Master thesis for the degree of

MASTER OF ENGINEERING

in Audiovisual Media

Faculty of Electronic Media

of Hochschule der Medien in Stuttgart

A Comparison of Render Engines in Nuke

presented by

Konstantin Holl (Matr. No.: 33942)

supervised by

Prof. Katja Schmid, examiner

Prof. Benjamin Seide, co-examiner

November, 2018, Stuttgart, Germany

Declaration of Authorship:

I hereby declare that the thesis submitted is my own unaided work. All direct or indirect sources used are acknowledged as references. I am aware that the thesis in digital form can be examined for the use of unauthorized aid and in order to determine whether the thesis as a whole or parts incorporated in it may be deemed as plagiarism. For the comparison of my work with existing sources I agree that it shall be entered in a database where it shall also remain after examination, to enable comparison with future theses submitted. Further rights of reproduction and usage, however, are not granted here. This paper was not previously presented to another examination board and has not been published.

Date: _____ Signature: _____

Abstract

In this thesis, a comparison of the available ray tracing renderers for the compositing software Nuke is performed. To see the differences of the comparison the renderers are integrated in the compositing pipeline with a case study. The goal is to create integrations of 3D elements while rendering in a compositing software instead of a conventional 3D software. In an empirical study experts from the compositing area were surveyed regarding the benefits they see in rendering in Nuke.

Kurzfassung

In dieser Thesis wurde ein Vergleich der verfügbaren raytracing Renderer für die Compositing Software Nuke erstellt. Um die Unterschiede der Ergebnisse aus dem Vergleich deutlich zu machen, wurden die Renderer in die Compositing Pipeline eingebaut und anhand einer Fallstudie ausgewertet. Das Ziel der Thesis ist es, 3D Elemente in Kameraaufnahmen nahtlos einzubetten und dabei das Rendering dieser Elemente in der Compositing Software anstatt, wie herkömmlich, in der 3D Software zu erstellen. In einer empirischen Studie wurden Fachleute aus dem Bereich Compositing befragt, wie groß der Nutzen des renderns in Nuke ist und welche Ansprüche sie an das Verfahren haben.

Acknowledgment

First of all, I want to thank my professor Katja Schmid, who gave me a lot of creative freedom to realize this thesis. I am grateful for the input and helpful discussions along the way to create this thesis. I also want to thank Benjamin Seide, who supported me throughout my work on this thesis. He provided valuable advice for the implementation and practical part of this thesis. Furthermore, I want to thank both for their contacts of professionals who answered the survey and thus helped me to gain further valuable insights. Last but not least, I want to thank Susanne Fendt and Dominik Holl for the proofreading of this thesis.

Contents

1 Introduction 10

1.1 Background 10

1.2 Question of issue 11

1.3 Outline 11

2 Theory 12

2.1 Pinhole camera 12

2.2 Ray tracing 14

2.3 Ray optics 16

2.4 BRDF 17

2.5 Distributed ray tracing 20

2.6 Global illumination 22

2.6.1 Path tracing 23

2.6.2 Photon mapping 24

2.7 Render software 26

3 Comparison of render software 30

3.1 Selection of render software 30

3.1.1 Features 30

3.1.2 Integration 33

3.2 Comparison 36

3.2.1 Image Quality 36

3.2.2 Ray depth 39

3.2.3 Motion Blur 40

3.2.4 Depth of field 47

3.2.5 Render times 50

3.2.6 Materials & Shadows 52

3.3 Compositing 53

3.3.1 VRay for Nuke 53

3.3.2 RayRender 55

3.3.3 VRay for Maya 57

3.4 Conclusion 59

4 Discussion 61

4.1 Data acquisition 61

4.2 Analysis of the survey 62

4.3 Conclusion of the discussion 64

5 Conclusion 65

5.1 Summary 65

5.2 Future work 65

5.3 Conclusion 66

6 References 67

Attachment 71

1 Introduction

The visual effects industry brings astonishing scenes to the big screen or television with beautiful landscapes, cars, or creatures that do not exist outside of the computer. The job of a compositor is to seemingly integrate those computer generated objects in the footage, that was shot on set, to make it look like it was all shot together (Fitzgerald 2018). While computers are getting faster, rendering becomes more complex. Global illumination for example, is a great way to make a 3D scene look very realistic nowadays, but it is a calculation heavy process. A compositor has to wait until the 3D elements he needs are rendered before he can integrate them in the footage. Changing those elements afterwards takes additional time where the compositor waits until he can proceed. Nowadays compositing software have a 3D space integrated and some render engines are already implemented to make use of this space. Rendering in a compositing software such as Nuke can provide a lot of flexibility for last minute changes.

The aim of this thesis is to determine, which ray tracing renderer works best within Nuke by comparing and evaluating the render systems with one another. The results should give an overview of the renderer and a scope for the individual preference of use. It should also bring the question to the attention of companies who could benefit from the flexibility during tight production pipelines.

1.1 Background

Rendering and compositing have always worked hand in hand but have been done in different software packages and handled by different kinds of artists. Since Chaos Group has announced *VRay for Nuke* 2015 (Seymore 2015), compositing has become more flexible. Since then not much has changed. VRay is the leading ray tracing renderer for Nuke. Even though Octane offers a plug-in, no other renderers have made the transfer to Nuke so far. Since 2016, Foundry offers an own option for ray trace rendering with RayRender (Wright 2016), which uses the common shaders and lights of Nuke.

1.2 Question of issue

This thesis will discuss the following questions of issue:
- How are the available ray tracing renderers in Nuke used in projects and what limitations do they have?
- What are the benefits and drawbacks of rendering in a compositing software such as Nuke compared to the traditional workflow with renderings from 3D software such as Maya.

1.3 Outline

In chapter 2 a brief history of ray tracing is given, which is the foundation the comparison is based on. It is explained how light travel works in and on different materials in the real world and how ray tracing tries to simulate this. Furthermore, the evolution of the ray tracing algorithm leads to creating desired effects of real world cameras such as motion blur and depth of field. Global illumination gives the foundation of today's photo-realistic renderings and uses either path tracing or photon mapping.
In chapter 3 the features of the render engines are evaluated and afterwards a comparison between the renderers is made according to the features they unite. In addition two of the four renderers are tested and compared to provide additional insights into the performance, with a real world example, of the integration in the compositing pipeline.
In chapter 4 a survey is performed in order to give insights on how many professionals are using ray trace rendering in Nuke as well as their expertise on the topic.
Subsequently, a conclusion of the comparison and the survey is given, as well as possible future work to the current proposal.

2 Theory

Modern 3D renderings look photo realistic. To understand how this is achieved, it is crucial to understand how real photos are developed and how light travels into a camera and to the sensor which captures the picture. This knowledge can then be translated and simplified to create a virtual camera for rendering 3D scenes. The results look similar to those in the real world. Through those simplifications, which are necessary to make up for computation time, certain aspects of the real world light travel are lost. Therefore, other approaches need to be made to compensate for that. To see how light rays behave on different surfaces and through different mediums, ray optics need to be considered. To perceive how light travels in the real world, the simplest form of a camera, the pinhole camera, is observed below.

2.1 Pinhole camera

A pinhole camera is a closed, light-proofed box or chamber with a tiny hole (pinhole) on one side. Light rays travel through the hole and cast an image of the outside world, horizontally as well as vertically inverted, on the opposite wall of the chamber. This optical phenomenon is called camera obscura (lat.: dark room) (Glassner 1989; Scratchapixel 2014).

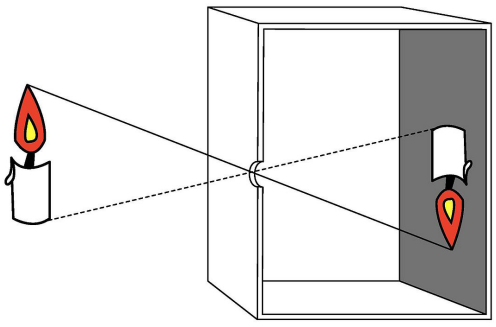

Figure 2.1.: Illustration of the camera obscura.
(Image source: https://en.wiktionary.org/wiki/pinhole_camera, 08.07.2018)

If distinctive parts of the projected images are connected with the real objects through the pinhole, the path of the individual rays can be reconstructed. This simplified model of real-world light travel is the basis for computer generated physically correct lighting. The

light in the real world comes from various light sources and spreads across the scene in different directions. Some of the light rays enter the camera through the pinhole (Glassner 1989; Scratchapixel 2014).

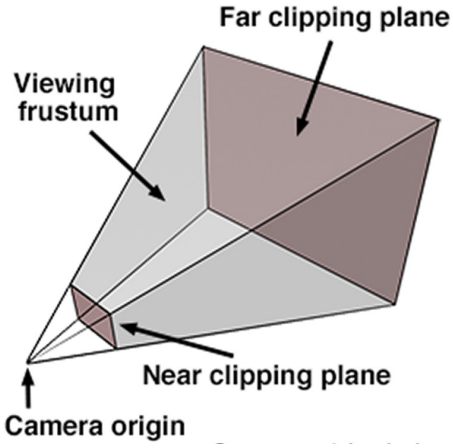

Figure 2.2.: A virtual pinhole camera as used in most 3D applications.
(Image source: https://www.scratchapixel.com/lessons/3d-basic-rendering/3d-viewing-pinhole-camera/virtual-pinhole-camera-model, 08.07.2018)

A virtual camera for 3D applications can also be described with a sort of pinhole camera. To simplify things the pinhole is reconstructed to be at the position of the camera origin (or eye) and the projection plane is moved in front of the camera so that no inversion of the picture takes place. Defining the virtual camera that way leads the camera to be a pyramid with a far and a near plane (see figure 2.2). Everything in between those planes, called frustum, is visible on the screen. Objects outside of the frustum are not visible or clipped off. Every light ray in the scene, that aligns with the eye, leaves a point on the canvas, thus constructing the image we see on screen (see figure 2.3) (Glassner 1989).

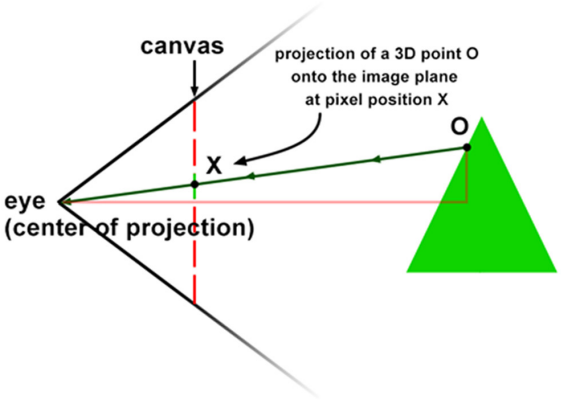

Figure 2.3.: Projection of the 3D scene on a 2D canvas or screen.
(Image source: https://www.scratchapixel.com/lessons/3d-basic-rendering/3d-viewing-pinhole-camera/virtual-pinhole-camera-model, 08.07.2018)

This visualisation of the 3D scene projected to the screen is a simplification of the rasterization algorithm. Rasterization and ray tracing are the two main rendering techniques in computer graphics (CG). To get a full picture of the scene, it is crucial to determine each pixel's color. Rasterization casts rays from the geometry to the eye to find and color in the pixels where these rays hit the image plane. For each pixel the color values of all the rays that hit the pixel are averaged together (Scratchapixel 2014).
In case of ray tracing, a ray leaves the light source, bounces around in a scene, hits the eye, and colors a pixel of the image plane. The color value results from the objects it hits while bouncing. A huge issue with that kind of ray tracing just described is that there would be a lot of rays cast without ever hitting the eye. A lot of computing therefore would be without purpose and it could take hours to get a well lit picture (Glassner 1989). In detail ray racing is more complex than that.

2.2 Ray tracing

Instead of tracing a ray forward, a better approach would be to trace it backwards. That means the tracing starts at the eye, travels through the individual pixel of the image plane, and into the scene. Thus, only rays that are crucial to the picture are considered.
"Because forward ray tracing is so expensive, the term 'ray tracing' in computer graphics has come to mean almost exclusively backward ray tracing (Glassner 1989, S.9)."
A ray that illuminated a pixel is easily found by connecting the line between the pixel and the eye and extending it into the scene. There it hits an object and bounces in different directions depending on the surface, or even through the object until it hits the light source from which it originated (Glassner 1989).

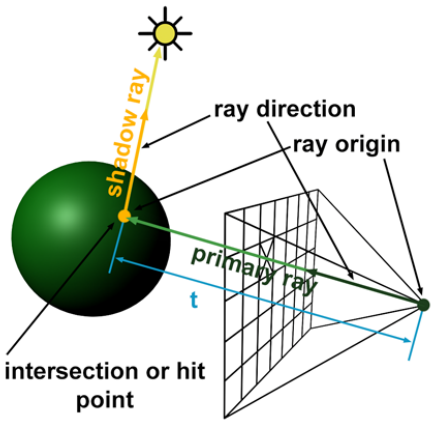

Figure 2.4.: Ray tracing algorithm.
(Image source: https://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-generating-camera-rays, 13.07.2018)

Therefore, four different rays are considered. *Primary rays* (also pixel or eye rays) are the first rays traveling from the eye through a pixel in the scene. *Shadow rays* travel from the object to the light source, where the ray needs to end in order to be considered a light ray. *Reflection rays* (also secondary rays) spread in different directions from a surface according to its material properties. *Refraction rays* (or transparency rays) travel through an object and get refracted according to the medium the material is made of (Glassner 1989).

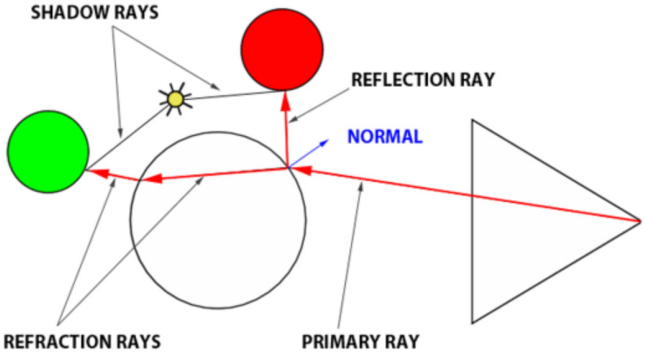

Figure 2.5.: Ray tracing algorithm with different rays.
(Image source: https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-ray-tracing/adding-reflection-and-refraction, 13.07.2018)

To determine if a ray, sent from the eye, lies in shadow or is fully illuminated, a shadow ray is cast to the nearest light source. As depicted in figure 2.6, if the shadow ray hits the light source without intersecting any opaque object this area is fully illuminated by that light source. If it hits an object on its path the area is covered in shadow (Glassner 1989).

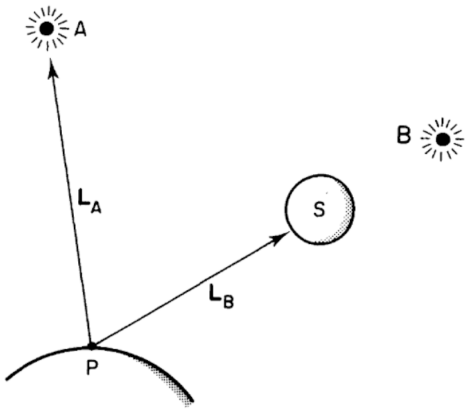

Figure 2.6.: Shadow rays from Point P with and without an object in between the path to the light source.
(Image source: Glassner, 1989, S.11)

Reflection and refraction rays are dependent on the surface material properties. Therefore, it is important to have a look at the light transport on different surfaces and through different mediums in the real world.

2.3 Ray optics

Ray optics or geometrical optics can be described as propagation of straight lines. This way reflections and refractions can be explained as followed:
If a light ray hits an opaque surface, some of the energy gets absorbed. The remaining part bounces off the surface in an angle which depends on the surface normal. Mathematically speaking, the angle of incidence equals the angle of reflection (Bühler et al. 2017; Palffy-Muhoray 2017).

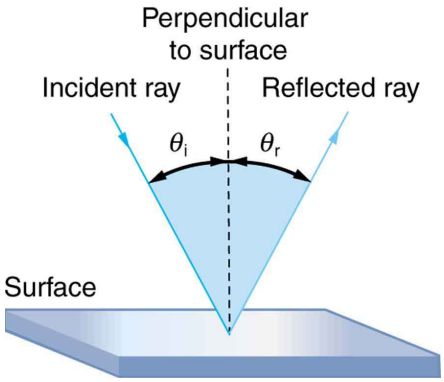

Figure 2.7.: Ray optics law of reflection: angle of incidence equals angle of reflection.
(Image source: https://courses.lumenlearning.com/physics/chapter/25-2-the-law-of-reflection/, 16.07.2018)

If the surface of the object is not mirror-like the reflected rays scatter in different angles off the surface and diffuse the light (see figure 2.8). If the surface of the object is transparent the light ray enters the surface and, depending on the medium it enters, the ray gets bent. If the new medium has a higher optical density than the old one, the ray bends towards the surface's normal. If the optical density is lower, it bends away from the surface normal

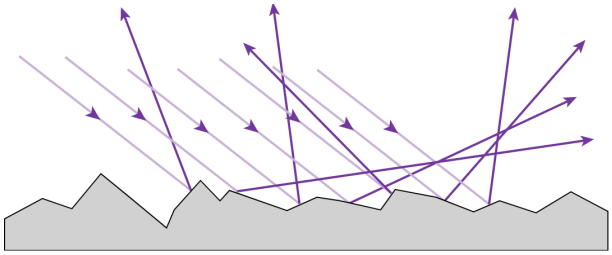

Figure 2.8.: Diffuse reflection of a rough surface.
(Image source: https://hedberg.ccnysites.cuny.edu/viewers/ebook.php?course=introphysics&topic=ray-optics&l=cal, 16.07.2018)

(see Figure 2.9). This density is distinguished as index of refraction (IOR) and refers to the speed of light traveling through different mediums (Bühler et al. 2017). For example, air has an IOR of 1.0, water 1.33 and diamond 2.42 (Gugick 2017).

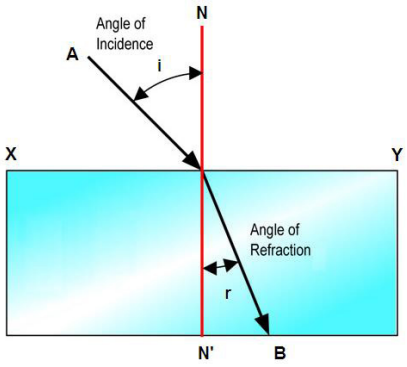

Figure 2.9.: Ray optics law of refraction.
(Image source: https://physics.stackexchange.com/questions/37731/refraction-reflection-and-what-is-total-reflection, 16.07.2018)

To gain a better understanding of how light reacts on different kinds of material surfaces, a real-world approach to light reflecting on surfaces is shown in figure 2.10. With ray optics the ray tracing algorithm can be built to distinguish rays that hit reflective or translucent surfaces.

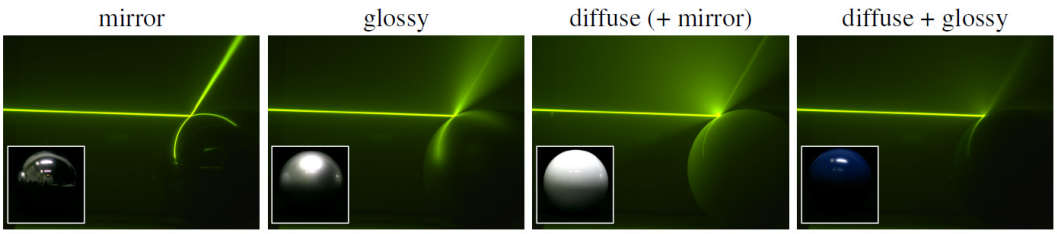

Figure 2.10.: Laser in a water tank hits spheres with different kind of surface materials.
(Image source: Hullin et al. 2008, S4)

2.4 BRDF

To reflect light off a surface, the material of that surface has to be defined. This is achieved with a bidirectional distribution function which describes an object's absorption, reflection, or scattering between incoming and outgoing illumination at a given point on the surface (Seymour 2012). BRDFs are used to improve realism in computer graphics.
"A BRDF is nothing else than a function that returns the amount of light reflected in the view direction for a given incident light direction (Scratchapixel 2015 a)."

A BRDF has three rules to maintain to be a approximation of physically correct lighting. The function must be positive everywhere over the range of valid incoming and outgoing directions. It also must be reciprocal, which means if the incoming and outgoing directions are swapped, the function still gives the same result. And lastly, a BRDF needs to be energy conserving. Like in physics, a surface can not create more light than it receives (Scratchapixel 2015 a). The degree to which light is reflected depends on the viewer's and the light's position relative to the surface normal and tangent. Additionally, when light interacts with a surface, different wavelengths of light may be absorbed, reflected or transmitted, which changes the color and the kind of the material. A BRDF also varies with the spatial position. That means, that light interacts differently with different regions of the material, i.e. the ringing and stripping patterns of wood. So, in general a BRDF needs to consider the incoming and reflected light direction, the spectral composition of light and the positional variance reaching the observer. Sometimes the positional variance is not included in the BRDF. It is then referred to as a position-invariant BRDF (Wynn n.d.). The bidirectional part of the BRDF means that there are two components, a specular and a diffuse. The specular component represents light that is reflected from the surface of the material. The diffuse component originates from internal scattering or from multiple surface reflections, if the surface is really rough. Both components can have different colors, if the material is not homogeneous (Cook and Torrance 1982).

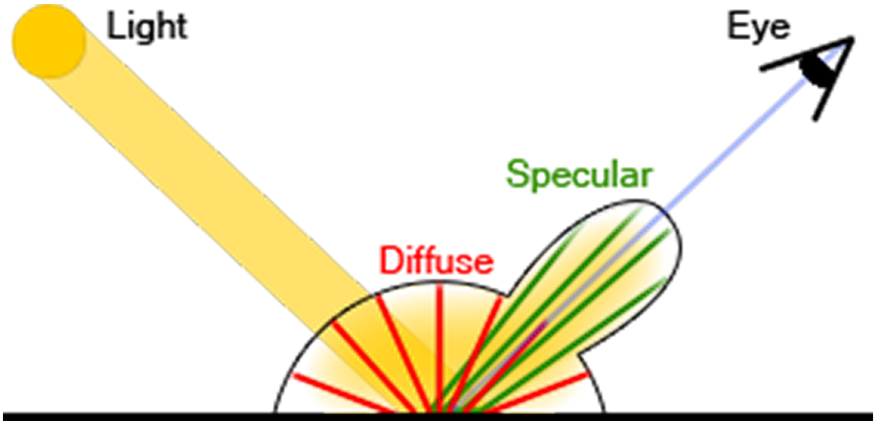

Figure 2.11.: Diffuse and specular component of the BRDF.
(Image source: http://www.codinglabs.net/article_physically_based_rendering_cook_torrance.aspx, 19.08.2018)

A BRDF can also be classified as isotropic or anisotropic BRDF. Isotropic is used to describe, that reflectance properties are invariant with respect to rotation of the surface around the surface normal vector. Anisotropic means, that the reflectance properties change significantly in respect to the rotation of the surface around the surface normal.

Some examples are brushed metal, satin or hair (Wynn n.d.).

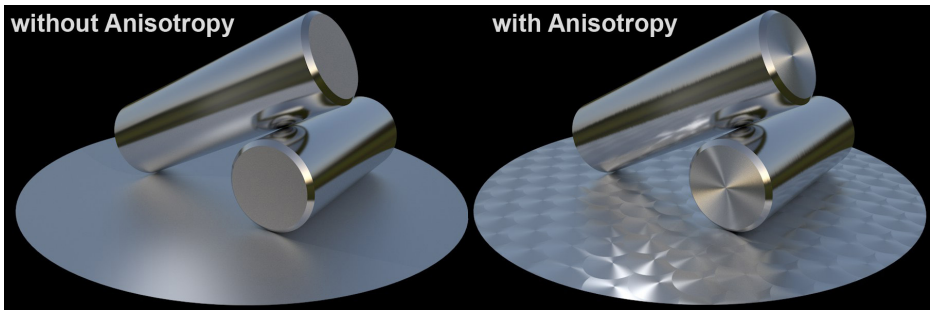

Figure 2.12.: Isotropic and anisotropic material.
(Image source: http://c4d.cn/help/r17/US/html/MMATERIAL-ID_MATERIALGROUP_REFLECTION.html, 19.08.2018)

One of the earliest BRDF models, that is still widely used, is the Phong model. Even though it is not physically accurate and does not obey the law of energy conservation nor reciprocity, it is simple and fast to calculate. Blinn or Blinn-Phong is the standard model used in OpenGL and DirectX (Montes and Ureña 2012). It is an alteration of the Phong model and fits the structure of a microfacet BRDF (Hoffmann 2013). Microfacets mimic the roughness of a surface as if it was made of a bumpy surface at a microscopic level. It is a more accurate representation of a real world surface. Rays are not reflected evenly across the surface, instead they are scattering in different directions based on the microfacets, which each have their own normal (Nichols 2016).

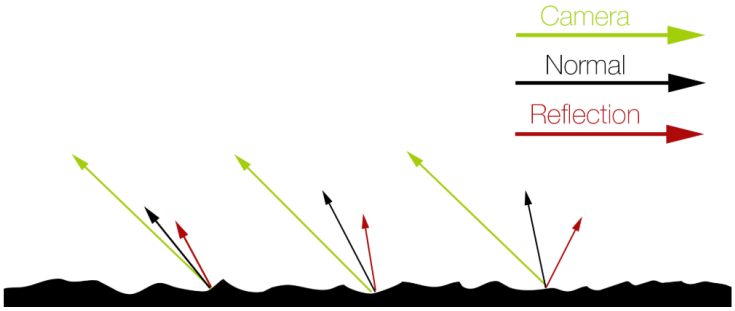

Figure 2.13.: Microfacet BRDF for more realistic reflections.
(Image soure: https://www.chaosgroup.com/blog/understanding-glossy-fresnell, 02.09.2018)

The Ward model takes anisotropic reflectance into account but it is also not physically plausible. It is still popular due to the cheap calculation of its versatile reflectance function. Lafortune is the most multifunctional BRDF model as it was used to fit measurements of real surfaces. It is considered a generalization of the Phong model but with reciprocity and the energy conservation law (Montes and Ureña 2012).

2.5 Distributed ray tracing

Ray tracing described so far is limited to sharp shadows, reflections and refraction. This is because rays are determined precisely while fuzzy phenomena would require more samples per ray. By distributing the rays it is possible to sample blurred reflections, blurred transparency and penumbras. Furthermore, it is possible to sample depth of field and motion blur, making the scene more realistic. Distributing the rays is not much more expensive than standard ray tracing and solves a lot of problems as such (Cook et al. 1984). Distributing rays over certain kinds of areas solves certain types of effects. I.e.: by sampling over the pixel area it results in anti-aliasing, by sampling over the lens area it results in depth of field, by sampling over time it results in motion blur (Waters 2003 a).
The reason for basic ray tracer to have sharp shadows is that point light sources are used. For creating soft shadows area light sources are needed. The umbra is the region which is fully occluded by an object and the penumbra the region where some light is occluded and some is illuminated (Figure 2.14a).

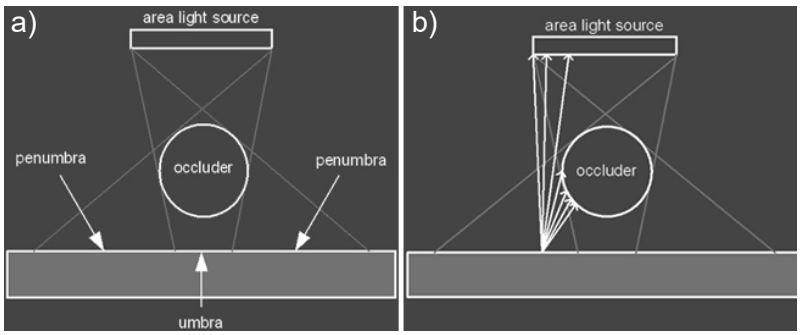

Figure 2.14.: a) Umbra and penumbra of a area light source. b) Distribution of secondary rays to calculate penumbras.
(Image source: http://web.cs.wpi.edu/~emmanuel/courses/cs563/write_ups/zackw/realistic_raytracing.html, 18.07.2018)

A possible way to approximate soft shadow with a standard ray tracer is to place several point lights near each other where each one has a part of the intensity of the base light. This will lead to sharp transitions in the penumbra, because its point lights nonetheless (Waters 2003 a). To achieve penumbras with area lights, the shadow rays are distributed and then weighted according to brightness and projected area of different parts of the light source (Figure 2.14b) (Cook et al. 1984). On rough surfaces reflected rays are spread. Here the distributed ray tracing algorithm comes in handy. Glossy and translucent reflections are achieved by distributing a set of reflection rays and randomly disturbing the ideal

specular reflection ray (Figure 2.15a). The wider those rays distribute, the rougher the surface. This is easily implemented by using a square which is perpendicular to the specular reflection vector. Rays are then shot through sample locations which are jittered across the surface. This square's width determines the glossiness or translucency of the surface (Figure 2.15b). While glossiness distributes the specular reflection, ray translucency distributes the specular transmission ray in order to set of secondary rays (Waters 2003 a).

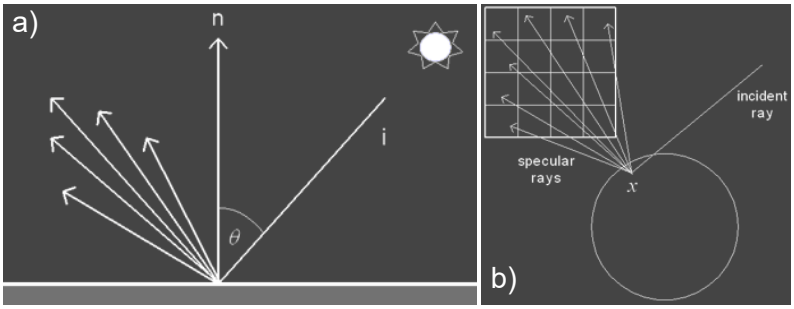

Figure 2.15.: a) Distributed reflection rays. b) Implemented square for determining where the secondary rays can travel.
(Image source: http://web.cs.wpi.edu/~emmanuel/courses/cs563/write_ups/zackw/realistic_raytracing.html, 18.07.2018)

To achieve depth of field, a ray from the center of the lens to a point on the screen is constructed. Then a ray is traced from any location on the lens to the focal point of the original ray. It needs to be determined which object is visible (Figure 2.16a).
Motion Blur is achieved by sampling the points in time. The path of motion can be arbitrarily complex as seen in Figure 2.16b. Shadows, depth of field, reflections and intersections are all correctly motion blurred with distributed ray tracing. It only needs the calculation of the position of the object at a specific time. With proper anti-aliasing sample artefacts or holes can be erased (Cook et al. 1984).
With ray tracing methods covered so far mostly direct illumination is simulated (Figure 2.17a). But what about indirect illumination? An answer to this is global illumination.

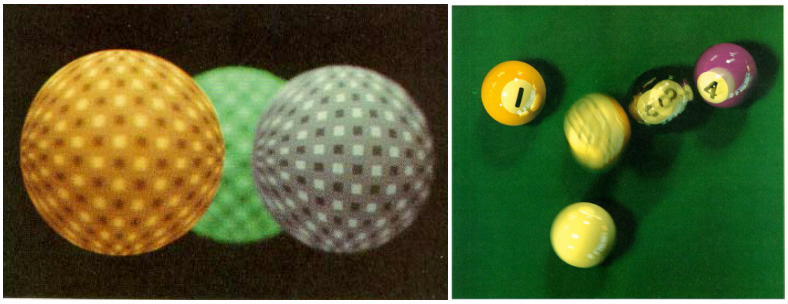

Figure 2.16.: a) Depth of Field with distributed ray tracing. b) Motion Blur with distributed ray tracing.
(Image source: Cook et al. 1984, S.6 & 9)

2.6 Global illumination

Global illumination simulates direct as well as indirect illumination, which is needed to produce realistic images. The only downside to it is, that it is a heavy calculation process which takes a lot of time. For real-time rendering this is not possible and therefore not offered until today (Scratchapixel 2015 b). Global illumination can be divided into direct illumination, specular reflection, indirect illumination, and caustics. The last two categories describe the light an object receives, but has bounced at least once before (indirect illumination) and light which concentrates in a particular area through bending, mostly as a result of refraction (caustics). To achieve those effects, the Monte Carlo method is required (Tuttle 2003).
"The idea of Monte Carlo ray tracing is to distribute rays stochastically in order to account for all light paths (Tuttle 2003, S.1)."

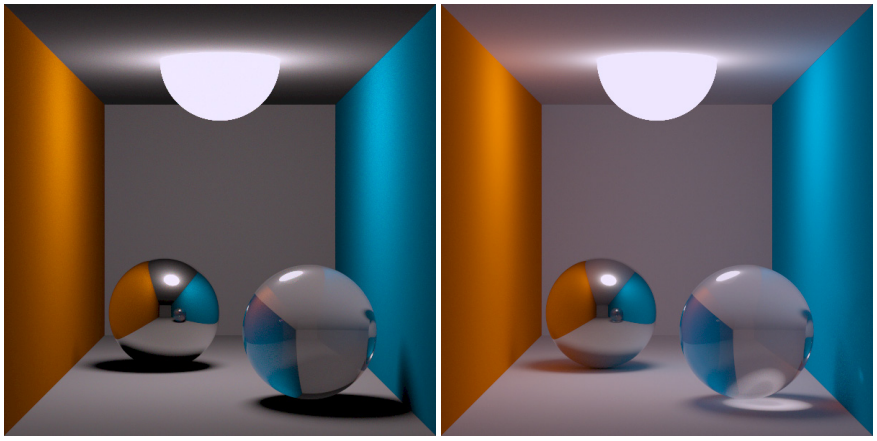

Figure 2.17.: a) Distributed ray tracing algorithm. b) Path tracing algorithm.
(Image source: Thomas Kabir, https://de.wikipedia.org/wiki/Diffuses_Raytracing & https://de.wikipedia.org/wiki/Path_Tracing, 20.07.2018)

2.6.1 Path tracing

Path tracing uses the Monte Carlo integration to solve the rendering equation (Vlans 2018). That means, that rays are distributed stochastically in order to account for all light paths. While path tracing seems similar to ray tracing, the important difference is that it traces the entire path of light rays. While ray tracing samples the light upon each hit with a diffuse surface, path tracing follows the ray until it eventually hits a light. That leads to a lot of paths not contributing to the final rendering and therefore a large number of samples is needed for a noise free image. If the algorithm is extended with distributed ray tracing, glossy reflections, motion blur, and depth of field effects can be simulated (Carr and Hulcher 2009).

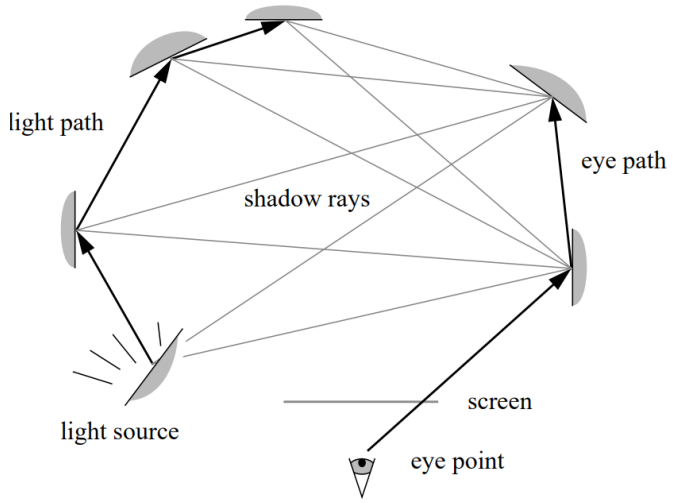

Figure 2.18.: Visualization of bidirectional path tracing.

(Image source: Lafortun and Willems 1997, S2)

Bidirectional path tracing combines forward and backward ray tracing. One path is traced from the eye and a second path is traced from the light source into the scene at the same time (see figure 2.18). All hit points of the rays are connected through shadow rays and the contributions are then added together to the light intensity of the pixel. This approach also takes secondary, tertiary, etc. light sources into account (Lafortun and Willems 1993).

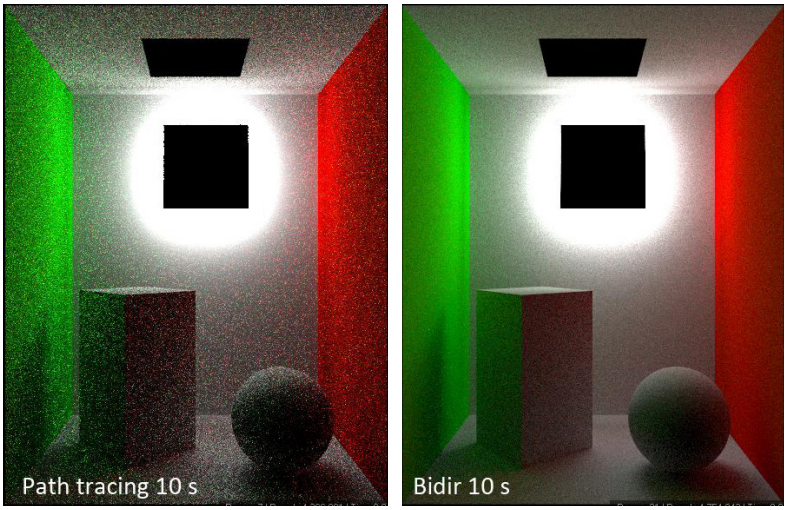

Figure 2.19.: Unidirectional vs. bidirectional path tracing with equal render time.

(Image source: https://corona-renderer.com/forum/index.php?topic=146.0, 05.08.2018)

2.6.2 Photon mapping

Photon mapping is a pre-calculated two pass algorithm which stores photons in a map to solve global illumination and caustics. In the first pass, photons are traced through the scene. They begin as a set at each light source of the scene and divide the overall power of the light among them. Brighter lights therefore emit more photons. The emitting direction of the photons depends on the type of light source. I.e.: for point lights photons are emitted uniformly in all directions. After the emission, the photons are scattered around the scene and bounce off surfaces dependent on the surface's material. It is then decided how much energy is absorbed, reflected or refracted. The millions of emitted photons are then stored in a compact photon map. In the second pass, this photon map is used to calculate indirect lighting and caustics. Specular/glossy effects and direct lighting would be to expensive to account for with the same technique. Rendering indirect lighting using photon mapping can be achieved through multiple solutions. The photon map is used either with path tracing or directly and with a subsequent final gathering step to improve the results. Latter dramatically increases the render time. For rendering caustics the photon map is visualized directly. That means the radiance can be estimated directly by gathering the nearest photons. Reducing the costs even further by creating a second photon map just for caustics is a possible solution (Waters 2003 b).

Figure 2.20.: Photon map showing indirect illumination and caustics.
(Image source: http://marctenbosch.com/photon/, 10.08.2018)

Progressive photon mapping (PPM) improves the concept of storing millions of photons in a separate process. By using progressive refinement, it is capable of computing correct solutions without storing any photons while retaining the robustness of a photon map. PPM is a multi-pass algorithm in which the first pass uses ray tracing where the rays stop bouncing as soon as they hit a non-specular surface. The following passes are photon tracing where a given number of photons are traced from the light source. The hit points of the ray tracing pass are then looped through in order to find photons within the radius of each hit point. Newly added photons refine the estimated illumination of the hit point and are then deleted. A new photon pass can be traced. The quality of the image progressively improves until the final image quality is reached (Hachisuka et al. 2008).

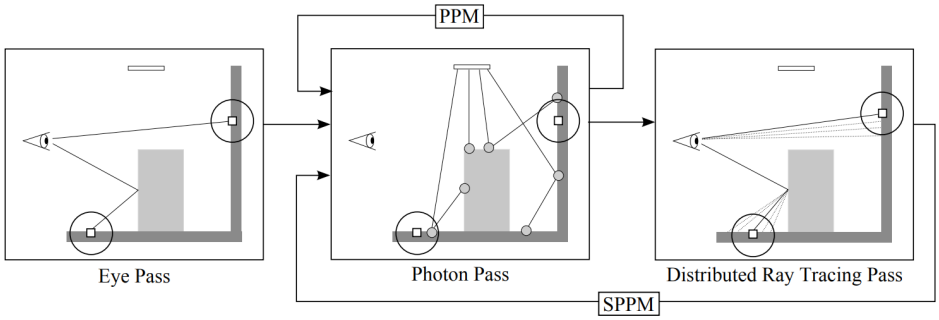

Figure 2.21.: Progressive photon mapping and stochastic progressive photon mapping algorithm visualized.
(Image source: Hachisuka, Jensen 2009, S.3)

To improve photon mapping even further, stochastic progressive photon mapping (SPPM) was introduced in 2009 by Hachisuka and Jensen. SPPM computes the average radiance values over a region to render distributed ray tracing effects like depth of field, motion blur and anti-aliasing. It therefore uses distributed ray tracing instead of standard ray tracing in the first pass. This pass generates new hit points after each photon pass (see figure 2.21). Especially for glossy reflections/refractions SPPM creates better results than PPM (figure 2.22) (Hachisuka, Jensen 2009).

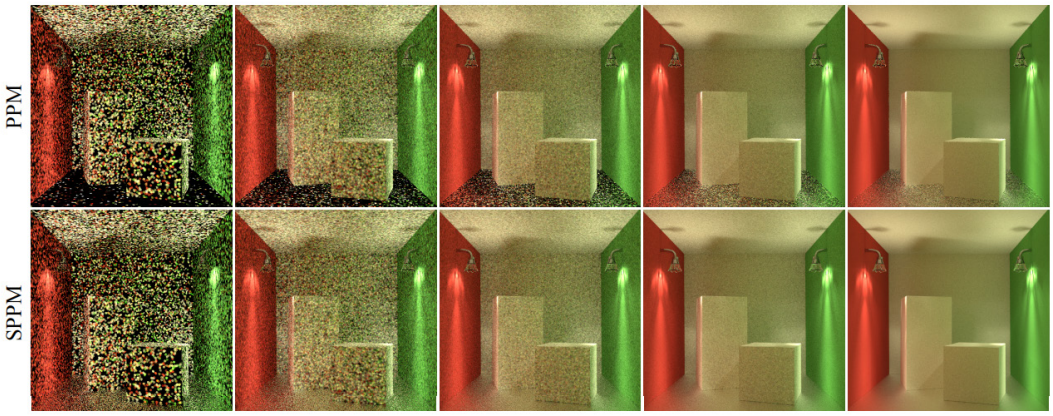

Figure 2.22.: Glossy reflections on floor rendered with PPM and SPPM.
(Image source: Hachisuka, Jensen 2009, S.9)

Photon mapping is used by many rendering software nowadays.

2.7 Render software

Rendering is the process of generating a 2D image from a virtual 3D scene. It is similar to photography or cinematography, because the scenes are lit and staged and are then generated into an image (Birn 2002). There are many rendering software available and to choose one means picking the best approach that will give the best looking result with the least amount of render time. Some renderers are better with complex shading, some with complex motion blur, sub-surface scattering or other light effects (Seymour 2012). Because rendering is used in many different industries like animation movies, architectural visualization or special effects, there is a large amount of different engines available. Some are just integrated in a 3D software, others come as a plug-in for multiple software or even have a standalone version (see figure 2.25). And then there is the option of rendering with CPU, GPU, or both (Von Übel 2018). GPUs allow for faster rendering due to their huge amount of core integration. So instead of rendering with 24 blocks of data with

Figure 2.23.: Interior scene rendered with Octane.
(Image source: https://home.otoy.com/render/octane-render/showcase/, 25.08.2018)

a CPU, a GPU can handle 3000 blocks of data at the same time. One of the limitations of GPU rendering is the memory that can only be stored on the graphics card. But the newest version of the Redshift renderer for example allows the GPU to use the main memory for rendering as well (Harcourt 2016).

Figure 2.24.: Interior scene by aTng, rendered with V-Ray.
(Image source: https://www.chaosgroup.com/en/gallery/atng-studio, 25.08.2018)

Biased and unbiased are often heard terms related to render engines. Biased in terms of rendering means, the algorithm is optimized to speed up the render time and approximating the physics of light rather than precisely calculating it. Unbiased therefore means, that no short-cuts are taken while tracing the path of rays. The results are from exceptional render quality, but the render times are huge which is a big issue. Even though it would seem that unbiased is the way to go for a physically accurate rendering, it is almost equally inaccurate as biased rendering. Using a BRDF such as Blinn or GGX is in itself a approximation of a real-world material. Some render engines use biased as well as unbiased methods as a trade-off for better render times while accomplishing the best rendering result (CGVizStudio 2015).

Name	Price	Hardware	OS	Plugin
3Delight	Free/$600	CPU	Windows, macOS, Linux	3ds Max, Katana, Maya
Arnold	$600/year	CPU	Windows, macOS, Linux	Maya, Houdini, Cinema 4D, 3ds Max, Katana and Softimage
Clarisse	Free / $999-$2,899	CPU/GPU	Windows, macOS, Linux	Standalone
Corona	$30/month	CPU	Windows	3ds Max
Iray	$295/year	GPU	Windows, macOS	3ds Max, Cinema 4D, Maya, Rhinoceros
Keyshot	$1,995	GPU	Windows, macOS	Standalone
LuxRender	Free	GPU	Windows, macOS, Linux	3ds Max, Blender, Carrara, Cinema 4D, DAZ Studio, Maya, Poser, SketchUp, XSI
Maxwell Render	$495	CPU/GPU	Windows, macOS, Linux	3ds Max, ArchiCAD, Cinema 4D, formZ, Maya, Modo, Revit, Rhinoceros, SketchUp, SolidWorks
Mental Ray	$295/year	CPU/GPU	Windows, macOS, Linux	3ds Max, Maya
Octane Render	$399	GPU	Windows, macOS, Linux	3ds Max, ArchiCAD, AutoCAD, Blender, Carrara, Cinema 4D, DAZ Studio, Houdini, Inventor, Lightwave, Maya, Modo, Nuke, Poser, Revit, Rhinoceros, SketchUp, Softimage
Redshift	$500	GPU	Windows, macOS, Linux	3ds Max, Cinema 4D, Houdini, Maya, Softimage
RenderMan	$495	CPU	Windows, macOS, Linux	Blender, Houdini, Katana, Maya
V-Ray	$750	CPU/GPU	Windows, macOS, Linux	3ds Max, Blender, Cinema 4D, Maya, Modo, Nuke, Revit, Rhinoceros, SketchUp, Unreal

Figure 2.25.: Cropped list of common render software 2018.

Figure 2.26.: (Image source: https://all3dp.com/1/best-3d-rendering-software/, 25.08.2018)

3 Comparison of render software

It is not easy to compare render software on a fair basis, since every renderer has different features and settings. So, the features the renderers share, determine to which degree a comparison is possible. Those features are then compared to each other and additionally render times per frame are set. This makes it easier to see which renderer has a lead in quality and speed. After the evaluation, a side-by-side comparison of the integration into the compositing pipeline is performed.

3.1 Selection of render software

There are multiple render engines available for different software as plug-ins or as a standalone version (see figure 2.25). But only a few of these are integrated in the compositing software Nuke. So, the choice of renderers to compare in Nuke comes down to four renderers. There are two external renderers: VRay by ChaosGroup and Octane by Otoy. And there are two renderers which are already available in Nuke are RayRender which comes with NukeStudio or NukeX, and PrmanRender by Pixar. PrmanRender is a already available node in Nuke, but is only usable if Pixar’s Renderman Pro Server 20 or an earlier version is installed. Since the current Version of Pixar's Renderman is 22 (available since 18.07.2018) (Pixar 2018), the PrmanRender node in Nuke is no longer supported. PrmanRender was considered to be an alternative to the ScanlineRender renderer of Nuke (Foundry 2018 a). Since Nuke 10, RayRender is the new ray tracing renderer and replaces the PrmanRender node (Wright 2016). It is also worth mentioning that the PrmanRender node and Pixars Renderman are not the same renderers, but for the purpose of a better readability, PrmanRender will be referred to as Renderman for the remainder of this thesis.

3.1.1 Features

RayRender supports the geometry, light, camera and most of the shaders within Nuke. The parameters of the render node are for refining the image quality, motion blur and AOVs (Arbitrary Output Variables) which can be used as render passes. The projection

mode allows to change the camera into an orthographic or a spherical camera which captures 360 degree images for virtual reality (VR) (Foundry 2018 b).

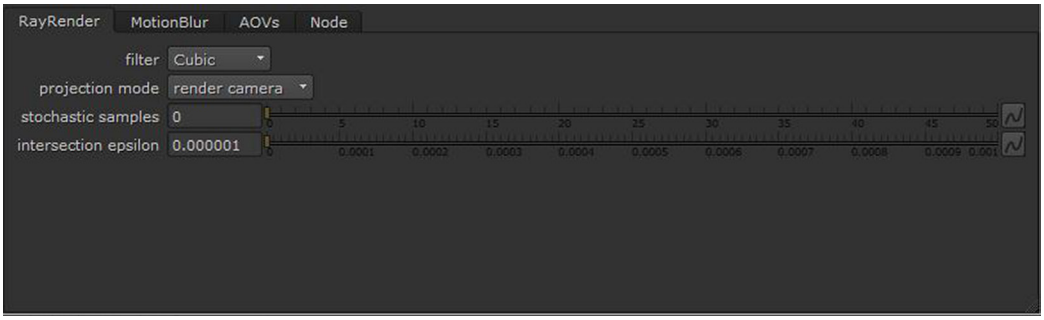

Figure 3.1.: Main settings of the RayRender node in Nuke.

Renderman offers similar features. It support the same nodes as RayRender but is able to use the refraction shader and is therefore able to render refractions with transparent objects. The main settings include refinement of image quality, ray depth and motion blur. While it does not offer a spherical camera projection, Renderman does have options for turning on shadows, reflections, refractions and depth of field each individually. Additionally, the output of motion vectors and RIB files is possible. RIB files are Renderman-compatible ASCII files which are created during the rendering process. Render passes are not available in Renderman (Foundry 2018 a).

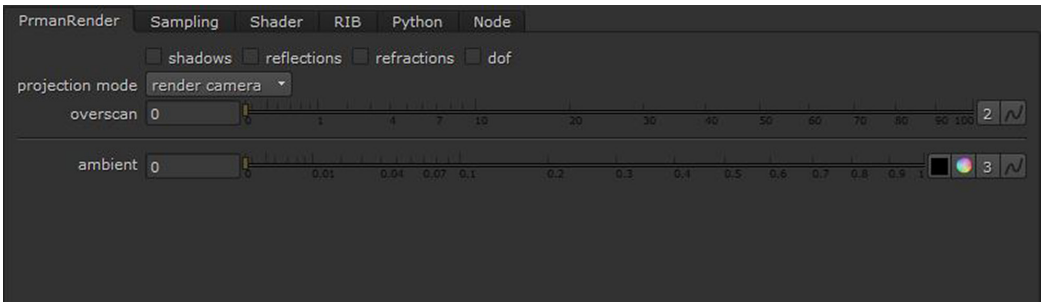

Figure 3.2.: Main settings of the PrmanRender node in Nuke.

VRay comes with its own materials, lights, textures, volumetric effects, render passes, denoiser, camera and render nodes. It also supports geometry, shader, light and some general nodes from Nuke itself. The virtual camera comes with physical properties to simulate settings of photo or film cameras. The camera can be turned into a dome camera (similar to the spherical camera of the RayRender node) or a stereoscopic camera which renders left and right pairs of images for stereoscopic 3D output. The physical camera gives full

control over motion blur and depth of field effects. These effects can be adjusted in the main settings along with global illumination, caustics, image quality and further render settings. There are three render types available to choose from: *Scanline* uses the CPU with the default scanline engine, *Progressive CPU* works as a progressive render engine but does not require a graphics card, Progressive *OpenCL* and *Progressive CUDA* are progressive render engines using graphics cards but have limited abilities with regards to shaders. *Progressive CUDA* uses Nvidia graphics cards that support the CUDA technology, for other graphics cards *Progressive OpenCL* is used, provided they support OpenCL. The plug-in contains a set of lights like spotlights or area lights which are designed specifically for rendering with VRay. The light dome supports HDRI and even a sun/sky system is integrated. By using render elements render passes can be created for more control of the final composition in Nuke (Chaos Group 2018 b).

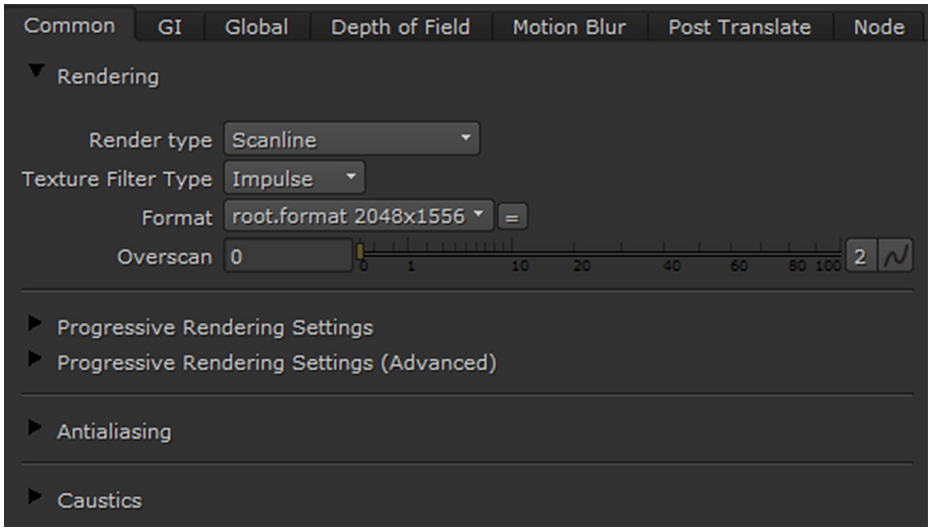

Figure 3.3.: Main settings of the VRayRender node in Nuke.
(Image source: https://docs.chaosgroup.com/display/VRAYNUKE/Render+Settings+%7C+VRayRenderer, 08.09.2018)

Octane comes with its own materials, which support subsurface scattering, chromatic dispersion and absorption. A material database is integrated as well. Furthermore, it supports HDRI, a sun/sky system and can use meshes as light sources by adding an emission material (Otoy 2018 a). There are three types of cameras Octane offers: *a thin lens camera* which represents a standard camera with parameters of photo or film cameras, *a panoramic camera* which is used for VR-related images and a *baking camera* which renders lighting, texture and material data directly into texture maps. Nuke's camera node is also supported (Otoy 2018 c). The most unique feature of Octane is that it uses only GPU

for rendering and is significantly faster than CPU-based render engines. It is also possible to use multiple GPUs to speed up renderings even more (Otoy 2018 a). But only Nvidia graphics cards with CUDA technology are supported (Otoy 2018 b). Octane offers post processing settings, render passes and a cloud rendering option. Also, volume and deep image rendering is possible (Otoy 2018 a).

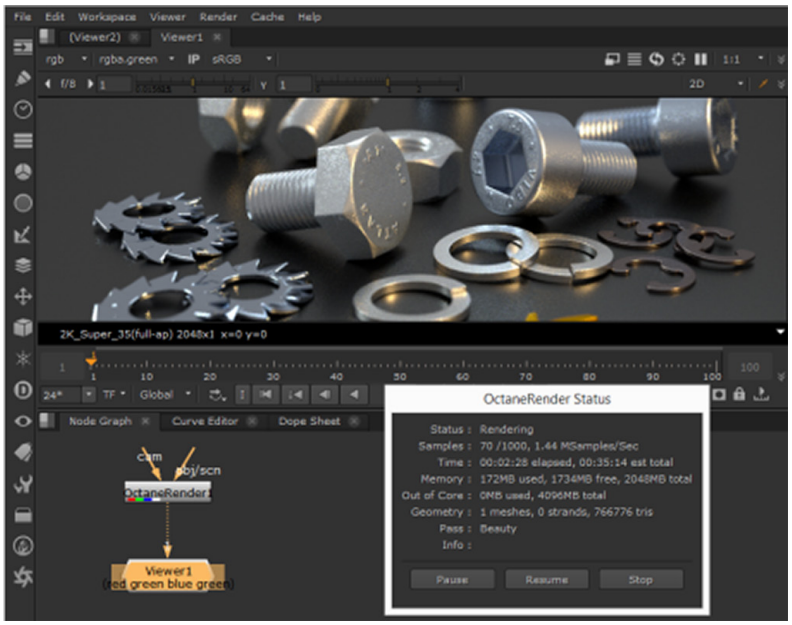

Figure 3.4.: Status window of the OctaneRender node in Nuke.
(Image source: https://docs.otoy.com/NukeH/NukePluginManual.htm#Nuke/MaterialsEditing.htm, 08.09.2018)

Furthermore, in the kernel settings there are several options for the kind of ray tracing being used. *Direct Lighting* is used for fast previews but does not lead to photo-realistic result. *Path tracing* and *PMC* are physical based and render photo-realistic images. *Path tracing* has difficulties with small light sources and may have issues with caustics. In those cases *PMC* is recommended (Otoy 2018 c).

3.1.2 Integration

RayRender is Nuke’s in-house ray tracing renderer and therefore comes with version 10 or higher (Wright 2016).
PrmanRender is already available in Nuke Studio or NukeX. But it is only accessible if Pixar's Render Pro Server is installed (Foundry 2018 a).
VRay’s plug-in for Nuke is integrated as a menu with several nodes for materials, lights and rendering. The rendering is calculated in the VRayRender node and can then be split

into individual render passes via render elements. The unique feature of the VRay plug-in is that there is no need to switch forth and back between software. Everything happens in Nuke.

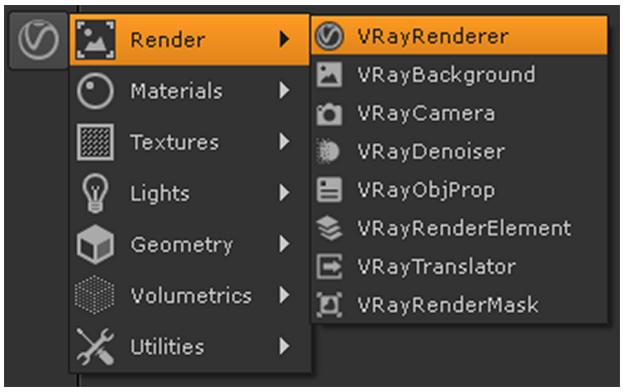

Figure 3.5.: VRay plugin as menu in Nuke.
(Image source: https://docs.chaosgroup.com/display/VRAYNUKE/Render+Settings+%7C+VRayRenderer, 08.09.2018)

The Octane menu offers only three nodes: the OctaneRender node, OctaneReference-Geometry node, and OctaneMaterial node. The OctaneRender node offers the main settings for rendering (see figure 3.6). Everything is categorised and overrides settings from the Octane scene. Under “render controls” the renderer needs to be started in order to output a picture in the viewer. After that, the rendering is interactive and every change restarts the progressive rendering process. “Edit octane scene” opens the octane stand-alone in a separate window. This allows better control over the scene geometry and materials (figure 3.7). Changes on camera, environment, kernel, render passes, and other settings will be saved back to the render node in Nuke. But, the settings inside of Nuke have priority over changes in the standalone window (Otoy 2018 c). It is more user-friend-ly to use the standalone version right from the beginning and import the Octane scene with the OctaneRender node. If a Nuke camera is connected to the OctaneRender node, the Octane camera position and target will be overwritten by the parameters of Nuke’s camera node (Otoy 2018 c).
The OctaneReferenceGeometry node gives the opportunity to import an Octane scene and add it to the render output by connecting it with the OctaneRender node. By putting a TransformGeo node between both nodes, it is possible to transform the geometry from inside of Nuke and not by opening and using the standalone window (Kinnane 2015).
The OctaneMaterial node offers the option to create an Octane material in a separate standalone window. This can also be done by using the standalone window described in

the OctaneRender node.

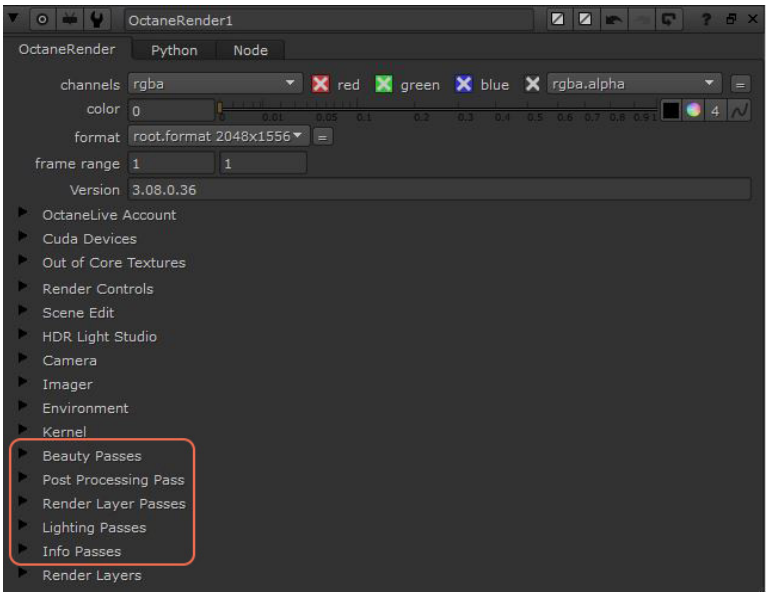

Figure 3.6.: OctaneRender node settings.

(Image source: https://docs.otoy.com/NukeH/NukePluginManual.htm#Nuke/RenderPasses.htm, 30.09.2018)

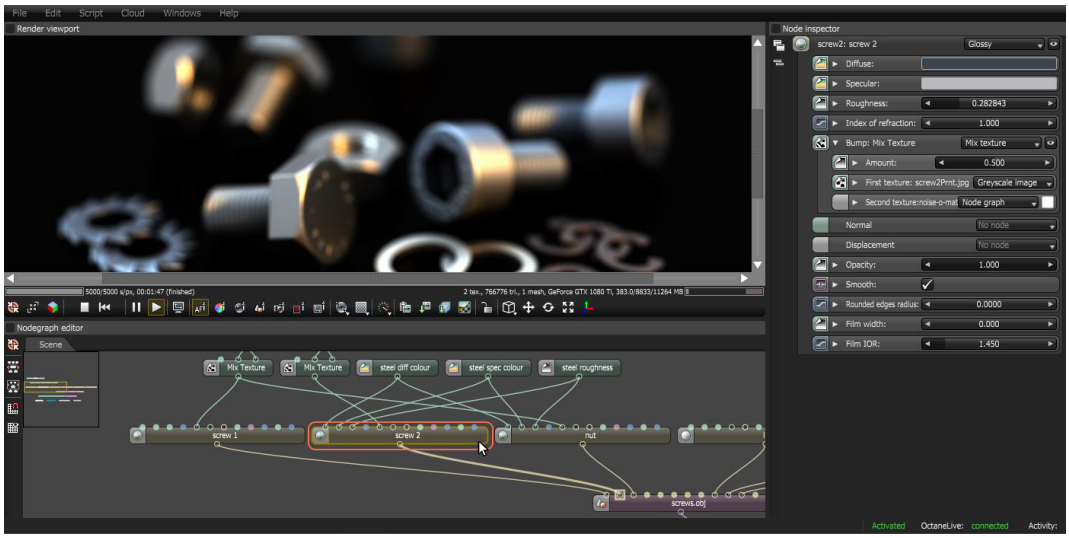

Figure 3.7.: OctaneRender standalone window.

(Image source: https://docs.otoy.com/NukeH/NukePluginManual.htm#Nuke/TheOctaneWindow.htm, 08.09.2018)

3.2 Comparison

An image quality comparison is the first basis on which the renderers need to perform. Also, a ray depth test is needed to define how each renderer reacts to reflection of objects with reflective materials. Furthermore, motion blur and depth of field can be compared very easily. With motion blur, there needs to be a distinction between object motion blur and camera motion blur. Since two of the renderers have their own materials a comparison of those is needed as well. And finally, a comparison of render times for each of the previous topics is given. All renderings have an image resolution of 4096x3112.

3.2.1 Image Quality

The first step for comparing renderers concerns the image quality. For traditional cameras there are camera charts to test the range of color, gradient, grayscale, resolution and contrast as well as artefacts when it comes to high frequencies, represented here with the thin lines seen in the gray chart.

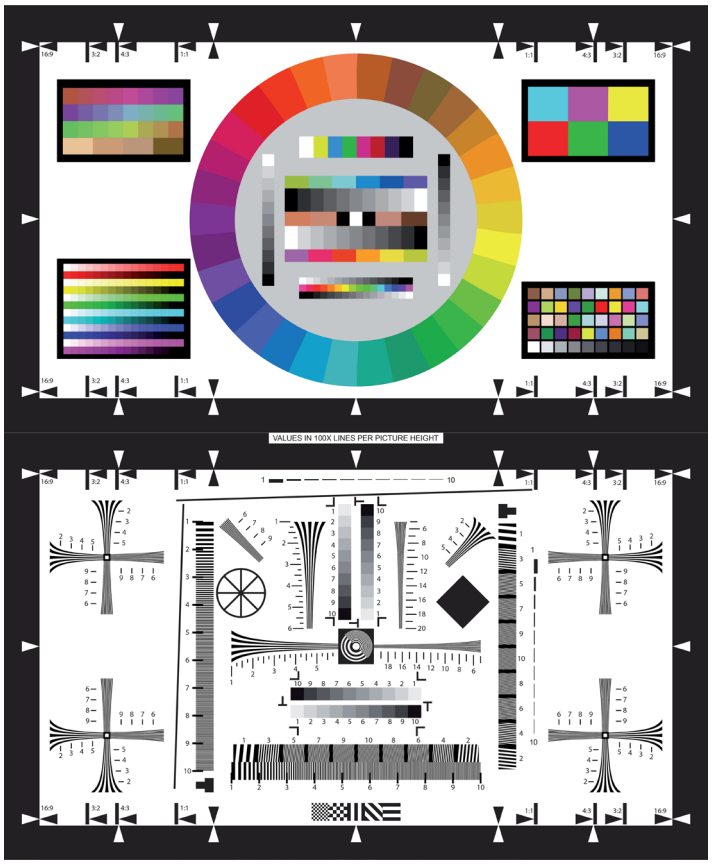

Figure 3.8.: Color chart and gray chart for testing the image quality of cameras.
(Image source: http://www.ageofarmour.com/3d/tutorials/AoA_metalized_glass_shader_help.html, 08.09.2018)

The renderings of the color chart and gray chart are compared to the original charts with a difference key. If a resulting pixel is black, there is no difference between the rendering and the original image. Regarding the color chart, this is exactly the case for all renderers. Taking a closer look, Octane has more information in the difference key than the others. This is due to a light falloff the Octane material creates. Other than that, the result is identical to the others.

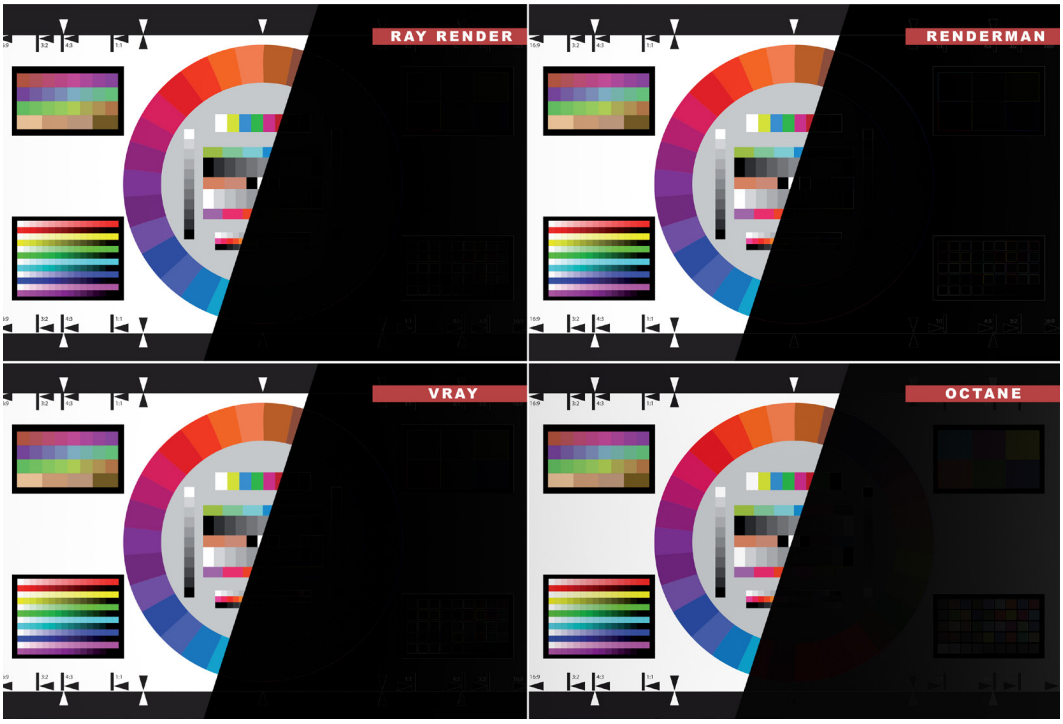

Figure 3.9.: Color chart rendering and difference key with the original chart.

In the case of the gray chart, there are some differences to spot. There seems to be an outline over all edges in the chart. To see the results in more detail, areas of interest are zoomed in and put next to each other in figure 3.11. The rendering is a little blurry, especially for Renderman. This may be a result of image filtering by the shader or of the resolution of the final image. Those errors are so tiny, you have to zoom in quite a lot to see differences and even then, they are very small. In a nutshell, it is safe to say, that all renderers provide a good image quality in terms of reproducing color, grayscale, gradient and contrast.

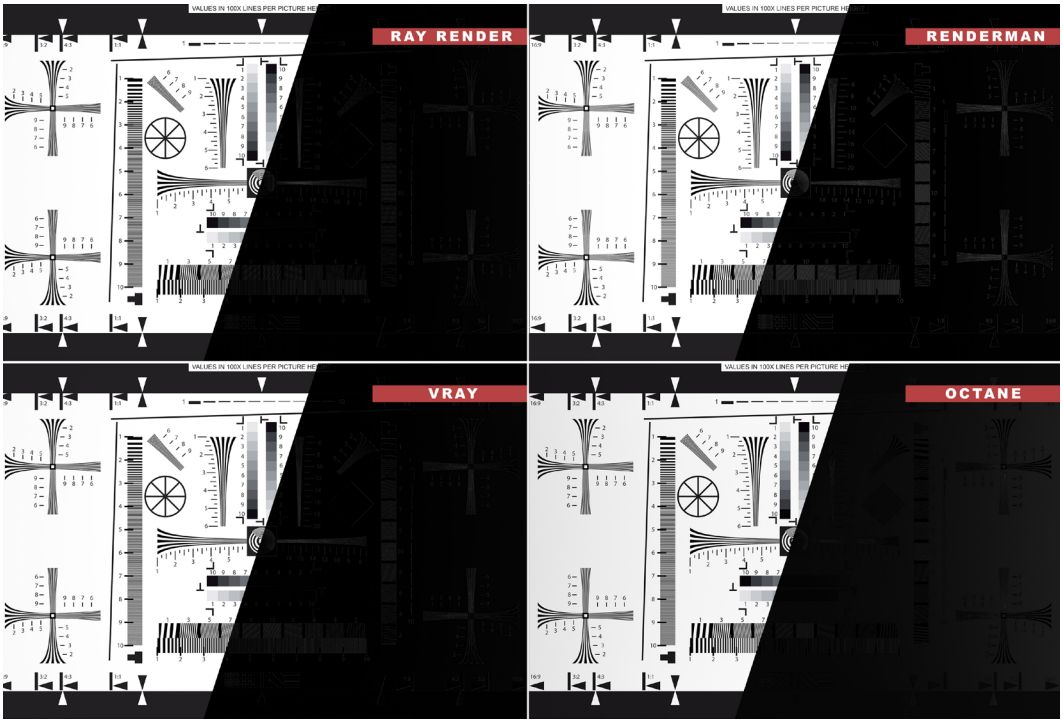

Figure 3.10.: Gray chart rendering and difference key with original.

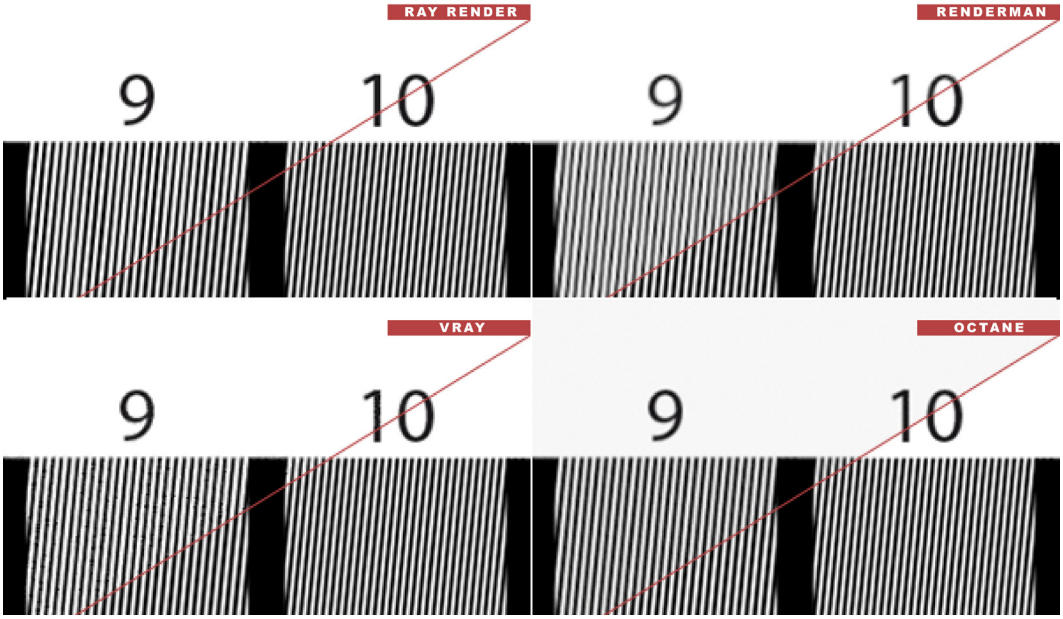

Figure 3.11.: Detailed comparison between rendering and original of the gray chart.

3.2.2 Ray depth

Ray depth defines the number of times a ray can be reflected. Higher numbers increase render times, smaller numbers lead to strange behaviours which are explainable by the ray tracing algorithm (see chapter 2.2) (Solid Angle 2018)

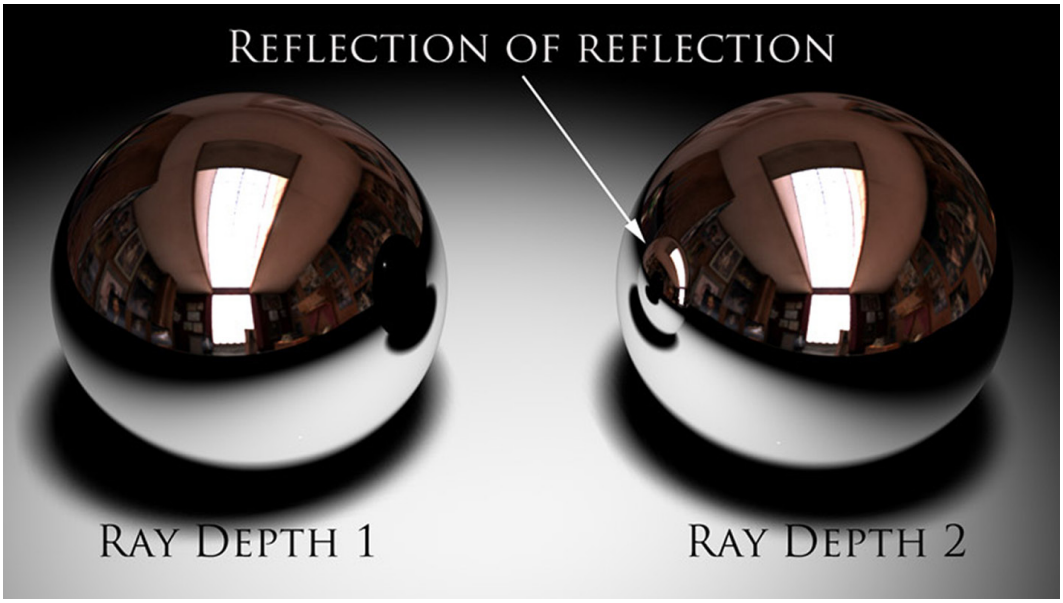

Figure 3.12.: Ray depth of 1 renders the object in the reflection black. Each additional depth renders one more level of the reflection.
(Image source: http://www.ageofarmour.com/3d/tutorials/AoA_metalized_glass_shader_help.html, 08.09.2018)

Knowing this, it is obvious that RayRender is only capable of creating a ray depth of 1, since only the environment, but not the reflection of the reflected object is visible (see figure 3.13). Renderman as well as VRay and Octane each have settings to adjust the ray depth level starting from 1.

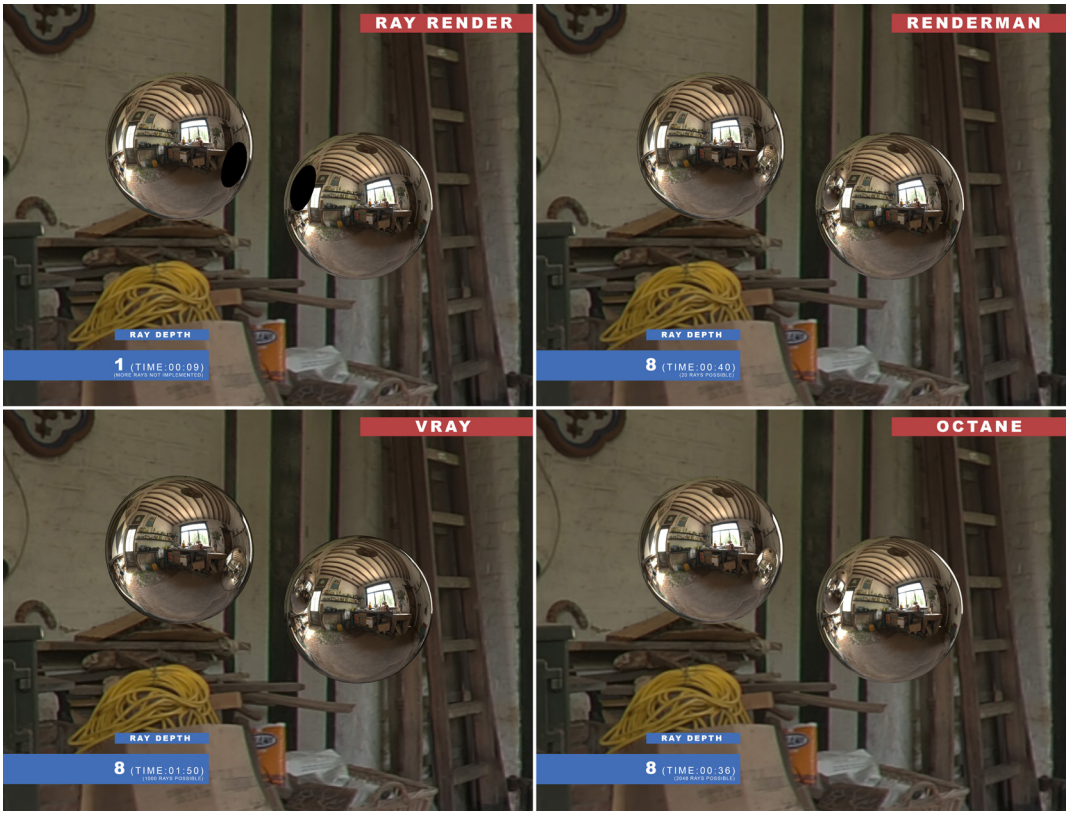

Figure 3.13.: Ray depth comparison. RayRender is only capable of a ray depth of 1.

3.2.3 Motion Blur

Motion blur is handled in different ways for each renderer. Renderman uses a workaround for motion blur, since it is not capable of calculating it through distributed ray tracing. There are three settings which influence the motion blur effect. *Pixel samples* duplicate the moving object and layer those duplications over one another with a different amount of transparency in the direction of movement (see figure 3.14).

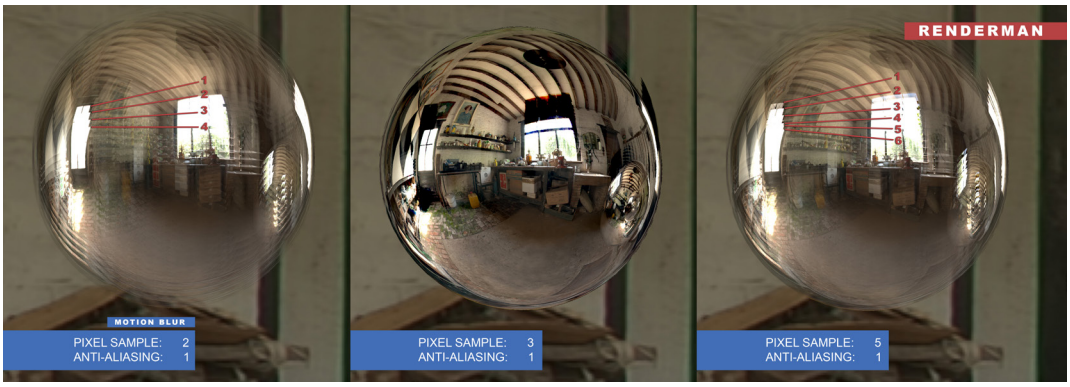

Figure 3.14.: Renderman motion blur settings with different values for pixel samples. The amount of overlaying images is not correspondent to the amount of samples. Also sometimes artefacts appear.

Anti-aliasing seems to interpolate and blur those duplicated versions to simulate a smoother look (see figure 3.15). The amount of pixel samples does not correspond to the number of duplications made. For some pixel sample levels, some strange effects occur when not anti-aliased, as seen in figure 3.14. The shutter indicates how far apart the samples are spread in the direction of movement. To get a good result, the right ratio of pixel samples to shutter needs to be set. Afterwards, anti-aliasing helps to get rid of possible artefacts.

Figure 3.15.: Renderman motion blur settings with higher anti-aliasing for different pixel samples. Artefacts are vanished. Higher pixel samples lead to longer render times.

But even with a smooth result, the motion blur is not accurate. For one, through the linear translation of the duplicated samples, the blurred motion is always linear per frame, even if the object moves in arbitrary directions. The second reason is that the reflected image of the moving object is not blurred and looks like it is frozen in time (see figure 3.16).

Figure 3.16.: Renderman's motion blur problems.

In comparison, each of the other renderers have a movement of the reflected image (see figure 3.18). While VRay and Octane are using distributed ray tracing in order to calculate realistic motion blur, RayRender seems to use a mix of both. While the layered samples implicate that RayRender uses a similar approach as Renderman to simulate a motion blur effect, the reflection in the right object is motion blurred as well. Even the reflection on the moving sphere has a different direction of motion blur than the sphere itself. This means that some ray tracing has to be involved in the calculation of the effect (see figure 3.17). Since there is no detailed documentation on how Foundry constructed RayRender, it is not certain to say how they achieved this kind of motion blur effect.

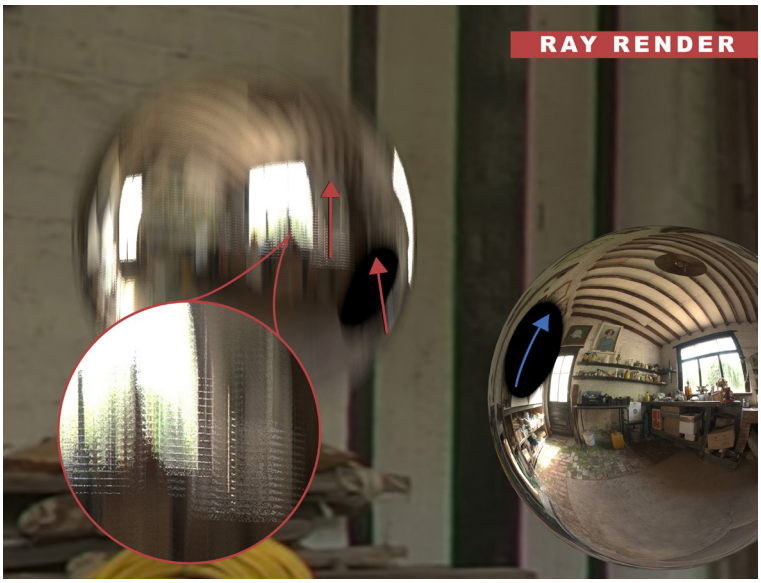

Figure 3.17.: RayRender motion blur effect. A mix of ray tracing and image processing to achieve for a more realistic motion blur effect while keeping render times low.

The shutter settings are different for almost all renderers as well, even though the length of the blurred object is the same. If the shutter of 0.5 is chosen for VRay, the same shutter is needed for Octane. RayRender needs to have shutter of 1 for a similar looking motion blur and Renderman even a shutter of 1.5. In a project this is important to know to match the shutter the footage was shot in. Octane and VRay use physical cameras to simulate the scene through the lens of a real camera. That is why the shutter speed can be adjusted precisely to the shutter of a photo or film camera. RayRender and Renderman have no physically accurate camera and therefore the shutter is more a approximation and needs to be set in resemblance to the footage.

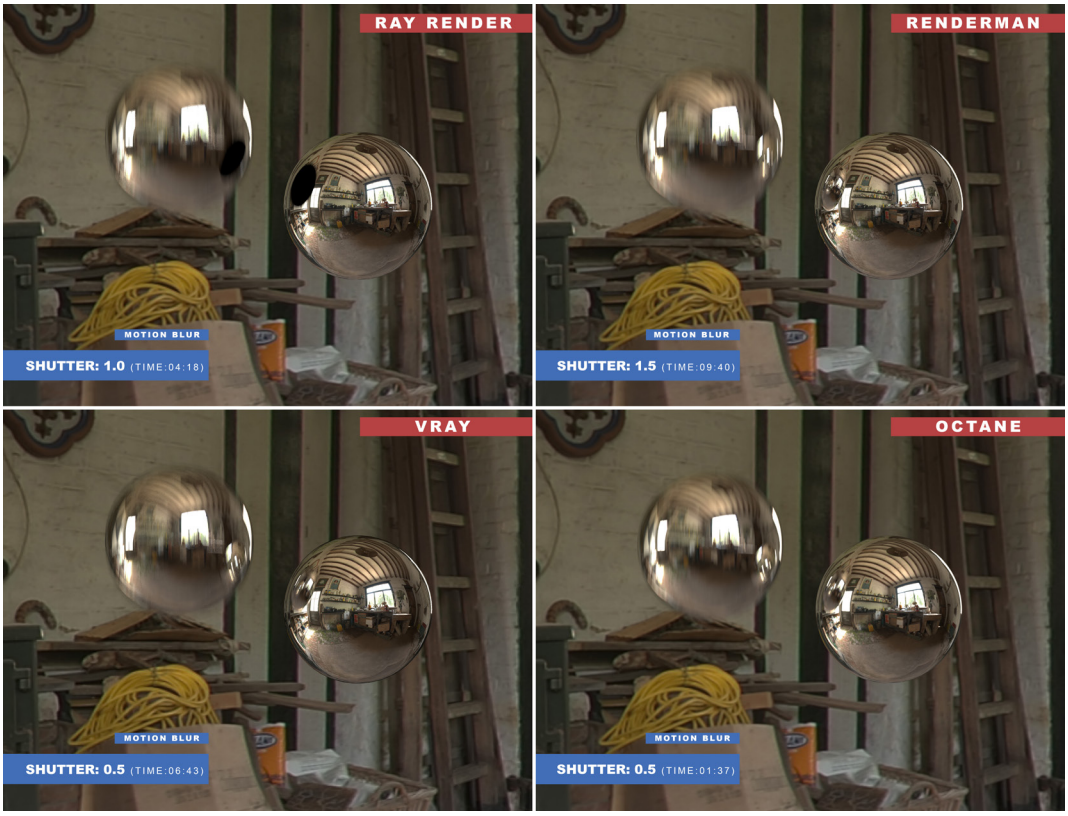

Figure 3.18.: Motion blur comparison. Renderman uses image processing to simulate a motion blur effect. RayRender uses image processing and ray tracing for a better looking motion-blur-like effect. VRay and Octane use ray tracing and are physically correct.

Every renderer generates a motion vector, if an object is animated. This vector can be passed on to a vector blur node. It is a simple solution to save render times and adjust motion blur in the post processing if needed. When looking at figure 3.20, the results are a smoother version of Renderman's motion blur in figure 3.18. The direction of the motion blur equals the direction of movement and the whole object is blurred that way. It does not account for motion blur in reflections or arbitrary movement. The vector blur distinguishes the direction due to color information. The amount setting changes the strength of the blur. To match the motion blur of the footage, an approximation is necessary.

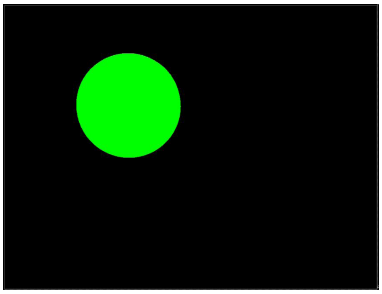

Figure 3.19.: Motion vector layer generated with RayRender.

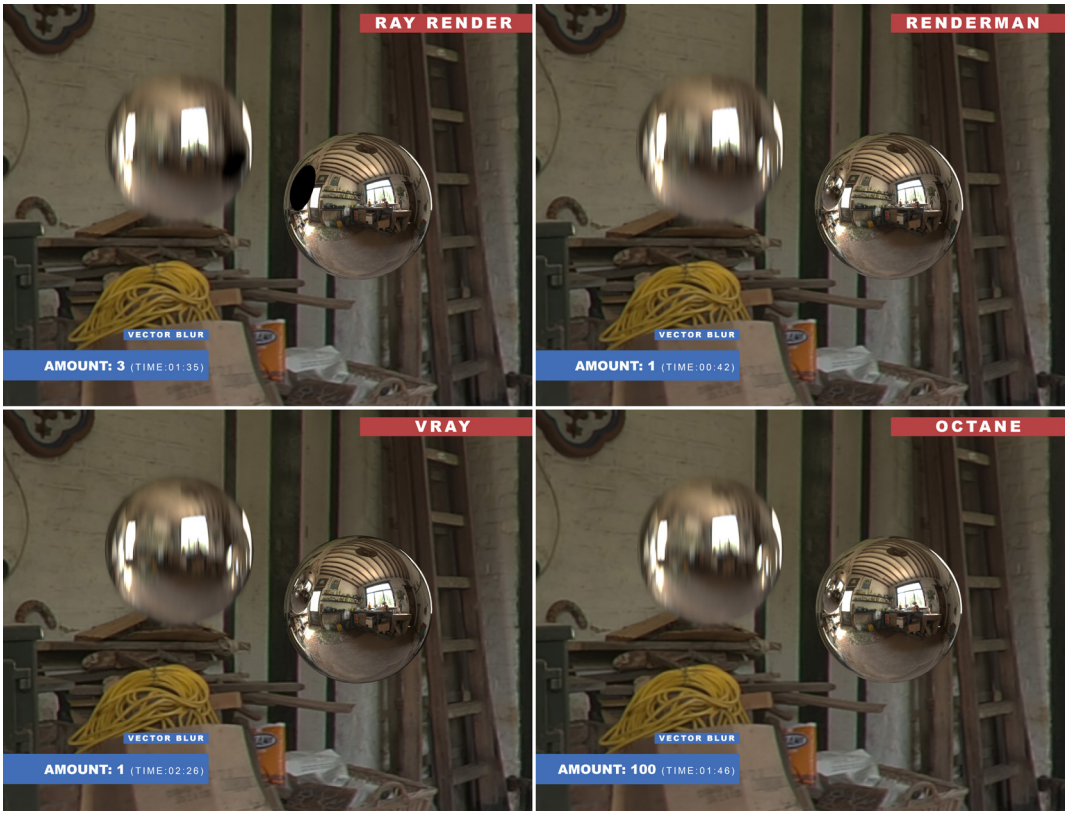

Figure 3.20.: Vector motion blur comparison. The results look similar for all renderers.

In addition to object movement, camera movement adds motion to the complete scene. Some renderers therefore have an extra check-box for camera motion blur. In this example, the camera is moving from left to right around the objects while targeting them. The result is an added motion to the moving sphere and a new motion, in the opposite direction of the camera movement, to the non-moving sphere (see figure 3.25). Having a closer look at figure 3.25, all the renderers handle camera motion blur very well. There are still differences in the details, which are mostly located in the reflection.

RayRender does a good job with camera motion blur. It still uses the image processing trick to simulate a motion blur effect, but the direction of movement in the reflections are clearly different to the object's movement (see figure 3.21). They appear similar to the reflections seen with Vray and Octane (figure 3.23 and 3.24).

Renderman does also a good job, but ignores motion in the reflection similar to vector motion blur (figure 3.22).

Figure 3.21.: RayRender camera motion blur effect. The motion of the camera is visible in the direction of the motion blur. The reflection has a different direction than the object.

Figure 3.22.: Renderman camera motion blur. The camera motion adds to the object motion but reflections have no different motion.

VRay and Octane both create a similar and accurate looking camera motion blur. Only artefacts are kinks of the motion blur visible in the reflection with VRay, which do not occur when using Octane (see figure 3.23).

Figure 3.23.: VRay camera motion blur. The reflections have a correct motion. Kinks are happen in the blurred reflection.

Figure 3.24.: Octane camera motion blur. Good results with correct reflections. No artefacts like VRay.

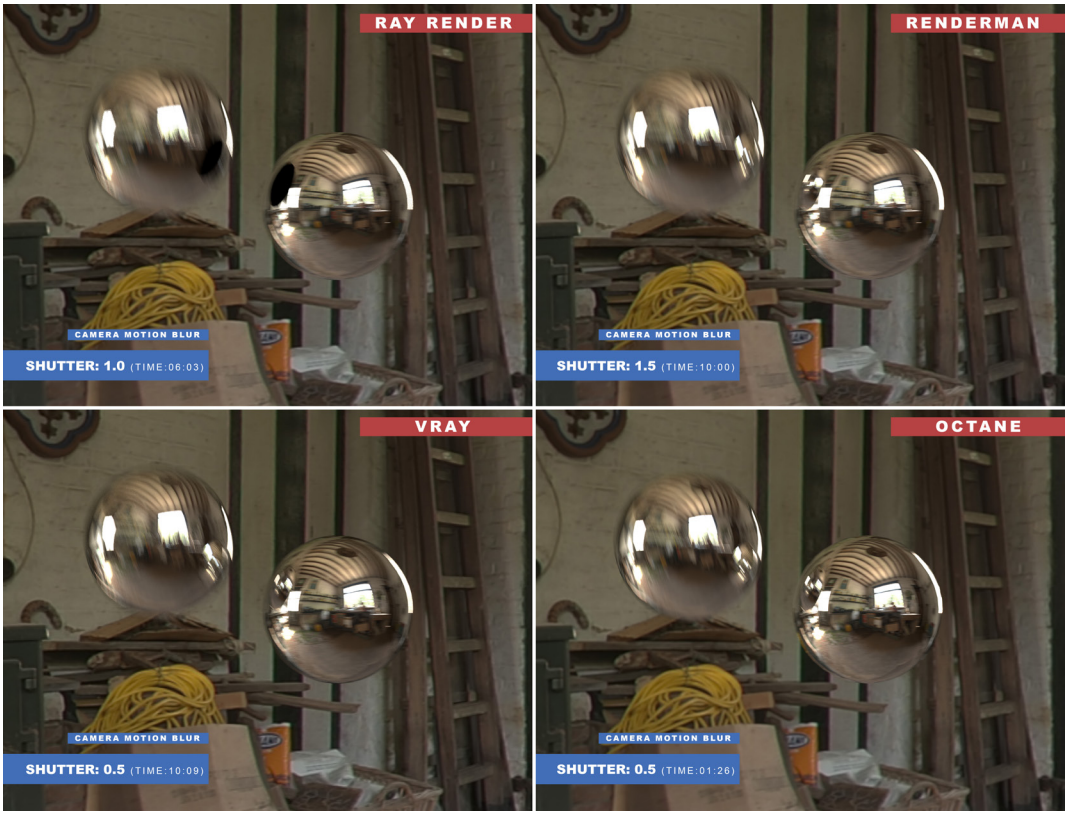

Figure 3.25.: Camera motion blur comparison.

In conclusion, RayRender produces great results, which are very close to real motion blur, even though they are created by some sort of image processing. Renderman just uses image processing to create a motion-blur-like effect. It is not accurate and similar to the results a vector blur node creates. Therefore, using vector blur with Renderman is clearly a plus, since it is much faster in calculation and gives the same results. VRay and Octane are perfect for creating physically correct motion blur. But VRay creates tiny artefacts in the reflection, when the motion is arbitrary.

3.2.4 Depth of field

Depth of field (DoF) is a calculation intensive effect and not all of the renderers are able to create it. While Renderman's depth of field calculations take a lot of time (see figure 3.27), RayRender does not even have the option for DoF. A first comparison between VRay and Octane shows a similar result, but the f-stop does not match (see figure 3.26). While VRay's f-stop is 2.1, Octane's is 0.025, which is an odd number. This is because the f-stop controller in Octane is limited to a number of 0.025 (or vice versa an aperture above 100). The f-stop needed to match VRays, would be 0.021, which is one hundredths

Figure 3.26.: Same scene size depth of field comparison between VRay and Octane.

of VRay's DoF. The reason for this is the scene size, since the calculation of DoF uses the camera aperture in millimeter. Nuke's scene size is 1 unit = 1 meter. So, a sphere with the scale of 1 has a 1 meter radius. Since Octane uses centimeter as a unit length, it scales the scene with a factor of 100. Therefore, the aperture of the camera needs to be reduced by the same factor. Since there is a limitation in this setting, another approach to get the real f-stop number would be to scale the scene down by a factor of 100. This results in the correct f-stop (see figure 3.27).

Figure 3.27.: Depth of field comparison between Renderman, VRay and Octane. The scene size is scaled to cm.

The reason VRay has no issues with the scene size inside of Nuke is, that the global settings can be changed for unit scale and unit metric. It is possible to choose a different scale (i.e.: 2 units = 1 meter) and different metrics (i.e.: centimeter, meter, inches,...). With the metrics set to centimeter, VRay calculates with the correct scene size and the f-stop as well as every other size-related issue is fixed.
In figure 3.27 Renderman is added and has a higher f-stop as well. As seen in chapter 3.2.3, Renderman does not calculate with distributed ray tracing. Therefore, it is possible that depth of field is not calculated via ray tracing as well. It must be assumed that again

some sort of image processing is used. Like with motion blur, the settings have to be approximated in order to match the footage.
Every renderer has the option to create a z-depth layer to adjust the depth of field in the post processing. With a Z-Defocus node the effect can be adjusted. Figure 3.28 shows the results of RayRender and Renderman, which look similar to the depth of field calculation in figure 3.27. It is again worth mentioning, that this post effect leads to an identical result for Renderman, but takes just a fraction of the render time. Even with a lower image resolution, which could lower the render times for Renderman, Z-Defocus calculates faster while creating identical results.

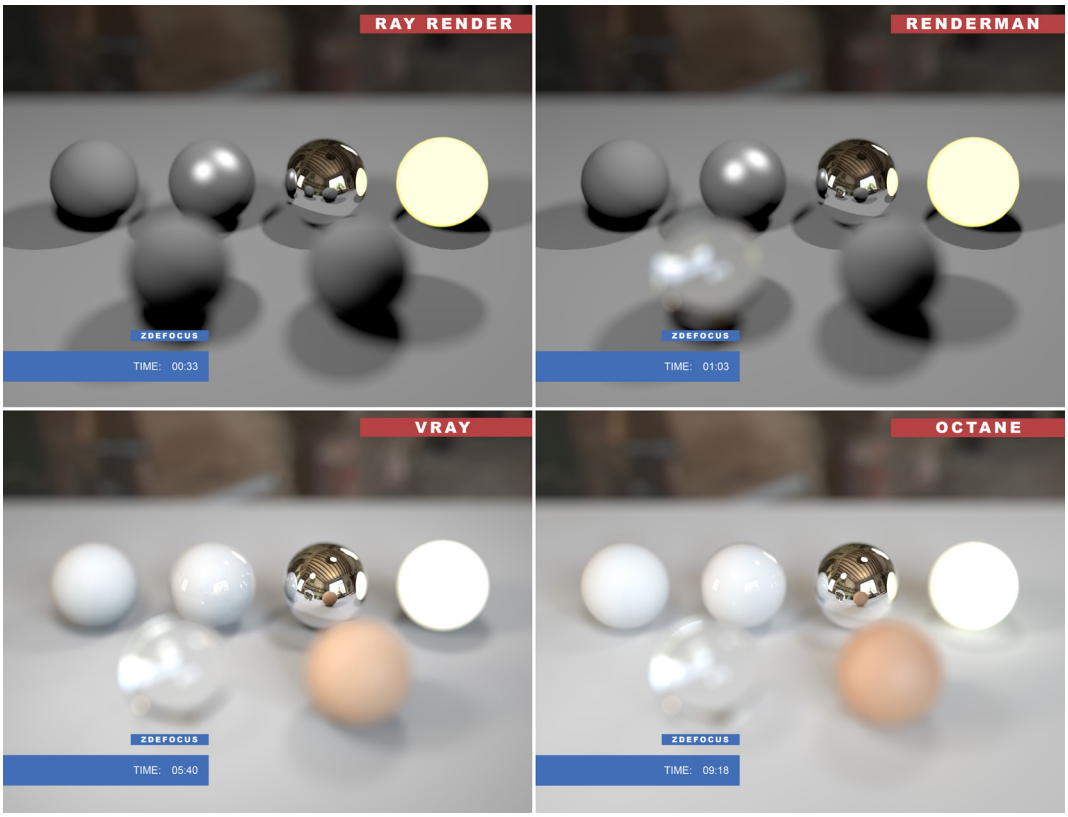

Figure 3.28.: Z-Defocus node to get depth of field for all renderers as a post effect.

VRay and Octane on the other hand create artefacts around the rim of the objects which are in front of the focus point (see figure 3.29). Since Octane takes roughly the same amount of time, Z-Defocus is not a necessary option. In case of VRay it is a choice between quality and render time.

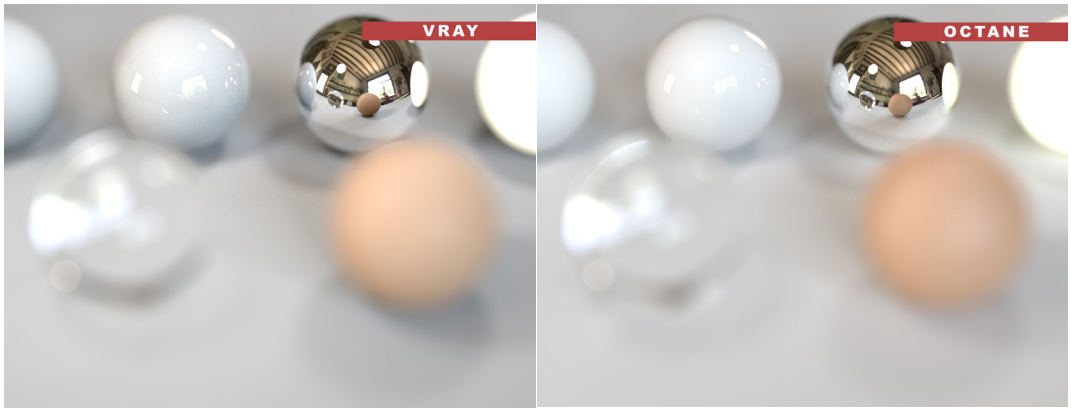

Figure 3.29.: VRay and Octane issues with Z-Defocus, close-up.

Even though RayRender has no option for depth of field itself, the Z-Defocus creates a great result, compared to the other renderers.

3.2.5 Render times

Having a closer look at the render times, it is perceivable that they vary a lot between the renderers. While Octane seems to be the fastest one in nearly every category, it is crucial to keep in mind, that it renders with GPU and is therefore inherently faster. VRay's render times increase with complexity of the scene and quality settings, but the results are physically accurate. Renderman in general is the slowest renderer. Vectorblur and Z-Defocus are vast improvements to Renderman since the render times decrease to a fraction, while the results stay similar. RayRender has decent render times, while keeping up with the physical renderers. Using the Vectorblur node is faster, but the results are further from the physically accurate rendering it creates itself.

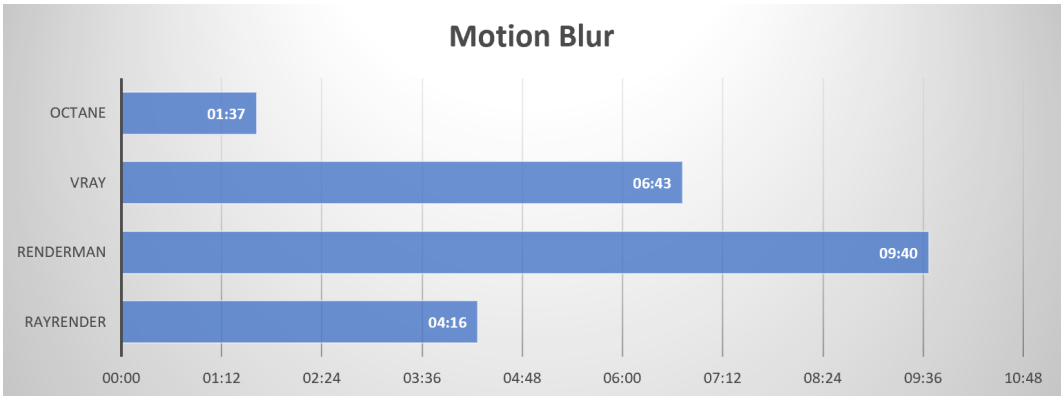

Figure 3.30.: Comparison of render times of motion blur.

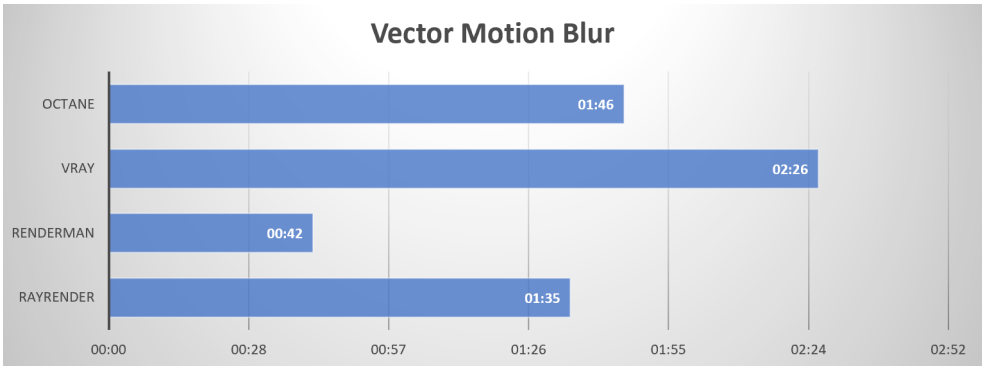

Figure 3.31.: Comparison of render times of vectorblur.

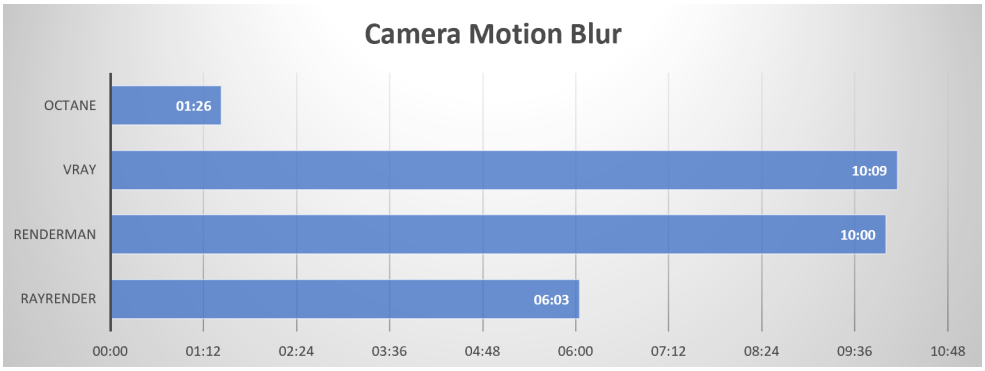

Figure 3.32.: Comparison of render times of camera motion blur.

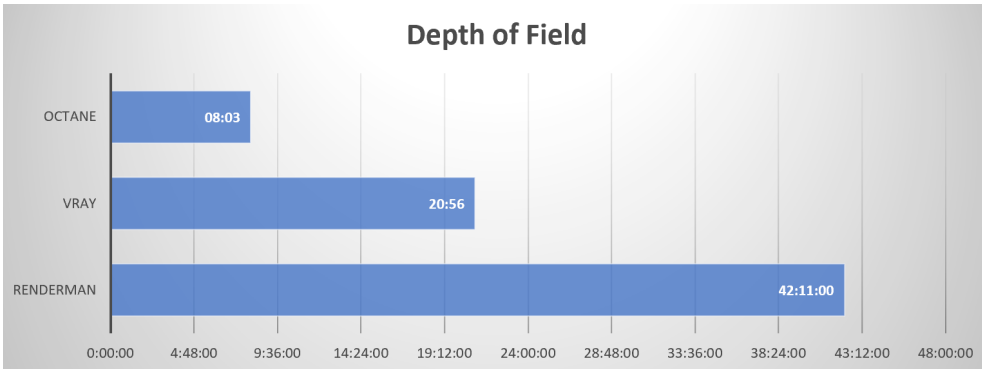

Figure 3.33.: Comparison of render times of depth of field.

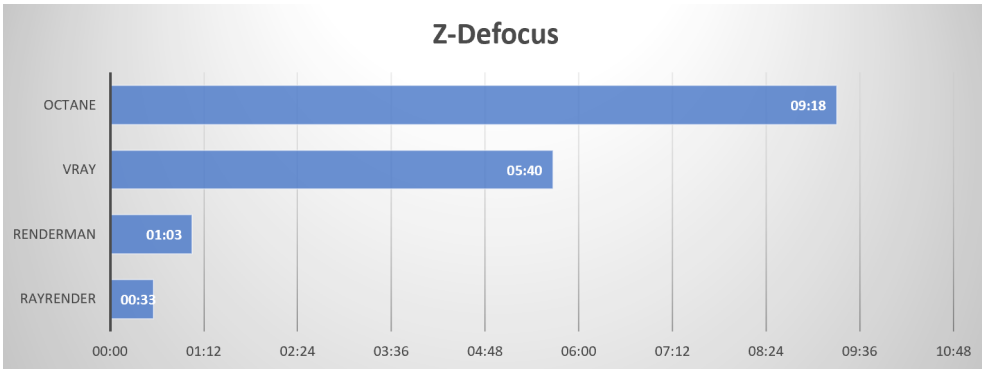

Figure 3.34.: Comparison of render times of Z-Defocus.

3.2.6 Materials & Shadows

While VRay and Octane come with their own materials, RayRender and Renderman are using the standard materials integrated in Nuke. With the BasicMaterial node, which allows to control diffuse, specular and emission individually, a bunch of simple materials can be created. If just the diffuse part is active, a lambert material is created. It is the most basic shader, which distributes the light equally across the surface without any highlights. If just the emission is active, an emissive material is created. Since RayRender and Renderman are standard ray tracer, they are not able to cast light from the emissive material. Using this emissive material with VRay or Octane will create a light source from the object. If the specular parameter is added to the diffuse, it creates a phong-like material. There are two additional shaders, which can be found under the RenderMan menu: Reflection, which creates a mirror-like material, and Refraction, which creates a transparent glass-like material (currently not supported with RayRender). The standard shaders have no option to create micro surfaces through bump, normal or displacement maps. Complex models with a low polycount can not be recreated with these shaders inside of Nuke.

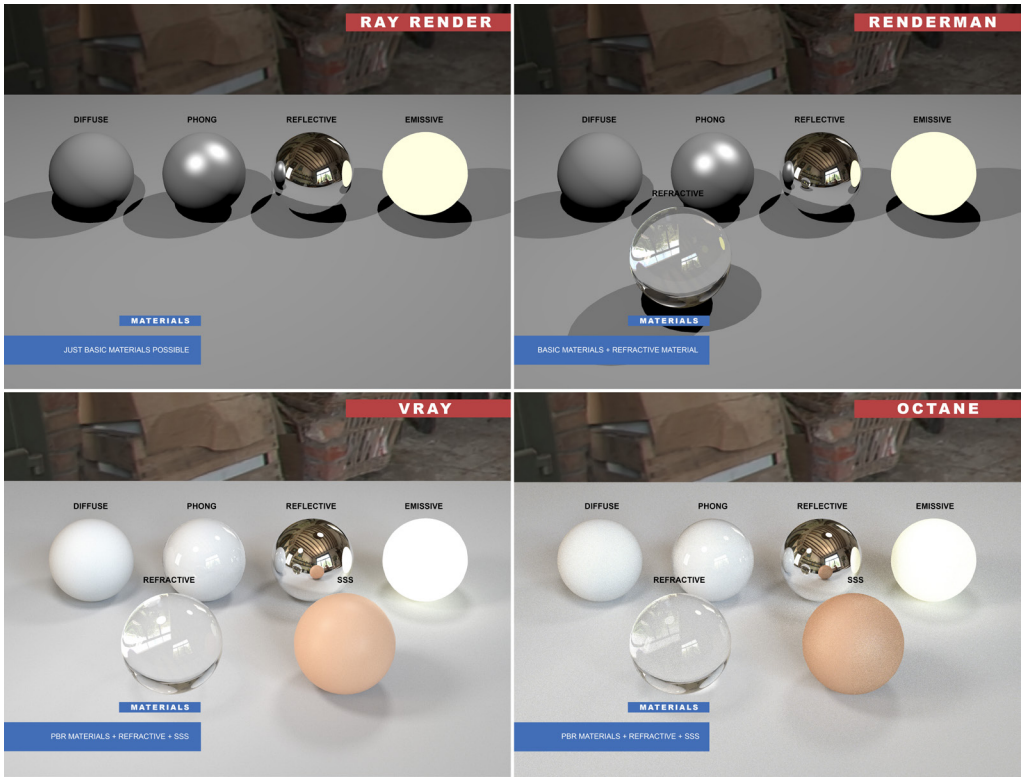

Figure 3.35.: Materials available for each renderer.

For VRay and Octane, those basic shaders can be replicated as well and adjusted as pleased. Additionally, they support subsurface scattering materials like skin. Both renderers have different settings for creating such materials, but once understood, it is easy to create great results with both engines. Shadows, as seen in figure 3.35, are also different for the physical and the non-physical renderers. The shadows created by RayRender and Renderman are always hard shadows. There are no settings to change the shadows with those renderers and according to chapter 2.5 this leads to the final evidence, that both renderers use in fact only standard ray tracing. VRay and Octane on the other hand have soft and tinted shadows due to indirect lighting. Therefore, they are considered physical renderers.

3.3 Compositing

Since an usual comparison between the render engines is not possible because of the vastly differences, a real world compositing scenario is performed to match the non-physical renderers as close as possible to the physical renderers in order to see how well they perform in the compositing pipeline. As physical renderer VRay is used, because its integration in Nuke is better than Octane's, which makes the workflow easier. It is to mention, that the same result can be achieved with Octane. As non-physical renderer RayRender is used, because the results of chapter 3.2 showed, that the renderer is much faster and achieves better results than Renderman. Since there is no motion blur or depth of field used in this scenario, the same result could not be recreated with Renderman, since it does not create render passes as mentioned in chapter 3.1. The scene does not contain refractions and has not more than a ray depth of 1 to assure the results can be similar. For the object in the scene, a free model from "free3D.com" by user "steve1244" is used as well as a HDR image from "hdrihaven.com", which is applied for image based lighting and as background.

3.3.1 VRay for Nuke

VRay makes it easy to create a composition in Nuke. Using RenderElements, there is a bunch of render passes that can be created. A beauty pass with VRay is created by combining the direct light, indirect light (global illumination), reflection, refraction, specular, self-illumination and subsurface scattering elements of the rendering. This is done by using the plus operation (Chaos Group 2018 a). Since this scene does not contain refraction,

self-illumination, or subsurface scattering, these elements are not considered and do not need to be rendered. To merge the object with the environment, a reflection of the floor is necessary. For that, a plane with a camera projection of the environment would be great, but Nuke creates a red shader on the plane wherever the projection ends. This leads to a dominant red in the reflection, which is an artefact that is not desired in the end-result (see figure 3.36).

Figure 3.36.: VRay for Nuke. Red color appears for everything not covered with camera projection.

To get rid of the artefact, a simple colored shader did the trick creating the best looking result. However, the downside is, that shadows from the environment are not visible in the reflection. This in the end leads to a slightly different result than *VRay for Maya* creates (see figure 3.40).
To create a contact shadow on the floor, an ambient occlusion (AO) pass is necessary. Ambient occlusion is added by applying the multiply operation to only have the dark elements affect the object. Since the shadow is a necessary element in the composition to create the illusion of the object being actually in the scene, it is separated into a shadow pass and then color corrected, so it matches the shadows in the scene. After that, the opacity of the shadow is lowered, so it becomes translucent and matches the other shadows in transparency. This creates the final compositing for *VRay for Nuke* (see figure 3.37).

Figure 3.37.: Compositing with VRay for Nuke. Individual render passes and final composition.

3.3.2 RayRender

RayRender has a vast disadvantage. It can not use HDR images for illumination and it is not able to create global illumination. So, the light has to be set in approximation to match the correct light direction, in order for the shadows to be correct. Additional lights need to be set up in the scene with different colors to create a sort of indirect lighting effect. I.e.: a light blue light from in front of the object tints the shadow side of the object. That simulates atmospheric light. To have better control over the lighting, a relight pass is created. It uses a 50% gray shader in order to get both, highlight and shadow informations, onto the model when using the multiply operation. This darkens the color of the object due to the gray values. A reference is required to match the colors and intensities best. In this case, Vray's global illumination pass is used as a reference. For the color information, a diffuse pass is created without any shadows on the object. The pass is very bright, which comes in handy when using the relight pass on top. The reflection pass is created with the reflection shader. This creates a highly reflective crome-like surface on the object. By looking at the reflection shader from VRay the reflection needs to be blurry and darker. Since the reflection of the eyes needs to be bright and not blurry, the reflection of the eyes is added with the diffuse pass. In the reflection pass, the eyes receive a black shader, in order to not affect

the reflection of the eyes with the reflection pass. The reflection pass also gets a camera projected environment onto the floor, so the floor and the shadow the object creates are visible in the reflection. This works perfect without any artefacts with RayRender. To separate the 3D object from the black background in the passes, an alpha pass is created and merged with every pass. To create the beauty pass the relight pass is multiplied to the diffuse pass and the reflection pass is then added through a screen operation. An ambient occlusion pass is created via the AmbientOcclusion shader to add a contact shadow on the floor plane. In order to get a shadow of the object, the ScanlineRender is needed, since RayRender is not able to create soft shadows. ScanlineRender has the ability to scatter and layer the shadows over each other with different transparencies to simulate a soft shadow. A high falloff leads to a wide spread of the individual layers which become visible. Even with a small falloff, the jittered layers are visible and the shadow should be blurred afterwards. Figure 3.38 shows the individual layers and the final composition with RayRender.

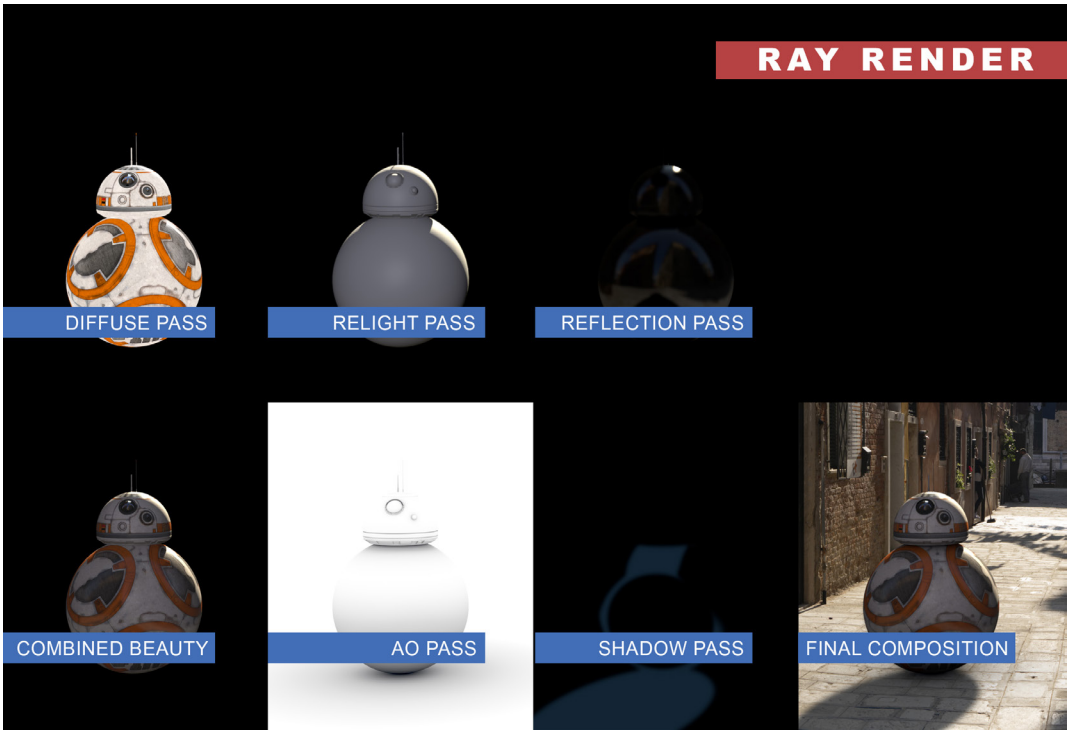

Figure 3.38.: Compositing with RayRender. Individual render passes and final composition.

ScanlineRender creates some artefacts with this sort of soft shadow which are visible in the animation. In some frames, the shadow seems to jump. During those frames, the jittering of the shadow layers are offset. Slight changes in the settings do not change this behaviour, only repositioning the light source for a tiny bit makes the jittering pop back into

position.

Figure 3.39.: Shadow issues (left) on random frames when using ScanlineRenderer.

3.3.3 VRay for Maya

To evaluate whether it is lucrative to use *VRay for Nuke*, the same scene is rendered with *VRay for Maya* and then composed in Nuke. The render settings in *VRay for Maya* are similar to the ones in Nuke. A multichannel EXR file is exported, so all the channels that were created with *Vray for Nuke* before are created as well and combined in one single OpenEXR file. This file is opened in Nuke and split with the shuffle node in order to separate the passes and combine them with the adequate operations, similar to the process with *VRay for Nuke*. Color corrections are applied and the 3D element is merged with the background. The result looks very similar to the result of *Vray for Nuke*. The differences are in the details. For one, as mentioned in 3.3.1, the reflection of the floor is not accurate with *VRay for Nuke*, since a red shading problem occurred. *VRay for Maya* does a better job with that and therefore the shadows in the environment are also visible in the reflection (see figure 3.40). This is a subtle detail, but it can make a difference when it comes to realism. Another detail is the reflection of the eyes, which are creating a different bending of

rays than the renderers in Nuke. It looks like the eye is bulged differently even though the geometry as well as the settings of the material are the same for both VRay versions.

Figure 3.40.: Reflection pass of all renderers.

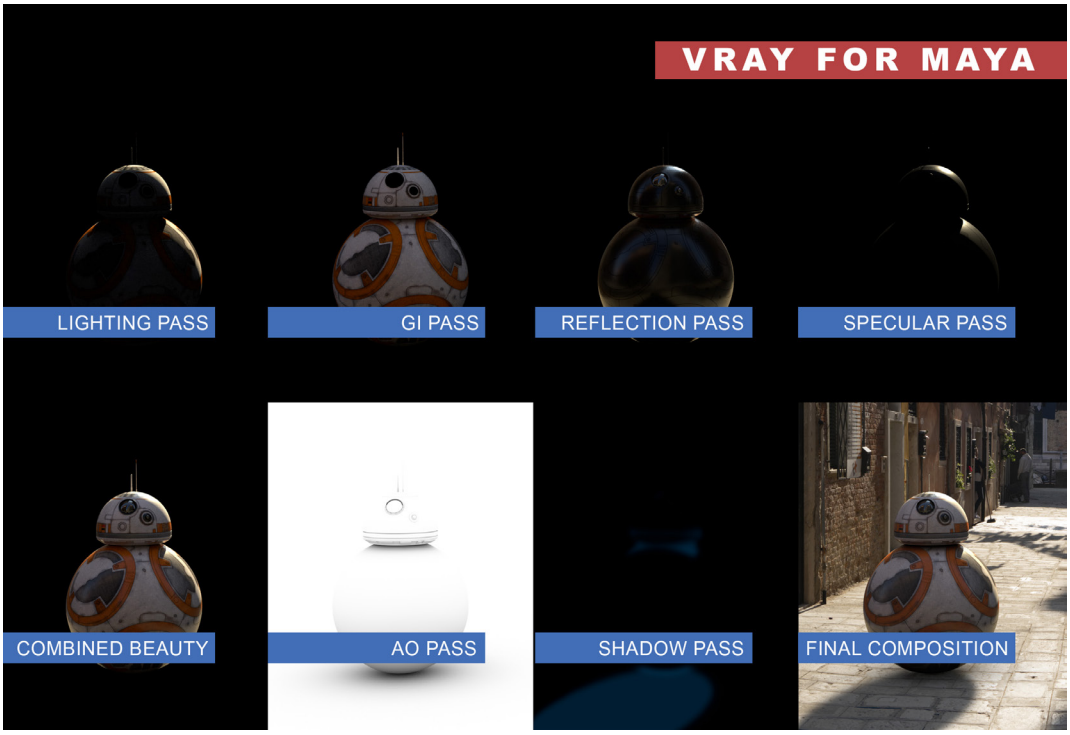

Figure 3.41.: Compositing with VRay for Maya. Individual render passes and final composition.

VRay for Maya is faster than *VRay for Nuke* (see figure 3.42).

Figure 3.42.: Final compositing for each renderer with time per frame.

3.4 Conclusion

The renderers presented can be categorised into two types: physical and non-physical ray tracing renderers. VRay and Octane are physical renderers with global illumination settings to create indirect lighting, as well as physical accurate motion blur and DoF. While Octane's render times are the fastest due to GPU, it is integrated with the stand-alone version. Even though many settings can be changed inside of Nuke, the software's standalone functions and material settings need to be understood. Switching between both software can be tedious. VRay as an integrated plug-in has the advantage of using Nuke's layout and therefore is more intuitive. Also, the settings are similar to other VRay adaptations, which makes it handy for people who are already familiar with VRay. Octane on the other hand always creates the same result no matter the plug-in, because of the standalone integration. Since the project has to be saved as an octane file, it can just be loaded, rendered, and composed in Nuke.
RayRender and Renderman as non-physical renderers do not have the option of global illumination. Also, effects like motion blur or shadows are not physically accurate, as the comparison has shown. This is due to the fact that they are using standard ray tracing, which does not solve for physical effects and a bypass is needed. Sometimes this can be achieved with post effects like Z-Defocus. RayRender has the least materials on hand, since the refraction shader is not available, and it only allows for a ray depth of 1, which also does not allow for scenes with more than one reflective material. Nevertheless, RayRender is a good alternative to the physical renderers if the project does not ask for a high level of realism and fast render times. Renderman is by far the slowest of the tested render engines and the lack of further support makes this renderer not usable anymore.

In the compositing pipeline physical ray tracer have an enormous advantage to match the light situation and the shadow direction with global illumination. For a non-physical ray tracer, a kind of a cheat has to be used to create indirect lighting. This is done with a relight pass. It can be matched best if a reference is available, like a physically accurate rendering or a grayball in the footage. For a smooth shadow another cheat is needed. The non-physical ray tracer uses standard ray tracing which only gives a hard shadow. Therefore, Nukes ScanlineRender is used to create a soft shadow. ScanlineRender creates a non-accurate effect and therefore needs to be approximated as well. The side by side comparison of the compositing shows a more unrealistic look of the 3D element of the non-physical renderer. But if the result is seen without a physical accurate version to compare to, it looks good enough for a low-budget production. To determine whether *VRay for Nuke* has disadvantages compared to *VRay for Maya*, latter is also integrated in the compositing pipeline. The results look very similar, except for some minor issues. *VRay for Maya* has the benefit of better render times. To choose between both, render time versus flexibility needs to be weighed.

4 Discussion

In the field of research, this thesis is the first to cover the topic of ray trace rendering in Nuke. However, there is one article who talks about *VRay for Nuke* and the new features it brings as well as first impressions of artist. For a beta test, some artists created a real-world test scene and emulated a production-like pipeline with last minute changes. They seemed to be very positive about the new workflow (Seymour 2015). But the article only focuses on *VRay for Nuke*. Whether other ray tracer in Nuke are used, if compositors in the industry are still positive about the workflow and if it improved the production pipeline of some companies since then is uncertain. Since the article was published, there were nearly no further insights on this matter. Therefore, a survey (see attachments) was performed and analysed for this thesis in order to get an overview over the state of ray trace rendering in Nuke in the CG industry. The survey highlights opinions from professionals on how useful they think the process is or how it could be improved. This will give further information on whether ray trace rendering in Nuke could become a future standard or at least should be considered by compositors.

4.1 Data acquisition

The goal of the survey was to get an in-depth look from people who work with this kind of software in order to figure out for which projects they are using it for, as well as how satisfied they are with the concept of rendering in a compositing software. The survey gives further insights on how and where the process is used in today's industry. This survey (created with surveymonkey.com) was not designed to get a statistically representative analysis, but rather a qualitative expertise from people who work with the software on a daily basis.

The survey's first question was designed to split the participants in those who have already worked with ray tracing renderers in Nuke and those who have not. This is to see how many of the participants have actually used the process of ray trace rendering in Nuke.
Those who have not used it before were then asked if they are at all interested in the process, the reason they have not used it yet, and what it would take them to consider using

it. The goal of those questions was to determine, what the participants think is the issue with the process without ever having used it before. Also it is interesting to know, whether they can be convinced to use it, if they are interested.
The subjects who have already worked with ray tracing renderers in Nuke were asked, which of the software compared in this thesis they have used and for which kinds of projects they used it for. This gives an overview of the kind of software which is used in the industry and what kinds of projects are more likely to use this process. To figure out if the participants were happy with the experience of using ray trace renderers in Nuke, they were asked how satisfied they were during the project and if they think it improved the pipeline. If they were not satisfied, they should give a statement on what could improve the process in their opinion. Furthermore, they were asked, if they think another ray trace renderer should have an implementation in Nuke. This was to get a further look at whether they liked the concept of ray trace rendering in Nuke. If so, they might want to have more renderers or their favourite render engines to be available in Nuke.
To get some general insights into how the participants work and whether they have used the process before or not, they were asked how close the compositing and the rendering artists work together. This gives an in-depth look into the companies' structure and whether the lack of exchange between the departments could be improved, if the rendering was done by the compositor. To give a further look on this matter, the participants were asked how much time changes take in general from the 3D department to end up back at the compositor's desk. To give an expertise on that question, the participants were asked to give an opinion on whether they think ray trace rendering in Nuke could in fact improve the process.
Finally, the subjects were asked for which department they work in order to give an overview of the participant's knowledge about the topic. The last question was asked to find out, on which system the participants operate in order to figure out if there could be a need for an implementation of rendering software in Nuke for another operating system.

4.2 Analysis of the survey

The survey was answered by 19 people, of which 14 completed all answers. About half of the people who answered have used ray trace rendering before. Amongst those, most participants have used VRay or RayRender. Two have used Renderman and one has used Cycles. This means, that VRay and RayRender are both software used by profes-

sionals. Octane is not used at all. One of the participants rather uses the physical based renderer Cycles, which is built in Blender, and managed to implement it in Nuke.
The participants have used the renderers mostly for feature films and TV/commercials. A few have used rendering in Nuke even for music videos and cooperate films. This means, that the process of rendering in Nuke has already been used in projects with big production pipelines as well as smaller projects.
The participants described the experience as overall satisfying and for some it even improved the pipeline. For one person however, the process was not satisfying at all and he is not going to use it again. A comment on that question stated, that rendering in Nuke helped when simple design decisions needed to be made. This shows, that those who have used the process before are mostly positive about it even if it is just for look development.
According to the respondents, features that could improve the workflow are better processing of the tools as well as better materials, features like GPU rendering and better shadows. Those comments mostly came from participants who have worked with RayRender or Renderman. This shows, that the missing features of RayRender and Renderman elevated in chapter 3 are exactly what the participants demanded as features for future updates.
Two-thirds of the participants do not need other renderers to be implemented in Nuke. However, one respondent would like to see the renderer, which is implemented in Modo, and another participant Cycles, which is implemented in Blender, as a ray trace renderer in Nuke. Another respondent would like to have a point or volumetric renderer with OpenVDB support. OpenVDB is a C++ library comprising of a hierarchical data structure and a suite of tools for the manipulation of volumetric data discretized on three-dimensional grids (OpenVDB 2018). The opinion of the majority on not wanting another renderer in Nuke could explain the lack of people who have tried using Octane for Nuke. But it could also be due to a lack of knowledge about the implementation of Octane inside of Nuke.
The participants who have not used ray trace rendering in Nuke were overall quite interested in the process. Some see it as a “toy” they would like to have a closer look at and as they have already read about the advantages. But they are happy with the current conditions of rendering and compositing separated. This leads to the assumption, that they have not yet used the process because they do not see a necessity in it. As a reason for not using ray trace rendering, the participants stated that they do not have a demand for it, that the elements they receive are already good enough, that Nuke is not very responsive

with real time computing, or that compositing itself is already heavy enough even without ray tracing. However, some would consider trying ray trace rendering in Nuke for specific tasks in big projects or for smaller projects that allow for experimentation.
All participants agree that compositors and render artists work together very closely in their company and that changes from the 3D department arrive on average within two days. Rendering in Nuke could improve those times, but in order to get specific results, an extensive study would need to be performed on that.
The overall statement on whether rendering in Nuke would be an advantage regarding the time spent with exchanges is bipolar. About half of the participants consider additional passes or immediate feedback to be benefits of ray tracing within Nuke. The other half states, that rendering must be done by render artists, and therefore do not think of it as beneficial. Among the participants who have used the process before as well as those who have not, were people who consider it an advantage and a disadvantage equally.
The participants work almost exclusively in the compositing department. While one of them is using Mac, a lot of the participants work with Windows and even more use Linux. This shows, that a porting of the render software within Nuke for Mac might not be as necessary as for Linux or Windows.

4.3 Conclusion of the discussion

The study shows a diversity of opinions on ray trace rendering in Nuke. The process is still fairly new and therefore a lot of professionals in the industry have not tested it yet. Regardless of whether they have used the process before or not, the opinion is split on whether the new workflow is an improvement for the pipeline or rather a “gimmick” which is nice to play around with. Some participants have experienced some issues with complex scenes, others are curious about it but have not had the opportunity to use ray tracer rendering in the Nuke pipeline, and some have used it and actually liked the workflow. Surely there are some improvements that can be made with future updates. For the moment, it seems to be a good addition to the compositing workflow, but only for smaller tasks and it will not replace rendering done by the 3D department.

5 Conclusion

5.1 Summary

In this work, the four ray tracing renderers for Nuke were compared to each other in order to identify their differences and to distinguish them into the categories physical and non-physical renderers. The aim was to present suitable ray trace renderers for using in Nuke and to examine whether rendering in a compositing software is viable for a compositing artist.

The suitability was tested with RayRender and VRay through a side by side integration in the compositing pipeline. This showed that physical renderers have a big advantage of integrating 3D elements seamlessly into the background. However, good results are achievable with non-physical renderers, if some tweaks are performed. Additionally, *VRay for Maya* was integrated in the compositing pipeline to see whether *VRay for Nuke* produces different results. The results are pretty much similar. Render times are better with *VRay for Maya* and camera projections are executed accurately. To gain more flexibility in the compositing process but with a slight reduction of the render times, it could be worth trying *VRay for Nuke*.
In order to give a more profound opinion on the results, a discussion was performed by evaluating a survey on whether ray trace renderers for Nuke are used in the industry and whether professionals think the workflow could improve current compositing pipelines.

5.2 Future work

Rendering in the same software that the compositing takes place in holds great potential. Blender as a 3D application provides the opportunity to make compositings within the 3D pipeline. This offers a lot of flexibility to the creation process. As an industry standard for compositing, Nuke has the ability to use that flexibility as well and could become even faster in adapting to client changes.
Future work could concern deeper analyses of particular mechanisms such as comparisons of CPU and GPU rendering in Nuke, or cloud rendering. This thesis is mainly focused on comparing all render engines on the lowest common factor. That excludes GPU

rendering as well as further insights into global illumination or image noise. Furthermore, a deeper analysis of complex scenes could be performed to determine how Nuke handles scenes with high numbers of polygons, volumetric effects, and particles within complex compositings. RayRender will be updated with newer versions of Nuke which could improve the ray depth settings and include more shaders. This could lead to the creation of more complex scenes, which should then be looked further into.

5.3 Conclusion

Rendering in Nuke can be a desired option, if flexibility is a crucial factor for the production. The ability to see changes in the rendering within the compositing pipeline is a great feature and has a lot of advantages. The comparison showed that the renderers can be categorised in physical and non-physical. Especially the present physical renderers in Nuke are capable of creating great renderings, which are similar to the ones created in a 3D software such as Maya. This leaves only render times and usability as considerable factors for a project to choose rendering in Nuke. The process of ray trace rendering in Nuke, especially with physical renderers, is a rather new feature and not yet tested extensively. The opinions on the usefulness of rendering in Nuke vary significantly among professionals.
Nonetheless, a closer look at ray tracing renderers in Nuke should be considered, when creating compositings with demand of high flexibility during the production or using it as look development.

6 References

Birn, Jeremy (2002): 3D Rendering (for Dummies). Available online at http://www.3drender.com/glossary/3drendering.htm, checked on 20.08.2018.

Bühler, Peter; Schlaich Patrick; Sinner Dominik (2017): Digitale Fotografie. Fotografische Gestaltung - Optik - Kameratechnik, Springer Verlag.

Carr, Robert; Hulcher, Byron (2009): Path tracing: a non-biased solution to the rendering equation. Available online at https://www.cs.rpi.edu/~cutler/classes/advancedgraphics/S10/final_projects/carr_hulcher.pdf, checked on 11.08.2018.

CGVizStudio (2015): Biased vs Unbiased Rendering Engine. Available online at https://www.cgvizstudio.com/biased-vs-unbiased-rendering-engine/, checked on 25.08.2018.

Chaos Group (2018 a): Rendering and Compositing Beauty in V-Ray for Nuke. Available online at https://docs.chaosgroup.com/display/VRAYNUKE/Rendering+and+Compositing+Beauty+in+V-Ray+for+Nuke, checked on 08.11.2018

Chaos Group (2018 b): Supported Nuke Features. Available online at https://docs.chaosgroup.com/display/VRAYNUKE/Supported+Nuke+Features, checked on 05.09.2018.

Cook, Robert L.; Torrance, Kenneth E. (1982): A Reflectance Model for Computer Graphics. ACM Transactions on Graphics, vol. 1, no.1.

Cook, Robert L.; Porter, Thomas; Carpenter, Loren (1984): Distributed Ray Tracing, Computer Graphics, vol. 8, no. 3.

Fitzgerald, Ryan (2018): What Is Compositing?. Available online at https://www.cgspectrum.edu.au/blog/what-is-compositing/, checked on 24.11.2018

Foundry (2018 a). Nuke Reference Guide - PrmanRender. Available online at https://learn.foundry.com/nuke/content/comp_environment/prmanrender/rendering_prmanrender.html,

checked on 04.09.2018.

Foundry (2018 b). Nuke Reference Guide - RayRender. Available online at https://learn.foundry.com/nuke/11.1/content/reference_guide/3d_nodes/rayrender.html, checked on 05.09.2018.

Glassner, Andrew S. (1989): An Introduction to Ray Tracing, Palo Alto CA: Academic Press.

Gugick, Jeffry (2017): Index of Refraction values (IOR). Available online at https://pixelandpoly.com/ior.html, checked on 16.07.2018.

Hachisuka, Toshiya; Ogaki, Shinji; Jensen, Henrik Wann (2008): Progressive Photon Mapping, SIGGRAPH Asia 2008, Singapore.

Hachisuka, Toshiya; Jensen, Henrik Wann (2009): Stochastic Progressive Photon Mapping. SIGGRAPH Asia 2009, Japan.

Harcourt, Tim (2016). What is GPU rendering? Available online at https://www.escape-technology.com/news/680-what-is-gpu-rendering, checked on 02.09.2018.

Hoffman, Naty (2013): Background: Physics and Math of Shading. Available online at https://blog.selfshadow.com/publications/s2013-shading-course/hoffman/s2013_pbs_physics_math_notes.pdf, checked on 30.08.2018.

Kinnane, Paul (2015): OctaneRender for Nuke 2.23.2.22 Update Details. Available online at https://www.youtube.com/watch?v=g0l7ilK3Ufo, checked on 11.06.2018.

Lafortune, Eric P.; Willems, Yves D. (1993): Bi-directional Path Tracing, Computer Graphcis.

Montes, Rosana; Ureña, Carlos (2012) An Overview of BRDF Models. Available online at http://digibug.ugr.es/bitstream/handle/10481/19751/rmontes_LSI-2012-001TR.pdf;jsessionid=055E9C97AA9496B61AD7C494167BFD17?sequence=1, checked on 24.11.2018.

Nichols, Christopher (2016): Understanding Glossy Fresnel. Available online at https://www.chaosgroup.com/blog/understanding-glossy-fresnel, checked on 30.08.2018.

OpenVDB (2018) About OpenVDB. Available online at http://www.openvdb.org/about/, checked on 18.11.2018.

Otoy (2018 a): Highlighted Features. Available online at https://home.otoy.com/render/octane-render/features/, checked on 06.09.2018.

Otoy (2018 b): OctaneRender FAQs. Available online at https://home.otoy.com/render/octane-render/faqs/, checked on 06.09.2018.

Otoy (2018 c): OctaneRender for Nuke Plugin Manual. Available online at https://docs.otoy.com/NukeH/NukePluginManual.htm, checked on 06.09.2018.

Palffy-Muhoray, Peter (2017): Introduction and Ray Optics. Available online at http://mpalffy.lci.kent.edu/optics/, checked on 16.07.2018.

Pixar (2018): Pixar Animation Studios Releases RenderMan 22. Available online at https://renderman.pixar.com/news/pixar-animation-studios-releases-renderman-22, checked on 25.11.2018.

Scratchapixel (2014): 3D Viewing: the Pinhole Camera Model. Available online at https://www.scratchapixel.com/lessons/3d-basic-rendering/3d-viewing-pinhole-camera/how-pinhole-camera-works-part-1, checked on 07.07.2018.

Scratchapixel (2015 a): The Phong Model, Introduction to the Concepts of Shader, Reflection Models and BRDF. Available online at https://www.scratchapixel.com/lessons/3d-basic-rendering/phong-shader-BRDF, checked on 24.11.2018.

Scratchapixel (2015 b): Global Illumination and Path Tracing. Available online at https://www.scratchapixel.com/lessons/3d-basic-rendering/global-illumination-path-tracing, checked on 24.11.2018.

Seymour, Mike (2012): The Art of Rendering. Available online at https://www.fxguide.com/featured/the-art-of-rendering/, checked on 12.08.2018.

Seymour, Mike (2013): The state of rendering – part 2. https://www.facebook.com/fxguide/. Available online at https://www.fxguide.com/featured/the-state-of-rendering-part-2/, checked on 20.08.2018.

Seymour, Mike (2015): Nuke and V-Ray. Available online at https://www.fxguide.com/featured/nuke-and-v-ray/, checked on 26.10.2018.

Solid Angle (2018). Arnold for maya user guide 5 - Ray Depth. Available online at https://support.solidangle.com/display/A5AFMUG/Ray+Depth, checked on 08.09.2018.

Tuttle, Claurissa (2003): An Analysis of Path Tracing and Photon Mapping in an Attempt to Simulate Full Global Illumination. Available online at http://www.cs.utah.edu/~ctuttle/research03/finalreport.pdf, checked on 19.07.2018.

Vlans, Michal (2018): Bidirectional Path Tracing, CESCG 2018, Brno.

Von Übel, Max (2018): 25 Best 3D Rendering Software Tools of 2018. Available online at https://all3dp.com/1/best-3d-rendering-software/, checked on 25.08.2018.

Waters, Zack (2003 a): Realistic Raytracing. Available online at http://web.cs.wpi.edu/~emmanuel/courses/cs563/write_ups/zackw/realistic_raytracing.html, checked on 18.07.2018.

Waters, Zack (2003 b): Photon Mapping. Available online at https://web.cs.wpi.edu/~emmanuel/courses/cs563/write_ups/zackw/photon_mapping/PhotonMapping.html, checked on 09.08.2018.

Wright, Steve (2016): RayRender workflow tips. Available online at https://www.lynda.com/Nuke-tutorials/RayRender-workflow-tips/474425/546843-4.html, checked on 05.09.2018.

Wynn, Chris (n.d.): An Introduction to BRDF-Based Lighting. Available online at http://vision.ime.usp.br/~acmt/hakyll/assets/files/wynn.pdf, checked on 12.08.2018.

Attachment

This thesis has attached the following items on a mirco sd card:

- The complete thesis as a pdf document.
- The original images used in this thesis.
- The survey of chapter 4 with collective data.
- The pdf documents of online resources if available.
- The Nuke files for creating the comparison and compositing results.
- The Octane files for creating the comparison results.
- The 3D elements as well as texture and HDR images in a footage folder for the particular topic each.
- The final compositing as animated video of each renderer as well as breakdown of the individual compositing techniques in video form.