Feasibility Study of Real Time Path Tracing

Or: How Much Noise Is Too Much?

Sven-Hendrik Haase

A thesis presented for the degree of
Bachelor of Computer Science

Department of Informatics
University of Hamburg
Germany
2015-20-08

Primary Supervisor: Prof. Dr. rer. nat. Leonie Dreschler-Fischer
Secondary Supervisor: Prof. Dr. Thomas Ludwig

Abstract

This study aims to investigate the viability of a physically-based technique called **path tracing** in lieu of or in corporation with classical techniques in interactive media such as video games and visual effects tools.

Real time path tracing has been prohibitively expensive in regards to computational complexity. However, modern GPUs and even CPUs have finally gotten fast enough for real time path tracing to become a viable alternative to traditional real time approaches to rendering. Based on that assumption, this thesis presents the idea, algorithm and complexity behind path tracing in the first part and extrapolates feasibility and suitability of real time path tracing on consumer hardware according to the current state of technology and trends in the second part.

As part of the research, the author has implemented a path tracing 3D engine in modern C++ in order to empirically test the assumptions made in this thesis. The study found path tracing to be a viable rendering technique for average commodity hardware in approximately 4 years.

Acknowledgments

I would like to express my sincere gratitude to the teachers throughout school and university for the knowledge they've passed on.

I thank my friends for the laughs, horrible mistakes and awesome successes we shared with one another.

Furthermore, none of this would have been possible without the incredible efforts and love of my parents who have supported me throughout the years and enabled me to live a carefree life until I was ready to fend for myself.

Lastly, but certainly not least, I would like to declare my gratefulness to Alisa, whose endless love has given my life a new meaning.

We all make choices in life, but in the end, our choices make us.

Andrew Ryan

Contents

Acronyms

AABB axis-aligned bounding box

AO ambient occlusion

BRDF bidirectional reflection distribution function

BSP binary space partitioning

CPU central processing unit

CSG constructive solid geometry

FPS frames per second

GI global illumination

GPU graphics processing unit

HDR high dynamic range

MLT Metropolis light transport

PBR physically based rendering

RT real time

RGB red-green-blue

SIMD single instruction multiple data

SPP samples per pixel

SSAO screen-space ambient occlusion

1 Introduction

As part of the quest for ever-improving game graphics, researchers, graphics hardware developers and video game developers alike have been coming up with more and more convoluted and technically challenging ways of improving the graphics in interactive media such as games and visualizations in order to give users a deeper sense of immersion or to provide special effects artists with faster feedback.

While rendering techniques are currently shifting from the traditional fixed pipeline approach towards the new, fully programmable approach that lets developers implement deferred renderers that can more closely mimic reality by using multiple combined shading and lighting algorithms and rendering the scene multiple times for different buffers, the fundamental concept of rasterization-based rendering has largely remained the same.

The real world photon-collecting approach that actual cameras use has so far not been adopted for interactive media by the industry in any capacity because the computational cost has historically been prohibitively expensive. It is, however, used extensively (and has been in use since decades) for offline, non-interactive rendering of computer-generated movies and visualizations of scientific simulations.

This study assumes that the next logical step for the industry will be to adopt this method for real time media as well. For the purpose of this thesis, a renderer is considered *real time* when it manages to render a frame within $16.67ms$ since that equals 60 frames per second (FPS) which is the current de facto standard refresh rate for most available computer screens. Conversely, a renderer is called *offline* when it is not designed for interactive rendering which usually means that it will renderer an image or a batch of images over the course of a few days. The differences of real time and offline path tracing renderers will be explained in the next chapter.

1.1 Motivation

Real time path tracing (and physically based rendering in general) offers many benefits over traditional real time rendering methods such as better visuals and simpler implementation but also allows for completely new types of graphics such as realistic caustics [1] and even light dispersion [2] (using a prism, for instance) since path tracers might simulate wavelengths instead of plain red-green-blue (RGB) colors. Modern video games tend to rely on a growing number of tricks to keep them visually appealing as the consumer grows more demanding. They're called *tricks* in this study because they merely trick the beholder into seeing something that appears to be physically accurate when it is, in fact, not the result of a physically-based calculation and as such this study aims to keep tricks and emergent phenomena separated by language. Some notable tricks include screen-space ambient occlusion (SSAO) [3], motion blur [4], lens flares [5], chromatic aberration [6], depth of field [7] and light mapping [8].

2 Real Time Path Tracing Explained

This chapter will explain the concepts, mathematics, physics and algorithms behind path tracing, how real time path tracing differs from offline path tracing and the trade-offs made to achieve acceptable performance. It will also explain how path tracing differs from rasterization and other global illumination (GI) techniques.

2.1 Physically Based Approach

In our physical world, we see pictures because our eyes collect photons emitted by light sources which then bounce around various surfaces until they eventually hit our eyes' photoreceptor cells. On every bounce, a bit of light is absorbed which is why light loses intensity when it bounces. Some surfaces absorb a particular band of wavelengths of the light when it bounces which we perceive as a change in the light's color. Cameras work exactly like this as far as collection of photons is concerned.

This physical approach would be extremely wasteful and computationally complex to simulate, however, since most photons never reach an observer. Consider, for instance, that only an extremely small percentage of all the photons sent by the Sun actually reach Earth and an even smaller percentage of those are ever observed (although photons don't have to be observed to have a physical effect, of course). Since we only care for photons that are relevant to the image that we are trying to render, it makes more sense to use *backwards ray tracing* in which rays (which simulate streams of photons) are shot from the observer into the scene for every sensor. It is called *backwards* because the rays go the reverse direction compared to their physical counterparts.

This is efficient since we usually only care about a single observer (the scene camera) for which we will trace every single ray that it can possibly perceive. In computer graphics terms, we will trace a ray for every pixel of the camera (and for now we will assume that the viewport is exactly the same resolution as the camera simplicity's sake). For every ray, we check for intersections with geometry and then either bounce a few more times or shoot directly towards a light. We might do this multiple times per pixel to improve image quality. This is called *sampling*. The more iterations we spend on sampling, the better the quality of our image becomes. This is called *converging*. There are a few approaches that improve on this such as bidirectional path tracing [9] and the Metropolis light transport (MLT) [10].

2.2 Theoretical Basis

2.2.1 The Rendering Equation

The fundamental problem solved by path tracing is the *rendering equation* originally described by James Kajiya [11]. This thesis uses the form from Wikipedia [12] since the author considers it easier to read:

$$L_\mathrm{o}(\mathbf{x},\, \omega_\mathrm{o},\, \lambda,\, t) \;=\; L_\mathrm{e}(\mathbf{x},\, \omega_\mathrm{o},\, \lambda,\, t) \;+\; \int_\Omega f_\mathrm{r}(\mathbf{x},\, \omega_\mathrm{i},\; \omega_\mathrm{o},\, \lambda,\, t)\, L_\mathrm{i}(\mathbf{x},\, \omega_\mathrm{i},\, \lambda,\, t)\, (\omega_\mathrm{i} \,\cdot\, \mathbf{n})\; \mathrm{d}\,\omega_\mathrm{i}$$

For our purposes, this can be simplified by removing the time and wavelength components which we will not make use of:

$$L_\mathrm{o}(\mathbf{x},\, \omega_\mathrm{o}) \;=\; L_\mathrm{e}(\mathbf{x},\, \omega_\mathrm{o}) \;+\; \int_\Omega f_\mathrm{r}(\mathbf{x},\, \omega_\mathrm{i},\; \omega_\mathrm{o})\, L_\mathrm{i}(\mathbf{x},\, \omega_\mathrm{i})\, (\omega_\mathrm{i} \,\cdot\, \mathbf{n})\; \mathrm{d}\,\omega_\mathrm{i}$$

This equation can be broken down into its individual parts to make it easier to explain and understand:

$$\boxed{L_\mathrm{o}(\mathbf{x},\, \omega_\mathrm{o})} = \boxed{L_\mathrm{e}(\mathbf{x},\, \omega_\mathrm{o})} + \boxed{\int_\Omega \boxed{f_\mathrm{r}(\mathbf{x},\, \omega_\mathrm{i},\; \omega_\mathrm{o})}\boxed{L_\mathrm{i}(\mathbf{x},\, \omega_\mathrm{i})}\boxed{(\omega_\mathrm{i} \,\cdot\, \mathbf{n})}\; \mathrm{d}\,\omega_\mathrm{i}}$$

$\boxed{L_\mathrm{o}(\mathbf{x},\, \omega_\mathrm{o})}$ is the **outgoing light** with $\mathbf{x}$ being a point on a surface from which the light is reflected from into direction ω_o.
$\boxed{L_\mathrm{e}(\mathbf{x},\, \omega_\mathrm{o})}$ is the **emitted light** from point $\mathbf{x}$. Usually surfaces don't emit light themselves unless they are area lights.
$\boxed{\int_\Omega \ldots \mathrm{d}\,\omega_\mathrm{i}}$ is the integral over Ω which is the hemisphere at $\mathbf{x}$ (and thusly centered around $\mathbf{n}$). All possible values for ω_i are therefore contained in Ω.
$\boxed{f_\mathrm{r}(\mathbf{x},\, \omega_\mathrm{i},\; \omega_\mathrm{o})}$ is the **bidirectional reflection distribution function (BRDF)** which determines how much light is reflected from ω_i to ω_o at $\mathbf{x}$.
$\boxed{L_\mathrm{i}(\mathbf{x},\, \omega_\mathrm{i})}$ is the **incoming light** at $\mathbf{x}$ from ω_o. It is not necessarily *direct light*. The rendering equation also considers *indirect light* which is light that has already been reflected.
$\boxed{(\omega_\mathrm{i} \cdot \mathbf{n})}$ is the **normal attenuation** at $\mathbf{x}$. The incoming light ω_i is weakened depending on the cosine of the angle between ω_i and the surface normal $\mathbf{n}$.

Path tracing offers a numerical solution to the integral found in this equation. For every pixel, every bounce and every sample of the camera, the rendering equation is solved. It becomes apparent why this is an expensive algorithm to run. For practical reasons, not every possible value for Ω is sampled since this would take a vast amount of time to calculate at physical photon density. Instead, only a few possible values for Ω are calculated each bounce. Depending on the exact algorithm used, usually only a low number of samples (approximately 20) is required for the image to converge to an acceptable level of quality.

2.2.2 Algorithm

The algorithm for path tracing and be Python-like pseudocode:

```
max_depth = 5

def trace_ray(ray, depth):
    if depth >= max_depth:
        # Return black since we haven't hit anything but we're
        # at our limit for bounces
        return RGB(0, 0, 0)

    collision = ray.check_collision()
    if not collision:
        # If we haven't hit anything, we can't bounce again so
        # we return black
        return RGB(0, 0, 0)

    material = collision.material;
    emittance = material.emittance # kill this

    # shoot a ray into random direction and recurse
    next_ray = Ray()
    next_ray.origin = collision.position
    next_ray.direction = random_vector_on_hemisphere(collision.normal)

    reflectance_theta = dot(next_ray.direction, collision.normal)
    brdf = 2 * material.reflectance * reflectance_theta
    reflected = trace_ray(next_ray, depth + 1)
```

```
    return emittance + (brdf * reflecte)

for pixel in pixels:
    trace_path(ray_from_pixel(pixel), 0)
```

2.3 Properties of Path Tracing

Advantages and Disadvantages blah

Slow Sharp caustics hard Subsurface scattering Light spectrum stuff: chromatic aberration, fluorescence, iridescence Accurate global lighting Good for dynamic scenes

Figure explaining how path tracers work

2.3.1 Comparison to Traditional Ray Tracing

Figure explaining how ray tracers work

2.3.2 Comparison to Rasterization

2.3.3 Comparison to Other Global Illumination Algorithms

This section compares some of the more popular GI algorithms beside ray tracing and path tracing. In general, GI is considered to be a group of algorithms that calculate direct light as well as indirect light for computer graphics scenes. However, not all algorithms that fulfill this purpose are in fact physically accurate. We will therefore take a look at how some of these algorithms compare to path tracing.

The algorithms that are compared to path tracing in this section are: *photon mapping*, *radiosity* and *ambient occlusion*. These were chosen due to their widespread use and their varied approaches. Other GI algorithms include: Lightcuts (and its variants), Point Based Global Illumination and Spherical harmonic lighting.

To note: This thesis considers path tracing at the current state of research which means that path tracing, bidirectional path tracing and the MLT are shortened to just *path tracing* and will therefore not be individually compared.

Photon Mapping

Photon mapping is a two-step process that was developed in 1996 by Henrik Wann Jensen [13] as an approximate way to simulate charged particles (*photons*) traversing the scene.

In the first step, every photon carries a *charge* and is traced through the scene. On every collision with scene geometry, it is stored to the *photon map* at that location. Afterwards, the photon is either reflected, refracted, scattered or absorbed depending on the material and loses a bit of its charge. The photon map serves as a cache for the second step in which a ray tracing-like process is used to calculate the radiance of the resulting image.

Compared to path tracing, photon mapping has a few advantages and a few disadvantages. In particular, photon mapping can simulate subsurface scattering and volume caustics which path tracing can't accurately calculate. On the other hand, photon mapping is unsuitable for real time applications with dynamic scenes since the photon map can only be used as long as the scene geometry or light position doesn't change. In the case the scene geometry changes, the cache is invalidated and a new photon map has to be calculated which is a slow process.

Radiosity

Made originally for simulating heat transfer in 1984 by Goral et al.[14], radiosity is one of the oldest algorithms for calculating GI. It outputs a light value for every patch (a smaller part of a surface) on a cache or map. It is a physically accurate way of simulating light transfer but cannot simulate volume scattering, fog, caustics, transparent objects or mirrors. These limitations make it unsuitable to use in complex modern scenes. Additionally, the cache is invalidated whenever the scene changes and therefore it is also usually not usable for dynamic scenes.

Ambient Occlusion

The idea for ambient occlusion (AO) was first presented by Gavin Miller in 1994 [15]. It is meant as an algorithm to calculate realistic occlusion of every point in a scene and cannot generate an accurate image on its own. It is usually used with a classic rasterization renderer whose output image is multiplied with the result of the AO and its resulting image is multiplied. The output of this algorithm is sometimes called the *ambient occlusion map* which serves as a cache. As such, rendering is extremely fast once the cache has been calculated. However, this cache is invalidated once the scene changes and is the algorithm is therefore unsuitable for dynamic scenes.

For real time applications, a variant of AO called screen-space ambient occlusion (SSAO) is usually used. While SSAO is inaccurate from a physical point of view, it results in some very fast and acceptable approximations that are suitable for real time applications.

2.4 History of Path Tracing

As with so many things in computer science and science in general, the modern idea of physically based rendering using path tracing builds upon many important past discoveries and algorithms such as ray tracing and ray casting. Arthur Appel is generally credited as being the father of *ray casting* as he was the first to describe the algorithm in a 1968 paper [16].

Ray casting is an important idea needed for *ray tracing* which was first published in a paper in 1980 by Turner Whitted [17].

Building upon ray tracing, an improved algorithm was published in 1986 by James T. Kajiya which used ray tracing combined with a Monte Carlo algorithm in order to create a new algorithm that was called *Monte Carlo ray tracing* [11]. Nowadays, Monte Carlo ray tracing is better known as *path tracing*.

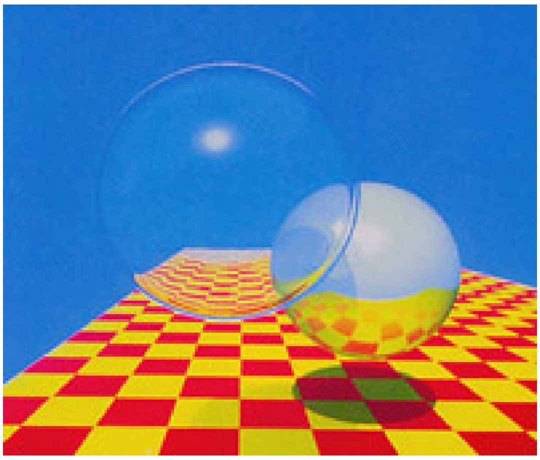

Figure 2.1: Turner Whitted's original 1980 [17] image showing off the usage of ray tracing for reflection, refraction and shadows.

It took another decade for path tracing to become the physically based rendering approach that it is known for today. In 1996, Eric Lafortune improved the algorithm by suggesting the usage of bidirectional path tracing [9] and finally the MLT was suggested in 1997 by Eric Veach and Leonidas J. Guibas [10] to improve performance in complex scenes.

This was the last notable improvement to the algorithm, though many micro optimizations have since been published. All of these achievements and improvements are generally collapsed into the term *path tracing* since they do not diverge from the general algorithm but instead improve upon it.

It took longer still for the industry to become interested in path tracing. The early interest in ray tracing was of mostly academical and recreational nature. One of the most notable creations of the early days of ray tracing is The Juggler created and published by Eric Graham in 1986 [18] on an Amiga 1000. It was a pre-rendered animation using ray tracing. Eric Graham stated that it took the Amiga 1 hour to render each frame [18].

Figure 2.2: Eric Graham's Juggler

While the animation seems very primitive compared to the animations of today, it was exceptional at the time. Ernie Wright's statement about The Juggler provides some contemporary context:

> Turner Whitted's paper (1980) is widely regarded as the first modern description of ray tracing methods in computer graphics. This paper's famous image of balls floating above a checkerboard floor took 74 minutes to render on a DEC VAX 11/780 mainframe, a $400,000 computer. The Juggler would appear a mere six years later, created and displayed on a $2000 Amiga. ([18])

The first feature-length computer-animated film, *Toy Story*, released in 1995 [19], is sometimes miscredited as being the first film using a ray tracing-like algorithm. However, it actually used traditional scanline rendering. The first feature-length film using ray tracing, *Cars*, was released much later, in 2006 [20] [21] and started a wave of interest in the movie industry.

The first example of *real time* path tracing was likely produced by the demo scene [22] which was quick to adopt it [23] for the purpose of producing complex graphics rendered and generated on the fly. One notable example of this is the WebGL Path Tracing by Evan Wallace made in 2010 [24] which runs in most modern web browsers, making path tracing very accessible.

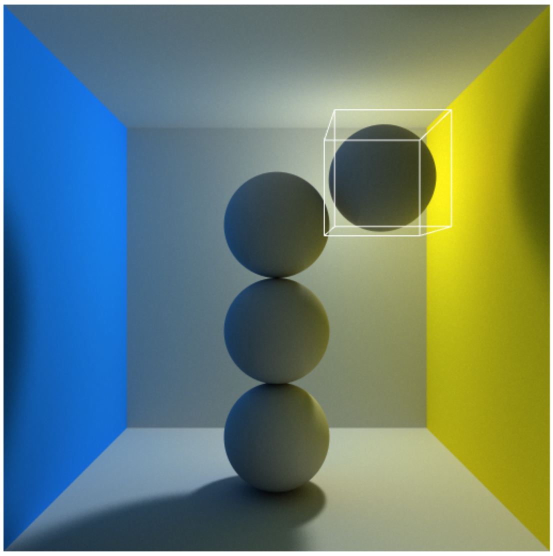

Figure 2.3: WebGL Path Tracer by Evan Wallace

Another example is the demo *5 faces* by Fairlight from 2013 [25] which uses a real time ray tracer running on the GPU to render a complex scene at 30 FPS.

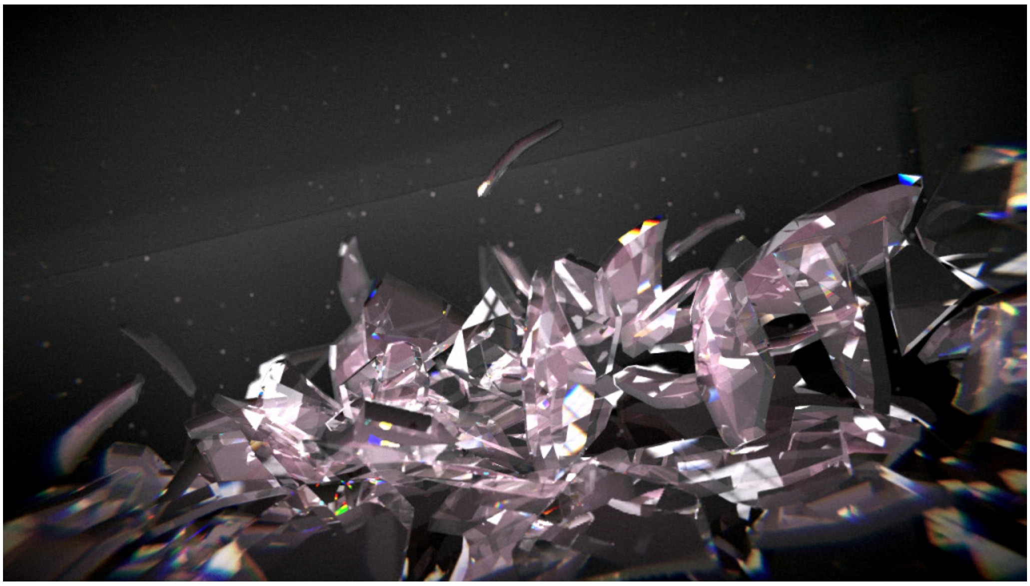

Figure 2.4: 5 faces by Fairlight

In the past, some critics have offered critical insights about why it might not be a viable alternative to rasterization on consumer hardware in the short term [26] [27].

2.5 Current State of Technology

lol Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

3 Research

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

3.1 Results

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

3.2 Evaluation

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

4 Conclusion

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

4.1 Outlook

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

List of Figures

List of Tables

Bibliography

[1] Wikipedia. Caustic (optics) — wikipedia, the free encyclopedia, 2015. [Online; accessed 16-September-2015].

[2] Wikipedia. Dispersion (optics) — wikipedia, the free encyclopedia, 2015. [Online; accessed 16-September-2015].

[3] Wikipedia. Screen space ambient occlusion — wikipedia, the free encyclopedia, 2015. [Online; accessed 16-September-2015].

[4] Wikipedia. Motion blur — wikipedia, the free encyclopedia, 2015. [Online; accessed 16-September-2015].

[5] Wikipedia. Lens flare — wikipedia, the free encyclopedia, 2015. [Online; accessed 16-September-2015].

[6] Wikipedia. Chromatic aberration — wikipedia, the free encyclopedia, 2015. [Online; accessed 16-September-2015].

[7] Wikipedia. Depth of field — wikipedia, the free encyclopedia, 2015. [Online; accessed 16-September-2015].

[8] Wikipedia. Lightmap — wikipedia, the free encyclopedia, 2015. [Online; accessed 16-September-2015].

[9] Eric Lafortune. Mathematical models and monte carlo algorithms for physically based rendering. Technical report, 1996.

[10] Eric Veach and Leonidas J. Guibas. Metropolis light transport. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '97, pages 65–76, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[11] James T. Kajiya. The rendering equation. In *Computer Graphics*, pages 143–150, 1986.

[12] Wikipedia. Rendering equation — wikipedia, the free encyclopedia, 2015. [Online; accessed 30-September-2015].

[13] Henrik Wann Jensen. Global illumination using photon maps. In *Proceedings of the Eurographics Workshop on Rendering Techniques '96*, pages 21–30, London, UK, UK, 1996. Springer-Verlag.

[14] Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, and Bennett Battaile. Modeling the interaction of light between diffuse surfaces. *SIGGRAPH Comput. Graph.*, 18(3):213–222, January 1984.

[15] Gavin Miller. Efficient algorithms for local and global accessibility shading. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '94, pages 319–326, New York, NY, USA, 1994. ACM.

[16] Arthur Appel. Some techniques for shading machine renderings of solids. In *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, AFIPS '68 (Spring), pages 37–45, New York, NY, USA, 1968. ACM.

[17] Turner Whitted. An improved illumination model for shaded display. *Commun. ACM*, 23(6):343–349, June 1980.

[18] Ernie Wright. The juggler, 1998. [Online; accessed 19-September-2015].

[19] Wikipedia. Toy story — wikipedia, the free encyclopedia, 2015. [Online; accessed 19-September-2015].

[20] Wikipedia. Cars (film) — wikipedia, the free encyclopedia, 2015. [Online; accessed 19-September-2015].

[21] P.H. Christensen, J. Fong, D.M. Laur, and D. Batali. Ray tracing for the movie 'cars'. In *Interactive Ray Tracing 2006, IEEE Symposium on*, pages 1–6, Sept 2006.

[22] Wikipedia. Demoscene — wikipedia, the free encyclopedia, 2015. [Online; accessed 30-September-2015].

[23] Stepan Hrbek. Realtime radiosity, 2015. [Online; accessed 30-September-2015].

[24] Evan Wallace. Webgl path tracing, 2010. [Online; accessed 30-September-2015].

[25] Fairlight. 5 faces, 2013. [Online; accessed 30-September-2015].

[26] Peter da Silva. Raytracing vs rasterization, 2006. [Online; accessed 30-September-2015].

[27] Jeff Atwood. Real-time raytracing, 2008. [Online; accessed 30-September-2015].