



Delft University of Technology

Image-based representations for efficient rendering and editing

Scandolo, Leonardo

DOI

[10.4233/uuid:1947fa3e-2e3c-4007-bc4f-12cb93f34be6](https://doi.org/10.4233/uuid:1947fa3e-2e3c-4007-bc4f-12cb93f34be6)

Publication date

2019

Document Version

Final published version

Citation (APA)

Scandolo, L. (2019). *Image-based representations for efficient rendering and editing*. [Dissertation (TU Delft), Delft University of Technology]. <https://doi.org/10.4233/uuid:1947fa3e-2e3c-4007-bc4f-12cb93f34be6>

Important note

To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright

Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy

Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

IMAGE-BASED REPRESENTATIONS FOR EFFICIENT RENDERING AND EDITING

IMAGE-BASED REPRESENTATIONS FOR EFFICIENT RENDERING AND EDITING

Dissertation

for the purpose of obtaining the degree of doctor
at Delft University of Technology,
by the authority of the Rector Magnificus, prof. dr. ir. T.H.J.J. van der Hagen,
chair of the Board of Doctorates
to be defended publicly on September 18, 2019 at 10:00 o'clock

by

LEONARDO SCANDOLO

Licenciate in Computer Science, Universidad Nacional de Rosario, Argentina

born in Rosario, Argentina.

This dissertation has been approved by the promotor.

Composition of the doctoral committee:

Rector Magnificus,	chairperson
Prof. dr. E. Eisemann,	Technische Universiteit Delft, promotor

Independent members:

Prof. dr. ir. M. C. Veraar,	Technische Universiteit Delft
Prof. U. Assarsson,	Chalmers Institute of Technology, Sweden
Prof. dr. ing. M. Stamminger,	Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany
Dr. ing. J. Bikker,	Universiteit Utrecht, The Netherlands
Prof. dr. ir. D.H.J. Epema,	Technische Universiteit Delft, reserve member
Dr. J. Munkberg,	NVIDIA Corporation

Other members:

Dr. R. Marroquim,	Technische Universiteit Delft
-------------------	-------------------------------



This work was partly supported by EU FP7-323567 project Harvest4D, the Intel VCI at Saarland University and VIDI NextView, funded by NWO Vernieuwingsimpuls.

Printed by: (to be determined)

Copyright © 2019 by L. Scandolo

ISBN 000-00-0000-000-0

An electronic version of this dissertation is available at
<http://repository.tudelft.nl/>.

CONTENTS

Summary	ix
Samenvatting	xi
Preface	xiii
1 Introduction	1
1.1 Image-based resolution strategies	3
1.2 Presented applications	4
1.3 Personal contributions	8
2 Compressed Multiresolution Hierarchies	9
2.1 Introduction	10
2.2 Related Work	11
2.3 Compressed Multiresolution Hierarchies	13
2.3.1 Construction.	14
2.3.2 Compressed Quadrees	17
2.3.3 Filtering	19
2.4 Shadow map stacks	20
2.5 Multiresolution Hierarchies Evaluation	22
2.6 Merged Multiresolution Hierarchies	25
2.6.1 Hashing	27
2.6.2 Subtree Matching	29
2.6.3 Serialized Tree Creation	31
2.6.4 Merged Multiresolution Hierarchies Evaluation	32
2.7 Conclusion and Future Work	35
3 Quad-Based Fourier Transform	37
3.1 Introduction	38
3.2 Related Work	39
3.2.1 Diffraction Modeling and Rendering.	39
3.2.2 Acceleration Techniques for Fourier Transform	40
3.3 Background.	40
3.3.1 Standard Fourier Transform	40
3.3.2 Diffraction with Fourier Transform.	41
3.4 Our approach	41
3.4.1 Primitive-based Fourier Transform	42
3.4.2 Far-Field Diffraction Rendering	43
3.4.3 Accelerations	44
3.4.4 Occlusion and Area Lights	46
3.4.5 Near-Field Diffraction	47

3.5	Results	49
3.5.1	Far-Field Diffraction	49
3.5.2	Near-Field Diffraction	52
3.6	Applications	53
3.6.1	Glare Rendering	53
3.6.2	Ringling at Dynamic Aperture Edges	54
3.6.3	Bloom/Glow Rendering	54
3.7	Discussion and Limitations	55
4	Gradient-Guided Local Disparity Editing	57
4.1	Introduction	58
4.2	Related Work	59
4.3	Disparity Editing	60
4.3.1	Disparity Map	60
4.3.2	Disparity Tools	61
4.3.3	Disparity Map Optimization	62
4.3.4	Stereo image creation	64
4.3.5	Implementation Details	66
4.4	Results	67
4.4.1	Memory usage	70
4.4.2	Timing	71
4.4.3	User studies	71
4.4.4	Limitations and Future Work.	73
4.5	Conclusion	74
5	Conclusion	75
	Epilogue	89
	Acknowledgements	91
	Curriculum Vitæ	93
	List of Publications	95

SUMMARY

OVER the years, technological improvement has led to the ability to acquire, create and store vast amounts of geometrical and appearance data to use in graphical applications. Nevertheless, efficiently creating images when using such large amounts of data remains an ongoing topic of research, given the computational resources required. This dissertation will focus on a particular kind of algorithms in order to tackle this problem: image-based representations.

Entities represented in a virtual 3-dimensional world can be projected into a regular 2-dimensional grid to form an image. This results in a much more compact, albeit incomplete, representation of the original information, since an image is a piece-wise constant discretization of the underlying data. Nevertheless, computing properties of the 3-dimensional data via its 2-dimensional projection is much more efficient. Still, it is important to understand when and how to use these types of representations, since the discretization of the original data can lead to artifacts in the computed results.

This work will focus on three key aspects of computer-graphics algorithms where using appropriate image-based representations can result in increased performance: memory requirements, computational efficiency, and interactivity. To do so, it will describe image-based solutions to different relevant problems in the field of computer graphics. Firstly, by exploring how using 2-dimensional intermediate representation can reduce memory requirements for storing large static shadow information in comparison to state-of-the-art voxel representations. Secondly, a hierarchical image-based approach for efficiently computing diffraction patterns will be demonstrated, which can outperform FFT-based solutions. Lastly, an efficient interactive optimization method for editing disparity when creating stereographic images will be described.

These algorithms will be described and evaluated in detail, in order to give the reader insight into the usage of image-based representations.

SAMENVATTING

OVER de jaren heen heeft technologische vooruitgang geleid tot de mogelijkheid om enorme hoeveelheden geometrische en uiterlijke gegevens te verkrijgen, te creëren en op te slaan voor toepassingen in grafische applicaties. Desalniettemin blijft het efficiënt creëren van afbeeldingen bij het gebruik van dergelijke grote hoeveelheden gegevens een voortdurend onderwerp van onderzoek, gezien de benodigde berekeningen. Dit proefschrift richt zich op een specifiek soort algoritmen om dit probleem aan te pakken, namelijk op afbeeldingen gebaseerde representaties.

Entiteiten die worden weergegeven in een virtuele driedimensionale wereld kunnen worden geprojecteerd op een regelmatig tweedimensionaal raster om een afbeelding te vormen. Dit resulteert in een veel compactere, zij het onvolledige, weergave van de originele informatie omdat een afbeelding een stuksgewijze constante discretisatie van de onderliggende data is. Niettemin is het berekenen van de eigenschappen van de 3-dimensionale gegevens via de 2-dimensionale projectie veel efficiënter. Toch is het belangrijk om te begrijpen wanneer en hoe dit type representaties te gebruiken is, omdat de discretisatie van de originele gegevens kan leiden tot artefacten in de berekende resultaten.

Dit werk zal zich concentreren op drie belangrijke aspecten van computergraphics algoritmen waarbij het gebruik van passende op afbeeldingen gebaseerde representaties kan resulteren in verhoogde prestaties namelijk de geheugenvereisten, de efficiëntie van de berekeningen en interactiviteit. Om dit te doen worden de op afbeeldingen gebaseerde oplossingen voor verschillende relevante problemen op het gebied van computergraphics beschreven. Ten eerste door te onderzoeken hoe het gebruik van tweedimensionale tussenrepresentaties geheugenvereisten kan reduceren voor het opslaan van grote statische schaduwinformatie in vergelijking met de beste huidige voxelrepresentaties die tegenwoordig worden gebruikt. Ten tweede zal een hiërarchische op afbeeldingen gebaseerde benadering voor het efficiënt berekenen van diffractiepatronen worden gedemonstreerd die beter kan presteren dan op FFT-gebaseerde oplossingen. Ten slotte zal een efficiënte interactieve optimalisatiemethode voor het bewerken van de dispariteit bij het maken van stereografische afbeeldingen worden beschreven.

Deze algoritmen zullen in detail worden beschreven en geëvalueerd om de lezer inzicht te geven in het gebruik van op afbeeldingen gebaseerde representaties.

PREFACE

LEONARDO SCANDOLO

Delft, March 2019

This work is in many ways the conclusion to four amazing years of work at TU Delft. When I contacted Prof. Eisemann to ask about the possibility of pursuing a PhD within his group, it was only a lucky misunderstanding that led to an interview and later to a position offer. During our initial conversations, I remarked that my main goal was to learn as much as possible about computer graphics in that time. Luckily, he agreed with this sentiment, and we set out to work on a variety of topics from the start.

This dissertation is based on the line of work I was most involved personally, and where we were able to successfully improve upon the state-of-the-art methods. Along the way were many failed, or rather not-quite-so-successful, attempts at solving the same problems that are presented in this dissertation. Understanding why these methods were not as useful was sometimes very enlightening, and I feel it was a very important part of improving as a researcher.

Furthermore, I was lucky enough to participate to a smaller degree in a diversity of other projects, from which I took away many important lessons. Although not part of this dissertation, they are a core part of the knowledge I acquired during my PhD, and hopefully part of that is reflected on the quality of my work as time went on.

1

INTRODUCTION

Highly organized research is guaranteed to produce nothing new.

Frank Herbert, Dune.

THE field of computer graphics plays a large role on a wide variety of industrial and artistic applications. Some popular fields include computer games, animated movies and shorts, and the special effects industry. Other applications include training simulators for a variety of skills, scientific visualization of large datasets, and industrial and architectural design. All of these areas are in constant growth, demanding the creation and visualization of ever-increasing amounts of data and details. Over the years, there has been a steady increase of quality, realism, and detail in the achieved results, as well as increased efficiency in *rendering*, the process through which images are created.

Although hardware advancements contributed to expanding the applications and amount of complexity that can be pursued when rendering, research into optimized data structures and algorithms remains crucial. Indeed, since its inception, the computer-graphics field has struggled with the limitations of hardware capabilities in order to produce large, detailed, physically accurate images. The development of distributed ray tracing[1], radiosity algorithms[2], and path tracing solutions to the rendering equation[3] provided correct, albeit slow (for current standards) methods for recreating the most prominent aspects of realistic lighting. Ever since, much of the research in the area of computer graphics has been devoted to providing developers with the tools and capabilities to represent and render increasingly larger and more detailed virtual scenarios in a more efficient manner.

Nowadays, most artistic content generated using computer graphics attempts to mimic our reality and is therefore 3-dimensional in nature. Nevertheless, working directly in three dimensions is inherently costly in terms of memory and processing requirements. Scaling becomes an important issue, since doubling the scale of a representation in three dimensions results in an eight-fold increase in memory and computational resource requirements. Virtual scenarios represented directly in three dimensions, via



Figure 1.1: Examples of increasing complexity and realism in computer graphics over the years. In reading order: A frame from the short computer animated film *A Computer Animated Hand* by Edwin Catmull and Fred Parke (1972), example of path tracing from the article *The Rendering Equation* by Jim Kajiya (1986), a frame from the computer animated movie *Toy Story* by Pixar Animation Studios (1995), and a frame from the game *Red Dead Redemption 2* by Rockstar Games (2018).

voxels or similar primitives, are difficult to employ in current-day high-performance applications. The state-of-the-art techniques for encoding rich virtual scenes using voxelized representations[4, 5] have memory requirements that are prohibitive for real-time interactive applications. For such high-performance applications, virtual scenes and characters are represented as a surface described by a set of triangles, an inherently 2-dimensional geometric entity, placed in a 3-dimensional world. Furthermore, material and geometrical attributes, such as color, surface detail and reflective properties are usually encoded using images, typically referred to as textures. A simple mapping from triangle vertices to texture coordinates allows rich details to be added to simple geometrical models, further reducing the need to create and store detailed geometrical objects. The success of such representations motivated the creation of highly specialized graphics processing units (GPUs) that can access and manipulate triangle representations and textures efficiently.

Following the same reasoning, this dissertation will explore new methods of reducing memory and processing requirements of computer graphics algorithms by finding an appropriate 2-dimensional (image-based) intermediate representation. To do this, we will tackle specific problems that will provide insight into their applicability and performance. For each problem, we propose new representations that are not only more efficient, but also easier to reason about and implement using current graphics processing software and hardware than their complete 3-dimensional counterparts.

1.1. IMAGE-BASED RESOLUTION STRATEGIES

Images are, by and large, the most common medium used to portray graphical information. Usually, a specific representation is used for images: a regular 2-dimensional grid of values, commonly called *pixels*. This representation is not coincidental, a regular grid is in most cases easier to design and build, and also easier to reason about and subdivide. CRT, LCD and LED-based televisions and monitors are, except for rare occasions, built using this same image representation, physically and/or logically, for this very reason.

In computer graphics, simulated virtual cameras are used to create images meant to be displayed on a physical screen. The parameters that define these cameras are responsible for the section of the virtual scenario that will be represented in the resulting image. The camera view is defined via a truncated¹ cone (or a cuboid, depending on the projection model used), named the *view frustum*. An image representing that view is created by projecting the contents of the view frustum into the camera view plane, and naturally resolving occlusions based on distance.

Due to the fast processing of image content achievable using graphics hardware, the produced images are not only used for final display on the screen but also gave rise to many different applications meant to improve rendering quality. By reasoning directly using this image representation, many effects can be created at a fraction of the computational cost that it would demand were it to be computed based on exploring the entire scene information. Screen-space effects that exploit this concept include ambient occlusion, simple screen-space anti-aliasing techniques, bloom effects, and many more.

Other entities in the virtual scene can also benefit from using image representations. A prime example of this are spot lights and directional lights, where shadow maps[6] are used to determine the part of the scene that is lit. A shadow map is an image that stores the distance to the first visible surfaces from the light. Reasoning about and evaluating shadows using a shadow map is far simpler (albeit not always as exact) than involving the entire geometrical structure of the scene.

The common focus on image-based representations of the techniques presented in this dissertation is not accidental. Many of the optimized resolution strategies for recurrent problems in computer graphics can be broadly grouped based on their fundamental approach. One example of a common approach is to *change the representation basis* of an image or other data, usually expressing information using a Fourier, wavelet or spherical harmonics decomposition. This approach has led to the jpeg[7] image compression algorithms and video compression algorithms[8]. In the context of rendering, some techniques that use this approach are convolution shadow maps [9], prefiltered single scatter effects[10], fast global illumination approximations [11, 12], and many more. Another example of a common strategy is to employ *statistical reconstruction* of information via sparse image sampling. Approaches that use this strategy are variance shadow maps[13], imperfect shadow maps[14], and moment-based algorithms[15, 16]. While many of the previous approaches mostly used image-based solutions as a means to cheaply approximate costly effects, this dissertation explores using image-based solutions that can achieve high-quality rendering results. We will show that image-based solutions are efficient and also accurate options for many different tasks. Specifically, to

¹The reason for not using a full cone is culling objects very close to the virtual camera

demonstrate the versatility of these representations, we will focus on three fundamental operations: storage, access, and modification. Therefore, this dissertation will put special emphasis on memory resources, efficiency, and interactive modification of image-based representations. In order to do this, we will specifically tackle problems where one or more of these properties are essential. Each of the applications will be fully explored in its own chapter, and the next section will outline the problem and resolution strategy employed in each case.

1.2. PRESENTED APPLICATIONS

SHADOW MAP COMPRESSION

Using shadow maps is the most common and efficient way of creating shadows in real-time computer graphics applications. Each texel of a shadow map records the distance to the first visible surface from the point of view of the light. When creating an image from the point of view of the main scene camera, the surface point at each pixel is compared against what is stored in the corresponding shadow map texel. This depth comparison, also called a *depth test*, determines whether a point is lit. If the point seen by the main camera matches what is stored in the shadow map, then the object is lit, and otherwise it is shadowed.

Shadow maps are not a perfect solution to the shadowing problem, as they suffer from aliasing. This problem stems from the fact that shadow maps only store a discrete representation of the scene. When the resolution of the shadow map is insufficient for the main image that is being rendered, the correct shadowing patterns cannot be reproduced. In order to alleviate this problem, many adaptive shadow map creation schemes have been devised, which attempt to match the main image resolution with the shadow map resolution. Popular among these techniques are perspective shadow maps[17], which applies a projective transformation to the shadow map, and cascaded shadow maps[18], which creates several shadow maps for a single light, each corresponding to different depth ranges as seen from the main camera.

Traditionally, shadow maps are generated for each frame in order to capture changes in the scene. Nevertheless, in most applications, most parts of an environment are static, such as the terrain or buildings. Optimally, one could create a shadow map containing only the static objects just once, and then update another shadow map with the dynamic objects, resulting in much less rendering done each frame. Unfortunately, using a static shadow map is, in most cases, infeasible, given the need to use adaptive techniques that depend on the main camera position.

In this thesis, a hierarchical method to compress shadow map information is presented. (Fig. 1.2). This compression allows creating very high resolution shadow maps (upwards of $128k \times 128k$ texels), which bypasses the need to use adaptive techniques, and thus a single compressed shadow map can be precomputed for the static scene objects. This greatly reduces the amount of objects that need to be rendered into shadow maps in each frame.



Figure 1.2: **Left:** A compressed multiresolution hierarchy is used to generate detailed shadows of trees. **Right:** The structure used encodes depth in a hierarchical manner, using smaller (deeper) nodes as needed. Nodes in deeper tree levels are shown in lighter colors.

EFFICIENT DIFFRACTION PATTERN SYNTHESIS

Diffraction patterns are a wave-optical phenomenon that occurs when a light wavefront interacts with an object blocking its path. The perturbed wavefront interferes with itself, creating interesting patterns of light when it reflects off surfaces. Furthermore, the wavefront behaves differently depending on its wavelength, resulting in colorful patterns.

A common occurrence of this phenomenon presents itself when light traverses a narrow aperture that is perpendicular to the wavefront direction and parallel to a sensor plane. This is the case for many photographic and video cameras in use today, which focus light using a series of lenses through a small aperture in order to stimulate a sensor. When bright light shines through this aperture, the refraction patterns become visible in the form of starburst patterns (see Fig. 1.3), whose exact shape depends on the aperture shape and camera setup.

Although these diffraction patterns are an artifact of the camera lens and sensor system, artists find the patterns visually pleasing, and thus attempt to incorporate them to virtual scenes. The guiding principle behind these patterns was researched by Christiaan Huygens in the 17th century, and summarized in the Huygens-Fresnel formula. This formula was then approximated and simplified for near-field diffraction, when the sensor is close to the aperture plane in relation to the considered wavelength, and for far-field diffraction, when the aperture plane is far from the sensor plane. Interestingly, the simplified formula for far-field patterns matches the integral for expressing a function (the aperture shape) in the frequency domain, the 2D Fourier Transform. Therefore, far-field diffraction patterns in computer graphics have classically been computed using the Discrete Fast Fourier Transform (FFT).

Chapter 3 presents an alternative method for computing these diffraction patterns, which fully leverages the regular structure of an aperture shape represented as an image. By decomposing an aperture pattern into multi-scale regular quadrangular components (quads), a single closed-form formula can be used for all components in order to efficiently compute far-field diffraction patterns. Leveraging the regular structure of

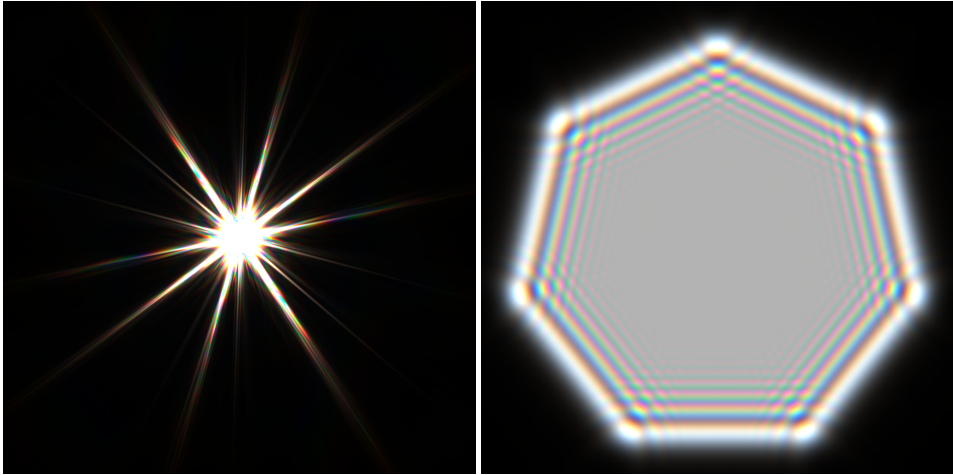


Figure 1.3: Examples of a far-field diffraction (left) and near-field diffraction patterns (right) from a heptagonal aperture. In the case of the far-field diffraction, noise has been added to the aperture shape to obtain a more colorful pattern.

a 2D grid and the additive nature of the far-field diffraction formula, this thesis outlines several optimization strategies that can further reduce the time required to create these patterns

Furthermore, the approach is extended to near-field diffraction patterns. Contrary to FFT-based methods, the method described in this work is able to seamlessly compute diffraction patterns efficiently for any sensor distance.

Finally, the chapter also outlines efficient methods for computing colorful patterns in the visible light spectrum based on the pattern of a single wavelength.

SCREEN SPACE DISPARITY EDITING

Stereographic content gives artists a medium that can more faithfully represent virtual 3-dimensional scenes. Stereo display devices require two images to be created each frame which will be displayed individually to each eye of the viewer. These images are created with virtual cameras placed in the scene with a horizontal distance similar to where the eyes of an observer would be placed. By altering the projection of these cameras, an artist can emulate different focal points, emulating in this way the rotation of the viewers' eyes, also called *vergence*. This allows the human brain to naturally interpret the depth of each object in the scene by accounting for the horizontal distance, called *disparity*, of the object in the left and right image.

Another depth cue used by the human visual system to identify distances is to change the curvature of the lens in the retina using specialized muscles. This process, known as *accommodation*, allows the eyes to adjust their focal distance, and is unconsciously used to estimate distance. Normally, vergence and accommodation cues agree and provide the visual system with a robust estimate of depth. In the case of a stereoscopic image, the screen distance, on which the images are displayed, remains constant, but the ver-

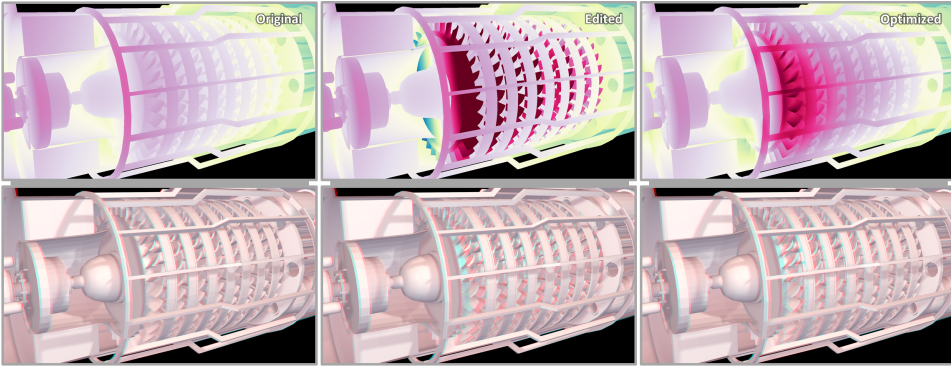


Figure 1.4: Example usage of our disparity edition tool. Left column shows the original version of the scene, with color-coded disparity and anaglyph stereo result. Middle column depicts the result of enhancing the disparity of the blades without harmonizing the result with the rest of the scene; looking at the image will generally cause discomfort in viewers. Right column shows the result of our tool, which maintains the disparity enhancement on the blades but modifies the rest of the scene to avoid discomfort. 🏠

gence cues may indicate different depths. If both cues disagree strongly, the viewer may be unable to properly fuse the images, and may experience discomfort or headaches. The range of difference that vergence and disparity cues can adopt while still remaining comfortable to the viewer is known as the stereoscopic *comfort zone*[19]. Content creators must ensure that the scene and camera parameters used to create stereoscopic images result in synthetic vergence cues that fall within this zone.

To ensure comfort bounds, stereoscopic content requires special processing, but artistic edition of such content is not straightforward. Once the camera parameters are established, each pixel of the resulting images has a defined disparity, which depends on its depth with respect to the cameras. Content creators may choose, for artistic purposes, to modify the disparity of individual objects in the scene, to make them appear closer or farther away from the camera, or make them rounder or flatter. Unfortunately, modifying the disparity of a single object without taking into account the rest of the scene can have negative consequences; disagreements between disparity and other visual cues, such as occlusion, texture or size, can occur.

In Chapter 4, we propose an image representation of disparity values, a *disparity map*, that will guide the editing process. A user can choose to modify disparity properties of individual objects, which will add constraints to the values of this disparity map. Certain guarantees on the disparity values are also taken into account, again in the form of constraints, in order to avoid conflicting depth cues. An optimization process is then performed in order to obtain a disparity map that follows artist constraints and is free from depth-cue conflicts. This optimized map will then guide the rendering process in order to obtain the final stereo pair (see Fig. 1.4).

1.3. PERSONAL CONTRIBUTIONS

Following this short introduction, this dissertation contains three main chapters which are based on scientific articles published as part of my studies.

Chapter 2 is based on *Compressed Multiresolution Hierarchies for High Quality Pre-computed Shadows*, Computer Graphics Forum, Vol. 35, No.7 (2016), and *Merged Multiresolution Hierarchies for Shadow Map Compression*, Computer Graphics Forum, Vol 35, No. 7 (2016). The solution was conceived in collaboration with Dr. Pablo Bauszat and Prof. Elmar Eisemann, and the articles were written as a collaborative effort as well.

Chapter 3 is based on *Quad-Based Fourier Transform for Efficient Diffraction Synthesis*, Computer Graphics Forum, Vol 37, No. 4 (2018). For this work, Dr Sungkil Lee and Prof. Elmar Eisemann proposed the use of an aperture decomposition approach for computing the diffraction patterns. Together with Dr. Sungkil Lee, we extended the hierarchical quad-based method for far-field and near-field diffraction, and proposed several optimization strategies described in the article.

Chapter 4 is taken almost verbatim from *Gradient-Guided Local Disparity Editing*, Computer Graphics Forum, Vol. 38, No. 1 (2019). For this work, Prof. Elmar Eisemann proposed using a resolution strategy based on constraints on the image disparity and its gradient, as well as the reprojection principle. Based on this idea and early work done by Dr. Leila Schemali, along with Dr. Pablo Bauszat we created the editing tools, disparity optimization procedure, and an improved reprojection approach described in the article.

In all cases, aside from the scientific contributions described, I was also the main programmer for the solutions, which led to many small optimizations contributed to the projects, too numerous to detail here.

2

COMPRESSED MULTIREOLUTION HIERARCHIES FOR HIGH QUALITY PRECOMPUTED SHADOWS

L. Scandolo, P. Bauszat, E. Eisemann

The quality of shadow mapping is traditionally limited by texture resolution. We present a novel lossless compression scheme for high-resolution shadow maps based on precomputed multiresolution hierarchies. Traditional multiresolution trees can compactly represent homogeneous regions of shadow maps at coarser levels, but require many nodes for fine details. By conservatively adapting the depth map, we can significantly reduce the tree complexity. Our proposed method offers high compression rates, avoids quantization errors, exploits coherency along all data dimensions, and is well-suited for GPU architectures. Our approach can be applied for coherent shadow maps as well, enabling several applications, including high-quality soft shadows and dynamic lights moving on fixed trajectories. Additionally we present an extension that leverages redundancy to further reduce the memory requirements by merging functionally equivalent subtrees.

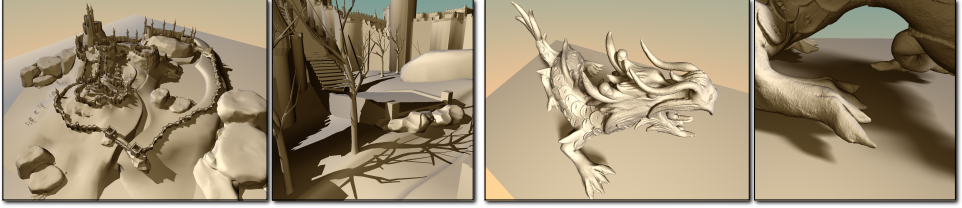


Figure 2.1: **Left:** High-quality shadows in a static large-scale environment rendered in 1 millisecond using a 32-bit shadow map with a resolution of $1.048.576 \times 1.048.576$ pixels. The shadow map is compressed from four terabytes down to 160.6 MB (26124:1 ratio) without loss of precision. **Right:** Precomputed soft-shadows from a high-detail model using 2,048 coherent shadow maps, each with a resolution of 2.048×2.048 pixels, rendered with 32 samples in 14 milliseconds and stored in 145 MB (227:1 ratio).

2.1. INTRODUCTION

HIGH-QUALITY shadows are an important challenge in many real-time rendering applications in computer graphics. Shadow mapping [20] is today's standard for real-time shadows, however, its quality is often limited by texture resolution. Ideally, when projecting a shadow map into the view of the main scene camera, the main camera resolution should match the projected shadow map texel resolution. Unfortunately, to achieve this in large scale scenes this would require resolutions and memory requirements that are not supported in commodity GPUs. As an example, in the castle scene presented in Fig. 2.1, a shadow map with a resolution lower than $128k^2$ would display blocky artifacts, which means that a minimum of 64GB of video memory is needed to use a single standard shadow map. Adaptive approaches such as Adaptive Shadow Maps [21] or Cascaded Shadow Maps [18, 22] are popular real-time solutions, but come at the cost of reduced run-time performance since they require the creation of several shadow maps for each frame. Virtual scenes often consist of large static parts (e.g., terrains or buildings), and therefore pre-computing detailed shadow information for them has become a common practice (e.g., *light maps*). In this case, shadows of the dynamic scene parts need to be computed with a different algorithm.

Recent advances have shown that precomputed compressed high-resolution shadows information can be a competitive alternative [23–25]. Such techniques fully handle shadows cast by static objects on both static and dynamic receivers. Dynamic shadow casters are handled using standard shadow mapping techniques at run-time, with the added benefit of not having to render the static parts of the scene. Unfortunately, conventional image compression techniques are not suitable for compressing shadow maps. Conventional lossy compression techniques result in light and shadow leaks, since they are not designed to satisfy strict per-pixel bounds. Globally diminishing allowable bounds or using lossless compression techniques does not result in satisfactory compression rates. Furthermore, many techniques rely on run-length encoding, which prohibits fast, random-access queries necessary for computing real-time shadows. Fast, random-access compression of color textures has been explored in the context of GPU architectures ([26–28]), but these algorithms rely on quantizing data or lead to low compression rates (up to 5%), which is insufficient for higher resolutions. Consequently, custom schemes for shadow-map compression have recently been proposed. These approaches typically

exploit the fact that, in static scenes, any depth value between the depth of the first and second surface underneath a pixel leads to a conservative occlusion test. However, previous approaches do not fully exploit the data coherency, or rely on depth quantization.

We introduce a novel compression scheme for creating very high-resolution shadow maps based on multiresolution hierarchies (MH). We propose a sparsification process, which exploits the concept of dual shadow mapping (a shadow map for the front faces and one for the back faces) to create an extremely sparse, but conservative, multiresolution decomposition of the original (front) shadow map. This decomposition is efficiently encoded in a compressed regular tree for fast random access during run-time. We show that our approach achieves higher compression rates than all previous approaches, can be queried with real-time performance, and can be efficiently built using GPU architectures. Our approach is the first to exploit coherency along all data dimensions, it does not rely on quantization (maintains full 32-bit precision) and supports shadow maps from arbitrary light sources. Another benefit is that it naturally incorporates all information required for hierarchical filtering operations since it offers a multiresolution representation, which means that each level of the resulting hierarchy by itself is a complete shadow map. We also show that our approach can be directly extended from single shadow maps (2D encoding using quadrees) to a coherent set of shadow maps (3D encoding using octrees). Fig. 2.1 shows an example of shadows created from both precomputed shadow maps and coherent shadow map sets encoded using our method. Finally, we present an algorithm to find and merge functionally equivalent subtrees within the resulting MH, which can reduce memory requirements of the structure by up to 40% while retaining the same run-time performance. This algorithm is based on the observation that the resulting hierarchical representation frequently exhibits similar hierarchical patterns around small structures and edges due to the regular nature of the underlying data. Our approach is the first to enable efficient compression of and rendering with high-quality shadow map sets to produce soft shadows, and moving light sources with known trajectories (e.g., sun lighting).

2.2. RELATED WORK

We will briefly discuss previous approaches for precomputed compressed shadows and compression of tree hierarchies. For a comprehensive overview of real-time shadow generation, we refer to the surveys of Eisemann et al. [29] and Woo et al. [30].

Texture compression Image compression techniques are abundant; two of the most widespread compression standards are JPEG[7] and PNG [31]. Traditional techniques often do not provide efficient random-access to the data, i.e., they require to decode larger parts or the whole image at once, which prohibits their use for real-time applications that require fast random access. In the context of real-time rendering, several GPU-supported compression formats that allow random-access queries exist, such as S3TC[32], ETC[33], ASTC[34] and others. While these methods work well for texture and normal maps, they are lossy and encoding an image with these techniques results in a globally bounded per-textel error. In the context of shadow map compression, it is necessary to ensure that the result of the depth test is always correct in order to avoid arti-

facts and thus, lossy techniques are not suitable. Although lossless compression methods based on decomposition and block-based matching exist [HC07], they achieve very modest results compared to specialized shadow map compression methods that can exploit per-texel error margins.

Compressing with line segments Based on the assumption that shadows are not cast inside of objects, Woo et al. [35] proposed midpoint shadow mapping as a solution to self-shadowing artifacts. Midpoint shadow mapping computes a new shadow map, which represents the intermediate surface lying between the two surfaces closest to the light source. Since all depth values stay between the front facing geometry (the surfaces that represent the original shadow map) and the back facing geometry (the first exit point out of the object), the resulting occlusion test is conservative. An extension of this approach is dual shadow mapping [36], where the shadow maps are kept separate and shadow biasing can be performed adaptively. Based on this concept, Arvo et al. [23] introduced Compressed Shadow Maps (CSM) and showed how to compress a shadow map by representing each scan-line with a set of line segments approximating the midpoint surface. This approach shows that shadow map compression can be understood as signal compression with a specific spatially-variant bound. Although our approach can be interpreted as a 2D or 3D extension, finding the exact analytic equivalent in higher dimensions is a significantly more complex task.

Ritschel et al. [37] similarly compresses a set of coherent shadow maps by encoding the depth values of each pixel for all images by using a set of lines. However, both approaches do not fully exploit data coherency along all dimensions (e.g., only along the vertical dimension or "through" the image stack) and, therefore, cannot achieve optimal compression rates. Additionally, since these compression schemes are non-hierarchical they do not adapt well to the underlying data and efficient filtering along dimensions other than the compression dimension becomes impractical.

Precomputed Voxelized Shadows Recently, Sintorn et al. [24] proposed to precompute shadow information for a voxelized scene representation in projective light-space, which is efficiently encoded in a 2-bit Sparse Voxel Octree [38]. The octree is further compressed by subtree merging using a Directed Acyclic Graph (DAG) [39]. The initial compression and construction performance was improved and resulted in the current state-of-the-art compression method for precomputed shadows [25]. Unfortunately, the voxelization process leads to depth quantization and, although the information is encoded hierarchically, it is not a multiresolution representation, and fast filtering requires additional memory, almost doubling the size of the tree. Despite their ability to greatly reduce memory requirements for 1-bit or 2-bit values, DAGs are not well suited for compressing more complex data, since equal subtrees become scarce, thus their usage to compress 32-bit shadow maps becomes unfeasible. State-of-the-art approaches in this area [4] require heavy quantizing of data in order to obtain acceptable results, which is incompatible with the lossless constraint of shadow map compression. Furthermore, their usage to encode shadow map sets is difficult since the compression is only efficient in the projected space of the light source.

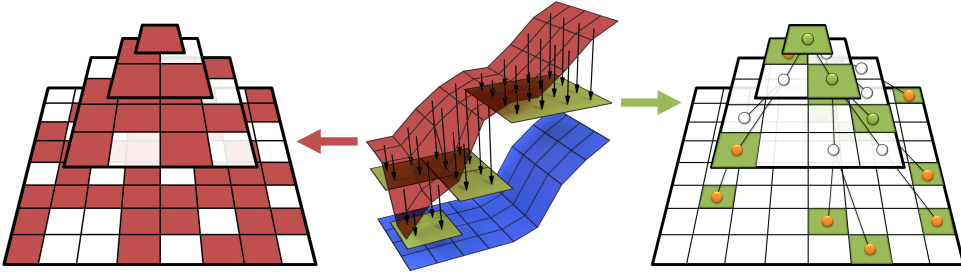


Figure 2.2: **Left:** A multiresolution decomposition of a shadow map requires many coefficients (red) at finer levels in varying regions and is typically not sparse. **Middle:** Using dual shadow mapping, an intermediate surface (green) can be found between the shadow map (red) and the auxiliary second-surface shadow map (blue). Here, the intermediate surface represents a linear, conservative approximation of the shadow map by a set of axis-aligned planes. Choosing these planes to represent common depth values for many pixels results in a more homogeneous occlusion surface. **Right:** A significantly sparser multiresolution decomposition encoding the set of axis-aligned planes. The overlayed quadtree shows the encoding of coefficients. Inner nodes are represented by green circles, while leaf nodes are marked as yellow. Note the empty inner nodes (white circles) which are required to encode the topology information, but do not store any depth values.

Tree Compression A large body of research exists for efficient tree-based encoding of MH (e.g., [40–42]). Unfortunately, our tree must support random access and exhibits certain uncommon characteristics, making it difficult to apply most previous techniques. However, to address the overhead introduced by storing topology information, we utilize the pointer compression technique proposed by Lefebvre et al. [42] and efficiently encode tree pointers using 16-bits. We do not employ any vector quantization [43–45] or other optimizations to the stored depth values themselves. Our results demonstrates that even without such data changes, our approach outperforms previous compression approaches, while maintaining full 32-bit depth precision.

2.3. COMPRESSED MULTIRESOLUTION HIERARCHIES

Multiresolution decompositions of images (e.g., wavelets [46] or quadtree images [47]) split features into components of different scales, typically storing homogeneous parts at coarser levels and details at finer levels. In consequence, finer levels (which contain more coefficients) are usually sparse, and coefficients are small if they are encoded as differentials to previous levels. Lossy compression exploits this characteristic and removes small coefficients assuming their influence to the composed image is not significant. However, for lossless compression all coefficients, independent of their magnitude, have to be considered. This results in decreased sparsity and diminished compression (Fig. 2.2, left).

Our key idea is to compress an alternative representation of the shadow map that is more homogeneous, but still conservative. Our goal is to find new depth representatives for each texel in order to increase the sparsity of the hierarchy, but such that a conservative depth test (i.e. equal depth test result for all visible parts of the scene) remains possible. This is achieved by choosing values inside the boundaries defined by the first entry and exit surface points. To compute these bounds, we employ the concept of *dual*

shadow mapping (Fig. 2.2, middle). As a whole, the procedure can alternatively be interpreted as the compression of an image with a spatially-varying error bound defined by an interval that must be met to maintain lossless compression. For fast random-access during run-time, we encode the sparse decomposition using a compressed quadtree (Fig. 2.2, right).

For non-watertight or one-sided objects, the upper and lower bounds of the depth interval need to be set to the depth value of the entry surface to ensure a conservative depth test. Although this reduces compression capability, our technique still handles these cases correctly and no artifacts are introduced.

In the following, we will first propose two greedy construction methods for finding sparse decompositions from conservative depth bounds. Then, we cover efficient encoding and traversal of the sparse representation using a compressed quadtree. Finally, we discuss shadow map filtering and propose an optimized traversal technique to significantly reduce filtering costs. While this section focuses on single shadow maps only, we will demonstrate in Sec. 2.4 how to extend our approach to a set of coherent shadow maps using a compressed octree.

2.3.1. CONSTRUCTION

Our first task is to define the allowable depth interval for each texel. While the lower depth bound is defined by the original shadow map, the upper bound is determined using the second layer obtained via depth peeling [48].

If the scene contains intersecting watertight objects, we can even further exploit the compression potential by ignoring surfaces inside of other objects. To exemplify this point, one can imagine each shadow map texel corresponding to a ray cast from the light source. For each ray, one can track the encountered surfaces with a counter while advancing in the scene. The counter is incremented for each front-facing surface and decremented when a back-facing surface is encountered. The first intersection corresponds to the minimum bound of the allowable depth interval, which sets the counter to one. After that, when the counter reaches zero, the ray has exited all objects and the corresponding depth is set as the maximum of the depth interval. This procedure can be efficiently carried out for all texels simultaneously via a depth-peeling algorithm or by using a k-buffer [49]. In the case of one-sided surfaces, they can be accounted for as coinciding front and back faces.

After computing per-texel depth intervals, we need to find a per-texel depth value inside such intervals which allows for a sparse decomposition. Interestingly, the task of finding an intermediate surface inside a given envelope is a common problem in mesh simplification, and for the 3D case it is known to be NP-hard [50]. We propose two greedy approaches, which perform a sparse decomposition and tree construction at the same time. The first one is a top-down approach which tries to globally minimize the number of distinct depth values, while the second one operates in a bottom-up manner and inspects only local texels from the next finer level. The resulting hierarchy will then be sparsified and encoded using a quadtree structure.

Top-down construction The top-down construction starts at the coarsest level, which represents the entire image domain, and greedily selects the depth value which covers

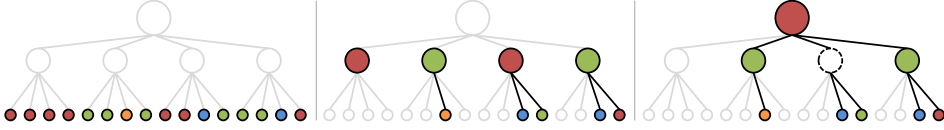


Figure 2.4: Three steps of the bottom up creation algorithm. **Left:** Initially all values are present in the finest level. **Middle:** The most frequent value is pulled up the hierarchy. **Right:** The procedure is repeated for the next level and an empty node is created to preserve the connectivity.

the largest number of depth intervals.

We then mark all texels that can be represented by this value as covered. These covered texels will inherit the depth value from the coarsest level and only the remaining uncovered texels need to store a separate depth value. The approach then proceeds to the next finer level by decomposing the domain into four quadrants. For each quadrant that contains at least one non-covered value, the algorithm is launched recursively. If all texels in a quadrant are covered, or the finest level is reached, the algorithm stops. Consequently, homogeneous areas result in an early termination of the process.

To find the depth value covering most intervals, a direct approach would be to discretize the depth, create a histogram, and find the value with the largest number of conforming intervals. To avoid discretization, we propose an analytic sweep-based algorithm instead. We start by sorting all interval-bound depths in ascending order.

Then, we sweep through the sorted list and keep track of the number of overlapping intervals by incrementing a counter each time an interval minimum is encountered (we enter an interval) and by decrementing it when a maximum is encountered (we exit an interval). The highest detected count during the sweep leads to the depth representative which covers the maximum amount of intervals possible (Fig. 2.3). The pseudo-code for the top-down decomposition is shown in Alg. 1.

The algorithm requires a single sorting in the beginning, which can be performed in $O(n \log n)$

with n being the number of intervals (shadow map texels). Once the initial list is sorted, all subsequent levels only require an $O(n)$ extraction step to retrieve the sorted list of uncovered-texel intervals for the corresponding quadrant. Since the extraction has to be performed for each level, the overall run-time becomes $O(n (\log n)^2)$.

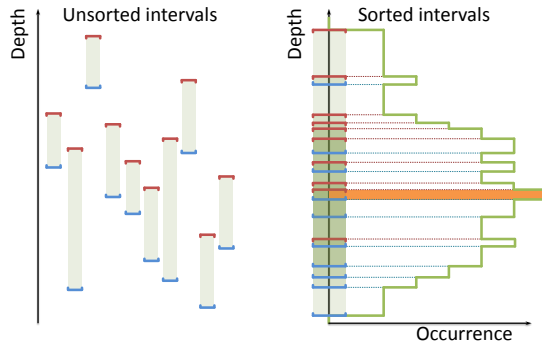


Figure 2.3: To find a depth value which intersects the maximum number of intervals from a given set (left), we propose a sweep-based mode finding scheme. The interval boundaries are sorted in ascending order first (left). We can then find the mode which corresponds to the best depth value by sweeping through the sorted list and keep track of the number of open intervals.

Algorithm 1 Pseudo-code for top-down hierarchy creation

```

function createHierarchy(intervals) :
    sortedIvals  $\leftarrow$  sort(intervals)
    createNode(rootNodeLocation, sortedIvals)
function createNode(nodeLocation, sortedIvals) :
    if isEmpty(sortedIvals) then
        return
    end if
    bestIval  $\leftarrow$  0
    numIvals  $\leftarrow$  0
    bestNum  $\leftarrow$  0
    for each ival  $\in$  sortedIvals do
        if isMin(ival) then
            numIvals++
        else
            numIvals--
        end if
        if numIvals > bestNum then
            bestNum  $\leftarrow$  numIvals
            bestIval  $\leftarrow$  ival
        end if
    end for
    saveNodeToHierarchy(nodeLocation, bestIval)
    sortedIvals  $\leftarrow$  extractUncoveredIvals(sortedIvals, bestIval)
    for each childLocation do
        childIvals  $\leftarrow$  extractChildIvals(childLocation, sortedIvals)
        createNode(childLocation, childIvals)
    end for

```

Bottom-up construction An alternative is a bottom-up construction, which only considers sibling groups of four texels and their shared parent during creation. This approach is better suited for parallel execution and, for all our test scenes, it performs competitively to the top-down construction, while being an order of magnitude faster.

The bottom-up construction is based on the idea of a min-max mipmap creation. Initially, we inspect the lower and upper depth bound of texels at the finest level. When defining the next coarser level, we analyze the four subjacent depth bounds of each texel P using the same sweeping algorithm as for the top-down approach to find a largest depth interval valid for most of these four texels. This interval I is then stored in P and all subjacent texels, whose depth interval contain I are flagged as empty texels. The algorithm then proceeds upwards to the next level. Once we reach the coarsest level, we will populate the map with actual depth values in every non-empty texel by storing the average of the interval bounds.

Since only four intervals are treated at a time, the costly sorting step is avoided and the bottom-up construction requires only a constant number of operations per texel,

Algorithm 2 Pseudo-code for bottom-up hierarchy creation

```

childbounds  $\leftarrow$  [lower,upper]
for level from finestlevel - 1 to coarsestlevel do
  for each pixel in level do
    childrenPixels  $\leftarrow$  getChildren(pixel, level+1)
    depthRepresentative  $\leftarrow$  findBestRepresentative(childrenPixels)
    bounds(pixel)  $\leftarrow$  []
    for each childPixel  $\in$  childrenPixels do
      if satisfiesBounds(depthRepresentative, childPixel) then
        setNonExistant(childPixel)
        bounds(pixel)  $\leftarrow$  bounds(pixel)  $\cap$  getBounds(childPixel)
      end if
    end for
  end for
end for

```

resulting in an $O(n \log n)$ construction time. The algorithm maps better to current GPU architectures, also leading to a practical speedup. The pseudo-code is shown in Alg. 2, and Fig. 2.4 shows an example of the creation of a three level tree. The approach shows similarities to the Mallat-algorithm for wavelet construction [46], but instead of using the average as representative we choose the mode from the set of intervals in order to sparsify the representation.

Tiled construction For the extremely large shadow map resolutions encountered in our approach, it is infeasible to compute the full uncompressed depth intervals in memory before the creation of the MH. Fortunately, the construction of the MH can be performed in tiles. First, the shadow map is divided in tiles of manageable size (in our implementation typically $4k \times 4k$ or $8k \times 8k$). For each tile, we compute its uncompressed bounds via depth peeling, compress it using the top-down or bottom-up algorithm, and store it along with the depth bounds of the root node. After all tiles have been compressed, the stored depth bounds from all root nodes form the bounds of a new shadow map, which is again compressed to create the top level structure of the complete tree. Since only the uncompressed data of a single tile is required in memory at once, this construction procedure is both efficient and maintains a small memory footprint.

2.3.2. COMPRESSED QUADTREES

The previous algorithms lead to a sparse hierarchy of depth values, which subsequently needs to be encoded efficiently while ensuring fast random-access at run-time — which are requirements fulfilled by a quadtree.

Encoding Our quadtree contains three node types: leaves (nodes with no children), inner nodes, and empty nodes. Inner nodes and leaves contain a 32-bit depth value. Empty nodes are only required to encode the quadtree connectivity, but do not contain a value themselves. Unlike other multiresolution decompositions, at this stage we do not

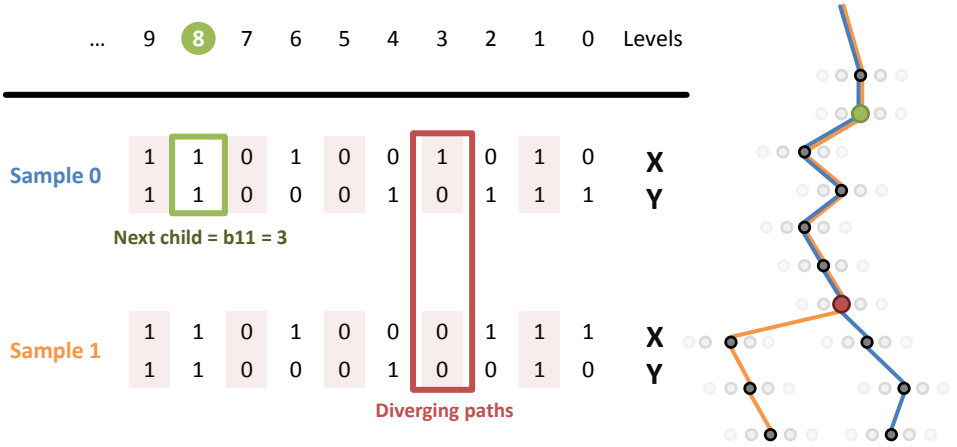


Figure 2.5: Finding the next child index can be done by inspecting the bits from the x and y position of the query point. The lowest common node of multiple query points can be found by finding the last level where the bits are equal.

encode the depth values using parent node differentials. In consequence, only a single value needs to be fetched during tree traversal, which reduces the memory throughput and accelerates lookups.

Inner and empty nodes contain an 8-bit mask indicating the type of each child node (stored in two bits) and a pointer to the first child. We distinguish four cases for the child type: a) non-existent, b) leaf node, c) inner node, d) empty node. As it is common practice, a single pointer is sufficient, as all present child nodes are stored contiguously in memory, and the location of a specific child can be obtained from examining the mask in the parent node.

We employ the pointer encoding scheme proposed by Lefebvre et al. [42] in order to reduce the amount of memory needed to store pointers. Their scheme stores subtrees close together and allows us to encode pointer offsets, whose magnitudes decrease rapidly per level. Using a per-level scaling, we can efficiently encode pointers for all of the test resolutions presented in this article with just 16 bits introducing only negligible (no more than a few KB for Fig. 2.1) padding overhead for alignment. We exhaustively search for the optimal per-level scaling factor as proposed by Lefebvre et al. Finally, we also pad full and empty nodes with a single byte, resulting in 4-byte aligned nodes (8 bytes for full nodes, and 4 bytes for empty nodes and leaves). The padding increases the memory footprint, but eases fetching the values on current GPU architectures, hence decreasing lookup times. Alternatively, 24-bit pointers could be used, however, we decided to keep the bit count compatible with the compressed octree representation which will be introduced in Sec. 2.4. In all our test scenes, no significant difference was introduced by using 16-bit pointers instead of 24-bit pointers for quadtree compression. If memory footprint is overall more critical than traversal performance, the padding can be removed to further improve the compression.

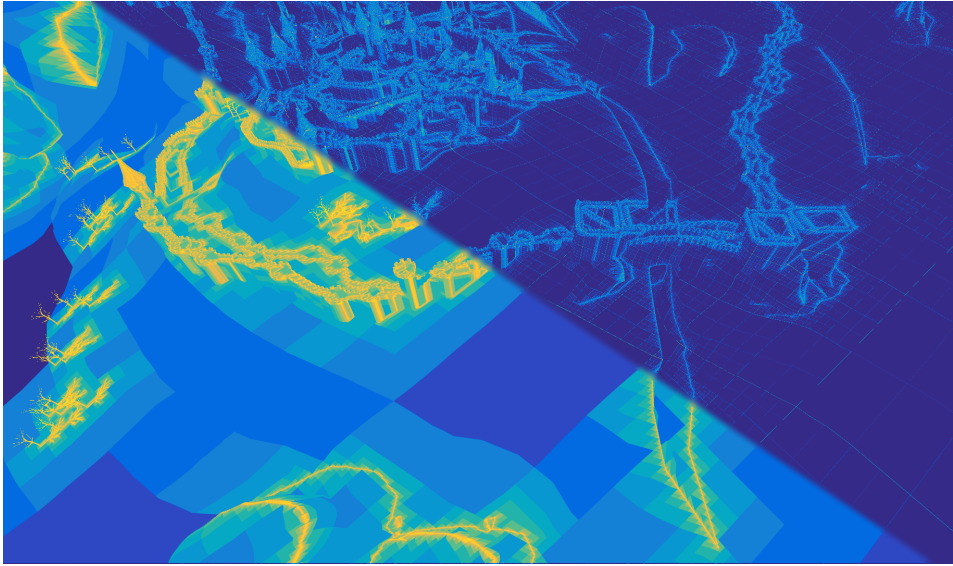


Figure 2.6: Required traversal steps in the CLOSED CITY scene for a 5x5 PCF filtering using a naive implementation (lower-left triangle of the image) and our optimization finding the lowest common node first (upper-right triangle). Bright colors represent numbers close to the maximum (tree height times number of samples), while dark ones mean that only few levels are traversed.

Traversal Traversal of our compressed-quadtree encoding follows the same procedure as standard quadtree traversal, but performs lazy fetching of the depth values to account for the presence of empty nodes. The traversal path through the tree is defined by the position of the query point. By keeping track of the current level, we can directly compute the index of the next child using a few bit-operations (see Fig. 2.5). We start traversal at the root node (which is always a full node) and initialize a pointer which holds the index of the last node containing a depth value. After reading the children mask and pointer, we compute the index of the next child and query the mask to validate the child's existence. We compute the offset of the next child node from the mask and 16-bit child pointer to recursively continue the traversal. When we traverse a full node the pointer to the last position of a depth value is always updated. If a child node does not exist or a leaf node is reached, the depth value is fetched from the last-stored position and the recursion is terminated. We do not query the depth value earlier, as this would result in unnecessary texture fetches, since we do not store differentials, but absolute values.

Although hierarchical traversal has typically a run-time depending on the tree height, the sparsity of our tree often leads to a termination after only a few levels.

2.3.3. FILTERING

Efficient filtering is an important aspect of shadow mapping. Percentage-closer filtering (PCF)[51] is a popular technique which performs averaging of several depth-test results in a fixed-size kernel (usually an $r \times r$ box). A naive implementation of PCF using our

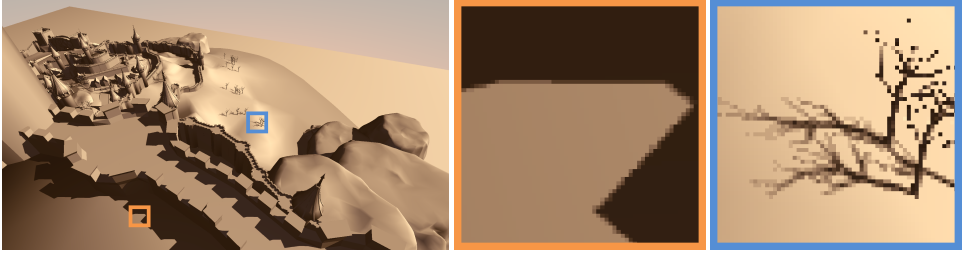


Figure 2.7: **Left:** An example of hierarchical PCF with a 3×3 kernel in the CLOSED CITY scene. Anti-aliased shadows are present at all distances. **Middle:** An inset showing anti-aliased shadows closer to the camera. **Right:** Another inset showing anti-aliased shadows cast by a complex occluder in the distance.

approach would perform a full tree traversal for each kernel sample. Since our quadtree encodes a multiresolution prediction, we can perform analytic filtering when all samples correspond to the same node. While this is not always the case, most samples share at least a common path from the root node until a certain level. This level can be directly computed from the minimum and maximum query points of the filter kernel and a few bit-operations (see Fig. 2.5). We propose to traverse all kernel samples together through the first few levels until we find the lowest common node where paths diverge. After that, each sample proceeds individually. This easy-to-implement optimization can lead to drastic improvements in PCF lookup time. A visualization of the amount of traversal steps for the naive implementation and our optimization for a 5×5 PCF kernel size is shown in Fig. 2.6. Another possible optimization is to keep a cache of the last queried sample and reuse it if the next sample shares the same path through the tree down to the retrieved value.

Multiresolution anti-aliasing such as *hierarchical PCF* computes the shadow map footprint of a image pixel and looks up the depth value for the corresponding resolution level. Since each of our tree levels encodes a full shadow map of the corresponding resolution, hierarchical filtering is natively supported. When performing PCF filtering, the appropriate sampling level of the hierarchy can be chosen to maintain a 1 to 1 correspondence between screen pixels and shadow map texels. This ensures that smooth shadows are present at any view distance regardless of the projected area of a pixel in the shadow map. Furthermore, tri-linear filtering can be performed by choosing two consecutive sample levels and interpolating their values in order to create smooth transitions during motion.

2.4. SHADOW MAP STACKS

We can extend our concept of compressed multiresolution hierarchies directly to a set of coherent shadow maps. By stacking shadow maps in a 3D image cube, we can compute a sparse decomposition in the same manner as for a single shadow map and encode it using an octree. This approach is useful for rendering of soft shadows from area lights (Fig. 2.9, left) or varying light positions (Fig. 2.9, right). For soft shadows, we sample multiple light positions on an area light using a Hilbert-curve sampler as also used by Ritschel et al. [37]. The light positions can be jittered in order to avoid banding for

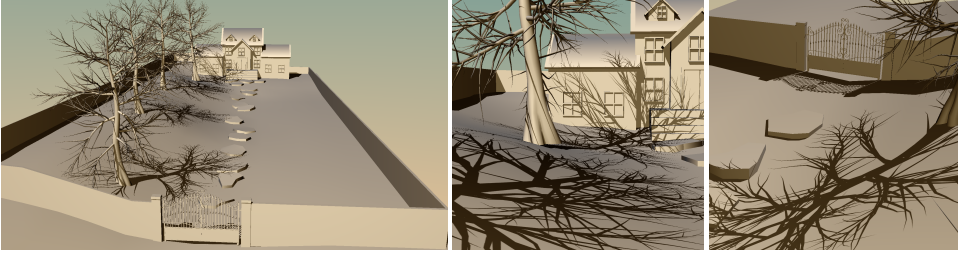


Figure 2.8: **Left:** An overview of the VILLA scene with an unfiltered $32K^2$ compressed shadow map. **Middle:** Close-up of filtered shadows rendered in 2 ms using a 3×3 non-hierarchical PCF kernel. **Right:** Another view-point with a 5×5 non-hierarchical PCF filtering kernel rendered in 5 ms.

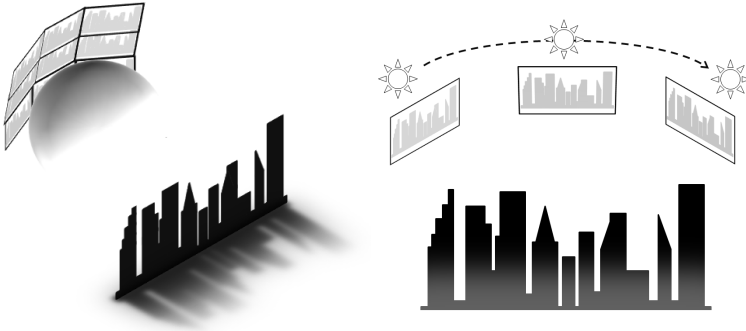


Figure 2.9: **Left:** Computing soft-shadows from an area light requires sampling multiple points on the light source. The set of shadow maps corresponding to these points can be stacked in a 3D image cube for efficient compression. **Right:** Motion of dynamic light source, which is known in advance, can be precomputed by discretely sampling the trajectory, e.g., for simulating high-quality shadows from sun lights.

smaller sampling rates. For moving light sources, the motion has to be known and the images are simply stacked in the order of discrete sample points along the trajectory. Our hierarchical structure is able to exploit coherency along all 3 dimensions by encoding homogeneous cubic regions in coarser levels of the hierarchy.

The construction and traversal techniques of the previous section can be directly applied for octrees as well. The only major difference, however, is that we now have to consider potentially eight children instead of four, which leads to 16-bit child masks. This conveniently removes the need for padding and makes the octree nodes perfectly aligned to 4-byte boundaries by default.

In the case of light trajectories, it is often only necessary to store a small number of different light positions. In this case, creating an equally-sized cube would restrict the resolution to match the number of images. Our approach allows for a convenient encoding of non-cubic image stacks by generating placeholder nodes with a depth boundary of $[0,1]$. Having the largest possible interval width, these nodes will be merged up the hierarchy during compression and introduce minimal overhead to maintain the octree connectivity.

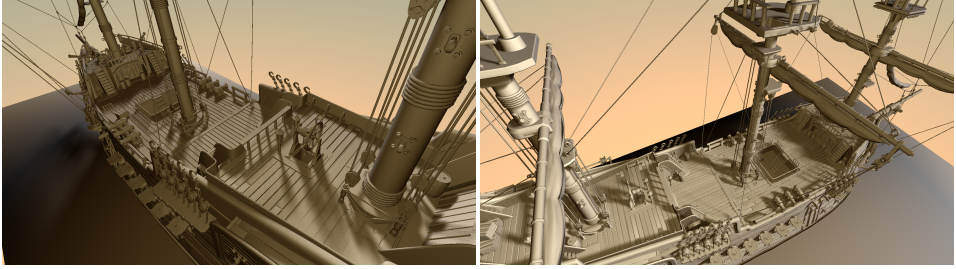


Figure 2.10: Realistic soft shadows in the SHIP scene generated with an octree from 512 depth maps of $1K^2$ resolution. Overall, the compressed octree is stored in 57 MB. **Left:** A closeup using 32 shadow-map samples per pixel rendered in 14 ms. **Right:** Another viewpoint using 64 samples rendered in 30 ms.

2.5. MULTIREOLUTION HIERARCHIES EVALUATION

In this section, we demonstrate the compression capabilities of our method for five test scenes and evaluate its memory requirements, construction times and run-time evaluation times. The CLOSED CITY scene (613K triangles) represents a typical open-world game setting with both large scale and detailed features. The CITYSCAPE scene (11K triangles) is an example of an architectural design model, while the VILLA scene (89K triangles) as well as the SHIP scene (810K triangles) are examples of scenes containing many fine scale details. The DRAGON scene (7.2M triangles) consists of a scanned model with a very high polygon count. Fig. 2.15 and Fig. 2.1 showcases the test scenes.

We implemented larger parts of the construction algorithm on the GPU using NVidia CUDA 7.5. The rendering is done using OpenGL 4.3 and deferred shading, and our measurements are reported for the evaluation of the shadows. All experiments were at a resolution of 1920×1080 on Windows 7 using a PC with Intel i7-5820K CPU with 16GB of system memory, and an NVidia Titan X GPU. We re-implemented the algorithm of Arvo et al. [23] for the comparison to scanline compression. For the comparison to DAG-based compression of voxelized shadows [24], we used the implementation kindly provided by the authors which includes all the improvements from Kampe et al. [25].

Quadtree compression Table 2.1 presents compression results for single shadow maps using multiresolution quadtree compression. We report memory footprints for the quadtree using the 1-byte padding for inner nodes and a full 32-bit depth precision. In all cases, our algorithm outperforms the previous approaches and is able to compress even large resolutions in the order of hundreds of thousands down to a few hundred megabytes. All results used the bottom-up construction. Note that, in contrast to the previous approaches, our method implicitly encodes a full multiresolution representation of the shadow information.

Table 2.2 showcases detailed construction times and total node quantity for several test scenes, as well as the construction time for the voxelized shadows approach from Kampe et al. [25]. While the preprocessing time is not interactive for larger resolutions, we report numbers in the same order of magnitude as the highly-optimized implementation from Kampe et al. It can be seen that most of the time is spent in the depth peeling

	Method	1K ²	2K ²	4K ²	16K ²	64K ²	256K ²	512K ²	1M ²
CLOSED CITY	Uncompressed	4 MB	16 MB	64 MB	1 GB	16 GB	256 GB	1 TB	4 TB
	Arvo et al.	0.092 MB	0.23 MB	0.56 MB	2.83 MB	12.85 MB	54.09 MB	109.8 MB	221.9 MB
	Kampe et al.	-	-	0.62 MB	3.40 MB	14.89 MB	60.46 MB	-	-
	MH	0.067 MB	0.17 MB	0.41 MB	2.10 MB	9.26 MB	38.43 MB	79.63 MB	160.5 MB
	MH (ratio)	1.68%	1.06%	0.64%	0.21%	0.056%	0.015%	0.0078%	0.0039%
CITYSCAPE	Uncompressed	4 MB	16 MB	64 MB	1 GB	16 GB	256 GB	1 TB	4 TB
	Arvo et al.	0.15 MB	0.36 MB	0.78 MB	3.42 MB	14.03 MB	56.61 MB	113.5 MB	-
	Kampe et al.	-	-	0.94 MB	3.94 MB	16.38 MB	63.34 MB	-	-
	MH	0.11 MB	0.26 MB	0.59 MB	2.70 MB	11.41 MB	46.73 MB	94.88 MB	190.4 MB
	MH (ratio)	2.75%	1.625%	0.92%	0.26%	0.069%	0.0178%	0.0090%	0.0045%
VILLA	Uncompressed	4 MB	16 MB	64 MB	1 GB	16 GB	256 GB	1 TB	4 TB
	Arvo et al.	0.14 MB	0.40 MB	1.10 MB	5.89 MB	25.25 MB	103.84 MB	-	-
	Kampe et al.	-	-	1.78 MB	9.26 MB	39.70 MB	166.47 MB	-	-
	MH	0.11 MB	0.35 MB	0.86 MB	5.01 MB	23.61 MB	101.52 MB	205.5 MB	414.6 MB
	MH (ratio)	2.75%	2.18%	1.34%	0.48%	0.14%	0.03%	0.02%	0.009%
SHIP	Uncompressed	4 MB	16 MB	64 MB	1 GB	16 GB	256 GB	1 TB	4 TB
	Arvo et al.	0.21 MB	0.60 MB	1.44 MB	6.73 MB	28.23 MB	115.34 MB	233.2 MB	-
	Kampe et al.	-	-	2.01 MB	8.66 MB	35.57 MB	153.67 MB	-	-
	MH	0.15 MB	0.43 MB	1.05 MB	5.26 MB	22.71 MB	94.18 MB	191.5 MB	392.1 MB
	MH (ratio)	3.75%	2.68%	1.64%	0.51%	0.13%	0.036%	0.018%	0.0093%

Table 2.1: Compression results for our 2D MH approach comparing to the scanline compression of [23] and the voxelized shadows approach of [25]. Our approach outperforms competing compression approaches consistently while retaining full depth precision.

in order to obtain the initial depth bounds. The compression itself is mostly dominated by the bottom-up construction that creates the sparse decomposition, and to a lesser extent by the encoding of the quadtree. Finding the optimal per-level scale for 16-bit pointer compression only takes up a small fraction of the overall construction time.

In Table 2.3 we report timings for single lookup performance and different PCF kernel sizes. We compare our optimized PCF implementation against a naive one, standard shadow mapping (for supported resolutions), and the methods from Arvo et al. [23] and Kampe et al. [25] for the VILLA scene. Since it is naturally provided, our implementations perform *hierarchical* PCF. Shared traversal is significantly faster for PCF filtering than a naive implementation (up to 2 ms for a 3x3 kernel, and 4.7 ms for 5x5). The method from Arvo et al. performs well for small PCF kernels at low resolutions, but does not scale well. The voxelized shadow approach is highly optimized for 9x9x9 and 17x17x17 cubic kernels, and achieves almost the same look-up times compared to single lookups. In their case, the filtering is done at the lowest tree level, and thus will still potentially result in aliasing when the footprint of a pixel is bigger than the kernel size.

In Fig. 2.8 and Fig. 2.7, we present visual results for unfiltered, non-hierarchical, and hierarchical PCF filtering using our method. It can be seen that even high-frequency shadows from small features can be faithfully rendered. Additionally, the insets in Fig. 2.7 show that anti-aliased shadows at any view distance can be achieved by sampling the appropriate level.

As a practical optimization, Sintorn et al. [24] and Kampe et al. [25] store the top 6 levels of the hierarchy in a simple dense grid for large resolutions. This requires a constant 8 MB of memory, which is negligible at higher resolutions. The numbers we report for their approach include this optimization. In our case, this would allow us to remove

	Resolution	Total nodes	Rendering	MH creation	Quadtree creation	Serialization	Total time	Kampe et al. time
CLOSED CITY	1K ²	14944	0.019	0.001	0.003	0.007	0.089	-
	4K ²	90775	0.067	0.003	0.004	0.017	0.242	0.098
	64K ²	2037131	3.253	0.555	0.584	0.260	5.301	4.878
	256K ²	8477953	39.53	9.052	3.653	1.001	55.18	65.23
CITYSCAPE	1K ²	25859	0.012	0.001	0.003	0.009	0.089	-
	4K ²	130974	0.044	0.004	0.009	0.021	0.221	0.061
	64K ²	2508119	1.755	0.551	0.612	0.299	3.888	4.089
	256K ²	10215660	22.45	8.975	4.287	0.984	38.85	51.58

Table 2.2: A detailed breakdown of construction timings (in seconds) and tree characteristics for the multiresolution quadtree compression for the CLOSED CITY and CITYSCAPE scene. Rendering times dominate construction times, while creation of the sparse decomposition and quadtree encoding constitutes around one third of the total. We also provide a comparison to the method from Kampe et al. [25].

	Method	4K ²	16K ²	64K ²	256K ²
Single	Shadow mapping	0.25	0.36	-	-
	MH	0.495	0.52	0.54	0.71
	Arvo et al.	0.39	0.51	1.04	2.6
	Kampe et al.	0.61	0.61	0.68	0.72
3×3	Shadow mapping	0.34	0.65	-	-
	MH PCF Naive	3.35	3.7	3.99	4.11
	MH PCF Optimized	1.61	1.72	1.89	2.04
	Arvo et al.	0.85	1.25	4.18	9.7
5×5	Shadow mapping	0.62	1.46	-	-
	MH PCF Naive	7.7	8.49	8.9	9.32
	MH PCF Optimized	3.45	3.95	4.4	4.72
	Arvo et al.	1.4	2.25	5.6	15.9
9×9×9	Kampe et al.	0.78	0.84	0.93	0.96

Table 2.3: Traversal time in ms for a single scene (VILLA) for our approach and comparing to standard shadow mapping and the approaches from Arvo et al [23] and Kampe et al. [25]. The latter is highly optimized for a cubic 9×9×9 kernel size and for a fair comparison, we only report these numbers.

the upper 11 levels of the quadtree and halve the number of traversed levels on average. If evaluation time is more critical than compression, this could potentially lead to faster lookups.

Octree compression We evaluate our 3D compression for high-quality soft shadows and light motion, and compare it against naively compressing each image separately with our 2D scheme. Table 2.4 reports memory sizes for our image stack compression algorithm for soft-shadows. In the table, we show the resulting memory footprint of compressing a set of shadow maps from an area light separately using our quadtree structure and compressing them with our octree approach. It can be seen that our 3D compression provides an additional gain and is able to reduce the compression rate down to 57.9% at best compared to 2D compression.

A visual impression of high-quality soft-shadows in the SHIP scene is given in Fig. 2.10.

Method		512 ³	1K ³	2K ³	4K ³
DRAGON	Uncompressed	512 MB	4 GB	32 GB	256 GB
	2D MH	11.5 MB	48.4 MB	205.8 MB	863.6 MB
	3D MH	6.7 MB	27.8 MB	145 MB	689.2 MB
	3D/2D ratio	57.9%	57.3%	70.4%	78.8%

Table 2.4: Memory footprint of 3D multiresolution octree-based compression for a set of N images with different resolutions. As a comparison we report the memory by naively using our 2D quadtree compression for each image individually.

Method		256 \times 2K ²	256 \times 4K ²	256 \times 8K ²
CITYSCAPE	Uncompressed	4 GB	16 GB	64 GB
	Compressed size	45.85 MB	136.08 MB	456.32 MB
	Construction time	12.3 s	36.2 s	135.9 s

Table 2.5: Memory footprint and construction times of 3D multiresolution octree-based compression for a non-cubic data set of 256 images taken a fixed-trajectory moving light source.

Please note that the shadow penumbrae generated in this way is geometrically correct and appears more realistic as opposed to PCF filtering. The lookup time for 32 random samples per pixel out of 512 depth maps is 14 ms, whereas evaluating 64 samples takes 30 ms.

Finally, we evaluate our 3D compression scheme for non-cubic image stacks for fixed-trajectory light sources in Table 2.5. We show different viewpoints for the CITYSCAPE scene in Fig. 2.11. Since the construction of the octree is based on cubic tiles, which need to be kept small to fit in GPU memory, the viewport size for rendering is restricted, leading to a large amount of render calls. Therefore, rendering makes up most of the octree creation time.

2.6. MERGED MULTIREOLUTION HIERARCHIES

The MH presented in Sec. 2.3 is able to sparsely represent a dense shadow map (or shadow cube) by exploiting local similarities and using a hierarchical compression scheme. Nevertheless, the proposed data structure still possesses redundant information, which cannot be captured by our proposed method. Namely, the tree representation of a MH may contain redundant equivalent subtrees. In this section, we will explore how to find

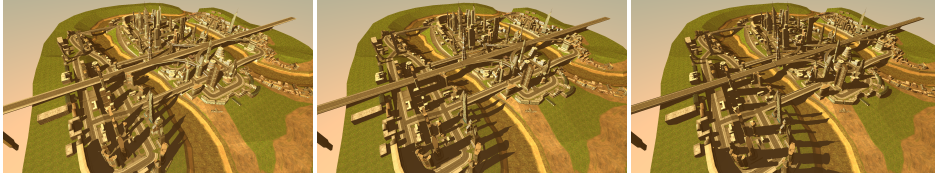


Figure 2.11: The CITYSCAPE scene shows shadows from different views taken from a compressed shadow map stack of 256 4K² images taken on a trajectory above the city. The octree has a compressed size of 136.08 MB (uncompressed 16 gigabytes) and is queried in under 1 ms.

redundant subtrees in the MH representation of a shadow map, and how to modify the structure so that only a single representative is left in memory. In other words, our goal is to create a structure that can capture both the hierarchical similarities exploited by MH representations as well as the non-hierarchical similarities exploited by DAG representations Fig. 2.12. We will refer to this structure as a *merged multiresolution hierarchy* (MMH). For clarity, in the following we will explain how to create an MMH from an MH representing a single shadow map. Nevertheless, the same ideas are fully applicable to shadow map cubes.

We propose to merge subtrees of an MH quadtree if we find that they (a) exhibit the same structure and (b) each corresponding pair of non-empty nodes have matching depth intervals. If both conditions are met, we can select one of the subtrees and replace its depth intervals for each node with the intersection of the corresponding nodes in the other trees. We then remove all other repeating occurrences and replace them with references to the modified subtree. This allows us to remove redundant subtrees from the hierarchy without violating the bounds of the dual shadow map.

In order to be able to merge similar structures at different depths, we change the MH representation to represent depth values as differentials relative to their parent node. For every node in the hierarchy, we subtract the mid-interval value of the parent from the interval bounds of the children. Since the mid-interval value will be stored in the final serialized quadtree, the original absolute value for a node is recovered by summing the depth values along a traversal path (Alg. 3). Changing the traditional MH representation from an absolute one to a relative one does not noticeably increase evaluation times since all ancestors of a node have to be visited during the top-down tree traversal, and will therefore be available due to cache pre-fetching. Having subtree values relative to their parent nodes is beneficial because it allows structurally similar subtrees at different levels of the hierarchy to be potentially merged.

A major task is to identify and match redundant subtrees in the hierarchy. Unfortunately, this task is not as straightforward as in the case of DAGs, where only identical subtrees are candidates for merging, and therefore keeping a dictionary of existing subtrees is all that is needed. Subtrees in a MH representation can potentially have very different depth intervals, but still be mergeable as long as an intersection for each pair of corresponding nodes and their intervals exists. Hence, building a dictionary of MH subtrees is not enough to capture similarity; we need to compare each corresponding depth interval between subtrees to establish mergeability. Further, multiple combinations for merges exist frequently, rendering this task a combinatorial problem.

High resolution MHs may potentially contain millions of subtrees, and so comparing each possible pair is infeasible in practice. Therefore, we restrict matching of subtrees to pairs of equal topology, which allows us to define clear partitions of subtrees. Although cases exist where a subtree is mergeable with another subtree of larger topology, these cases are unusual and would result in increased evaluation times for the replaced smaller topology branch. We propose a two-step approach to greatly reduce the number of tests and efficiently find small sets of similar subtrees.

The overview of our technique is as follows. We start by creating the high-resolution dual shadow map, which defines the per-textel depth intervals via depth-peeling as described in Sec. 2.3. We then iteratively sparsify each level bottom-up and create a quadtree

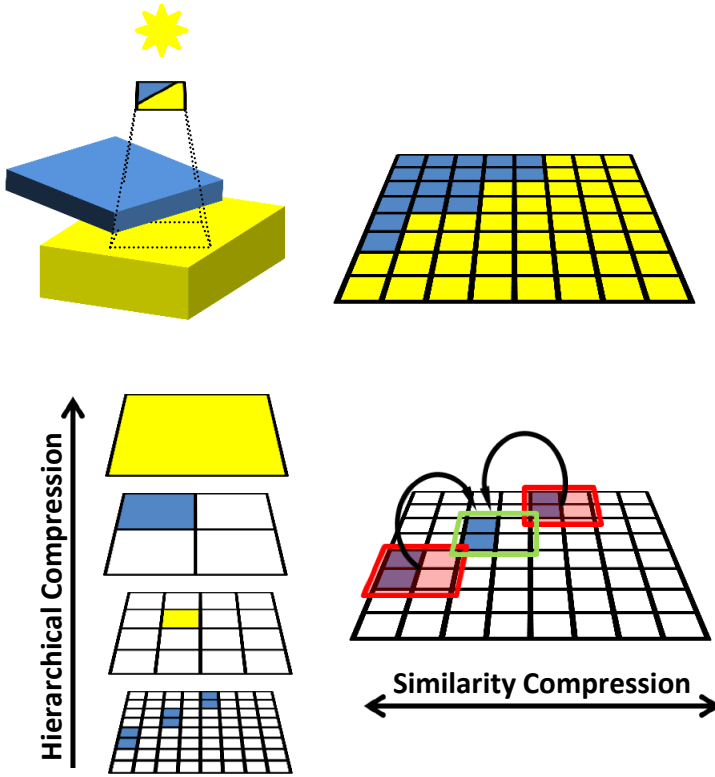


Figure 2.12: MMH captures both the hierarchical compression present in MH representations as well as the similarity compression present in DAGs.

representation, but keep the depth intervals for each node as auxiliary information. We then convert them to a relative representation as explained earlier. In the next step, we use a hash-based technique to partition the set of all inner nodes according to the topology of their descendants (Sec. 2.6.1). We then perform the pair-wise matching solely between elements in the hash collision lists, hereby enforcing comparisons only between trees of matching topology (not necessarily at the same depth in the hierarchy). Fig. 2.13 illustrates this procedure. To reduce the number of tests further, we restrict comparisons within each collision list to local areas by inserting elements following a spatial ordering. To accelerate the pair-wise subtree matching, we introduce an interval-based rejection test (Sec. 2.6.2). Finally, we discuss how to perform the final serialization step while taking into account the presence of merged subtrees (Sec. 2.6.3).

2.6.1. HASHING

Given an inner node (empty or full) in the initial MH quadtree structure, its child pointer always indicates the first existing child node in memory. All siblings of a child node lie

Algorithm 3 Pseudo-code for relative quadtree evaluation

```

function evaluateQuadtree(rootNode, coord) :
    value  $\leftarrow$  rootNode.value
    node  $\leftarrow$  rootNode
    while true:
        child  $\leftarrow$  getChildNode(node, coord)
        if isLeaf(child):
            return value + child.value
        if not isEmpty(child):
            value  $\leftarrow$  value + child.value
        node  $\leftarrow$  child

```

Algorithm 4 Pseudo-code for bottom-up hash code construction

```

function bottomUpHashCreation() :
    for level from deepestLevel to rootLevel
        for every node in level
            updateHash(node)

function updateHash(node) :
    node.size  $\leftarrow$  typeSize(node.type)
    if isLeaf(node)
        node.hash  $\leftarrow$  1
        return
    hash  $\leftarrow$  node.childFlags
    for each child in node.children
        node.hash  $\leftarrow$  node.hash + (child.size  $\ll$  childIndex)
        node.hash  $\leftarrow$  node.hash * (child.hash  $\ll$  childIndex)
        node.size  $\leftarrow$  node.size + child.size
    return hash

```

consecutively in memory and the parent node contains a flag byte that is used to indicate every child's existence and type (full inner node, empty inner node, or leaf node). Since each group of 4 sibling nodes is packed consecutively in memory, the subtrees we merge have a 4-node pack at their root level (see Fig. 2.13). Hence, each subtree has exactly one parent node, whose child pointer indicates the first node in the 4-node root level. If two subtrees can be merged, we redirect the pointer in one of their parents to the other subtree.

Testing all subtree combinations is impractical due to the very large quantity of them present in an MH. We therefore assign a hash code for each subtree, which we store in their parent node, such that potential merge candidates collide when inserted in a hash table. We then only need to test node pairs in the same collision lists.

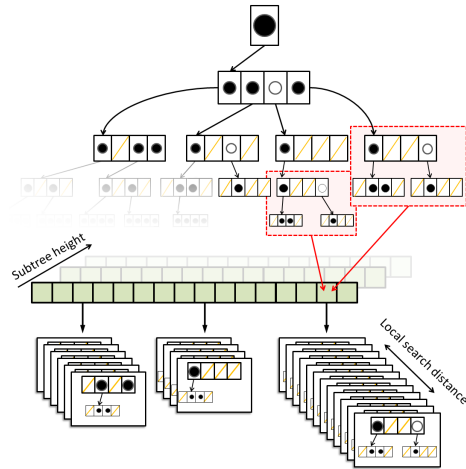
The hash encodes the topology of a subtree by involving the flags of all its nodes. Note that the parent node type does not have to match for two subtrees to be compatible, since the parent node is not part of the subtree. Further, we only match subtrees of the

same height, and thus we keep a separate hash table for each subtree height to avoid collisions. We insert each node into its corresponding hash table, according to the hash code stored in it. This allows us to keep a reference to the subtree parent in order to redirect its child pointer if a merge candidate is found. Hence, after each node has been inserted into the hash table corresponding to its height, each hash table entry contains a collision list representing all equal-topology subtrees (Fig. 2.13).

Since the hash value stored at each inner node represents its descendant subtree, it is convenient to produce the hash codes for the whole hierarchy in a single bottom-up sweep of the hierarchy. This allows us to easily track the height of the subtree, in order to insert it into the corresponding hash table. Further, we insert nodes into the hash table in a depth-first order to preserve a spatial locality inside each collision list. Our hash function, which fulfills these properties, is presented in Alg. 4.

2.6.2. SUBTREE MATCHING

So far, we have produced the collision lists of the hash tables. Now, the actual matching takes place and the hash tables are processed in the order of their height from large to small. Hereby, we make sure that larger subtree matching can remove larger parts of the tree before proceeding to test the smaller subtrees. The collision lists themselves can be treated in arbitrary order, as no elements of two separate collision lists will ever be tested. Therefore we can speed-up the subtree matching step by processing all collision lists of a single hash table in parallel, since there is no dependency between them. To further increase performance, we reduce the number of overall tests by a local matching and speed up the individual pair-wise tests by an early rejection test.



Local matching Testing all possible subtree pairs in a collision list would lead to a creation time that is quadratic with respect to the number of nodes. This would render the algorithm impractical, and thus we propose a local matching scheme to keep the run-time bounded. Having used a depth-first insertion, the nodes in each collision list correspond to spatially-close neighborhoods. It is likely to find matches between elements close

Figure 2.13: The matching procedure inserts subtrees into their corresponding hash table based on their height using a topology-based hash code. Subtree pairs in each hash table collision list within a certain distance are then tested to determine their mergeability.

Algorithm 5 Pseudo-code for subtree matching

```

function matchPairs(hashTable, searchLimit) :
  for each collisionList in hashTable
    listSize  $\leftarrow$  collisionList.size
    for i from listSize - 1 to 0
      for j from i + 1 to min(i + searchLimit, listSize - 1)
         $n_1 \leftarrow$  collisionList[i];  $n_2 \leftarrow$  collisionList[j]
        if ( $n_1.lMax < n_2.lMin$ ) || ( $n_1.hMin > n_2.hMax$ )
          continue
        if offsetTooLarge( $n_1, n_2$ )
          continue
        if fullTest( $n_1, n_2$ )
          merge( $n_1, n_2$ )
          break

```

in the list as they tend to be located nearby similar features. In consequence, we limit the testing of a node to a certain distance within a collision list. In our experiments, a distance limit of 1000 entries provided a good trade-off between compression and construction time. Furthermore, increasing this limit further will eventually lead to testing subtrees that cannot be merged since their distance in the final structure will be too large to represent as an offset pointer. We discuss the influence of this parameter further in Sec. 2.5 and show that the usage of local matching decreases compression only slightly while reducing the run-time of the algorithm by several orders of magnitude.

Early Rejection When testing two subtrees for merge compatibility, all of their nodes and depth intervals need to be compared. While this process can become costly, specially for large trees, a single mismatch is enough to stop the comparison. Hence, we introduce a fast early rejection test.

For our rejection test, we aggregate information about the depth intervals of each subtree and store it in their parent node. This is done during the previously described bottom-up traversal that computes the hash codes of each subtree. We exploit the observation that merging is only possible if the depth interval of all corresponding node pairs in the tested subtrees intersect (otherwise, there is no possible conservative depth value). In consequence, a conservative test checks if there is at least one node in one of the subtrees whose depth intervals fails to intersect the union of all intervals in the other subtree. In order to test this property, we keep track of the lowest and highest minimum and maximum depth of all node intervals for each subtree. If the highest minimum depth of a subtree is higher than the highest maximum of another subtree, at least one node exists in the first tree without an intersection with any node in the second one. The same is true if the lowest maximum depth of a subtree is lower than the lowest minimum of another subtree. Fig. 2.14 provides a graphical example of this test. In both cases, merging the two subtrees can be immediately rejected if the test fails.

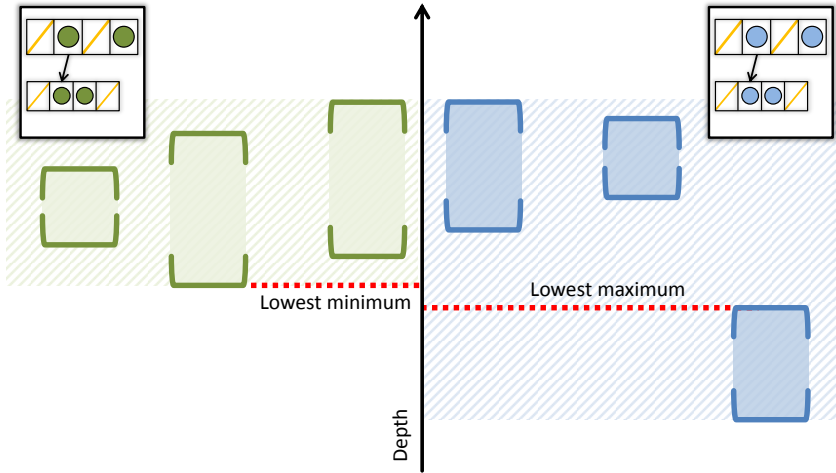


Figure 2.14: Compatibility testing of two trees with equal topology can be sped up by observing the aggregated statistics of their nodes' depth intervals. In this example, we see that the lowest minimum depth of all nodes in the left tree is higher than the lowest maximum depth of all nodes in the right tree. We can then conclude that the lowest interval in the right tree is guaranteed to not have an intersection with any intervals of the left tree.

Subtree Merging Once a pair of compatible subtrees has been found, we have to decide which one to remove and which one to keep. We apply a simple rule and always remove the one that appears earlier in the collision list. The reason is that the serialized quadtree uses positive offsets instead of full pointers, and removing the earlier subtree avoids a negative offset. The depth intervals of all nodes in the surviving subtree are modified to intersect the intervals of the nodes in the removed subtree. Hereby, we ensure that this subtree correctly represents both original subtrees. If the surviving subtree is later merged with another one, the intervals in its nodes will be the intersection of all the intervals of the corresponding nodes in the original trees. At this point, instead of actually removing the redundant subtree immediately from the hierarchy, we only mark its parent as merged and save a reference to the surviving subtree's root. All marked subtrees will later be skipped during the serialization step.

Alg. 5 summarizes the matching algorithm. Note that the outer loop over each element in the collision list proceeds in back-to-front order. This ensures that each subtree indexed by the outer loop is not already merged, thus allowing us to safely merge it with another subtree if we find a suitable match.

2.6.3. SERIALIZED TREE CREATION

Once the subtree matching is completed and redundant subtrees are marked in the hierarchy, we perform the quadtree serialization. While the original serialization applied a single depth-first order traversal, our altered serialization requires two passes through the hierarchy. The reason is that to correctly encode the redirected child pointers, we need to know the final position of a subtree in the serialized representation. To this extent, we perform a first pass where we only compute the final position of all nodes in

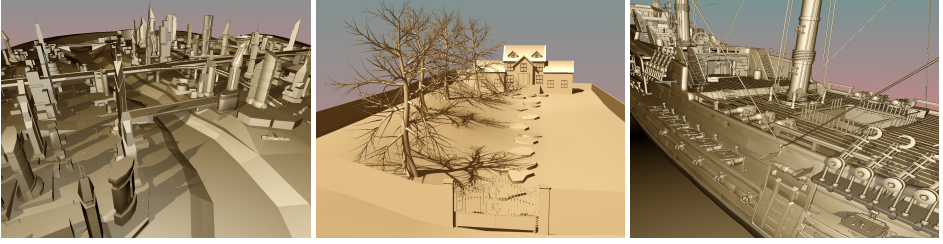


Figure 2.15: An overview of the tested CITYSCAPE, VILLA, and SHIP scene. The CLOSED CITY scene is shown in Fig. 2.1.

the serialized representation. In the second pass, we write the serialized structure and use the previously computed positions to assign the correct offset to the nodes that were marked as merged.

Offset scaling After the merging process, each inner node contains a pointer to a subtree, which might be shared by several nodes. These pointers will be represented in the form of 24-bits offsets into the serialized quadtree. The original MH quadtree used 16 bit offsets and added 8 bits of padding. In our case, merged subtrees may be very distant in the final serialized quadtree layout, and therefore we use the padding bits to increase the offset size to 24 bits.

Two nodes at different levels in the quadtree might share the same subtree, since subtrees are merged according to their topology and depth intervals, and not their position in the final structure (see Fig. 2.13). This means that during the quadtree traversal, the same subtree may be encountered at two different levels. Consequently, we need to ensure the offset scaling factors of those levels are the same. Fortunately, the quadtree depth-first order ensures that the scale factor will always be *one* for the lower levels; imagine a complete quadtree, then the first 12 levels ($4^{12} = 2^{24}$) contain only parent nodes whose offset to the children will fit into 24 bits. Therefore, we restrict merge operations to the lowest 12 levels to ensure that all merged nodes are at levels with scale *one*. Since the lowest tree levels typically contain most of the nodes, very little compression is lost by this restriction.

To ensure that all merged subtrees use unscaled offsets and fit in 24 bits, we determine whether there is a risk to exceed this bound during each merge test. We compute a conservative bound on the number of required bits for the resulting offset and forbid the merging operation if it is too large. This bound is the distance within the original (unmerged) quadtree as computed in [52], since our merging algorithm can only cause a decrease in this offset.

2.6.4. MERGED MULTIREOLUTION HIERARCHIES EVALUATION

In this section, we will compare MMH-based compression with MH-based and DAG-based compression, for our test scenes. We will also explore the impact of the rejection schemes and parameters involved in the merging process. The software implementation and test hardware matches the one described in Sec. 2.5.

	Method	4K ²	16K ²	64K ²	256K ²
CLOSED CITY	Kampe et al.	0.62	3.40	14.90	60.46
	MH	0.41	2.10	9.33	38.43
	MMH	0.30	1.43	5.80	23.73
CITYSCAPE	Kampe et al.	0.94	3.94	16.38	63.34
	MH	0.60	2.73	11.53	47.07
	MMH	0.40	1.67	6.83	27.71
VILLA	Kampe et al.	1.78	9.27	39.70	166.47
	MH	0.88	5.01	23.61	101.52
	MMH	0.63	3.62	17.36	74.11
SHIP	Kampe et al.	2.01	8.66	35.57	153.67
	MH	1.11	5.64	24.57	102.36
	MMH	0.82	3.83	16.34	67.64

Table 2.6: Size comparison in MB for DAG-based compression[25], MH-based, and MMH-based compression for our four test scenes and varying shadow map resolutions.

	Method	4K ²	16K ²	64K ²	256K ²
CLOSED CITY	Kampe et al.	0.098	0.53	4.88	65.23
	MH	0.27	1.24	5.80	56.17
	MMH	0.37	1.59	8.72	72.54
SHIP	Kampe et al.	0.12	0.67	6.16	78.74
	MH	0.31	1.59	7.08	69.30
	MMH	0.54	3.1	15.6	109.9

Table 2.7: Creation time comparison in seconds of DAG-based compression[25], MH-based compression, and MMH-based compression at varying resolutions for the CLOSED CITY and SHIP test scenes.

In Table 2.6, we compare the memory footprint of traditional MH compression, DAG compression and our MMH compression method. Our solution reduces the memory footprint by up to 60% compared to DAG compression and up to 40% compared to MH-based approaches. Averaged over all scenes and resolutions, our approach achieves 31.3% higher compression rates than traditional MH-based compression.

Table 2.7 shows a comparison of creation times for our method and the competing approaches. It can be seen that our approach adds an overhead to the construction time that is between 30% and 150% at higher resolutions in comparison to MH-based and DAG-based compression. However, precomputing and compressing a static shadow map only needs to be done once and construction time is often not considered as significant as memory reduction in applications typically employing this technique.

As explained in Sec. 2.5, we use a tile-based construction method since the entire uncompressed depth map would not normally fit in memory. CPU memory requirements are smaller, since the CPU only performs the merging operations on the already compressed MH structure tiles. For the scene in Fig. 2.1, the CPU memory usage never exceeded 600 MB during construction.

Evaluation performance at run-time is unaffected by the merging, remaining below 1 ms for single queries full HD image at 256k resolution, and between 1 ms and 2.5 ms using a 3x3 hierarchical PCF filter kernel, as shown in Table 2.3.

	Resolution	Interval mismatch	Offset too large	Full test rejection
CLOSED CITY	16K ²	67.2 %	0.00 %	32.63%
	64K ²	64.8 %	0.28 %	34.76 %
	256K ²	63.5 %	11.9 %	24.39 %
CITYSCAPE	16K ²	64.88 %	0.00 %	34.90 %
	64K ²	62.15 %	1.49 %	36.21 %
	256K ²	61.94 %	13.51 %	24.44 %

Table 2.8: Early rejection statistics showing the effectiveness of each individual test during the examination of node pairs. Note that the rejection criteria are given in the order that they are performed in the implementation, e.g., a pair rejected for interval mismatch will never be tested for offset incompatibility.

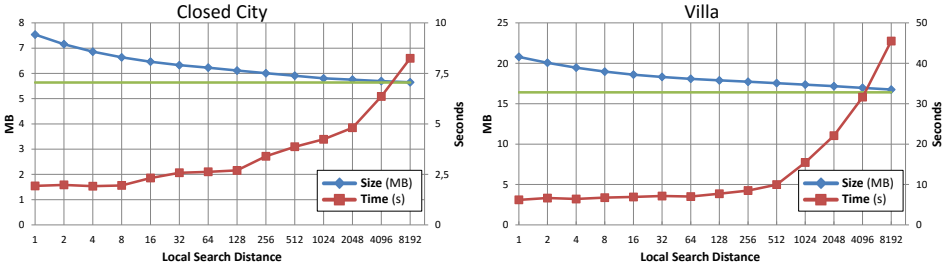


Figure 2.16: Compressed size vs. subtree merging time for varying local search distances in the CLOSED CITY and VILLA scenes at 64k resolution. The graph shows the diminishing returns of larger distances during the pair matching of subtrees. The green line shows the minimum attainable size, resulting from an infinite search distance. A reasonable trade-off between final size and creation time can be achieved using search distances between 500 and 1000.

We also evaluated the influence of the local search distance parameter (Fig. 2.16). Our standard choice of 1000 leads to a good trade-off between the achieved compression and construction time. Beyond this point the cost of comparing nodes dominates the total time, and the diminishing returns in achievable compression do not seem to justify the strong increase in construction time. Furthermore, local search distance is eventually limited by the maximum offset representable in the final quadtree structure.

Table 2.8 shows statistics for the early rejection test during the individual pair-wise subtree matching. It can be seen that the interval-based rejection test quickly eliminates most (avg. 64%) of the comparisons and only one quarter to one third of the subtree pairs perform a full node-by-node comparison. Furthermore, only a small amount of merges are rejected due to the 24-bit offset limit for our choice of local search distance.

Finally, Fig. 2.17 showcases a visual example of the merged nodes for a small section of the CLOSED CITY scene. As can be seen in the figure, most of the merging happens at the lower levels in the hierarchy. This is to be expected as for higher levels the amount of topology variations grows exponentially, and so fewer subtrees share the same topology, which leads to less merging possibilities.

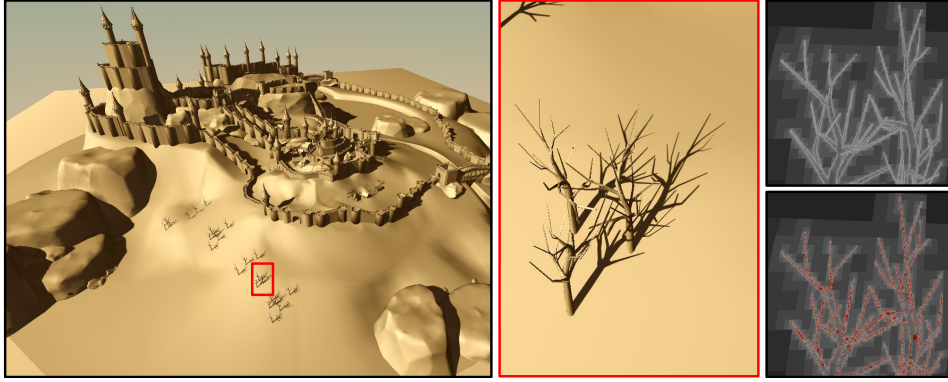


Figure 2.17: Part of a shadow map with a resolution of $256k \times 256k$ compressed with an MMH. The merging process eliminates the nodes marked in red and reduces memory usage from 38.43 MB down to 23.77 MB.

2.7. CONCLUSION AND FUTURE WORK

We presented a novel compression scheme for shadow maps based on multiresolution hierarchies. We demonstrated that our approach creates high-quality shadows for real-time rendering and achieves high compression rates. For example, our method is able to compress a 32-bit shadow map with a resolution of $2^{20} \times 2^{20}$ (uncompressed 4 terabytes) down to 0.0045% at best. Another benefit of our approach is that a multiresolution representation is highly beneficial for fast hierarchical filtering. Using a set of coherent shadow maps, we are able to create soft shadows or dynamic lights on a fixed trajectory. We also showcased a method to modify this structure in order to obtain an up to 40% extra compression rate while retaining the same run-time performance.

While our approach can handle non-closed geometry, these parts as well as very thin objects lead to a reduction of compression performance. This stems from the reduced size of the depth interval, diminishing the possibility for creating homogeneous regions. Nonetheless, this problem is shared by all related compression methods. Another issue is geometry that is viewed at grazing angles since our shadow map representation encodes a set of strictly axis-aligned planes. In the future, we would like to investigate alternative, non-linear representations to overcome these limitations.

Additionally, we would like to investigate non-regular subdivision schemes (e.g., based on multiresolution kd-trees) to provide more adaptivity to the underlying depth signal. Still, it is not clear how to efficiently construct these non-regular trees and if the overhead of storing subdivision information introduces a too large overhead. Finally, we want to investigate sparse decompositions that avoid storing topological information for empty inner nodes, e.g., matrix trees[53].

3

QUAD-BASED FOURIER TRANSFORM FOR EFFICIENT DIFFRACTION SYNTHESIS

L. Scandolo, S. Lee, E. Eisemann

Far-field diffraction can be evaluated using the Discrete Fourier Transform (DFT) in image space but it is costly due to its dense sampling. We propose a technique based on a closed-form solution of the continuous Fourier transform for simple vector primitives (quads) and propose a hierarchical and progressive evaluation to achieve real-time performance. Our method is able to simulate diffraction effects in optical systems and can handle varying visibility due to dynamic light sources. Furthermore, it seamlessly extends to near-field diffraction. We show the benefit of our solution in various applications, including realistic real-time glare and bloom rendering.

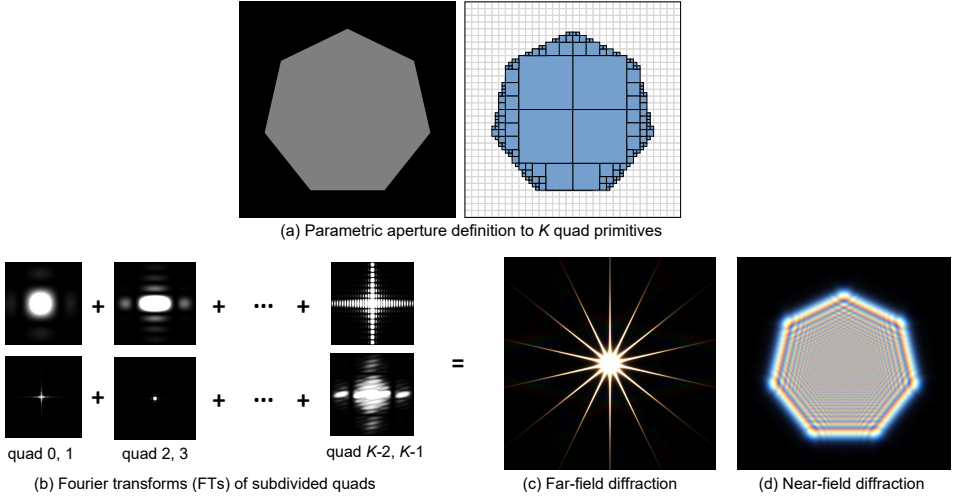


Figure 3.1: Given an aperture image, we subdivide it into a list of parametric quad primitives forming a quadtree (a). The Fourier transform of each quad is efficiently evaluated with a closed-form solution (b), and combined to render the final far-field diffraction (c). We extend this approach to also produce near-field diffraction (d). (c) and (d) are rendered in 0.85 ms and 2.1 ms at 1024×1024 resolution, respectively.

3.1. INTRODUCTION

DIFFRACTION phenomena can be observed when taking a photo of a strong light source, which typically results in bursting streaks (or glares) [54–57]. While often used for artistic purposes, these effects also increase the perceived brightness of light sources [54, 58] and are important elements of realistic rendering beyond typical ray optics.

By definition, diffraction refers to the interference and superposition of spherical waves propagating around obstructions (i.e., apertures) in the path of light. Fresnel’s *diffraction integral* [59], based on Huygens’ principle of wavelet superposition, mathematically models diffraction. In practice, additional geometric approximations (Fresnel and Fraunhofer approximations) can be applied for distant observation planes (at near and far field).

Near/far-field diffraction has typically been computed by formulating the problem as a Fourier Transform (FT) and resorting to optimized FT implementations [55], usually relying on an image-space Discrete FT (DFT) under the assumption of a periodic function. Given a 2D sampling resolution of $M \times N$, the per-pixel time complexity of the DFT is $O(MN)$. Separable integration and Fast DFT (FFT) are more efficient alternatives but still costly and oriented mostly to offline processing.

While many applications use static aperture patterns, a faster simulation of diffraction can enable applications to capture physical details caused by dynamic aperture changes from area lights, stray lights, and optical elements. However, real-time applications have hardly used dynamic diffraction due to the low FT performance.

Our key observation to accelerate diffraction computations is that the optical aperture image is relatively simple in its color and shape; real light sources are strong (mostly

white) and iris apertures are polygon-like. Consequently, we opt for a geometrical representation of the aperture and formulate a closed-form solution for its shape. Similar ideas have been applied in electromagnetics and optics for polygon support and specific patterns [60–62] but no generalization for rendering exists. While similar in spirit, our solution is generalizable, flexible, and parallelizable. Specifically, we present an efficient diffraction rendering technique. Instead of DFT, we combine closed-form FT of vector primitives in the continuous domain. At its basis, our approach uses *quads* with constant intensity as such vector primitives and builds upon their closed-form solution to the diffraction integral. We show that quads are an efficient basis even for input images with complex shapes, since we use a hierarchical tessellation. Contrary to previous solutions, the complexity of our transformation relies on the number of quads (tessellating an aperture) instead of the image resolution, which proves typically advantageous. We also introduce a progressive refinement, which exploits spatiotemporal coherence for incremental updates. Our method is suitable for graphics-processing-unit (GPU) execution, enabling interactive and dynamic diffraction rendering. Besides achieving a high-quality far-field approximation, our approach can be extended to a high-quality near-field approximation as well.

3.2. RELATED WORK

3.2.1. DIFFRACTION MODELING AND RENDERING

Diffraction and wave optics in computer graphics are used mostly for glare rendering and reflectance modeling.

Glare rendering relates to the properties of an optical system with an aperture for strong light sources. Early studies to reproduce the patterns used analytic approximations where diffraction gratings are mapped to streaks [63]. It is simple but has limited modeling power for arbitrary apertures. Kakimoto et al. introduced a practical rendering framework and improved accuracy via the FT-based diffraction integral [55], which was later used for lens flares [57]. Other work attempted to model glares in the human visual system [54, 56, 64]. Spencer et al. intensively discuss physiological glare components and their causes [54]. Additional modeling (e.g., light scattering) and temporal fluctuation were introduced for higher perceptual accuracy [56].

The reflectance modeling to spectral interference has focused on bidirectional reflectance functions of surface microstructures. An effective diffraction analysis was proposed (avoiding the evaluation of the Kirchhoff integral) for the microstructure of surfaces (e.g., a compact disc) [65]. Further efficiency can be obtained with spherical harmonics and the Chebyshev approximation [66, 67]. A high-quality solution to generate multiple scattering effects was introduced using destructive interference [68]. Recently, many studies have extended microfacet models to simulate iridescence from scratches [69], metal surfaces [70], and thin-film coating [71], often integrating data-driven acquisition/rendering [72] or a two-scale geometry model [73].

In the previous work, simple texturing/billboard or iterative filtering are common in interactive applications [64, 74], but high-quality modeling relies on FT [55, 56, 65]. The FT results are stored in lookup tables as Point-Spread Functions (PSFs) for online usage due to their cost. In contrast, our work accelerates the FT itself, which enables real-time

evaluation and applications.

3.2.2. ACCELERATION TECHNIQUES FOR FOURIER TRANSFORM

A FT of digital images typically relies on numerical integration. FFT [75] is typically applied when the sampling interval is uniform over input/output and the signal is periodic. For general cases, FFT is currently the most efficient solution and recent work focuses on optimizations. FFTW is an optimal CPU implementation [76] and parallel versions of FFT have received attention due to the increased throughput on modern hardware [77–79].

A substantial amount of literature exists for near-optimal or approximate DFTs, attaining sublinear complexity. We refer the reader to [80] for a survey. A notable attempt is Sparse FT [80], which excludes negligible coefficients from the FT evaluation, leading to an approximate but high-quality result.

For simple input functions, the continuous FT can have a closed-form solution. The function can be either piecewise constant [81] or piecewise discontinuous [82, 83]. Several parametric functions admit closed-form solutions, including sunburst patterns [60], polygonal shapes [61, 62], and triangular meshes [84, 85]. The majority of such closed-form solutions are applied for electromagnetic analysis or the detection of diffraction [86]. Rendering of dispersion and near-field ringing has hardly been explored in this direction [87].

Our work introduces and extends concepts for accelerated FT computation to rendering for real-time diffraction effects. We exploit that large constant areas in an image are quite common for diffraction rendering; e.g., an area light source or an aperture. Quad primitives approximate such regions well and lead to GPU-friendly light-weight computations, resulting in good performance. We also propose a hierarchical solution and progressive refinement that lead to additional acceleration and enable dynamic visibility integration. These concepts are difficult to couple with general supports, such as triangles [84, 85], where the performance benefits of closed-form solutions can be lost.

3.3. BACKGROUND

We first revisit the standard FT and its formulation for diffraction before introducing our contributions.

3.3.1. STANDARD FOURIER TRANSFORM

Given a function f in the spatial domain, the FT operator \mathcal{F} defines $\mathcal{F}(f)$ as a continuous integration over the spatial domain:

$$\mathcal{F}(f)(u, v) = F(u, v) = \iint_{-\infty}^{\infty} f(x, y) h(ux + vy) dx dy, \quad (3.1)$$

where $i = \sqrt{-1}$, $h(x) := e^{-i2\pi x}$, and (u, v) is a frequency-domain position. If f is regularly sampled and periodic, its DFT is:

$$F(u, v) \approx \frac{1}{\sqrt{MN}} \sum_{y=0}^{N-1} \sum_{x=0}^{M-1} f(x, y) h\left(\frac{ux}{M} + \frac{vy}{N}\right), \quad (3.2)$$

where M, N are the sampling resolutions over x, y respectively. An effective DFT evaluation relies on FFT, reducing the per-pixel complexity to $O(\log M + \log N)$, but is still costly in a real-time context.

3.3.2. DIFFRACTION WITH FOURIER TRANSFORM

Given the source aperture plane position $\mathbf{p} := (x, y)$ and the destination position (x', y') at the observation plane at distance z , the Huygens-Fresnel equation describes diffraction as:

$$D(x', y', \lambda) = -\frac{i}{\lambda} \iint f(x, y) \frac{e^{ikR}}{R} dx dy, \quad (3.3)$$

where λ is a wavelength of light, $k = 2\pi/\lambda$, and $R = ((x' - x)^2 + (y' - y)^2 + z^2)^{1/2}$ [59]. The power spectrum of $D(x', y', \lambda)$ corresponds to the intensity that we wish to visualize in the final image. Computational efficiency can be increased by using Fresnel's approximation for Near-Field Diffraction (NFD) [59]:

$$D(x', y', \lambda) \propto \frac{1}{z\lambda} \iint f(x, y) h\left(-\frac{x^2 + y^2}{2z\lambda}\right) h\left(\frac{xx' + yy'}{z\lambda}\right) dx dy. \quad (3.4)$$

Note that we drop a phase term as it is irrelevant to the power spectrum. When the destination plane is assumed to be far (i.e. $(x^2 + y^2) \ll z\lambda$), Far-Field Diffraction (FFD) becomes:

$$D(x', y', \lambda) \propto \frac{1}{z\lambda} \iint f(x, y) h\left(\frac{xx' + yy'}{z\lambda}\right) dx dy, \quad (3.5)$$

When we let $(u, v) = (x'/z\lambda, y'/z\lambda)$, FFD becomes a scaled FT:

$$D(x', y', \lambda) \propto F(u, v)/(z\lambda), \quad (3.6)$$

where u and v can be considered spatial frequencies [59].

$$D(x', y', \lambda) \propto \frac{1}{z\lambda} \iint f(x, y) e^{-i2\pi(\frac{xx' + yy'}{z\lambda})} dx dy. \quad (3.7)$$

3.4. OUR APPROACH

Our approach uses axis-aligned *quad* primitives with *constant* intensity. Quads allow us to efficiently approximate arbitrary shapes via tessellation into a set of disjoint quads. Given that our input is a digital image, which is by definition composed pixels (small quads), the constancy assumption does not restrict our solution. While different shapes can be used as primitives, obtaining a tessellation of the input image into more complex shapes becomes a bottleneck. Conversely, quads lend themselves well to a hierarchical representation, which reduces their number, and results in an efficient implementation.

We first introduce our novel approach on a primitive-based FT and specialize it for quad primitives (Sec. 3.4.1). Then, we present our approach to render FFD effects (Sec. 3.4.2). In this context, we will also describe the efficient tessellation of the input into a hierarchical representation. We then explain accelerations to our solution (Sec. 3.4.3) and show how to integrate occlusion and area lights (Sec. 3.4.4). Finally, we extend our approach to NFD (Sec. 3.4.5).

3.4.1. PRIMITIVE-BASED FOURIER TRANSFORM

Here, we introduce our primitive-based FT, which we use for efficient diffraction synthesis. Fundamental to our method is the reformulation of the image-space FT. The idea is to decompose the input signal into a sum of primitives (e.g., polygons). The primitive-based FT then uses the superposition principle, which states that the FT of each primitive can be computed independently:

$$\mathcal{F}(af(x) + bf(y)) = a\mathcal{F}(f(x)) + b\mathcal{F}(f(y)). \quad (3.8)$$

The superposition principle has also been shown for diffraction in optics (e.g., Babinet's principle) [59, 88].

Initially, we assume all primitives are disjoint, i.e., the original signal is a union of all primitives without intersection. Then, the FT can be computed as a sum:

$$F(\mathbf{q}) = \sum_{k \in \mathcal{K}} W_k(\mathbf{q}), \quad (3.9)$$

where $\mathbf{q} := (u, v)$, \mathcal{K} is the set of primitives, and W_k the transform of the primitive k . To compute W_k , we assume k is given by a function $f_k(\mathbf{p})$ and a domain Ω_k , typically bounded in \mathbb{R}^2 . We then obtain:

$$W_k(\mathbf{q}) = \mathcal{F}(f_k)(\mathbf{q}) = \int_{\Omega_k} f_k(\mathbf{p}) h(\mathbf{p} \cdot \mathbf{q}) d\mathbf{p}. \quad (3.10)$$

Quad-based Closed-Form Solution Axis-aligned quads with constant intensity are easy to integrate, and result in an efficient and concise closed-form solution. Additionally, we show that using the shift property of the FT, we can achieve further acceleration.

A quad k with constant intensity I_k is defined as the inside of a bounded rectangular domain $\Omega_k = \{(x, y) \in \mathbb{R}^2 \mid |x - c_x| \leq s_x/2, |y - c_y| \leq s_y/2\}$, which is centered at $\mathbf{c}_k = (c_x, c_y)$ with a size of $\mathbf{s}_k = (s_x, s_y)$. Since the function f_k of k is separable along the x and y axes, f_k is defined as a tensor product of boxcar functions:

$$f_k(x, y) = I_k (\Pi_{c_x, s_x} \otimes \Pi_{c_y, s_y})(x, y) = I_k \Pi_{c_x, s_x}(x) \Pi_{c_y, s_y}(y), \quad (3.11)$$

where the boxcar function is defined as $\Pi_{c, s}(x) = H(x - c + s/2) - H(x - c - s/2)$, and $H(x)$ is the Heaviside step function. The closed-form FT of the boxcar function $\Pi_{c, s}(x)$ is given as:

$$\Gamma_{c, s}(u) = \mathcal{F}(\Pi_{c, s}(x))(u) = s \operatorname{sinc}(\pi u s) h(c), \quad (3.12)$$

where $\operatorname{sinc}(x) = \sin(x)/x$. Then, we can obtain the closed-form solution to $W_k(\mathbf{q})$ as a tensor product of Γ :

$$W_k(\mathbf{q}) = I_k \left(\Gamma_{c_x, s_x} \otimes \Gamma_{c_y, s_y} \right)(\mathbf{q}). \quad (3.13)$$

While this evaluation is simple for a single quad, having several quads leads to many redundant computations. We can reduce the number of computations using the shift property, which states:

$$W_m(\mathbf{q}) = h(\mathbf{d} \cdot \mathbf{q}) W_k(\mathbf{q}). \quad (3.14)$$

This holds for two quads k and m of the same spatial support size, with a displacement \mathbf{d} , such that $f_m(\mathbf{p}) = f_k(\mathbf{p} + \mathbf{d})$. Hence, we can reuse the FT of a single quad for the FTs of all equally-sized quads.

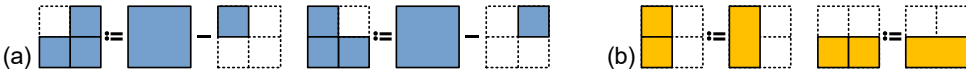


Figure 3.2: Examples of the non-uniform quadtree subdivision: subtractive superposition (a) and non-square rectangles (b).

Rendering the object-space FT The rendering is straightforward. For a given list of quads, we evaluate the FT for each pixel using Eqs. 3.9, and 3.13, efficiently on the GPU. We employ Eq. 3.14 to compute the quad FT only once.

3.4.2. FAR-FIELD DIFFRACTION RENDERING

Here, we introduce the application of the previous analysis to achieve efficient FFD effects. We describe the approach for a point light source and how to decompose the aperture into a small set of quads to achieve a fast diffraction computation.

Principle Given a point light source and a camera with an aperture and image plane, we can produce a realistic diffraction pattern in real time. As pointed out in Sec. 3.3, assuming that the aperture is given in the form of a binary image, its FT will yield the desired diffraction pattern. In principle, we could treat each texel of the aperture image as a quad, collect them in a list, and apply the solution from the previous section. This means that for each image pixel, we need to perform a summation over all quads. Therefore, minimizing the number of quads will have a big impact on the computation time.

Hierarchical Decomposition of Aperture and Rendering To reduce the number of quads, we use a *quadtree*, which hierarchically tessellates the aperture (Fig. 3.1) for a compact representation.

We perform a bottom-up quadtree construction, which is fast to compute. The process is initiated from the binary aperture image and proceeds from mipmap to mipmap level, outputting a list of quads in the process. We consider groups of four texels at a time, which correspond to a single texel from the next mipmap level, and depending on their value we initialize the texel value from the next mipmap level. If the four considered texels have the same value, we consider them merged, write their common value on the next mipmap level and proceed without outputting any quads. Otherwise, we output a variable amount of quads depending on the values of the texels. Contrary to a typical quadtree, we introduce a special non-uniform subdivision procedure (Fig. 3.2). Specifically, if only one quad has a value of one, we attach this single quad to the list and initialize the next mipmap level texel with a zero. We process the case of two non-adjacent quads similarly. If two adjacent quads have a value of one, a merged quad is outputted (Fig. 3.2b) and again the next mipmap level texel is initialized to zero. If three quads have a value of one, we initialize with a one the next mipmap level pixel, and output a negative quad (Fig. 3.2a), which has the support of the quad with value zero but will be subtracted after transformation.

During the evaluation of the quad list, we employ the shift property. For each quad shape and size, we compute its FT only once, and apply Eq. 3.14 for congruent quads. Hereby, we accelerate the FT evaluation without reducing quality.

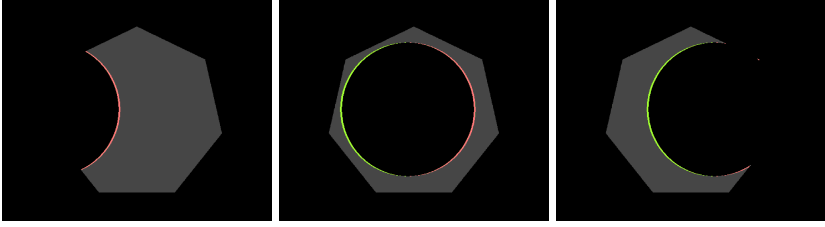


Figure 3.3: Example frames of a moving circle occluding a heptagonal aperture. Our progressive method can update the FT for the entire aperture by adding/subtracting only the FTs of newly disoccluded/occluded (green/red) quads to/from the previous calculation.

3

3.4.3. ACCELERATIONS

Culling and Symmetry An important observation is that a significant portion of the FT output contains zeros (in terms of the power spectrum) and skipping these pixels would accelerate computations. Testing for zeros in the power spectrum at full resolution would be too expensive. Instead, we use a lower-resolution image (in practice $1/8^2$) and additionally test if the *analytical* derivatives of the power-spectrum of the FT are zero. While not strictly conservative, the continuity of the derivatives leads to a very good estimate and we found no differences with respect to the actual zero test at full HD resolution. The previous pixel-based FTs lack analytical derivatives and cannot profit from such an acceleration.

Additionally, we exploit mirror symmetry in the FT. We evaluate only the half-plane and mirror it, obtaining the other half for free. When the input image has additional symmetries, which are common for apertures, we can exploit these as well.

Progressive Refinement Another benefit of the object-space transformation (Eq. 3.9) is the use of *progressive* refinement. We capture *dynamic* spatio-temporal changes, which significantly distinguishes our solution from the pixel-based FTs. When the input apertures show coherence, instead of recreating the FT from scratch, we update the FT by exploiting the superposition principle. To this extent, we derive a quad list that represents the difference between the current and previous aperture. We perform a bottom-up procedure on this difference (Fig. 3.3), where we extract all positive and negative quads. This procedure is similar to the original extraction, but we deal with the positive and negative parts independently in different texture channels, which allows us to apply the same merging strategies as before without making the quadtree creation procedure more complex. Progressive refinement proves very beneficial for animation, since typically only a reduced number of quads are needed to update the FT. Additionally, it is highly important when including visibility changes to the aperture shape (Sec. 3.4.4).

Accelerated Spectral Integration So far, we have described a method for computing the monochromatic diffraction pattern for a single wavelength. For visible diffractions, there is usually a need to cover the whole visible spectra. For an accelerated spectral integration, we extend that introduced in [57].

We start with a reference wavelength λ_r (e.g. 587.6 nm), for which we compute its FT F_{λ_r} . Each different λ gives a relative wavelength scale $\rho(\lambda) = \lambda_r / \lambda$. Given the observation

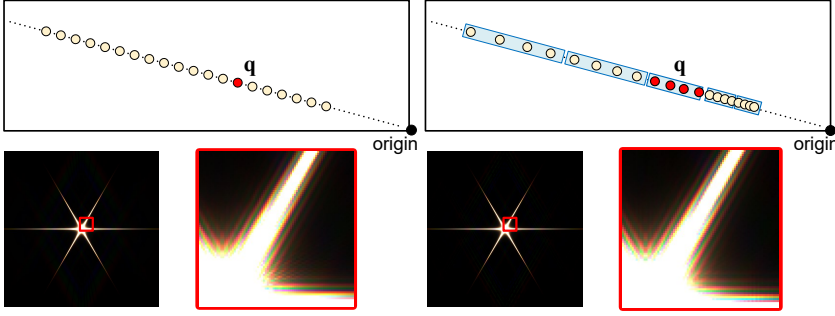


Figure 3.4: Uniform spectral scaling (left) and 4-channel batch spectral scaling (right), with examples for the diffraction of an hexagonal aperture. Our batch scaling saves the amount of lookups from the base pattern by roughly a factor of four.

distance z , the diffraction for λ can be expressed in terms of the diffraction of λ_r as:

$$D(x', y', \lambda) \propto \frac{1}{z\lambda} F_\lambda(\mathbf{q}) = \frac{1}{z\lambda_r} \rho(\lambda) F_{\lambda_r}(\rho(\lambda)\mathbf{q}). \quad (3.15)$$

The required spectral samples correspond to scaled texture locations $(\rho(\lambda)\mathbf{q})$ in Eq. 3.15, lying on the same line passing through \mathbf{q} and the origin. For a fixed z , $1/(z\lambda_r)$ is a constant. Using Eq. 3.15, we establish a relation between the color pattern S_k created by a quad k :

$$\begin{aligned} S_k(\mathbf{q}) &\propto L_k(\mathbf{q}) = \int_{\lambda \in \Lambda} \mathcal{X}(\lambda) \|F_\lambda(\mathbf{q})\|^2 d\lambda \\ &= \int_{\lambda \in \Lambda} \mathcal{X}(\lambda) \rho(\lambda)^2 \|F_{\lambda_r}(\rho(\lambda)\mathbf{q})\|^2 d\lambda, \end{aligned} \quad (3.16)$$

where $\|F\|^2$ is the power spectrum of the FT, Λ is the spectral domain, $\mathcal{X}(\lambda)$ is a normalized spectral response function for a particular chromaticity channel (e.g., X, Y, or Z in XYZ color space), and the scale relates to the exposure time. In general, $\mathcal{X}(\lambda)$ is empirically defined in terms of piecewise measurements; a recent analytic approximation exists [89] but does not lead to a closed-form integration. Hence, we numerically integrate it via:

$$L_k(\mathbf{q}) \approx \sum_{\lambda \in \Lambda'} \underbrace{\mathcal{X}(\lambda) \rho(\lambda)^2}_{\text{dispersive weights}} \|F_{\lambda_r}(\rho(\lambda)\mathbf{q})\|^2 d\lambda, \quad (3.17)$$

where Λ' is a discrete subset of Λ in a finite visible spectral range, and $\sum_{\lambda \in \Lambda'} \mathcal{X}(\lambda) d\lambda = 1$.

While the scaling-based integration (Eq. 3.17) avoids brute-force spectral sampling, many samples (e.g., > 60) are needed to avoid spectral aliasing. Therefore, we propose *batch-scaling* using an intermediate four-channel texture. In this texture, we store four nearby samples of $\|F_{\lambda_r}\|^2$ along the scaling line $\rho(\lambda)\mathbf{q}$ in a RGBA quadruplet. In a second pass, we sparsely sample this texture along the same line recovering four samples at a time, which are then scaled by the dispersive weights (Eq. 3.17). In this way, we only need a quarter of the original texture lookups. As reference wavelength λ_r , we use the geometric mean of the extrema of the visible wavelength range (e.g., 529 nm for [400, 700])

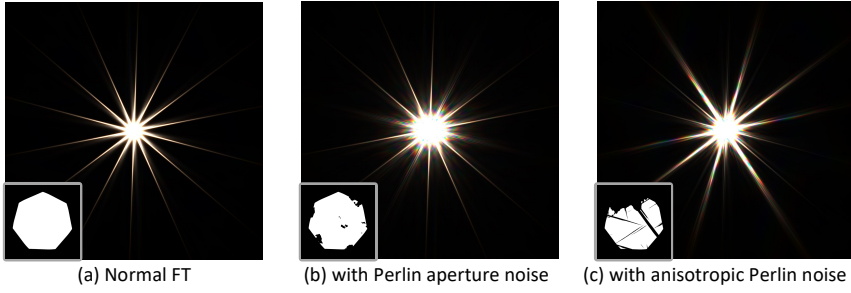


Figure 3.5: Examples of diffraction patterns without noise (a) and with the addition of Perlin noise (b) and anisotropic noise (c).

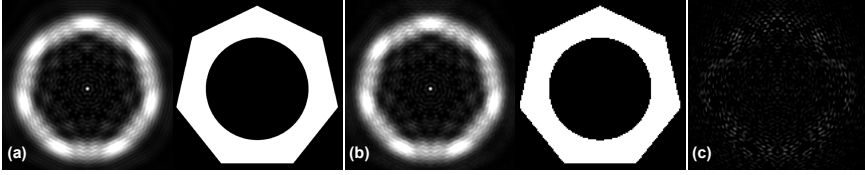


Figure 3.6: Near-field diffraction pattern of a partially occluded aperture using a quadtree resolution of 1024^2 (a) and 128^2 (b), and a $5\times$ difference visualization (c). Both results are very similar but the lower quadtree resolution results in a $\sim 7\times$ speedup.

nm). Using batch-sampling leads to non-uniform spacing towards the extrema but the difference to uniform sampling is marginal, as seen in Fig. 3.4.

3.4.4. OCCLUSION AND AREA LIGHTS

Up to now, we have ignored any kinds of occlusions additional to the aperture shape. Given the efficiency of our approach, we can integrate dynamic visibility changes in the scene.

One simple type of occlusion results from imperfections such as floating dust particles on lenses. These occlusions could be approximately handled by integrating perturbation of the input aperture pattern. We can use arbitrary noise patterns but Perlin noise offers spatiotemporal coherence for animation (Fig. 3.5).

For more structural occlusions due to a blocker, our approach can show major advantages. In this case, we can render the aperture as seen from the light source and project the scene geometry on top of it to derive the unblocked part of the aperture. The resulting aperture image can then be transformed with our approach. In this scenario, the progressive refinement proves particularly useful.

So far, we have assumed a point light source and obtained only the Point-Spread Function (PSF) of the resulting aperture image. In reality, light sources cover an area. Consequently, we convolve diffraction patterns over this area by using them as PSFs [56].

However, when integrating visibility, each point in the area light source may exhibit different viewing conditions, impacting the diffraction pattern. Here, for each point sample, we need to render the scene towards the aperture. For large area lights, this step remains costly. Although we did not investigate this direction, image-warping approxi-

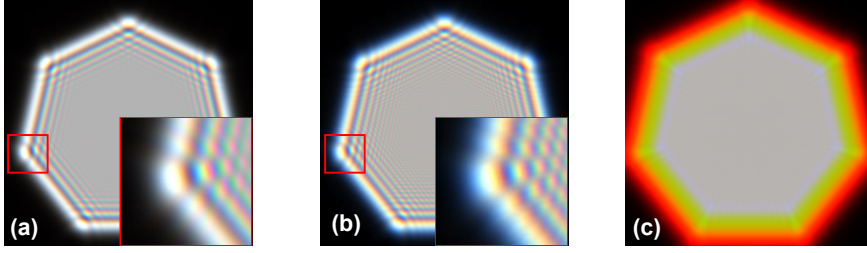


Figure 3.7: Spectral integration (over 60 wavelengths) of the NFD pattern (a) cannot be approximated as done for the FFD (c). However, a heuristically chosen scaling factor (here, $1 + 0.05(1 - \rho)$) still can produce a plausible approximation (b).

mations, or hierarchical point-based solutions such as the one described by Holländer et al. [90] can be used. As a cheaper alternative, we could also consider a small neighborhood in the scene image around each light sample and intersect the light-source pixels with the aperture image. In this way, we approximate a projection of the aperture from the image plane into the scene and onto the light. Our approximate projection is meaningful under the assumption that all scene geometry is closer to the aperture than the light source. When the geometry is fairly far from the aperture, the quality of the resulting approximation is reduced.

3.4.5. NEAR-FIELD DIFFRACTION

Having derived FFD patterns (Eq. 3.5), we also seamlessly extend our work to NFD, where the far-field assumption does not hold. NFD is often related to *ringing* at the edges of bokeh patterns or ghosting apertures in lens-flare rendering [57, 91].

It is known that the superposition principle also holds for near-field diffraction [59], as evidenced by Eq. 3.4. Consequently, we can compute it as a sum in terms of the NFD of a set of primitives:

$$G(u, v) = \sum_{k \in \mathcal{K}} V_k(u, v), \quad (3.18)$$

where V_k is the contribution of quad k in the set \mathcal{K} . Using Eq. 3.4 and the constancy assumption, we obtain:

$$V_k(u, v) = (I_k / z\lambda) \left(g_{c_x, s_x}(u) \right) \left(g_{c_y, s_y}(v) \right), \quad (3.19)$$

where

$$g_{c, s}(t) = \int_{c-s/2}^{c+s/2} h(tx - \frac{1}{2z\lambda}x^2) dx. \quad (3.20)$$

Unlike the FFD, the integral we need to solve involves square terms of x , which have no analytical solution in the general case, and so we use a numerical approximation. The integral to approximate can be decomposed into real and imaginary parts:

$$\int_0^t e^{i(ax+bx^2)} dx = \int_0^t \cos(ax+bx^2) dx + i \int_0^t \sin(ax+bx^2) dx. \quad (3.21)$$

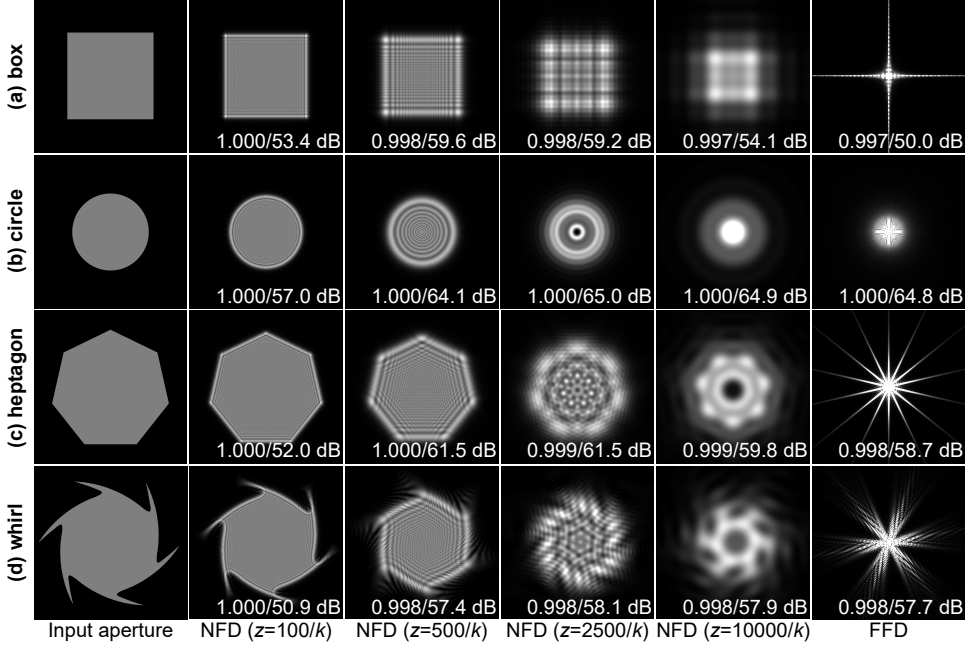


Figure 3.8: FFD and NFD outputs (scaled for illustration) for the input aperture images (the first column) used for the experiment. The numbers at the bottom indicate SSIM/PSNR values measured against the references (generated by discrete NFD/FFD solutions).

Using the Fresnel integrals $C(x)$ and $S(x)$, we can express these terms as:

$$\int_0^t \cos(ax + bx^2) dx = \frac{\sqrt{\pi}}{\sqrt{2b}} \left(\cos \frac{a^2}{4b} C(\phi_t) + \sin \frac{a^2}{4b} S(\phi_t) \right) \quad (3.22)$$

$$\int_0^t \sin(ax + bx^2) dx = \frac{\sqrt{\pi}}{\sqrt{2b}} \left(\cos \frac{a^2}{4b} S(\phi_t) - \sin \frac{a^2}{4b} C(\phi_t) \right), \quad (3.23)$$

where $\phi_t = \frac{a+2bt}{\sqrt{2\pi b}}$ and $C(t)$ and $S(t)$ are defined as:

$$C(t) = \int_0^t \cos(\pi x^2/2) dx \quad \text{and} \quad S(t) = \int_0^t \sin(\pi x^2/2) dx. \quad (3.24)$$

In general, $C(t)$ and $S(t)$ do not have analytic solutions, and thus, we use their numerical approximations [92] by precomputing a 1D look-up table to use during evaluation.

Efficient Rendering Our approach for NFD is more expensive than our FFD solution given that we need to compute the complex-valued Fresnel integrals four times for each quad. Yet, we can observe that the product in Eq. 3.19 is again separable. To avoid redundant computations, we propose an effective two-pass optimization, such that unique combinations of (k, u) and (k, v) are evaluated only once. In the first pass, for each quad, we precompute $g_{c_x, s_x}(u)$ and $g_{c_y, s_y}(v)$ for all u and v , respectively, and store these results

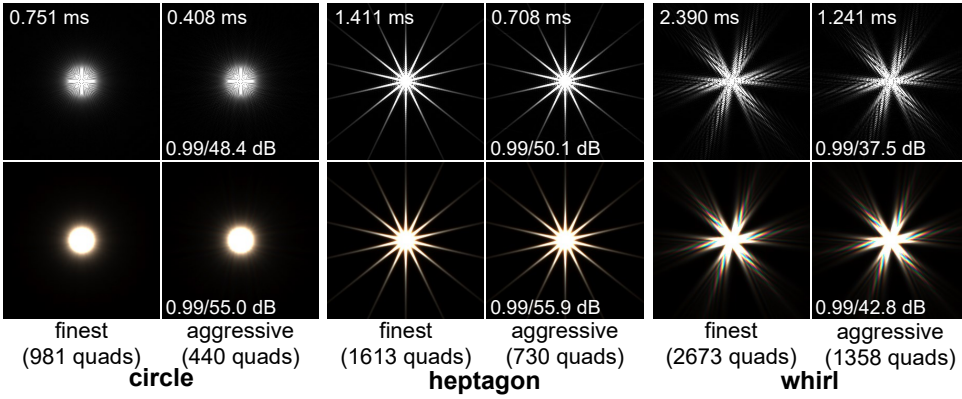


Figure 3.9: Comparison of quality in terms of quadtree resolution for monochromatic and dispersive FFDs. The one-step further downsampling from the finest quadtree resolution results in a negligible visual difference (SSIM/PSNR at the bottom), while increasing speed by a factor of two (timings given at the top).

aperture	256 ²	512 ²	1024 ²	2048 ²	4096 ²
box	40	40	40	617	617
circle	109	227	440	981	1976
heptagon	176	389	730	1613	2996
whirl	303	647	1358	2673	5278

Table 3.1: Number of quads produced at the quadtree tessellation.

in a texture. Then, in the next pass, for every pixel, $V_k(u, v)$ can be obtained by combining these partial results according to Eq. 3.19, which reduces the number of evaluations drastically.

For the FFD, we speed up computations by performing a culling pass at a lower resolution, and enabling progressive refinement for dynamic scenes. Our experience shows that the NFD result is not sensitive to high-frequency content for low z values and a coarse quadtree can be used without significant quality loss (Fig. 3.6).

With respect to the spectral integration, the previously-used simple scaling for λ_r cannot be applied (Fig. 3.7c). For correct results, we need to repeat Eq. 3.19 for dispersive samples (Fig. 3.7a). However, we empirically found that scaling with smaller amounts (e.g., $1 + 0.05(1 - \rho)$) leads to a plausible approximation but the result will no longer be physically accurate (Fig. 3.7b).

3.5. RESULTS

In this section, we assess performance and quality of our solutions for FFD and NFD and provide several examples of our results.

3.5.1. FAR-FIELD DIFFRACTION

The experiments used four input aperture shapes; the monochromatic images produced for FFD (and NFD) are shown in Fig. 3.8, along with quality comparison with respect to the brute-force DFT-based reference solutions. Their resolution scales from 256^2 to

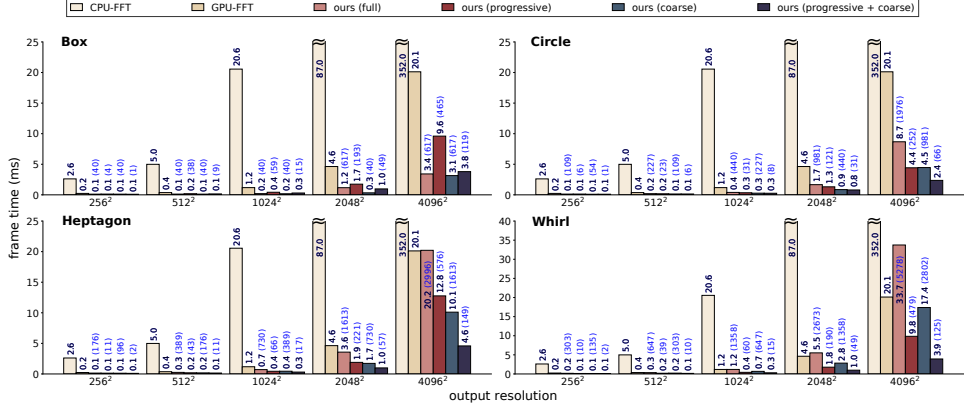


Figure 3.10: Timing comparison of our object-based FT methods to the pixel-based FT techniques. Ours include the four versions and the frame times are shown on each bar. For ours, the average numbers of evaluated quads are shown with blue color in parentheses.

4096². Unlike standard DFT techniques, our technique scales with the number of quads, where the degree of tessellation depends on the image content (Table 3.1); rectangle and circle would already have optimal analytical solutions, but we showcase them as general cases. In particular, although the rectangle aperture could be represented by a single quad, the quadtree decomposition results in a larger amount which differs at different resolutions depending on its pixel alignment to the quadtree grid and pixel differences at each resolution. Nevertheless, any slightly different shape would result in a similar amount of quads.

We initiate the finest mipmap level of the quadtree construction to the half size of the input image to avoid redundancy from anti-aliased boundaries (rasterized from vector images); note that this does not reduce the output resolution, unlike a downsampling for the pixel-based FT techniques. We also use a coarser quarter-size mipmap for a more aggressive quadtree approximation. Their visual differences are marginal; Fig. 3.9 shows the comparison of quality and performance in terms of the quadtree tessellation resolution.

We implemented our solution in OpenGL 4.5 on an Intel i7-5820K with 3.3 GHz and an NVIDIA GeForce GTX 1080 Ti graphics card. We first compare our FFT techniques with different implementations of the standard FT: the well-known FFTW [76] (CPU-FFT), and our GPU implementation of FFT (GPU-FFT); we note that our GPU-FFT is equivalent to those used in [55, 57]. Our techniques use four versions in terms of aggressive tessellation and progressive evaluation; the symmetry and culling optimizations are used in all cases. The progressive evaluation uses animated sequences, as shown in Fig. 3.3; we overlaid a circular shadow on each aperture, which moves at 5 pixels per frame. Then, we report the individual effects of our optimization techniques.

Fig. 3.10 shows the performance comparison of our techniques against CPU-FFT and GPU-FFT for monochromatic FFD. Our solutions are faster than CPU-FFT and GPU-FFT, because our methods evaluate far fewer quads than pixels present in the image. The exception being the Whirl aperture for high resolutions at the finest tessellation level.

output resolution	256 ²	512 ²	1024 ²	2048 ²	4096 ²
no optimizations (ms)	0.13	0.40	2.25	19.46	139.35
culling (ms)	0.21	0.32	1.06	5.65	31.63
symmetry (ms)	0.14	0.25	1.20	9.76	74.70
culling+symmetry (ms)	0.14	0.26	0.71	3.57	20.22

Table 3.2: Impact of optimizations on the timing (measured in ms) of the FFD for the Heptagon aperture.

output resolution	256 ²	512 ²	1024 ²	2048 ²	4096 ²
full (ms)	0.040	0.127	0.530	1.960	8.180
batch (ms)	0.019	0.038	0.114	0.430	1.710

Table 3.3: Cost comparison (measured in ms) between the full spectral scaling and our four-sample batch spectral scaling.

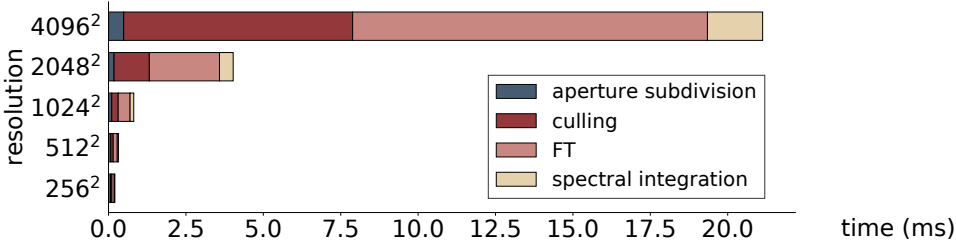


Figure 3.11: Per-stage performance breakdown of our FFD rendering for the Heptagon aperture.

Using a coarser input quadtree, our solution outperforms the competing methods while producing visually indistinguishable results (Fig. 3.9). Similarly to FFTs, our solutions also scale with the output resolution, but depend more on the input shape. Our progressive refinement significantly reduces the effective number of quads to evaluate without any quality loss; the Box aperture is an exception due to its simple shape. The number of quads are reduced roughly down to 10–20% (see the figure for the average number of the effective quads), and the speedup factor ranges from 1.3 to 4.5 for larger resolutions. The aggressive approximation with the coarser quadtree representation gains an additional speedup with marginal quality loss. When combining the progressive evaluation and aggressive approximation, our solution achieves a significant speedup in comparison to GPU-FFT (between 4 and 8.3 times faster for resolutions above 1024²). Hence, in practice, we suggest using a coarser resolution for plausible fast rendering. With our object-based approach, it is also easier to decouple the input and output resolutions than it is when using pixel-based FTs.

Table 3.2 shows the performance gain using culling and symmetry. Culling improves performance by up to 4× and symmetry up to 2×. Combining both, we achieve up to roughly a 8× speedup.

Table 3.3 shows the performance gain of the batch spectral scaling against the full spectral processing. The full processing with scaling requires 60 spectral samples while our batch scaling requires 4+15 spectral samples and indeed performs about 4 times faster.

output resolution	CPU-FrFT	GPU-NFD	Our NFD			
			Box	Circle	Heptagon	Whirl
256^2	15.98	4.87	0.12	0.11	0.14	0.16
512^2	33.01	20.94	0.14	0.23	0.36	0.49
1024^2	128.72	94.73	0.28	0.84	1.71	2.93
2048^2	533.85	—	7.88	5.43	15.78	23.42
4096^2	2133.19	—	30.88	41.30	137.16	255.72

Table 3.4: Performance (measured in ms) comparison of CPU-FrFT [93], GPU-NFD, and our NFD.

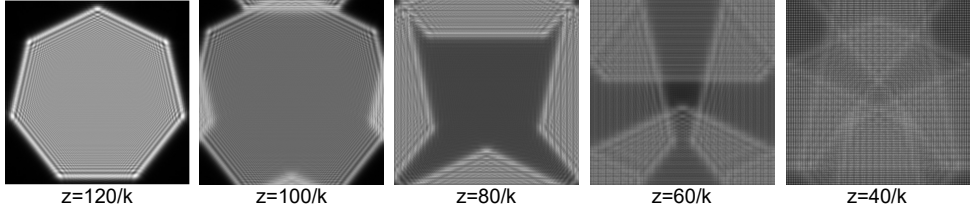


Figure 3.12: As z decreases, the discrete NFD without enough zero padding leads to interference patterns. Our NFD solution does not suffer from these artifacts.

Fig. 3.11 shows the performance breakdown of the entire rendering pipeline of FFD, which includes the aperture subdivision, culling, monochromatic FT, and spectral integration. We note the noise patterns are optional, for realistic imperfections. Their addition to the main pattern takes less than 0.5 ms at a 4096^2 resolution.

3.5.2. NEAR-FIELD DIFFRACTION

For the same data and the same machine used in Sec. 3.5.1, we compare our NFD technique against our GPU-based implementation of the reference discrete NFD (GPU-NFD) and a recent CPU-based discrete fractional FT [93] (CPU-FrFT; written in Matlab); in many cases, the NFD evaluation relies on an optimized FrFT implementation [94]. Unlike our closed-form solution of the *continuous* integrals, the *discrete* techniques may exhibit severe aliasing for near-source distances (see Fig. 3.12), which requires significant zero padding to avoid it. For GPU-NFD, we had to use $8\times$ larger resolutions than the input to completely remove aliasing in our experiments. Thus, in our experiments, GPU-NFD cannot be produced at input resolutions higher than 1024^2 (requiring 8192^2 for the evaluation) due to limitations of the texture sizes in our test hardware. This indicates an important benefit of our solution against the discrete techniques, where ours accelerates at a lower resolution and uses less memory.

Our NFD solution uses the two-pass optimization (separable computation of the tensor product), which is roughly $5\times$ faster than without the optimization. We used culling, but no symmetry. The culling used function-only sampling at a lower resolution, since it is impossible to find the analytic gradient of the NFD power spectrum. Given that the NFD depends less on high frequencies when compared to the FFD, this culling produces no artifacts.

Table 3.4 shows the comparison of the monochromatic NFD generated with our solu-

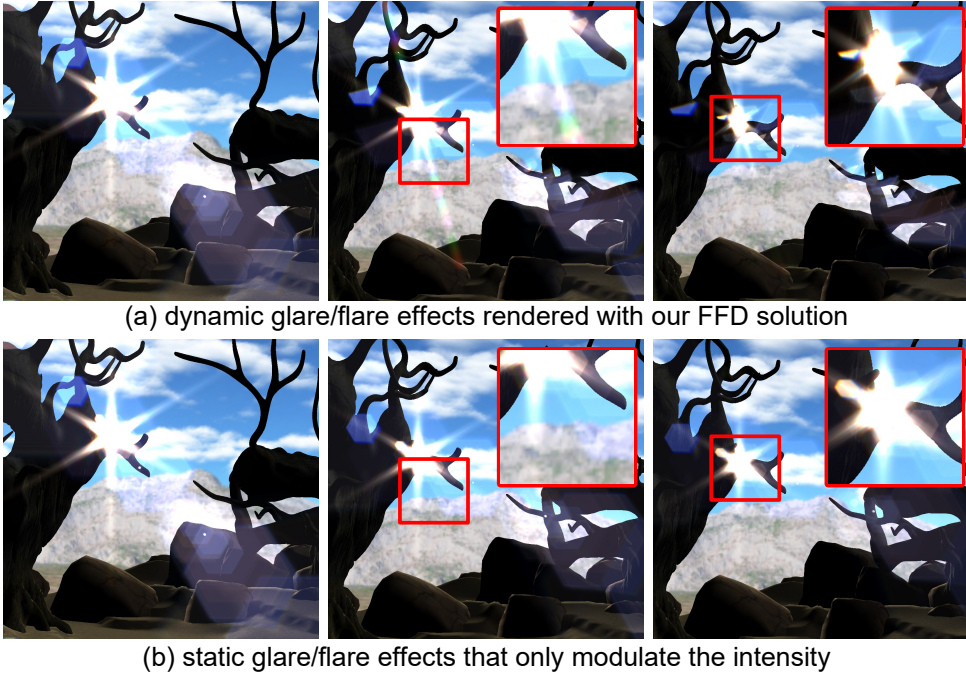


Figure 3.13: Comparison of glare/flare rendering [57] using a static glare image and our dynamic solution. Our FFD solution (a) for area-light integration reflects the varying visibility of the dynamic light source. The shape of the refraction changes based on visibility. In contrast, static glare effects (b) only modulates intensity.

tion compared to CPU-FrFT and GPU-NFD. Similarly to the FFD techniques, CPU-FrFT and GPU-NFD scale only with the image resolutions, while ours depends on the aperture shapes as well as the image resolutions. For all resolutions, ours are significantly faster than CPU-FrFT and GPU-NFD; speedup factors are $34\text{--}182\times$ for the most complex Whirl aperture. GPU-NFD is also slow due to its significant zero padding, which results in $30\text{--}42\times$ slower performance than our NFD (for the Whirl aperture). Our approach does not suffer from such problems and can be used at any resolution, which also provides more fine-grained control over performance. As expected, CPU-FrFT is the slowest variant.

As shown in Fig. 3.6, the results of the coarse quadtree are less sensitive than those of FFDs. Thus, in practice, a more aggressive downsampling can be used. Our spectral approximation using linear scaling for NFD is much faster than a precise evaluation with speedup factors in the range of $93.1/244.7$ for a resolution of $256^2/512^2$. Given the marginal quality loss, it is a practical alternative to the full evaluation. The progressive approach has a similar effect as for FFD, which is why we left this evaluation out.

3.6. APPLICATIONS

3.6.1. GLARE RENDERING

Glare rendering is a major application for our FFD and NFD techniques, which can be used to achieve more realistic results when compared to previous techniques that rely

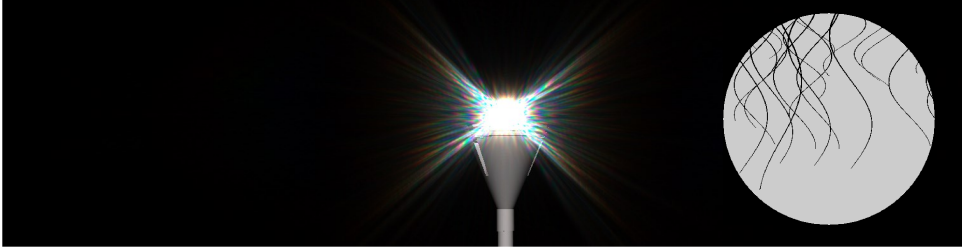


Figure 3.14: An example glare rendering of a bright light on a lamp post when using an aperture simulating a human eye with eyelashes.

on static diffraction patterns [57]. To render dynamic glare, the diffraction pattern (i.e., PSF) is applied only to the relevant parts of the scene. For an area light source, we extract the scene image around the light source, integrate the PSFs of individual light samples with varying occlusions, and blend the result with the scene image.

Fig. 3.13 shows examples rendered for an area-light source with and without dynamic diffraction. The aggressive approximations still result in plausible outcomes, which suits real-time applications well. A variation of glare rendering simulates the effect of eyelashes (Fig. 3.14), which simplifies the previous models of the human visual system [55, 56]. More extensive modeling involves additional components (e.g., iris, cornea, pupil, vitreous humor, and retina [54, 56]).

The performance of our area-light integration is roughly $2\times$ faster than that of FFT. In our experiments (81 light point samples at 1024^2 resolution), the timings of ours and FFT for monochromatic outputs are 31.9 ms and 63.4 ms, respectively. In addition, the aperture creation (9.42 ms) and spectral integration (13.5 ms) are required, which are shared with the FFT-based solution.

3.6.2. RINGING AT DYNAMIC APERTURE EDGES

Our NFD enables the simulation of an aperture pattern cropped by the housing, as is typical for a composite lens. Here, some light rays from the entrance pupil are consumed by the housing. Hence, light reaching the iris aperture is already partially culled (often dubbed as the cat's eye effect). The NFD of such patterns becomes visible in lens-flare ghosts [57]. In contrast to previous work [57, 95], our solution can handle these dynamic changes. Fig. 3.15 shows examples of dynamic diffraction with cropped apertures.

3.6.3. BLOOM/GLOW RENDERING

Bloom rendering is another application of our FFD technique, which shows the halos of light sources or their reflections. Real-time rendering typically uses simple postfiltering [96], destroying significant details. In our implementation, we create a glare pattern that incorporates dynamic lens effects (e.g., noise) and convolve it with the scene image at a low resolution using pixel intensities as weights. Thereby, we can capture the physical impact of source shapes in the diffraction patterns (Fig. 3.16).

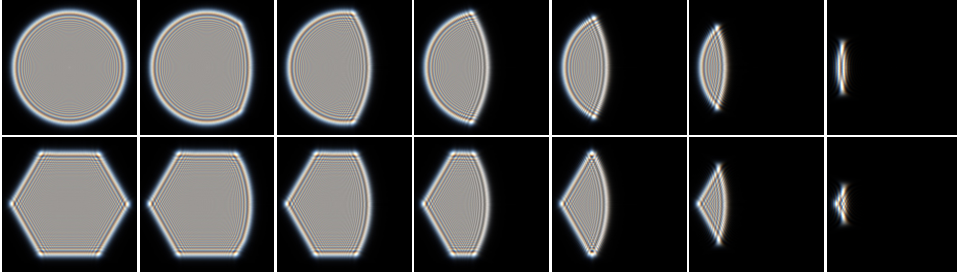


Figure 3.15: The dynamic ringing of the aperture cropped by the lens housing, which can be used for realistic lens-flare rendering.



Figure 3.16: The examples of the realistic bloom rendering generated with our FFD solution.

3.7. DISCUSSION AND LIMITATIONS

We have shown the utility of our primitive-based FT and NFD in terms of performance, quality, and flexibility in parametrization. While pixel-based FT techniques have a fixed cost, our primitive-based FT solution can employ many optimization techniques. This results in a higher performance without quality reduction in all proposed applications.

Our experiments showed that reducing the number of quads is crucial for higher performance. At present, we exploited only quads, but integrating more primitives is an interesting direction. For instance, a triangular mesh [84, 85] can spawn less primitives, but the hierarchical construction can be difficult. Investigating a full vectorial representation is an interesting direction.

In addition to our solution of introducing negative quads when extracting the list from the quadtree, we also described the merging of neighboring quads into rectangles, which reduced the number of primitives for FFD and NFD. Merging across several levels of the quadtree is an interesting direction for future work. While promising, the application of the shift property becomes more difficult.

Our approach is most efficient if larger homogeneous areas appear in the input image. Natural images might lead to a larger quad set, which eventually becomes equivalent to processing every pixel independently, making our approach unpractical. On current hardware, around 1500 quads can be handled competitively in comparison to a GPU FFT approach. This amount is largely sufficient for the presented applications involving typical camera lens shapes, especially when using progressive refinement.

4

GRADIENT-GUIDED LOCAL DISPARITY EDITING

L. Scandolo, P. Bauszat, E. Eisemann

Stereoscopic 3D technology gives visual content creators a new dimension of design when creating images and movies. While useful for conveying emotion, laying emphasis on certain parts of the scene, or guiding the viewer's attention, editing stereo content is a challenging task. Not respecting comfort zones or adding incorrect depth cues, e.g. depth inversion, leads to a poor viewing experience. In this chapter, we present a solution for editing stereoscopic content that allows an artist to impose disparity constraints and removes resulting depth conflicts using an optimization scheme. Using our approach, an artist only needs to focus on important high-level indications that are automatically made consistent with the entire scene while avoiding contradictory depth cues and respecting viewer comfort.



Figure 4.1: **Left:** Original stereographic image with disparity map inset **Middle:** An edited version, where we increased the roundness of the lion head and moved the background wall away from the viewer, presents depth-cue conflicts where the edited elements meet **Right:** Our optimization procedure preserves edits while removing inconsistencies. 🇧🇪 🇩🇪

4.1. INTRODUCTION

STEREOSCOPIC images provide the viewer with a better understanding of the geometric space in a scene. Used artistically, it can convey emotions, emphasize objects or regions, and aid in expressing story elements. To achieve this, stereo ranges are increased or compressed and relative depths adapted [97]. Nevertheless, conflicting or erroneous stereo content can result in an uncomfortable experience for viewers. In this regard, stereo editing is a delicate and often time-consuming procedure, performed by specialized artists and stereographers. Our solution supports these artists by allowing high-level definitions to set and modify stereo-related properties of parts of a scene. These indications are propagated automatically, while ensuring that the resulting stereo image pair remains plausible and can be viewed comfortably.

For a known display and observer configuration, the terms *depth* (distance to the camera), *pixel disparity* (shift of corresponding pixels in an image pair), and *vergence* (eye orientation) are linked [98]. For the sake of simplicity, we will use these terms interchangeably throughout this chapter. Although disparity is typically a function of camera parameters and the object that is observed, stereographers manipulate depth content to influence disparity. While some artists work with 2D footage only [99], we will focus on 3D productions, where disparity values can be changed by interacting with the 3D scene, i.e., changing the depth extent and position of objects.

Modifying depth directly can result in depth cue conflicts and affect the observer's interpretation of the scene, which can cause visual discomfort. For example, in Fig. 4.1 the background wall was moved away from the viewer (by increasing its disparity), while the lion head was extended in depth. These edits result in conflicts with the rest of the scene: the lion head appears to extend beyond the wall and the wall seems detached from other scene elements.

Depth relationships between objects render depth manipulations complex. Manipulating one object can induce an entire chain of operations and quickly result in a trial-and-error process. Therefore, we propose to manage disparity edits as a global process, taking all parts of a scene into account to avoid unwanted results. Our approach aims at fulfilling the artist's indications while testing for depth-cue errors. As for many artistic tools, providing fast feedback is important. This goal is achieved via an efficient opti-

mization procedure that derives suitable disparity values that are used to produce a new stereoscopic image pair.

4.2. RELATED WORK

Over the last century, stereo vision and depth perception has received much attention from the clinical and physiological perspective. A detailed explanation of the mechanisms involved in human stereo vision can be found in [100] and [101]. More recently, work has been devoted to understanding discomfort and fatigue related to distortions present in stereo image displays. Lambooi et al. [19] and Meester et al. [102] provide reviews that detail distortion effects in stereoscopic displays and their effect on viewer comfort. In particular, vergence and accommodation conflicts [103–105] are a leading cause of visual fatigue, which can be reduced by keeping depth content to a depth comfort zone. Camera parameters can be automatically adapted for this purpose in virtual scenes [106] or even real-life stereoscopic camera systems [107]. Other methods to reduce discomfort rely on post-processing [108] of the final stereo pair, introducing blur [109], and depth of field effects [110].

Research towards perceptual stereo models can also help reduce or eliminate viewer discomfort [111–113]. Such models can also be used to enhance depth effects, for example using the Cornsweet illusion [114] or adding film grain to a video [115], or to efficiently compress disparity information [116]. Templin et al. [115] and Mu et al. [117] modeled user response times for rapid disparity changes, such as video cuts, which allows artists to know when fast vergence changes will be acceptable for observers.

In the context of stereo content editing and post-process, rotoscoping [99, 118] is a widely used technique, where image elements are placed in layers at different depths. The depth of these layers can be moved and scaled, and commercial products [119–121] are available to facilitate this process. Some of these tools can detect color inconsistencies between the stereo pair images and also possible violations to the stereo vision comfort zone, but the detection and correction of depth conflicts is left to the artist. Furthermore, Wang et al. [122] provide tools to insert depth information to a 2D image via scribble-based tools and the use of an image-aware dispersion method.

Other artistic stereo editing methods focus on globally modifying the available depth range, akin to global tone-mapping used in images. Wang et al. [123] and Kellnhofer et al. [124] propose different methods to modify disparity globally in order to enhance depth perception in certain areas of an image pair or stereoscopic video. Lang et al. [125] present a method to automatically create and apply a global disparity warping that affects the complete scene but does not allow for localized editing (see Fig. 4.4). Optimizing for depth perception during motion in depth [126] and parallax motion [127] have also been explored.

Nevertheless, most of the previous approaches do not allow for user-defined local edits, which are common in movie productions, or they do not ensure consistency after an edit has been made. Our work addresses this problem. We will rely on a global optimization strategy that shares similarity with gradient-guided optimizations that have been explored in different settings, e.g., editing and filtering [128, 129], video editing [130], or image stitching [131]. Luo et al. [132] propose an automated system for stereoscopic image stitching that can preserve borders and correct perspective projection. They do

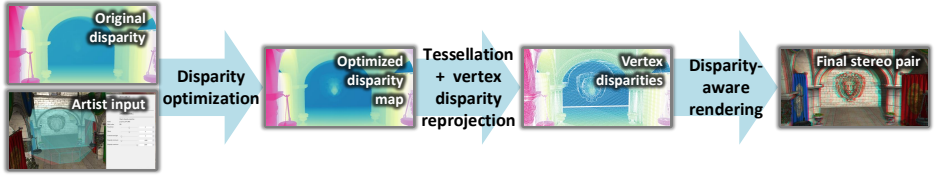


Figure 4.2: Our pipeline takes the initial disparity values and artist input to create an optimized disparity map, which is used by the vertices of the scene to allow a disparity-aware render algorithm to create a stereographic image pair.

4

so via a gradient-preserving optimization process similar to Perez et al [128], but unlike the work presented here, it targets images with no defined underlying mesh and only handles the use case of image-stitching.

4.3. DISPARITY EDITING

The goal of our proposed method is to allow an artist to edit disparity values for a given view of a 3D scene without having to consider potential conflicts. In this context, we strive for real-time performance to be able to provide instant feedback.

To explain our solution, we will first describe how we will model the tools that influence the original scene disparity (Sec. 4.3.2). In practice, this process will be linked to a *disparity map*, which, for a given view, stores in each pixel a disparity value (Sec. 4.3.1). Our algorithm will derive an optimized disparity map, integrating the artist’s constraints defined with the aforementioned tools, while avoiding depth conflicts (Sec. 4.3.3). To additionally prevent artifacts due to hidden geometry and temporal changes, we rely on a scene reprojection technique. It transfers the information from this disparity map to the 3D scene, which is then rendered to an image pair following the disparity map (Sec. 4.3.4). Fig. 4.2 showcases our proposed pipeline.

4.3.1. DISPARITY MAP

The *disparity map* stores the final pixel disparity between the left and right view as an image taken from a camera located precisely between the left and right view. This map can be derived very efficiently by rendering the scene from the middle camera and converting the depth buffer by taking the focal plane distance and the interaxial distance of the stereoscopic cameras into account [98]. We refer to this unedited disparity map as $D : \mathbb{N}_0^2 \rightarrow \mathbb{R}$.

The tools we will provide to the user will influence this disparity map. As user commands might cause conflicts or inconsistencies, a depth conflict resolution strategy will override them where necessary before performing an optimization.

The tools and conflict resolution procedures will shape a target gradient \tilde{G} that will

be linked to the optimized disparity map \bar{D} via a set of linear equations:

$$\begin{aligned}\bar{D}_{x,y} &= \bar{D}_{x+1,y} - \bar{G}x_{x,y} \\ \bar{D}_{x,y} &= \bar{D}_{x-1,y} + \bar{G}x_{x-1,y} \\ \bar{D}_{x,y} &= \bar{D}_{x,y+1} - \bar{G}y_{x,y} \\ \bar{D}_{x,y} &= \bar{D}_{x,y-1} + \bar{G}y_{x,y-1}\end{aligned}\tag{4.1}$$

These equations will be part of a larger linear system that will be solved in the least-squares sense in order to obtain \bar{D} .

For brevity and readability, we will use subscripts to refer to the sampling of these maps, i.e., D_p instead of $D(p)$. Likewise, the first and second component of \bar{G} will be noted as $\bar{G}x$ and $\bar{G}y$ respectively.

4.3.2. DISPARITY TOOLS

We will describe several editing tools, which are found in actual practice [99]. These tools act on properties of individual objects, properties relating pairs of objects or world-space points, and global parameters. We will express their effect directly in terms of constraints for the optimized disparity map or its target gradient.

Roundness Roundness refers to a change of an object's disparity range. Increasing roundness is commonly used to put emphasis on main objects or to convey emotion; in the movie *UP*, the roundness of the main character contrasted drastically with the roundness of a happy character when the latter approached his house to express the different emotional states.

Roundness R scales the disparity difference of every point of an object with respect to its center. This operation amounts to enlarging or decreasing the disparity gradient in a pixel p , if it belongs to the pixel set θ_{obj} corresponding to the manipulated object :

$$\forall p \in \theta_{obj} : \bar{G}_p = G_p \cdot R\tag{4.2}$$

Disparity anchoring Disparity anchoring means that a certain disparity value is enforced for a chosen location. This option is important to specify the overall layout of a 3D environment [98]. Usually, the artist will enforce a specific depth for certain scene elements, e.g., the main object at screen distance to minimize the vergence-accommodation conflict. In our solution, the artist chooses an offset O_d to the initial disparity. As roundness will affect the disparity as well, we include it in the computation:

$$\forall p \in \theta_{obj} : \bar{D}_p = D_m + (D_p - D_m) \cdot R + O_d\tag{4.3}$$

where D_m is the object's center point disparity.

Interface preservation Interface preservation is used to maintain the local depth contrast between objects. It is known that local depth contrast can have a global effect [114, 133]. Further, it helps separating objects clearly in space.

We allow users to specify pairs of objects for which the disparity difference should be maintained. Consequently, the pixels on the shared boundary maintain their disparity

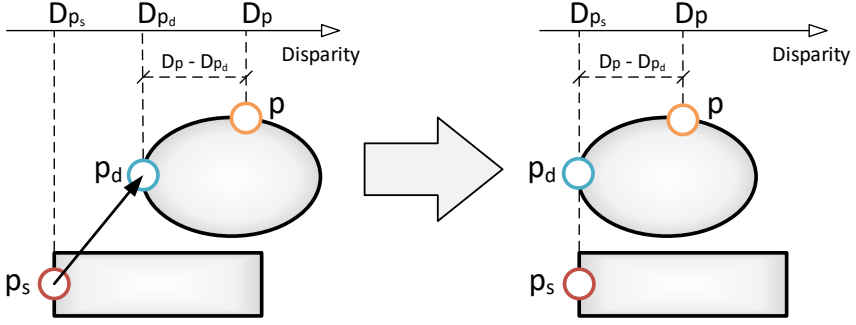


Figure 4.3: The matching points constraint matches the disparity of two points plus an offset (zero in this case) while maintaining the original disparity difference of all points in their influence set.

4

gradient ($\tilde{G}_p = G_p$). This definition can also be extended by allowing the user to draw a pixel mask to indicate where the disparity gradient should remain unaffected. This option is particularly useful for static imagery in the background.

Matching points Besides overlapping objects, a user can also couple the disparity of different elements in the scene. For example, in a view of a soccer ball flying through the air, one might want to keep the disparity between player and soccer ball constantly at the limit of the comfort zone to obtain the highest comfortable depth contrast. A more subtle application is for objects that are in contact. Fig. 4.1 shows an example, where the wall has been moved back. The attached objects become disconnected and appear to float in the air. An artist can easily connect the objects to the wall using matching points.

Specifically, a user can mark a destination point p_d to match the disparity of a source point p_s plus an optional offset O_m . This constraint affects the disparity value of a set of screen-space points θ_{mp} defined as belonging to the object indicated by p_d , or, optionally, a specified area around p_d . For all points p within θ_{mp} , the constraint attempts to maintain the original disparity difference between p and p_d , but takes as pivot point ($\tilde{D}_{p_s} + O_m$) instead of \tilde{D}_{p_d} (Fig. 4.3):

$$\forall p \in \theta_{mp} : \tilde{D}_p = (\tilde{D}_{p_s} + O_m) + (D_p - D_{p_d}) \quad (4.4)$$

4.3.3. DISPARITY MAP OPTIMIZATION

The aim of the optimization stage is to solve the sparse linear system that arises from the constraints imposed by the tools described in Sec. 4.3.2. Specifically, the optimization procedure will produce an optimized disparity map \tilde{D} which minimizes the sum of a per-pixel energy function E over all pixels:

$$\arg \min_{\tilde{D}} \sum_p E_p(\tilde{D}) \quad (4.5)$$

The energy function E is the sum of four terms which arise from the gradient constraints and the editing tools, and whose weights can be adjusted by the user:

$$E_p(\tilde{D}) = c_1 E_p^g(\tilde{D}) + c_2 E_p^a(\tilde{D}) + c_3 E_p^m(\tilde{D}) + \epsilon E_p^r(\tilde{D}). \quad (4.6)$$

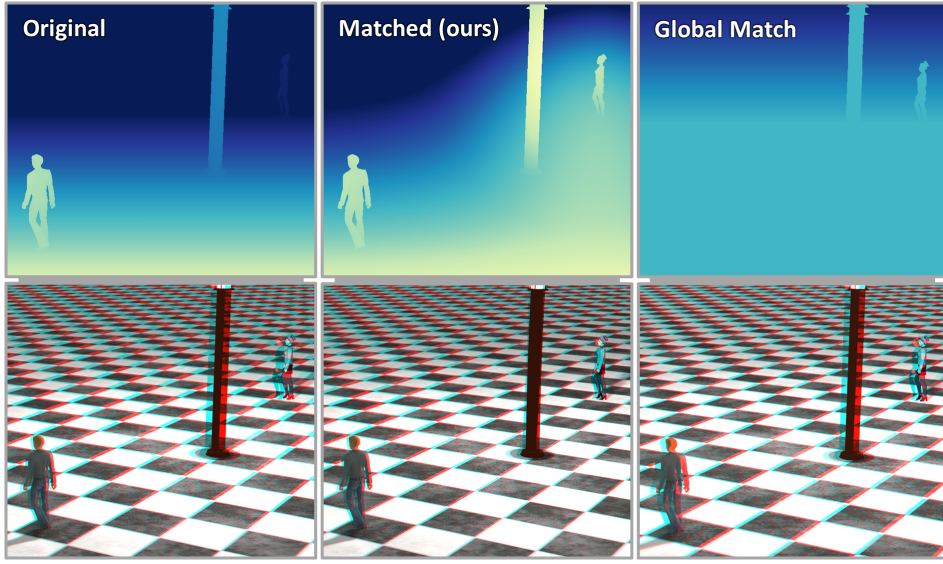



Figure 4.4: **Left:** Original disparity values and resulting stereoscopic pair for a scene with two characters. **Middle:** Result after using our method, matching the disparity of both characters. **Right:** Adjusting the disparity globally to match the character disparities, as in [125], results in loss of stereo contrast between the front character and the floor. 

The individual energy functions are the square residuals of the linear system formed by Eqs. Eq. 4.1, Eq. 4.3, Eq. 4.4, and a regularizing term:

- E^g , the gradient energy term, is the sum of the square difference of the sides of Eqs. 4.1, defined for all pixels.
- E^a , the disparity anchor term, is the square difference of Eq. 4.3. It is present for the pixels corresponding to objects for which an anchor disparity has been defined.
- E^m , the matching points term, comes from the squared difference of the sides of Eq. 4.4 for each matching point defined. Each set of matching points has a different pixel influence set θ .
- $E_p^r = |\bar{D}_p - D_p|^2$, the regularization term, ensures that there is a single solution in the absence of user defined constraints. It is defined for all pixels with a very low weight factor.

The gradient energy term ensures that the solution follows the target disparity gradient and that discontinuities or edges are correctly preserved. The target gradients are created using Eq. 4.2 for intra-object gradients. For inter-object gradients, we use the gradient of \bar{D} , which is the field we obtain by applying Eq. 4.3; this is the edited disparity map showcased in Figs. 4.1, 4.8, 4.9, 4.10, and 4.11. Finally, for areas where interface preservation is specified we revert to the original disparity map gradient.

Before the optimization procedure is performed, the linear system is inspected and modified to avoid depth inconsistencies which can potentially arise from using the tools. The most important inconsistencies are depth inversions, where for two overlapping objects, one should be behind another but their disparities imply the opposite. Such changes are reflected by differing signs of the gradient in the original and goal map gradients, which makes them easy to detect. In this case, the target gradient can be reset to the original gradient. Our framework can be expanded to deal with other conflicts in a similar fashion. For example, depth conflicts can arise at image borders for objects that are supposed to appear in front of the screen, as they are cut by the screen boundary. This case can be solved by adding an appropriate constraint to the system that penalizes pixel disparities larger than the pixel distance from the nearest vertical image border.

In general, the weight of each user-defined constraint is initialized to a default value of one and can be controlled by the user manually and intuitively since we provide instant feedback. However, some effects may only be required when viewing an object from a certain direction, or at a specified distance. Especially in image sequences, an artist may want a smooth transition between different sets of constraints when the camera or scene objects move. Our system provides the means to control the weight of a specific constraint based on different geometrical factors. A video of this use case is included in the supplementary material to the article that was the basis of this chapter.

Given that we follow a target gradient, the final optimization method is a modified Poisson reconstruction problem with added screening constraints. For large resolution images, directly solving the linear system is usually infeasible due to memory constraints, and thus iterative methods are preferable, such as Jacobi, SOR, or gradient descent methods[134]. For our implementation, we opted to use a GPU-based multi-resolution solver, since it maps well to GPU usage, avoids expensive GPU-CPU memory transfers, and is fast enough to provide real-time results. We create successively halved resolution versions of the full-resolution grid (via rendering or sampling), and solve each one with ten iterations of the Jacobi method. The initial solution for each grid level is obtained by upscaling the solution for the next coarser grid and the coarsest grid is initialized to \tilde{D} . We create the initial disparity value for pixel p_f in the finer grid level f using the optimized disparity value of pixel p_c in coarse grid level c using the formula $\tilde{D}_{p_c}^c + (\tilde{D}_{p_c}^c - \tilde{D}_{p_f}^f)$, where the disparity map superscript denotes the grid level used. This formula uses the nearest pixel at the lower resolution grid, and adds the disparity difference in \tilde{D} to ensure that discontinuities are preserved between source and destination.

4.3.4. STEREO IMAGE CREATION

In principle, one can create a stereoscopic image pair by warping a middle-view image according to the optimized disparity map [135]. Unfortunately, such image-based procedures can lead to holes due to disocclusions that reveal content not visible from the middle view. In the case where the only available information is a single segmented image plus a depth or disparity estimation, this is the best possible solution.

A more interesting case arises when we have access to the complete scene information. In this situation, we can provide a more robust solution, that relies on assigning a disparity value to each mesh vertex based on the optimized disparity map. With a per-vertex optimized disparity value, we can perform a disparity-aware render of the scene

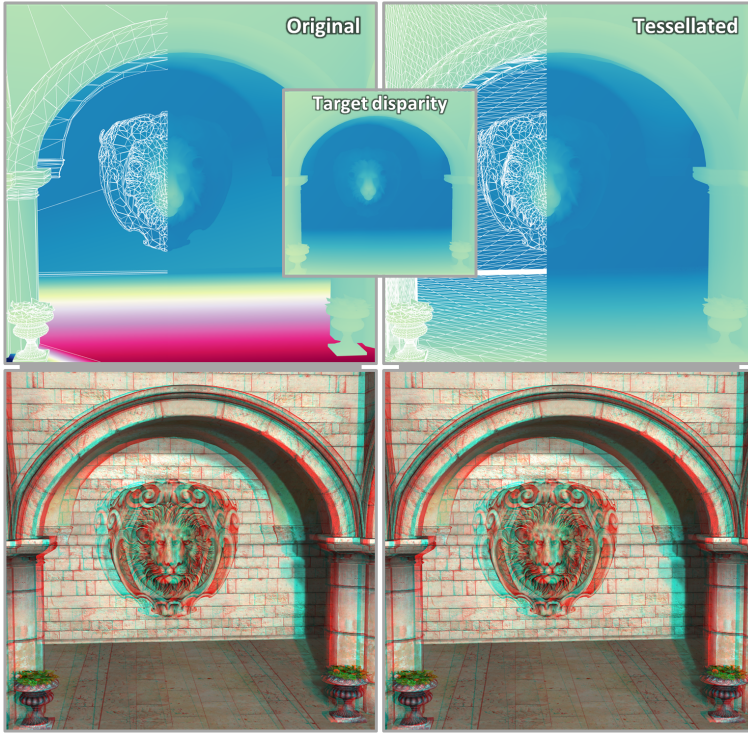


Figure 4.5: **Top inset:** Optimized screen-space disparity **Left:** Resulting disparity (top) and final stereoscopic image (bottom) when projecting screen-space disparity to original vertices fails to reproduce the target optimized disparity. **Right:** Disparity projected to tessellated geometry closely matches the optimized result. 🇺🇸🇩🇪

to obtain a hole-free stereo image pair that matches the optimized disparity map. This method is similar to the one described in [126], but since we target real-time performance, several adaptations are needed. As we will detail below, we target a much lower tessellation level and employ a different heuristic for hidden vertices, as well as a bilateral filter pass in order to improve temporal stability. We begin by describing how to perform the stereo rendering step in order to give insight into some restrictions that will apply to the disparity reprojection step.

Disparity-aware rendering In the simplest case of a single triangle and a target disparity map, we want to render a stereo image pair that renders the triangle according to the map. We do this by sampling the disparity map at each projected vertex position. We then render the triangle from the middle-view camera once for each view, and add an offset to the viewport-space position in opposite horizontal directions for each view. This added offset corresponds to half of the disparity value assigned to the vertex being processed. Consequently, disparity values are respected precisely at the vertices, and are a linear interpolation of the vertex disparities at all other locations. Therefore, as described, this method cannot correctly follow non-linear disparity gradients inside

the triangle that may be present in the target disparity map. In order to overcome this limitation, we can apply tessellation to the original triangle before projecting the disparity values to the vertices, resulting in a piecewise-linear approximation of the original disparity map.

This procedure can be applied to a complete 3D scene to create a stereo image pair that correctly handles disocclusions. We use the hardware tessellation capabilities of modern GPUs to avoid any modifications to the original mesh. However, the disparity reprojection step needs to carefully handle the cases of vertices that are occluded or fall outside the middle-view camera field of view. Fig. 4.5 shows the poor approximation of the target disparity for parts of a scene with low triangle count, such as large flat walls, and the improvement achieved when applying tessellation.

4

Disparity reprojection For visible vertices, we can directly assign a disparity value by sampling the disparity map. In order to determine vertex visibility, we compare its projected depth to the depth map created during the initial disparity map creation. Instead of relying only on the corresponding disparity map pixel to which each vertex projects, we sample a small neighborhood of pixels to increase robustness. This step also allows us to assign a disparity for vertices just outside the view frustum or close to the occlusion boundary. To integrate the result of the samples, we make use of a cross bilateral filter [136, 137], using filter weights based on screen-space position, depth difference, normal orientation, and object id [138]. In this way, samples not related to the current vertex will be discarded automatically. Furthermore, filtering values avoids sudden disparity jumps and improves temporal coherency.

If we cannot determine any valid sample for a vertex, we can still estimate its disparity by comparing D to \bar{D} at this location and applying the difference to the original disparity of the vertex.

4.3.5. IMPLEMENTATION DETAILS

We initially render the middle view via a deferred rendering pass, outputting several textures that contain properties used in the optimization pass, such as depth information, normals, object id and an initial disparity value. The optimization pass is then performed as a series of compute shader dispatches that create the target gradients, and is followed by a multi-resolution solver [139] that acts as outlined in Sec. 4.3.3, and outputs an optimized disparity texture. Using 16-bit floating point values provides sufficient precision for the optimization procedure and leads to real-time rates. Therefore, an artist can interactively edit the scene, and receive instant feedback.

The final rendering pass uses the optimized disparity map to determine vertex disparities during the tessellation evaluation stage, with the tessellation level determined by screen-space area. The disparity value is then added to the vertex viewport x coordinate in a geometry shader, which is invoked twice with multi-viewport support to efficiently generate a side-by-side image pair.

Optimizing Texture Resolution The use of cross bilateral filtering to obtain a per-vertex disparity lifts the strict correspondence in terms of resolution for the optimized disparity map and the final image. In our experiments, the optimization resolution can be much

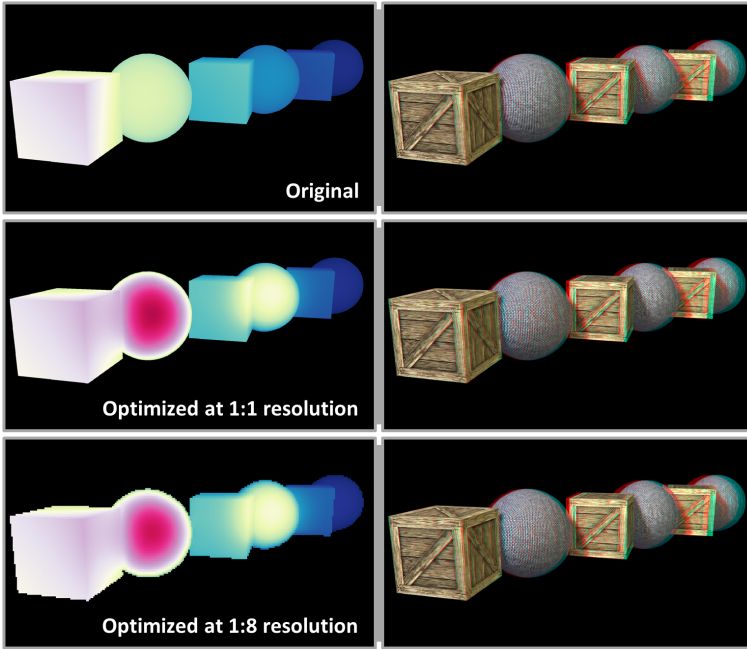


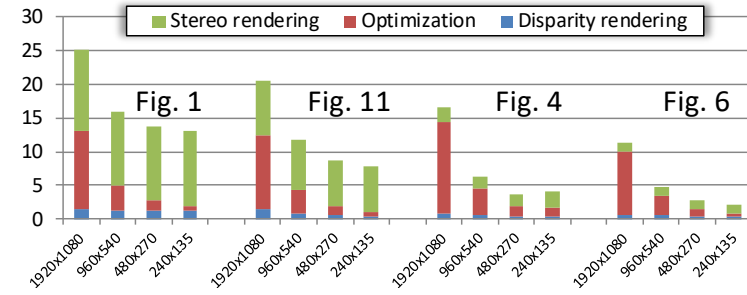
Figure 4.6: **Top:** Original disparity map and resulting image pair. **Middle:** Optimization result after increasing the roundness of the first two spheres. **Bottom:** Performing the optimization at 1:8 resolution yields very similar results. 🇧🇪 🇩🇪

lower than the final stereo image resolution without a noticeable difference in quality. Thus, we can target very large stereo image resolutions, while maintaining low memory usage and real-time performance. Fig. 4.6 illustrates the result and stereo images using a 1:1 and 1:8 scale between optimized disparity and final image pair. Moreover, this performance gain can be invested into placing the middle view differently and increasing its field-of-view projection to encompass both views to well handle the screen borders.

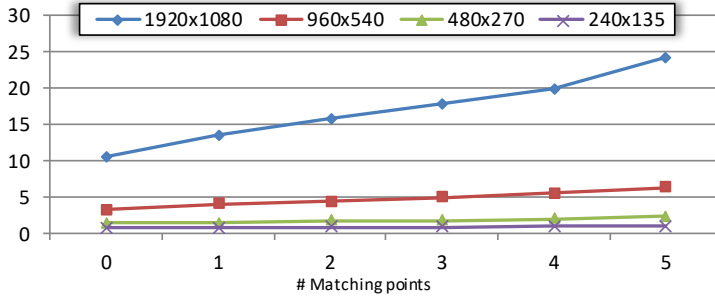
Optimizing Convergence In most optimization techniques, and ours in particular, a good initial estimate of the solution results in a faster solver convergence. During the course of a typical animation, the resulting disparity maps will be similar from one frame to the next, which implies that a previous frame is a good estimate of the next frame disparity. Using our reprojection technique, we can create a view for the current frame using the previous disparity values and use it as an initial solution for the optimization procedure.

4.4. RESULTS

We implemented our method into a tool where a user can easily edit depth content in a scene by accessing the stereoscopic properties described in Sec. 4.3. We tested our method for different scenes and with varied artistic purposes to show the range of stere-



(a) Creation times in ms for selected test scenes using different disparity map resolutions and full HD stereo image resolution.



(b) Timings in ms of the optimization stage for Fig. 4.4 using different amounts of matching points and disparity-map resolutions.

Figure 4.7: Timings for our optimization procedure

ographic modifications that can be easily performed. In the following, we will detail some examples. Figs. 4.1, 4.6, and 4.11 showcase scenes where elements are highlighted by increasing their roundness or offsetting them in depth, making them more prominent while still harmonizing with the rest of the scene elements. To illustrate the interaction on an example, in the fairy scene, the user simply clicked on both trees in the background and increased their roundness by a factor of around five. The arising conflicts, that are very visible in the image that directly integrates the indications, were fully removed automatically by our algorithm, while maintaining the overall consistency of the scene. In Fig. 4.4 we show an example of matching the disparity of two characters which are at different depths. This is a useful application in practice since it means an observer does not need to adjust their vergence when switching their gaze from one character to the other. Such quick vergence shifts are known to cause discomfort, and stereographers usually employ various methodologies to avoid them [140] or in some cases may need to redesign a scene [98]. In this case, a user created a matching point constraint between the two characters. As is visible in the result, local contrasts are maintained. This property gives the illusion of maintaining the original scene arrangement, while the disparity of the two characters is actually matched despite them being in different 3D locations in the scene.

Edits are sometimes useful to evoke emotions. We show an example of a disparity

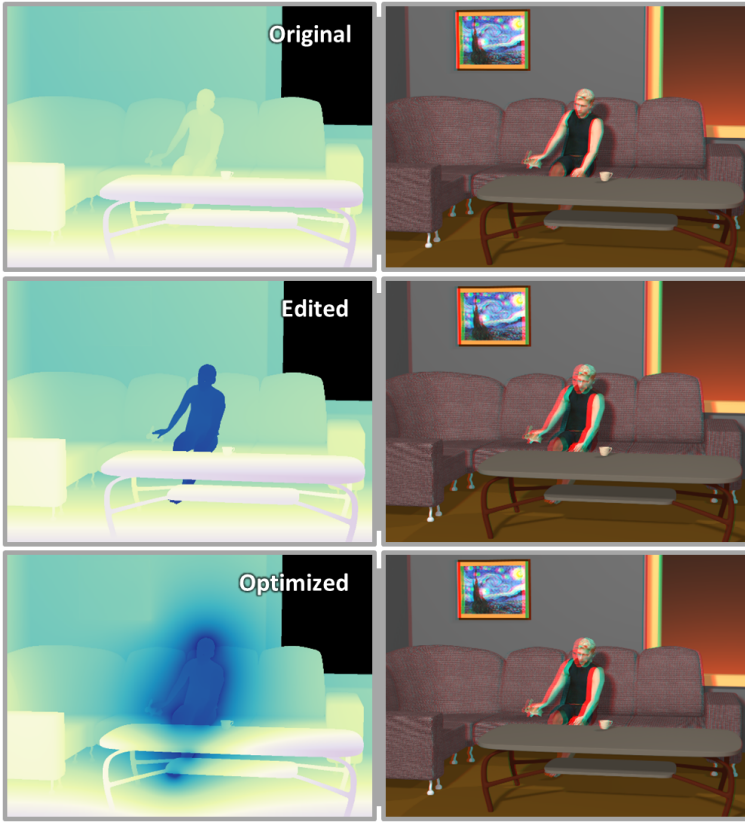


Figure 4.8: Stereoscopic manipulation can be used to showcase negative feelings as well. In this scene we convey a feeling of loneliness by flattening the sitting man and pushing him back in depth. 🐰 🐰

manipulation meant to increase the feeling of scale in Fig. 4.10, by making the cliffs seem more prominent and dangerous. In this case, the user selected parts of the mountain and increased their roundness, while anchoring them at a preferred depth. Additionally, the constraints can be dampened depending on the view of the camera. For example, during the course of an animated sequence, the weight of the user's constraints can be linked to the camera location, which allows them to vanish when the camera is rotated away from the cliffs. Another example to convey emotion is shown in Fig. 4.8, where depth edits were used to convey a feeling of loneliness in the scene by flattening the character and pushing him away from the viewer. Hereby, a feeling of distance is created. The editing operation moved the person backwards, which created a conflict with the couch, but also the bunny in his hand. The optimization process adjusts the disparity to correct for these mistakes. As shown, this solution is robust and also handles smaller objects, such as the bunny.

Finally, we also believe our method is useful beyond artistic purposes. In Fig. 4.9, we show a visualization of a human jaw, where we want to enhance the shape of the three

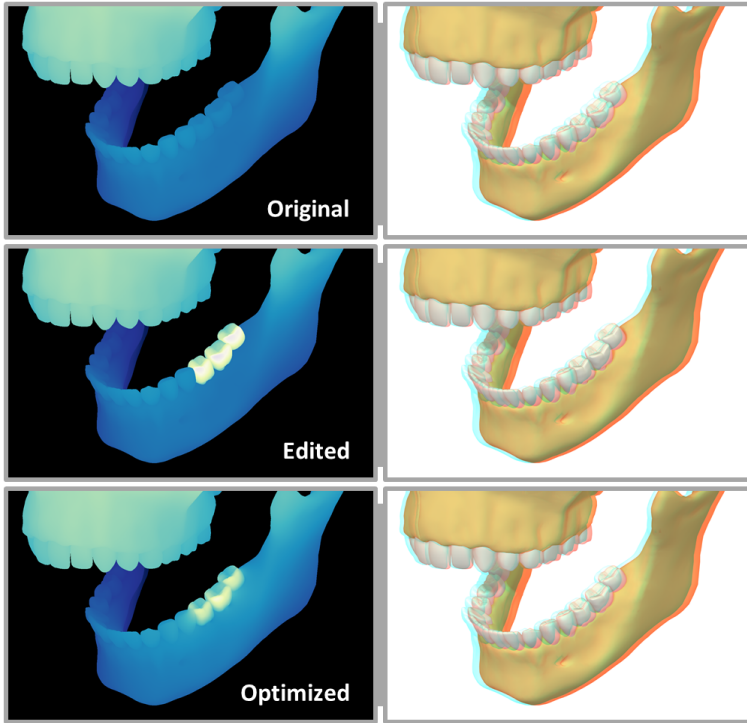


Figure 4.9: Our method can be used to aid in visualization applications. Here, the three molars are made rounder to improve the understanding of their shape. 🇧🇪 🇩🇪 🇫🇷

lower left molars. Such a solution is useful in an educational context to focus attention to important elements. The user only manipulated the roundness to increase the shape perception.

All edits in these scenes required less than a few seconds of interaction. By default object interfaces are maintained, which causes the optimization process to spread the deviation induced by the constraints over all objects. In general the user input can be very sparse, which supports our goal of simplifying interaction and having the artist focus only on important indications. Additional examples and animations are presented in the accompanying material of the article on which this chapter is based.

4.4.1. MEMORY USAGE

Memory consumption is almost entirely linked to the textures used for the optimization. It includes the deferred buffers containing the scene properties, and a series of mipmapped textures used for the multigrid optimization procedure. At full HD resolution, around 160 MB are used, which is directly linked to the disparity map resolution, i.e. at half that resolution, the memory usage is four times smaller.

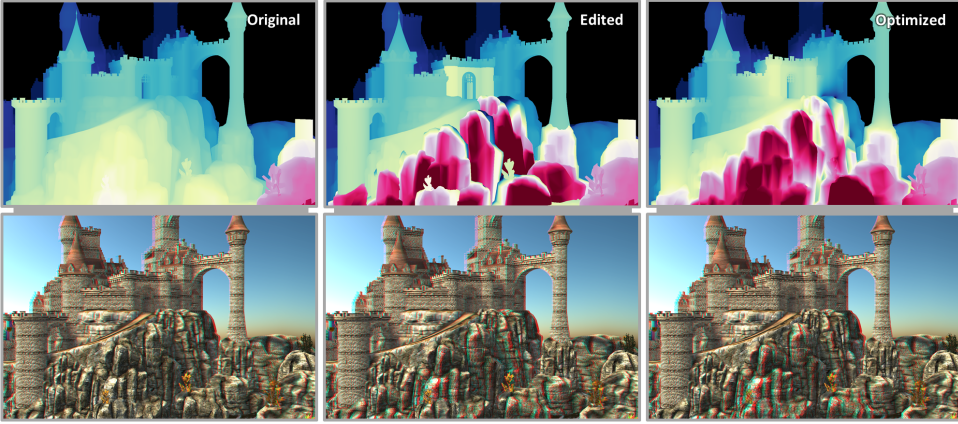


Figure 4.10: A more menacing look for the rock cliffs can be achieved by making them more prominent in depth. This is a hard case for manual editing, since many small features such as trees and bushes need to be made consistent with the edited parts of the mountain. 🇧🇪

4.4.2. TIMING

Our pipeline is implemented using C++ and OpenGL and all tests were run on an Intel i7-5820K CPU running Windows 7, with 32GB of main system memory and an NVidia Titan X GPU.

Our method employs three stages: the disparity map and scene property extraction, the optimization, and the stereo pair rendering. Three parameters affect the efficiency of these stages: the disparity map resolution R_o , the final stereo image pair resolution R_s and the geometric complexity of the scene G . The first stage is only dependent on R_o and G , the second stage depends solely on R_o and the final stage is only affected by R_s and G . The optimization stage can also be affected by the amount of matching point constraints set by the artist as they render the linear system less sparse and requires additional texture lookups. Out of these parameters, R_o is the one an artist has most control over, and can be selected to obtain a desired time/quality balance.

Fig. 4.7a showcases the timing of the three stages in some of our test scenes for different resolutions of the optimized disparity map. The optimization procedure is most heavily affected by different matching points and the scene from Fig. 4.4 shows an increased time spent on the optimization procedure. Fig. 4.7b shows the effect of different amount of matching point constraints on the optimization timing.

4.4.3. USER STUDIES

Stereo perception study We performed a small-scale user study to evaluate the effectiveness of the stereographic images created through our algorithm.

We presented the different versions of the stereo output of Fig. 4.10 to a sample of seven participants, after verifying that they were able to perceive stereoscopic content.

Fig. 4.12 showcases the view frustum and some of the parameters used for this task. All participants had normal or corrected-to-normal vision and no knowledge in the field of stereoscopic content creation or editing.

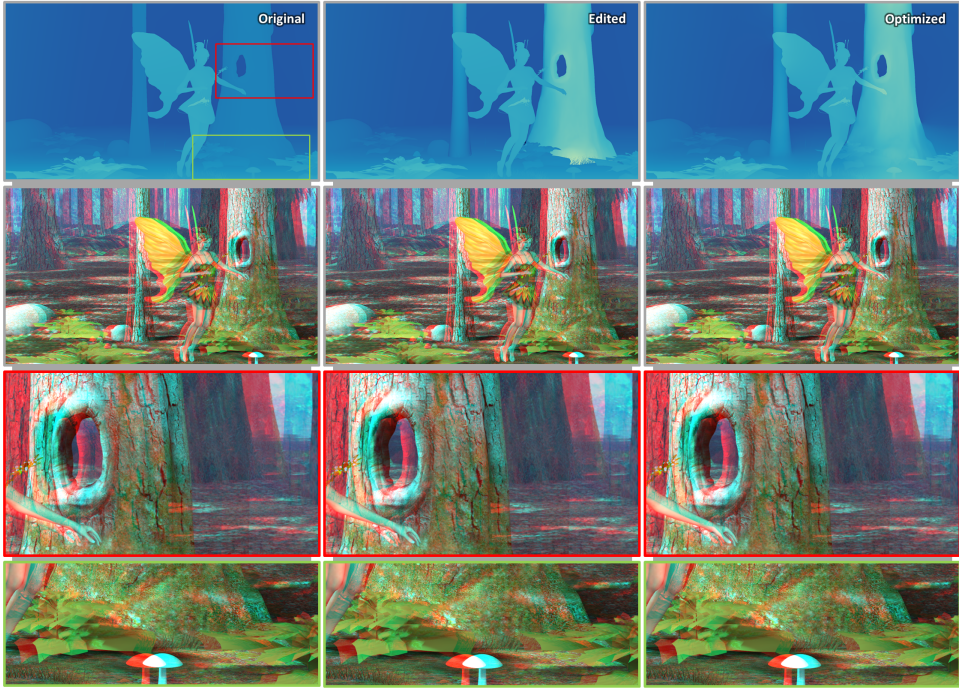



Figure 4.11: In the fairy forest test scene, the trees are made more prominent by increasing their roundness and slightly offsetting their disparity towards the viewer. The unoptimized version shows depth conflicts where the trees meet the ground and the fairy character, which are fixed in the optimized version. 

In the first part of the trial, the participants were shown the original stereoscopic image and our optimized version, and they were freely able to switch between both versions with no time constraints. They were asked to explain the difference between both images and recorded whether they were correctly able to identify the expected edition effect, namely that the cliffs look rounder and more prominent. All participants noticed that the difference between both images were constrained to the cliff area, and five out of the seven (71.5%) described the effect stating that the cliffs were better defined and there was better depth perception in their area.

In the second part, we allowed the participants to observe both our optimized version and the unoptimized edited version, and again they could switch between both images. No time constraint was imposed and they were asked to choose their preferred image. In this case, 100% of the participants expressed a preference for the optimized version, alleging either discomfort or visible artifacts when looking at the unoptimized version.

Usage study In order to test our solution against a traditional 3D modeling approach, we tasked an expert 3D modeler to create a similar modification as was created with our

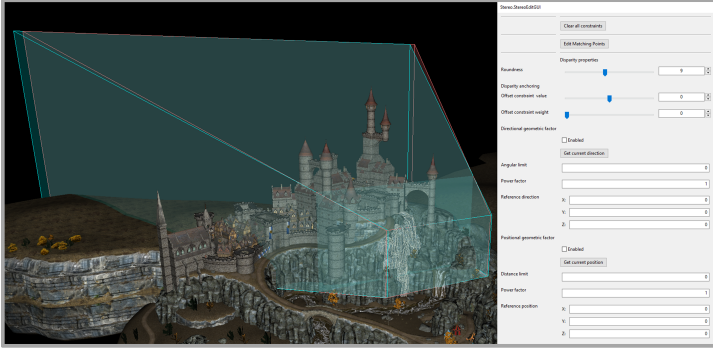


Figure 4.12: A screenshot of the editing interface used for our experiments, showing the camera frustum and some of the parameters used for our perceptual user study.

method in Fig. 4.13, namely to enhance the disparity of two rock models in the scene. The task was performed in Autodesk Maya and the artist reported that around 45 minutes of work were required. The results are shown at the bottom of Fig. 4.13. The same edition was done in under one minute with our framework. The artist mentioned difficulties to correctly maintain the geometric interfaces between the mesh parts intended to be enhanced and the rest of the scene. Additionally, when asked if the effect could be enhanced, he reported that he would have to basically start over. He also underlined that he considers the task as very challenging. The final image shows a large depth enhancement for the target regions, but as expected, the geometrical shape of the area has been significantly altered. Furthermore, some objects are missing, such as one of the trees. This highlights a key feature in our proposed solution: the ability to largely decouple disparity edition from geometrical shape, thus altering only depth perception while maintaining the geometrical shape of the scene.

4.4.4. LIMITATIONS AND FUTURE WORK

We obtain good results for realistic use cases. Nevertheless, introducing highly contradictory constraints can create unwanted results. In those circumstances, high disparity gradients can occur in the inside of originally flat objects as the least-squares optimizer spreads the constraint error. In such cases, a user can adjust the constraint weights to achieve good results. A limitation exists for thin geometry, and large disparity changes. A large disparity gradient can result in a stretched version of the object to fulfill the indicated disparity constraints. We do not explicitly tackle the problem of temporal stability for image sequences, but the produced disparity values do not show stability problems in our tests, as the optimization procedure has a well-defined behavior and provides a smooth fit to the artist's constraints. If the constraint changes are smooth, the result is typically smooth. We could envision reprojecting the optimized disparity map between consecutive frames using optical flow and rely on equalizing constraints with a small weight to avoid large changes but found it unnecessary in practice.

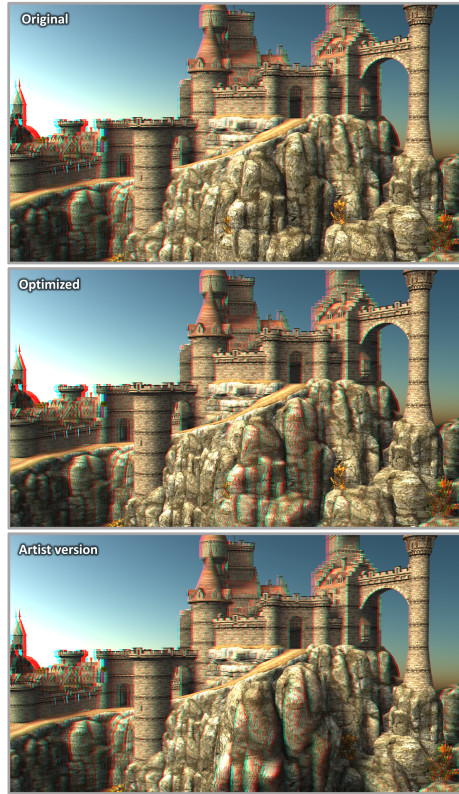


Figure 4.13: A comparison of our optimized version with an artist edited version where the geometry has been modified. Directly modifying geometry is time consuming and alters the original look of the scene, while our solution is much faster to create and preserves the geometric shape, only altering stereo perception. 🏰👁️

4.5. CONCLUSION

We have presented a method for editing stereoscopic content in 3D scenes by modifying high-level properties of the scene elements. Our approach then identifies regions where depth conflicts may arise from the user input and creates and performs an optimization procedure to obtain a conflict-free disparity map. Although image-based, coupling the map to our reprojection leads to a hole-free stereoscopic image pair. The solution runs fully on the GPU, which leads to instant feedback even for very large image resolutions. Our approach is an important addition to the toolbox of stereographers that simplifies dealing with the various conflicts. It allows the artist to focus on semantics instead of the technical underpinnings and delivers convincing results even when used by novice users.

5

CONCLUSION

IN this dissertation, we set out to demonstrate the importance of image-based representations for many computer graphics tasks. For this reason, we have striven towards providing new tools and techniques for researchers, artists, and engineers to create applications that can outperform current ones. Throughout the chapters of this work, we have explored three aspects in particular of these representations, to illustrate that many of the previous challenges related to image-based representations can be addressed algorithmically. Specifically, we focused on memory requirements, efficiency, and interactive edition of images for 3-dimensional rendering tasks.

In general, images are a discretization of the projection of a 3-dimensional object, scene, or feature onto a plane. The original information is discretized into individual pixels, providing a piece-wise constant representation of the original information. As such, care must be taken when using such representations so that the loss of information does not result in objectionable artifacts, since sub-pixel features cannot be correctly captured. Therefore, artifacts can arise, and one example of these are the presence of blocky shadows when using shadow maps of insufficient resolution. As Chapter 2 demonstrated, very large resolutions shadow maps can be encoded and efficiently accessed during the rendering process, even for large scale scenes. In contrast, accessing the entire geometrical information would not result in acceptable framerates.

Some applications require the projection of an image back into the 3-dimensional scene, as is the case exemplified in Chapter 4. In such case, the discretization of the information is an issue to consider, since the image may not encode the exact information point that is required. As shown in the aforementioned chapter, this problem can be overcome by exploring a neighborhood around the desired projected point. The neighborhood data may need to be filtered based on geometrical or other types of similarity in order to obtain a correct approximation. Another issue arises from the fact that images record only a single surface, usually closest to the camera. The lack of background information can be tackled by clever reconstruction techniques, but care must be taken to ensure that the result is sensible and correctly approximates the missing information. Using several images to capture different layers is a possibility, but the extra computing

resources need to be carefully managed in order to still provide an efficient solution.

Finally, another important feature of images, given their regular nature, is their natural hierarchical decomposition into quadtrees. Chapter 2 and Chapter 3 explored both the improvements in memory requirements and computational resources that can be achieved when hierarchically subdividing images and working with as-large-as-possible homogeneous pieces. Chapter 4 uses this decomposition to its advantage as well for efficiently optimizing the disparity map using a multi-scale approach. Current hardware, with its trend towards greater parallelization, is able to deal with the decomposition and individual processing of each part exceedingly efficiently. In general, hierarchical decomposition is an approach that should always be considered when dealing with image representations.

In Chapter 2 we tackled the problem of increasing memory requirements for creating very large shadow maps. Shadow maps reduce the problem of finding blockers for a light by discretizing the relevant blockers in a scene and storing this information in an image. Finding a potential light blocker using such a representation is far simpler than exploring the scene geometry. Nevertheless, the discretization of the blockers into texels can potentially result in artifacts, such as blocky shadows, which can be avoided by using larger shadow map resolutions. For large virtual scenes, the necessary resolutions would require far more memory than is available in commodity hardware. Current high-end GPUs sport 8 to 32 GBs of VRAM, whereas the memory requirements for creating high-quality shadows in large outdoor scenes can range in the hundreds of TBs. Our solution to this problem is an algorithm that can create a fast random-access compressed version of the original large resolution shadow map. The compression method described in Chapter 2 exploits local and non-local similarities to reduce the amount of information that needs to be stored in memory. We utilize the regularity of an image representation to group values hierarchically, which results in very efficient compression and random-access algorithms. This regularity is also exploited to find non-local similarities efficiently at any level in the hierarchy. Although we provide an algorithm specifically aimed at compressing shadow maps, the same ideas can be used for any problem where values in a regular grid can be moved within a per-element threshold.

Current graphics hardware is highly optimized for efficient access and modification of images. We took full advantage of this in Chapter 3 in order to create an efficient algorithm for rendering light diffraction patterns. A FFT is commonly used to compute such patterns. For the case of apertures, the input signal is binary, and this allows us to compute the result of the integral at the heart of the diffraction formula using a closed-form solution. Given an aperture shape defined in an image, each pixel represents a square quadrilateral in the 2D plane. We describe the closed-form solution for a single quadrilateral aperture, and given the associative nature of integral domains, we can compute the full solution as the sum of the results of each pixel. In order to reduce the amount of quads to process, we can group locally close pixels together, and repeat the process hierarchically. This process can be done exceedingly quickly using current GPUs. We describe several optimizations to this base algorithm that can drive efficiency even further. Furthermore, the same technique can be used to compute near-field and far-field diffraction patterns. Our choice of quads as primitives was inspired in the usage of image pixels as atomic components of the aperture shape, but a closed-form solution of the

diffraction integral can be found for other shapes. Therefore, finding a minimal combination of different primitives can be explored as a means of increasing efficiency further.

Image representations can also be used for facilitating editing tasks in 3-dimensional scenes. We explored this concept in Chapter 4, where we tackle the task of editing the 3-dimensional stereo effect in a stereographic image pair for a given camera view. The problem in this case resides in efficiently harmonizing editing constraints imposed by an artist without introducing any conflicting depth cues. Instead of altering the scene geometry directly to meet the artistic constraints, we rely on a disparity map as an intermediate representation. Via the use of such a map, both artist constraints and depth cue conflicts can be defined formally. We then find an optimal conflict-free disparity map that meets the artistic intent provided by the user. The optimization method is very efficient since its domain is 2-dimensional, and we can leverage the regularity of the image representation to use a well-known fast multi-scale approach, the multigrid method. The optimized map is then used to guide the rendering of the stereo image pair, such that the correct disparity values are obtained. This method provides a much more intuitive way of altering disparity values than directly modifying the scene geometry, since errors are detected and corrected automatically. In our work, we propose and test several tools for disparity edition, but they are far from a complete set. Building upon the same framework, many new tools can be created by formally defining the constraints they impose on the disparity map.

The success of regular grid structures in general is visible in other fields as well. Recently, the success of convolutional neural networks, and similar techniques, have even led to the creation of specialized hardware that can perform grid convolution and other grid operations very efficiently. Moreover, voxel grids are enjoying rising popularity in rendering, as seen by the development of fast voxel-based methods to compute global illumination [141] and highly compressed shadow and scene information [4, 142, 143]. Furthermore, the same ideas presented here for images are applicable for higher-dimensional grids, as was briefly explored with shadow cubes in Chapter 2. Perhaps in the future, specialized hardware for general and hierarchical regular grid operations will enable a new generation of algorithms that can further expand the range of effects and realism achievable in interactive applications.

In the preceding chapters, three main use cases were explored, namely memory efficiency, computational efficiency, and intuitive interaction. The specific problems tackled in this dissertation do not span the whole range of problems where intermediate image representations can be used. Nevertheless, it is the goal of this dissertation to provide practical, well-explored examples of its applicability that can not only solve a specific problem, but also provide a blueprint, or at least an intuition, for applying the same type of methods to other applications. The results presented in this dissertation show that in each case the state of the art was improved, which highlights the range of problems that can be approached in this manner.

BIBLIOGRAPHY

- [1] R. L. Cook, T. Porter, and L. Carpenter, *Distributed ray tracing*, SIGGRAPH Comput. Graph. **18**, 137 (1984).
- [2] D. S. Immel, M. F. Cohen, and D. P. Greenberg, *A radiosity method for non-diffuse environments*, SIGGRAPH Comput. Graph. **20**, 133 (1986).
- [3] J. T. Kajiya, *The rendering equation*, SIGGRAPH Comput. Graph. **20**, 143 (1986).
- [4] B. Dado, T. R. Kol, P. Bauszat, J.-M. Thiery, and E. Eisemann, *Geometry and attribute compression for voxel scenes*, Computer Graphics Forum (Proc. Eurographics) **35** (2016).
- [5] D. Dolonius, E. Sintorn, V. Kämpe, and U. Assarsson, *Compressing color data for voxelized surface geometry*, IEEE transactions on visualization and computer graphics (2017).
- [6] L. Williams, *Casting curved shadows on curved surfaces*, in *ACM Siggraph Computer Graphics*, Vol. 12 (ACM, 1978) pp. 270–274.
- [7] W. B. Pennebaker and J. L. Mitchell, *JPEG Still Image Data Compression Standard*, 1st ed. (Kluwer Academic Publishers, Norwell, MA, USA, 1992).
- [8] S. Battista, F. Casalino, and C. Lande, *Mpeg-4: a multimedia standard for the third millennium, part 1*, IEEE multimedia , 74 (1999).
- [9] T. Annen, T. Mertens, P. Bekaert, H.-P. Seidel, and J. Kautz, *Convolution shadow maps*, in *Proceedings of the 18th Eurographics conference on Rendering Techniques* (Eurographics Association, 2007) pp. 51–60.
- [10] O. Klehm, H.-P. Seidel, and E. Eisemann, *Prefiltered single scattering*, in *Proceedings of the 18th meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (ACM, 2014) pp. 71–78.
- [11] F. X. Sillion, J. R. Arvo, S. H. Westin, and D. P. Greenberg, *A global illumination solution for general reflectance distributions*, SIGGRAPH Comput. Graph. **25**, 187 (1991).
- [12] P.-P. Sloan, J. Kautz, and J. Snyder, *Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments*, ACM Trans. Graph. **21**, 527 (2002).
- [13] W. Donnelly and A. Lauritzen, *Variance shadow maps*, in *Proceedings of the 2006 symposium on Interactive 3D graphics and games* (ACM, 2006) pp. 161–165.

- [14] T. Ritschel, T. Grosch, M. H. Kim, H.-P. Seidel, C. Dachsbacher, and J. Kautz, *Imperfect Shadow Maps for Efficient Computation of Indirect Illumination*, ACM Trans. Graph. **27** (2008).
- [15] C. Peters and R. Klein, *Moment shadow mapping*, in *Proceedings of the 19th Symposium on Interactive 3D Graphics and Games* (ACM, 2015) pp. 7–14.
- [16] C. Münstermann, S. Krumpfen, R. Klein, and C. Peters, *Moment-based order-independent transparency*, *Proceedings of the ACM on Computer Graphics and Interactive Techniques* **1**, 7 (2018).
- [17] M. Stamminger and G. Drettakis, *Perspective shadow maps*, in *ACM transactions on graphics (TOG)*, Vol. 21 (ACM, 2002) pp. 557–562.
- [18] W. Engel, *Cascaded shadow maps*, *ShaderX5: Advanced Rendering Techniques* (2006).
- [19] M. Lambooi, M. Fortuin, I. Heynderickx, and W. IJsselstein, *Visual discomfort and visual fatigue of stereoscopic displays: A review*, *JIST* **53**, 30201 (2009).
- [20] L. Williams, *Casting curved shadows on curved surfaces*, *SIGGRAPH Comput. Graph.* **12**, 270 (1978).
- [21] R. Fernando, S. Fernandez, K. Bala, and D. P. Greenberg, *Adaptive shadow maps*, in *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01 (ACM, 2001) pp. 387–390.
- [22] F. Zhang, H. Sun, L. Xu, and L. K. Lun, *Parallel-split shadow maps for large-scale virtual environments*, in *Proceedings of the 2006 ACM International Conference on Virtual Reality Continuum and Its Applications* (2006) pp. 311–318.
- [23] J. Arvo and M. Hirvikorpi, *Compressed shadow maps*, *Vis. Comput.* **21**, 125 (2005).
- [24] E. Sintorn, V. Kämpe, O. Olsson, and U. Assarsson, *Compact precomputed voxelized shadows*, *ACM Trans. Graph.* **33**, 150:1 (2014).
- [25] V. Kämpe, E. Sintorn, and U. Assarsson, *Fast, memory-efficient construction of voxelized shadows*, in *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (ACM, 2015).
- [26] M. D. and B. J., *Directx 6 texture map compression*, *Game Developer* , 42 (1998).
- [27] Z. Hong, K. Iourcha, and K. Nayak, *Fixed-rate block-based image compression with inferred pixel values*, (2004), uS Patent 6,775,417.
- [28] T. Inada and M. D. McCool, *Compressed Lossless Texture Representation and Caching*, in *Graphics Hardware* (The Eurographics Association, 2006).
- [29] E. Eisemann, M. Schwarz, U. Assarsson, and M. Wimmer, *Real-Time Shadows* (A.K. Peters, 2011) p. 398.

- [30] A. Woo and P. Poulin, *Shadow Algorithms Data Miner* (A K Peter/CRC Press, 2012) p. 268.
- [31] T. Boutell, *Png (portable network graphics) specification version 1.0*, (1997).
- [32] K. Iourcha, K. Nayak, and Z. Hong, *System and method for fixed-rate block-based image compression with inferred pixel values*, (1999), uS Patent 5,956,431.
- [33] J. Ström and T. Akenine-Möller, *ipackman: High-quality, low-complexity texture compression for mobile phones*, (ACM, NY, USA, 2005) pp. 63–70.
- [34] J. Nystad, A. Lassen, A. Pomianowski, S. Ellis, and T. Olson, *Adaptive scalable texture compression*, in *HPG '12* (Eurographics Association, 2012) pp. 105–114.
- [35] A. Woo, *The shadow depth map revisited*, in *Graphics Gems III*, edited by D. Kirk (Academic Press, 1992) pp. 338–342.
- [36] D. Weiskopf and T. Ertl, *Shadow mapping based on dual depth layers*, Eurographics 2003 Short Papers (2003).
- [37] T. Ritschel, T. Grosch, J. Kautz, and S. Müeller, *Interactive illumination with coherent shadow maps*, in *Proceedings of the 18th Eurographics Conference on Rendering Techniques*, EGSR'07 (Eurographics Association, 2007) pp. 61–72.
- [38] S. Laine and T. Karras, *Efficient Sparse Voxel Octrees – Analysis, Extensions, and Implementation*, NVIDIA Technical Report NVR-2010-001 (NVIDIA Corporation, 2010).
- [39] V. Kämpe, E. Sintorn, and U. Assarsson, *High resolution sparse voxel dags*, ACM Transactions on Graphics **32** (2013), SIGGRAPH 2013.
- [40] J. R. Woodwark, *Compressed quad trees*, The Computer Journal **27**, 225 (1984).
- [41] H. Samet, *Data structures for quadtree approximation and compression*, Commun. ACM **28**, 973 (1985).
- [42] S. Lefebvre and H. Hoppe, *Compressed random-access trees for spatially coherent data*, in *Proceedings of the 18th Eurographics Conference on Rendering Techniques*, EGSR'07 (Eurographics Association, 2007) pp. 339–349.
- [43] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression* (Kluwer Academic Publishers, 1991).
- [44] N. Chaddha, P. A. Chou, and R. M. Gray, *Constrained and recursive hierarchical table-lookup vector quantization*. in *Data Compression Conference* (IEEE Computer Society, 1996) pp. 220–229.
- [45] M. Kraus and T. Ertl, *Adaptive texture maps*, in *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, HWWS '02 (Eurographics Association, 2002) pp. 7–15.

- [46] S. G. Mallat, *A theory for multiresolution signal decomposition: the wavelet representation*, IEEE Transactions on Pattern Analysis and Machine Intelligence **11**, 674 (1989).
- [47] H. Samet, *The quadtree and related hierarchical data structures*, ACM Comput. Surv. **16**, 187 (1984).
- [48] C. Everitt, *Interactive order-independent transparency*, (2001).
- [49] L. Bavoil, S. P. Callahan, A. Lefohn, J. a. L. D. Comba, and C. T. Silva, *Multi-fragment effects on the gpu using the k-buffer*, in *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*, I3D '07 (ACM, New York, NY, USA, 2007) pp. 97–104.
- [50] P. K. Agarwal and S. Suri, *Surface approximation and geometric partitions*, in *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '94 (Society for Industrial and Applied Mathematics, 1994) pp. 24–33.
- [51] W. T. Reeves, D. H. Salesin, and R. L. Cook, *Rendering antialiased shadows with depth maps*, ACM Siggraph Computer Graphics **21**, 283 (1987).
- [52] L. Scandolo, P. Bauszat, and E. Eisemann, *Compressed multiresolution hierarchies for high-quality precomputed shadows*, Computer Graphics Forum (Proc. EG) **35** (2016).
- [53] N. Andryscio and X. Tricoche, *Matrix trees*, Computer Graphics Forum **29**, 963 (2010).
- [54] G. Spencer, P. Shirley, K. Zimmerman, and D. P. Greenberg, *Physically-based glare effects for digital images*, in *Proc. ACM SIGGRAPH* (ACM, 1995) pp. 325–334.
- [55] M. Kakimoto, K. Matsuoka, T. Nishita, T. Naemura, and H. Harashima, *Glare generation based on wave optics*, in *Proc. Pacific Graphics* (2004) pp. 133–140.
- [56] T. Ritschel, T. Grosch, and H.-P. Seidel, *Approximating dynamic global illumination in image space*, in *Proc. of the 2009 symposium on Interactive 3D graphics and games* (2009) pp. 75–82.
- [57] M. Hullin, E. Eisemann, H.-P. Seidel, and S. Lee, *Physically-based real-time lens flare rendering*, ACM Trans. Graph. **30**, 108:1 (2011).
- [58] A. Yoshida, M. Ihrke, R. Mantiuk, and H.-P. Seidel, *Brightness of the glare illusion*, in *Proc. Symp. Applied Perception in Graphics and Visualization* (2008) pp. 83–90.
- [59] J. Peatross and M. Ware, *Physics of Light and Optics* (Bringham Young University, 2015).
- [60] B. K. Yap and S. D. Fantone, *Application of a sunburst aperture to diffraction suppression*, JOSA **64**, 978 (1974).

- [61] S.-W. Lee and R. Mittra, *Fourier transform of a polygonal shape function and its application in electromagnetics*, IEEE Trans. Ant. and Prop. **31**, 99 (1983).
- [62] K. McInturff and P. S. Simon, *The Fourier transform of linearly varying functions with polygonal support*, IEEE Trans. Ant. and Prop. **39**, 1441 (1991).
- [63] E. Nakamae, K. Kaneda, T. Okamoto, and T. Nishita, *A lighting model aiming at drive simulators*, ACM Trans. Graphics **24**, 395 (1990).
- [64] P. Rokita, *A model for rendering high intensity lights*, Computers & graphics **17**, 431 (1993).
- [65] J. Stam, *Diffraction shaders*, in *Proc. ACM SIGGRAPH* (ACM, 1999) pp. 101–110.
- [66] C. Lindsay and E. Agu, *Physically-based real-time diffraction using spherical harmonics*, Advances in Vis. Comp. , 505 (2006).
- [67] D. S. J. Dhillon and A. Ghosh, *Efficient surface diffraction renderings with Chebyshev approximations*, in *SIGGRAPH ASIA Technical Briefs* (ACM, 2016) p. 7.
- [68] T. Cuypers, T. Haber, P. Bekaert, S. B. Oh, and R. Raskar, *Reflectance model for diffraction*, ACM Trans. Graphics **31**, 122 (2012).
- [69] S. Werner, Z. Velinov, W. Jakob, and M. B. Hullin, *Scratch iridescence: Wave-optical rendering of diffractive surface structure*, ACM Trans. Graphics **36**, 220:1 (2017).
- [70] Z. Dong, B. Walter, S. Marschner, and D. P. Greenberg, *Predicting appearance from measured microgeometry of metal surfaces*, ACM Trans. Graphics **35**, 9 (2015).
- [71] L. Belcour and P. Barla, *A practical extension to microfacet theory for the modeling of varying iridescence*, ACM Trans. Graphics **36**, 65 (2017).
- [72] A. Toisoul and A. Ghosh, *Practical acquisition and rendering of diffraction effects in surface reflectance*, ACM Trans. Graphics **36**, 166 (2017).
- [73] N. Holzschuch and R. Pacanowski, *A two-scale microfacet reflectance model combining reflection and diffraction*, ACM Trans. Graphics **36**, 66 (2017).
- [74] C. Oat, *A steerable streak filter*, in *Shader X3*, edited by W. Engel (Charles River Media, 2004) pp. 341–348.
- [75] J. W. Cooley and J. W. Tukey, *An algorithm for the machine calculation of complex Fourier series*, Mathematics of computation **19**, 297 (1965).
- [76] M. Frigo and S. G. Johnson, *FFTW: An adaptive software architecture for the FFT*, in *Proc. Acoustics, Speech and Signal Processing*, Vol. 3 (1998) pp. 1381–1384.
- [77] K. Moreland and E. Angel, *The FFT on a GPU*, in *Proc. Graphics Hardware* (Eurographics Association, 2003) pp. 112–119.

- [78] N. K. Govindaraju, B. Lloyd, Y. Dotsenko, B. Smith, and J. Manferdelli, *High performance discrete Fourier transforms on graphics processors*, in *Proc. Supercomputing* (2008) p. 2.
- [79] F. Franchetti, M. Puschel, Y. Voronenko, S. Chellappa, and J. M. Moura, *Discrete Fourier transform on multicore*, *IEEE Signal Processing Magazine* **26** (2009).
- [80] H. Hassanieh, P. Indyk, D. Katabi, and E. Price, *Simple and practical algorithm for sparse Fourier transform*, in *Proc. ACM-SIAM Symp. Discrete Algorithms* (2012) pp. 1183–1194.
- [81] E. Sorets, *Fast Fourier transforms of piecewise constant functions*, *Journal of Computational Physics* **116**, 369 (1995).
- [82] G.-X. Fan and Q. H. Liu, *Fast Fourier transform for discontinuous functions*, *IEEE Trans. Ant. and Prop.* **52**, 461 (2004).
- [83] Y. Liu, Z. Nie, and Q. H. Liu, *DIFFT: A fast and accurate algorithm for Fourier transform integrals of discontinuous functions*, *IEEE Microwave and Wireless Components Letters* **18**, 716 (2008).
- [84] B. Houshmand, W. C. Chew, and S.-W. Lee, *Fourier transform of a linear distribution with triangular support and its applications in electromagnetics*, *IEEE Trans. Ant. and Prop.* **39**, 252 (1991).
- [85] Y.-H. Liu, Q. Liu, Z.-P. Nie, and Z.-Q. Zhao, *Discontinuous fast Fourier transform with triangle mesh for two-dimensional discontinuous functions*, *J. Electromag. Waves and Appl.* **25**, 1045 (2011).
- [86] A. Lucat, R. Hegedus, and R. Pacanowski, *Diffraction prediction in HDR measurements*, in *Proc. Eurographics Workshop on Material Appearance Modeling* (2017).
- [87] S. Lee and S. Lee, *Interactive Additive Diffraction Synthesis*, in *EG 2016 - Posters*, edited by L. G. Magalhaes and R. Mantiuk (The Eurographics Association, 2016).
- [88] M. Born and E. Wolf, *Principles of optics: electromagnetic theory of propagation, interference and diffraction of light* (Elsevier, 2013).
- [89] C. Wyman, P.-P. Sloan, and P. Shirley, *Simple analytic approximations to the CIE XYZ color matching functions*, *Journal of Computer Graphics Techniques* **2**, 1 (2013).
- [90] M. Hollander, T. Ritschel, E. Eisemann, and T. Boubekeur, *ManyLoDs: Parallel many-view level-of-detail selection for real-time global illumination*, *Computer Graphics Forum* **30**, 1233 (2011).
- [91] E. C. Kintner, *Edge-ringing and Fresnel diffraction*, *Optica Acta: International Journal of Optics* **22**, 235 (1975).

- [92] K. D. Mielenz, *Algorithms for Fresnel diffraction at rectangular and circular apertures*, J. Research of the National Institute of Standards and Technology **103**, 497 (1998).
- [93] R. Tao, G. Liang, and X.-H. Zhao, *An efficient FPGA-based implementation of fractional Fourier transform algorithm*, J. Signal Proc. Systems **60**, 47 (2010).
- [94] H. M. Ozaktas, Z. Zalevsky, and M. A. Kutay, *The fractional Fourier transform with applications in optics and signal processing* (Wiley, 2001).
- [95] S. Lee and E. Eisemann, *Practical real-time lens-flare rendering*, Computer Graphics Forum **32**, 1 (2013).
- [96] G. James and J. O'Rorke, *Real-time glow*, in *GPU Gems*, edited by R. Fernando (Addison Wesley Professional, 2004) pp. 343–362.
- [97] S. Gateau, R. Neuman, and M. Salvati, *Siggraph 2011 stereoscopy course*, (2011).
- [98] B. Mendiburu, *3D movie making: stereoscopic digital cinema from script to screen* (CRC Press, 2012).
- [99] A. Smolic, P. Kauff, S. Knorr, A. Hornung, M. Kunter, M. Muller, and M. Lang, *Three-dimensional video postproduction and processing*, Proc. of the IEEE **99**, 607 (2011).
- [100] D. M. Kent, *Foundations of binocular vision: A clinical perspective*. (2001).
- [101] I. P. Howard, *Perceiving in depth, volume 1: basic mechanisms* (Oxford University Press, 2012).
- [102] L. M. Meesters, W. A. IJsselstein, and P. J. Seuntiëns, *A survey of perceptual evaluations and requirements of three-dimensional tv*, IEEE Trans. Circuits Syst. **14**, 381 (2004).
- [103] P. Burt and B. Julesz, *A disparity gradient limit for binocular fusion*, Science **208**, 615 (1980).
- [104] D. M. Hoffman, A. R. Girshick, K. Akeley, and M. S. Banks, *Vergence–accommodation conflicts hinder visual performance and cause visual fatigue*, Journal of vision **8**, 33 (2008).
- [105] T. Blum, M. Wiecezorek, A. Aichert, R. Tibrewal, and N. Navab, *The effect of out-of-focus blur on visual discomfort when using stereo displays*, in ISMAR (IEEE, 2010) pp. 13–17.
- [106] T. Oskam, A. Hornung, H. Bowles, K. Mitchell, and M. H. Gross, *Oscam-optimized stereoscopic camera control for interactive 3d*. ACM Trans. Graph. **30**, 189 (2011).
- [107] S. Heinzle, P. Greisen, D. Gallup, C. Chen, D. Saner, A. Smolic, A. Burg, W. Matusik, and M. Gross, *Computational stereo camera system with programmable control loop*, in *ACM Trans. Graph.*, Vol. 30 (2011) p. 94.

- [108] T. Kawai, T. Shibata, T. Inoue, Y. Sakaguchi, K. Okabe, and Y. Kuno, *Development of software for editing stereoscopic 3-d movies*, in *Proc. of SPIE*, Vol. 4660 (2002) pp. 58–65.
- [109] A. T. Duchowski, D. H. House, J. Gestring, R. I. Wang, K. Krejtz, I. Krejtz, R. Mantiuk, and B. Bazyluk, *Reducing visual discomfort of 3d stereoscopic displays with gaze-contingent depth-of-field*, in *ACM SAP* (2014) pp. 39–46.
- [110] K. Carnegie and T. Rhee, *Reducing visual discomfort with hmds using dynamic depth of field*, *IEEE Comput. Graph. Appl.* **35**, 34 (2015).
- [111] P. Didyk, T. Ritschel, E. Eisemann, K. Myszkowski, and H.-P. Seidel, *A perceptual model for disparity*, in *ACM Trans. Graph.*, Vol. 30 (2011) p. 96.
- [112] P. Didyk, T. Ritschel, E. Eisemann, K. Myszkowski, H.-P. Seidel, and W. Matusik, *A luminance-contrast-aware disparity model and applications*, *ACM Trans. Graph.* **31**, 184 (2012).
- [113] S. Du, B. Masia, S. Hu, and D. Gutierrez, *A metric of visual comfort for stereoscopic motion*, *ACM TOG* **32**, 222 (2013).
- [114] P. Didyk, T. Ritschel, E. Eisemann, K. Myszkowski, and H.-P. Seidel, *Apparent stereo: The cornsweet illusion can enhance perceived depth*, in *IS&T/SPIE Electronic Imaging* (2012).
- [115] K. Templin, P. Didyk, K. Myszkowski, and H.-P. Seidel, *Perceptually-motivated stereoscopic film grain*, in *CGF*, Vol. 33 (2014) pp. 349–358.
- [116] D. Pająk, R. Herzog, R. Mantiuk, P. Didyk, E. Eisemann, K. Myszkowski, and K. Pulli, *Perceptual depth compression for stereo applications*, in *CGF*, Vol. 33 (2014) pp. 195–204.
- [117] T.-J. Mu, J.-J. Sun, R. R. Martin, and S.-M. Hu, *A response time model for abrupt changes in binocular disparity*, *The Visual Computer* **31**, 675 (2015).
- [118] K.-Y. Lee, C.-D. Chung, and Y.-Y. Chuang, *Scene warping: Layer-based stereoscopic image resizing*, in *CVPR* (IEEE, 2012) pp. 49–56.
- [119] *Pftrack*, <http://www.thepixelfarm.co.uk/pftrack>.
- [120] *Mistika*, <http://www.sgo.es/mistika-ultima>.
- [121] *Ocula*, <http://www.foundry.com/products/ocula>.
- [122] O. Wang, M. Lang, M. Frei, A. Hornung, A. Smolic, and M. Gross, *Stereobrush: interactive 2d to 3d conversion using discontinuous warps*, in *Proc. SBIM* (ACM, 2011) pp. 47–54.
- [123] M. Wang, X.-J. Zhang, J.-B. Liang, S.-H. Zhang, and R. R. Martin, *Comfort-driven disparity adjustment for stereoscopic video*, *Computational Visual Media* **2**, 3 (2016).

- [124] P. Kellnhofer, P. Didyk, K. Myszkowski, M. M. Hefeeda, H.-P. Seidel, and W. Matusik, *Gazestereo3d: seamless disparity manipulations*, ACM Trans. Graph. **35**, 68 (2016).
- [125] M. Lang, A. Hornung, O. Wang, S. Poulakos, A. Smolic, and M. Gross, *Nonlinear disparity mapping for stereoscopic 3d*, ACM Trans. Graph. **29**, 75 (2010).
- [126] P. Kellnhofer, T. Ritschel, K. Myszkowski, and H.-P. Seidel, *Optimizing disparity for motion in depth*, in *CGF*, Vol. 32 (2013) pp. 143–152.
- [127] P. Kellnhofer, P. Didyk, T. Ritschel, B. Masia, K. Myszkowski, and H.-P. Seidel, *Motion parallax in stereo 3d: model and applications*, ACM Trans. Graph. **35**, 176 (2016).
- [128] P. Pérez, M. Gangnet, and A. Blake, *Poisson image editing*, ACM Trans. Graph. **22**, 313 (2003).
- [129] P. Bhat, C. L. Zitnick, M. Cohen, and B. Curless, *Gradientshop: A gradient-domain optimization framework for image and video filtering*, ACM Trans. Graph. **29**, 10 (2010).
- [130] S. Fleishman, D. Cohen-Or, I. Drori, T. Leyvand, and H. Yeshurun, *Video operations in the gradient domain*, Tel-Aviv Univ, Tech. Rep (2004).
- [131] A. Levin, A. Zomet, S. Peleg, and Y. Weiss, *Seamless image stitching in the gradient domain*, Computer Vision-ECCV 2004 , 377 (2004).
- [132] S.-J. Luo, I. Shen, B.-Y. Chen, W.-H. Cheng, Y.-Y. Chuang, *et al.*, *Perspective-aware warping for seamless stereoscopic image cloning*, ACM Trans. Graph. **31**, 182 (2012).
- [133] S. M. Anstis, I. P. Howard, and B. Rogers, *A Craik-O'Brien-Cornsweet illusion for visual depth*, Vision Research **18** (1978).
- [134] C. L. Lawson and R. J. Hanson, *Solving least squares problems*, Vol. 15 (Siam, 1995).
- [135] P. Didyk, T. Ritschel, E. Eisemann, K. Myszkowski, and H.-P. Seidel, *Adaptive image-space stereo view synthesis*. in *VMV* (2010) pp. 299–306.
- [136] E. Eisemann and F. Durand, *Flash photography enhancement via intrinsic relighting*, ACM Transactions on Graphics **23**, 673 (2004).
- [137] G. Petschnigg, R. Szeliski, M. Agrawala, M. Cohen, H. Hoppe, and K. Toyama, *Digital photography with flash and no-flash image pairs*, ACM Trans. Graph. **23**, 664 (2004).
- [138] C. Tomasi and R. Manduchi, *Bilateral filtering for gray and color images*, in *Proc. of the International Conference on Computer Vision* (1998) pp. 839–846.
- [139] J. Bolz, I. Farmer, E. Grinspun, and P. Schröder, *Sparse matrix solvers on the gpu: conjugate gradients and multigrid*, ACM Trans. Graph. **22**, 917 (2003).

- [140] K. Templin, P. Didyk, K. Myszkowski, M. M. Hefeeda, H.-P. Seidel, and W. Matusik, *Modeling and optimizing eye vergence response to stereoscopic cuts*, ACM Trans. Graph. **33**, 145 (2014).
- [141] C. Crassin, F. Neyret, M. Sainz, S. Green, and E. Eisemann, *Interactive indirect illumination using voxel cone tracing*, in *Computer Graphics Forum*, Vol. 30 (Wiley Online Library, 2011) pp. 1921–1930.
- [142] E. Sintorn, V. Kämpe, O. Olsson, and U. Assarsson, *Compact precomputed voxelized shadows*, ACM Transactions on Graphics (TOG) **33**, 150 (2014).
- [143] V. Kämpe, S. Rasmuson, M. Billeter, E. Sintorn, and U. Assarsson, *Exploiting coherence in time-varying voxel data*, in *Proceedings of the 20th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (ACM, 2016) pp. 15–21.

EPILOGUE

Computer graphics is an amazingly interesting field where math, physics, computer engineering, and other disciplines come together. As such, there is a feeling of constant marvel at the new techniques and methods that are published every year. The great community behind these works are thus always enthusiastic about what can be achieved next, since they know there is still much more to discover.

It is my sincere hope that you, the reader, will have learned something new by the time this book is put back into its place in your bookshelf, or safely archived in your computer. Hopefully the work presented in the preceding pages will spark new ideas, encourage some further reading, or at least trigger the thought 'oh, that's clever, maybe I should try something similar'. And, after all, that is what research is all about.

ACKNOWLEDGEMENTS

A years-long endeavor such as a PhD can never be accomplished without the invaluable help of family, friends and colleagues.

I was very lucky to get a position at the Computer Graphics and Visualization group in TU Delft under the supervision of Prof. Elmar Eisemann. His easy-going attitude combined with his large knowledge on all topics related to my work meant any discussions we had about my work were always very productive. For a large part of my time here I was also lucky enough to work under the supervision of Dr. Pablo Bauszat and Dr. Sungkil Lee. Their technical and theoretical knowledge, but most importantly their never-ending imagination and constant stream of ideas meant we would always find new ways to tackle problems.

I was very lucky to be involved in some projects outside my main line of research. For that I mostly have Dr. Christopher Brandt and Dr. Klaus Hildebrandt to thank, who approached me to work on applications of their amazing research line on geometry modeling. Working with Christopher was always a delight, because of his quick wits, great work ethic and deep scientific knowledge.

Of course, everyone in the graphics group was, to different degrees (but always positively), influential. So, to Changgong, Thomas(x2), Renata, Noeska, Jingtang, Philip, Nicola, Anna, Jean Marc, Bert, Michael, Timothy, Ruud, Bart, Niels, Nestor, Leo, Ahmad, Faizan, Peiteng, Baran, Markus, Julian, Tom, Victor, Annemieke, Jerry, Xuejiao, Ricardo, a big thanks for all the fun I had during my time with you. I would like to especially mention the great secretaries at our group: Stefanie, Sandra and Marloes. They went above and beyond trying to help me and everyone else with all the administrative hassle, and, most importantly for some of us, with setting up our life as expats. A special mention to Dr. Timothy Kol, who was the first person I randomly found when coming for the first time to TU Delft and would sit across from me for the next three years, chat with me about anything and everything, and help me out with anything Dutch (or otherwise) related. Another special mention goes to Dr. Michael Stengel, who became a great influence and friend during his time at Delft.

Finally, I would like to thank all of my family for their unending support on my long way here. From early on they instilled in me the value of education, the importance of hard work, and encouraged me to follow my dreams. Without them I would not have made it this far. And last but not least to Maru, who stood by my side all these years, I have no words to express my gratitude.

CURRICULUM VITÆ

LEONARDO SCANDOLO

Leonardo Scandolo was born in Rosario, Argentina.

He graduated as an electronics technician in 2002 and as a Licenciante in Computer Science in 2013 at the National University of Rosario (UNR). During his studies, he performed research internships at the IMDEA¹ Software institute in Madrid, Spain, researching topics of parallel programming, and also at INRIA² Rhone-Alpes in Grenoble, France, on the topic of trajectory planning for autonomous wheelchairs. He also interned at Evolution Robotics (now part of iRobot³) in Los Angeles, United States. During and after his studies he also worked as a software engineer building systems for energy distribution companies at Tesis S.A., and precision agriculture display software at Forkworks S.A. in Argentina. He also worked as a teaching assistant at UNR, teaching topics of data structures, computer architecture, algorithms and linear algebra.

In 2015, he started his PhD at TU Delft, under the supervision of Prof. Elmar Eisemann, researching a variety of topics relating to real-time rendering, animation, and editing. That work resulted in the current dissertation.



¹www.software.imdea.org

²www.inria.fr/centre/grenoble

³www.irobot.com

LIST OF PUBLICATIONS

10. **L. Scandolo**, P. Bauszat, E. Eisemann, *Gradient-Guided Local Disparity Editing*, Computer Graphics Forum, Vol. 38, No. 1 (2019).
9. **L. Scandolo**, S. Lee, E. Eisemann, *Quad-Based Fourier Transform for Efficient Diffraction Synthesis*, Computer Graphics Forum, Vol. 37, No. 4 (2018).
8. C. Brandt, **L. Scandolo**, E. Eisemann, K. Hildebrandt, *Modeling n-Symmetry Vector Fields using Higher-Order Energies*, Computer Graphics Forum, Vol. 37, No. 2 (2018).
7. C. Brandt, **L. Scandolo**, E. Eisemann, K. Hildebrandt, *Spectral processing of tangential vector fields*, Computer Graphics Forum, Vol. 36, No. 6 (2017).
6. R. Baravalle, **L. Scandolo**, C. Delrieux, C. García Bauza, E. Eisemann, *Realistic modeling of porous materials*, Computer Animation and Virtual Worlds, Vol. 28, No. 2 (2017).
5. **L. Scandolo**, P. Bauszat, E. Eisemann, *Merged multiresolution hierarchies for shadow map compression*, Computer Graphics Forum, Vol. 35, No. 7 (2016).
4. **L. Scandolo**, P. Bauszat, E. Eisemann, *Compressed Multiresolution Hierarchies for High Quality Precomputed Shadows*, Computer Graphics Forum, Vol. 35, No. 2 (2016).
3. Q. Hendrickx, **L. Scandolo**, M. Eisemann, E. Eisemann, *Adaptively layered statistical volumetric obscuration*, Proceedings of the 7th Conference on High-Performance Graphics (2015).
2. **L. Scandolo**, T. Fraichard, *An anthropomorphic navigation scheme for dynamic scenarios*, Proceedings of the IEEE International Conference on Robotics and Automation (2011).
1. **L. Scandolo**, C. Kunz, M. Hermenegildo, *Program parallelization using synchronized pipelining*, Proceedings of the International Symposium on Logic-Based Program Synthesis and Transformation (2009).