
      $currentAngle \leftarrow a \cdot \frac{\pi}{angleRes}$
      $targetVec \leftarrow (0, currentDistance, \frac{\pi}{2}, currentAngle)^\top$
      $dummyTarget \leftarrow (0, currentDistance, \frac{pi}{2}, 2\pi - currentAngle)^\top$
                                  // $dummyTarget$ used to streamline calculations
      $results[a][d][0] \leftarrow startToTarget(startVec, targetVec)$
      $results[a][d][1] \leftarrow startToTarget(startVec, dummyTarget)$
    end for
  end for
  return results
end procedure

Algorithm 3.2 Calculating Light Path between points

procedure STARTTOTARGET($start, target$)
  $minAngle \leftarrow 0, maxAngle \leftarrow \pi$
  while true **do**
    $direction_1 \leftarrow (1, -\cos(minAngle), 0, \sin(minAngle)^\top$
    $direction_2 \leftarrow (1, -\cos(maxAngle), 0, \sin(maxAngle))^\top$
    $ray_1 \leftarrow calculateRay(start, direction_1, target)$
    $ray_2 \leftarrow calculateRay(start, direction_2, target)$
    if ray_1 or ray_2 contain $target$ **then**
      return time component and angle of ray containing $target$
    end if
    $newAngle \leftarrow \frac{minAngle+maxAngle}{2}$
    $direction_{new} \leftarrow (1, -\cos(newAngle), 0, \sin(newAngle))^\top$
    $ray_{new} \leftarrow calculateRay(start, direction_{new}, target)$
    if ray_{new} contains $target$ **then**
      return time component of ray_{new} and $newAngle$
    end if
    if target is between ray_1 and ray_{new} **then**
      $maxAngle \leftarrow newAngle$
    end if
    if target is between ray_{new} and ray_2 **then**
      $minAngle \leftarrow newAngle$
    end if
  end while
end procedure

3.2.2 Visualization

The entire visualization process is done in the shader pipeline (except for the usual setup and preprocessing like camera parameters etc.). The generated lookup-table from the previous segment is read as a 2D-Texture and provided to the shaders as a sampler object. Objects are loaded with the tinygltf library[1] and then rendered twice with a $glDrawElementsInstanced$-call, once for each 'direction' a vertex is visible in. The observer can freely rotate his camera and move loaded objects around in real time. Each stage's role in the pipeline will be laid out here:

Vertex Shader

The Vertex shader transforms the vertices to world coordinates by multiplying the vertex coordinates with the model matrix.

[1] https://github.com/syoyo/tinygltf

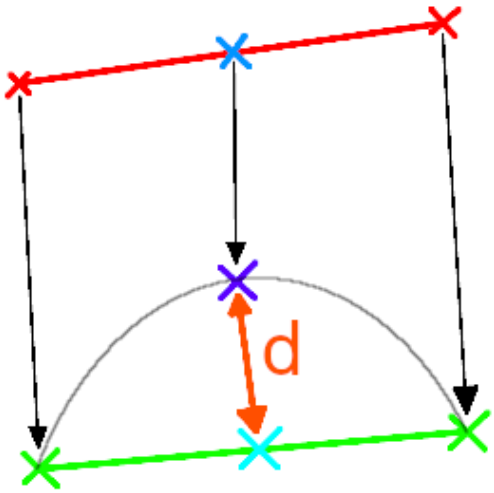

Figure 3.8: Determining the distortion of an edge (the grey arc indicates a possible 'true' image of the edge)

Tessellation Control Shader (TCS)

The TCS is responsible for determining the amount of distortion an edge experiences and setting its tessellation levels accordingly. This is done by computing the distorted coordinates in clip space of both vertices of said edge, as well as their midpoint's image. Then, we compare this image of the midpoint with the midpoint of the vertex images to determine how accurate the image would be without any further tessellation (see 3.8). The distance d from here is plugged into a function to 'squeeze' it into the interaval $[0, 1]$ to determine the level of tessellation (with a maximum of 64.0 in our case). Possible functions for this are 3.10 and 3.11. Inner tessellation level is set to the maximum of outer tessellation levels for a primitive.

$$\text{tessellationLevel} = 64.0 \cdot \left(-e^{-k \cdot d} + 1\right) \tag{3.10}$$

$$\text{tessellationLevel} = 64.0 \cdot \left(-\frac{1}{k \cdot d + 1} + 1\right) \tag{3.11}$$

Where $k > 0$ is a parameter determining the sensitivity of the tessellation level (in our visualizations, k=4 is used). The transformation to clipping coordinates is done do avoid unnecessary high tessellation levels where edges are barely visible (If we applied the same process to world coordinates instead, we might produce very high tessellation levels for edges that point away from the observer, appearing as a point, or only very short compared to their 'true' length). Note that the TCS does not actually pass on any transformed vertices, it only transforms them for the purpose of determining tessellation levels.

Tessellation Evaluation Shader (TES)

The TES now processes the generated vertices by applying the offset vectors stored in the lookup-texture provided to the pipeline: First, we need to calculate the texture coordinates. As mentioned in 3.2.1, the coordinates consist of the distance to the black hole and the angle between the black hole/vertex vector and the black hole/observer-vector. These are calculated as follows:

$$\text{distance} = \sqrt{vertexPos.x^2 + vertexPos.y^2 + vertexPos.z^2} \tag{3.12}$$

$$\text{angle} = \arccos\left(\frac{vertexPos \cdot obsPos}{\text{distance} \cdot obsDist}\right) \tag{3.13}$$

Where $vertexPos$ is the vertex position in world coordinates and $obsPos/obsDist$ are the observer position/distance respectively.

Using these coordinates, we obtain a set of two offset-parameters (angle and distance). Depending on the drawing instance we are in (as checked with $gl_Instance ID$ which has been passed through the pipeline from the vertex shader), we apply one of the other. To apply the offset in the plane containing the obsever, the original vertex and the black hole, we rotate a normalized and inverted version of the $obsPos$-vector around a normal vector of this plane, then scale it with the distance from the lookup-texture and add it to the original $obsPos$-vector (See Figure 3.9) The rotated vector is calculated by applying Rodrigues' rotation formula (Equation 4.16 in [Kok06]):

$$v_{rot} = v \cdot \cos(\alpha) + (\hat{n} \times -\widehat{obsPos}) \sin(\alpha) + \hat{n}(\hat{n} \cdot -\widehat{obsPos})(1 - \cos(\alpha)) \tag{3.14}$$

Where $\widehat{obsPos}$ is the normalized location vector of the observer, α is the viewing angle extracted from the lookup-texture and $\hat{n}$ is the normalized normal to the plane we want to rotate in:

$$n = \frac{obsPos \times vertexPos}{||obsPos \times vertexPos||} \tag{3.15}$$

Geometry Shader

Now that we know the proper final positions for the vertices, one problem remains: If the mesh is rendered as is, some triangles behind the black hole will be stretched out immensely to cover the entire black hole, and if they are rendered visible as a face, will obstruct vision and produce unpleasant artifacts (see Figure 3.10). To combat this, TES passes the 'original' vertex position along with the 'distorted' position, and the Geometry Shader discards triangles that have blown up in size too much during their distortion (indicating that they are 'stretching' over the black hole). This produces a more appropriate result (Figure 3.11)

Fragment Shader

The Fragment Shader has the sole purpose of applying color to the vertices according to the texture coordinates passed along (and interpolated in the TES).

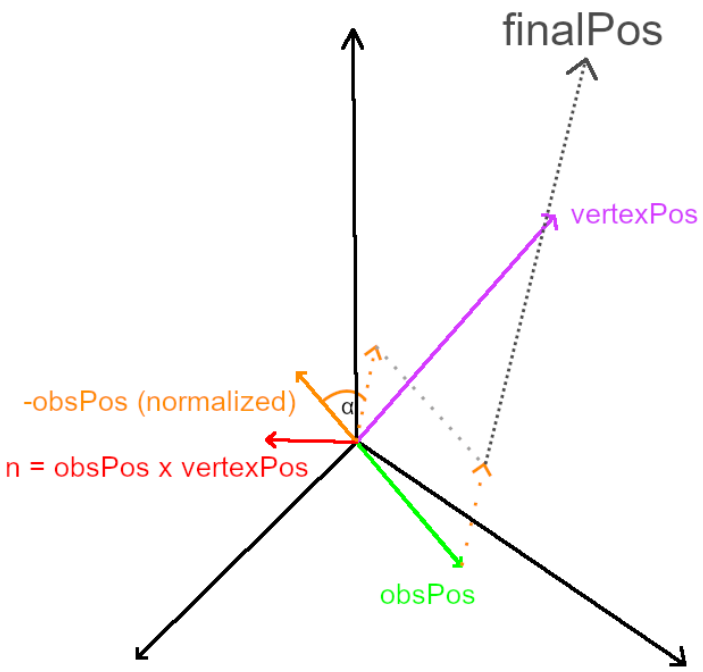

Figure 3.9: Calculation of the apparent position of a vertex

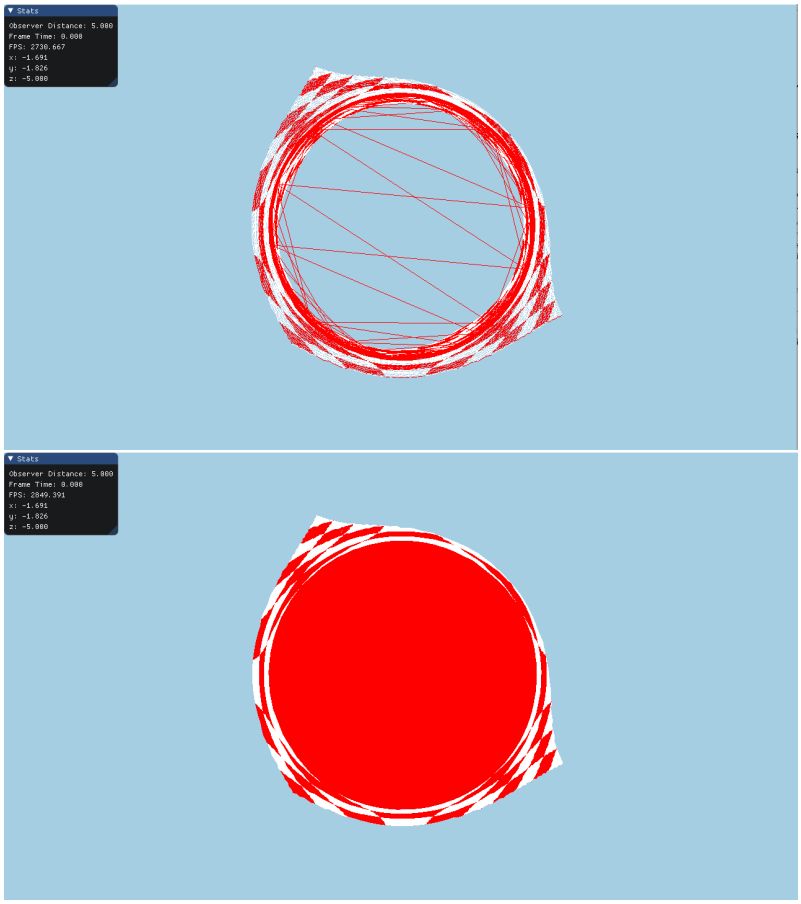

Figure 3.10: The triangle mesh (top) produces undesirable visual results when filling faces (bottom)

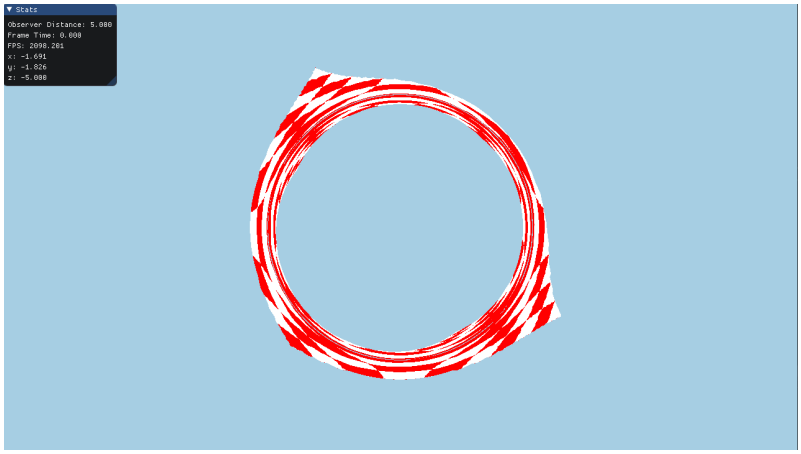

Figure 3.11: Heavily distorted triangles are discarded

4 Discussion

4.1 Visual Quality

We create visualizations of different objects (An overview is provided in Appendix A) with different lookup-textures: Fig. 4.1 shows a comparison between the raytracing visualization by GeoVis [Mül14] and our own. Lowering the resolution of the original mesh significantly decreases visual quality (See 4.2 and 4.3: while the outline of the triangle stays roughly the same, the texture quality close to the black hole degrades significantly). If the original mesh only has very few or very big triangles, the result can even be almost unusable (Fig. 4.5) Decreasing the resolution of the underlying lookup-texture does not seem to affect visual quality as much (See 4.4), but seems to lead to significant gaps between the first and second image of the triangle (too many triangles are discarded in the Geometry Shader).

Additionally, it seems like just increasing the resolution of the lookup-texture can possibly decrease the visual quality of results (See Fig. 4.9). Fixing this requires a more strict version of the method determining whether or not a ray contains the target in algorithm 3.2 (and accordingly a smaller step size for the integration process). Fig. 4.10 compares two such lookup-textures, both with the same resolution, but the texture used on the right was calculated with a much less forgiving method of determining whether a ray is 'good enough' or not (Note the 'staircasing' effect at triangle edges on the left and the much smoother edges on the right).

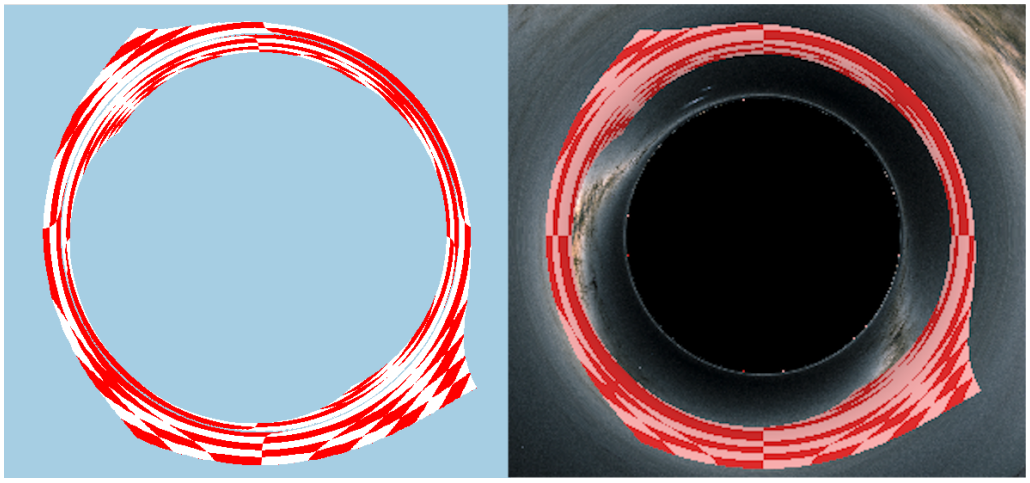

Figure 4.1: Triangle 5 behind black hole, our method (left) vs GeoVis (right). (Observer Distance: $5r_s$, maxDist = 20, lookup-texture resolution: 1000x1000)

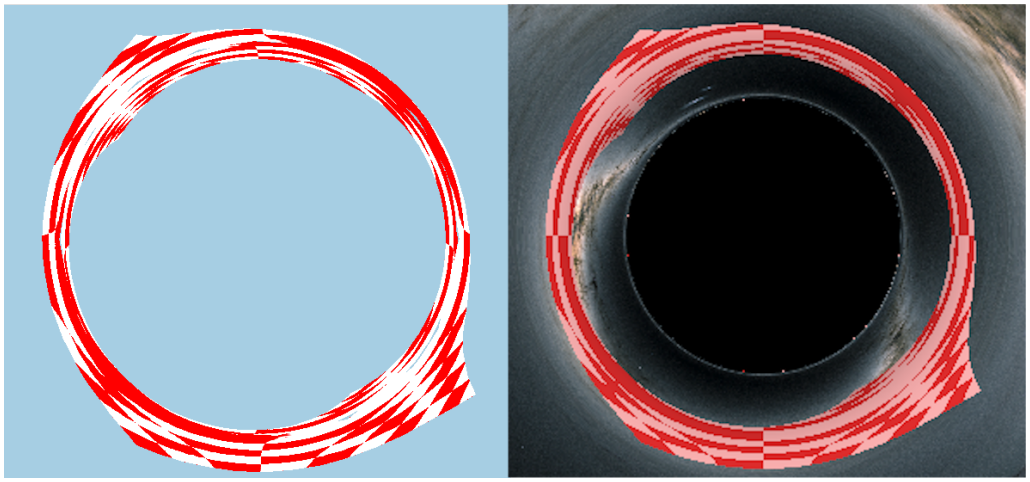

Figure 4.2: Triangle 3 behind black hole, our method (left) vs GeoVis (right). (Observer Distance: $5r_s$, maxDist = 20, lookup-texture resolution: 1000x1000)

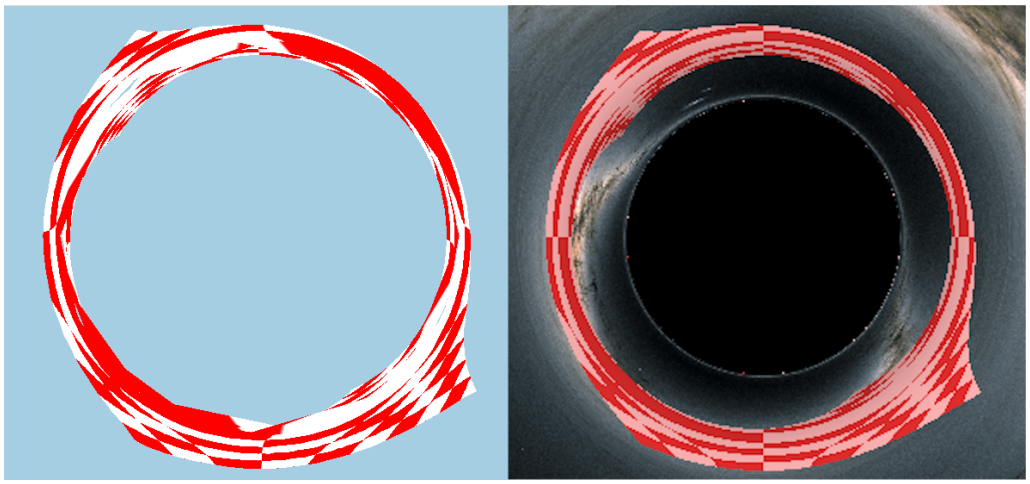

Figure 4.3: Triangle 2 behind black hole, our method (left) vs GeoVis (right). (Observer Distance: $5r_s$, maxDist = 20, lookup-texture resolution: 1000x1000)

4.2 Benchmarks

All renders are done on a desktop computer with the specs shown in 4.1 in a 1280×720 viewport.

GPU	AMD Radeon RX 5700 XT
CPU	AMD Ryzen 5 3600 6 Cores@3,60 GHz
RAM	16 GB

Table 4.1: Hardware used

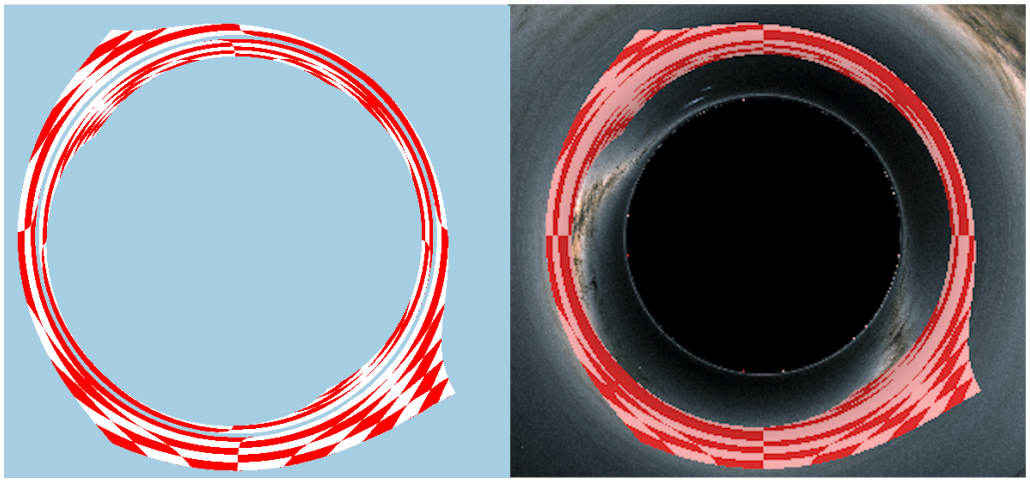

Figure 4.4: Triangle 5 behind black hole, our method (left) vs GeoVis (right). (Observer Distance: $5r_s$, maxDist = 10, lookup-texture resolution: 100x100)

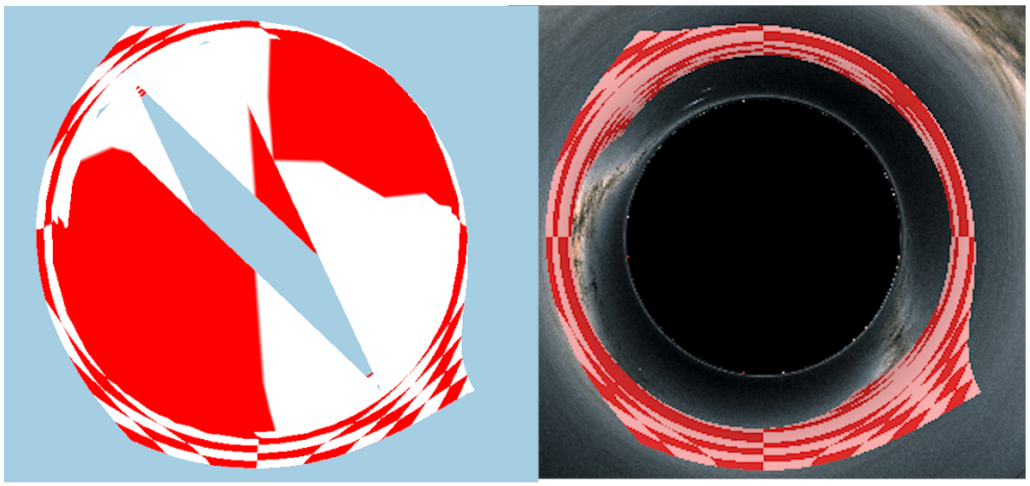

Figure 4.5: Triangle 1 behind black hole, our method (left) vs GeoVis (right). (Observer Distance: $5r_s$, maxDist = 10, lookup-texture resolution: 100x100). Criterion for discarding triangles fails because of their big size in the original mesh.

4.2.1 Table generation benchmarks

As seen in table 4.2, it seems like the parameters affecting runtime the most are the angular and distance resolutions, followed by step size and ϵ. Combined with the observation made about diminishing returns of increased resolution implies that, when calculating a table, one should first decrease step size and ϵ before increasing resolution. Generating tables with a higher maxDist also takes more time than calculating a table with the same parameters except for a smaller maxDist due to the higher number of integration steps required for points in the outer parts of the sampled area.

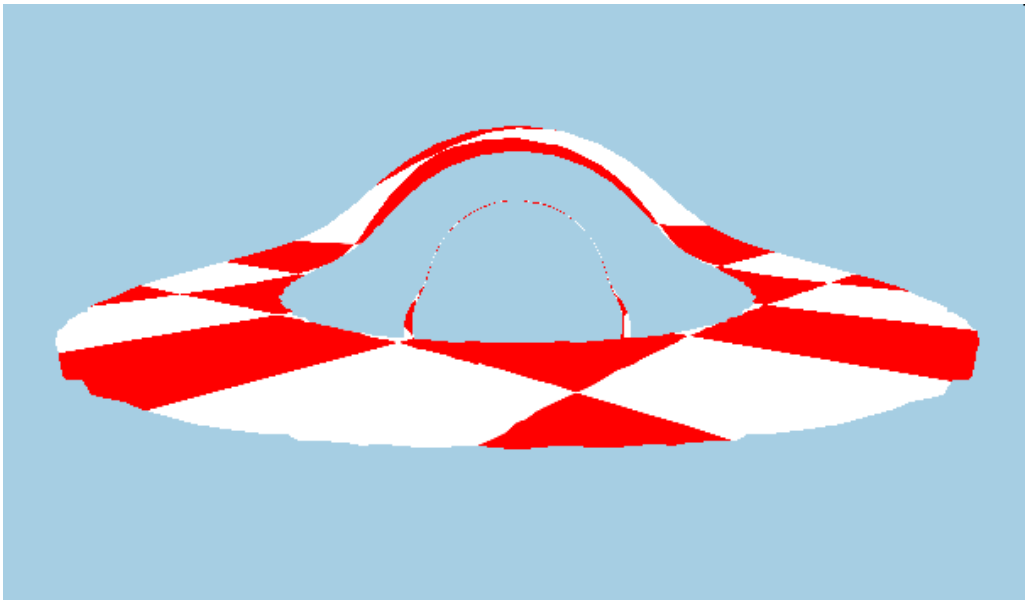

Figure 4.6: Accretion disk between $r_{min} = 6r_s$ and $r_{max} = 10r_s$ (Observer Distance: 15 r_s, maxDist = 30, lookup-texture resolution: 200x500 (angle × distance))

Step size	maxDist	distanceRes	angularRes	ϵ	Runtime
0.01	10	100	100	0.003	25 min
0.03	10	300	300	0.003	80 min
0.05	10	300	300	0.01	40 min
0.05	20	1000	1000	0.01	8 hours
0.05	10	50	50	0.01	70 sec

Table 4.2: Different runtimes for the table calculation algorithm (ϵ specifies the accuracy of the generated rays, smaller means more accurate)

4.2.2 Visualization benchmarks

The visualization itself is very fast with frametimes of ≤ 1ms for all figures shown except for Fig. 4.8, where the frametime fluctuates between 10-20ms (50-100 Frames per Second (FPS))depending on viewing direction. This is most likely caused by the amount of vertices and tessellation in the model of Sphere 2 (See A.8), since replacing it with Sphere 1 (A.7) reduces the frametime to 2ms (500 FPS)

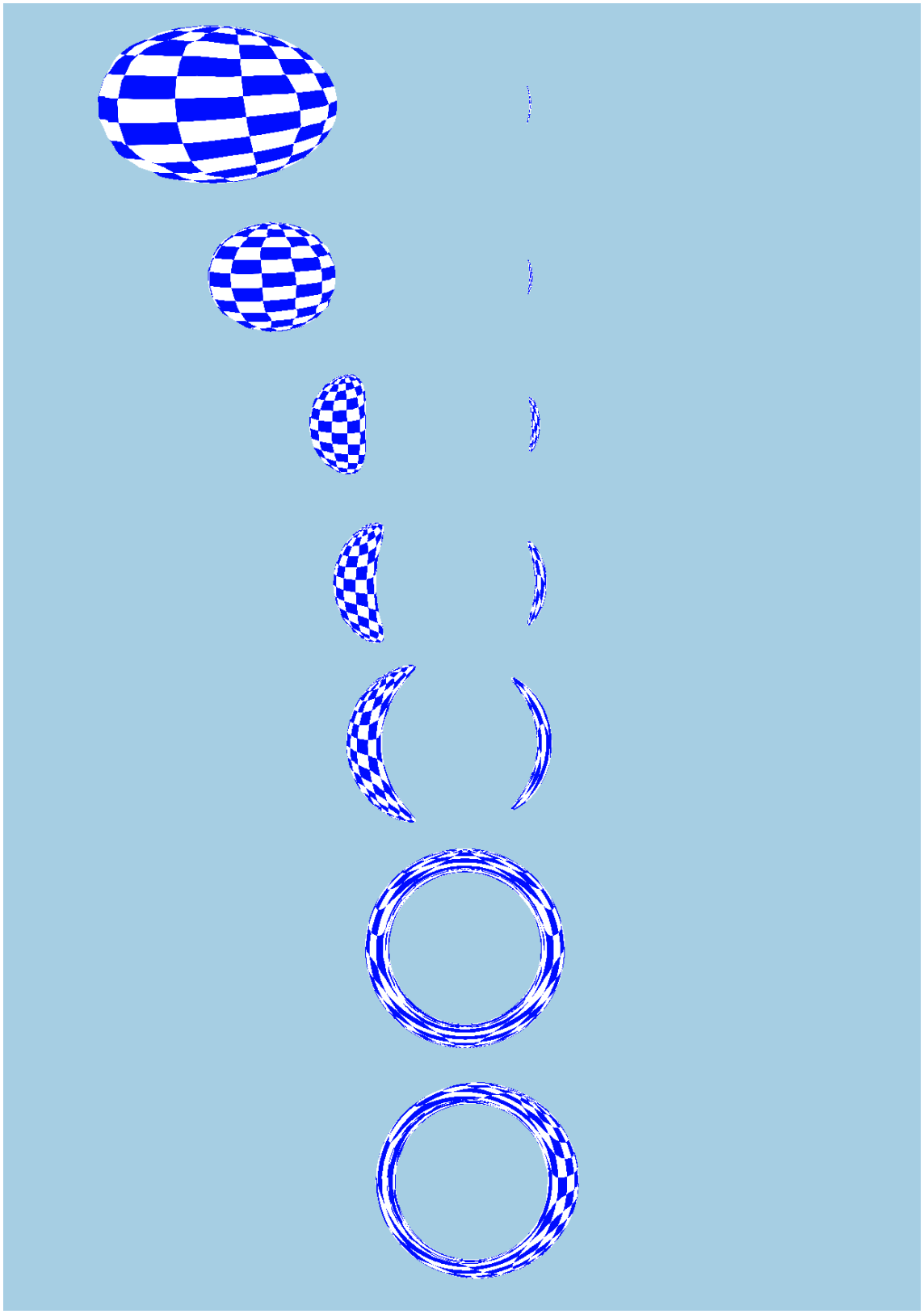

Figure 4.7: Sphere 2 with radius $2r_s$ on a circular trajectory around the black hole at angles 120°, 90°, 45°, 30°, 20°, 0°, -10° (top to bottom). (Observer Distance: $10r_s$, maxDist = 20, lookup-texture resolution: 100x100)

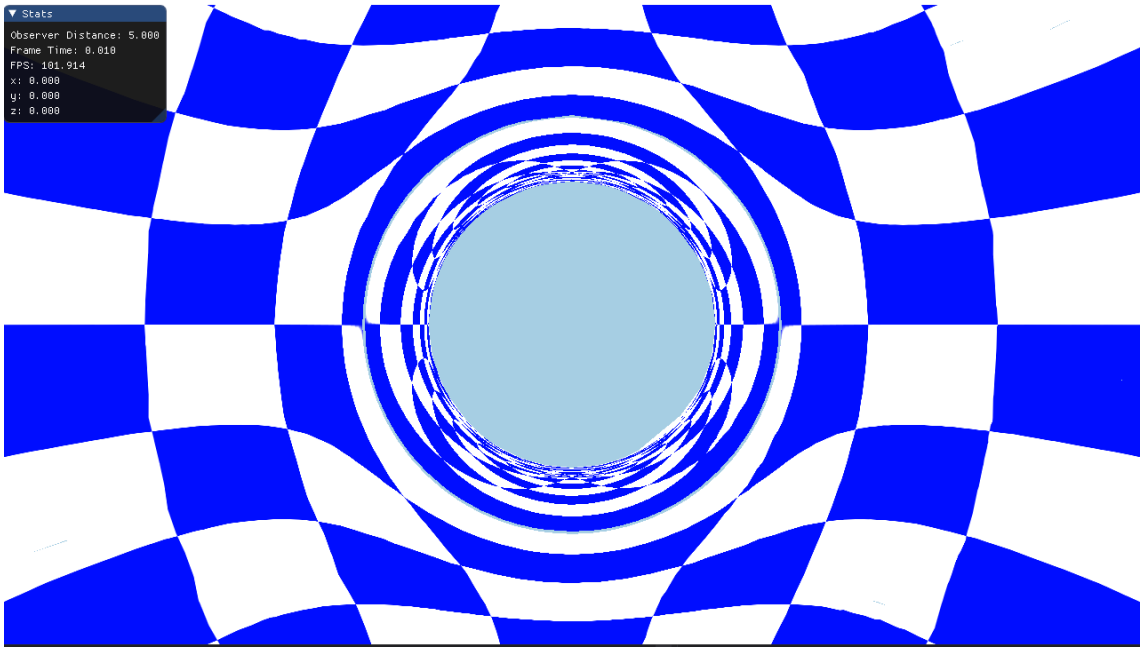

Figure 4.8: Sphere 2 with radius $8r_s$ with the black hole in its center. (Observer Distance: $5r_s$, maxDist = 20, lookup-texture resolution: 200x200)

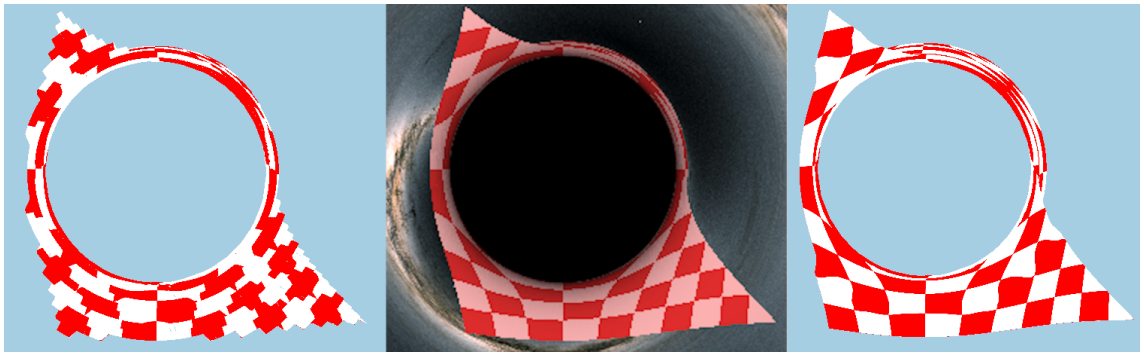

Figure 4.9: GeoVis-rendering of a scene (middle) and the same scene with a low-res lookup-texture (100x100, right) and a high-res texture (1000x1000, left)

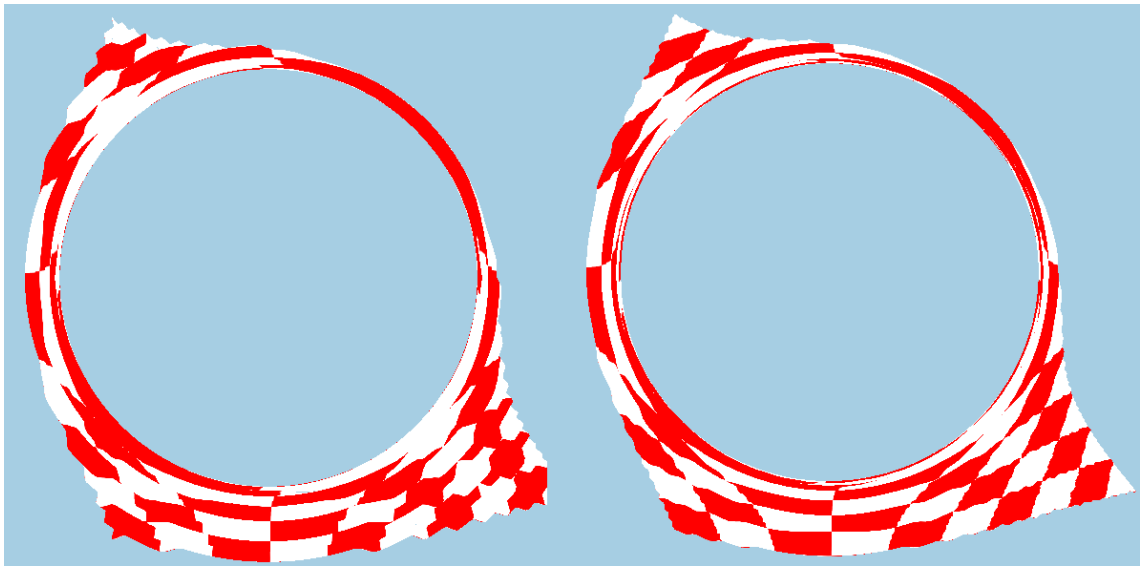

Figure 4.10: The same scene with textures of the same resolution, however a higher ray accuracy on the right side

5 Conclusion and Outlook

5.1 Conclusion

The presented method seems like a promising approach to general relativistic visualization in real time, since it separates expensive and time-consuming calculations from the interactive part of the visualization, enabling high framerates (compared to approaches like ray tracing). However, there are (currently) drawbacks to this method: It seems to be rather sensitive to parameters (integration step size, ray accuracy tolerance, original mesh size (especially when using models not intended for this kind of application without preprocessing them)). Additionally, using numerical methods brings with it inherent inaccuracies (although it makes transferring the model to different scenarios easier). The method 'as is' also still struggles with the merging of multiple images of the same object when it is behind the black hole. The boundary between the two images becomes very thin when using high-resolution high-accuracy lookup-textures in combination with high tessellation levels or fine (original) triangle meshes, but it is still visible. Another obvious limitation is the spacial one, as the lookup-tables are only generated in a certain (spherical) space region around the black hole. It should be possible to tackle most of these drawbacks with approaches outlined in the next section.

5.2 Improvements/Outlook

To improve the visual quality of images, one could subdivide any triangle mesh used even before forwarding it into the pipeline, avoiding big triangles causing effects like in Figure 4.5. This takes more computing power however and might drastically reduce framerates. Parameter sensitivity is a bit rougher to tackle, since it is not at all obvious which parameters lead to which inaccuracies/undesired results. A first step could be to determine how the integration step size (or estimated error if a method with variable step size is used) relates to the accuracy parameter ϵ (See also table 4.2). A good estimate of this would increase performance of the table calculation (by increasing step size) while maintaining accuracy (determined by ϵ). Another approach to table calculation could be a different sampling process - so far we have sampled uniformly in spherical coordinates, but other methods are possible, like sampling in cartesian coordinates, or sampling something like $1/r$ uniformly instead of r itself (causing more dense sampling close to the black hole, and less dense sampling far away from it - also helping to mitigate the spacial limitations by covering a bigger area with the same table size). For the Schwarzschild metric, there even exist analytical solutions to the geodesic equations 2.4 - 2.7, which would remove some of the numerical inaccuracies inherent to methods like Runge-Kutta. However, this would make it harder to transfer to different scenarios with possibly different metrics (and geodesics). Probably the biggest potential speedup can be achieved by parallelizing the calculation process. Similar to ray tracing, each iteration of the table calculation

is independent of the other ones, meaning it is possible to implement the precomputation in a highly parallelized way. This speedup would also allow more precision during ray calculation without having the pre-calculation span multiple days.

Bibliography

[BMW16] S. Boblest, T. Müller, G. Wunner. *Spezielle und allgemeine Relativitätstheorie: Grundlagen, Anwendungen in Astrophysik und Kosmologie sowie relativistische Visualisierung*. ger. Lehrbuch. OCLC: 930016813. Berlin Heidelberg: Springer Spektrum, 2016. ISBN: 9783662477663 9783662477670 (cit. on pp. 17–21).

[But03] J. Butcher. *Numerical Methods for Ordinary Differential Equations: Butcher/Numerical Methods*. Chichester, UK: John Wiley & Sons, Ltd, June 2003. ISBN: 9780470868270 9780471967583. DOI: 10.1002/0470868279. URL: http://doi.wiley.com/10.1002/0470868279 (visited on 11/22/2020) (cit. on p. 28).

[CPÖ13] C.-k. Chan, D. Psaltis, F. Özel. "GRay: A MASSIVELY PARALLEL GPU-BASED CODE FOR RAY TRACING IN RELATIVISTIC SPACETIMES". In: *The Astrophysical Journal* 777.1 (Oct. 2013), p. 13. DOI: 10.1088/0004-637x/777/1/13. URL: https://doi.org/10.1088%2F0004-637x%2F777%2F1%2F13 (cit. on p. 22).

[EA03] A. Einstein, E. P. Adams. *The meaning of relativity*. English. OCLC: 56109489. London; New York: Routledge, 2003. ISBN: 9780203449530 9781134449798 9781138171190 9781280031267 9781134449781. URL: http://public.eblib.com/choice/publicfullrecord.aspx?p=181874 (visited on 11/16/2020) (cit. on p. 19).

[Gol80] H. Goldstein. *Classical mechanics*. 2d ed. Addison-Wesley series in physics. Reading, Mass: Addison-Wesley Pub. Co, 1980. ISBN: 9780201029185 (cit. on p. 18).

[Kok06] D. Koks. *Explorations in mathematical physics: the concepts behind an elegant language*. OCLC: ocm68804198. New York: Springer, 2006. ISBN: 9780387309439 9780387327938 (cit. on p. 34).

[MGW10] T. Müller, S. Grottel, D. Weiskopf. "Special Relativistic Visualization by Local Ray Tracing". In: *IEEE Transactions on Visualization and Computer Graphics* 16.6 (Nov. 2010), pp. 1243–1250. ISSN: 1077-2626. DOI: 10.1109/TVCG.2010.196. URL: http://ieeexplore.ieee.org/document/5613464/ (visited on 11/16/2020) (cit. on p. 22).

[Mül14] T. Müller. "GeoViS—Relativistic ray tracing in four-dimensional spacetimes". In: *Computer Physics Communications* 185.8 (2014), pp. 2301–2308. ISSN: 0010-4655. DOI: https://doi.org/10.1016/j.cpc.2014.04.013. URL: http://www.sciencedirect.com/science/article/pii/S0010465514001362 (cit. on pp. 22, 37).

[MW10] T. Müller, D. Weiskopf. "Distortion of the stellar sky by a Schwarzschild black hole". en. In: *American Journal of Physics* 78.2 (Feb. 2010), pp. 204–214. ISSN: 0002-9505, 1943-2909. DOI: 10.1119/1.3258282. URL: http://aapt.scitation.org/doi/10.1119/1.3258282 (visited on 11/16/2020) (cit. on p. 22).

[VE20] A. Verbraeck, E. Eisemann. "Interactive Black-Hole Visualization". In: *IEEE Transactions on Visualization and Computer Graphics* (2020), pp. 1–1. ISSN: 1077-2626, 1941-0506, 2160-9306. DOI: 10.1109/TVCG.2020.3030452. URL: https://ieeexplore.ieee.org/document/9226126/ (visited on 11/16/2020) (cit. on pp. 22, 23).

A Models used

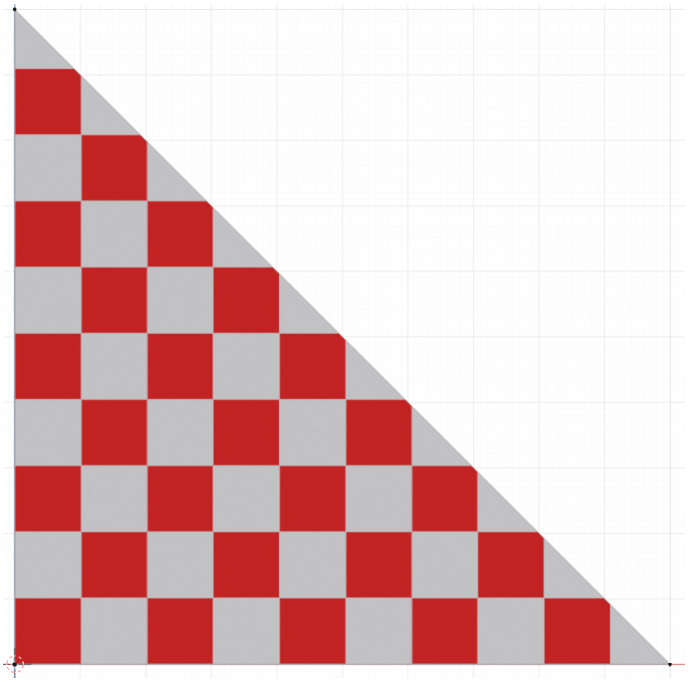

Figure A.1: Triangle 1, #triangles = 1

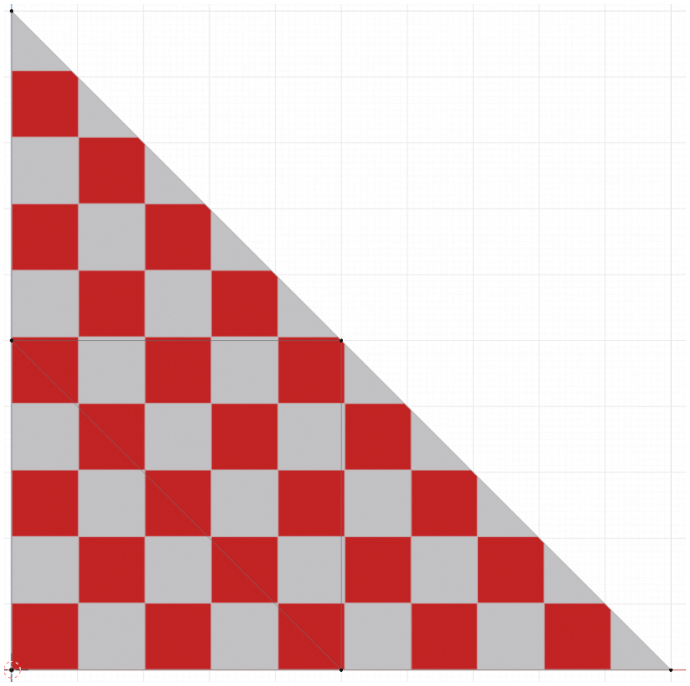

Figure A.2: Triangle 2, #triangles = 4

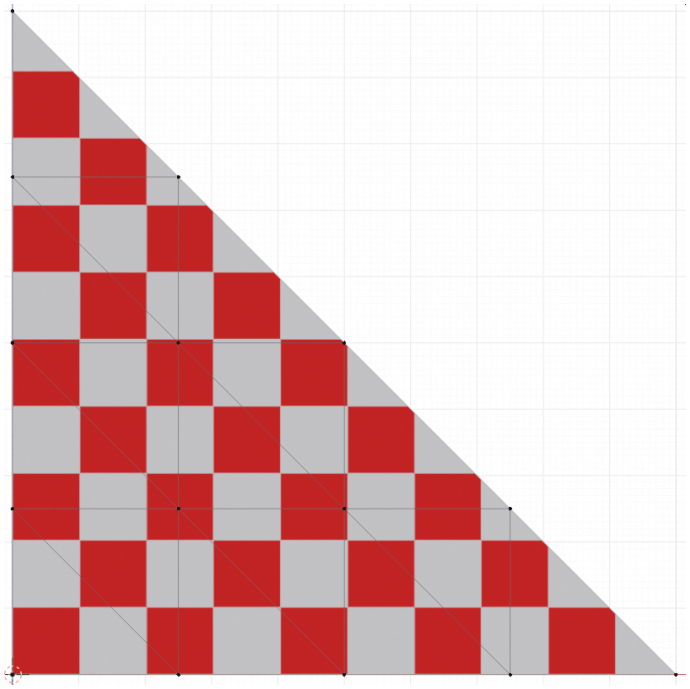

Figure A.3: Triangle 3, #triangles = 16

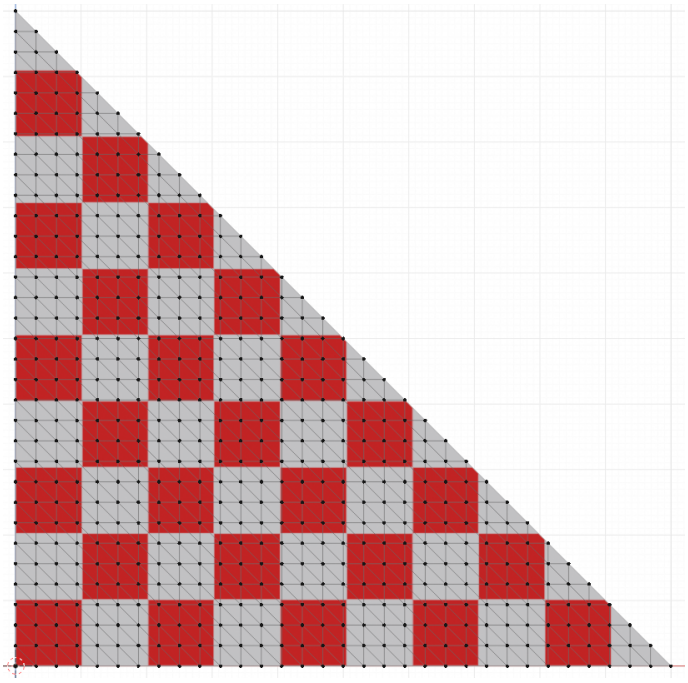

Figure A.4: Triangle 4, #triangles = 1024

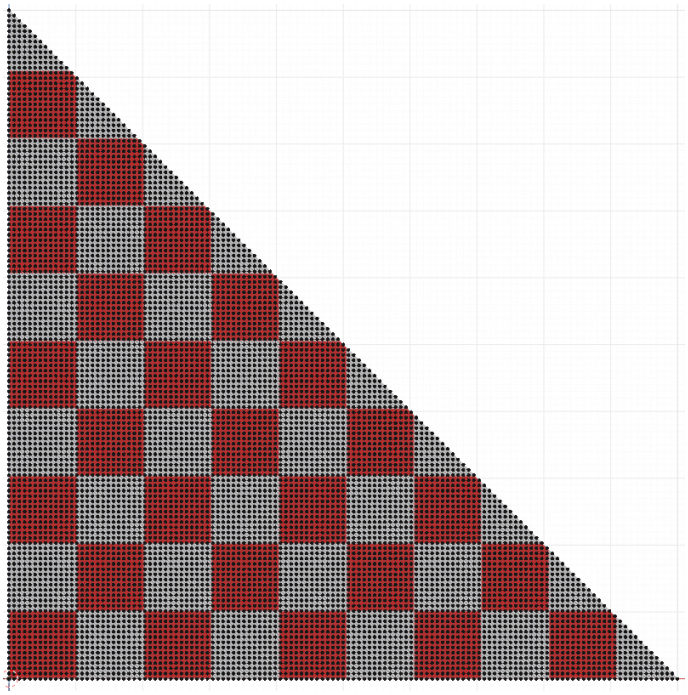

Figure A.5: Triangle 5, #triangles = 16384

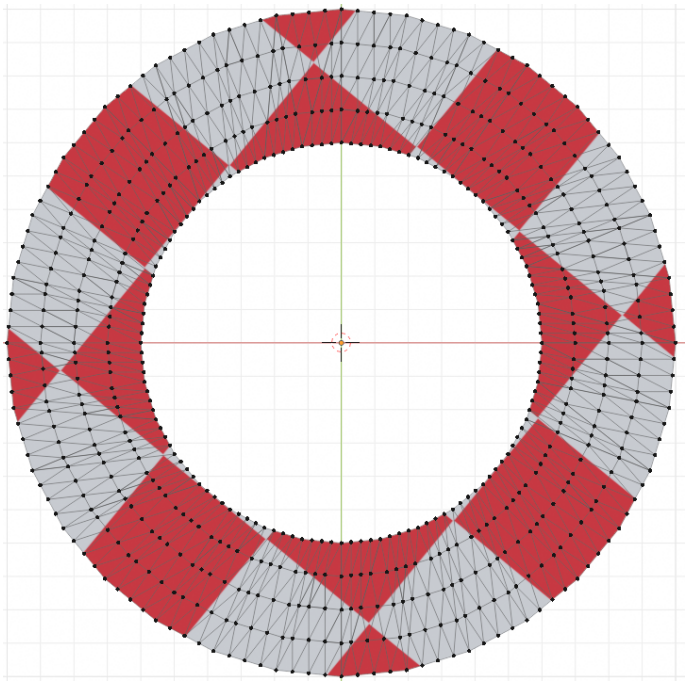

Figure A.6: Disk, #triangles = 1024

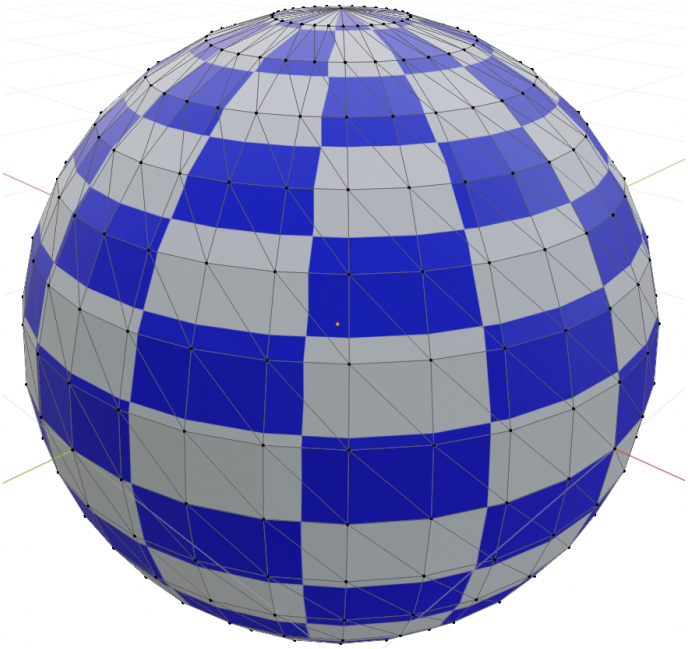

Figure A.7: Sphere 1, #triangles = 960

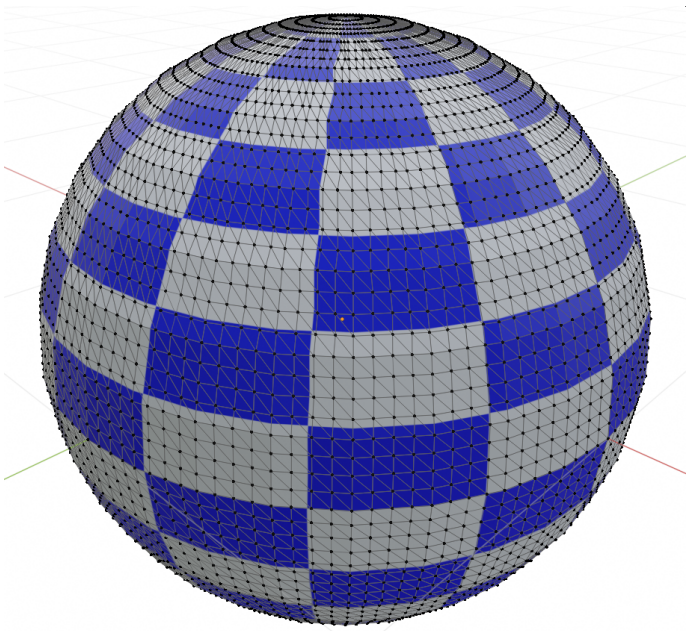

Figure A.8: Sphere 2, #triangles = 15360

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature