Development of a Plug-In to Visualise Effects Caused by the Bending of Light by Masses

(Rendering Black Holes in Maya)

Bachelor Thesis

Marcel Ritter, 9817001

20th April, 2005

Supervisor: Univ.-Prof. Dr. Aart Middeldorp

Institute of Computer Science

Leopold-Franzens-Universität Innsbruck

Abstract

Until now tools to visualise effects caused by the bending of light as predicted by General Relativity are highly specialised software packages. In this thesis a different approach is presented: a plug-in for a standard visualisation software (Maya), which allows to simulate the bending of light. The implemented algorithm is based on an approximation, which speeds up the rendering process, drastically.

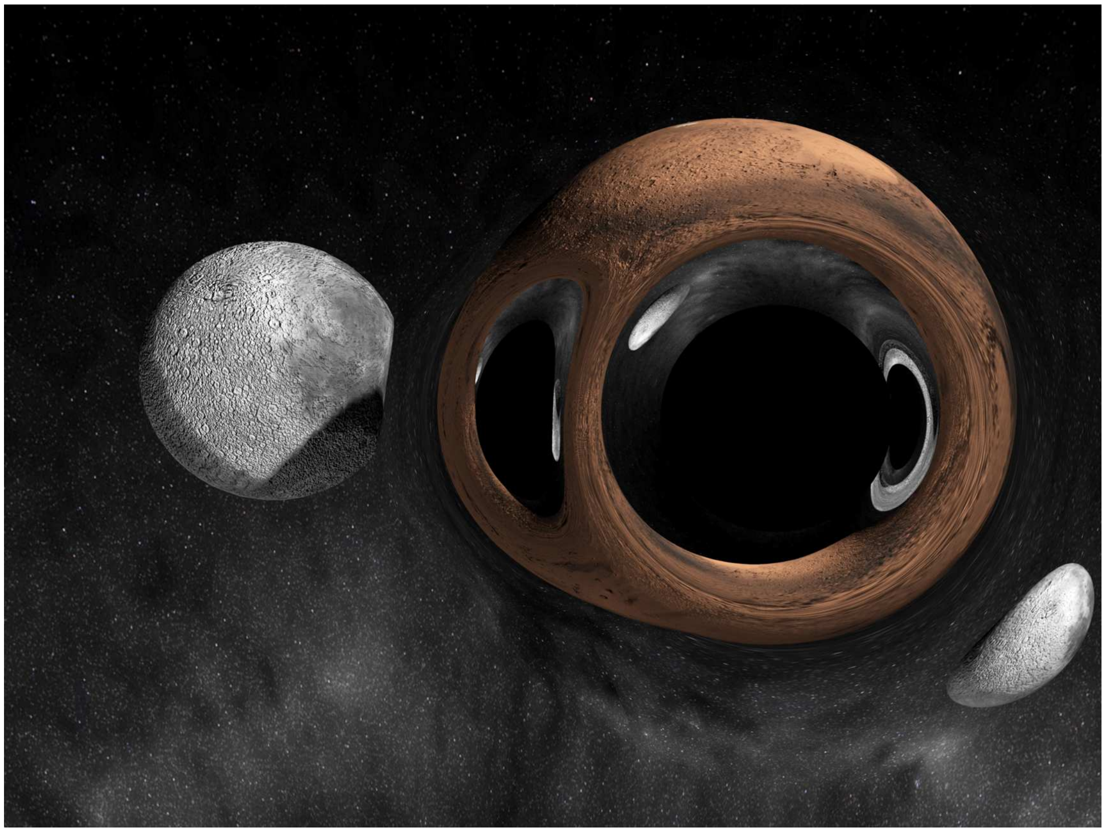


Figure 1: *Spacetime of Two Black Holes*

Contents

Chapter 1

Introduction

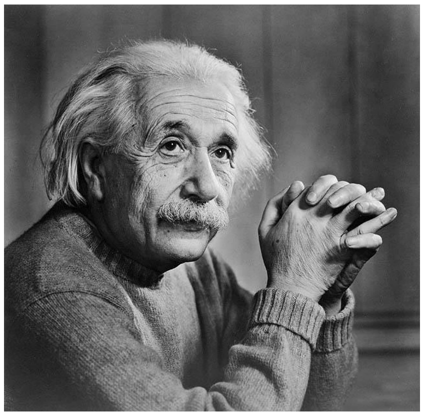


Figure 1.1: *Albert Einstein*

The year 2005 was celebrated as "The World Year of Physics" and was dedicated to Albert Einstein. Exactly 100 years ago, Albert Einstein wrote four scientific papers, that were published within only one year. They revolutionised the world of physics and how we view the universe.

The first paper in this year ("Über einen die Erzeugung und Verwandlung des Lichts betreffenden heuristischen Gesichtspunkt") explains the photoelectric effect and the behaviour of light which can be described as a stream of particles called photons as alternative to an electromagnetic wave. 1921 he was rewarded the Nobel Price "for his services to Theoretical Physics, and especially for his discovery of the law of the photoelectric effect" [Nob05].

The second paper ("Zur Elektrodynamik bewegter Körper") describes what is now known as Special Relativity.[1] In this paper Einstein welded together time and space to a combined four dimensional structure called spacetime. He realised that time is not an absolute quantity but a relative one, depending on the observer. It is important to add that his theory of Special Relativity was built upon work and thoughts of others.

Because of these contributions, the year 1905 is now known as the "Annus Mirabilis"[2] of Einstein.

[1]The other two papers are: "Über die von der molekularkinetischen Theorie der Wärme geforderte Bewegung von in ruhenden Flüssigkeiten suspendierten Teilchen" and "Ist die Trägheit eines Körpers von seinem Energieinhalt abhängig?"

[2]latin for "miraculous year"

Eleven years later Einstein published his work on General Relativity (see [Ein16]), a theory he developed solely by himself[3]. General Relativity describes how mass and spacetime are correlated. An important consequence is that light does not travel along straight lines, but on curved paths because it is bent by mass.

Figure 1.2: *Arthur Eddington*

Arthur Eddington was the first who proved one of Einstein's purely theoretical predictions. During a total eclipse in 1919 he took pictures of stars in the region around the sun when they were clearly visibly. He observed that the stars appeared to be shifted away from the sun's centre, when compared to their positions in the night sky. This effect and the amount of the shift was exactly predicted by Einstein's General Relativity. *Figure 1.3*[4] shows one of Eddington's photographs, taken on an island near Africa.

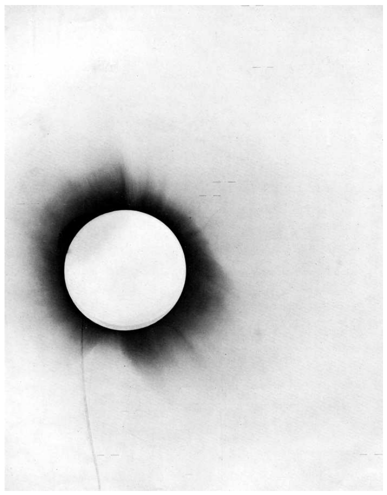


Figure 1.3: *The Eclipse 1919*

Nowadays modern computer graphics can be used to visualise these distortion effects in artificial environments. These tools have been developed by specialists mostly for scientific research and are not accessible to the general graphic artists or other end users.

In my thesis I present a technique to create pictures of 3D scenes incorporating the effect of bending of light by masses. I implemented the technique as a plug-in module for a commercial 3D-application, so that the simulation of curved space becomes more accessible.

The thesis is organised as follows: chapter 2 contains the theoretical background necessary for the following chapter 3, which describes the main work. The last chapter provides some thoughts about possible extensions and future work.

[3]He was supported by Marcel Grossmann on the mathematical aspects.

[4]Picture from [DED20]

Chapter 2

Theoretical Background

2.1 Bending of Light by Masses

2.1.1 General Relativity

An object that experiences acceleration due to a gravitational force in classical mechanics, is actually modelled as being unaccelerated in General Relativity: According to the principle of equivalence, gravity is treated as being undistinguishable from acceleration. Thus, all objects under influence of pure gravitation are moving along straight lines. In curved spacetime, a straight line is described by a *geodesic*, and it might be quite different from a straight line in flat, euclidean spacetime.

The geometrical properties of spacetime are described by a four dimensional metric tensor field, the four dimensions being three spatial coordinates and one temporal coordinate. The field is called metric because it defines the distance between two neighboring points[1] in spacetime. The curvature of spacetime is a tensor field as well and determined from the metric tensor field and its first and second derivatives. Another tensor field is used to describe quantities like energy and mass distribution in every point of the four dimensional spacetime.

Tensor fields provide a mathematically convenient way to integrate many properties and quantities in a unified formulation. They are the common formalism in the mathematics of differential geometry. Tensor calculus is a central concept of differential geometry, which allows to model what is necessary here.

Einstein developed his field equations that relate the distribution of matter to the curvature of spacetime. The Einstein field equations can be written in an elegant

[1]infinitesimal close

symbolic way:

$$G_{\mu\nu} = \kappa \, T_{\mu\nu} \tag{2.1}$$

The left hand side of the formula contains all information about the metric and curvature of spacetime. $G_{\mu\nu}$ is called Einstein tensor. On the right hand side of the equation the stress-energy tensor $T_{\mu\nu}$ contains all the information about location and motion of matter (mass, energy and momentum). κ is a constant scalar.

Although these equations do not look very complicated, it is by no means an easy task to solve them, because they are build upon ten coupled non-linear partial differential equations, which makes it difficult to deal with them mathematically.

Only a few analytical solutions are known. The first one was found by Karl Schwarzschild during the first world war. He made strong simplifications and found a solution for the special case of a perfectly spherical mass distribution at rest in vacuum. The solution is now known as the *Schwarzschildlösung* (see chapter 2.1.3). It contains a mathematical singularity: at this point the spacetime curvature gets infinite.

In case of non-rotational symmetries the differential equations get very complicated. At the full level of complexity, with no simplifications applied, equations with huge numbers of terms have to be solved. Today, simulations using numerical methods, like the finite difference method, can be run on high performance supercomputers to solve these partial differential equations.

For a more detailed introduction into General Relativity see [Sch03] and of course [Ein16].

2.1.2 Gravitational Lenses

As mentioned above, masses cause curvature of space. The path of a ray of light that passes a (heavy) object, e.g. a sun, will be diverted. If light is distorted by a mass located between the light source and an observer, the distortion effect seen by the observer is called a *gravitational lens.*

Figure 2.1 shows the principle of bending of light caused by a gravitational lens. On the right side is the observer (red), in the middle a mass (black), and on the left side is the observed object (blue). Light that is emitted from the object and reaches the observer gets bent by the mass on (blue and yellow lines). Now the observer, who is only familiar with flat space and straight rays of light, sees several images of one and the same object (grey). The images show the object from different perspectives and distances, because of the different paths the light takes (blue and

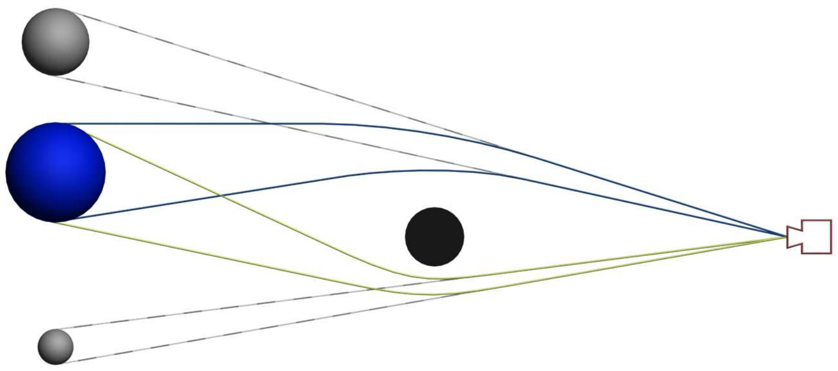


Figure 2.1: *Bending of Light by a Mass*

yellow lines). More pictures showing the optical effect from the observers point of view can be found in chapter 4.

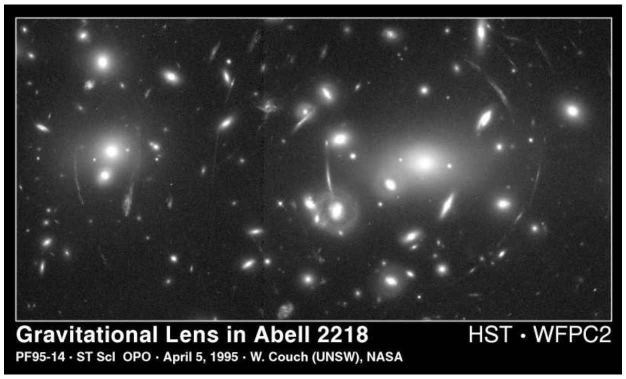

Gravitational Lens in Abell 2218
HST • WFPC2
PF95-14 • ST ScI OPO • April 5, 1995 • W. Couch (UNSW), NASA

Figure 2.2: *Example of a Gravitational Lens*

Gravitational lenses can be observed in nature looking for example at the galaxy cluster Abell 2218. *Figure 2.2* shows a picture of Abell 2218 taken by the Hubble Space Telescope. A gravitational lens bends the light from galaxies, which are located (far) behind this lens. These far galaxies cause the arcs seen in the picture. A gravitational lens also magnifies objects located behind it. This property can be used as a gravitational telescope to get pictures of galaxies very far away. Another application of gravitational lensing is the reconstruction of mass distributions in certain sectors from the amount of the measured gravitational lens effects.

In this thesis an approximation for the deflection of light by a "small but heavy" mass derived from the Schwarzschild solution is used to simulate the gravitational lens effect. Such a mass could be an astronomical object called a *black hole*. Gravitational force near a black hole is so strong, that a body in order to "escape" from the gravitational field, would have to be faster than the speed of light, which is impossible by Einstein's special relativity. This now also implies that light itself cannot escape the gravitational field caused by a black hole and so they appear to be perfectly black. The spherical boundary around the centre of a black hole, where its gravitational field gets strong enough to prevent light from escaping, is called *event horizon*. Everything that crosses this event horizon will never be able to escape the black hole.

2.1.3 Derivation of an Approximation

Starting from the Schwarzschild solution we are going to derive a simple approximation for the path of light:

$$ds^2 = \left(1 - \frac{2m}{r}\right) dt^2 - \frac{1}{1 - \frac{2m}{r}} dr^2 - r^2 d\Omega^2 \tag{2.2}$$

m ... mass of "lens"
with $\Omega^2 = d\vartheta^2 + sin^2\vartheta d\varphi^2$
W.l.o.g.: $\vartheta = \pi/2$, thus: $d\Omega = d\varphi$. The trajectory of a light-like geodesic (path of light) $\dot{q}(s) = \{\dot{t}, \dot{r}, \pi/2, \dot{\varphi}\}$ satisfies the following equation:

$$G(\dot{q}, \dot{q}) = 0 \tag{2.3}$$

where $G(\dot{q}, \dot{q})$ represents the Lagrange-function $\mathfrak{L}$ of ... (so q is an extremum):

$$\mathfrak{L} = \left(1 - \frac{2m}{r}\right) \dot{t}^2 - \frac{1}{1 - \frac{2m}{r}} \dot{r}^2 - r^2 \dot{\varphi}^2 \tag{2.4}$$

The following two laws of conservation are called conservation of "energy" and conservation of "angular momentum", in analogy to Newton's mechanics:

$$\frac{\partial \mathfrak{L}}{\partial t} = 0 = \frac{d}{ds} \frac{\partial \mathfrak{L}}{\partial \dot{t}} \quad \rightarrow \quad \frac{\partial \mathfrak{L}}{\partial \dot{t}} = const. \equiv 2 \left(1 - \frac{2m}{r}\right) \dot{t} =: 2E \tag{2.5}$$

$$\frac{\partial \mathfrak{L}}{\partial \varphi} = 0 = \frac{d}{ds} \frac{\partial \mathfrak{L}}{\partial \dot{\varphi}} \quad \rightarrow \quad \frac{\partial \mathfrak{L}}{\partial \dot{\varphi}} = const. \equiv -2r^2 \dot{\varphi} =: -2L \tag{2.6}$$

Inserting E, and taking the constraint for "light-like" into account ($G(\dot{q}, \dot{q}) = 0$) we get:

$$\frac{E^2}{1 - \frac{2m}{r}} - \frac{1}{1 - \frac{2m}{r}} \dot{r}^2 - r^2 \dot{\varphi}^2 = 0 \qquad \Big| \cdot \frac{1}{\dot{\varphi}^2} \tag{2.7}$$

$$\frac{1}{1-\frac{2m}{r}}\left(\frac{E^2}{\dot{\varphi}^2}-\frac{\dot{r}^2}{\dot{\varphi}^2}\right)-r^2=0 \tag{2.8}$$

Where

$$\frac{\dot{r}}{\dot{\varphi}}=\frac{\frac{dr}{ds}}{\frac{d\varphi}{ds}}=\frac{dr}{d\varphi}=: r' \tag{2.9}$$

With the conservation of "angular momentum" ($\dot{\varphi}=L/r^2$):

$$\frac{1}{1-\frac{2m}{r}}\left(\frac{E^2r^4}{L^2}-r'^2\right)-r^2=0 \tag{2.10}$$

respectively:

$$r'^2=\frac{E^2}{L^2}r^4-r^2\left(1-\frac{2m}{r}\right) \tag{2.11}$$

With this equation $r(\varphi)$ principally could be evaluated. But the equation becomes a lot simpler with the substitution $u(\varphi):=1/r(\varphi)$ (radial-inverse polar-coordinates):

$$r:=1/u \qquad r':=-u'/u^2 \tag{2.12}$$

Then the geodesic equation is:

$$u'^2=-u^2+2mu^3+\frac{E^2}{L^2} \tag{2.13}$$

Differentiation with respect to φ leads to a differential equation of second order for u:

$$u''=-u+3mu^2 \tag{2.14}$$

$$u''+u=3mu^2 \tag{2.15}$$

To derive the total deflection angle δ between the two asymptotes of the light-like geodesic, we introduce an approximation (see *Figure 2.3*) and assume:

$$u=\frac{sin\varphi}{b} \tag{2.16}$$

This leads to the following equation after insertion into 2.15:

$$u''+u=3m\frac{sin^2\varphi}{b^2}=3m\frac{1-cos^2\varphi}{b^2} \tag{2.17}$$

The solution in the weak field limit, at far distances from the centre, is given by:

$$u_1=3m\frac{1+\frac{1}{3}cos2\varphi}{2b^2} \tag{2.18}$$

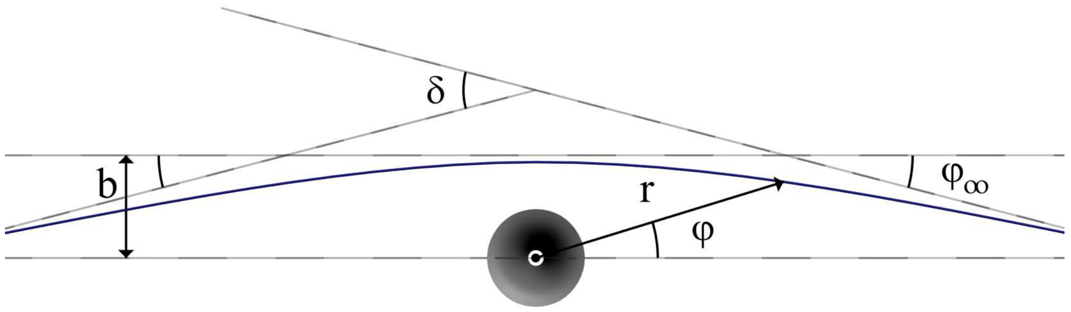


Figure 2.3: *Linearisation of a Geodesic*

In second order we have:

$$u = \frac{sin\varphi}{b} + \frac{3m}{2b^2}(1 + \frac{1}{3}cos2\varphi) \tag{2.19}$$

If we assume that r is large compared to u, we have $sin\varphi \approx \varphi$ and $cos\varphi \approx 1$. Taking the limit $u \to 0$ and $\varphi \to \varphi_\infty$, leads to:

$$0 = \frac{\varphi_\infty}{b} + \frac{3m}{2b^2}(\frac{4}{3}) \tag{2.20}$$

The total deflection angle δ is twice φ_∞:

$$\delta = 2|\varphi_\infty| = \frac{4m}{b} \tag{2.21}$$

Also from the Schwarzschild solution (at large distances) the integraton constant m can be interpreted as the mass of the spacetime:

$$m = \frac{GM}{c^2} \tag{2.22}$$

Now we can replace m, which is mass in [kg] by M, which is mass in [m]. Insertion into the equation above leads to:

$$\delta = 4\frac{GM}{c^2}\frac{1}{b} \tag{2.23}$$

The Schwarzschild solution has an apparent singularity at:

$$r = R_s := 2\frac{GM}{c^2} \tag{2.24}$$

This radius R_s is called *Schwarzschild radius*, which also is the radius of the event horizon. Final insertion leads to the final equation:

$$\delta = 2\frac{R_s}{b} \tag{2.25}$$

Now we have found an expression for the deflection angle that is only dependent on the Schwarzschild radius (which is equivalent to the mass of an object) and the distance b. In chapter 3.3 we will use this equation to approximate a curved ray of light by a “kinked” one.

Parts of this approximation and much more mathematical material about General Relativity and astronomy can be found in [Str91].

2.2 Visualisation using Raytracing

Nowadays, computer aided 3D visualisation techniques are used routinely in many fields. During a visualisation process pictures and movies are produced. In the field of science, structures and processes that go beyond our everyday experience are often visualised e.g. a tensor field of a numerically simulated fluid or a tensor field of spacetime curvature (see [Ben04]). Scientific visualisations need an excellent readability of the information contained in the structure in order to help to understand the problem under investigation.

Entertainment is another field, where visualisation is used frequently. Its purpose is to allow cheaper productions and/or to visualise fantastic structures, e.g. an alien form of life in a science fiction movie. In the entertainment industry it is important to generate realistically looking images of very high quality to convince and impress the audience.

To generate an image of an (artifical) scenery, one has to define the shapes of 3D objects geometrically, the material properties (e.g. roughness, colour, etc) of the 3D object’s surfaces, light sources and a camera object. The combination of the geometric data, the material data, the light sources and the camera is called a *3D scene* and the process of image generation is called *rendering*.

Commonly, two main techniques are used for rendering. Projection[2] is the first and raytracing is the second. Since raytracing is important for this thesis, it is described in detail in the following sections.

2.2.1 Principles of Raytracing

The approach of raytracing is the following: Light sources emit rays of light into a 3D scene. These rays intersect 3D objects. At the point of intersection light is absorbed, reflected or refracted, dependent on the material assigned to the 3D

[2]In fact these techniques often use a transformation instead of a projection, because transformations are reversible whereas projections are not.

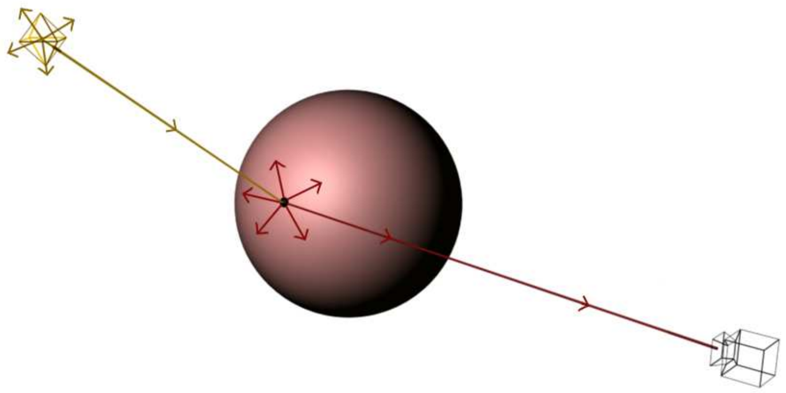


Figure 2.4: *The Approach of Raytracing*

object. Some of these reflected and refracted rays of light finally find their way to the camera, or more precisely intersect with the camera image plane, see *Figure 2.4*.

Since it would be quite inefficient to emit many rays that do not intersect the camera image plane, the raytracing algorithm proceeds “backwards”.

Initially, one ray, the so called *camera ray* or *first order ray*, is sent from the camera into the 3D scene. In our example, see *Figure 2.5*, the camera ray intersects a 3D object, a sphere. At the point of intersection with the sphere a colour value dependent on the surface normal, the material properties and the lightsources is calculated. If the material is a reflective or a refractive one, new rays (*second order rays*) are sent into the 3D scene in appropriate directions. The directions can be defined by simple laws, e.g. incoming angle equals outgoing angle for reflection or Snell’s law for refraction. These second order rays then again can intersect some 3D objects, where a colour value depending on their material properties is evaluated etc. A more precise definition of the recursive raytracing algorithm can be found in [Fol97].

Shading, a term used frequently later, is the process of evaluating the colour and transparency at a given surface point. The entity that evaluates the colour is usually called a material, or in terms of the 3D package Maya, a *shader*.

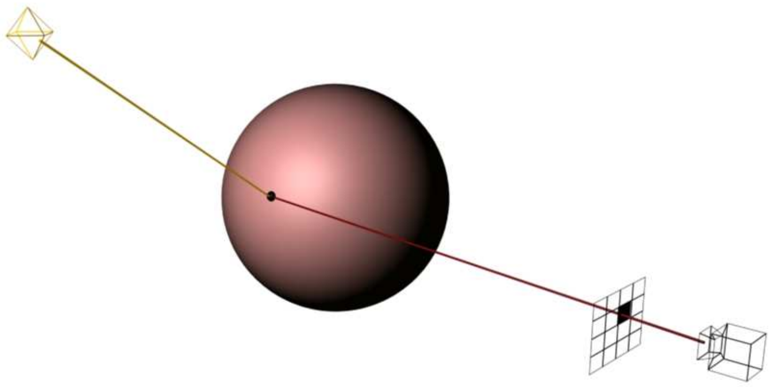


Figure 2.5: *The Raytracing Algorithm*

2.2.2 Examples, Advantages and Disadvantages

In contrast to projection techniques the concept of raytracing is more based on a physical model. The possibility to calculate light effects physically correct is a great advantage of raytracing. However, the physical model has its cost. The overall rendering process using projection techniques is generally faster than using raytracing. It has furthermore the advantage that less data have to be kept in memory during rendering. This is especially important if very large and detailed 3D scenes as used in motion picture productions are generated.

Rendering using projection techniques requires to use simple geometric primitives that can be projected easily. In most cases these geometric primitives are polygons (triangular primitives) or quads (rectangular primitives). All higher order objects have to be tessellated into these simple primitives before rendering.

In raytracing, any geometric object can be used if a calculation for an intersection between a ray and the object can be formulated. For example, a sphere is for a raytracer a simple object which is “perfectly round”. In contrast, a projection rendering engine would first tessellate the sphere into triangles and then render the triangle representation of the sphere.

Thus, modern high-end rendering engines combine both techniques. They use raytracing only were it is really necessary. Such rendering engines are called hybrid rendering engines. Examples of pure raytracing engines are POV Ray and Light++. An example of a pure projection rendering engine is OpenGL. A modern high-end rendering engine that is widely used for movie production is PIXAR Renderman, which was originally designed to be a pure projection rendering engine. In the last years it became a hybrid rendering engine. Another high-end rendering engine is Mental Ray by Mental Images, which was initially a pure raytracer and became a hybrid rendering engine later. The rendering engine of Maya is also a hybrid one.

Chapter 3

Description of the Plug-In

3.1 Motivation and Basic Idea

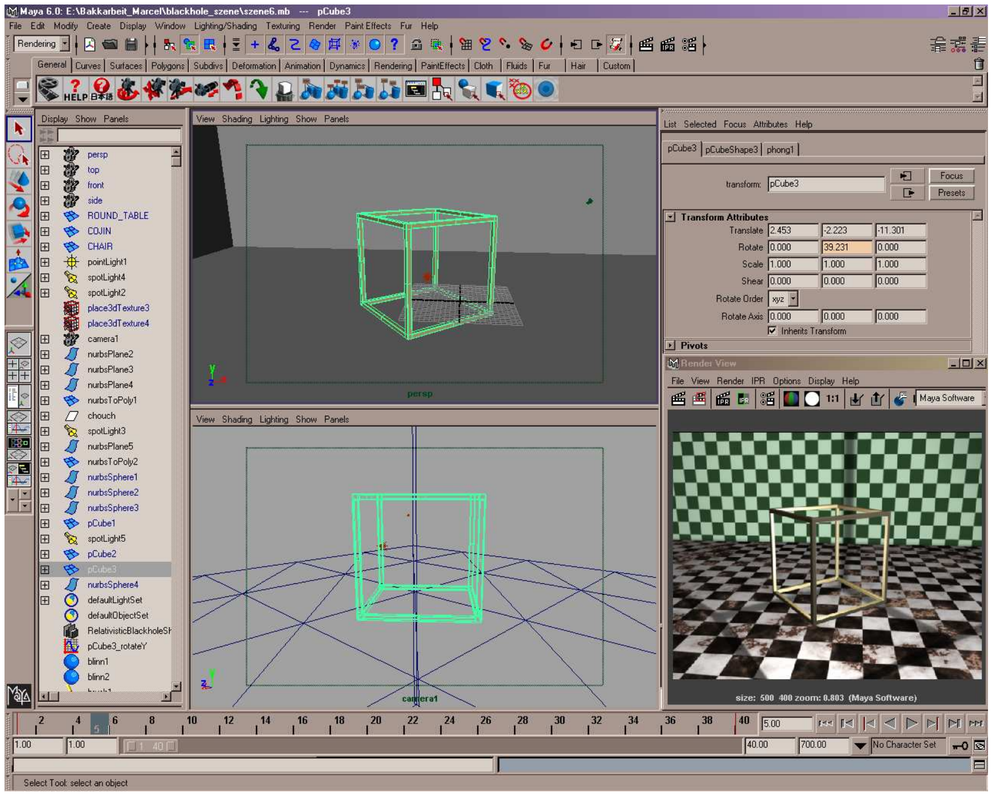
My intention was to provide a possibility for an end user who is familiar with common 3D applications, which allows to generate correct images of the bending effect and, thus, can easily be used for scientific visualisation as well as for entertainment purposes. The idea was to extend the functionality of a widely used application by a plug-in. It goes without saying that raytracing is a perfect rendering technique to simulate the bending effect. A commercial rendering engine, however, is usually not capable of simulating curved rays of light. So, a simple way to make this possible had to be found.

The idea was not to extend the raytracing engine itself by the possibility to simulate curved rays but to create a shader that "does" the light bending. This would work in a similar way as with a shader that simulates the refraction of glass: A sphere centred around the black hole with an assigned shader could "produce" the light bending effect. So, the bending of the rays would "happen" inside the shader and so could be rendered in a conventional way by the rendering engine. In fact, the spectator would see a sphere on which is projected what is to be simulated.

3.2 Maya

One of the many 3D applications is Maya. It was originally developed by Alias-Wavefront. Alias-Wavefront has changed its name to Alias recently and was taken over by Autodesk last year. Maya can be customised to a high extend, which is the reason why it is often first choice in large scale productions. Two very famous examples are its use as the main 3D application for Peter Jackson's "Lord of the

Rings" and as the main animation package for George Lukas' "Star Wars" episodes 2 and 3. Maya might not show its full strength in a single user environment but its flexibility is still a great advantage for what concerns implementation. This was the reason why I chose Maya for my relativistic shader plug-in.


Maya 6.0: E:\Bakkarbeit_Marcel\blackhole_szene\szene6.mb --- pCube3
File Edit Modify Create Display Window Lighting/Shading Texturing Render Paint Effects Fur Help
Rendering
General Curves Surfaces Polygons Subdivs Deformation Animation Dynamics Rendering PaintEffects Cloth Fluids Fur Hair Custom
Display Show Panels
persp
top
front
side
ROUND_TABLE
COJIN
CHAIR
pointLight1
spotLight4
spotLight2
place3dTexture3
place3dTexture4
camera1
nurbsPlane2
nurbsPlane3
nurbsPlane4
nurbsToPoly1
chouch
spotLight3
nurbsPlane5
nurbsToPoly2
nurbsSphere1
nurbsSphere2
nurbsSphere3
pCube1
spotLight5
pCube2
pCube3
nurbsSphere4
defaultLightSet
defaultObjectSet
RelativisticBlackholeSh
pCube3_rotateY
blinn1
blinn2
View Shading Lighting Show Panels
persp
camera1
List Selected Focus Attributes Help
pCube3 pCubeShape3 phong1
Focus
Presets
Pivots
Render View
File View Render IPR Options Display Help
Maya Software
size: 500 400 zoom: 0.803 (Maya Software)
No Character Set
Select Tool: select an object

transform: pCube3

Transform Attributes

Translate	2.453	-2.223	-11.301
Rotate	0.000	39.231	0.000
Scale	1.000	1.000	1.000
Shear	0.000	0.000	0.000
Rotate Order	xyz		
Rotate Axis	0.000	0.000	0.000

Inherits Transform

Figure 3.1: *The User Interface of Maya*

3.2.1 General Description

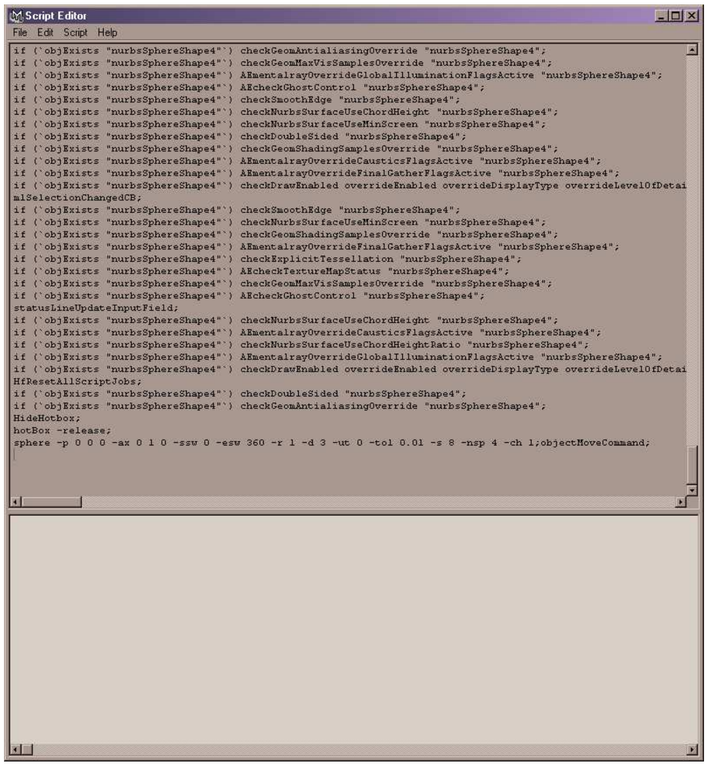
From a 3D artist's point of view, Maya is a powerful application for creating and editing 3D computer graphics. It provides a lot of tools for modeling[1], animating and rendering. The artist uses the common graphical user interface to do his work and can extend the GUI[2] by writing scripts in an interpreted programming language called MEL[3].

[1]computer aided "sculpting" of 3D objects

[2]**G**raphical **U**ser **I**nterface

[3]**M**aya **E**mbedded **L**anguage

From a programmer's point of view, Maya provides a large collection of routines and tools that can be accessed via the API[4].



```
if (`objExists "nurbsSphereShape4"`) checkGeomAntialiasingOverride "nurbsSphereShape4";
if (`objExists "nurbsSphereShape4"`) checkGeomMaxVisSamplesOverride "nurbsSphereShape4";
if (`objExists "nurbsSphereShape4"`) AEmentalrayOverrideGlobalIlluminationFlagsActive "nurbsSphereShape4";
if (`objExists "nurbsSphereShape4"`) AEcheckGhostControl "nurbsSphereShape4";
if (`objExists "nurbsSphereShape4"`) checkSmoothEdge "nurbsSphereShape4";
if (`objExists "nurbsSphereShape4"`) checkNurbsSurfaceUseChordHeight "nurbsSphereShape4";
if (`objExists "nurbsSphereShape4"`) checkNurbsSurfaceUseMinScreen "nurbsSphereShape4";
if (`objExists "nurbsSphereShape4"`) checkDoubleSided "nurbsSphereShape4";
if (`objExists "nurbsSphereShape4"`) checkGeomShadingSamplesOverride "nurbsSphereShape4";
if (`objExists "nurbsSphereShape4"`) AEmentalrayOverrideCausticsFlagsActive "nurbsSphereShape4";
if (`objExists "nurbsSphereShape4"`) AEmentalrayOverrideFinalGatherFlagsActive "nurbsSphereShape4";
if (`objExists "nurbsSphereShape4"`) checkDrawEnabled overrideEnabled overrideDisplayType overrideLevelOfDetai
mlSelectionChangedCB;
if (`objExists "nurbsSphereShape4"`) checkSmoothEdge "nurbsSphereShape4";
if (`objExists "nurbsSphereShape4"`) checkNurbsSurfaceUseMinScreen "nurbsSphereShape4";
if (`objExists "nurbsSphereShape4"`) checkGeomShadingSamplesOverride "nurbsSphereShape4";
if (`objExists "nurbsSphereShape4"`) AEmentalrayOverrideFinalGatherFlagsActive "nurbsSphereShape4";
if (`objExists "nurbsSphereShape4"`) checkExplicitTessellation "nurbsSphereShape4";
if (`objExists "nurbsSphereShape4"`) AEcheckTextureMapStatus "nurbsSphereShape4";
if (`objExists "nurbsSphereShape4"`) checkGeomMaxVisSamplesOverride "nurbsSphereShape4";
if (`objExists "nurbsSphereShape4"`) AEcheckGhostControl "nurbsSphereShape4";
statusLineUpdateInputField;
if (`objExists "nurbsSphereShape4"`) checkNurbsSurfaceUseChordHeight "nurbsSphereShape4";
if (`objExists "nurbsSphereShape4"`) AEmentalrayOverrideCausticsFlagsActive "nurbsSphereShape4";
if (`objExists "nurbsSphereShape4"`) checkNurbsSurfaceUseChordHeightRatio "nurbsSphereShape4";
if (`objExists "nurbsSphereShape4"`) AEmentalrayOverrideGlobalIlluminationFlagsActive "nurbsSphereShape4";
if (`objExists "nurbsSphereShape4"`) checkDrawEnabled overrideEnabled overrideDisplayType overrideLevelOfDetai
HfResetAllScriptJobs;
if (`objExists "nurbsSphereShape4"`) checkDoubleSided "nurbsSphereShape4";
if (`objExists "nurbsSphereShape4"`) checkGeomAntialiasingOverride "nurbsSphereShape4";
HideHotbox;
hotBox -release;
sphere -p 0 0 0 -ax 0 1 0 -ssw 0 -esw 360 -r 1 -d 3 -ut 0 -tol 0.01 -s 8 -nsp 4 -ch 1;objectMoveCommand;
```

Figure 3.2: *The Script Editor*

Maya was developed in C++ and is available for the following platforms: Irix (SGI/PC), Linux (PC), Windows (PC) and OS X (Mac). Its core consists of the C++ classes and functions that provide all data manipulation mechanisms and, secondly, of the MEL script interpreter. Everything that can be accomplished with a mouse click can be done by executing a line of MEL script. In fact, the mouse click first produces a script command that is then executed by the script interpreter. This can be made visible in Mayas *script editor* window (see *Figure 3.2*) by enabling "echo all commands". E.g. the last line of text in *Figure 3.2* shows the command that is generated and executed when clicking on "create NURBS sphere" in the GUI.

Additionally, the GUI elements are also controlled via the script language. So, an experienced user can easily customise the GUI of the whole application to his own needs.

[4]**A**pplication **P**rogramming **I**nterface

3.2.2 The Dependency Graph

Let us have a look at the data organisation model of Maya. All information of a 3D scene in Maya is stored in a network of nodes. A node, the atomic entity of the network, has inputs and outputs that are connected to inputs and outputs of other nodes. The data is "flowing" through this network of nodes, the "operants" that modify or create data. Such a design is often refered to as *data flow model.* Data, fed into the network, flows through and comes out at the other end. In fact, Maya does not use such a strict data flow model, but a so called *push-pull model.* But let us have a more detailed look at the atomic entities first.

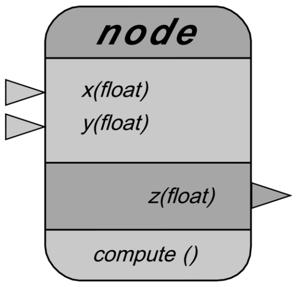


Figure 3.3: *Schematic Node*

A *node* is the fundamental component in the representation of a Maya scene. It holds *attributes* and a *compute function.* The attributes store the data in the node. As mentioned above, a node has some input and some output connectors so that a network of nodes can be set up. Any attribute of a node can, but does not have to, be declared as either an input or an output attribute. An attribute can be of a simple data type such as a `boolean`, `char`, `integer` or `float` or it can be a more complex type like `point` or `vector` or even `polygon surfaces` or `particle clouds`. All types of attributes that may be used can be found in the API. *Figure 3.3* shows a node with two input and one output connections.

The compute function operates on the attributes of the node. It computes the output attributes depending on the input attributes. The compute function should be restricted to the local data of the node. The idea of the network is that every node is "responsible" for a small part of the scene and the network as a whole performs complex tasks. In our example node shown in *Figure 3.3*, the compute function computes a value z from x and y.

If we construct a network where we want to get certain results for an output attribute, it is important to think carefully when which nodes call their compute functions. If we think of the network as an acyclic directed graph, one could start at all source nodes, call their compute functions and then update the neighbouring nodes, then repeat this successively for the neighbours until the final output is computed. This way, the whole network will be updated even if it is not necessary for the output attribute of interest.

As example, take the network shown in *Figure 3.4*. A data change in node *n1* would involve the update of all other nodes (*n2* to *n5*). This approach is called a *push model* because data is pushed in a top-to-bottom fashion through the network.

Often, it is not necessary that all nodes are up to date. Perhaps it is just node *n5* that needs to be up to date. Then nodes *n3* and *n4* are recomputed unnecessarily decreasing performance.

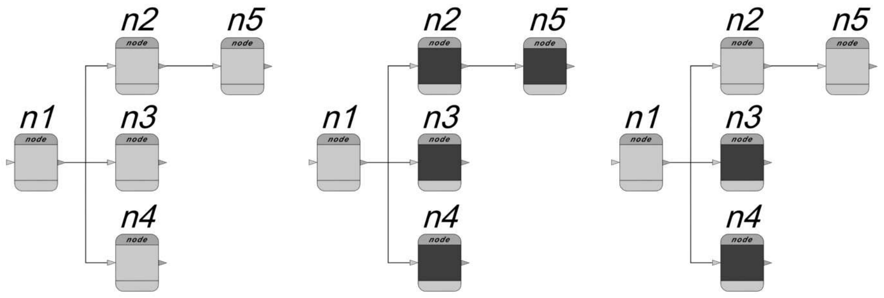


Figure 3.4: *Schematic Network*

To solve this problem, an *update flag* is introduced. Every node is associated with a flag that marks whether a node needs to recompute or not. Assume, node *n1* is changed, and only node *n5* needs to be up to date. Now, in a first step, the update flag is pushed through the network starting from node *n1*. All nodes that depend on *n1* are marked (see middle diagram of *Figure 3.4*). The source node itself is recomputed when the values are changed. Thus, *n1* itself is not marked. In a second step, values are requested from *n5*. Now starting from *n5* the update flags are checked and if they are set, an update has to be done. As *n5* is marked, it has to be updated. Then, the values from *n2* are requested to recompute *n5*. But as *n2* itself is marked, the values from *n1* have to be requested first. Since they are not marked, they are up to date and passed to *n2*. *n2* recomputes, unmarks its update flag and passes its outgoing attributes to *n5*. Finally, *n5* recomputes. Note, that nodes *n3* and *n4* were not updated and are still marked (see right diagram of *Figure 3.4*).

The approach described above is called a push-pull model, because first the flag is pushed through the network and then the data is pulled down.

The update mechanism in Maya works in a similar fashion, but it is a bit more sophisticated. Not every node has an update flag but every attribute (input, output or other attribute) does. The programmer of a node additionally has to tell Maya on which input attributes the ouput attributes dependent. A more precise schematic diagram of a node can be seen in *Figure 3.5*. Here, the update flag is represented by a rectangle next to the attribute name and the dependency of the attributes is shown by the white lines.

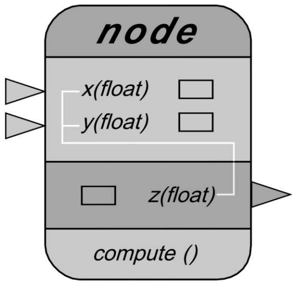


Figure 3.5: *Schematic Node Revisited*

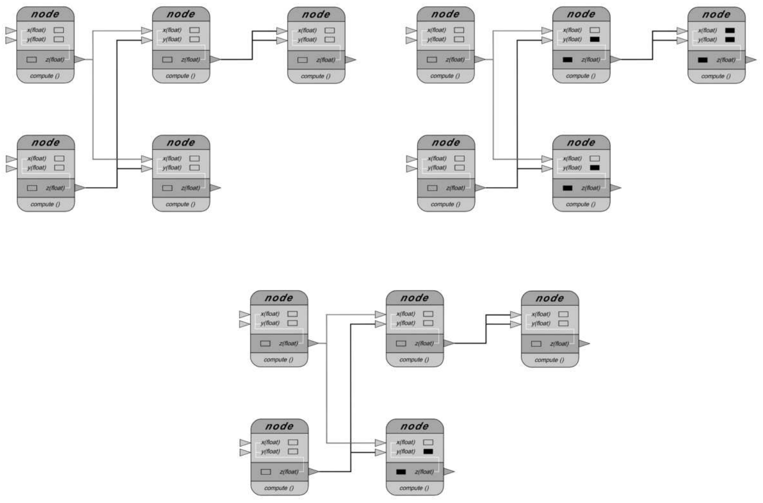


Figure 3.6: *Schematic Network Revisited*

Figure 3.6 shows a network of revisited nodes (upper left network). Here again the update process is illustrated. The lower left node is changed, the update flag is pushed through the nodes marking the attributes (upper right network), the rightmost node is asked to be up to date and the according nodes recompute (lower network).

A node is a very general entity and can be anything: a timer, a simple mathematical operation, a polygon object or a complex fluid simulation.

This data organisation and computation model provides a high flexibility. For example, if a user alters a 3D model, a node could be created and inserted after the original 3D model creation node. The newly created node gets the model as input and outputs the altered model. All modifications made by the user can hereby later easily be changed or canceled, either by changing some input values of the inserted new node or by deleting it.

The network, or parts of it, can be analysed and edited with a hypergraph editor within Maya. In *Figure 3.7* we see all the nodes and connections of an example network. Nodes can be deleted, altered or connections can be created between attributes on a very low level. This allows a very powerful and flexible, but not very user friendly, control over the entire 3D scene.

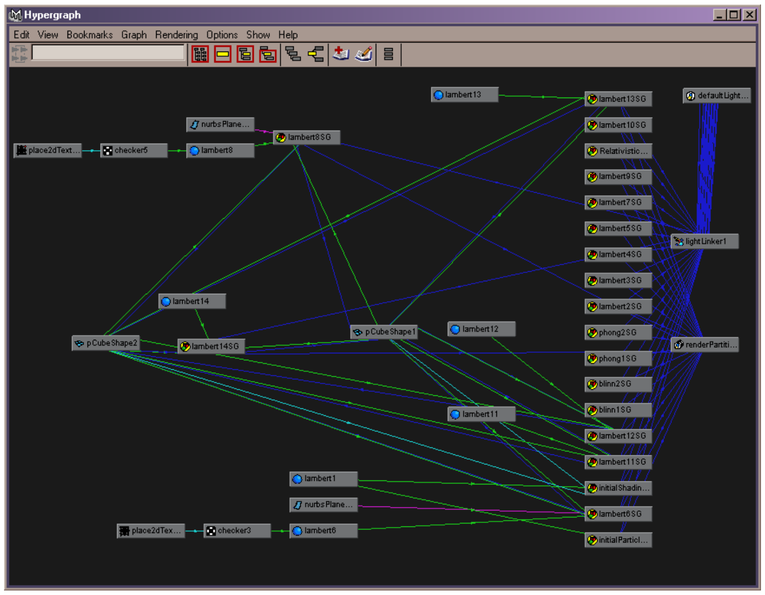
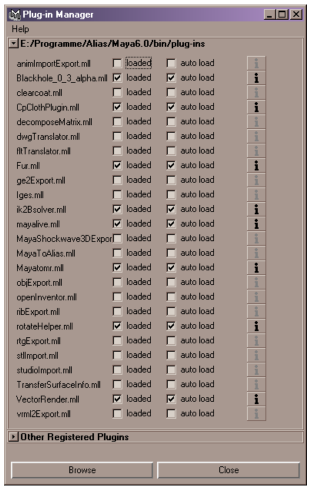


Figure 3.7: *The Dependency Graph*

3.2.3 API

The **A**pplication **P**rogramming **I**nterface enables to develop either stand-alone applications with access to Maya's data or plug-ins which extend the functionality of Maya. In this section, only the latter will be described.

Maya provides a developer toolkit with a lot of examples and prepared files that help to get started and get the right configurations for paths and linking. Compiled plug-ins are runtime linked into Maya. So, one has to compile the code to a *.dll[5], *.so[6] or *.lib[7]. The plug-in has to be copied to the `../maya/bin/plug-ins` directory and then can be loaded and unloaded by the plug-in manager, (see *Figure 3.8*).



E:/Programme/Alias/Maya6.0/bin/plug-ins

animImportExport.mll	☐ loaded	☐ auto load
Blackhole_0_3_alpha.mll	✓ loaded	✓ auto load
clearcoat.mll	☐ loaded	☐ auto load
CpClothPlugin.mll	✓ loaded	✓ auto load
decomposeMatrix.mll	☐ loaded	☐ auto load
dwgTranslator.mll	☐ loaded	☐ auto load
fltTranslator.mll	☐ loaded	☐ auto load
Fur.mll	✓ loaded	✓ auto load
ge2Export.mll	☐ loaded	☐ auto load
Iges.mll	☐ loaded	☐ auto load
ik2Bsolver.mll	✓ loaded	✓ auto load
mayalive.mll	✓ loaded	✓ auto load
MayaShockwave3DExpor	☐ loaded	☐ auto load
MayaToAlias.mll	☐ loaded	☐ auto load
Mayatomr.mll	✓ loaded	✓ auto load
objExport.mll	☐ loaded	☐ auto load
openInventor.mll	☐ loaded	☐ auto load
ribExport.mll	☐ loaded	☐ auto load
rotateHelper.mll	✓ loaded	✓ auto load
rtgExport.mll	☐ loaded	☐ auto load
stlImport.mll	☐ loaded	☐ auto load
studioImport.mll	☐ loaded	☐ auto load
TransferSurfaceInfo.mll	☐ loaded	☐ auto load
VectorRender.mll	✓ loaded	✓ auto load
vrml2Export.mll	☐ loaded	☐ auto load

Other Registered Plugins

Figure 3.8: *Plug-In Manager*

The API consists of about 250 C++ classes, but only a few are typically needed for a plug-in. The classes can logically be grouped as follows:

Proxy objects (MPx): Proxy classes are those classes, from which other classes are thought to be derived. For example `MPxNode` is the general node class which is the start for the black hole shader plug-in.

Iterator classes (MIt): Iterator classes are used to traverse through different kinds of data objects. Examples are an iterator for the dependency graph nodes, for key-frames or polygon edges.

Function sets (MFn): The function sets provide all the data manipulation functions for data class objects. In Maya the functions are separated from the data classes[8]. Thus, for any kind of data, at least one function set is needed that can operate on it.

[5]Windows
[6]Linux, Irix
[7]OS X
[8]In terms of computer science this is often refered to as the *Strategy Pattern*.

Maya classes (M): All other classes. They also include all necessary data classes, for example `MPoint` or `MVector`. Even more complicated ones like `floatpointarrays` or similar data types. One usually gets by with these types and does not need to use the STL[9] or other libraries. Using Maya types and self-written types only has the advantage that the source code of a Maya plug-in is portable to all other platforms that can run Maya.

The interface itself is defined in header files that can be included into the personal code. The functionality comes along in five precompiled libraries, namely: *OpenMaya*, *OpenMayaUI*, *OpenMayaAnim*, *OpenMayaFX* and *OpenMayaRender*. They provide the fundamental functionality, user interface elements, animation classes, classes for dynamic simulations and classes for rendering.

A plug-in is typically of one of the following types:

Command: Customised commands can be written. They work the same way as any native MEL command of Maya does. They can be executed by the script interpreter. Features like undo/redo and help are supported.

Dependancy Graph Node: One can derive a class from the general DAG Node class `MPxNode`. This allows to write an own single node, which can then be connected into the dependency graph like a native Maya node. The API also provides more specialised node classes for locators, ik-solvers[10], deformers, fields, emitters, springs, manipulators, surfaces, object sets, shaders and transform nodes.

File Translator: A class for deriving a custom file translator is provided by the API. File translators are responsible for loading and saving data. This and a prepared example from the developer toolkit enables to support your own file formats.

The Class MObject

When using the Maya API one never gains direct control over the internal data of Maya. Instead, one uses the `MObject` class to access all internal objects. `MObject` is a handle to other objects in the Maya core. An instance of it represents a specific attribute or a node and can be of many different types. This has the advantage

[9]**S**tandard **T**emplate **L**ibrary, a collection of algorithms and data structures, part of the C++ standard

[10]inverse kinematic solvers, used for skeleton animation

that for the plug-in, the internal representation is not important. This allows the developers at Alias to change the internal structure of the core by still supporting the old plug-in interface.

A Dependency Graph Node Plug-In

Let us have a look at a small example of a dependency graph node plug-in. Listing 3.1 shows a class declaration of a simple node like the node in *Figure 3.3*. The class `ExampleNode` is derived from the base class for all DAG nodes of the API.

When a node in Maya is created, it internally generates two objects. An `MObject` and the user derived object. The association between these two objects is established when the `MPxNode` constructor is called. Thus, it is not possible to call member functions of the derived class in its constructor. To make this possible, the function `postConstructor()` can be used.

The compute function is the heart of the plug-in. It recomputes the output attributes based on the input attributes.

The `MDataBlock` storages the node's attributes and connectors (in Maya API called plugs) and provides smart handles for reading and writing the attributes. When reading an attribute of the node, an `MDataHandle` has to be created from the `MDatablock` and the `MObject` representing the according attribute. Then the values of the attribute can be read or written using the created `MDataHandle`. The `MDataBlock` is only valid during the call of the compute function of the according node.

For example, when the attribute `aX` is read, one has first to create a handle to the attribute by using the `MDataBlock` and the static `MObject` member `aX`: `MDataHandle aXhnd = data.inputValue(aX)`. This ensures that the input value is valid and not marked by the update flag. Using `aXhnd.set(4.5)` would assign a new value to the attribute. Of course, there are a lot of functions for getting or setting values and also for type checking.

`MPlugs` are generally used to access attributes of nodes. An attribute can be connected to a plug to access it. In simple cases, the attribute and the plug are equivalent, but they need not, for example, when accessing array types. Plugs can have a hierarchy, thus, every node has an internal tree of plugs indicating connections that have been made to attributes of that node. Here the `MPlug` represents the data value that needs to be recomputed.

Listing 3.1: DAG Node Class

```
class ExampleNode : public MPxNode
{
  public:
                          ExampleNode();
      virtual             ~ExampleNode();

      virtual void        postConstructor();
      virtual MStatus     compute( const MPlug&, MDataBlock& );

      static void*        creator();
      static MStatus      initialize();
      static MTypeId      id;

  private:

      static MObject      aX;
      static MObject      aY;

      static MObject      aZ;
};
```

The function `creator()` is used to register our external code to Maya by its plug-in mechanism. The `initialize()` function is responsible for creating instances of new classes of the node and to register the node's attributes with type and two names. The variable `id` is a four byte identifier. It must be unique and identifies the node.

3.2.4 MEL

The Maya Embedded Language is an interpreted script language. As already mentioned, MEL is a central design point in the Maya architecture. All interactions with the GUI first create MEL scripts, which then are executed. MEL is also used to control and extend the GUI.

There are several ways of executing MEL code in Maya. One can use the *command line* which is always visible at the bottom of the main window, the *script editor* (see *Figure 3.2*) or a shell like window, the *command shell*. If one wants to execute the same lines of code repeatedly during a work flow, one can create a button to execute these lines or one can assign a hot-key.

The syntax of MEL is very similar to C. MEL contains the following basic structures: Variables, vectors, arrays, matrices, scopes, functions, commands, operators, loops and conditional statements and comments.

Variables can be of the basic types integer (`int`), 64 bit floating point (`float`) and strings (`string`). A specific type boolean does not exist, an integer has to be used instead. The keywords `true`, `on` and `yes` are synonyms for 1 and `false`, `off` and `no` for 0. In MEL a variable name always has to start with `$`.

Vector is a type that is composed of three floats. Like with variables, names have to start with `$`. The components of the vector can be accessed by the suffices `.x`, `.y` and `.z`.

Arrays are lists of elements of the same type. MEL manages the size of the arrays independently, they will automatically be resized on demand. An array name must also start with `$`. To access certain elements the common “[ ]” brackets have to be used.

Matrices are two dimensional arrays with the restriction that they can only store floats and that they are not resized automatically.

Scopes are the areas of validity of variables. A scope is a section of MEL code that is enclosed by { and }. A variable exists in the scope it was defined in, including inner scopes. In other words, a variable is local to its scope. A variable can be made global by writing `global` in front of a variable declaration. If a variable is global it is accessible from anywhere in Maya.

Functions are called procedures in MEL. A procedure can take an unlimited number of arguments and return no or a single value. The keyword `proc` is used to define a procedure.

Commands are similar to functions but they are provided by Maya and not written by a user. Commands take some input arguments, do some action and have a return value. One of the simplest commands is `print()` which prints out a `string` in the script editor console. Commands build up the functionality of Maya, more than thousand commands are provided. A complete list including descriptions can be found in the *Maya help files*, ([MH65]). Some commands that are used frequently are `setAttr` and `getAttr` to set and get values of certain attributes of a node. Another very commonly used command is `ls`[11] which returns the names of the objects in a scene. I also used `connectAttr`, `listConnections`, `shadingNode`, `sphere` and `sets` in my MEL scripts, which will be described later.

[11]stands for list

Operators come with the following precedence: `()`,`[]`; `!`, `++`, `--`; `*`, `/`, `%`, `^`; `+`, `-`; `<`, `<=`, `>`, `>=`; `==`, `!=`; `&&`; `||`; `?:`; `=`, `+=`, `-=`, `*=` and `/=`.

Loops and conditional statements: Three types of loops are supported by MEL: the `for`, the `while` and the `do-while` loop. The common constructs `continue` and `break` to proceed to the next loop iteration or to leave the loop are provided, too. The conditional statements of MEL are `if` and `switch`. MEL also supports the conditional construct `<bool expr>?<code>:<code>`.

Comments are similar to C++. `/*` and `*/` are used for multi-line comments and `//` for single-line comments.

More information about MEL scripting and plug-in development can be found in [Gou03] and [MH65].

3.3 The Plug-In

The rendering engine of Maya is a hybrid rendering engine, it uses both, projection and raytracing. Only the objects that have a shader assigned that needs raytracing are actually raytraced. A simple example would be a sphere with a glass shader assigned. The glass shader would use Snell's Law to calculate refraction, which is only possible by raytracing. Thus, the parts of the picture that show the glass sphere are raytraced when rendered.

The rendering engine only is capable of visualising 3D scenes in euclidean geometry but not curved spacetime. In order to visualise the effect of bending of light the black hole is surrounded by a sphere, which is assigned a shader that uses, in analogy to the refraction example, the approximation derived in 2.1.3 to simulate the effect. Now, if a ray intersects the sphere the shader is called and computes a colour value, by sending a "curved" ray into the scene. The rendering engine finally gets a colour value back as if the original ray sent by it was sent into curved space.

In other words, a shader on a sphere surrounding the black hole projects the scene a spectator would see if space would be curved onto the sphere.

This approach requires the following setup: a black hole represented by a black *inner sphere*. A sphere surrounding the black hole, called the *outer sphere*. Finally, the *relativistic black hole shader* for which I developed the plug-in. Additionally, I developed two MEL scripts that integrate the whole in the GUI of Maya.

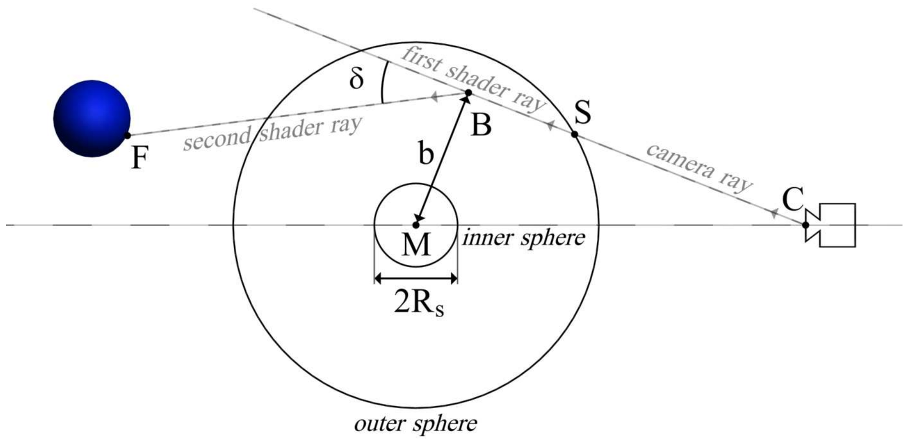


Figure 3.9: *Black Hole Setup*

3.3.1 The Relativistic Black Hole Shader

Rendering a single pixel works as follows: Starting from the camera the rendering engine sends a first order ray into the scene. In our example (see *Figure 3.9*), it intersects with the outer sphere in S and therefore the shader assigned to the outer sphere is called.

A standard shader would compute a colour and transparency value using the first order ray's incoming direction, the point of intersection, the surface normal at the intersection point and the parameters of the light sources. The relativistic black hole shader only takes into account the intersection point, the centre point of the black hole, and a mass attribute to compute the colour.

First, the "bend point" B is computed by determining the shortest distance between the centre of the black hole and the first order ray. Next, the deflection angle based on the equation 2.25 derived in chapter 2.1.3 is computed:

$$\delta = \frac{2R_s}{b}$$

with R_s is the Schwarzschild radius (the radius of the black hole's event horizon), which equals the radius of the inner sphere in our approach. b is the distance between the centre of the black hole and the point B and δ is the angle between the first order and the second shader ray. The second shader ray always starts at point B. An intersection with the backside of the *outer sphere* is intentionally prevented.

In our example, an object is located "behind" the black hole. The second shader ray intersects this object in F. So, the shader assigned to the object is called to evaluate the colour value at this point. This colour is then returned to the rendering engine by the relativistic black hole shader at the intersection point S.

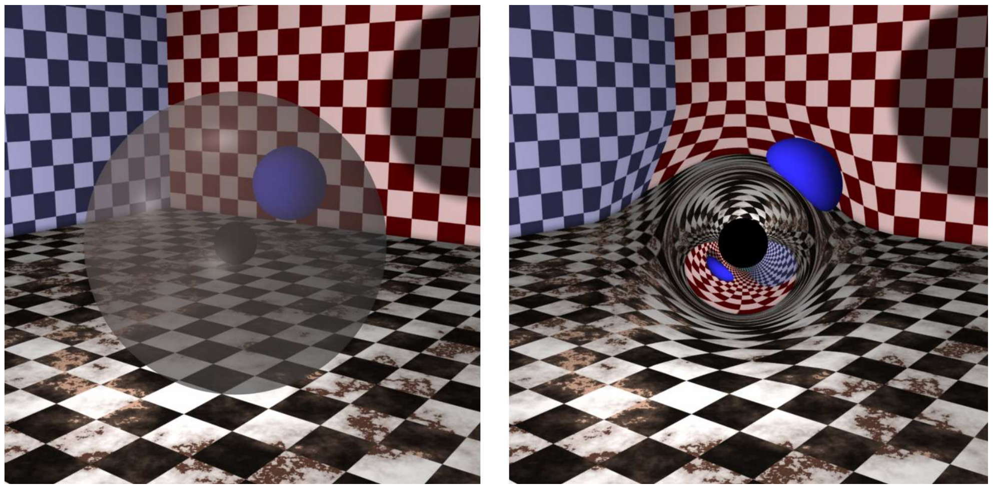


Figure 3.10: *Rendering a Blackhole*

Figure 3.10 shows an example scene composed 3D equivalent to the simplified example used to illustrate the algorithm. The picture on the left shows the inner sphere (the small black sphere in the centre), the outer sphere (the transparent grey sphere) and a real scene object, a blue sphere located behind the outer sphere. The shader is not assigned or activated for the outer sphere and therefore a transparent sphere can be seen. In the picture on the right, the relativistic black hole shader was assigned to and activated on the outer sphere, which makes it invisible as sphere and simulates the light bending effect.

Without any further measures using formula 2.25 would lead to a strong discontinuity at the border of the outer sphere, the "border between curved and euclidean space".

To ensure a continuous and seamless optical integration at the border of the outer sphere, the deflection angle δ has to be modified. A linear factor dependent on the distance to the centre of the black hole turned out to be suitable. Using this a high simulation accuracy is achieved at the inner region, where curvature is high. Near the border of the outer sphere the high correction factor leads to a nearly seamless integration into the scene.

The shader was developed with the Maya API as a shader node plug-in. It was derived from the most general base class `MPxNode`. All parameters listed above are set as input attributes in the shader node. The source code of the class declaration (see Listing 3.2) shows additional attributes that are needed to gather all information necessary for raytracing. For example the intersection point, the direction of the incoming ray and the depth of the incoming ray[12]. The attribute `aOutColor` is the only output of the node and is the final computed colour of the intersection point on the outer sphere.

The `compute()` function, again, is the heart of the shader node. It gets all the information needed for the computation of the positions and directions of the rays that the shader sends into the 3D scene (the first and second shader ray), computes them by vector analysis and then uses the functions `raytrace()` and `raytraceFirstGeometryIntersections()` of the API class `MRenderUtil` to send these rays.

The function `raytrace()` only returns the colour for the computed intersection point. Because the ray that has to be traced for an intersection bends at B, intersections between an object and the first shader ray occurring "behind" B can be ignored. The shader first uses `raytraceFirstGeometryIntersections()` to test whether there is an intersection between S and B, and if there is one, it calls `raytrace()` with the point returned by `raytraceFirstGeometryIntersections()` to evaluate the colour and transparency values of this point.

For the second shader ray only the colour information of the intersection is needed and `raytrace()` can be called directly to evaluate it.

Listing 3.2: Relativistic Black Hole Shader

```
class RelativisticBlackholeShader : public MPxNode
{
   public:
                     RelativisticBlackholeShader();
      virtual    ~RelativisticBlackholeShader();

      virtual MStatus   compute( const MPlug&, MDataBlock& );
      virtual void      postConstructor();
      static void *     creator();
      static MStatus    initialize();
      static MTypeId    id;
   private:
      static MObject aBlackholeMass;
      static MObject aXBlackholeCenter;
      static MObject aYBlackholeCenter;
```

[12]Every time a ray gets reflected, refracted or deflected by a shader, the ray's depth is increased by one. This is used to exit the raytracing algorithm in case of infinite recursion.

```
        static MObject aZBlackholeCenter;
        static MObject aInfinityRadius;
        static MObject aRenderCameraName;
        static MObject aRenderObjectsBeforeBlackhole;

        static MObject aPointCamera;
        static MObject aNormalCamera;
        static MObject aRayOrigin;
        static MObject aRayDirection;
        static MObject aObjectId;
        static MObject aRaySampler;
        static MObject aRayDepth;

        static MObject aOutColor;
};
```

An in-depth description of Maya's internal raytracing functions can be found in [MH65]. Next I will describe the integration of the relativistic black hole shader into Maya's dependency graph.

3.3.2 Integration in the Dependency Graph

To achieve a simple integration in the Maya workflow the shader is integrated in a scene using two standard NURBS[13] spheres for the inner sphere and the outer sphere connected to the shader attributes.

The position of the black hole has to be defined by the position of the outer sphere, which can be moved with the standard move tools in Maya. The radius of the outer and inner sphere are controlled by the two shader parameters `aInfinityRadius` and `aBlackholeMass`. When a user changes one of those shader parameters, the sizes of the spheres change accordingly. To accomplish this connections between the shader node and the NURBS sphere's nodes have to be established.

The attributes x-, y-, and z-position of the outer sphere's transform node are connected to the relativistic black hole shader attributes `aX-`, `aY-` and `aZBlackholeCenter`. The same outer sphere transformation attributes are then connected to the transformation attributes of the inner sphere. Now the outer sphere position attributes control the position of the black hole centre in the shader as well as the inner sphere position.

[13]non uniform rational b-splines

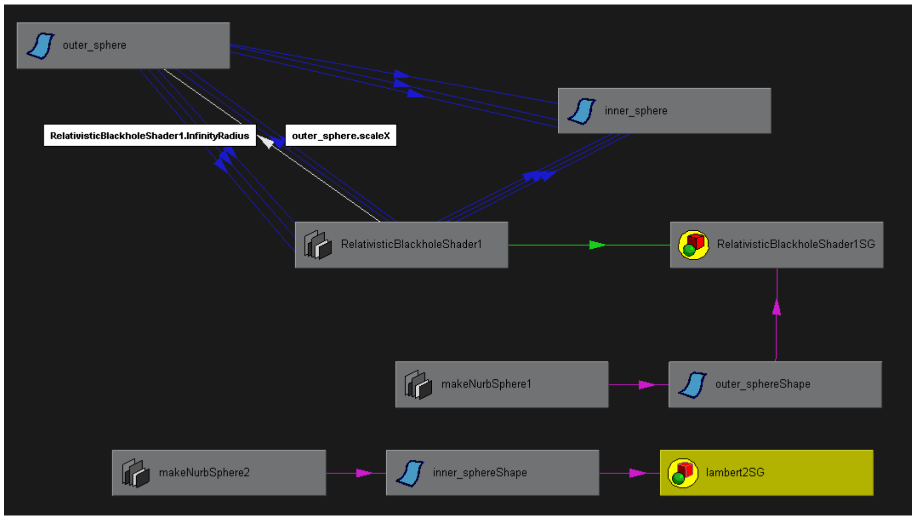


Figure 3.11: *Connection Setup*

Next the shader's attribute `aInfinityRadius` is connected to the scale attributes of the outer sphere and the shader's `aBlackholeMass` is connected to the scale values of the inner sphere. *Figure 3.11* shows the established connections and the final network that is part of the dependency graph.

Creating all these connections manually is a tedious work, which an end user should be spared. A MEL script creates all objects necessary for the black hole and sets up all the connections automatically. It also does some additional configurations.

Listing 3.3: MEL Script to Create a Black Hole

```
string $shadername = `shadingNode -asShader RelativisticBlackholeShader`;
setAttr ($shadername+".BlackholeMass") 0.5;
setAttr ($shadername+".InfinityRadius") 5.0;

sets -renderable true -noSurfaceShader true -empty -name ($shadername + "SG");
connectAttr -f ($shadername + ".outColor") ($shadername+"SG.surfaceShader");

sphere -p 0 0 0 -ax 0 1 0 -ssw 0 -esw 360 -r 1 -d 3 -ut 0 -tol 0.01 -s 8 -nsp 4 -ch 1;
string $selection[] = `ls -selection`;
string $outer_sphere = $selection[0];

string $relatives[] = `listRelatives`;
setAttr ($relatives[0]+".doubleSided") 0;

string $source[] = `listConnections -source on -destination off - plugs off
                    -connections off -shapes off $relatives[0]`;
setAttr -l true ($source[0] +".radius");

sets -forceElement ($shadername + "SG") $outer_sphere;
```

```
20
21 connectAttr -f ($shadername+".InfinityRadius") ($outer_sphere+".scaleX");
22 connectAttr -f ($shadername+".InfinityRadius") ($outer_sphere+".scaleY");
23 connectAttr -f ($shadername+".InfinityRadius") ($outer_sphere+".scaleZ");
24 connectAttr -f ($outer_sphere+".translateX") ($shadername+".BlackholeXCenter");
25 connectAttr -f ($outer_sphere+".translateY") ($shadername+".BlackholeYCenter");
26 connectAttr -f ($outer_sphere+".translateZ") ($shadername+".BlackholeZCenter");
27
28
29 sphere -p 0 0 0 -ax 0 1 0 -ssw 0 -esw 360 -r 1 -d 3 -ut 0 -tol 0.01 -s 8 -nsp 4 -ch 1;
30 string $selection[] = `ls -selection`;
31 string $inner_sphere = $selection[0];
32
33 string $relatives[] = `listRelatives`;
34 setAttr ($relatives[0]+".doubleSided") 0;
35
36 string $source[] = `listConnections -source on -destination off - plugs off
37                    -connections off -shapes off $relatives[0]`;
38 setAttr -l true ($source[0] +".radius");
39
40 connectAttr -f ($shadername+".BlackholeMass") ($inner_sphere+".scaleX");
41 connectAttr -f ($shadername+".BlackholeMass") ($inner_sphere+".scaleY");
42 connectAttr -f ($shadername+".BlackholeMass") ($inner_sphere+".scaleZ");
43 connectAttr -f ($outer_sphere+".translateX") ($inner_sphere+".translateX");
44 connectAttr -f ($outer_sphere+".translateY") ($inner_sphere+".translateY");
45 connectAttr -f ($outer_sphere+".translateZ") ($inner_sphere+".translateZ");
46
47 $shadername = `shadingNode -asShader lambert`;
48 sets -renderable true -noSurfaceShader true -empty -name ($shadername + "SG");
49 connectAttr -f ($shadername + ".outColor") ($shadername+"SG.surfaceShader");
50 setAttr ($shadername+".color") -type double3 0 0 0 ;
51 sets -forceElement ($shadername + "SG") $inner_sphere;
52
53 setAttr "defaultRenderQuality.enableRaytracing" 1;
54 int $tmp = `getAttr "defaultRenderQuality.reflections"`;
55 if( $tmp < 4 )
56         setAttr "defaultRenderQuality.reflections" 4;
57 $tmp = `getAttr "defaultRenderQuality.refractions"`;
58 if( $tmp < 4 )
59         setAttr "defaultRenderQuality.refractions" 4;
```

In line 1-3 a shader node is created, and two default values for the shader's input attributes are specified. The MEL command `shadingNode` creates a shading node of specified type and returns its name. To make a shader functional, a shading group has to be connected, which is done in line 5 and 6. In line 8-10 the outer sphere is generated by using the `sphere` command.

After the execution of the `sphere` command, the NURBS sphere is a selected object in Maya. To get the name of the sphere the command `ls` can be used. It lists the names of selected objects. Maya automatically assigns names to objects. If an object of the same type is created more than ones, a number is appended to the name. The access to objects in MEL commands is usually done using their names.

A NURBS sphere object in Maya consists of multiple nodes. A transform node, a shape node, a makeNurbsSphere node and a shader node. The node name which is returned by `ls`, if a NURBS sphere is selected, is the name of the transform node.

To prevent an intersection of the second shader ray when it leaves the outer sphere the *render stat* attribute `Double Sided` of the sphere's shape node has to be set to zero. `listRelatives` gives the name of the related shape node (line 12) when only the name of the sphere's transform node is known. The attribute is then set using the name of the shape node in line 13.

A similar procedure is applied to find the makeNurbsSphere node name using `listConnections`. Here we want to lock the attribute `Radius` so that only the scale values of the sphere's transformation node can be used to control its size. Only the shader attribute `Inifity Radius` should be used to control the size of the sphere. It will be connected to the sphere's scale attributes in the next step.

The relativistic black hole shader is then assigned to the sphere in line 19 and all connections described above are set up in lines 21-26.

The inner sphere is assigned a standard shader that generates a black surface in lines 47-51.

At the end some rendering attributes of Maya are set. In lines 53-59 raytracing is enabled and the maximum ray depth is set to 4. Wihout setting these parameters, a rendering using default values would cause the outer sphere to appear black.

The execution of the script creates a black hole with all its parts and configurations and integrates it in the dependency graph. Multiple black holes can be created be executing the script several times.

3.3.3 Integration in the GUI

A button that executes the script to generate a black hole can easily be added by loading the script code into the script editor, highlighting it with the mouse and draging it with the middle mouse button in the *shelf*. Maya automatically creates a button for the dragged code. A picture can be assigned to the button by using the *shelf editor*. *Figure 3.12* shows the shelf with the *create-black-hole button* on the right.

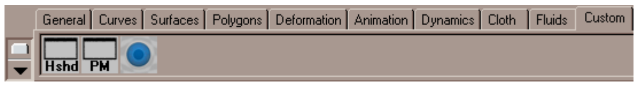
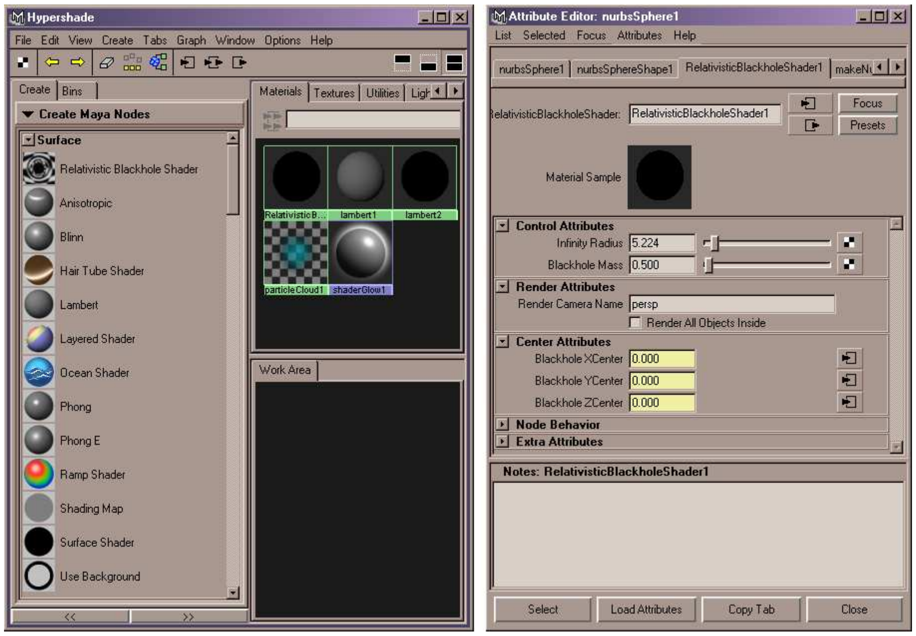


Figure 3.12: *The Shelf*

For the integration of the attributes of the shader plug-in in the general *attribute editor* a MEL script is the recommended way. It must be named `AE[name]Template.mel`[14], so the attribute editor template script for the relativistic black hole shader must be named `AERelativisticBlackholeShaderTemplate.mel`. This file has to be located in `../Maya/scripts/AETemplates/`. The script contains a global procedure that builds up the necessary GUI elements for the attributes of the shader (see script in Appendix B). The attribute editor contains the GUI elements for the shader's attributes (see *Figure 3.13*).

The integration in the shader window *hypershade* is done automatically by Maya. A picture for the button can be added by placing it in the folder `../Maya/icons/` with following name `render_RelativisticBlackholeShader.xpm`. The icon can be seen in the upper left corner of the left window of *Figure 3.13*.


Hypershade
File Edit View Create Tabs Graph Window Options Help
Create Bins
Create Maya Nodes
Surface
Relativistic Blackhole Shader
Anisotropic
Blinn
Hair Tube Shader
Lambert
Layered Shader
Ocean Shader
Phong
Phong E
Ramp Shader
Shading Map
Surface Shader
Use Background
Materials Textures Utilities Ligh
RelativisticB... lambert1 lambert2
particleCloud1 shaderGlow1
Work Area
Attribute Editor: nurbsSphere1
List Selected Focus Attributes Help
nurbsSphere1 nurbsSphereShape1 RelativisticBlackholeShader1 makeN
Focus
Presets
Material Sample
Control Attributes
Render Attributes
Center Attributes
Node Behavior
Extra Attributes
Notes: RelativisticBlackholeShader1
Select Load Attributes Copy Tab Close

RelativisticBlackholeShader: RelativisticBlackholeShader1

Infinity Radius 5.224

Blackhole Mass 0.500

Render Camera Name persp

Render All Objects Inside

Blackhole XCenter 0.000

Blackhole YCenter 0.000

Blackhole ZCenter 0.000

Figure 3.13: *Hypershade and the Attribute Editor for the Shader Node*

Now a user can create a black hole by pressing the create button and editing the parameters of the shader in the graphical attribute editor. The position of the black hole can then be modified by moving the outer sphere object. The black hole behaves like a standard Maya object and can be used as such. To animate it, for example, the standard key-frame procedure can be applied.

[14]AE stands for attribute editor

Chapter 4

Examples and Limitations

After having described the underlying technique and its implementation, in this chapter the plug-in is demonstrated. The visual output is explained by means of a simple scene: the relativistic living room. Also the limitations of the shader approach are explained.

After that, real examples of previous work on black hole visualisations is presented. Later, pictures of the shader approach are compared to pictures rendered using mathematically exact solutions. Finally the usage of the plug-in in a cross media opera project, performed in public at the "Wissenschaftssommer 2005 in Berlin"[1], is presented.

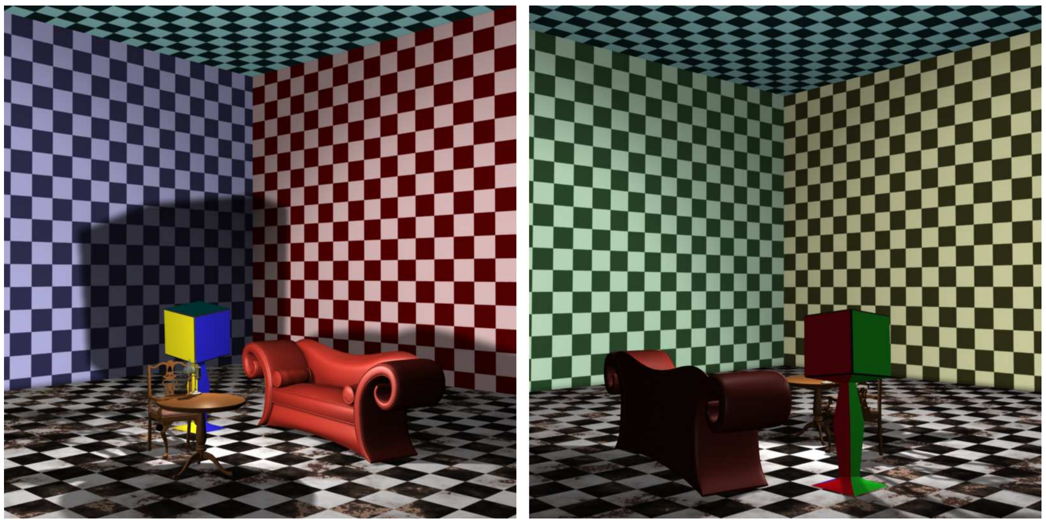


Figure 4.1: *The Relativistiv Living Room*

[1]Summer of Science

4.1 The Relativistic Living Room

The *Relativistic Living Room* is a scene showing a room with walls coloured in blue, yellow, red and green with a checker board texture. The ceiling is coloured cyan, the floor has a marble like texture in black and white. Some furniture is placed in it: a table, a chair, a sofa, a lamp and a box-like sculpture, each side coloured differently (see *Figure 4.1*). The aim was to generate a familiar 3D scene to make the bending effect accessible to everyday experience.

A closer look will reveal a black hole "growing" at the centre of the scene (see *Figure 4.2*). In the first picture everything looks perfectly normal. In the second picture at the right armrest of the sofa a tiny black hole is located.

The diagrams in the lower right corner of the pictures show the placement and relation of the inner and outer sphere and the camera schematically.

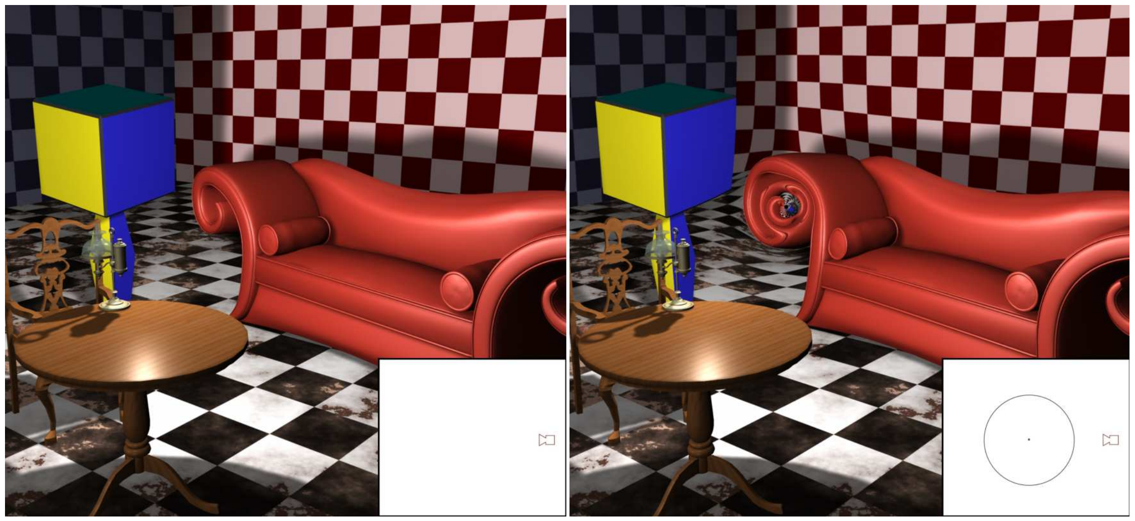


Figure 4.2: *Normal Living Room and Tiny Black Hole*

In *Figure 4.3* the mass of the black hole was increased. The gravitational lens effect can now be seen very well. Let us now have a look at the box-like sculpture. It can be seen twice from different viewing angels.

Notice that the right red wall now appears at the left hand side of the gravitational lens and the left blue wall at the right hand side. Also the ceiling (cyan) can be seen, that is not visible at all in the normal picture. Also you can see the floor directly above black hole.

The red sofa gets "bent around" the black hole and so forms a red "ring". This happens when objects are located behind the black hole. This phenomenon is called

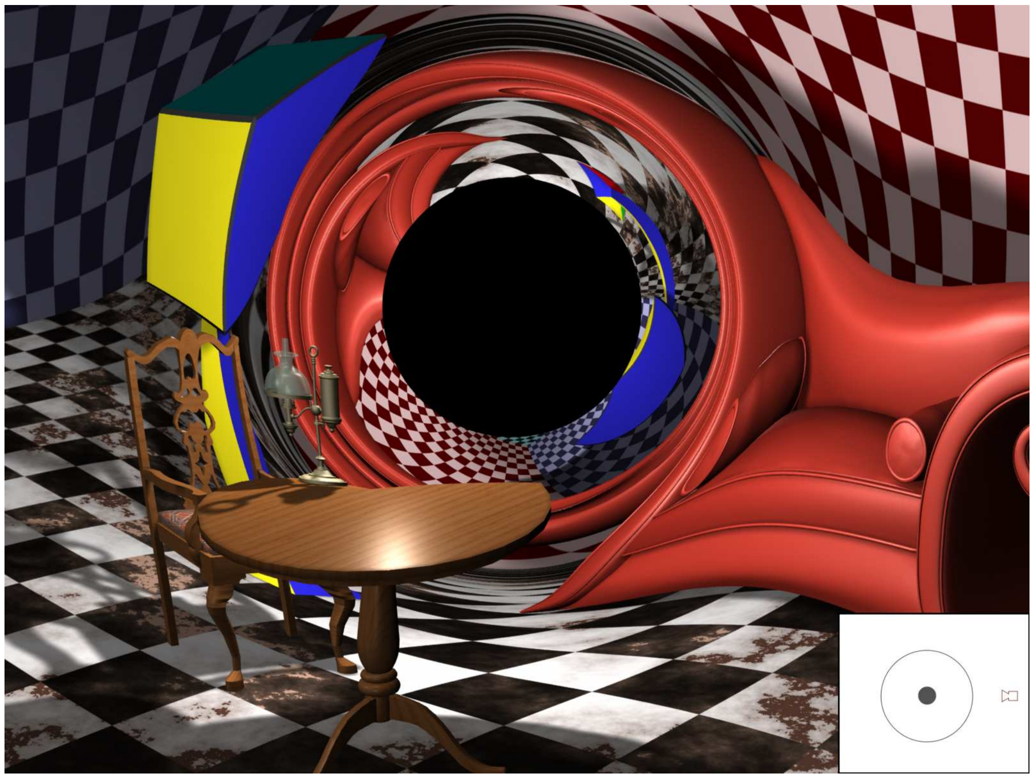


Figure 4.3: *Black Hole*

a “Einstein ring”.

We now construct two examples to show the limitations of the applied technique (see *Figure 4.4*). In the left picture the outer sphere is small in relation to the inner sphere. We see that there is a discontinuity at the border of the sphere despite the linear correction.

In the right picture a cube is added to the scene. The cube intersects the inner sphere and is placed in a way that first and also second order shader rays intersect the cube. This causes a discontinuity between the pixels that are intersected by the first order shader ray located next to the pixels intersected by the second shader ray, what can be seen at the edges of the green and the yellow side of the cube.

4.2 Previous Work

The first work on computing pictures containing black holes that I found, was done by Robert J. Nemiroff[2] in 1991. He published articles 1991 and 1993 in the American

[2]Professor at the Department of Physics, University Michigan

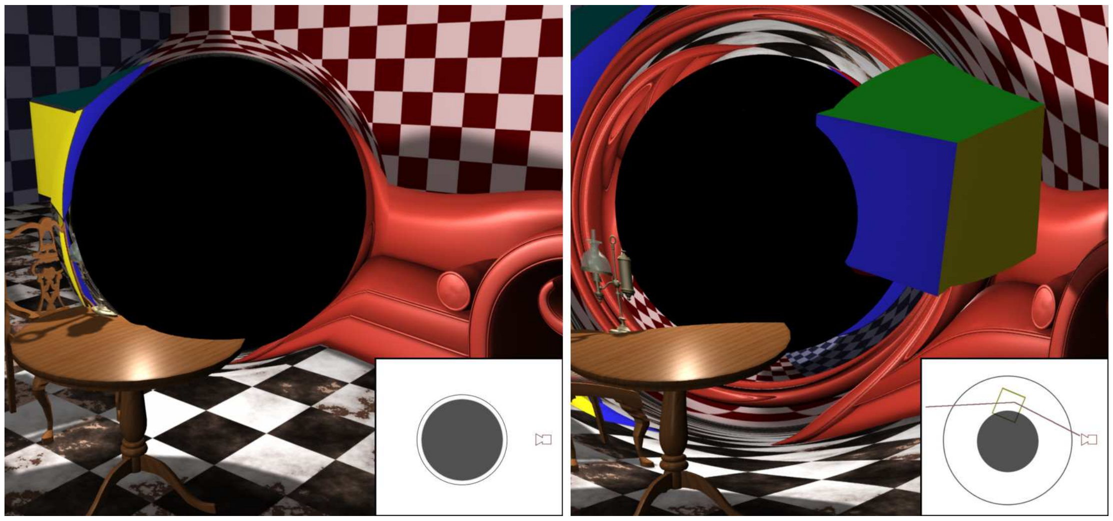


Figure 4.4: *Limitations*

Journal of Physics (see [Nem91] and [Nem93]). He also generated movies showing the approach to a black hole, orbiting a black hole etc. The images he created are not very appealing but quite understandable. An example of a scene without and with a black hole created by him is shown in *Figure 4.5*. The differently coloured disks represent stars. The left picture shows the scene without a black hole, the right one with a black hole in the centre. Notice that each star appears twice on opposite sides of the black hole's centre.

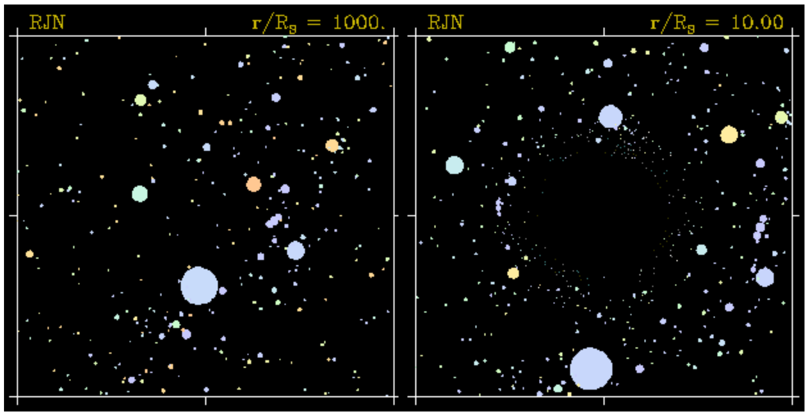


Figure 4.5: *Black Hole, Robert J. Nemiroff*

In the same year (1991) Corvin Zahn wrote a diploma thesis on relativistic

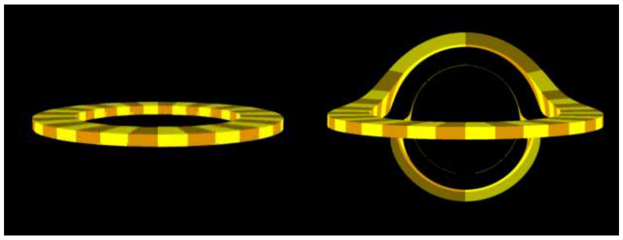


Figure 4.6: *4D Raytracing, Corvin Zahn*

4D raytracing at the University of Tübingen[3] [Zahn91]. He implemented his own raytracing engine using a mathematically exact solution and created pictures and movies of neutron stars and black holes. One of the movies shows a ring that rotates around a black hole. *Figure 4.6* shows the ring in a "normal" scene (left) and in a relativistic scene containing a mass in the centre of the ring (right). Some rays are bent so strongly that the backside of the ring can be seen.

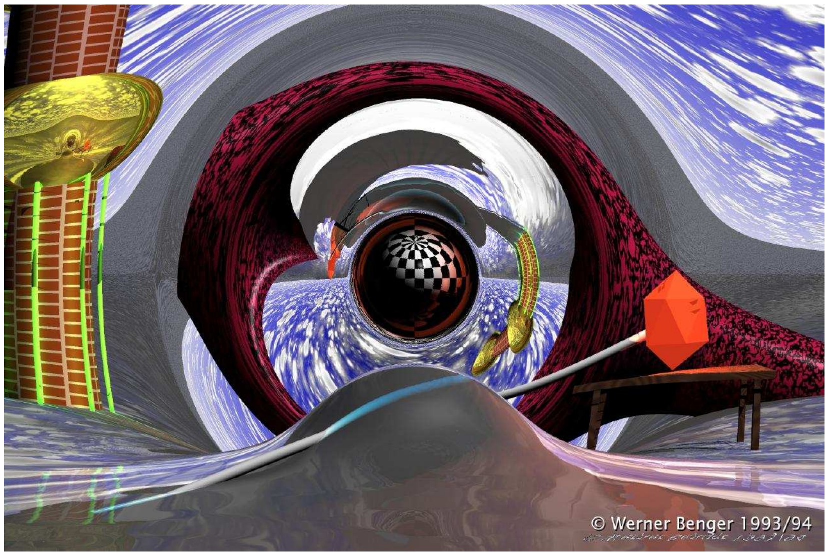


Figure 4.7: *Black Hole, Werner Benger*

Werner Benger[4] made first experiments based on equation 2.25 in 1992 with his raytracing engine Light++. He then implemented a exact solver for simulating curvature of space and created a scene containing a black hole in 1993 (see [Ben96]). Although primarily developed to study gravitational effects Light++ offers most

[3]Department for Theoretical Astrophysics

[4]Laboratory for Creative Arts and Technology, Center for Computation Technology at Louisiana State University

of the standard and many non-standard features. With this tool pictures of more appealing 3D scenes in curved space can be rendered. The scene in *Figure 4.7* shows a black hole that is surrounded by a black and white checkered membrane, which is a little bit bigger than the event horizon. Almost the whole surface of the membrane can be seen. The backside of the membrane is illuminated by the red icosaeder on the right hand side. Werner Benger is still working on the raytracer and renderings of black holes. Since 1993 a lot more features were added.

Andrew Hamilton[5] wrote some web pages in 1997 describing the optical effects when falling into a black hole. His pictures are very schematic but also very informative. He has added detailed descriptions to the pictures, [Ham97].

Later he started to develop a real time flight simulator that uses real-time graphic techniques to simulate black holes. The pictures that are produced with the flight simulator are impressive. He can simulate symmetrical Schwarzschild solutions and also electrically charged black holes in real time. *Figure 4.8* shows a view after having passed the singularity in an imaginary universe where electric charge is imaginary. He describes: "Including parallel universes, no less than four distinct universes are visible from this vantage point."

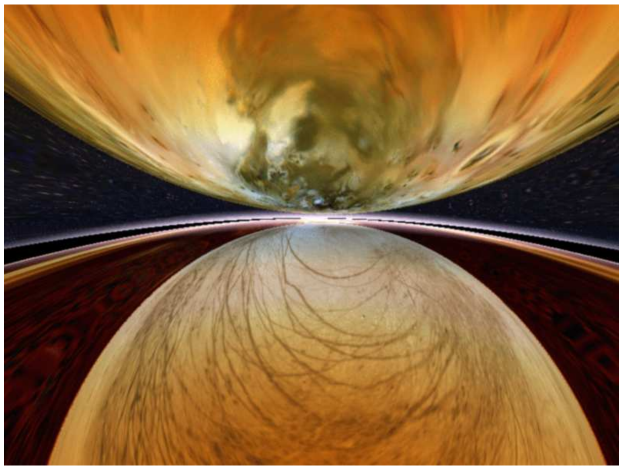


Figure 4.8: *Black Hole Flight Simulator, Andrew Hamilton*

4.3 Exact versus Approximated Solutions

Rays of light that in reality would be strongly bent or would orbit around the centre of a black hole a few times before they escape from it cannot be simulated with the

[5]Professor at the Department of Astrophysical and Planetary Sciences, University Colorado

Maya plug-in. An exact solution uses curved light rays, for example implemented by a piecewise linear approximation. Since the according differential equations have to be solved this leads to longer computation times. The advantage of these methods is of course that strongly curved rays as well as rays that orbit the black hole can be simulated exact.

Let us now compare pictures of similar scenes created with exact solution tools and the with simple plug-in approach by means of two examples. The first example shows a ring similar to the scene created by Corvin Zahn rendered in Maya. The second example shows a scene rendered in Werner Benger's *Light++* and a similar scene rendered in Maya.

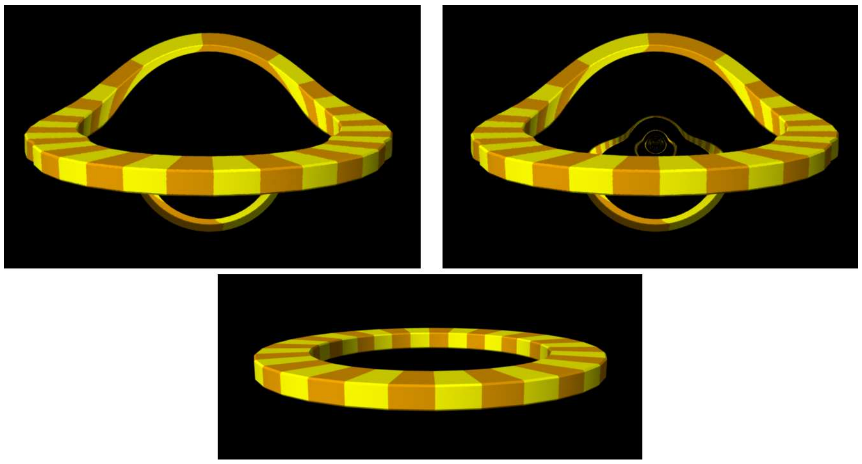


Figure 4.9: *Ring around Black Hole*

If we have a look at *Figure 4.9* we notice, that the overall deformation of the ring occurs in a similar fashion as in the exact solution shown in *Figure 4.6*. The second image of the ring also occurs, but there is a difference compared to the exact solution. The second image of the ring occurs all the time in the movie of Corvin Zahn even if the ring is perpendicular to the image plane. This is because light rays are bend by 180° around the black hole. This is not possible with the approximation approach, because the inner sphere is hit when the deflection angle δ gets that large. When removing the inner sphere the second image is completely visible, as shown in the right picture of *Figure 4.9*. One can again see the discontinuities caused by the kink of the ray when trying to get a similar deformation like in the exact solution.

The second example shows a picture by Werner Benger of a cube placed around a black hole. I created a similar scene using Maya. *Figure 4.10* shows Werner Benger's picture on the left and mine on the right. Again, taking a closer look at the edges

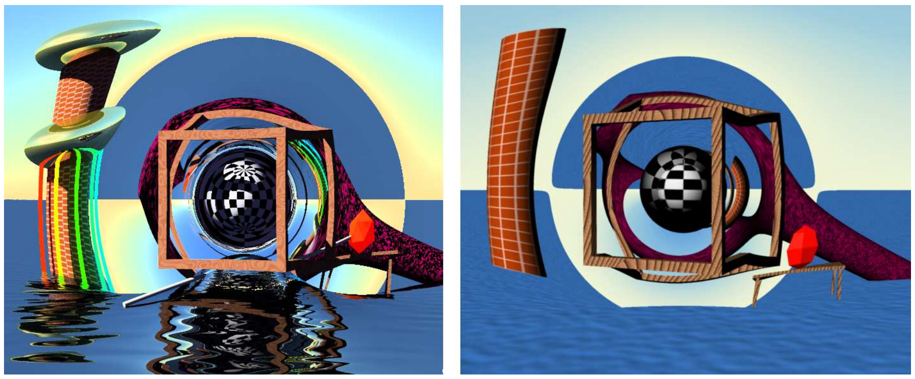


Figure 4.10: *Cube around Black Hole*

of the cube, we can see the discontinuity caused by the kink. Another effect also caused by the kink can be seen at the sphere in the centre of the image. Here the brightness of the sphere changes in a discontinuous circle. Also the sphere appears to be smaller in the right picture. Again the visual outcome is similar to the exact solution, but objects must not be located too near to the centre of the black hole.

Knowing the limitations and taking them into account when designing a scene it is possible to get fast nice results with the plug-in.

4.4 "C - The Speed of Light"

"C - The Speed of Light" is a cross media opera that was performed at the "Wissenschaftssommer 2005"[6] in Berlin. It was organised by the company "Wissenschaft im Dialog"[7] [WID06], which was founded to communicate science to the public in Germany. The program in 2005 was especially dedicated to Einstein.

Together with Phase-7 [Ph706], "Wissenschaft im Dialog" created a multi-media opera on Einstein and his ideas that influenced science so much. Phase-7 is a creative arts company specialised in developing and performing multi-media shows. They combine classic forms of art, like dance and song, with modern media. In the opera they brought together three singers, two dancers and a violinist and digitally created classical music using the vienna sound sample library. For the backdrops they used projections onto a 360° screen, showing rendered videos, real-time video effects and real-time 3D graphics.

Phase-7 wanted to use scientifically correct video material for the backdrop

[6]Summer of Science 2005

[7]Science in Discourse

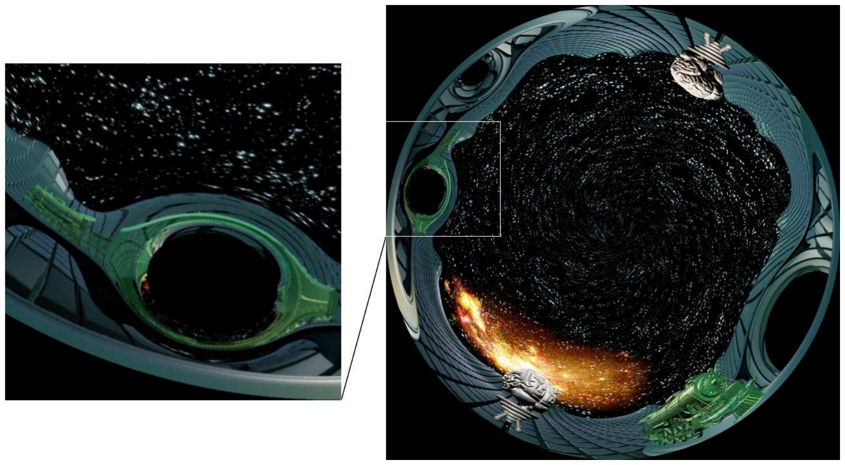


Figure 4.11: *Picture of Opera Animation*

matching the topic closely. They collaborated with Werner Benger from the visualisation department of the Konrad Zuse Institute for Information Technology Berlin. They found out about my work on a Maya plug-in through this collaboration and asked me whether I could contribute a 30 second video clip showing black holes.

Together with Alexander Senfter from Living Liquid Software [Liv06] I designed and created a scene suiting the multi-media spectacle within two weeks. The video had to be in fish-eye projection to match the projection setup and equipment used to fill the 360° screen. We rendered five camera views (front, back, left, right and top) and stitched them together with a software tool provided by the company from that Phase-7 hired the 360° screen. Every single picture has a resolution of 2048×2048 pixels. The scene shows a landscape formed out of box-like objects, where two steam trains orbit black holes. *Figure 4.11* shows a picture of the final video (right) and a detailed view of a train behind two black holes (left). On one side a train orbits the black hole, on the other side a train moves around two black holes, whereby the two black holes are orbiting around a common centre. This leads to an quite visually impressive appearance.

The idea of using old fashioned steam trains in the animation clip was born during a discussion of Werner Benger with Sascha Rieger and Susanne Milde from MildeMarketing [Mil06].

The opera was performed for ten days in Berlin at the Bebelplatz (in front of the "Staatsoper unter den Linden"), next to the Einstein exhibition running over a longer period in this year. The spectacle was received very well by the audience

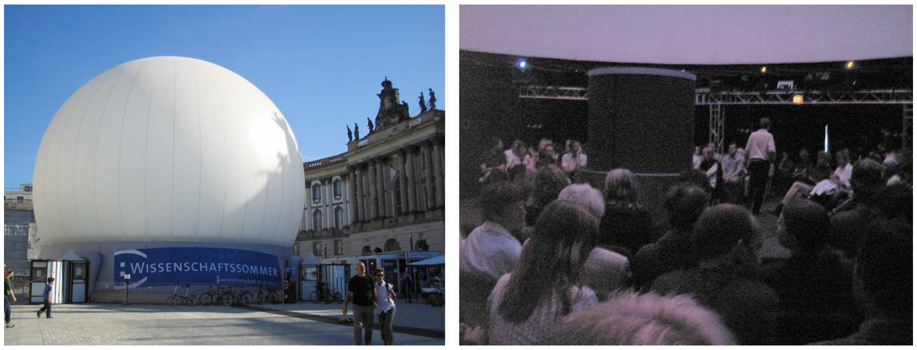


Figure 4.12: *Out- and Inside the Dome*

and the number of visitors increased every day. Of course Alexander and I had to experience it ourselves. *Figure 4.12* and *Figure 4.13* give an impression of the event.

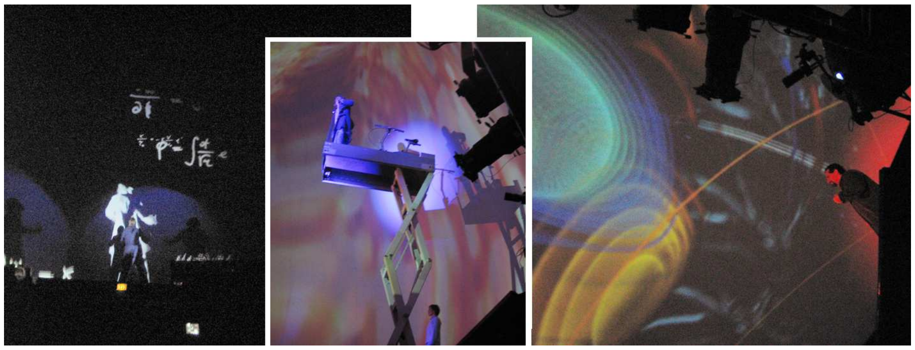


Figure 4.13: *Impression of the Performance*

Chapter 5

Future Work

This work can be extended in several ways: using a better approximation for the curved rays for example. A stepwise linear function would make the area closest to the black hole look more realistic. It also would be possible to implement an exact solution in the shader. That would not solve the problem of the seamless integration in the scene, though.

Correct red and blue shift would be a nice extension. Together with this implementation an interesting problem is to find a satisfying mapping between wavelength, i.e. a colour in nature, and an RGB colour value used for the internal colour representation.

Porting the shader to Mental Ray would be of much use, because this rendering engine is superior to the software rendering engine of Maya and is integrated in different professional 3D packages. Thus, the shader also could be employed by users that are familiar with Softimage or 3dsMAX.

Appendix A

Acknowledgements

I want to thank Werner Benger, who was the one who brought me in contact with black holes. He is an expert in the field of General Relativity and scientific visualisation. His enthusiasm for photo realistic rendering of black holes is so contagious, that it is hard to get not infected somehow. After some inspiring discussions I decided to find a way to create pictures of black holes described by the Schwarzschild solution using a popular 3D application.

Christian Vogt was my tutor at the Institute of Computer Science at the Research Group "Computational Logic". I would like to thank him for his guidance of my work. Besides pure logic and mathematical topics he is very interested in problems related to physics and so was happy to accept my suggestion as topic of my bachelor thesis. He has a wonderful way of organising and working on things. This helped me a lot in structuring and finishing my work.

Special thanks go to my supervisor Aart Middeldorp, head of the Research Group "Computational Logic", for his help in getting financial support for the participation in the "2nd Highend Visualization Workshop 2005" in Obergurgl, where I presented my work.

Finally, I want to thank the people of the company Living Liquid Software who allowed me to use their software licenses to create the plug-in for Maya. Also I want to thank Stephan Prohaszka for discussing some implementation related issues and Nikolaus Stickler, who allowed and supported this kind of cooperation.

In conclusion, I thank Alexander Senfter, from Living Liquid Software too, who supported me in the cross-media opera project.

Figure A.1: *Living Liquid*

Appendix B

MEL-Source Code

Listing B.1: AERelativisticBlackHole.mel

```
global proc AERelativisticBlackholeShaderTemplate( string $nodeName )
{
        AEswatchDisplay $nodeName;

        editorTemplate -beginScrollLayout;

        editorTemplate -beginLayout "Control Attributes" -collapse 0;
                editorTemplate -addControl "ir";

        editorTemplate -endLayout;

        editorTemplate -beginLayout "Render Attributes" -collapse 0;
                editorTemplate -addControl "cam";
                editorTemplate -addControl "rao";
        editorTemplate -endLayout;


        editorTemplate -beginLayout "Center Attributes" -collapse 1;
                editorTemplate -addControl "bxc";
                editorTemplate -addControl "byc";
                editorTemplate -addControl "bzc";
        editorTemplate -endLayout;

        // include/call base class/node attributes
        AEdependNodeTemplate $nodeName;

        editorTemplate -addExtraControls;
        editorTemplate -endScrollLayout;
}
```

Appendix C

Final Notes

This document was created with LaTeX.

All trademarks used are properties of their respective owners.

A pdf of the thesis can be found at
$http://www.rittertec.at/marcel/bakk/bakk\_bh.pdf$.

Information about the up to date version of the plug-in can be found at
$http://www.rittertec.at/blackhole$.

Bibliography

[Ben96] Werner Benger, *Relativity and Scientific Computing - Computer Algebra, Numerics, Visualization*, Springer Verlag, Berlin Heidelberg New York, 1996: 2-3

[Ben04] Werner Benger, *Tensor Field Visualisation via a Fiber Bundle Data Model*, Department of Mathematics and Computer Science, University of Berlin, 2004

[DED20] F. W. Dyson, A. S. Eddington, and C. Davidson, *A Determination of the Deflection of Light by the Sun's Gravitational Field, from Observations Made at the Total Eclipse of May 29, 1919* Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character (1920): 291-333, on 332

[Ein16] Albert Einstein, *Die Grundlage der allgemeinen Relativitätstheorie*, Annalen der Physik, Volume 354, 769-822, 1916

[Fol97] James D. Foley, Andries van Dam, Steven K. Feiner, John F. Hughes, *Computer Graphics* PRINCIPLES AND PRACTICE, Addison-Wesley, 1997

[Gou03] David A. D. Gould, *Complete Maya Programming*, Morgan Kaufmann Publishers, Elsevier Science, 2003

[Ham97] Andrew Hamilton, http://casa.colorado.edu/ ajsh/schw.shtml, 1997-2001

[Liv06] Homepage of Living Liquid Software GmbH, http://www.livingliquid.com, Innsbruck, 2006

[MH65] *Maya Help for Maya 6.5*, Alias Systems, 2005

[Mil06] Homepage of Milde Marketing Wissenschaftskommunikation, http://www.milde-marketing.de, Potsdam, 2006

[Nem91] Robert J. Nemiroff, *Trip to a Neutron Star: The Movie*, Bull. Am. Astron. Soc. 23, 1418 (1991)

[Nem93] Robert J. Nemiroff, *Visual Distortions Near a Black Hole and Neutron Star*, American Journal of Physics, 61, 619+, 1993

[Nob05] *The Official Web Site of the Nobel Foundation*, Nobel Foundation, http://nobelprize.org/physics/laureates/1921, 2005

[Ph706] Sven Sören Beyer, Homepage of phase-7 performing.arts, http://www.phase-7.de, Berlin, 2006

[Sch03] B.F. Schutz, *Gravity from the Ground Up*, Cambridge University Press, 2003

[Str91] Norbert Straumann, *General Relativity and Relativistic Astrophysics*, Springer-Verlag, 1991

[WID06] Homepage of Wissenschaft im Dialog gGmbH, http://www.wissenschaft-im-dialog.de, Berlin, 2006

[Zahn91] Corvin Zahn, *Vierdimensionales Raytracing in einer gekrümmten Raumzeit*, Diplomarbeit 1991, Universität Stuttgart

List of Figures