Representing Large Virtual Worlds

Kol, Timothy

DOI
10.4233/uuid:02f47a5f-9699-478b-95db-d7163d33912e

Publication date
2018

Document Version
Final published version

Citation (APA)
Kol, T. (2018). *Representing Large Virtual Worlds*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:02f47a5f-9699-478b-95db-d7163d33912e


Representing
Large Virtual Worlds
Timothy R. Kol

About the cover: this is a modified version of the San Miguel *scene, famous in computer graphics and originally modeled by Guillermo M. Leal Llaguno of Evolución Visual, based on a hacienda that he visited in San Miguel de Allende, Mexico. The image was rendered in real time and stylized using the techniques presented in Chapter 5. The LEGO bricks on the back symbolize the voxel representations of virtual worlds discussed in Chapter 2.*

REPRESENTING LARGE VIRTUAL WORLDS

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 for Doctorates
to be defended publicly on Friday, May the 4th, 2018 at 12:30 o'clock

by

Timothy René KOL

Master of Science in Computer Science, Utrecht University, The Netherlands
born in Schiedam, The Netherlands

This dissertation has been approved by the promotor.

Composition of the doctoral committee:

Rector Magnificus,	chairperson
Prof. dr. E. Eisemann,	Delft University of Technology, promotor
Independent members:	
Prof. dr. ir. P. J. M. van Oosterom	Delft University of Technology
Dr. M. Wimmer	TU Wien, Austria
Dr. E. Gobbetti	Center for Advanced Studies, Research, and Development in Sardinia (CRS4), Italy
Dr. M. Billeter	Delft University of Technology
Prof. dr. ir. M. J. T. Reinders	Delft University of Technology, reserve member

This work was partly supported by the EU Seventh Framework Programme as part of the project HARVEST4D: *Harvesting Dynamic 3D Worlds from Commodity Sensor Clouds* under grant number EU 323567. This work was carried out in the ASCI Graduate School. ASCI dissertation series number 389.

Printed by: Drukkerij Haveka

ISBN 978-94-6186-896-1

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

Voor opa Teip

Contents

Summary ix

Samenvatting xi

Preface xiii

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Representations . . . 2
- 1.3 Selected Challenges . . . 3
- 1.4 Contributions . . . 5
  - 1.4.1 Underlying Representations: Compressing Voxel Scenes . . . 5
  - 1.4.2 Realistic Representations: Many-View Rendering . . . 6
  - 1.4.3 Illustrative Representations: 3D Virtual Cities . . . 7
  - 1.4.4 Artistic Representations: Expressive Single Scattering . . . 8
- 1.5 Summary . . . 9

2 Geometry and Attribute Compression for Voxel Scenes 11
- 2.1 Introduction . . . 12
- 2.2 Related Work . . . 13
- 2.3 Background . . . 14
- 2.4 Compression . . . 15
  - 2.4.1 Voxel Attribute Decoupling . . . 16
  - 2.4.2 Palette Compression . . . 16
  - 2.4.3 Attribute Quantization . . . 19
  - 2.4.4 Geometry Compression . . . 19
- 2.5 Results . . . 20
  - 2.5.1 Decoupling and Palette Compression . . . 21
  - 2.5.2 Attribute Quantization . . . 22
  - 2.5.3 Offset and Pointer Compression . . . 24
  - 2.5.4 Comparison . . . 24
  - 2.5.5 Performance . . . 26
  - 2.5.6 Applications . . . 27
- 2.6 Conclusions . . . 27

3 MegaViews: Scalable Many-View Rendering 29
- 3.1 Introduction . . . 30
- 3.2 Related Work . . . 31
- 3.3 Scalable Many-View Rendering . . . 32
  - 3.3.1 Scene and View Hierarchies . . . 32
  - 3.3.2 Many-View Rendering . . . 34

3.4 Results . . . 38
3.5 Applications . . . 41
3.5.1 Instant Radiosity . . . 42
3.5.2 Glowing Particles . . . 44
3.6 Discussion and Limitations . . . 44
3.7 Conclusion . . . 45

4 Real-Time Canonical-Angle Views in 3D Virtual Cities 47
4.1 Introduction . . . 48
4.2 Related Work . . . 49
4.3 Canonical Views . . . 50
4.3.1 Building Transformation . . . 51
4.3.2 Occlusion Test . . . 52
4.3.3 Obtaining the Canonical Angle . . . 54
4.4 Results . . . 55
4.4.1 Evaluation . . . 55
4.4.2 Finding Buildings Using the Canonical View . . . 56
4.4.3 Memorizing Routes . . . 56
4.4.4 Discussion . . . 57
4.5 Conclusions and Future Work . . . 59

5 Expressive Single Scattering for Light Shaft Stylization 61
5.1 Introduction . . . 62
5.2 Related Work . . . 64
5.2.1 General Stylization . . . 64
5.2.2 Stylized Scattering . . . 64
5.2.3 Specific Techniques . . . 65
5.3 Real-Time Scattering Background . . . 66
5.4 Stylized Single Scattering . . . 66
5.4.1 Occluder Manipulation . . . 67
5.4.2 Color Modifications . . . 72
5.4.3 Heterogeneity Modification . . . 75
5.5 Results and Discussion . . . 76
5.5.1 Stylization . . . 76
5.5.2 Performance . . . 81
5.5.3 Discussion . . . 83
5.6 Conclusion . . . 83

6 Conclusion 85

Bibliography 89

Epilogue 101

Acknowledgements 105

Curriculum Vitæ 109

List of Publications 111

SUMMARY

The ubiquity of *large virtual worlds* and their growing complexity in computer graphics require efficient *representations*. This means that we need smart solutions for the *underlying storage* of these complex environments, but also for their *visualization*. How the virtual world is best stored and how it is subsequently shown to the user in an optimal way, depends on the goal of the application. In this respect, we identify the following three *visual representations*, which form orthogonal directions, but are not mutually exclusive. *Realistic* representations aim for physical correctness, while *illustrative* display techniques, on the other hand, facilitate user tasks, often relating to improved understanding. Finally, *artistic* approaches enable a high level of expressiveness for aesthetic applications. Each of these directions offers a wide array of possibilities. In this dissertation, our goal is to provide solutions for strategically selected challenges for all three visual directions, as well as for the *underlying representation* of the virtual world.

To work with virtual environments, we first need to efficiently *store* them. Two common approaches rely on either surfaces or voxels. Since voxels are easily queried anywhere in space, they are beneficial for, e.g., realistic lighting and collision detection. However, naively storing large worlds is problematic, since the available memory typically does not suffice, requiring better representations and compression. Existing methods often exploit sparsity, but scale insufficiently well, since they do not consider repeating patterns. In Chapter 2, we present an improved data representation and subsequent compression based on repetition and coherence. Our technique enables us to significantly reduce the memory footprint, with better performance for large, complex scenes.

With this improved storage method, we can now focus on how to display a virtual world. Many graphics applications aim at producing physically correct images, for which we need a *realistic* visual representation of the environment. In this regard, there is an increasing number of realistic lighting algorithms that rely on rendering a scene from different viewpoints. While rendering highly complex scenes is difficult by itself, it becomes exceptionally hard in the presence of *multiple* viewpoints, as performance is severely limited if no view coherence is exploited. In Chapter 3, we solve this problem by taking into account the redundancy present in many-view scenarios. In other words, we enable views that see similar parts of the scene to share their rendering. Our algorithm scales well, as the presence of more views typically corresponds to more coherence. Thus, we are able to produce realistic lighting effects for large virtual environments.

However, real-world environments and their realistic representations are not always easy to comprehend due to high scene complexity. For many user tasks, it is beneficial to rather show the world in an *illustrative* fashion, which requires special visualization techniques. Navigation in particular is important for large environments, but for 3D virtual city models, this offers a visualization paradox. On the one hand, the street network

needs to remain visible for planning and maintaining a good overview, for which a top-down view works well. On the other hand, users benefit significantly from seeing the building facades, as this aids them in recognizing their position and memorizing routes. In Chapter 4, we better facilitate navigation tasks by employing the *canonical view*, a user-preferred view that improves object recognition. By using the canonical view for buildings, we can combine the best of both worlds, enabling a top-down view while still showing building facades. Our viewer gives users a better understanding of large-scale city models, as validated in our user study.

While user understanding or realism are often desired properties of display methods, in some instances, aesthetics have a high priority, too. Here, *artistic* representations are needed, which provide the user with efficient control over the virtual world. The environment's appearance is greatly influenced by illumination and atmospheric effects, of which light shafts in participating media form one of the many challenging aspects. These effects are often generated by physically-based simulations, limiting modifications to changing the physical parameters, of which the outcome can be difficult to predict. Also, the user is restricted to physically correct results, which is not always desirable. Therefore, we present smart manipulation techniques in Chapter 5, enabling quick and efficient modifications that potentially go beyond physical correctness. Our solution is able to change the light behavior on a large scale, resulting in a significant reduction of the required labor. We hereby offer an artistic representation with increased expressiveness for light shafts.

We believe that the complexity of virtual environments will continue to grow exponentially in computer graphics, necessitating efficient representations like ours. Overall, our methods contribute to facing several challenges of the storage of virtual worlds and their realistic, illustrative and artistic display. Precisely, we are now able to better store high-resolution voxel scenes. Additionally, we can render more efficiently for a high number of viewpoints, which enables realistic lighting techniques. Furthermore, we improve the visualization of large-scale city models for better navigation. Finally, we facilitate artistic control of light shafts, enabling effortless, expressive changes. With this, we take a step towards better representing large virtual worlds.

SAMENVATTING

De alomtegenwoordigheid en de groeiende complexiteit van *grote virtuele werelden* in computergraphics vereist efficiënte *representaties*. Dit betekent dat we slimme oplossingen nodig hebben voor zowel de *onderliggende opslag* van complexe omgevingen, als hun *visualisatie*. Hoe de virtuele wereld het best wordt opgeslagen, en hoe deze vervolgens optimaal aan de gebruiker wordt weergegeven, ligt aan het doel van de toepassing. In dit opzicht identificeren we de volgende drie *visuele representaties*, welke verschillend, maar niet wederzijds exclusief zijn. *Realistische* representaties richten zich op natuurkundig correcte resultaten, terwijl *illustratieve* weergavetechnieken juist gebruikerstaken faciliteren, vaak gerelateerd aan een verbeterde begrijpelijkheid. *Artistieke* methodes ten slotte stellen gebruikers in staat tot een hoge graad van expressiviteit voor toepassingen waar esthetiek van belang is. Elk van deze richtingen biedt een scala aan mogelijkheden. Ons doel in dit proefschrift is het bieden van oplossingen voor strategisch gekozen uitdagingen voor alle drie de visuele richtingen, en tevens de *onderliggende representatie* van de virtuele wereld.

Om met virtuele omgevingen te werken, moeten we ze eerst op efficiënte wijze *opslaan*. Twee veelvoorkomende methodes berusten op oppervlaktes of voxels. Aangezien voxels gemakkelijk op elke positie in de ruimte opgevraagd kunnen worden, zijn ze voordelig voor bijvoorbeeld realistische belichting en collisiedetectie. Het naïef opslaan van grote werelden is echter problematisch, aangezien de hoeveelheid beschikbaar geheugen meestal niet voldoet, en dus betere representaties en compressie vereist zijn. Bestaande methodes maken vaak gebruik van schaarsheid, maar schalen onvoldoende, omdat ze geen rekening houden met herhalende patronen. In hoofdstuk 2 presenteren wij een verbeterde datarepresentatie en daaropvolgende compressie gebaseerd op herhaling en gelijkenis. Onze techniek stelt ons in staat de benodigde opslagruimte significant te verminderen, met betere prestaties voor grote, complexe scènes.

Met deze verbeterde opslagmethode kunnen we ons richten op de weergave van virtuele werelden. Veel toepassingen in graphics richten zich op het produceren van waarheidsgetrouwe afbeeldingen, waarvoor een *realistische* visuele representatie van de omgeving nodig is. In dit verband is er een toenemende hoeveelheid algoritmes voor realistische belichting die berusten op scèneweergaves vanuit verschillende oogpunten. Hoewel het weergeven van zeer complexe scènes op zich al niet gemakkelijk is, wordt het nog meer bemoeilijkt bij *meerdere* aangezichten, aangezien de prestaties ernstig gelimiteerd worden als de samenhang tussen gezichtspunten niet wordt uitgebuit. In hoofdstuk 3 lossen we dit probleem op door rekening te houden met de overtolligheid die zich in zulke scenario's voordoet. In andere woorden, we stellen oogpunten die gelijke delen van de scène zien in staat om de resulterende weergave met elkaar te delen. Ons algoritme schaalt goed, omdat de aanwezigheid van meer aangezichten over het algemeen

leidt tot meer samenhang. Zodoende creëren we realistische belichting voor grote virtuele omgevingen.

Zowel de echte wereld als realistische representaties zijn door de hoge complexiteit echter niet altijd gemakkelijk te begrijpen. Voor veel gebruikerstaken is het juist voordelig om de omgeving op een *illustratieve* wijze te laten zien, wat speciale visualisatietechnieken vereist. Navigatie is bijzonder belangrijk voor grote omgevingen, maar voor driedimensionale virtuele steden biedt dit een paradox. Aan de ene kant moet het straatnetwerk zichtbaar blijven voor planning en om een goed overzicht te behouden, waarvoor een bovenaanzicht goed werkt. Aan de andere kant behalen gebruikers een groot voordeel uit het zien van façades van gebouwen, aangezien dit helpt bij het herkennen van hun positie en het onthouden van routes. In hoofdstuk 4 faciliteren we navigatietaken beter door middel van het *canonieke aanzicht*, een geprefereerd oogpunt dat objectherkenning verbetert. Door gebruik te maken van het canonieke aanzicht kunnen we het beste van beide werelden combineren, wat ons in staat stelt een bovenaanzicht te hanteren terwijl de façades zichtbaar blijven. Onze techniek geeft gebruikers een beter begrip van virtuele steden op grote schaal, zoals we valideren in ons gebruikersonderzoek.

Hoewel begrijpelijkheid of realisme vaak gewenste eigenschappen zijn van weergavemethodes, heeft esthetiek in sommige gevallen ook een hoge prioriteit. Hier zijn *artistieke* representaties nodig, die de gebruiker efficiënte controle over de virtuele wereld verschaffen. Het uiterlijk van de omgeving wordt sterk beïnvloed door belichting en atmosferische effecten, van welke lichtstralen veroorzaakt door deeltjes in de atmosfeer een van de vele uitdagingen vormen. Dergelijke effecten worden vaak gegenereerd door natuurkundige simulaties, wat aanpassingen limiteert tot het wijzigen van fysieke eigenschappen, waarvan de uitkomst vaak moeilijk te voorspellen is. Hierom presenteren wij slimmere manipulatietechnieken in hoofdstuk 5, wat gebruikers in staat stelt tot snelle en efficiënte wijzigingen, die mogelijkerwijs niet natuurkundig correct zijn. Met onze oplossing kan het gedrag van lichteffecten op een grote schaal aangepast worden, wat tot een aanzienlijke vermindering van de benodigde arbeid leidt. We bieden hiermee een artistieke representatie met verbeterde expressiviteit voor lichtstralen.

Wij geloven dat de complexiteit van virtuele omgevingen in computergraphics exponentieel zal blijven groeien, waardoor efficiënte representaties als de onze nodig zijn. In het algemeen dragen onze methodes bij aan het oplossen van meerdere uitdagingen betreffende het opslaan van virtuele werelden en hun realistische, illustratieve en artistieke weergave. Om precies te zijn kunnen we voxelscènes met een hoge resolutie nu beter opslaan. Verder kunnen we een scène vanuit meerdere oogpunten efficiënter weergeven, wat de weg vrij maakt voor realistische lichtsimulaties. Daarnaast verbeteren we de visualisatie van steden op grote schaal voor een betere navigatie. Ten slotte faciliteren we artistieke controle over lichtstralen, wat gebruikers in staat stelt moeiteloos expressieve wijzigingen te maken. Hiermee nemen we een stap in de richting van een betere representatie van grote virtuele werelden.

PREFACE

Before you lies the dissertation *Representing Large Virtual Worlds*, which is the culmination of the work carried out during the past four years as part of my PhD candidacy.

The research here presented was partly supported by the EU project HARVEST4D: *Harvesting Dynamic 3D Worlds from Commodity Sensor Clouds*, a collaboration between six institutions: TU Wien (Austria), Technische Universität Darmstadt (Germany), University of Bonn (Germany), Télécom ParisTech (France), CNR Institute of Information Science and Technology (Italy) and Delft University of Technology (The Netherlands).

I started off as a relatively inexperienced researcher. My previous work on computer graphics [Kol12, Kol13], however, had sparked my interest. While I had learned a lot during my master's program at Utrecht University, I still felt there was a whole wealth of information waiting to be unearthed, and contributions to be made. Professor Elmar Eisemann kindly gave me the opportunity to do so, by serving as my promotor and supervisor, for which I am very grateful. Thus, I became a PhD candidate at Delft University of Technology in the Computer Graphics and Visualization group.

From here, the main part of the dissertation commences. After the introduction, it contains four technical chapters, which are all based on either submissions or publications. As the presented work covers a broad research area, each chapter contains its own introduction and background section.

All chapters are based on multi-author papers. Therefore, in the footnote at the beginning of each Chapter, I will shortly mention my personal contributions.

I hope you find this dissertation an enjoyable and interesting read.

Timothy René Kol
Delft, December 30, 2017

1

INTRODUCTION

To really ask is to open the door to the whirlwind.
The answer may annihilate the question and the questioner.

Anne Rice

THE field of computer graphics revolves around the creation and manipulation of images. In most domains where graphics are applied, like architecture, visual effects, entertainment, visualization, and medicine, this visual content is typically generated based on a 3D virtual environment that we wish to show. Governed by graphics algorithms, the computer runs calculations on this virtual scene to draw it – a process called rendering. The environment needs to be stored in an *underlying representation* for the computer to read and process it. The resulting image, on the other hand, offers a *visual representation* of the scene. These two representations, and the techniques to generate them, lie at the core of much research in computer graphics, as well as this dissertation.

1.1. MOTIVATION

As hardware capabilities grow, today's virtual environments are becoming larger and more detailed, and the calculations more numerous and complex, in order to satisfy increasing user expectations of realistic, beautiful, or informative images. Cinema audiences have come to expect more impressive visual effects every year, while architects require the highest level of realism to pre-visualize their projects, and gamers want to immerse themselves in enormous, detailed worlds. Nevertheless, companies are more than willing to try and meet the increasing demands. After all, in 2017, global revenue for the gaming industry is projected to exceed 108 billion dollars[1], and recent films spend millions of dollars on computer-generated imagery (CGI) alone. Indeed, with these developments, we are now often tasked with rendering *large virtual worlds*. For instance,

[1] Newzoo Global Games Market Report, 2017

the recently developed game *The Witcher 3: Wild Hunt* (2015) contains a game world covering 136 square kilometers[2].

While these large environments potentially enable impressive visual results, they require a vast amount of computations and storage capacity. For example, the animated movie *Frozen* (2013) took a total of 60 million hours of compute time and 6 petabytes (PB) of storage[3], which comes with very high costs in both hardware and energy consumption. To face these challenges, we need storage and rendering solutions that perform well under the increasing complexity of 3D environments. Consequently, in this dissertation, we address the question of how to efficiently represent large virtual worlds, in the context of both the underlying and the visual representation.

To achieve improved performance, it is important that our solutions scale favorably with scene size and complexity. That is, the additional computation cost should be less than proportional to the added virtual world data. To this end, we identify the exploitation of repeating patterns and similarity as a crucial aspect. By reducing such redundancy, we can benefit from the fact that less storage space and fewer computations are needed. Besides scalability, making use of the strengths and weaknesses of human perception offers additional opportunities to improve the efficiency of visualizations. With these insights, we aim to achieve better representations of large virtual worlds, in order to cut down on computation costs and further improve user experience.

1.2. REPRESENTATIONS

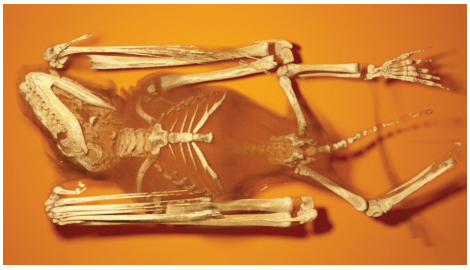
Due to the sheer variety of the domains in which virtual environments are applied, and the different purposes these applications serve, we cannot define a general, optimal representation; different demands result in distinct criteria for optimality. For the digital storage of large virtual worlds, i.e., their *underlying representation*, the preferred solution often depends on the scene's properties. For instance, Figure 1.1c shows a medical visualization of a bat's skeleton, generated from a dense, discrete volumetric dataset, which can be well represented by little cubes (voxels) laid out in a grid. The cityscape in Figure 1.1d on the other hand is sparse, and only contains surfaces to describe the exterior of rectangular buildings. It is therefore better suited for surface-based approaches, i.e., simply storing the corner points of each surface.

For *visual representations*, too, different strategies need to be employed depending on the goal of the visualization. In general, we identify three categories of visual representations. One possibility is a *realistic* depiction, relying on physically-based simulations or approximations thereof, which is beneficial for immersion and understanding. An orthogonal option is an *illustrative* approach, to visualize the world in such a way that best facilitates performing a certain task, which is often linked to user understanding. Sufficient comprehension may entail the generation of abstract, but instructive images. Finally, applications such as games and visual effects can depend on aesthetics, leading to *artistic* representations that aim at conveying a certain mood or message. Here too, we commonly prefer to forgo realism in favor of expressiveness.

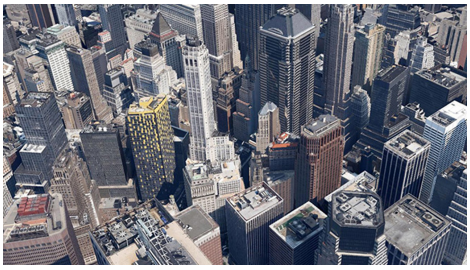
[2] CD Projekt Red presentation at Game Developers Conference, 2014
[3] Walt Disney Animation Studios presentation at Large Installation System Administration Conference, 2013

(a) Voxel-based ambient occlusion in games. © Crystal Dynamics and Square Enix.

(b) Point-based global illumination in movies. © Disney Enterprises and Jerry Bruckheimer.



(c) Medical visualization of voxelized volume data.



(d) Visual clutter in city models. © Google.

Figure 1.1: Various applications where large virtual worlds and the challenges they pose play a role.

While compromises may be needed, these directions are not mutually exclusive. Indeed, many applications require a combination: e.g., while an illustrative depiction may have to sacrifice some faithfulness, in many situations, a certain degree of realism can still be desirable, and establishing a link to real-world physical phenomena helps the viewer to better grasp the depicted elements. To illustrate this point, let us take an example from the medical domain (e.g., Figure 1.1c), where having a realistic depiction allows a non-expert to better understand the captured data, as it is represented in a way that is closer to the viewer's experience. Nevertheless, illustrative approaches that highlight the area of interest are a crucial feature as well. For games (Figure 1.1a) and visual effects (Figure 1.1b), both realism and artistic freedom are important components to create a convincing and expressive representation.

1.3. SELECTED CHALLENGES

The aforementioned underlying and visual representations offer a wide array of options. Since it is impossible to cover all in this dissertation, we select relevant challenges in each. We first focus on the underlying representation of large virtual worlds, for which surface-based and voxel-based approaches are most commonly employed. The former can represent large surfaces at a low memory cost, but lack efficient random access. The latter can be queried anywhere in space, facilitating applications like efficient collision detection and advanced lighting, which is crucial for realistic representations. Voxels are the current standard for encoding volume data, such as in medical visual-

ization (Figure 1.1c). Even for applications originally dominated by surface representations, keeping voxelized copies of the scene is becoming more common. For instance, in the recent game *Rise of the Tomb Raider* (Figure 1.1a), voxels are used to approximate ambient occlusion using cone tracing techniques [CNS*11]. Unfortunately, when representing highly complex voxel scenes, the memory footprint can easily grow beyond the capacity of the used hardware. Approaches that rely on streaming can partly overcome this [GMIG08, CNLE09, CNSE10] and are sometimes even inevitable for truly enormous datasets [LKT*17]. Nevertheless, these techniques result in additional performance costs and require data transfer, hindering usage for applications like games, where memory and computational resources are severely limited. Therefore, we need to look into compression to provide more efficient underlying voxel-based representations.

For *realistic* representations, the faithful simulation of the interplay of light and the environment plays a vital role. Indeed, the best indication that an image is generated by a computer is often a lack of plausible lighting, such as incorrect shadows, lack of scattering, and unrealistic reflections. One effect that is particularly difficult to simulate, yet present in virtually all scenes, is global illumination. In real life, light bounces many times off the environment's materials before it attenuates. Because of this, we rarely see completely black areas, and can observe so-called color bleeding, a subtle but important feature for realistic renderings. Path tracing simulates this bouncing adequately, yet it can be so computationally expensive that even for offline applications like visual effects, naive methods are infeasible. Therefore, many solutions exist that approximate global illumination, which often require a scene to be seen from a large variety of different viewpoints [DKH*14]. This is the case for instant radiosity [Kel97] and point-based global illumination [Chr08] (Figure 1.1b), but also holds true for virtual worlds that contain many light sources (e.g., glowing particles). At their core, these algorithms produce views of the scene from many different locations. This is typically done to establish which part of the scene would potentially be illuminated from this location. Sequentially rendering the scene from all of these light locations is highly detrimental to the performance, and adequate, scalable solutions are needed.

Contrary to visualizing the environment as is, *illustrative* representations pursue a certain goal that is linked to user interaction. In large virtual worlds, naturally, *navigation* is an important aspect of interaction. Applications like route planning and tourist maps specifically focus on navigation-related interaction, but also for disaster simulations, games, and medical applications, navigating the environment often plays a large role. While tools such as Google Earth exist that are capable of efficiently handling enormous amounts of data, they do not always provide an optimal visual representation for navigation purposes, which becomes most evident in large-scale virtual cities (Figure 1.1d). Here, a good overview of the streets for route planning is beneficial, as is the ability to discern landmarks such as buildings, which are a key element for route memorization and recognizing your current location [Den97]. Top-down views preserve a good overview of the street network, but only show rooftops, which significantly reduces the recognizability of buildings. While a bird's-eye view gives a better perspective, enabling users to discern building facades, it suffers from visual clutter near high-rises, as in Figure 1.1d. Likewise, street-level views show buildings from a good angle, but com-

pletely lack the street overview. Showing multiple views side by side is an option, but this requires users to divide their attention, which can be disadvantageous to a good scene understanding. We therefore require illustrative visualizations, that manage to show an overview while maintaining building recognizability.

A different dimension is the *artistic* depiction of the environment. This is especially relevant in the game and visual effects industries, where the end product is often a work of art, enabled by computer graphics algorithms. The aesthetics of large virtual worlds are for a major part governed by illumination and atmospheric effects. Artistic solutions do not always need to obey the laws of physics, especially since there are many light effects for which humans can not easily judge their physical correctness. This is true in particular when there is a complex interplay of light and matter at work, like for caustics, multiple scattering, or light shafts. As these effects are often considered visually pleasing, they are commonly employed by artists. To achieve expressive results, we need to grant the artist control over the virtual world and the simulation of light within, which is a difficult challenge [SPN*16]. As an example, we can consider caustics, which are caused by reflection and refraction of light by curved surfaces, as with a glass of water. To achieve a desired appearance for the caustics, without artistic tools, an artist would have to know exactly how the materials, curved geometry and light properties need to be modified, which is an impossible task. For caustics, several approaches exist that enable more efficient control [SNM*13, GSLM*08], but many light effects remain for which artistic depictions are necessary.

1.4. CONTRIBUTIONS

In the above, we have identified four specific challenges pertaining to storing virtual worlds and their realistic, illustrative and artistic representation. To each of these challenges, we dedicate a chapter. Below, we briefly state the contributions per chapter.

1.4.1. UNDERLYING REPRESENTATIONS: COMPRESSING VOXEL SCENES

Large, navigable virtual worlds are often sparse. For instance, the 3D grid that represents the scene from Figure 1.2 is 99.999% empty. Therefore, a common approach to reduce the memory footprint of sparse voxel scenes is to store them in a spatial data structure, like sparse voxel octrees (SVOs) [JT80, Mea82].

Still, large, detailed environments will exceed available memory, preventing in-core storage. To store these scenes, we need alternative representations that are subsequently compressed, for which it is necessary to consider repetition and similarity. In doing so, we still need to be able to efficiently query the compressed data for interactive solutions, requiring a careful selection of the compression technique. Recently, geometry data was successfully compressed by encoding it as a directed acyclic graph (DAG) [KSA13]; however, additional information, such as colors and normals, requires a different approach.

We present a method for compressing SVOs. By decoupling the geometry information from the voxel attributes, such as colors and normals, we can apply a separate compression for both. The geometry is encoded as a DAG, where our decoupling strategy enables a significant size reduction. Furthermore, the attributes are quantized and

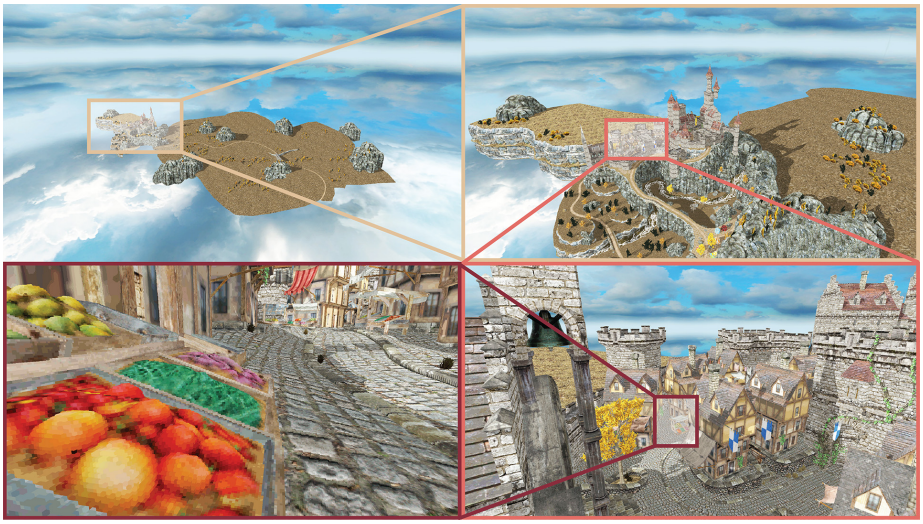


Figure 1.2: Compression of large, navigable virtual worlds represented by voxels.

compressed using our tailor-made palette compression technique. These contributions enable storing voxel scenes fully in-core at resolutions previously not attainable (up to $128K^3$) while retaining real-time random access. Figure 1.2 shows an exemplary voxel scene with the aforementioned resolution, which is stored fully on the GPU. We further discuss this work in Chapter 2.

This work was published as Geometry and Attribute Compression for Voxel Scenes *in Computer Graphics Forum* ***35****, 2 (2016), by Bas Dado, Timothy R. Kol, Pablo Bauszat, Jean-Marc Thiery and Elmar Eisemann [DKB*16]. It was presented at Eurographics 2016 in Lisbon, Portugal.*

1.4.2. REALISTIC REPRESENTATIONS: MANY-VIEW RENDERING

Efficiently querying voxels is especially beneficial for realistic lighting. In this respect, we consider global illumination as an important and challenging aspect. Efficient approximations of the global illumination generally require many views of the scene. Naive techniques, however, cannot handle such scenarios efficiently. Even methods that are highly parallelized and improve scalability by using either a special scene [HREB11] or view [WFA*05] representation, fall short on performance, as they are not able to exploit all redundancy.

We propose a novel solution that uses both a scene and a view hierarchy to identify redundant information and exploit coherence. We make use of an efficient concurrent traversal of the hierarchies to find and perform shared rendering, enabling nearly real-time performance for up to a million views of a complex scene. We describe our technique in detail in Chapter 3, where we also showcase many-light rendering applications, such as the glowing particles depicted in Figure 1.3, and the aforementioned instant radiosity to approximate global illumination.

Figure 1.3: Many-view rendering. Here, we show many-light rendering for glowing particles.

This work was published as MegaViews: Scalable Many-View Rendering with Concurrent Scene-View Hierarchy Traversal *in Computer Graphics Forum (2018), by Timothy R. Kol, Pablo Bauszat, Sungkil Lee and Elmar Eisemann [KBLE18].*

1.4.3. ILLUSTRATIVE REPRESENTATIONS: 3D VIRTUAL CITIES

To obtain an illustrative visualization of a city model that better facilitates efficient navigation, we need to see landmarks like buildings from a recognizable perspective [Den97]. In this context, recognition is optimal for a low-angle view, since this is the natural way in which humans normally see buildings; it is our *preferred viewpoint* for navigation tasks. In fact, using such a preferred viewpoint improves recognition for many objects [EB92,

Figure 1.4: Canonical-angle views in virtual city models.

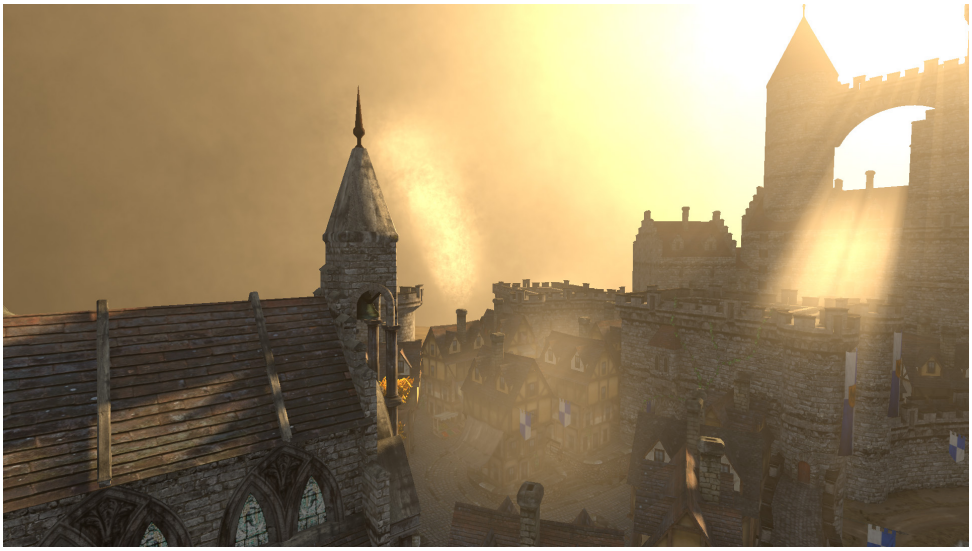


Figure 1.5: Stylized scattering: chimney smoke, a thick fog between the houses, and enhanced light shafts.

VB95, BTBV99]. This viewpoint is known as the *canonical view* [PRC81].

We propose to use a modified version of the canonical view for buildings in 3D cities to improve landmark recognition. In this sense, we provide an illustrative representation of large-scale city models. Our viewer (see Figure 1.4) applies a view-dependent transformation on buildings, enabling better route memorization and building recognizability. We discuss our method in Chapter 4, which includes a user study to assess its usefulness.

This work was published as Real-Time Canonical-Angle Views in 3D Virtual Cities *in the proceedings of VMV: Vision, Modeling & Visualization by Timothy R. Kol, Jingtang Liao and Elmar Eisemann [KLE14]. It was presented at VMV 2014 in Darmstadt, Germany.*

1.4.4. ARTISTIC REPRESENTATIONS: EXPRESSIVE SINGLE SCATTERING

Numerous light phenomena are suitable for artistic representations, particularly for indirect effects. We focus on light shafts, which are one of the most frequently stylized effects in traditional art, and can add not only realism, but also depth cues and improved understanding to the scene. A good starting point for an artistic representation is a physically correct simulation of the light shafts, which can then be manipulated to the user's liking. However, restricting the modifications to physical parameters or scene geometry severely limits the expressiveness. For an artistic representation of light shafts caused by single scattering, we therefore present several efficient manipulation tools that create plausible results, while not necessarily adhering to the laws of physics. Our stylization of scattering effects enables quick expressive changes, which is especially useful for large environments that would otherwise require laborious manual tweaking. We support changing the appearance with occluder manipulation, corresponding to adding, removing or enhancing light shafts. Since we rely on a shadow map, the resulting scat-

tering is fast to calculate and largely independent of the complexity of the environment. Furthermore, colors can be easily controlled using transfer functions and a light map optimization approach. Finally, we enable heterogeneity modification, which allows local variation of the participating medium density. We show a result obtained with our tools in Figure 1.5. Chapter 5 contains an in-depth description of our techniques, including a wide array of results.

This work was published as Expressive Single Scattering for Light Shaft Stylization *in the IEEE Transactions on Visualization and Computer Graphics* ***23****, 7 (2017), by Timothy R. Kol, Oliver Klehm, Hans-Peter Seidel and Elmar Eisemann [KKSE17]. This in turn was an extension of a previous publication in the proceedings of GI: Graphics Interface by the same authors that was presented at GI 2015 in Halifax, Canada [KKSE15].*

1.5. SUMMARY

In Chapter 2, we will show how voxel-based representations can be compressed to such an extent that they can be used in a wide array of applications. Especially with an eye on the future, approaches like ours may become more desirable to store large virtual worlds, as an addition to, or even a replacement for, current representations.

Besides storage, we have identified realistic, illustrative and artistic representations as the major categories for graphics applications, and propose contributions to selected challenges in each of these.

We will discuss a novel many-view rendering technique in Chapter 3, which can be applied for global illumination and many-light scenarios to produce more realistic visual representations. Like our storage solution, it relies on coherence, which we see as a crucial factor for dealing with the growing complexity of virtual environments.

Illustrative representations that better facilitate a certain user task, form another important aspect. To improve navigational tasks, we present an alternative visualization method for virtual 3D cities in Chapter 4. Such illustrative approaches remain an open problem, as they depend on human perception, and may require specific solutions for different tasks and scenes. Therefore, we believe techniques like ours to be important for the advancement of visual representations.

For artistic depictions, perception plays a big role as well, but in a somewhat different way. Here, artists often want users to perceive a desired mood, or incur certain feelings. In Chapter 5, we will see how our work makes a step in the direction of improved artistic control, which we consider of vital importance in the field of computer graphics.

Overall, we hereby aim to offer representations of large virtual worlds that improve the efficiency of computations and user interaction for selected challenges. With these means, our work contributes towards the production of more realistic, informative and beautiful images in the future.

2

Geometry and Attribute Compression for Voxel Scenes

Never accept the proposition that just because
a solution satisfies a problem, that it must be the only solution.

Raymond E. Feist

Voxel-based approaches are today's standard to encode volume data. Recently, directed acyclic graphs (DAGs) were successfully used for compressing sparse voxel scenes as well, but they are restricted to a single bit of (geometry) information per voxel. In this chapter, we present a method to compress arbitrary *data, such as colors, normals, or reflectance information. By decoupling geometry and voxel data via a novel mapping scheme, we are able to apply the DAG principle to encode the topology, while using a palette-based compression for the voxel attributes, leading to a drastic memory reduction. Our method outperforms existing state-of-the-art techniques and is well suited for GPU architectures. We achieve real-time performance on commodity hardware for colored scenes with up to* 17 *hierarchical levels (a* $128K^3$ *voxel resolution), which are stored fully in core.*

Save for an extended introduction, this chapter is a verbatim copy of a publication in Computer Graphics Forum **35**, 2 (2016), by Bas Dado, Timothy R. Kol, Pablo Bauszat, Jean-Marc Thiery and Elmar Eisemann [DKB*16]. It was presented at Eurographics 2016 in Lisbon, Portugal. As for the distribution of work, I implemented the initial framework, devised the palette compression with attribute quantization, the additional geometry compression, and wrote most of the paper.

Figure 2.1: Compressed voxelized scene at different levels of detail, rendered in real time using raytracing only. Our hierarchy encodes geometry and quantized colors at a resolution of $128K^3$. Despite containing 18.4 billion colored nodes, it is stored entirely on the GPU, requiring 7.63 GB of memory using our compression schemes. Only at the scale shown in the right bottom image the voxels become apparent.

2.1. INTRODUCTION

WITH the increase of complexity in large virtual worlds, alternative representations, which enable small-scale details and efficient advanced lighting, have received a renewed interest in computer graphics [LK10]. *Voxel-based* approaches encode scenes in a high-resolution grid. While they can represent complex structures, the memory cost grows quickly. Fortunately, most scenes are sparse – i.e., many voxels are empty. For instance, Figure 2.1 shows a scene represented by a grid of 2.25 quadrillion voxels ($128K^3$), but 99.999% are actually empty. Although hierarchical representations like *sparse voxel octrees* (SVOs) [JT80, Mea82] exploit this sparsity, they can only be moderately successful; a large volume like the one in Figure 2.1 still contains over 18 billion filled voxels.

For large volumes, specialized out-of-core techniques and compression mechanisms have been proposed, which often result in additional performance costs [BRGIG*14]. Only recently, *directed acyclic graphs* (DAGs) have shown that even large-scale scenes can be kept entirely in memory while being efficiently traversable. They achieve high compression rates of an SVO representation with a single bit of information per leaf node [KSA13]. Their key insight is to merge equal subtrees, which is particularly successful if scenes exhibit geometric repetition. Unfortunately, extending the information beyond one bit (e.g., to store material properties) is challenging, as it reduces the amount of similar subtrees drastically.

Our contribution is to associate attributes to the DAG representation, which are compressed separately, while maintaining efficiency in rendering tasks. To this extent, we introduce a *decoupling* of voxel attributes from the topology and a subsequent compression of these attributes. Hereby, we can profit from the full DAG compression scheme for the geometry and handle attributes separately. Although the compression gain is significant, the representation can still be efficiently queried. In practice, our approach enables real-time rendering of colored voxel scenes with a $128K^3$ resolution in full HD on commodity hardware while keeping all data in core. Additionally, attributes like normals or reflectance can be encoded, enabling complex visual effects (e.g., specular reflections).

Our main contributions are the decoupling of geometry and voxel data, as well as the

palette compression of quantized attributes, delivering drastic memory gains and ensuring efficient rendering. Using our standard settings, high-resolution colored scenes as in Figure 2.1 require on average well below one byte per voxel.

2.2. RELATED WORK

We only focus on the most related methods and refer to a recent survey by Balsa Rodríguez et al. [BRGIG*14] for other compression techniques, particularly for GPU-based volume rendering.

Large datasets can be handled via *streaming*; recent approaches adapt a reduced representation on the GPU by taking the ray traversals through the voxel grid into account [GMIG08, CNLE09, CNSE10]. Nonetheless, data transfer and potential disk access make these methods less suitable for high-performance applications. Here, it is advantageous to keep a full representation in GPU memory, for which a compact data structure is of high importance.

Dense volume compression has received wide attention in several areas – e.g., in medical visualization [GWGS02]. These solutions mostly exploit local *coherence* in the data. We also rely on this insight for attribute compression, but existing solutions are less suitable for *sparse* environments. In this context, besides SVOs [JT80, Mea82], *perfect spatial hashing* can compress a sparse volume by means of dense hash and offset tables [LH06]. While these methods support efficient random access, exploiting only sparsity is insufficient to compress high-resolution scenes.

Efficient sparse voxel octrees (ESVOs) observe that scene geometry can generally be represented well using a contour encoding [LK11]. Using contours allows early culling of the tree structure if the contour fits the original geometry well, but this can limit the attribute resolution (e.g., color). While it is possible to reduce the use of contours in selected areas, this choice also impacts the compression effectiveness drastically. Voxel attributes are compressed using a block-based DXT scheme, requiring one byte for colors and two bytes for normals per voxel on average. For high-resolution scenes, a streaming mechanism is presented.

Recently, Kämpe et al. observed that besides sparsity, *geometric redundancy* in voxel scenes is common. They proposed to merge equal subtrees in an SVO, resulting in a *directed acyclic graph* (DAG) [KSA13]. The compression rates are significant and the method was even used for shadow mapping [SKOA14, KSA15]. Nonetheless, the employed pointers to encode the structure of the DAG can become a critical bottleneck. *Pointerless SVOs* (PSVOs) [SK06] completely remove pointer overhead and are very well suited for offline storage. However, they do not support random access and cannot be extended to DAGs, as PSVOs require a fixed, sequential memory layout of nodes. While several reduction techniques for pointers have been proposed [LK11, LH07], they are typically not applicable to the DAG. These methods assume that pointers can be replaced by small offsets, but in a DAG, a node's children are not in order but scattered over different subtrees. Concurrent work presented a pointer entropy encoding and symmetry-based compression for DAGs, but does not support attributes [JMG16].

Adding voxel data reduces the probability of equal subtrees, making DAGs unsuitable for colored scenes. The recently proposed *Moxel DAGs* [Wil15] address this problem. In every node, they store the number of empty leaf voxels (assuming a complete grid) in the first child's subtree. During traversal, two running sums are kept – the number of empty leaves and total leaves – to compute a sequential unique index for every existing leaf voxel, with which the corresponding attributes are retrieved from a dense but uncompressed array. Our method is more efficient (with only one running sum) and requires less memory, as the number of empty leaf voxels grows to quadrillions for scenes like in Figure 2.1, leading to large storage requirements for the additional index per node. Furthermore, Moxel DAGs do not encode a multi-resolution representation and, hence, cannot directly be used for level-of-detail rendering.

Uncompressed voxel attributes quickly become infeasible for higher resolutions, especially on GPU architectures, where memory is limited. Here, *attribute compression* can be used. Specialized algorithms exist for textures [SAM05, NLP*12], colors (via effective quantization [Xia97]) or normals (*octahedron normal vectors* (ONVs) [MSS*10]). For the latter, careful quantization is necessary [CDE*14]. We decouple the geometry of a voxel scene from its attributes, which enables exploring such compression schemes.

2.3. BACKGROUND

A *voxel scene* is a cubical 3D grid of resolution 2^{N^3} with N a positive integer. Each voxel is either empty or contains some information, such as a bit indicating presence of matter, or multiple bits for normal or material data. SVOs encode these grids by grouping empty regions; each node stores an 8-bit mask denoting for every child if it exists – i.e., is not empty. A pointer connects the parent to its children, which are ordered in memory. Thus, 8 bits are needed for the childmask, plus a pointer of typically 32 bits. Furthermore, for level-of-detail rendering, parent nodes usually contain a representation of the children's data (e.g., an average color). If only geometry is encoded, the childmask gives sufficient information and no data entries are needed. Note that literature typically considers SVO nodes that are not leaves as voxels as well, so that reported voxel counts equal the number of tree nodes.

The DAG algorithm [KSA13] is an elegant method to exploit redundancy in a geometry SVO, and forms the basis of our topology encoding. For ease of illustration, Figure 2.2 uses a binary tree, but the extension to more children is straightforward. On the left, a sparse, colored, binary tree is shown. Dangling pointers refer to empty child nodes without geometry. We ignore the colors and numbers for now and only focus on the topology. The DAG is constructed in a greedy bottom-up fashion. Starting with the leaves at the lowest level, subtrees are compared and, if identical, merged by changing the parent pointers to reference a single common subtree. The DAG contains significantly fewer nodes than the SVO (Figure 2.2, middle-left). Note that for a DAG as well as an SVO, leaf nodes do not require pointers, and, when encoding geometry only, the leaves can even be stored implicitly by using the parent childmask.

One disadvantage of the DAG in comparison to an SVO is that pointers need to be

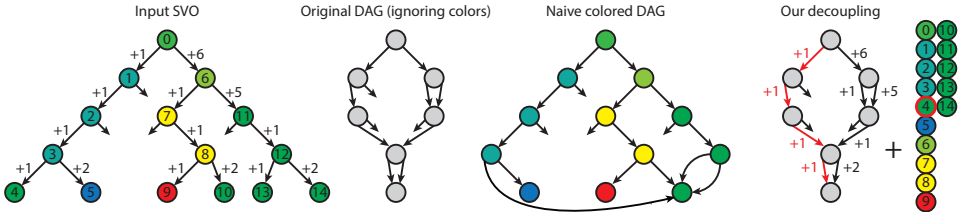


Figure 2.2: The input to our approach is an SVO with data (left). DAGs are only efficient when storing the topology (middle-left); when considering attributes, merging fails to compress the SVO sufficiently (middle-right). Our approach decouples data (colors in this case) from topology by storing offsets in the pointers, enabling us to apply the DAG principle on the geometry (right). The offsets then allow access to an attribute array, which is compressed independently. The red descent shows how the accumulated offsets deliver the correct array element.

stored for *each* child, because they can no longer be grouped consecutively in memory (in which case, a single pointer to the first child is sufficient). In practice, the 40 bits per node in a geometry SVO (8-bit childmask and a 32-bit pointer), become around $8 + 4 \times 32 = 136$ bits in a DAG – assuming a node has four children on average, e.g., for a voxelized surface mesh. The high gain of the DAG stems from the compression at low levels in the tree. For example, an SVO with 17 hierarchical levels usually has billions of nodes on the second-lowest level while a DAG has at most 256 – the amount of possible unique combinations of eight child voxels having each one bit. For higher levels, the number of combinations increases, which reduces the amount of possible merging operations; this also reflects the difficulty that arises when trying to merge nodes containing attribute data. With only three different data elements (colors of leaves), the merging process already stops after the lowest level (Figure 2.2, middle-right).

2.4. COMPRESSION

The possibility of merging subtrees is reduced when voxel attributes such as normals and colors are used. While the data usually exhibits some spatial coherence, exploiting it with a DAG is difficult because the attributes are tightly linked to the SVO's topology. We propose a novel mapping scheme that decouples the voxel geometry from its additional data, enabling us to perform specialized compression for geometry and attributes separately, which greatly amortizes the theoretical overhead caused by the decoupling.

Using our decoupling mechanism, which is described in Section 2.4.1, the geometry can be encoded using a DAG. The extracted attributes are stored in a dense *attribute array*, which is subsequently compressed. During DAG traversal, the node's attributes can efficiently be retrieved from the array. The attribute array itself is processed via a palette-based compression scheme, which is presented in Section 2.4.2. It is based on the key insight that the array often contains large blocks of similar attributes due to the spatial coherence of the data (e.g., a large meadow containing only a few shades of green). In consequence, using a local palette, the indices into this palette require much less memory than the original attributes.

While the original design for the palette compression is lossless, we show in Sec-

tion 2.4.3 that compression performance can be significantly improved by quantizing attributes beforehand. Hereby, a trade-off between quality and memory reduction is possible, which can be steered depending on the application. We demonstrate that significant compression improvements can already be achieved by using perceptually almost indistinguishable quantization levels.

Finally, we show in Section 2.4.4 that the DAG itself can also be further compressed using pointer and offset compression, as well as an entropy-based pointer encoding, which is a valuable addition to the original DAG method as well. These techniques greatly amortize the additional storage required for the decoupling.

2.4.1. VOXEL ATTRIBUTE DECOUPLING

To decouple data from geometry, we first virtually assign *indices* to all nodes in the initial SVO in depth-first order (Figure 2.2, left, the numbers inside the nodes). Next, for every pointer, we consider an *offset* (Figure 2.2, left, the positive numbers next to the edges), which equals the difference between the index of the child and parent associated with this pointer. Summing all offsets along a path from the root to a node then reproduces its original index.

Based on this insight, we propose to store these offsets together with each child pointer and to extract and store the node attributes in a dense *attribute array* in the same depth-first order (Figure 2.2, right, the stacked colors). During traversal from the root, a node's index is reconstructed via these offsets. This index can then be used to efficiently retrieve the corresponding voxel attribute from the array.

While our mapping introduces an overhead in the form of an additional offset for every child pointer, it has the benefit that subtrees with identical topology can be merged to a DAG again. In fact, a depth-first indexing automatically leads to identical offsets in geometrically identical subtrees. Further, we show in Section 2.5.3 that these offsets can be compressed very efficiently. Figure 2.2, right, illustrates an exemplary index retrieval from the resulting DAG for the node with index 4, where the red arrows denote the tree descent.

2.4.2. PALETTE COMPRESSION

After decoupling and storing the geometry in a DAG, we are left with an efficient representation of the topology, but the uncompressed attribute array still requires a large amount of memory. We propose a variable-length compression scheme for the attribute array, which is efficient and still allows for fast accessing at run time. To explain our method, we first describe the use of a global material array, making it possible to store indices instead of full attributes. Because of spatial coherence in the scene, consecutive indices will often be similar, which leads to the idea of working on blocks of entries in the attribute array. For each block, we define a palette (local index array) and each entry in a block only stores a local index into this palette. The palette then allows us to access the correct entry in the global material array.

Specifically, our approach works as follows. We denote the attribute array as $A = \{a_0, ..., a_{\Lambda-1}\}$, where Λ is the total number of entries. Note that Λ equals the voxel count in

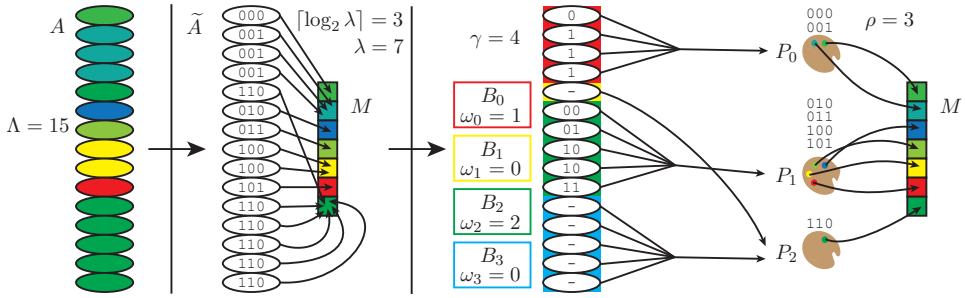


Figure 2.3: Palette compression. From left to right: the initial attribute array $A = \{a_0, ..., a_{14}\}$ stores 24-bit colors; we construct the material array $M = \{\widetilde{a}_0, ..., \widetilde{a}_6\}$ to store the 24-bit colors while $\widetilde{A}$ contains 3-bit indices into M; four blocks $\{B_0, B_1, B_2, B_3\}$ are created, containing 0-bit to 2-bit palette indices into the three associated palettes $\{P_0, P_1, P_2\}$, which in turn contain 3-bit material indices into M.

the original SVO. We observe that A usually contains many duplicates and the number of unique voxel attributes λ is typically orders of magnitude smaller than Λ. For this reason, a first improvement is to construct a material array $M = \{\widetilde{a}_0, ..., \widetilde{a}_{\lambda-1}\}$, which stores all λ unique attributes in the scene, and replace A with an indexed version pointing into M. We denote the index array as $\widetilde{A} = \{m_0, ..., m_{\Lambda-1}\}$, where m denotes an index into M. Since indices require fewer bits than attributes, it usually results in a reduced memory footprint and decouples the content of the material array from the attribute array. An example is provided in Figure 2.3.

Since the data in A is in depth-first order, we retain most of the spatial coherence of the original scene. Consequently, if a large area exhibits a limited set of attributes (e.g., a blue lake represented by millions of blue voxels with little variation) they are likely to be consecutive in A. Hence, it would be beneficial to partition the attribute array into multiple *blocks* of consecutive entries, where each only contains a small number of different indices. We describe how to determine these blocks later.

Each block has an associated *palette*, which is an array of the necessary unique indices into the material array to retrieve all attributes in the block. The block itself only stores (possibly repeating) indices into its associated palette. While each index in a block originally requires $\lceil \log_2 \lambda \rceil$ bits, it is now replaced by a new index with only ω bits, where ω depends solely on the number of unique entries inside the block. Note that there is no one-to-one correspondence between palettes and blocks; a palette can be shared by several blocks, but each block is linked to a single palette only.

Blocks have a variable length, which makes it necessary to keep a block directory to indicate where blocks start and what their corresponding palette is. The block directory has its entries ordered by the starting node index, which makes it possible to perform a binary search to find the corresponding block information given a node index. Generally, the memory overhead of the directory is negligible.

Our representation ultimately consists of an array of blocks $\{B_0, ..., B_{\gamma-1}\}$ and an array of palettes $\{P_0, ..., P_{\rho-1}\}$, where γ and ρ denote the total number of blocks and palettes,

respectively. For the example in Figure 2.3, it can be seen that we obtain three palettes and four blocks (i.e., $\rho = 3$, $\gamma = 4$), because B_1 and B_3 use an identical palette that does not have to be stored twice.

Algorithm 1 Palette compression

```
1: function FINDLARGEBLOCKS({m_i,...,m_j})
2:     if j < i then return
3:     ω ← 0
4:     while ω < 4 do
5:         {m_k,...,m_l} ← largest block with 2^ω unique m
6:         B ← {m_k,...,m_l}
7:         if MEMORY(B,ω) < (l − k + 1) · (ω + 1) then
8:             P ← CREATEPALETTE(B)
9:             for all m ∈ B do m ← index into P
10:            FINDLARGEBLOCKS({m_i,...,m_{k−1}})
11:            FINDLARGEBLOCKS({m_{l+1},...,m_j})
12:            return
13:        else
14:            ω ← ω + 1
15:    FINDREMAININGBLOCKS({m_i,...,m_j})
16: function FINDREMAININGBLOCKS({m_i,...,m_j})
17:    if j < i then return
18:    ω ← {0,...,8}
19:    for all ω do
20:        {m_i,...,m_{k_ω}} ← largest block with 2^ω unique m from m_i
21:        B_ω ← {m_i,...,m_{k_ω}}
22:        S_ω ← MEMORY(B_ω,ω) / (k_ω − i + 1)
23:    B,k ← B_ω,k_ω with minimal S_ω
24:    P ← CREATEPALETTE(B)
25:    for all m ∈ B do m ← index into P
26:    FINDREMAININGBLOCKS({m_{k+1},...,m_j})
27: function MEMORY({m_i,...,m_j},ω)
28:    return (j − i + 1) · ω + 2^ω · ⌈log_2 λ⌉ + size(directory entry)
```

Palette Selection Finding the optimal set of blocks with respect to their memory requirement is a hard combinatorial problem, and the attribute array contains billions of entries for high-resolution scenes. Hence, we propose a greedy heuristic to approximate the optimal block partitioning.

The algorithm consists of two phases (see Algorithm 1). First, we greedily find the largest blocks that only require a few bits per entry, as these blocks form the best opportunities for high compression rates. This first phase takes a consecutive subset of $\widetilde{A}$ as its parameter, and is initially invoked for the complete array ($\{m_i,...,m_j\}$ with $i = 0$ and $j = \Lambda - 1$). It finds the largest block that appears in this set consisting of 2^ω unique material indices in a brute-force fashion (line 5). Since we start with $\omega = 0$ (line 3), it first finds the largest consecutive block with only one unique index. If the total overhead introduced by creating a palette is outweighed by the memory reduction (line 7), we generate a palette (if we could not find an existing matching palette) and replace the material indices m with indices into this palette (lines 8 and 9). The remainder of $\widetilde{A}$ is then processed recursively (lines 10 and 11). If the criterion is not satisfied, we increment

ω and repeat (line 14). When ω becomes too large, we stop the first phase, as finding the largest block becomes computationally infeasible. In our case, we terminate for $\omega \geq 4$, corresponding to 16 unique indices or more (line 4).

The second phase is invoked for the data that could not be assigned to blocks in phase one (line 15) which is now partitioned into blocks sequentially. For this, nine possible blocks (for each $\omega = \{0,...,8\}$) are considered, all starting at m_i (line 20). Of these nine blocks, the one with the minimal memory per entry (including directory overhead) is used (line 23), and a palette is attributed to this block, after which we replace the indices again (lines 24 and 25). This is repeated for the remaining data (line 26). To compute a block's memory overhead (line 28), we multiply the block entries by the bits required for a palette index ($(j-i+1)\cdot\omega$) and add the palette entries multiplied by the bits required for a material index ($2^\omega \cdot \lceil \log_2 \lambda \rceil$). Finally, we add the block's directory overhead.

For the example in Figure 2.3, only B_3 is created in phase one, as other possible blocks do not satisfy the memory criterion (line 7). The remaining data is processed in phase two, which results in three additional palettes, one of which can be shared.

2.4.3. ATTRIBUTE QUANTIZATION

The palette-based compression scheme for the attribute array is lossless and can provide a significant reduction in memory. However, since human perception is not as flawless as a computer's, and many scenes exhibit similarity in voxel attributes, we can apply a certain degree of *quantization* on many kinds of attributes without losing much visual quality. This can greatly improve the compression capability of our proposed approach.

In principle, any standard quantization could be applied to the attribute array, but specializing the method based on the data type leads to improved results. In particular, we present solutions for colors and normals, as they seem most valuable to be supported for voxel scenes. Detailed scenes can potentially result in millions of different colors with small variations in the attribute array. Fortunately, color quantizers can reduce the amount of distinct values significantly without resulting in perceivable differences [Xia97]. While Xiang's original method relied on a clustering in a scaled RGB space, we improve the result by working in the (locally) perceptually uniform CIELAB color space. The amount of colors can be freely chosen by the user; we typically use 12-bit (4096) colors throughout this chapter. Note that the method is a data-driven clustering and requires preprocessing to analyze the colors, but yields high-quality results even for a small amount of colors.

For normals, we rely on *octahedron normal vectors* (ONVs), leading to an almost uniformly distributed quantization [MSS*10, CDE*14]. Using ONVs is beneficial as it yields higher precision for the same number of bits compared to storing one value per dimension. Again, the bit depth of the quantization can be freely chosen.

2.4.4. GEOMETRY COMPRESSION

By using an attribute array, we still have to encode additional offsets in the DAG structure, which increases its size. We propose to reduce the DAG's memory consumption by compressing the introduced offsets, as well as the child pointers, which typically make

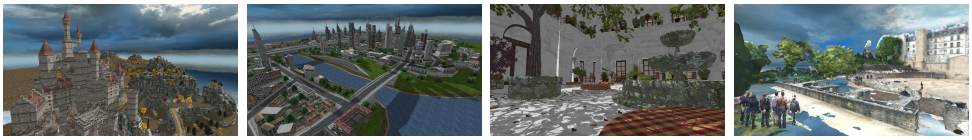


Figure 2.4: Datasets used for evaluation. From left to right: *Citadel*, *City*, *San Miguel* and *Arena*.

up a large part of the total memory usage.

Offset Compression We observe that the offset from a node to its first child is always +1 (see Figure 2.2), implying that this offset can be stored implicitly. Further, offsets are typically small in the lower levels of the tree due to the depth-first assignment. Hence, fewer bits are required to represent the offset. To this extent, we analyze each level and find the minimum number of bits required to encode offsets in this level. We round up to bytes for performance reasons, as a texture lookup on the GPU retrieves at least a single byte. In practice, a two-byte offset is sufficient for the lowest five levels in all our examples, leading to a significant improvement. Four or even five bytes are still required for offsets on the highest levels, but these represent much fewer nodes ($\approx 0.1\%$), which makes the increased memory usage non-critical.

Pointer Compression We apply the same compression technique as for the offsets to the child pointers as well. While this leads to a slight improvement, the compression does not work as well as for offsets, since the levels that contain most pointers generally require the full four bytes per pointer. However, we observe that some subtrees are used significantly more often than others, which makes entropy encoding [BRGIG*14] a well suited candidate for memory reduction. We create a table of the most common pointers per level – much in the spirit of our material array in Section 2.4.2 – which is sorted by occurrence in descending order. In the DAG, we then store only an index into the pointer table, which is usually smaller than the original pointer and can be represented with fewer bits.

In practice, we found the following setup to be most effective: each pointer is initially assumed to be one byte. Its first two bits store the type, which then indicates the pointer's actual bit length. Two bits can encode four types; the first three are used to indicate if 6, 14, or 22 bits are used to encode a pointer into the lookup table, and the last type is reserved to indicate that the remaining 30 bits correspond to an absolute pointer (as before, this ensure multiples of bytes). The latter could also be increased to 46 bits, but 30-bit pointers proved sufficient for the DAG nodes in all our examples. While we achieve significant compression with the entropy encoding, it does decrease the performance, as evaluated in Section 2.5.

2.5. RESULTS

Our method aims at large sparse navigable scenes. For evaluation, we choose a set of very distinct datasets deliberately (see Figure 2.4): architectural structures (the *Citadel* and *City* scenes); complex geometry (tree and plants in the *San Miguel* scene); and a

3D model obtained from real-life photographs using *floating scale surface reconstruction* [FG14] (*Arena* scene), which is noisy, contains diverse colors, and is a good test case for a realistic dataset. The datasets were produced by voxelizing triangle meshes through depth peeling [Eve01], using the standard extension proposed by Heidelberger et al. [HTG03]. While our compression schemes can handle any spatially coherent data, in practice, we evaluate our method using colors and normals, which are crucial for realistic lighting. We define the compression rate as the memory size of the compressed data over that of the uncompressed data, expressed as a percentage.

In the following, we discuss the results of our compression components separately as described in Section 2.4. Starting with the palette approach, we then analyze the gain of lossless and lossy compression, and present the results of our offset and pointer compression. Next, we compare our approach to existing techniques. Finally, we present results to illustrate the performance of our method and discuss its properties before showcasing several application scenarios.

2.5.1. DECOUPLING AND PALETTE COMPRESSION

We show statistics for the DAG-based geometry encoding and the attribute array for our four test scenes in Table 2.1. We list the number of DAG voxels in millions, as well as the memory footprints in MB of the standard and offset-augmented version of the DAG, which is needed to decouple geometry and attributes. The additional offset and pointer compression is analyzed in Section 2.5.3.

Table 2.1: Decoupling and palette compression. The numbers are computed for a $64K^3$ resolution (16 hierarchical tree levels) using non-quantized, 24-bit colors for the attributes.

	Scenes			
Geometry	*Citadel*	*City*	*San Miguel*	*Arena*
DAG voxels (M)	18.3	10.2	18.8	34.7
DAG size (MB)	382	207	395	737
With offsets (MB)	693	374	719	1342
24-bit colors				
Λ (M)	4760	10487	14788	3263
λ (M)	1.66	1.42	3.15	1.57
A (MB)	13619	30004	42309	9336
$\widetilde{A}+M$ (MB)	11922	26257	38792	8174
PC (MB)	10124	24051	10877	2220
Compression rate	74%	80%	26%	24%

Further, Table 2.1 shows the number of attributes Λ, which equals the SVO node count, and the number of unique attributes λ, both in millions. The memory size in MB of the attribute array (indicated as A) exceeds that of the DAG by far. Still, λ is usually much smaller than Λ; indeed, the memory cost of the attribute array can already be decreased by using a material array (indicated as $\widetilde{A}+M$). Using our palette compression (indicated as PC) reduces the cost again; we report the compression rate by comparing

to the original attribute array A. While overall significant, the use of a lossless scheme seems overly conservative in most practical scenarios and implies that even very similar attributes will have to be represented individually. By allowing for a slightly lossy quantization, the attribute costs can be reduced significantly.

2.5.2. ATTRIBUTE QUANTIZATION

Table 2.2: Attribute quantization memory footprints and quality. The numbers are computed for a $64K^3$ resolution using 24-, 14- and 12-bit colors, and for a $32K^3$ resolution using 32- and 12-bit ONVs.

	Scenes			
24-bit colors	*Citadel*	*City*	*San Miguel*	*Arena*
A (MB)	13619	30004	42309	9336
14-bit colors				
PC (MB)	3645	8163	4495	656
Mean RGB err.	2/1/2	2/1/2	2/2/2	2/1/2
Max. RGB err.	20/4/5	6/7/7	9/26/11	34/5/4
Mean ΔE	1.05	0.91	0.80	0.90
Max. ΔE	2.57	2.44	3.00	2.36
Comp. rate	27%	27%	11%	7.0%
12-bit colors				
PC (MB)	2609	5703	3099	438
Mean RGB err.	3/2/3	3/2/3	3/2/3	3/2/3
Max. RGB err.	10/7/29	11/12/11	11/9/25	48/5/3
Mean ΔE	1.72	1.60	1.47	1.75
Max. ΔE	4.25	3.85	5.28	4.38
Comp. rate	19%	19%	7.3%	4.7%
32-bit ONVs				
A (MB)	4496	9994	14081	3110
PC (MB)	912	417	2678	2151
Comp. rate	20%	4.2%	19%	69%
12-bit ONVs				
PC (MB)	135	85.9	407	464
Mean/max. err.	2°/8°	2°/8°	2°/8°	2°/8°
Comp. rate	3.0%	0.9%	2.9%	14%

Data quantization might impact precision, but leads to an often similar appearance and a large memory benefit. In Table 2.2, we show the size of the attribute array for 24-bit colors again, and the drastic memory gain of our result using palette compression and quantized colors (14 and 12 bits). To assess the fidelity of our quantization, we report the mean absolute error for each RGB channel over all voxels, as well as the maximum deviation. However, since these numbers do not always give a good impression of perceptual quality, we further report mean and maximum ΔE-values as defined by the CIE94 standard. We use $k_L = 1$, $K_1 = 0.045$ and $K_2 = 0.015$ and the D65 illuminant as the reference white, as per graphics industry standards [Kle10]. Finally, we show the compression rates

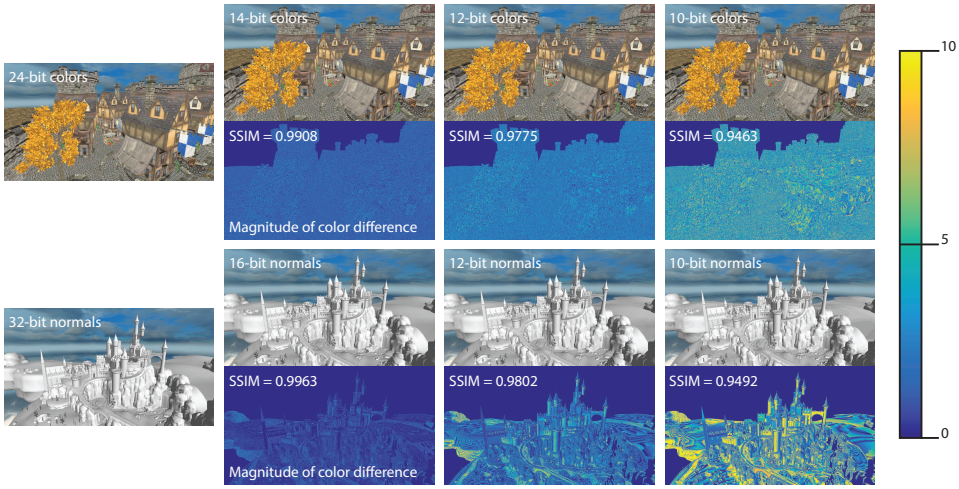


Figure 2.5: Perceptual quality of our color and normal quantization. We show the quantized result for 14-bit, 12-bit and 10-bit colors, with their corresponding magnitude of the color difference (i.e., $\sqrt{dR^2 + dG^2 + dB^2}$) per pixel. We do the same for quantized normals, showing the results for 16-bit, 12-bit and 10-bit normals. The difference values are mapped using the color map on the right, where a difference of 10 corresponds to a bright yellow color. We further report SSIM values for each image to assess the perceptual similarity [WBSS04].

obtained with our palette approach for quantized colors.

To illustrate the impact during rendering, Figure 2.5 shows images from two viewpoints in the *Citadel* scene. These exhibit many unique values, as well as color and normal gradients, which represent a difficult case for quantization. We provide SSIM values (a perceptual similarity metric, where SSIM = 1 means identical) for every image [WBSS04], comparing the result to its non-quantized counterpart. We note that 14-bit colors produce very good results, and even for 12-bit colors the only indication of quantization is the presence of minor banding artifacts at some locations. For 10-bit colors the quality is reduced, as evidenced by the color difference image, but the result is still relatively close to the reference.

Similarly, we report memory footprints for normals in Table 2.2. We consider a $32K^3$ resolution with 32-bit ONVs as a reference, since 96-bit normals at 15 levels could not be handled by our hardware, and the mean error for 32-bit ONVs compared to regular 96-bit normals is only 0.001°, with a theoretically proven maximum error below 0.004°. For quantization, we use 12-bit ONVs, for which we report mean and maximum errors in degrees and show the attained compression rates. As expected, normals compress better than colors for scenes that contain many aligned surfaces, like the *City* scene. Visually, 16-bit normals produce results indistinguishable from the 32-bit reference while 12-bit and 10-bit normals produce minor and more visible banding on smoothly varying surfaces, respectively (see Figure 2.5). Nonetheless, for diffuse shading, such artifacts are barely perceivable and even 10 bits may suffice. For effects like specular reflections, 16-bit normals are preferred.

Table 2.1 and 2.2 show that the memory footprint of the attributes is now potentially compressed to a similar order of magnitude as the geometry. We can see that the combined use of quantization and our palette compression is very fruitful in practice.

Furthermore, the combination of colors, normals or even reflectance information rarely leads to a linear increase of memory. For example, the night-time version of the *City* scene at a $32K^3$ resolution (Figure 2.8, left) uses 12-bit colors, 10-bit normals, and 8-bit reflectance information. The total memory footprint is 1492 MB, compared to 1186 MB for just encoding colors. This means that the normals and reflectance information yield a 25.8% overhead, even though the voxel data grew by 150%. This outcome is a consequence of materials with similar colors often having similar normals and reflectance values as well (e.g., a roughly uniformly colored wall).

2.5.3. OFFSET AND POINTER COMPRESSION

To evaluate our geometry compression, we compare the influence of all our offset and pointer optimizations separately. In Table 2.3, we reiterate the memory footprint of the standard offset-augmented DAG (as in Table 2.1). We then report results for implicitly storing the first child offset; the per-level byte-precise offset compression; the per-level byte-precise compression for pointers; 8-bit childmasks (the original DAG uses 24 bits of padding); pointer entropy encoding; and, finally, a combination of all these techniques, for which the shown compression rate compares to the standard offset-augmented DAG. We can see that our approach is quite effective, as we observe that the final memory footprint is on par or even less than the memory cost of the original DAG without the offsets (see Table 2.1).

Table 2.3: Offset and pointer compression for a $64K^3$ resolution.

	Scenes			
DAG size (MB)	*Citadel*	*City*	*San Miguel*	*Arena*
Uncompressed	693	374	719	1342
Implicit offset	623	335	647	1210
Offset compression	499	271	505	907
Pointer compression	629	330	623	1228
8-bit childmask	641	345	665	1243
Pointer entropy	543	290	531	1027
Combined	348	186	316	591
Compression rate	50%	50%	44%	44%

2.5.4. COMPARISON

Now that we have discussed all components, we can compare our compression scheme to existing techniques. As we use 12-bit colors for the comparison, the memory footprint of our complete data structure now equals the geometry size for the combined methods in Table 2.3 plus the attribute size for 12-bit colors in Table 2.2. We compare the cost per voxel of our approach to four other techniques; SVOs, PSVOs [SK06], ESVOs [LK10], and CDAGs (naively adding colors to the original DAG [KSA13]).

For the standard SVO implementation, the memory footprint is computed as follows: we have an 8-bit childmask, a 32-bit pointer, and a 12-bit color value for every node – note that the leaf nodes have no childmask or pointer. The PSVO contains exactly the same data in every node, except for the child pointer. Besides voxel attributes, ESVOs store additional contour data, but also make use of compression. For color, a DXT1 compression is used while normals are compressed using a novel scheme, which is also lossy, but provides up to 14 bits of precision per axis. In this section, we report ESVO memory footprints as obtained by using the implementation supplied by the authors, which makes use of the aforementioned attribute and contour-based compression. Finally, we have the original DAG [KSA13], augmented with color data, so that every node contains a 32-bit childmask, one to eight 32-bit pointers, and a 12-bit color value (CDAGs).

For a direct comparison of our attribute compression to that used by ESVOs, we built the Sibenik scene at a $8K^3$ resolution. Here, ESVOs reported a memory footprint of 2120 MB without using contours [LK10]. Not using contours is important, as, contrary to geometry, a similar quality as regular colored SVOs can only be achieved for attributes if they are not cut off during traversal. Further, as the data quality for ESVOs is not evaluated, it is difficult to provide a comparison; hence, we use 24-bit colors and 32-bit ONV normals for the palette compression, which ensures better quality than the lossy schemes applied by ESVOs. Our palette compression is more flexible when compared to the constant DXT1 rate, which results in only 1171 MB for our entire data structure.

Table 2.4: Comparison to the state of the art for a $64K^3$ resolution.

	Scenes			
	Citadel	*City*	*San Miguel*	*Arena*
SVO voxels (M)	4760	10487	14788	3263
Size (MB)	13285	27966	39122	9520
Bytes/voxel	**2.93**	**2.80**	**2.77**	**3.06**
PSVO size (MB)	8105	17595	24748	5638
Bytes/voxel	**1.79**	**1.76**	**1.75**	**1.81**
ESVO voxels (M)	1533	2782	1168	
Size (MB)	10374	18506	8174	
Bytes/voxel	**2.29**	**1.85**	**0.58**	
CDAG voxels (M)	286	629	251	117
Size (MB)	5540	11922	4791	2326
Bytes/voxel	**1.22**	**1.19**	**0.34**	**0.75**
Our voxels (M)	18	10	19	34
Geometry (MB)	348	186	316	591
Attributes (MB)	2609	5703	3099	438
Total size (MB)	2957	5889	3415	1029
Bytes/voxel	**0.65**	**0.59**	**0.24**	**0.33**
Compression rate	22%	21%	8.7%	11%

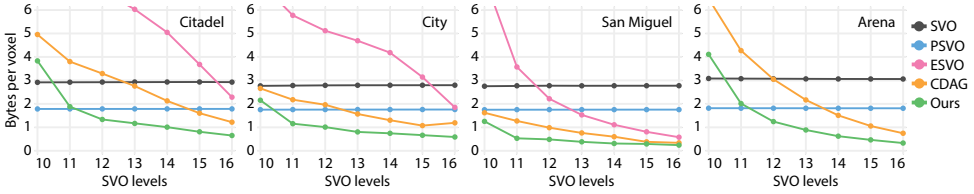


Figure 2.6: Memory usage per voxel for our test scenes at different SVO levels. We compare our approach to a colored SVO, PSVOs [SK06], ESVOs [LK11] and a naive colored DAG implementation (CDAGs). Note that the ESVO implementation was unable to load the *Arena* scene.

Figure 2.6 and Table 2.4 illustrate that our approach outperforms other methods by a significant margin. We report bytes per voxel for all techniques, *always* with reference to the SVO node count. We list the actual voxel count in millions for SVOs, ESVOs, CDAGs, and our method separately. We show the geometry and attribute size separately and combined for our method. Finally, we report compression rates as compared to a standard SVO implementation.

2.5.5. PERFORMANCE

Construction As our focus was mostly on compression quality, not performance, we did not investigate significant acceleration techniques for the DAG algorithm, nor for our palette compression. Still, the construction times for building our data structure are interesting, as they illustrate the computational overhead of involving attributes. In Table 2.5, we report timings on an i5 CPU in minutes for both standard colored DAGs (before the slash) and our method (after the slash); for the latter, we see an increase up to an order of magnitude. Still, it is feasible to compute high-resolution scenes on commodity hardware, and the slow construction does not hurt performance during rendering. The construction time depends mostly on the number of compression attempts that our algorithm explores; if large blocks are already found in the first phase, as for the *Arena* scene, the cost of the palette compression is significantly reduced.

Table 2.5: Construction times in minutes at different resolutions for our four test scenes, using the naive colored DAG implementation and our decoupling and palette compression, with 12-bit colors.

	Scenes			
Resolution	*Citadel*	*City*	*San Miguel*	*Arena*
$4K^3$	1.60/2.58	0.75/6.01	1.40/5.31	1.08/1.83
$8K^3$	2.65/17.1	2.93/30.3	5.90/22.0	3.13/7.35
$16K^3$	10.8/64.5	13.2/217	20.0/157	10.0/21.3

Rendering We did not particularly optimize our rendering algorithm; in each frame, we cast rays from the camera and traverse the SVO with a standard stack-based approach to find the first intersecting voxel that projects to an area smaller than a pixel. To still demonstrate that our method is capable of real-time performance, Figure 2.7 shows timings for a walk-through in the *Citadel* scene in full HD at a $32K^3$ resolution, obtained

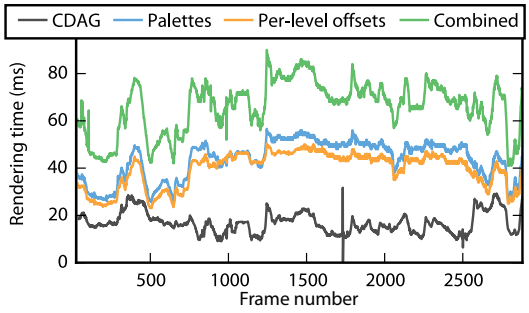


Figure 2.7: Rendering times while navigating through the *Citadel* scene at a $32K^3$ resolution, obtained by raycasting in full HD.

using an NVIDIA GeForce GTX 980 Titan X. We compare the rendering times for palette compression, per-level byte-precise offsets, and using all our compression techniques, to naive colored DAGs (CDAGs). We can conclude that palettes and offset compression have some impact on the performance, but still enable real-time rendering while yielding significant compression rates. The entropy encoding on the other hand has a bigger influence and we only achieve interactive rates. Still, it can be useful for memory gain, especially when the geometry is relatively large, like for the *Arena* scene (see Table 2.4). Further, the rendering cost is several orders of magnitude lower than for pointerless solutions, while still avoiding high memory costs.

2.5.6. APPLICATIONS

To demonstrate the versatility of our approach, and of SVOs in general, we showcase several applications. Like the original DAG, we are able to obtain high-resolution hard shadows for the whole scene. With normals, however, we can look into more interesting applications, such as reflections, by shooting secondary rays while maintaining real-time performance (Figure 2.8, left).

We have also implemented a simple method for color bleeding from single-bounce global illumination (Figure 2.8, middle). We shoot multiple secondary rays via stratified sampling of the hemisphere – which means the samples are uniformly distributed, but contain a random offset – and shoot tertiary rays from the intersecting voxels to determine if they are in shadow. We attain interactive rates with 8 secondary rays per pixel.

Since our method, like the DAG, exploits both similarity and sparsity, we can to some extent compress dense data as well (Figure 2.8, right). For the shown *Christmas Tree* scene, we are able to obtain a lossless compression rate of 38.6% when comparing our data structure to the original input file, which is approaching state-of-the-art methods for dense datasets (29.4%) [GWGS02]. When applying a filtering to remove scanning noise in the air, we additionally profit from the sparsity and achieve rates below 10%.

2.6. CONCLUSIONS

We have presented a novel SVO compression scheme, which relies on the decoupling of geometry from additional voxel data. Our mapping is efficient and introduces little over-

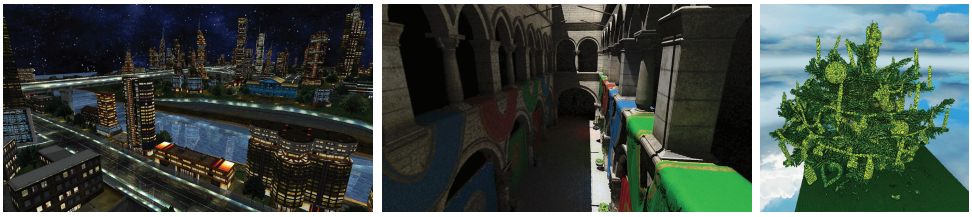


Figure 2.8: Several applications of our compressed SVO. From left to right: encoding reflectance information in materials for the *City* scene, rendered at a resolution of $32K^3$; color bleeding in the *Sponza* scene, at an SVO resolution of $4K^3$, using 16 samples per pixel, and secondary and tertiary ray tracing at a 512^3 resolution; rendering of the dense *Christmas Tree* dataset at a $512 \times 512 \times 999$ resolution.

head, enabling separate compression methods for topology and voxel attributes. Furthermore, we introduced compression schemes for child pointers, which also reduces the cost of traditional DAGs. For attribute compression, we proposed a combination of quantization and our lossless palette approach, implicitly exploiting spatial coherence.

We showed that our solution reduces memory usage from 4.49 times (for the *Citadel* scene) up to 11.5 times (for the *San Miguel* scene) compared to standard SVO implementations. Our method outperforms state-of-the-art SVO compression methods for all test scenes. The high compression rates allow us to store colored SVOs with up to 17 levels (a voxel resolution of $128K^3$) completely on the GPU.

We demonstrated real-time rendering performance using commodity hardware and showcased several applications such as color bleeding and reflections, for which additionally normal and reflectance attributes were encoded. For future work, investigating advanced material properties for the voxel data (e.g., BRDFs encoded via spherical harmonics, or transparency) is an interesting direction.

3

MegaViews: Scalable Many-View Rendering with Concurrent Scene-View Hierarchy Traversal

Every flight begins with a fall.

George R. R. Martin

In this chapter, we present a scalable solution to render complex scenes from a very large amount of viewpoints. While previous approaches rely either on a scene or a view hierarchy to process multiple elements together, we make full use of both, enabling sublinear performance in terms of views and scene complexity. By concurrently traversing the hierarchies, we efficiently find shared information among views to amortize rendering costs. One example application is many-light global illumination. Our solution accelerates shadow map generation for virtual point lights, whose number can now be raised to over a million while maintaining interactive rates.

An extended version of this chapter was published in Computer Graphics Forum (2018), by Timothy R. Kol, Pablo Bauszat, Sungkil Lee and Elmar Eisemann [KBLE18]. As for the distribution of work, I implemented everything, devised the hierarchical representations, the culling, view equivalence and subdivision strategies, as well as the hole filling and fusion of renderings. Furthermore, I wrote most of the paper.

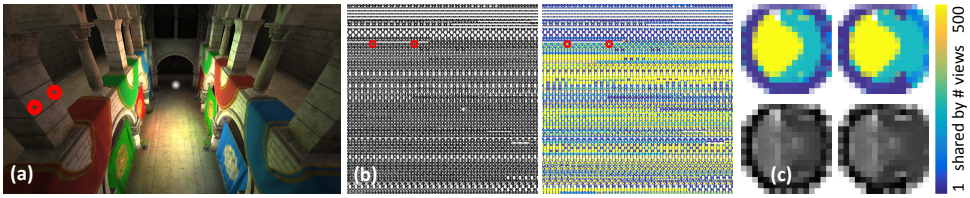


Figure 3.1: Indirect illumination computed from $1M$ animated virtual point lights (VPLs) with shadow maps of 16×16 resolution generated at interactive rates (100 ms out of 194 ms in total) by our many-view rendering algorithm (a). We show shadow maps of a subset of 2048 VPLs, for which many pixels are shared and rendered only once for multiple views (b). We highlight two close VPLs in (a) and (b), which can share a large part of their rendering (c). We note that faraway pixels are logically shared by more views.

3.1. INTRODUCTION

In the previous chapter, we have presented a solution for storing virtual worlds. The next step is to produce a *visual representation*. While displaying complex scenes is already difficult for a single viewpoint, in this chapter we aim for *realistic* rendering, which adds another dimension to the problem. Recent work has shown that producing *multiple* views simultaneously can be very beneficial for realistic rendering [DKH*14]. For example, when many light sources are present in a scene, each requires its own shadow map. Similarly, indirect illumination can be well approximated when first distributing *virtual point lights* (VPLs), each illuminating the scene [Kel97, WFA*05, HPB07]. Also, reflective objects can be simulated by creating cube maps from various locations on the surface [BN76, SKALP05, HREB11]. Unlike typical multi-view rendering, such as stereoscopy, soft-shadow mapping, and motion or defocus blur [ABC*91, CPC84, HA90], indirect lighting scenarios show less coherence among the views. Furthermore, the number of views has to be high to ensure a convincing quality, while maintaining a high framerate for interactive applications. We here address this many-view rendering problem.

The use of a hierarchy is the most common way to obtain sublinear rendering scalability. Coarse representations [RGK*08] or scene hierarchies are widely used [LWC*03]. For each view, an adequate *level of detail* (LOD) can be chosen, typically represented by a cut through the hierarchy that determines the nodes whose content will be rendered. However, the use of only a scene hierarchy does not scale well with the number of views. The cost per view is reduced, but the total cost stays linear in the amount of viewpoints.

MegaViews is a novel scalable many-view rendering algorithm. It provides sublinear performance on both the scene complexity and number of views. The idea is to rely on two hierarchies; one on the scene and one on the views. We concurrently traverse both hierarchies, with pairs of scene and view nodes fed into the double traversal. This way, we can exploit coherence among different views, which enables us to employ early culling techniques, as well as shared rendering, when possible. Our solution is well adapted to GPUs and achieves interactive rates for a large amount of views (we demonstrate a million 16×16 views) in complex scenes on standard hardware. We show the benefit of our solution in several applications, including many-light global illumination [Kel97]. The major contributions of this chapter can be summarized as:

- a scene-view hierarchical representation;

- an efficient traversal method;
- a shared rendering solution; and
- many-light applications using our approach.

3.2. RELATED WORK

Level-of-detail representations can reduce the rendering workload per view and are a well explored area. There are many surveys [DFKP05] and books [LWC*03] on this topic, and we refer the interested reader to this literature. Here, we will discuss only approaches closer to our work that amortize costs over several views.

Rendering many views is required for devices like stereoscopic displays [ABC*91]. Realistic rendering also benefits from many views over time, lenses, and area or volume lights [CPC84, HA90]. For some of these problems either a small number of views is sufficient, or they follow a certain regular pattern, leading to many approaches that exploit this predictable consistency [Hal98, HAM06, LES10]. Nevertheless, other scenarios show less coherence. For example, indirect illumination requires rendering thousands of relatively random views, making it much harder to propose an efficient solution [HPB07].

One relatively direct way of handling many views is the use of *imperfect shadow maps* [RGK*08]. Here, the scene is sampled and the points are distributed randomly over all views. In this way, the rendering time is independent of the number of views, but the quality becomes increasingly worse if the sampling rate is not increased. The approach relies on hole filling to complete the sparse images [MKC07]. A key insight is that low-resolution shadow maps tend to work well for low-frequency indirect lighting, and even imperfections do not necessarily create visible artifacts. We build upon these insights to share rendering between views in our work.

Other approaches [REG*09, Chr08] deal with many views of a scene. Here, each produced image is mostly accurate. The approach relies on a scene hierarchy that is traversed for each view individually. While the solution is well suited for mapping it onto the GPU [REG*09], the workload distribution is not optimal, as each view can take a very different path through the hierarchy. *ManyLoDs* [HREB11] build upon this insight and enforce a traversal that takes one step at a time. All node-view pairs have the same cost per iteration, which renders the approach much more efficient on modern GPUs. Nevertheless, the cost remains linear in the number of views or VPLs.

To reduce the workload further, there are attempts to reduce the number of VPLs or cluster their contributions. *Lightcuts* cluster VPLs, defining a cut through a light hierarchy [WFA*05, WABG06]. The number of VPLs can also be reduced by choosing an effective subset [GS10, REH*11]. The effect of VPLs is also the basis of *matrix row-column sampling* (MRCS), which sparsely samples combinations of senders (light sources) and receivers (scene) via shadow maps, organized in a matrix [HPB07]. This solution can be combined with lightcuts [OP11] and extended to animated scenes [HVAPB08], as the sparse view evaluation by itself leads to flickering. Nevertheless, the involved matrix analysis is often too costly for real-time performance. Furthermore, their goal is to

choose a low number of good views, while we actually consider many views.

Light culling and selection is also used by screen-space-clustering methods, linked to tiled shading [OA11, OBA12, HMY12]. Views are then produced for each tile instead of each light in the form of a cube map that can be organized into a tiled virtual shadow map, which facilitates resolution optimizations [OSK*14, OBS*15]. Nevertheless, the method is mostly limited to light gathering, as no actual renderings are produced for the VPLs. Furthermore, the performance gain depends on the effectiveness of the employed resolution heuristics, which can overestimate. Tiled methods usually build upon a cutoff of the VPL influence in screen space. However, this leads to lower quality compared to randomized sampling [TH16], which benefits from higher resolution shadow maps and a shadow map per VPL. Our solution can produce many shadow maps and is more general in terms of view placement and the choice of resolution.

Image-space clustering is also employed in *point-based global illumination* (PBGI), where tiles are repartitioned using a k-means clustering [WHB*13]. Assuming coherence of grouped pixels, a baseline cut through the scene hierarchy is established per tile. This cut is rendered into a texture, which is shared per tile. It is then refined per cluster and new views are stored. The performance gain lies in the incremental cut refinement [HREB11], which requires additional memory, and the shared map. Nonetheless, sharing information in this way can lead to artifacts if a cluster covers a large extent of the scene and depth fusion can be incorrect. Furthermore, at least one full traversal is performed per tile; the costs per generated view, hence, remains linear in the number of views. Our solution handles arbitrary views and lowers the rendering cost.

Furthermore, ray-space hierarchies and their traversal have been extensively used in conjunction with object-space hierarchies, including impostor placement [JWSP05], ray tracing [RAH07], potential visibility sets [MBWW07], ray-packet reordering [BWB08], and coherent hierarchical culling [MBJ*15]. They commonly achieve high efficiency by addressing the joint double-hierarchy traversal with different subdivision criteria (render cost, memory cost, distance, and visibility) on the ray-object pair subdivision. However, they focus more on ray intersections instead of rendering complete views. We extend the previous approaches beyond ray groups to rendering for multiple object-space viewpoints, where our shared rendering leads to high efficiency in terms of both rendering and memory costs.

3.3. SCALABLE MANY-VIEW RENDERING

In this section, we present our *MegaViews* algorithm, of which an overview is shown in Figure 3.2. We first describe the data representation (Section 3.3.1) before presenting the rendering algorithm (Section 3.3.2), which results in a hierarchical representation of rendered images. While this can be used directly, we also describe a conversion step to obtain independent images for each view.

3.3.1. SCENE AND VIEW HIERARCHIES

Scene Hierarchy We assume the scene to be provided in the form of a multi-resolution spatial tree structure, such as an octree. Each node stores scene attributes: color (or

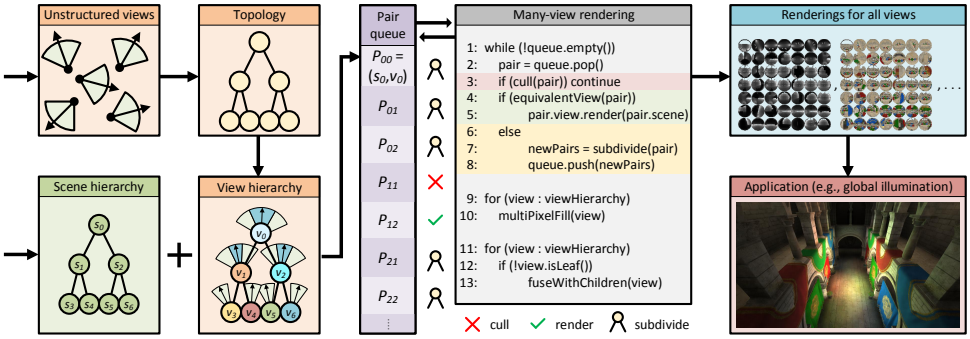


Figure 3.2: Overview of our framework. Many unstructured views are organized in a hierarchy. Together with the scene hierarchy, this serves as the input to our many-view rendering solution. It keeps a work queue, initialized with a pair of both roots, to efficiently process scene-view node pairs in parallel. Pairs are either culled, rendered or subdivided. The resulting renderings can be applied, e.g., for global illumination.

material), position, and a surface normal (or normal cone). The material property or color is typically chosen to be the average of its children. For each node, a bounding volume is assumed available, which is typically a box or bounding sphere enclosing all children. Such scene hierarchies can be generated offline, but dynamic solutions could be used [CG12]. In this sense, our approach is not limited to a static scene hierarchy, although we consider this problem orthogonal to our approach.

View Hierarchy Besides a scene hierarchy, we also rely on a view hierarchy, which groups views spatially in a tree structure. Each view node stores attributes similar to a scene node, but the normal is now defined by a *cone* [WFA*05, JWSP05, RAH07] encompassing the view directions of all contained cameras (Figure 3.3). Then, for each node, we have $v_j := (\mathbf{p}_j, \mathbf{n}_j, \theta_j, \phi_j)$, where $\mathbf{p}_j$ is the center of projection, $\mathbf{n}_j$ the viewing direction, θ_j half the angular extent of the bounding cone, and ϕ_j half the field of view of the frustum. If a view is omnidirectional, we assume $\phi_j = \pi$, and we refer to a global variable ϕ if all cameras share the same opening angle. Again, we assume bounding volumes are available for each node (the yellow circle in Figure 3.3).

Rendered-Image Representation It might sound counterintuitive at first, but instead of rendering the actual image that corresponds to each camera, we always produce an omnidirectional map from the camera's position. We rely on the actual view direction to then query the relevant information from this omnidirectional map. Several options

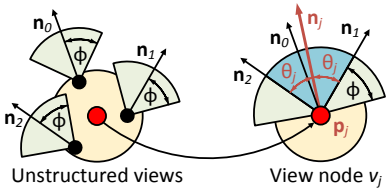


Figure 3.3: Cone-based representation of a multi-view node.

exist for view parameterization, and we only need to impose that all views are parameterized in the same way, including the orientation. In practice, we opted for dual paraboloid maps [BAS02], which are the spherical expansion of a paraboloid map [HS98]. Nevertheless, our solution can be implemented with different representations and we will simply refer to omnidirectional maps in the following. The globally consistent parametrization is crucial to facilitate shared rendering among many different views, as each node in the view hierarchy will contain an omnidirectional map that is a partial rendering of the scene, shared by all its children.

3.3.2. MANY-VIEW RENDERING

Given the scene and view hierarchies, we concurrently traverse them in a top-down fashion during rendering. To keep track of the cut through the double hierarchy, we rely on scene and view node pairs $P_{ij} = (s_i, v_j)$, where s_i and v_j are scene and view nodes in their own hierarchies, respectively. A breadth-first traversal is employed, maintaining a work queue of these pairs, initialized with $P_{00} = (s_0, v_0)$, corresponding to the roots of both hierarchies (Figure 3.2).

A naive traversal would subdivide pairs (by popping them from the queue and pushing its children) when either of the nodes have children, and renders once both nodes are leaves. This process however does not take advantage of redundancy and does not scale well; a million scene nodes with as many views can produce a trillion pairs. We therefore want to process and render for multiple elements from *both* hierarchies at once, which means sharing renderings among many views. Using only scene [HREB11] or view hierarchies [WFA*05] misses a large amount of this shared information, and can not lead to sublinear rendering performance over both scene complexity and the number of views.

We improve the traversal as follows. As shown in Figure 3.2, for each pair $P_{ij} = (s_i, v_j)$, we conservatively test if s_i would contribute to any of the views in v_j, and if not, cull it. Otherwise, if s_i projects to less than a pixel for all children of v_j, we verify if the rendered result would activate the *same* pixel in all views of s_i. If so, we render s_i into the omnidirectional map of v_j, which is shared by all of its children. Otherwise, we subdivide in a way favoring the aforementioned conditions, and process the new pairs in the next iteration. In what follows, we describe the details of our algorithm.

Culling Each view will typically only see a part of the scene, which enables us to cull scene nodes, similar to frustum culling. The test should be conservative and light weight

Culling

```
1:  function cull(pair)
2:      vs = pair.scene.position - pair.view.position
3:      alpha = acos(dot(pair.view.normal, normalize(vs)))
4:      psi = min(PI, pair.view.theta + phi)
5:      fs = sin(alpha - psi) * length(vs)
6:      return alpha > psi && fs > pair.scene.radius + pair.view.radius
```

Figure 3.4: Pseudocode for culling with spherical bounding volumes.

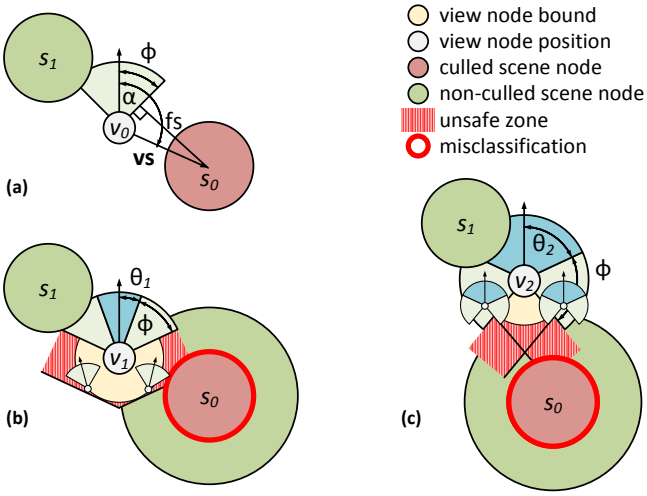


Figure 3.5: Culling. For a single view (a), we can cull scene nodes outside the view frustum. For multiple views (b), we test by virtually enlarging the scene node bound (large green area behind misclassification). Hereby, we avoid incorrect culling, as for s_0. We use the same process for nodes with an angular extent $2(\theta_j + \phi) > \pi$ (c).

to minimize any overhead.

For a single view, culling means ignoring a scene node if its bounding volume lies outside the view frustum (s_0 in Figure 3.5a), in a fashion similar to Roger et al. [RAH07]. Figure 3.4 contains pseudocode for the culling function from line 3 in Figure 3.2, for the case of spherical bounding volumes. To ensure we only cull nodes that are entirely outside the frustum, we require $\alpha > \psi$ and fs (computed on line 5, and shown in Figure 3.4a) larger than the scene node's bounding sphere radius; the view radius is 0 for a single view (line 6). The extension to other bounding volumes is straightforward; we can either take a sphere encompassing the bounding volume, or directly use the tighter bound, resulting in a more complex computation.

For multiple views in a node of our view hierarchy, we want to avoid an individual test per view. While the stored normal cone is conservative, the assumption that centers of projection coincide with the center of the view node's bounding volume is not, and can lead to misclassification (s_0 in Figure 3.5b). In the worst case, child views are located on the bounding volume surface with a view frustum parallel to that of the parent (the unsafe zone in Figure 3.5 indicates where incorrect culling could occur). The extent of this unsafe zone is then at most equal to the maximum extent of the view node bound.

For bounding spheres, the test can efficiently be made conservative. We can cull as before, with the additional requirement that fs is larger than the scene node radius plus that of the view node (large green area behind s_0 in Figure 3.5b). This addition of the view node radius is not smaller than the extent of the unsafe zone, resulting in a conservative test (line 6). Figure 3.5c shows that for v_2, with angular frustum extent $2(\theta_2 + \phi) > \pi$, we can apply the same strategy.

Equivalent view

```
1: function equivalentView(pair)
2:     vs = pair.scene.position - pair.view.position
3:     vsConservativeLength = max(0, length(vs) - pair.view.radius)
4:     alpha = acos(abs(normalize(vs).x))
5:     ps = projectedSize(pair, vsConservativeLength, alpha)
6:     return ps < pixelS || pair.scene.isLeaf() && pair.view.isLeaf()
```

Figure 3.6: View equivalence computation for dual paraboloid mapping and spherical bounding volumes.

Shared Rendering In addition to culling, we employ a second acceleration technique. The idea is to avoid rendering a scene node s_i into each individual view of a view node v_j if the rendered result would be the same for all children of v_j. In other words, for a view node v_j, we will test if the projection of the scene node s_i would fill the *exact same* single pixel in the omnidirectional map of each child view. If so, we render s_i directly into the omnidirectional map of v_i (and not into that of its children) and remove the pair from the queue. This technique quickly becomes effective, as distant geometry will only have minimal parallax if views differ slightly.

Figure 3.6 shows pseudocode for the view equivalence testing function from line 4 in Figure 3.2, in the case of bounding spheres and dual paraboloid mapping. We test for the projected size of s_i, which needs to be less than a pixel. We can directly compute the projection (line 5) using the length of **vs**, which is the vector from the camera to the scene node (line 2), and the angle α between **vs** and the camera direction. For the latter, a dual paraboloid parameterization with the front view looking down the positive x-axis results in the angle between **vs** and the positive or negative x-axis, depending on whether s_i projects into the front or back view, respectively (line 4). While the computation of α and the projected size (line 5) varies for different camera parameterizations, the values can always be found.

To handle a view node v_j that contains multiple cameras, we need to give a conservative upper bound on the projected size of s_i for all child views in v_j. Again, the individual views are not guaranteed to be at the center of v_j's bounding volume. As shown in Figure 3.7b, the result is that the projected size of s_i can vary depending on the child view's displacement, with the worst case being a vertical offset in the direction of **vs**. A conservative test for bounding spheres is then to shorten the length of **vs** by the bound radius of v_j, which results in a larger projected size and a conservative upper bound (line 3).

Given that the projection is smaller than a pixel (line 6), we want to predict if it projects to the same pixel for all views in v_j. A conservative assumption is to consider any position inside of v_j's bounding volume as a potential view location, with horizontal displacement towards the bound surface a worst-case scenario. When the bounding volume of s_i is not smaller than that of v_j, the horizontal offsetting results in filling the same pixel, since s_i will be sampled for all views in v_j and its projection remains identical (Figure 3.7c). Our view equivalence algorithm is therefore valid if we keep the view node bound at most equal to the scene node's. As we show in Figure 3.2, if the equivalence test fails, the pair is subdivided. However, this is only possible when one of the

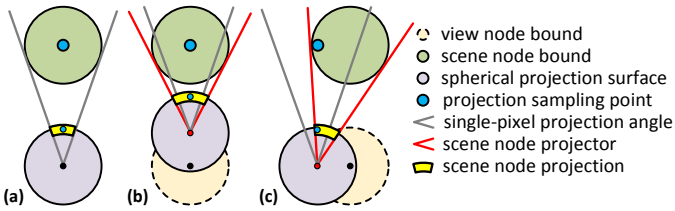


Figure 3.7: Discrepancies of scene node projections for vertical (b) and horizontal (c) displacement of a child view against the projection for the center of the view node (a).

scene and view nodes is not a leaf, which we confirm on line 6.

Pair Subdivision Whenever a scene-view pair $P_{ij} = (s_i, v_j)$ is taken from the queue and a subdivision is required, it is not obvious whether to descend into the scene hierarchy from s_i or into the view hierarchy from v_j. Always subdividing the scene node first would negate the benefits of the scene hierarchy, while first subdividing the view node reduces the approach to a scene-only hierarchy. To benefit from our double hierarchy, we instead opt for a strategy that allows us to optimize for shared rendering.

To validate our aforementioned determination of view equivalence, our subdivision strategy compares the node bounds. If the smallest bounding volume of the children of s_i is smaller than that of v_i, we subdivide v_i. If not, we subdivide s_i. In other words, the view node's bounding volume is always ensured to be smaller than or equal to that of the scene node. For two identical octree structures encoding the view and scene hierarchies, this strategy results in an alternating subdivision. We illustrate this scenario with pseudocode in Figure 3.8, which shows the subdivision from line 7 of Figure 3.2.

Pair subdivision

```
 1: function subdivide(pair)
 2:     if (pair.scene.level < pair.view.level)
 3:         children = pair.scene.children
 4:         other = pair.view
 5:     else
 6:         children = pair.view.children
 7:         other = pair.scene
 8:     for (child : children)
 9:         newPair = createPair(other, child)
10:         newPairs.add(newPair)
11:     return newPairs
```

Figure 3.8: Pseudocode for pair subdivision for octree structures.

Multi-Pixel Filling for Nearby Geometry When rendering scene nodes, most project to a single pixel. However, a pair of leaf nodes cannot be subdivided further, forcing us to potentially falsely report view equivalence (line 6 in Figure 3.6). If a leaf node is very close, it might project to an area larger than a single pixel, especially when using high-resolution renderings. If the view node represents multiple views, the projection of

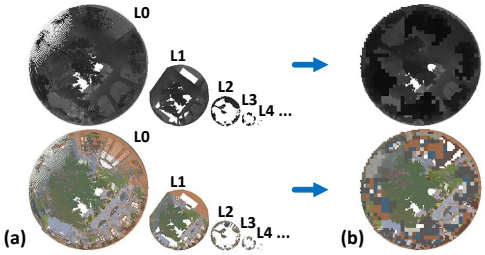


Figure 3.9: Example of mipmap-based hole filling (b) from multi-level point-only renderings (a) at a 256 × 256 resolution for a single view.

the scene node can then potentially differ. Consequently, we need to render the nearby scene nodes into each view individually. While this operation degrades performance, it is relatively uncommon; in practice, this situation occurs for < 5% of the rendered pairs and only for scene nodes in direct proximity.

Since these nodes cover more than a pixel, we could fill the pixels one by one. However, mipmap splatting [LH13] is more efficient. Here, render targets are defined in multiple levels of coarser resolutions (level 0 is the finest resolution). Whenever a scene node projection is larger than a single pixel, we splat it into a higher mipmap level. If wanted, once rendering is completed, we can then postprocess each map by pushing the higher level pixels down to the lower levels, which is a push-only application of a pull-push synthesis [SKE06, RGK*08] (Figure 3.9).

Image Queries After the entire rendering is completed, we can query any pixel of any view in the scene. To this extent, we first map the pixel of the view to its corresponding pixel in the omnidirectional map. Then, we descend the view hierarchy from the root and look up the values in this location in each view node's map. The last encountered non-empty value corresponds to the wanted pixel value.

If many queries are performed, it can be beneficial to perform a *fusion* of the omnidirectional maps to produce a complete image per single view. To this extent, it is sufficient to perform a top-down processing, where the pixel values of the parent node are fused with the map of the child nodes, which means that we fill up holes in the child map with the content of the parent map. Ultimately, this process results in a completely filled image for each leaf view.

Finally, some applications, such as shadow mapping, require depth information. Initially, we use the distance to the center of v_j as the depth value for its omnidirectional map. If we query an individual view v, there would then be a small discrepancy with regard to the actual depth value. This difference is easily rectified during the fusion step by taking the actual positions of v and v_j into account.

3.4. RESULTS

We implemented our solution entirely on the GPU using the OpenGL API, with no CPU-GPU communication at run time. We tested it on an NVIDIA GeForce GTX 1080 Ti in

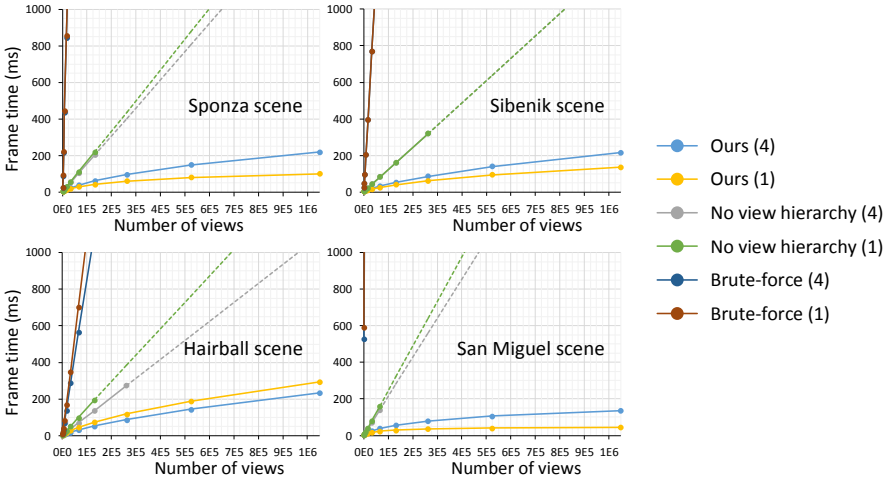


Figure 3.10: View render timings for four scenes. We compare our method to not using a view hierarchy (Many-LoDs [HREB11]). To test two distributions of views, we initialized the view set as 1- and 4-bounce VPLs. Additionally, a brute-force sequential rasterization without any hierarchy is presented. Dotted lines represent an extrapolation where data is missing due to memory limitations on the pair queue.

full HD. Unless otherwise specified, we made use of sparse voxel octrees with 11 levels for both hierarchies. We use bounding sphere volumes encompassing the cubical voxels for our culling and view equivalence computation. Further, we use a 16×16 resolution for single views. At this resolution, multi-pixel filling is not necessary in practice and therefore excluded in timings except when specifically mentioned.

The scene SVO is generated in a few seconds with an unoptimized depth peeling preprocess [KSA13], which builds the hierarchy down to the specified maximum depth. We consider more efficient voxelization as an orthogonal problem. Using more advanced solutions [ED06, SS10] would significantly reduce construction time and even enable animated scenes, since our many-view rendering is unaffected by changes in the hierarchies.

We construct the view hierarchy each frame and support fully dynamic lights. After initialization with a root node, the view hierarchy is generated from a set of single views. For each view, we refine the tree down to the deepest level, such that the view ends up in a leaf node. We count the number of views per leaf node, which is then used to construct an offset into a global array, containing all information about each view.

Many-View Rendering Performance We tested a total of four scenes: *Sponza* (Figures 3.1, 3.11 and 3.17, $6M$ leaf nodes), *Sibenik* (Figure 3.15, $3.1M$ leaf nodes), *Hairball* (Figure 3.16, $6.9M$ leaf nodes), and *San Miguel* (Figure 3.18, $1.6M$ leaf nodes). As indicated in Section 3.3, the views are rendered using an omnidirectional map with the same coordinate system regardless of the view direction. In practice, we use dual paraboloid maps [BAS02], which are the spherical expansion of a paraboloid map [HS98]. Each view has a 180° frustum using a dual paraboloid mapping. We tested two view distributions

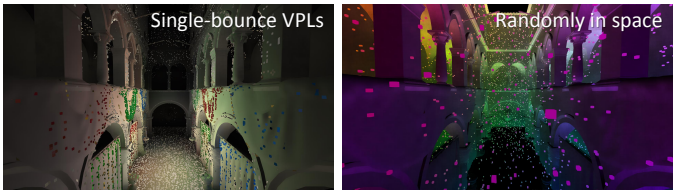


Figure 3.11: Different distributions of 64K views in the *Sponza* scene.

of up to 1M views. The first are 1M VPLs generated directly from the light source, the second are 4-bounce VPLs. Here, 256K VPLs were released from the light and bounced three times, leaving one VPL behind at each bounce and at the final impact point.

We compare our timings to ManyLoDs [HREB11] and a brute-force rasterization, which does not rely on any hierarchies. We also tested a solution with a view hierarchy but no scene hierarchy. However, the amount of pairs exceeded the available queue memory as soon as more than a hundred views were used. For the same reason, we extrapolate the obtained data with dotted lines in Figure 3.10.

Our approach achieves sublinear performance in all scenes, while competing methods show a clearly worse scalability with respect to the number of views. For a million views, our solution outperforms ManyLoDs by roughly an order of magnitude. Single-bounce VPLs in close proximity of a smaller part of the scene, like in the *Sponza* and *San Miguel* scenes, have a high correlation and result in the largest speedup. In the worst case, views are distributed randomly in space, reducing the coherence. We show this extreme case for the *Sponza* scene in Figure 3.11, where the random distribution results in rendering times a factor 4 slower. Still, we observe sublinear performance as we scale up to many views. Naturally, if the number of views is sufficiently reduced, we lose opportunities for shared rendering, which causes our performance to roughly match that of ManyLoDs for VPL numbers below 2048.

We break up our timings into individual components for 1M single-bounce VPLs in Figure 3.12, and furthermore show performance for the view hierarchy construction, which includes VPL placement. In our tests, culling reduces the frame time by 20% at most, but at a very small cost (< 2 ms). In cases such as the *Hairball* scene, where each

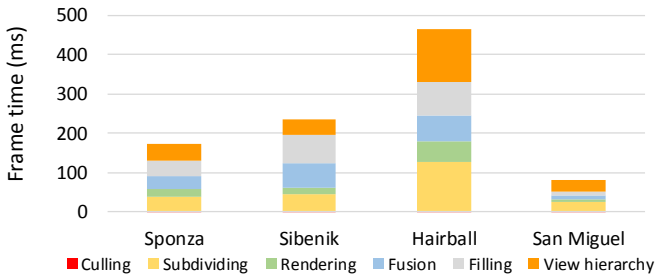


Figure 3.12: Individual timings for 1M single-bounce VPLs in four scenes.

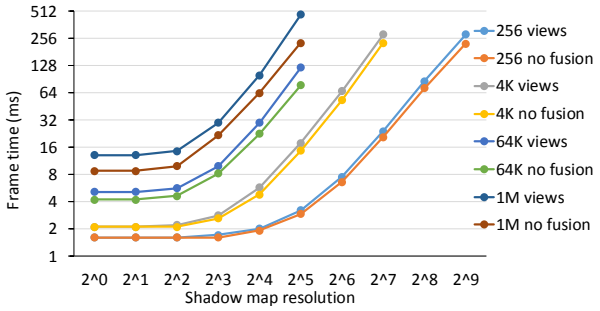


Figure 3.13: Timings for many-view rendering with and without fusion for increasing shadow map resolution in the *Sponza* scene.

VPL on the wall potentially sees the entire scene, culling provides no gain at all. In these scenarios, the speedup is solely due to our double hierarchy and shared rendering.

Our method scales linearly with shadow map resolution. We can see in Figure 3.13 that for the *Sponza* scene, when the resolution doubles, the frame time for rendering includes a constant overhead and then roughly scales with a factor 4, corresponding to the increase in pixels. The fusion step is relatively cheap and only adds a small amount of compute time. Without it, querying is an order of magnitude slower, since each query visits a number of potentially sparse shadow maps, up to the hierarchy depth. For VPLs, querying is often a bottleneck, which makes the fusion a valuable option.

We identify four components that take up significant GPU memory during runtime, showing their consumption for $1M$ single-bounce VPLs in Figure 3.14. The pair queue that is kept for the double hierarchy traversal contains 64 bits per pair. For the rendered images themselves, we report consumption for fused 16×16 shadow maps with 32-bit depth values. The scene and view hierarchies contain 512 and 256 bits of information for non-leaf and leaf nodes, respectively.

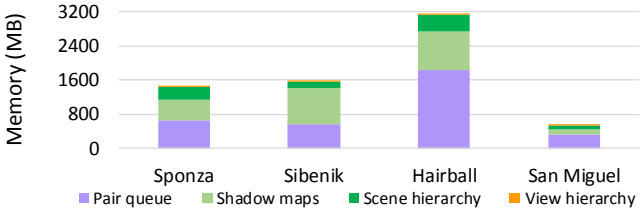


Figure 3.14: Memory usage for $1M$ single-bounce VPLs in four scenes.

3.5. APPLICATIONS

Our algorithm is general but particularly well suited for low-resolution views or an extreme amount of views, as the amount of shared information increases. For this reason, real-time global illumination techniques are a good test case for our solution.

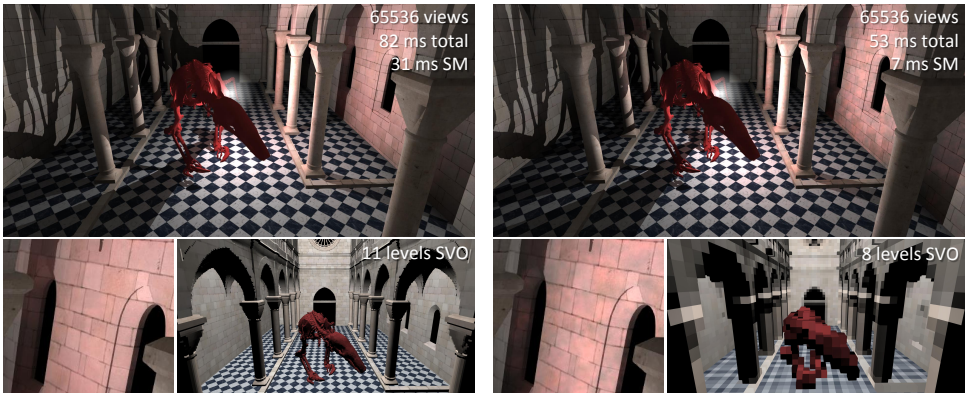


Figure 3.15: Effect of hierarchy resolution in the *Sibenik* scene. While the shadow map rendering cost is significantly reduced, using a too low hierarchy resolution causes inaccurate shadow maps, resulting in missed or exaggerated occlusions and artifacts.

3.5.1. INSTANT RADIOSITY

In our implementation, VPLs and their view hierarchy are generated on the fly (see Figure 3.12 for timings), storing their propagated radiance as attributes. Each VPL has a hemispherical frustum ($\phi = \pi/2$), which is taken into account for our culling.

We rely on our *MegaViews* approach to generate shadow maps for many VPLs, but producing a final image still requires gathering the VPL contributions for each screen pixel. Recovering all contributions would be too costly for an interactive application. Fortunately, our algorithm enables an acceleration. We can apply our culling during the gathering step as well. For this traversal, we stop at a coarse level in the hierarchies, and cull pairs as before. Additionally, we enable an optional distance-based cutoff to prevent gathering from distant, often negligible VPLs, which is a common approximation [OBS*15]. This test can be conveniently accelerated using the view hierarchy by culling faraway view nodes.

We employ a per-pixel random subsampling of the VPLs, after which we apply a cross bilateral filter [ED04, PSA*04, MML12], which works well in generating smooth results due to the very large number of VPLs that we sample from.

Timings To evaluate performance, we illustrate the effect of reducing the resolution of the scene and view hierarchy for the *Sibenik* scene in Figure 3.15, while maintaining $64K$ 4-bounce VPLs. Reducing the scene hierarchy from 11 to only 8 levels speeds up the shadow map generation (SM in the figure) from 31 ms to 7 ms due to the faster scene traversal. Nevertheless, the resulting shadow maps lose precision, which translates to missed or exaggerated occlusions. Consequently, artifacts start to appear. This effect is also reflected by having around 15 times more leaf pairs that project to more than a pixel and can therefore potentially introduce errors (Section 3.3.2).

The effect of changing the shadow map resolution is shown for the *Hairball* scene in Figure 3.16, where we compare a 16×16 to a 1024×1024 resolution. For the higher

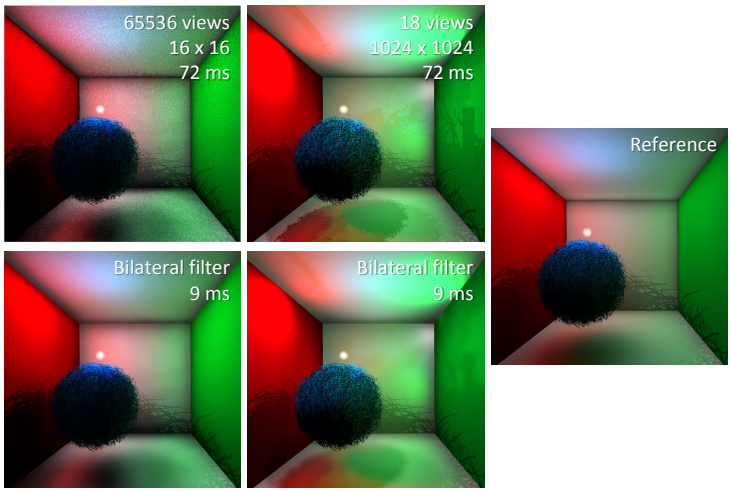


Figure 3.16: Effect of shadow map resolution in the *Hairball* scene.

resolution, an equal-time comparison results in 18 views. Such a small amount of views cannot deliver a convincing quality. Low-resolution shadow maps are very fast to compute, and can still deliver good quality, as the light energy is distributed across many VPLs. However, we do see some over- and underestimation of occlusion due to the lower precision of the shadow maps, when compared to a reference solution. These shortcomings are not due to our method, but are shared by all VPL-based solutions when relying on low-resolution shadow maps.

We also evaluate our culling and the distance-based cutoff during gathering. Here, we use 64 random samples per pixel from $1M$ single-bounce VPLs for the *Sponza* scene, which were all rendered using our solution. We can eliminate on average 94% of the leaf view nodes during our concurrent traversal up to a hierarchy level of 6. Consequently, mostly samples are used that actually contribute to a pixel's indirect illumination. This

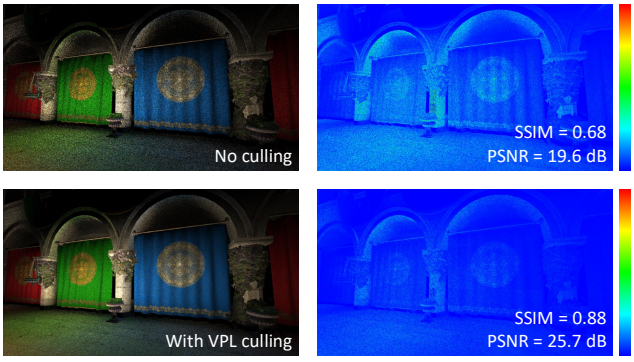


Figure 3.17: Hierarchical culling in VPL gathering. By only sampling from non-culled VPLs, noise is significantly reduced.

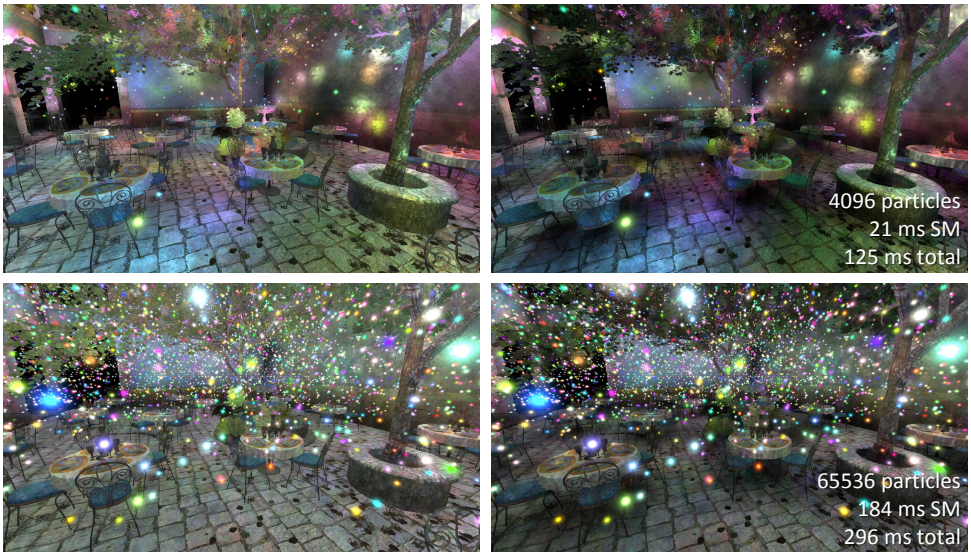


Figure 3.18: Examples of glowing particle rendering without (left) and with (right) shadows. By simply setting the view volume to a sphere, we can render shadow maps for glowing particles.

significantly reduces the noise, as becomes apparent from the SSIM [WBSS04] and PSNR comparison, where a higher SSIM value signifies increased perceptual similarity. Figure 3.17 shows a closeup comparison with absolute differences.

3.5.2. GLOWING PARTICLES

Similar to instant radiosity, we can perform many-light rendering. Again, we build the view hierarchy on the lights each frame to enable animation. Our many-view rendering enables us to efficiently approximate visibility for a large number of particles, which adds realism to a scene when compared to not evaluating visibility.

Timings We show results for glowing particles in Figure 3.18. Since they represent omnidirectional lights, each node's view frustum is now a complete sphere, making it impossible to use culling. Additionally, since the particles are randomly distributed in space, performance is reduced compared to VPLs, since there is less coherence. In the *San Miguel* scene, our solution requires 21 and 184 ms for rendering shadow maps for $4K$ and $64K$ particles, respectively.

3.6. DISCUSSION AND LIMITATIONS

Our method scales well with the amount of views and scene nodes. We presented sublinear performance for both dimensions, which makes our solution very effective. Several applications could benefit from our solution. We presented indirect illumination using our method, but other examples, such as visibility for crowd simulation, fast collision detection or reflections via cube maps are also among the possible applications. Our method is relatively easy to implement and can be entirely executed on modern graphics hardware in an efficient manner, since our hierarchy traversal ensures just one operation

per thread: either culling, rendering, or subdivision.

A limiting factor of our approach is memory consumption. The pair queue, which often consumes the most (see Figure 3.14), requires only volatile memory. To reduce the queue size, we can apply a multi-pass solution [RAH07] on either the full pipeline, or by sequentially treating view hierarchy subdivisions. For instance, subdividing the view root into octants reduces memory requirements by a factor up to 8. Additionally, temporal coherence [HREB11] and aggressive approximation, such as allowing projections to larger than a pixel, can further reduce the queue size. The renderings themselves can be compressed using texture compression, sparse-texture extensions (typically 30% of the fused omnidirectional map is non-filled), or, in the case of shadow maps, precision reduction. For instance, the 32-bit depth values that we use can be reduced to 16 bits, since their precision only needs to match the hierarchy resolution. When using SVOs, the scene hierarchy can be compressed using directed acyclic graphs [DKB*16, DSKA17], while the view hierarchy overhead is typically negligible, since it is a sparse subset of the scene hierarchy.

As for micro-rendering solutions, choosing a low resolution can lead to aliasing and occlusions can be overestimated (e.g., sub-pixel objects still fill entire pixels). One remedy is to increase resolution, but it results in additional compute time. While our approach scales linearly in resolution, adequate anti-aliasing solutions are an interesting avenue for future work. Similarly, the resolution of the hierarchies needs to be carefully chosen to find an acceptable tradeoff between visual quality, and requirements on performance and memory. In our experiments, we could no longer perceive a visual difference for hierarchy resolutions of 11 levels and more.

Furthermore, as in all VPL approaches, temporal coherence is an interesting factor. It is possible to reuse information over time if scene and view changes are insignificant. Our shared rendering solution seems like a good starting point by keeping high-level omnidirectional maps in the hierarchy stable over several frames.

Our approach is compatible with a different parametrization of the omnidirectional maps. Our choice was inspired by its usefulness in an instant radiosity context. An interesting direction would be adaptively controlling the resolution based on the image content.

3.7. CONCLUSION

We have presented *MegaViews*, a scalable algorithm to efficiently render complex scenes from a very large number of viewpoints. Our concurrent traversal on both scene and view hierarchies enables shared rendering and early culling. Consequently, we reach sublinear performance over the scene complexity and the amount of views. Our algorithm is general enough to be applied to many multi-view problems, and fits well with real-time many-light rendering. For future work, we want to exploit coherence in animation. A first solution could reuse cuts from previous frames in the spirit of [HREB11].

4

Real-Time Canonical-Angle Views in 3D Virtual Cities

Your life is your own.
Rise up and live it.

Terry Goodkind

Virtual city models are useful for navigation planning or the investigation of unknown regions. However, existing rendering systems often fail to provide optimal views during the exploration, introduce occlusions, or show the buildings from the top only, which limits the amount of useful visual information accessible to the user. In consequence, users are forced to interact more extensively with the application to avoid these shortcomings. This process can be quite time-consuming. In this chapter, we propose a new technique based on canonical views to address these problems. We compute every building's canonical view and, dynamically, transform it correspondingly, so that it is easy to identify under all camera angles. A user study was conducted to assess how this technique compares to a regular view; our method improves the recognizability of the buildings and helps the users explore the virtual city more efficiently. The results indicate that using canonical views is beneficial for efficient navigation in virtual cities.

Save for an extended introduction, this chapter is a verbatim copy of a publication in the proceedings of VMV: Vision, Modeling & Visualization by Timothy R. Kol, Jingtang Liao and Elmar Eisemann [KLE14]. It was presented at VMV 2014 in Darmstadt, Germany. As for the distribution of work, I implemented and devised a large part of the algorithm, came up with the modified depth test, and wrote much of the paper.

Figure 4.1: Standard top-down views make buildings hard to recognize (left). Our interactive canonical view reveals facades better, without causing confusion or overly unrealistic deformations. Our work facilitates navigation and exploration.

4.1. INTRODUCTION

ALTHOUGH our realistic representation of virtual worlds works well, directly presenting the data as is does not always benefit user interaction. In this chapter, we therefore consider an *illustrative* approach to improve navigation. This can especially be beneficial for 3D virtual city models, which are becoming more prevalent in numerous applications. Virtual cities play a role in the entertainment industry and visualization systems, but also in navigation applications, tourist maps and for disaster management simulations. An increasing number of tools are available to produce such models (e.g., Google Earth), and the number of available models increases constantly.

Nonetheless, most navigation planning tools still rely on combinations of satellite imagery, street-level views, and aerial photographs taken at a 45° angle, as each of these alone has certain shortcomings. Satellite photographs give an excellent overview of the city layout, but buildings are difficult to recognize from the top, while landmarks play a significant role in memorizing and describing a route [Den97]. On the contrary, street-level views fail in giving a global context, but ease recognition. When combined, users need to divide their attention between two windows. Ultimately, 45° aerial photographs seem to lead to a good compromise, but having only four viewing angles can lead to visual clutter and occlusions of the street in dense areas, hindering successful navigation planning. Similar problems persist for a full 3D visualization, which can still exhibit high visual clutter (Figure 4.2), or, when moving towards a top-down view, can lead to reduced recognition rates. Furthermore, finding appropriate 3D views can be difficult.

For landmark recognition, the viewing angle is important. Tourist maps often rely on iconified versions of the landmarks using a specialized viewing angle, as there is a strong preference for certain viewpoints in the context of object perception [EB92, VB95, BTBV99]. The characteristic viewpoint that users are most comfortable with is called the *canonical view* [PRC81]. Nonetheless, in a dynamic navigation environment, it is not possible to maintain such a view while the user is freely navigating through the scene.

In this chapter, we modify the standard perspective to approach a canonical view by ensuring a certain observation angle for each building (Figure 4.1). Hereby, we facilitate recognition and orientation, making our algorithm useful for navigation tasks. Our work makes the following contributions:

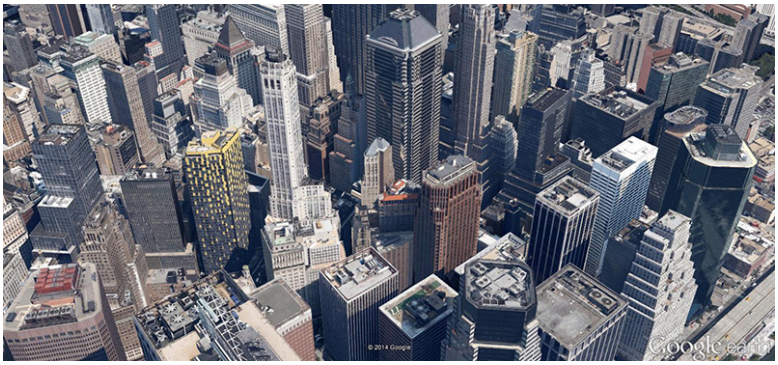


Figure 4.2: Manhattan as seen in GOOGLE EARTH with 3D buildings enabled. While impressive, note the large amount of visual clutter that prevents users from discerning the street, rendering this view useless for navigation planning.

- a viewer exploiting canonical viewing angles for buildings;
- a transformation-conform depth test to avoid sorting; and
- a user study to determine preferred viewing angles.

4.2. RELATED WORK

Obtaining the canonical view of 3D objects [PRC81] has been an active research topic for many years in computer graphics [DCG10, PPB*05] as well as psychology and neuroscience [Pet00]. Originally, Palmer et al. [PRC81] indicated that the quality of a view angle depends on both the visual information that is objectively available, as well as the subjective importance of this information to the user. They found that participants had a strong preference for off-axis views, such as the three-quarter perspective, showing three sides of the object. Congruent results were obtained by [PHL92], who used computer generated images instead of physical objects.

In this work, we focus on building-like structures. Many experiments were conducted with unfamiliar [PHL92], abstract [EB92, CE94] or irrelevant [PH88, HPL91] objects. Even in extensive user studies carried out to determine influential object properties [BTBV99], the model coming closest to the shape of a building was a truck. One conclusion that could be drawn was that the preferred viewing elevation is significantly below 45° and depends on a complex interplay between the geometry, the user's familiarity with the object and the tasks to be performed [BTBV99]. Hence, a universally valid view might not even exist [CE94].

Measures such as *silhouette entropy* and *curvature entropy* [PKS*03], the *visible area* and *silhouette length* [PPB*05], *mesh saliency* [LVJ05], *view entropy* [VFSH01], or *semantics* [MS09] have been proposed to determine best views. Some derivations might be possible for particular objects, but as observable via recognition benchmarks [DCG10, PPB*05], no general solutions seem to exist. Consequently, Secord et al. [SLF*11] focused on a selection of views per object, precomputed from a general model, while Yamauchi et al. [YSY*06] proposed to cluster a large set of uniformly sampled viewpoints

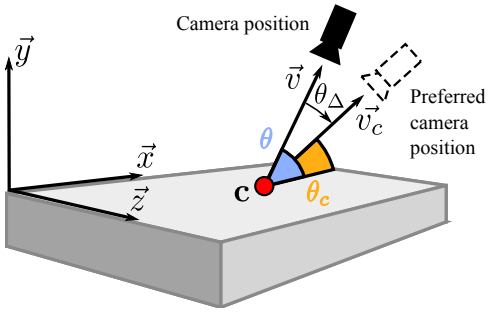


Figure 4.3: Pointing from the center of the bounding box top **c** to the preferred camera position is the vector $\vec{v}_c$ with elevation θ_c. Vector $\vec{v}$ is pointing from **c** to the actual camera position **C**, having an angle θ. We call the difference between these angles the elevation offset θ_Δ.

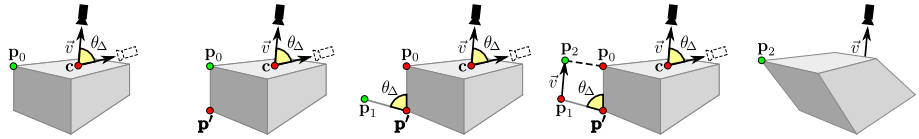
into several centroids based on their similarity. Our work focuses on buildings, and despite many different appearances, the basic shape is relatively consistent.

In contrast to the previous methods, our goal is not a viewpoint for a compact 3D model, but an optimization of an entire city, in which each building is manipulated to best convey its context and appearance. Other approaches pursued similar goals. Automatically generated tourist maps [GASP08] use optimized views of landmarks to facilitate orientation, but the views and 2D maps remain static. Semmo et al. [STKD12] presented a mix to seamlessly transition between a 3D visualization and a 2D schematic overview of a virtual city. In the 2D view, the landmarks are shown as billboards that transform to regular 3D models when the user zooms in on the map. A different fusion to combine pedestrian and bird's eye views is to bend the city model [LTDJ08]. Furthermore, viewport variations can improve the perception of the spatial relations in 3D [JD08]. While both methods can be useful for navigation, they do not improve landmark views dynamically, which makes investigation tasks more difficult.

A real geometric transformation can make area-of-interest visualizations more successful [MDWK08]. Similar to our method, they applied a shear transformation on buildings to reveal hidden facades in top-down views next to important streets, while distant geometry remains unchanged. Our transformation however, is dynamically based on the camera and optimizes for a determined viewing angle to produce a more preferable view. In this way, it also differs from [QWC*09], where landmarks were emphasized along navigation routes, while widening relevant streets to prevent occlusion. While these previous results show the benefit of such modifications for navigation, the production of a single view does not allow free exploration, which is supported in our solution.

4.3. CANONICAL VIEWS

Initially, we experimented with different visualization algorithms, but quickly discovered that adhering to the strict definition of a canonical view would restrict interaction and significantly reduce the realism of the resulting rendering, potentially even leading to confusing temporal discontinuities. Therefore, we decided to impose two constraints on the deformation of buildings. First, building footprints should remain fixed to give users



(a) Compute θ_Δ. (b) Project $\mathbf{p}_0$ on to the bottom face to obtain $\mathbf{p}'$. (c) Rotate $\mathbf{p}_0$ around $\mathbf{p}'$ to obtain $\mathbf{p}_1$. (d) Project $\mathbf{p}_1$ along $\vec{v}$ to obtain $\mathbf{p}_2$ at the height of $\mathbf{p}_0$. (e) All of the vertices transformed.

Figure 4.4: Schematic overview of the algorithm simulating a canonical angle, where the vertex $\mathbf{p}$, denoted by a green dot and originally at position $\mathbf{p}_0$, is transformed.

a good sense of the structure's location and to prevent floating, which is typical for icon-based maps. Second, we want to enable rotation around the building for exploration purposes, hence we deliberately avoid fixing the view orientation to a three-quarter view. Furthermore, we assume that the effect of the distance to the camera (i.e., the zoom) on the canonical view is negligible, which is similar to assumptions made for tourist maps.

In consequence, optimal (i.e., preferred) views are derived from a *canonical angle* θ_c per building, which is defined as follows. First, we assume a simpler shape by focusing solely on the bounding box, which is also the shape we will use to derive a preferred viewing angle, making our approach less dependent on attributes such as building styles. The angle of a box is then measured using spherical coordinates with the origin at the center point $\mathbf{c}$ on the top (Figure 4.3). The goal of our algorithm is to measure this subtended angle based on the bounding box and compare it to the canonical angle, to obtain a corrective elevation offset, which is then transferred to the vertices of the actual building.

Specifically, the vector $\vec{v}$ pointing from $\mathbf{c}$ to the actual camera position $\mathbf{C}$ and the top plane form an angle θ, which optimally should match θ_c. The difference between θ and θ_c is the elevation offset θ_Δ, which will be used to adapt the building to achieve improved viewing conditions. We will assume that the heights of the buildings are along the y-axis and the buildings' floors are parallel to the x,z-plane.

4.3.1. BUILDING TRANSFORMATION

To transform the building according to the canonical angle, we first compute the subtended angle θ of the bounding box. Note that this value is negative if the camera is positioned below $\mathbf{c}$, i.e., $\mathbf{C}$ has a smaller y-coordinate than $\mathbf{c}$. Next, by subtracting the canonical angle, we obtain the corrective elevation offset $\theta_\Delta = \theta - \theta_c$.

If θ_Δ is negative, corresponding to a rotation *towards* the camera, we do not apply our algorithm, as this would hide nearby facades instead of improving the view on the model. If not, we rotate the box by θ_Δ to establish the canonical angle. In other words, given a vertex $\mathbf{p}$ at position $\mathbf{p}_0$, $\mathbf{p}$ is projected on the bounding box *bottom* resulting in $\mathbf{p}'$ (Figure 4.4b). Then, $\mathbf{p}_0$ is rotated around $\mathbf{p}'$ in the opposite direction of $\vec{v}$, i.e., away from the camera, with a rotation angle of θ_Δ (Figure 4.4c), leading to $\mathbf{p}_1$. Performing this operation, the building is rotated to match the intended angle, but its base remains static. This transformation respects the first constraint; the building footprint should remain at the same location to avoid the impression of the building floating above the

(a) In normal view.

(b) All vertices are rotated.

(c) All vertices are translated.

Figure 4.5: Transformation of a building in a city model. Note that this is the same top-down view as in (a), yet the building is transformed in such a way that the viewing angle appears much more comfortable.

ground. Figure 4.5 illustrates the final result for a building in a city model.

Note that we use the vector $\vec{v}$ pointing from $\mathbf{c}$ to the camera position $\mathbf{C}$ throughout our algorithm. We do not calculate a distinct $\vec{v}$ for every vertex, pointing from $\mathbf{p}_0$ to the camera. This choice is useful to maintain consistency within the building. If we rotate every vertex separately, parts of the object will rotate in different directions, as demonstrated in Figure 4.6, which can lead to a confusing appearance.

We can establish an interesting link between the previous method and a well-known operation; if we translate $\mathbf{p}_1$ in the direction of $\vec{v}$ until its y-coordinate matches $\mathbf{p}$'s original value ($\mathbf{p}_2$ in Figure 4.4d), only a small deviation is introduced with respect to the rotation angle (stemming from the perspective projection), but the operation becomes a simple shearing. The modification is visually negligible, but makes this operation linear and easy to implement on graphics hardware. Furthermore, this insight will be key in resolving the visibility relationships between the different buildings, as simply using the deformed building's geometry can lead to visual artifacts, as analyzed in the next section.

4.3.2. OCCLUSION TEST

We have seen that the angle can be optimized by applying a shearing transformation to each building. Nonetheless, using a standard z-buffer can lead to occlusions that are introduced by buildings that might now overlap; especially for low viewing angles (Fig-

(a) Using $\vec{v} = \mathbf{C} - \mathbf{c}$.

(b) Using $\vec{v} = \mathbf{C} - \mathbf{p}_0$.

Figure 4.6: Calculating $\vec{v}$ per vertex causes deformation.

(a) Unintuitive intersections may occur.

(b) We solve this using a modified depth function.

Figure 4.7: Intersections may occur for low buildings standing very close to skyscrapers.

ure 4.7a), such situations are common. In theory, all buildings could be sorted and rendered back to front, but this would be costly as it needs to be done per frame, and for the structures within a building, like balconies, one would need to wipe the z-buffer after each building is rendered. In order to solve the visibility problem efficiently, a more careful analysis is needed.

First, we will concentrate on the task of ordering the buildings with respect to each other. Instead of using the standard z-buffer, we redefine the depth function as follows. Conceptually, for each pixel, we cast a ray from the camera to the point on the transformed building, and project this ray along the y-axis onto the x,z-plane. We then find the intersections of this ray with the footprint of the building, and use the distance of the closest intersection point to the camera as a depth function. This process is shown in Figure 4.8. Our transformation algorithm ensures that each ray that reaches a transformed building will, when projected to the x,z-plane, intersect an edge of the building's footprint. The only exception occurs when the corrective elevation offset $\theta_\Delta = 0$. This situation can be handled easily by introducing a small extension to the ray to ensure at least one intersection.

It should be noted that we compute the actual footprints of the buildings in a preprocessing step and write them to a file, which is read during initialization along with the mesh file itself. The footprints are generally very lightweight, which makes it easy to maintain real-time frame rates. The building footprints can be passed along to the shader either compactly stored in a texture or as uniform variables. If necessary however, the footprints can always be simplified during the preprocessing step.

This modified depth function enables a correct per-pixel ordering of the buildings for non-overlapping footprints, which is usually the case in city models. If footprints do overlap, they should be considered as one building and fused. However, this approach obviously produces artifacts due to the fact that the depth function only ensures correct ordering between buildings, without considering the order of the fragments inside a given building.

To solve this problem, we rely on a two-step process. In the first pass, we discover for

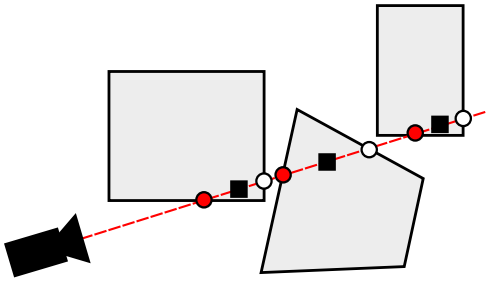


Figure 4.8: In this top-down view, the considered fragments for some pixel are shown as black squares. The nearest intersections of this ray with the footprints are shown in red (other intersections are shown in white), and their distance to the camera is used for the depth function. To find these points, a standard algorithm to compute intersections in 2D for line segments is used.

each pixel which building should be rendered. To this extent, we make use of the modified depth function by computing the distance to the closest intersections. The buildings are thus sorted per pixel according to their relative distance to the camera, and we draw each building with a unique ID to derive a mask that helps us resolve occlusion issues. In the next pass, we make use of the standard z-buffer, but add a check in the fragment shader, in which we rely on the previously determined mask to find out which building the currently drawn pixel should be associated with. If the given fragment comes from a different building, we simply discard it to prevent intersections. Since this second z-buffer test relies on a standard depth function, artifacts are avoided and occlusions are correctly handled, as is shown in Figure 4.7b.

4.3.3. OBTAINING THE CANONICAL ANGLE

To find an optimal canonical angle, we relied on a small user study involving eight participants. We showed a bounding box with varying ratios on the screen, resized to fit into a sphere of radius one at the origin. A camera with a 60° opening angle was placed at a distance to the observed object to roughly match the sphere's projection on the screen in pixels. While the box was automatically rotating around the y-axis, we allowed the users to change the angle θ subtended by the bounding box and the camera, as defined previously. Each user performed the test for a total of 64 boxes of varying size ratios, as indicated in Figure 4.9.

A slight correlation between the canonical angle and dimension ratios can be seen, but, especially for very differing dimensions, we observed a high variance. This result is not surprising, seeing as we rotate the camera around the bounding box. Thus, for these extreme cases, the building can appear very small seen from one side, but extremely stretched from another, producing rather uncertain results. Nonetheless, for larger x/z-ratios, there seems to be a tendency that the preferred camera angle is lowered slightly, which makes sense, as it leads to views that reveal more of the elongated side. Similarly, the canonical angle increases when the object height grows in the y-direction. This observation seems to reflect that the view for tall objects should be more from above to be able to see the whole box.

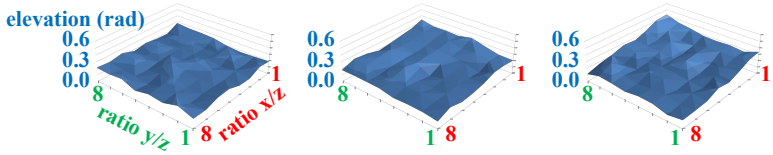


Figure 4.9: Results of our user test on the canonical angle. Here, three subjects are shown to illustrate the strong variance. The ratio changes influence the relative height (y) and aspect (x) of the building.

Overall, we considered the results too noisy to draw general conclusions. In theory, this seems to imply that each user should define a personalized canonical angle. Nonetheless, to ease implementation and avoid such a configuration step, we decided to investigate the use of an average canonical angle, namely $\theta_c = 0.273$ radians (15.6°), which already leads to an improved performance in several scenarios (see Section 4.4.1).

Ultimately, a differing preferred angle does not imply that any adaptation is unwanted. It seems that some users simply preferred more drastic degrees of adaptation, which is also illustrated in the user study, showing that the results using the fixed angle are still generally preferred over a standard illustration.

4.4. RESULTS

The implementation of our algorithm easily reaches real-time rates – only a few basic operations are required per building in the vertex shader at each frame. On a desktop computer with an Intel Core i7 3.7 GHz CPU and an NVIDIA GeForce GTX 980 Titan GPU, for a city model of 40K triangles with approximately 300 objects, all images were generated in far beyond real-time rates. Results of our algorithm are given in Figure 4.10 as well as in the various examples showcased throughout the chapter.

4.4.1. EVALUATION

To assess the impact of our algorithm on navigation planning and investigation tasks, we conducted two user studies. The first user study tested the effect of our algorithm on assisting users in finding buildings in a virtual city model, while the second test focused on its effect on memorizing navigation paths within the same virtual city. All tests were performed with the aforementioned equipment.

Figure 4.10: Renderings using the canonical view. We show in each the same scene rendered using the normal view (left) and the canonical view (right).

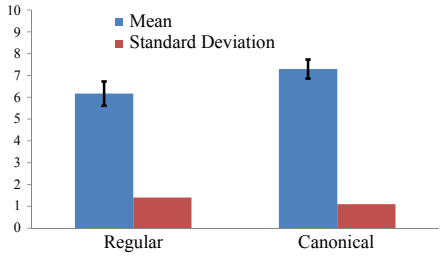


Figure 4.11: User rating of all 24 participants on their own performance using the regular and canonical views for finding buildings, showing the means and standard deviations, with confidence intervals of $p = 0.05$.

In total, 24 participants with normal or corrected-to-normal vision, with ages ranging from 23 to 34, participated in our first user study, which took 30 minutes per user. The second study featured a comparable group of 12 participants, taking 15 minutes per user.

4.4.2. FINDING BUILDINGS USING THE CANONICAL VIEW

The first task was to find a building in a city model consisting of 300 buildings of different sizes and shapes, as illustrated throughout the chapter. The target building to be found was shown in a separate window. Participants were asked to navigate through the city in free camera mode, clicking on the object when they thought they had found it. We timed how long the users took before successfully identifying the object.

In total, users had to perform 12 of these recognition tasks for different buildings at distinct locations. We toggled the canonical view after every task, and varied the canonical view and the order of buildings between participants to ensure unbiased results.

This task turned out to be very demanding. Initially, it is not known in which direction to move, causing participants to move in the direction of, or away from, the target building. This led to longer recognition times for some participants, resulting in a high variance. There was still a 7% improvement in the timings for the canonical view. Nonetheless, the result was not significant due to the high variance.

For this reason, we also conducted a subjective study which was presented to the users right after the experiments. They rated themselves on a scale from 0 to 10 for their performance in finding the buildings using the normal and the canonical view. The results of this evaluation are shown in Figure 4.11, which shows a preference for the canonical view and a relatively low standard deviation. This indicates that users rank their performance significantly higher for our algorithm than for a regular camera model. Furthermore, participants indicated that they were not confused by the viewing algorithm, and that the canonical view felt quite intuitive.

4.4.3. MEMORIZING ROUTES

The second evaluation focused on navigation planning. We showed 12 participants a video of a route in our city model two times, after which they were asked to follow the same route themselves, with the camera constrained to the roads. This time, we alternated between a regular top-down view, a canonical top-down view, and a street-level

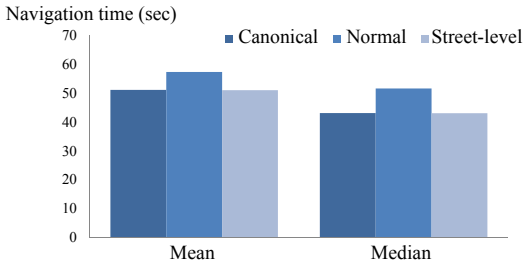


Figure 4.12: The mean and median time it took the 12 users to arrive at the endpoint of the memorized route.

view for the preview videos. In total, three navigation tasks had to be performed, with a different route and preview video every time. We switched the order of routes around for unbiased results.

When users took a wrong turn, we immediately notified them, but also introduced a 5-second penalty. By timing how long it took the participants to arrive at the endpoint of the route, we obtained the results shown in Figure 4.12. Showing the preview videos in canonical view or street-level view led to slightly better results regarding the navigation time. However, street-level views always have the disadvantage of not showing the whole context. For memorizing a route this may work, but efficiently planning complicated routes is simply not possible, and getting lost might be riskier.

The most interesting result of this user study however, was the amount of wrong turns that were taken given the respective preview videos. We kept track of which participants went the wrong way, and found that for the canonical view, only two users failed to memorize the route. For the street-level view, three out of 12 users got lost, while for the regular top-down view, half of all participants took a wrong turn. This indicates that our algorithm significantly improves the ability to memorize a route in comparison to a standard top-down view. Finally, when asked about their preferred view for these preview videos, half the users opted for the canonical view algorithm, three picked the street-level perspective and one the regular top-down view. Two indicated that they would prefer a combination of the canonical view with the street-level or regular view.

4.4.4. DISCUSSION

The nature of our algorithm causes buildings to shear in a different direction very suddenly when traversing over them from a top-down view. This is due to the fact that the direction of $\vec{v}$ changes immediately. While we experimented with smoothing these transitions, such a solution diminishes the canonical view and results in a limited facade visibility when viewing a building from the top, which was one of the main problems we were aiming to solve. We also investigated shrinking the roofs, but this led to confusing configurations and negative user remarks.

Our current viewing algorithm does not seem to confuse participants and is sometimes even relatively subtle. Only when toggling back to a regular view did some of the users realize that there was an actual deformation applied, but they did immediately notice the loss of visual information. Furthermore, when keeping the camera above the

(a) Street in regular view.

(b) With canonical view.

Figure 4.13: Buildings are transformed away from the camera, which always prevents the street from being occluded when the camera is straight above it.

street, all houses will always fold outwards in a top-down view, giving a better overview of the street than with a regular view (Figure 4.13). In navigation applications, one could imagine constraining the camera to be located above the street to ensure this behavior.

Our algorithm produces transformed objects that are close to the canonical view, but as it is a shearing transformation, the roof is only translated. Nonetheless, a rotation of the roof would not be a good solution; when looking at a building in three-quarter view, it would result in deformations at the edges of the structure. Due to our shearing operation, such unwanted deformations are avoided, as demonstrated in Figure 4.14.

One may think of other, simpler methods to transform the buildings. One option is to simply widen the field of view. However, this causes all objects to appear significantly deformed, and results in a lot of occlusions. Another way is to rotate all vertices around the center of the bounding box's bottom face. We have implemented this method for the sake of comparison; however, it leads to severe deformations, resulting in the buildings being significantly stretched and sometimes even difficult to recognize.

(a) Additional rotation of the roof.

(b) Result with our view algorithm.

Figure 4.14: Rotating the roof causes deformation at the edges of the building.

(a) $\theta_c = 45°$.

(b) $\theta_c = 15°$.

Figure 4.15: The algorithm can be easily parameterized by supplying a canonical angle corresponding to the user's preference.

Overall, our approach has several interesting properties. The visualization seems clear without causing confusion, buildings approach canonical views without restricting navigation, a location on a street is never occluded when in focus, and the algorithm is parameterizable, in the sense that users can apply angle preferences (Figure 4.15).

4.5. CONCLUSIONS AND FUTURE WORK

We have presented a system to apply canonical views to buildings in 3D virtual cities without restrictions on the viewpoint. The technique dynamically transforms objects in real time based on the current camera position. Our algorithm rotates buildings away from the camera in top-down views, which reveals facades that are otherwise hidden. The transformation is easily parameterizable, allowing users to choose their own preferred canonical angle. User tests indicate that our viewing tool is subtle and does not cause significant confusion, and improves the recognizability of buildings while also being beneficial for navigation planning in the context of route memorization tasks. In the future, we would like to investigate more dimensions, including distance and orientation, when examining the influence of the canonical angle.

5

Expressive Single Scattering for Light Shaft Stylization

It does not do to dwell on dreams and forget to live.

J. K. Rowling

Light scattering in participating media is a natural phenomenon that is increasingly featured in movies and games, as it is visually pleasing and lends realism to a scene. In art, it may further be used to express a certain mood or emphasize objects. Here, artists often rely on stylization when creating scattering effects, not only because of the complexity of physically correct scattering, but also to increase expressiveness. Little research, however, focuses on artistically influencing the simulation of the scattering process in a virtual 3D scene. In this chapter, we propose novel stylization techniques, enabling artists to change the appearance of single scattering effects such as light shafts. Users can add, remove, or enhance light shafts using occluder manipulation. The colors of the light shafts can be stylized and animated using easily modifiable transfer functions. Alternatively, our system can optimize a light map given a simple user input for a number of desired views in the 3D world. Finally, we enable artists to control the heterogeneity of the underlying medium. Our stylized scattering solution is easy to use and compatible with standard rendering pipelines. It works for animated scenes and can be executed in real time to provide the artist with quick feedback.

Save for an extended introduction, this chapter is a verbatim copy of a publication in the IEEE Transactions on Visualization and Computer Graphics **23**, 7 (2017), by Timothy R. Kol, Oliver Klehm, Hans-Peter Seidel and Elmar Eisemann [KKSE17], extending a publication in the proceedings of GI: Graphics Interface by the same authors, presented at GI 2015 in Halifax, Canada [KKSE15]. As for the distribution of work, I implemented and devised the majority of the generated results, came up with the idea and implementation of the heterogeneity modification and animated transfer functions, wrote much of the GI paper, and nearly all of the TVCG paper.

Figure 5.1: Example of our stylized scattering. Left: physically correct single scattering using the original occluders. The leaves of the tree block most of the light, causing a rather subtle effect. Right: stylized scattering with *occluder manipulation*. Using our system, an artist can easily add holes into the shadow map of the tree, producing more pronounced scattering effects. While physically incorrect, it is not obvious for the viewer that the right image uses fake occlusion information. Surface shadows are created from the original shadow map.

5.1. INTRODUCTION

REALISTIC lighting, like the global illumination presented in Chapter 3, is crucial for many applications in computer graphics. However, the desired appearance may be subject to an artist's preferences. While a physically correct simulation is often a good starting point, we need *artistic* representations to further facilitate expressiveness. Besides global illumination, the *scattering* of light is another natural phenomenon that can drastically change the look of a scene. It occurs when light travels through a *participating medium*, such as air, where it can interact with particles. Among the most dominant effects caused by scattering are crepuscular rays, which are best described as being no more than *light shafts*. Light shafts not only add realism and spatial cues for a better scene understanding, but can also serve artistic purposes. They are visually pleasing, and therefore frequently found in art pieces, where they have a history of being *stylized* rather than being physically correct.

Figure 5.2 shows two pieces of art that illustrate some ways in which scattering can be stylized. In the left image, light shafts are used to emphasize the sailboat, a technique that is common in comics and animation movies to highlight a focus object. For this painting, the harshness of the lighting is particularly striking, as most real-world light shaft boundaries are rather smooth. In this sense there is a clear contrast between the upper light shaft, that exhibits a smooth behavior, and the lower part, which is more pronounced and clearly highlights the object of interest.

In the right image, light shafts seem to be added and removed at will, taking little note of solid objects that would normally block the light, such as the group of trees on the left. The color of the light shafts seems to vary between green and yellow hues in a physically incorrect but appealing manner. Another point of interest is that, unlike in the image on the left, the light shafts have a rather irregular appearance. While perhaps somewhat exaggerated here, this effect can also occur in nature due to a medium being *heterogeneous*, which means that the scattering particles are not uniformly distributed, creating a spatially varying density. Note that both images look plausible and visually pleasing, despite the physically incorrect light shafts and the fact that they do not correlate with

Figure 5.2: Example of stylized scattering in (concept) art. Left: the light shafts have a very sharp boundary in order to emphasize the sailboat. Source: Roberto Gatto (www.robertogattoart.com); used with permission. Right: light shafts seem to start and stop in mid-air for no clear reason other than aesthetics. Source: Jonas De Ro (www.jonasdero.be); used with permission.

surface shadows. It can be concluded that for artists, stylization of light shafts is more of a rule than an exception, and serves an important role in many artworks. Further, stylized scattering does not necessarily produce confusing or implausible results. Such manipulations seem widely accepted, and may not even be noticed by the untrained observer.

Since the dawn of computer graphics, however, most research in scattering has focused on efficiently obtaining *physically correct* and *realistic* images. For a realistic scattering process, light shafts manifest as follows; when light illuminates an optically thin participating medium like air, it often bounces off particles before reaching the eye. If a single bounce occurs, the process is called *single scattering*. Visually, the medium through which the light travels is lit. However, when part of the light is blocked by an object – which we call an *occluder* – before it bounces, it cannot be scattered towards the viewer and these areas appear darker.

Simulating single scattering in homogeneous media can be achieved with real-time performance [CBDJ11, KSE14]. As in nature, light shafts depend on the scene layout and not on any specific scene element, which makes stylization difficult as the appearance cannot be modified directly. In consequence, there is a need for specialized rendering techniques to enable direct and easy artistic control, but very few algorithms exists [NJS*11, KISE13].

Our work offers new ways of modifying light scattering to produce effects similar to the aforementioned stylization used in art. Hereby, we hope to give artists more freedom, enabling them to carry on the trend of stylized scattering from more traditional art to video games and film effects. We employ three manipulation concepts in our work. First, we introduce *occluder manipulation*, which allows the user to add and remove light shafts (Figure 5.2, right) or sharpen them (Figure 5.2, left) . Figure 5.1 illustrates the addition of light shafts, which leads to a brightened scene and additional details, influencing the appearance significantly. Second, our algorithm provides simple controls to achieve various styles, expressive color changes, and mood alterations in a scene. For this, we introduce two techniques: *transfer functions* and *light map optimization*. Third, we introduce a solution to control the heterogeneity of the medium by enabling interactive changes of the volume's properties.

As with most artistic tools, it is crucial that users receive interactive feedback to explore possible parameter choices. For this reason, we focus on real-time methods. Our high-level definitions enable the transfer of a general style to scenes with different geometry, camera, and light settings, and support animation. Hereby, we introduce effective means for controlling the scattering in 3D scenes for interactive applications.

Specifically, our work makes the following contributions:

- light shaft addition, removal, and enhancement using image-based occluder manipulation;
- light shaft color changes via user-editable transfer functions based on view ray properties, and light map optimization based on user-drawn strokes;
- light shaft irregularity by controlling the medium's heterogeneity with a 3D painting tool; and
- light shaft animation through dynamic occluder manipulation, and key-framed transfer functions for animated scenes.

5.2. Related Work

Cinematic relighting [PVL*05, RKKS*07] generally gives artists the possibility to predict the final rendering in order to support them in tasks such as lighting design and material definitions. However, the underlying calculations in these systems are usually physically correct, and the possibilities for stylization are often restricted to the scene.

5.2.1. General Stylization

In recent years, solutions to influence physics for the purpose of expressiveness have received increasing attention and there are several approaches to stylize natural phenomena. Modifications of the light transport [OKP*08, KPD10, STPP09], shadows [MIW13, DCFR07, ASDW14, PŠNB13], caustics [GSLM*08], motion blur [SSBG10], or depth of field [LES09], have been proposed to significantly influence the appearance of a scene and to guide the observer to specific regions of interest. Similarly, other stylization techniques have been demonstrated for focus control [CDF*06]. For a detailed exposition of similar work on appearance and lighting editing, we refer to a recent overview [SPN*16].

5.2.2. Stylized Scattering

Regarding scattering stylization, only a few approaches have been suggested. Artistic beams [NJS*11] let the user modify individual light rays by influencing shape, falloff, and color. The modifications are used to find a plausible, optimized mapping to properties of the participating medium. Treating light rays individually can be advantageous, but global control becomes more time-consuming and difficult. In contrast, our stylization method uses parameters derived from scattering to directly map to the final result.

One can also rely on a set of painted input images to find the optimal volume parameters to best match the provided target images [KISE13]. In this case, the volume parameters are stored in a voxel grid, which limits the possible resolution and performance, despite the employment of an efficient process. Furthermore, defining the input

requires a certain artistic skill, and the final rendering is bound by the actual physical process, limiting the potential expressiveness.

Furthermore, Hašan and Ramamoorthi [HR13] presented a method for efficiently re-rendering a scene for which the volume's single scattering albedo values have been modified. While stylized scattering is an application, their approach focuses rather on increasing the performance of re-rendering, and not so much on design and stylization tools. Moreover, they require a computationally expensive pre-process due to taking the full light transport for dense volumes into account, while we aim for real-time solutions for single scattering in optically thin media.

In our solution, we want to make it easy to define plausible results, but also enable more expressive solutions, potentially leaving the physical behavior. For this, efficient computation is key to allow artists to rapidly explore various options. Hence, we focus on existing real-time solutions for single scattering. There are several options for efficient computation of single scattering; min-max mipmaps [CBDJ11], voxelized shadow volumes [Wym11], shadow volumes based on shadow maps [BSA10], or prefiltered single scattering [KSE14]. While approaches for multiple scattering exist [ERDS14], their precision is still relatively low due to the use of a coarse grid, which is why we concentrate on single scattering only.

5.2.3. SPECIFIC TECHNIQUES

Occluder manipulation as a possible means of stylization has been applied before in the form of proxy geometries to modify light transport in a scene. Schmidt et al. [SNM*13] introduced the idea of path-proxy linking, defining invisible copies of scene objects, which are modified using affine transformations and only affect a certain individual component of the light transport, such as shadows. While we also modify occluders to change scattering behavior, we propose specialized and parameterizable manipulation methods that are useful for stylization. To this end, we manipulate occluders using morphological operators and work in the space of a 2D shadow map as this efficiently gives direct control over creating, removing, and enhancing light shafts.

A change in color is often achieved by the use of transfer functions, such as for surface shading [BTM06] or volume visualization [GMY11]. Unlike in volume visualization [GMY11], we do not input medium properties at a point in space, but evaluate scattering-related values along the view ray as parameters for our transfer functions. Alternatively, we propose a light map optimization that employs inverse rendering, a general concept that derives scene parameters from 2D user input. Schoeneman et al. [SDS*93] were first to optimize light properties (intensity and color) in a least-squares sense to satisfy user-defined target images. Klehm et al. [KISE14] applied the concept to environmental illumination of heterogeneous media. We focus on the special case of thin homogeneous media and optimize a light map for a directional light source. In contrast to the aforementioned work, users only draw strokes to define light shaft colors instead of a full image.

Due to performance constraints, heterogeneous media have received limited attention for real-time applications. Zhou et al. [ZHG*07] proposed a composition of simple

radial basis functions. This approach, despite limiting the amount of detail of the heterogeneity, allows for easy designing of a heterogeneous medium by placing the radial basis functions with the aid of a brush and eraser. However, the focus lies on an approximation of the physically correct result and uses an analytic model to evaluate the basis functions. Instead of designing a volumetric element, we use heterogeneity to locally enhance the indirect effect of light shafts.

5.3. REAL-TIME SCATTERING BACKGROUND

Before discussing our algorithm, we will first give a brief introduction to single scattering. Radiance caused by single scattering from a single directional light towards a camera at $\mathbf{x}$ from direction ω_i is computed by integrating the view ray up to the first visible surface at distance s:

$$L_\mathrm{scat}(\mathbf{x},\omega_\mathrm{i}) = \sigma_\mathrm{t}\,\rho\,f\int_0^s e^{-t\sigma_\mathrm{t}}\,V(\mathbf{x}_t)\widetilde{L}_\mathrm{i}(\mathbf{x}_t)\,dt, \tag{5.1}$$

assuming a homogeneous medium with extinction coefficient σ_t, scattering albedo ρ, and constant phase function f. $\widetilde{L}_\mathrm{i}(\mathbf{x}_t)$ denotes the unoccluded, incoming light at the scattering point and $V(\mathbf{x}_t)$ the corresponding visibility from the light source. Note that we skip attenuation from the light source to the sampling point $\mathbf{x}_t$, as this results in complete attenuation for a directional light source located at infinity.

Factoring out visibility [BCRK*10, CBDJ11, Wym11, KSE14] is a common approximation, which we also identify as a useful parameter for stylization purposes. The equation then becomes:

$$L_\mathrm{scat}(\mathbf{x},\omega_\mathrm{i}) = \sigma_\mathrm{t}\,\rho\,f\int_0^s e^{-t\sigma_\mathrm{t}}\widetilde{L}_\mathrm{i}(\mathbf{x}_t)\,dt\;\;s^{-1}\int_0^s V(\mathbf{x}_t)\,dt. \tag{5.2}$$

The first integral can be computed analytically [PP09] for a variety of light sources and is typically assumed to be constant for a directional light source. The second integral represents an average visibility and can be computed efficiently with an image-based solution, using a shadow map rendered from the light source, and a depth map rendered from the camera. Given a pixel in the image from the camera, its underlying depth value (distance to surface s) and point $\mathbf{x}$ define a segment in space, along which the visibility should be integrated. Using a ray marching process on the shadow map along this segment, the light visibility $V(\mathbf{x}_t)$ for each of these positions can be tested with a simple shadow map lookup. While being conceptually simple, this approach is not very efficient. Various acceleration methods [BCRK*10, CBDJ11, Wym11, KSE14] have been proposed, and we use the solutions by Chen et al. [CBDJ11] and Klehm et al. [KSE14], which work comparably well and are executed on a shadow map. In all examples, we use the Henyey-Greenstein phase function for scattering to increase the initial physical correctness. However, other choices (even a simple constant) would be valid, too.

5.4. STYLIZED SINGLE SCATTERING

Our method consists of three major techniques to perform scattering stylization, which can also be combined. The first modifies occluders in order to influence the scattering

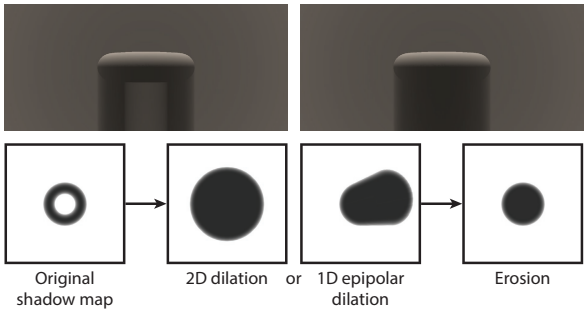


Figure 5.3: Morphological filtering. Top: rendering a torus with directional light coming straight from the top with the unmodified (left) and hole-filled (right) shadow map. Bottom: shadow maps. From left to right: original shadow map; 2D dilation; 1D epipolar dilation; subsequent erosion, which for this particular scene yields identical results for both dilations.

appearance by enhancing, adding, or removing light shafts. The second technique consists in the colorization of light shafts. We enable the definition of a transfer function that can be used to drastically influence color, brightness, and contrast. We rely on values (e.g., scene depth) derived along the view ray to define the final output color. As an alternative to the transfer function, the light shaft colors can also be modified by a light map optimization, driven by strokes drawn by the user. The third technique focuses on giving the light shafts a more irregular look by using an efficient algorithm to approximate a heterogeneous medium. The heterogeneity can be manipulated in multiple ways using a 3D painting tool. In the following, we will give an overview of these three techniques.

5.4.1. OCCLUDER MANIPULATION

The main observation is that the appearance of single scattering in a scene largely depends on the number, size, and contrast of light shafts. They become visible due to differences in the light visibility along neighboring view rays (i.e., screen pixels). These differences are often caused by openings in the occluder (i.e., holes through which the light can shine), such as a gap in the clouds. Our system enables artists to modify light shafts by editing the shadow map, which is used to capture the occluders in the scene. Here, we present the various modification options.

HOLE FILLING

Physically-based scattering can sometimes produce unwanted effects. As an example, while the left image in Figure 5.13 is visually pleasing, the tiny light shafts along the ground give a somewhat chaotic nature to this otherwise serene scene. To appreciate the image more, we *remove* these distracting details caused by small holes in the occluders. By closing holes, we reduce the trees' emphasis and simplify the light shaft appearance.

For this hole filling, we make use of an image-based approach in which we directly modify the shadow map used for the scattering evaluation. One solution for hole filling is the use of 2D morphological filters. More precisely, a *closure* operation is applied, consisting of a *dilation* followed by an *erosion* (both are elementary operators). A dilation in the shadow map replaces a value by the minimum in a certain neighborhood, whereas

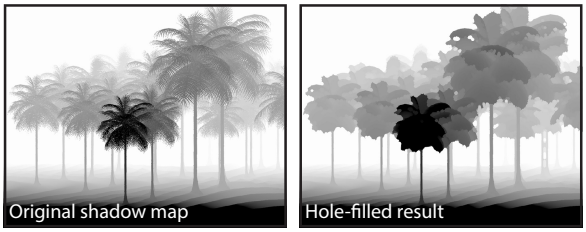


Figure 5.4: Hole filling applied to the shadow map of the *Palm tree* scene.
Left: original shadow map. Right: result using a kernel size of 10 pixels.

an erosion replaces a value by the maximum. All holes that are removed in this way lead to the elimination of the corresponding light shafts. For illustration, an exaggerated example is shown in Figure 5.3, where we remove the entire hole in the torus.

One important factor is the neighborhood to be considered around each pixel, often referred to as the *filter kernel*. Traditionally, a square or circle is applied; per default, we use the latter. We let the user control the kernel size, which defines the strength of the hole filling process. The resulting shadow map for the *Palm tree* scene is shown in Figure 5.4.

Silhouette Enhancement We have shown how to remove details from the scattering result, but in some cases the opposite is desired. If scattering is very subtle, an artist may want more prominent light shafts; e.g., as in the middle image in Figure 5.13. To this extent, we can take an approach very similar to hole filling to *enhance* light shafts caused by the silhouette of an object. The idea is to extrude objects along the view rays, hereby increasing their thickness. We achieve the extrusion by a 1D dilation in the shadow map away from the epipole (the camera position projected onto the shadow map). Hereby, the occluder extension is always hidden by the object itself, as the camera only sees the first surface. However, the object's volumetric shadow is increased, as the extrusion is usually visible from the light source.

The *1D epipolar dilation* works as follows. For each texel t, we construct a 1D kernel along the line in the shadow map from t towards the epipole l. A standard dilation computes the minimum of all samples within the kernel; however, this does not satisfy the required extrusion from the camera position, as it effectively extrudes points in the plane orthogonal to the light direction. To solve this, we need to consider the changing z-coordinate along the view ray, which forms a sloped filter kernel. Thus, we modify the new depth z'_t of t given an input sample s as follows:

$$z'_t = \min(z_t, \frac{\text{dis}_{2D}(t,l)}{\text{dis}_{2D}(s,l)}(z_s - z_l) + z_l), \tag{5.3}$$

with $\text{dis}_{2D}(t,l)$ denoting the 2D distance between texel t and epipole l as projected on the shadow map, with z_t and z_s the depth values of t and s in the shadow map, and z_l the z-coordinate of the epipole l. This results in a mix of z_s and z_l modulated by the ratio between the distances t to l and s to l. This value, as per the definition of a dilation, is used only if it is lower than the current lowest depth value z_t (hence, the min). The

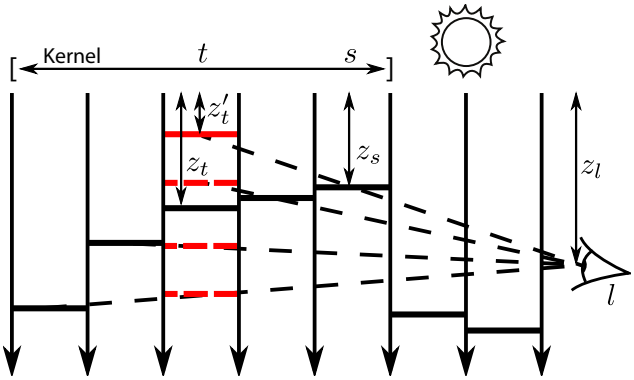


Figure 5.5: 1D epipolar dilation. Areas between the parallel light rays represent texels, with the black horizontal bars denoting their depth in the shadow map. A 1D kernel (of size 5 in this example) is constructed for texel t with depth z_t. Given the epipole l with depth z_l, the whole kernel is sampled and Equation 5.3 is applied (red horizontal bars). The minimum value z'_t is in this case found for the sample s with depth z_s.

process is illustrated in Figure 5.5.

When applied, the enhancement of the light shafts is twofold. First, the 1D epipolar dilation fills holes, removing small light shafts. Second, as the extrusion process is aligned with the view rays, it increases contrast between neighboring pixels, leading to sharper boundaries, as seen on the left in Figure 5.2. In consequence, occluders block more light as they are effectively larger and the contrast difference between light shafts is pronounced. Figure 5.6 shows the resulting shadow map for the *Scarecrow* scene.

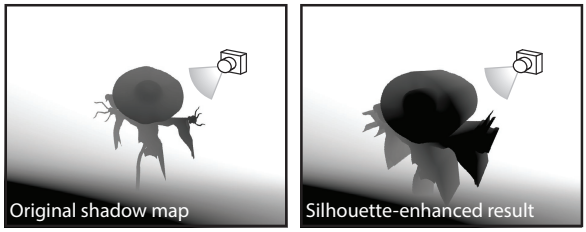


Figure 5.6: Silhouette enhancement applied to the shadow map of the *Scarecrow* scene. Left: original shadow map. Right: result using a kernel size of 100 pixels; the object is visibly extruded in the view direction.

Hole Creation Silhouette enhancement makes the light shafts more prominent, but to obtain a visible effect, sufficient light shafts need to be present in the scene. Situations can occur where this is not the case, like in the right image in Figure 5.13. Here, only a few light shafts fall through the flowerbed, creating only subtle scattering, no matter how much silhouette enhancement is applied. Yet an artist may want more light to burst through the flowers. For this reason, besides removal, we also offer a solution to *add* additional light shafts. In contrast to the previous hole filling operation, we instead create random holes, the result of which can also be seen in the teaser. Two simple steps are applied: we first generate a hole map and subsequently use it as a mask to modify the

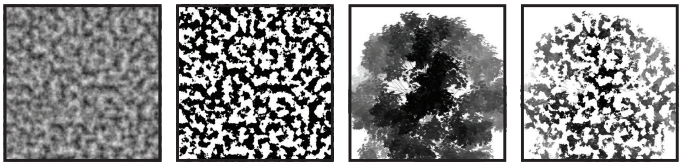


Figure 5.7: Hole creation results are shown in Figure 5.1. Left to right: Perlin noise with a user-defined frequency; thresholding with a user-defined hole probability; original shadow map of the tree; shadow map with holes created from the hole map.

rendering of a shadow map.

The hole creation process is steered by various parameters to influence the average size, number, and density of the holes. In order to avoid perfectly uniform holes, for which the resulting light shafts can look too regular, we make use of *Perlin noise* [Per85]. By using a thresholding operation, we can transform it into a binary mask exhibiting randomization in shape, which we use as our hole map. Given some 2D texture coordinates t, the binary mask has a value of $H(t)$ given by

$$H(t) = \begin{cases} 0 & \text{if } N(tgf) \geq h \\ 1 & \text{otherwise} \end{cases}, \tag{5.4}$$

with h the user-defined hole probability or threshold value, g the Perlin noise gradient grid size, f the user-defined frequency, and $N(q)$, for $q = tgf$, given by

$$N(q) = \sum_{i=0}^{o-1} \frac{p^i P(q)}{\max(1, o-i-1)}, \tag{5.5}$$

where o is the number of user-defined octaves, p the user-defined persistence and $P(q)$ the classical 2D Perlin noise. The process is illustrated in Figure 5.7, where Perlin noise is created with parameters $f = 0.3$, $p = 0.5$, and $o = 5$, which is then thresholded using a hole probability $h = 0.5$, which results in the displayed hole map.

There are several ways to apply the hole map to influence visibility queries. Typically, we want to limit the effect of holes to pre-defined objects and the following solutions are only applied to those objects. One way is to discard fragments in the shader when rendering the shadow map if they belong to one of the pre-defined objects and the hole map value is 1 at their location. Alternatively, we render the objects in a separate shadow map and apply a max composition with the hole map. Basically, a hole is defined by pushing the depth values at a certain location to 1, i.e., the far plane. However, in this case, shading calculations have to test visibility against two shadow maps. While the first method is more elegant, the latter may be easier to integrate into an existing pipeline and makes a local adaptation of resolution per object possible.

Using Perlin noise naturally enables animation, for which we use a two-step lookup. First, a time-based offset is used to rotate the sample point before reading the value. We then use the resulting noise value as an offset to the original sample point and perform

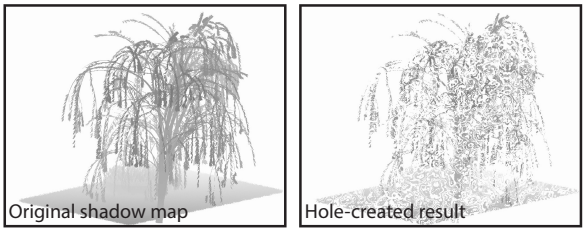


Figure 5.8: Animated hole creation applied to the shadow map of the *Flowerbed* scene. Left: original shadow map. Right: resulting shadow map. Note that in this case, the hole creation was also applied on the ground, which may give strange results depending on the viewpoint. As mentioned before, we simply solve this by using two shadow maps and compositing them later on when necessary.

an additional noise lookup. Effectively, we apply a time-dependent noise to the lookup position, which causes an apparent global movement of the holes, but also a change in their size and structure. Hereby, a time-based rotation proved to give appealing results, while keeping the overall appearance consistent. Using this technique, we can simulate the effect of leaves moving in the wind or fake the notion of movement through participating media. Figure 5.8 shows the result when applied to the shadow map of the *Flowerbed* scene.

While this method might seem mostly unsuitable for solid objects, under certain conditions, e.g., to emphasize a character or simulate motion (Figure 5.9), it can still be useful. The main application area is still on less recognizable shapes, such as foliage or other detailed geometry.

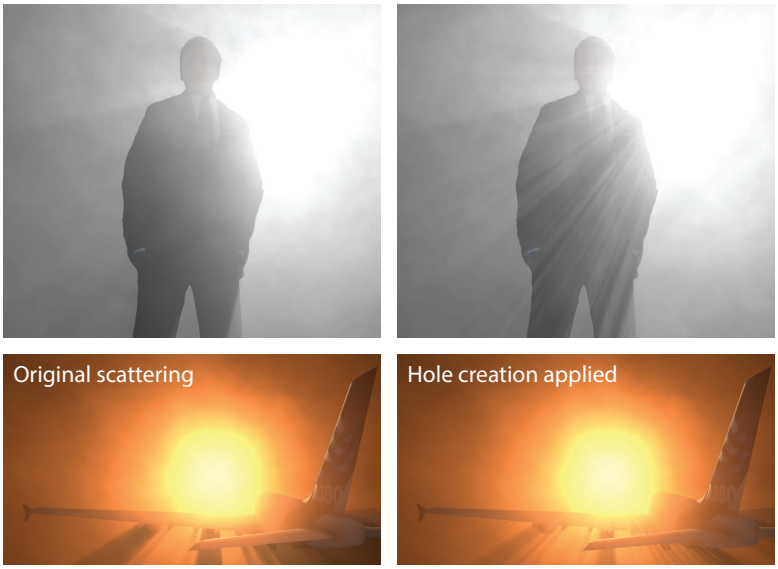


Figure 5.9: Hole creation applied to solid objects. Left: the original image. Right: image obtained by creating holes in the shadow map. Top: emphasizing a character. Bottom: simulating motion.

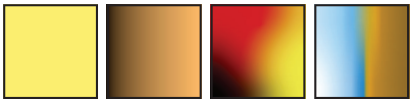


Figure 5.10: Different layer types supported by our TF editor: solid, gradient, diffusion, and image layers.

5.4.2. COLOR MODIFICATIONS

Typically, the resulting color of volumetric scattering is defined by the color of the light source or albedo of the scattering volume (see Equation 5.1). We propose two orthogonal techniques that enable users to colorize light shafts individually: transfer functions and optimized light maps. Both approaches target a goal-driven editing experience, where the user directly specifies color sets.

Transfer Functions *Transfer functions* are an effective way to influence the light shaft colors. More specifically, a transfer function (TF) maps the properties of a view ray (effectively a pixel) to the scattering component's output color. In order to make their definition easy to specify for the user, we focus on a mapping of two parameters to a color. Consequently, the transfer function can be defined by a 2D texture, similar to the *X-Toon* approach [BTM06]. In this way, a TF is also not limited to a single scene; instead, it is possible to transfer the mood caused by a TF to another scene.

As a view ray is uniquely defined by its underlying pixel, an artist can easily influence the entire scene appearance in a consistent and effective way by defining and modifying a transfer function. As an example, an artist might want a certain set of pixels with similar properties to change to an orange color for stylization purposes, which can easily be achieved by a transfer function. To this extent, the properties along a ray would simply be mapped to the desired color.

In practice, we use the average visibility along the view ray and the linearized depth of the first surface as parameters for influencing the scattering component's color. Based on these parameters, when using a gradient in the transfer function, the result remains plausible, as the two parameters are often used in realistic approaches as well. Further, please note that these values are along the view ray so that we rely on 3D information; otherwise the stylization would appear to be a 2D overlay, which becomes very apparent when the camera moves. We also tested other parameters, such as the average position of visible samples along the view ray, and the angle between the view ray and light direction, but these seem more difficult to use and are therefore not included in our results.

Artists are able to interactively design the TF in our framework to explore various possibilities in real time. The on-the-fly editing of the TF texture uses a painting utility based on layers. In our prototype, solid colors, gradients, images, and special *diffusion layers* are supported (see Figure 5.10), which can be blended to produce the final TF, using multiplicative, additive, or alpha blending.

Diffusion layers contain constraints, consisting of a position in the TF texture, a color, and an alpha value. The user can place constraints anywhere in the texture, whose axes represent the parameter domain of the TF. The constraints are diffused throughout the

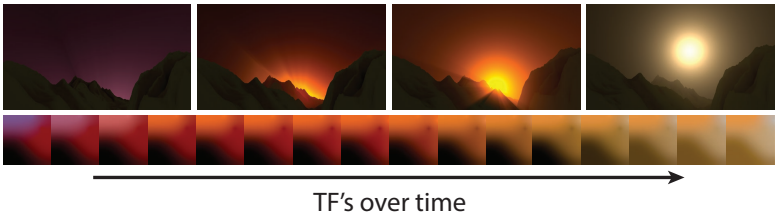


Figure 5.11: A sunrise created using several transfer functions in the *Terrain* scene. Top: resulting scattering. Bottom: corresponding TFs due to linear interpolation between five user-defined TFs. It can be seen that key-framing the TF easily extends color modifications of light shafts to the time domain.

entire layer similar to diffusion curves [OBB*13]. A single constraint results in a uniform color, two produce a gradient, and more create complex color combinations, like the third image in Figure 5.10. Additionally, other diffusion constraints could be integrated [BEDT10]. For stacked layers, alpha blending is guided by the alpha values of the constraints. This diffusion creates smooth transitions in parameter space and, thus, all pixels in the scene with similar parameters change similarly. This property makes it easier to produce consistent definitions and renders the tool very effective.

However, defining the TF directly in parameter space can be cumbersome, and it may be more desirable to define scattering directly at a point in space. For this reason, an artist can simply select a location by clicking on the screen to define a corresponding 3D constraint. The constraint's position in parameter space is computed by querying the pixel's underlying view ray parameters, and the user can choose the color constraint to be placed in the transfer function. Additionally, we allow the recovery of the underlying position in the scene. This position is expressed in barycentric triangle coordinates, which makes it possible to project the point in each frame to the screen and move the constraint accordingly in the parameter space of the transfer function. To further extend the support of dynamic scenes, we also introduce key-framed transfer functions to produce stylized animations by smoothly interpolating between the TFs defined over time. In this way, expressive scattering can be extended to dynamic scenes, which is especially interesting for pre-defined scene animations, e.g., an in-game cut-scene with a known camera path. Figure 5.11 shows this technique to produce a sunrise in the *Terrain* scene.

Finally, the stylization can be used in combination with a standard scattering model; here, the TF can be monochrome, to serve as a modulator of the scene appearance, while the light color is defined by the source itself. Consequently, surface lighting and scattering remain consistent.

Light Map Optimization Our approach also offers control over light color and, thus, the color of light shafts for a directional light source. A light map stores the light's color for a given angular direction (in case of a directional source, it can be interpreted as a projection of a texture in the scene). By only influencing the light emission, the results remain physically plausible, which can be a desirable goal in certain situations.

Formally, a light map defines the spatial variation of the unoccluded incoming light $\widetilde{L}_{\mathrm{i}}(\mathbf{x}_t)$ in Equation 5.1. At a point $\mathbf{x}$ the light color is determined by projecting the point

into light space and performing a texture lookup using its x, y coordinates. As before, we focus on directional light. Hence, the light space is defined via an orthographic projection along the light direction. In principle, it could be the same transformation as the shadow map of the light source, but we allow an independent definition to make a differing resolution and focus possible.

We face two main challenges when using such light maps to stylize light shafts. First, there is no intuitive link between individual texels of a light map and the colors perceived after the scattering. Second, we target the creation of a static light map for the entire scene; i.e., for every possible view. Light shafts, however, are highly view-dependent, which in case of static illumination (and thus a view-independent light map), leads to an over-constrained problem. We solve these issues by using an optimization scheme to find a light map, which fulfills the user constraints for a given set of input views as well as possible. In other words, for a given view, the user can draw simple color strokes to express a desired color for the seen light shafts (top row in Figure 5.17). When all constraints have been collected, we apply an approach similar to the one proposed by Klehm et al. [KISE14] to derive an optimal light map.

Formally, we treat the strokes in the user-defined views as a collection of N *constraint pixels*. Given the corresponding view, a pixel with index $k \in [1, N]$ corresponds to a ray (origin $\mathbf{x}^k$ and direction ω^k). The stroke's color to which the pixel belongs defines the desired color L^k_{desired}. While we want to enable users to define colors, we do not want them to provide exact scattering results. Instead, we multiply the painted color with the value that would be obtained for a white light map. $L^k := L^k_{\text{desired}}\, L^{\text{white}}_{\text{scat}}(\mathbf{x}^k, \omega^k)$. Hence, the color definition is independent of any shading effects. Our goal is to derive RGB color values $L_e(s,t)$ for the texels of a light map such that rendering with the light map, yields the expected scattering result $L_{\text{scat}}(\mathbf{x}^k, \omega^k)$, which should match the target value: $L^k = L_{\text{scat}}(\mathbf{x}^k, \omega^k)$.

Stacking the texels of the light map in a vector $\vec{x} := (\ldots, L^i_e, \ldots)^\top$ and the user indications (as defined above) in a vector $\vec{b} := (\ldots, L^k, \ldots)^\top$ defining the target radiance values, we can describe the light transport in form of a linear dependency: $\mathbf{T}\vec{x} = \vec{b}$, where the values of $\mathbf{T}$ are fully described by Equation 5.1. Unfortunately, it is unlikely that a general solution exists, which would satisfy all equations at once. A simple example is a checkerboard, which cannot be achieved by modifying the light map alone, as light shafts always become darker the farther they are away from the light source. Hence, we opt for a compromise by finding a solution in the least-squares sense: $\mathbf{T}^\top\mathbf{T}\vec{x} = \mathbf{T}^\top\vec{b}$. Accordingly, the optimal $\vec{x}$ is found by minimizing the quadratic function $f(\vec{x}) = \frac{1}{2}||\mathbf{T}\vec{x} - \vec{b}||^2$ with gradient $\nabla f(\vec{x}) = \mathbf{T}^\top(\mathbf{T}\vec{x} - \vec{b})$.

While any off-the-shelf optimizer can be used, we employ an iterative conjugate gradient method [She94] to find the optimal $\vec{x}$. The advantage of a conjugate gradient method is that matrix $\mathbf{T}$ does not need to be stored explicitly and we can easily employ the GPU. We only need to be able to compute the matrix-vector multiplications $\mathbf{T}\vec{x}$ and $\mathbf{T}^\top\vec{b}$. $\mathbf{T}\vec{x}$ corresponds to evaluating the same equations as for rendering with the light map $\vec{x}$, while the term $\mathbf{T}^\top\vec{b}$ is an inversion; i.e., pixel values are back-projected onto the light map. While the standard rendering process reads a light map value at each march-

ing step to accumulate in-scattered light, the back-projection to compute $\mathbf{T}^\top \vec{b}$ writes the pixel value $\vec{b}^k$ modulated by the scattering weights to the light map. For the j^{th} marching step with step size t for pixel k, the scattering weight is $\mathbf{T}[k,j] = \mathbf{T}^\top[j,k] = \rho\, f\, V(\mathbf{x}_j)\, e^{-jt\sigma_t}$ (cf. Equation 5.1) with $\mathbf{x}_j = \mathbf{x}^k + jt\omega^k$. The equation being quadratic, the solution is unique, and we can initialize our light map with a constant.

While the underlying process of the light map optimization is more involved in comparison to the use of transfer functions, this complexity is completely hidden from the user. The approach maintains physical plausibility and is easily controlled via a painting metaphor, in which users can freely decide on the viewpoints and strokes they paint, leading to a spatially varying control.

5.4.3. HETEROGENEITY MODIFICATION

As illustrated on the right in Figure 5.2, light shaft irregularity – caused by a heterogeneous medium – plays an important role for stylization. We aim to enable artistic control over the heterogeneity, for which we propose three stylization concepts: first, denoting for locations in the scene to what extent they exhibit heterogeneity; second, controlling the variation and patterns in these irregular areas with locally varying frequencies; and finally, indicating the scattering intensity for scene locations.

In order to apply these concepts, we first need to address the way in which we will represent heterogeneity. A naive representation would define the medium in a fine 3D grid that contains a spatially varying extinction coefficient. To compute the final scattering, the grid is then sampled using ray marching, simultaneously resolving the visibility (cf. Equation 5.1). For high-quality light shafts, we would need a substantial amount of sample points to avoid discretization artifacts in the visibility sampling, which would yield low performance, but efficient solutions for heterogeneous media are currently lacking. Instead, we make use of two approximations. First, we apply heterogeneity in a post-process by modulating the homogeneous result, which delivers a pleasing appearance. Second, we observe that heterogeneity in thin media does not lead to strong high-frequency changes, which allows us to coarsely sample this information, making it possible to use a coarser grid.

In practice, we consider a 256^3 grid, the *noise volume*, filled with 3D Perlin noise values between 0 and 1, like its 2D counterpart described in Section 5.4.1. Note the shadow map resolution is typically much higher. To compute a pixel's final color, we multiply the scattering result L_{scat} (see Equation 5.2) by the average heterogeneity $h(\mathbf{x},\omega_i)$ for this view ray, obtained from ray marching through the noise volume (Figure 5.12):

$$\hat{L}_{\text{scat}}(\mathbf{x},\omega_i) = L_{\text{scat}}(\mathbf{x},\omega_i)\; h(\mathbf{x},\omega_i)$$
$$h(\mathbf{x},\omega_i) := s^{-1}\int_0^s N(\mathbf{x}_t)\, dt.$$

To stylize this result, we define three *blending volumes*, which control the amount of heterogeneity, the noise frequency, and the scattering intensity:

$$h(\mathbf{x},\omega_i) := \int_0^s I(\mathbf{x}_t)\; (\; \alpha(\mathbf{x}_t) N(\mathbf{x}_t, f(\mathbf{x}_t)) + (1-\alpha(\mathbf{x}_t))\;)\, dt.$$

Figure 5.12: Comparison between using a homogeneous medium (left) and our heterogeneity approximation (right) in the *San Miguel* scene.

The heterogeneity blending volume defines the linear interpolation α between the values obtained from the noise volume and a constant homogeneity. The noise frequency blending volume influences the noise frequency by defining f, the mipmap level used to look up the noise. Hereby, local noise modifications become possible, while the actual Perlin noise parameters can be used to define a global control of variations and patterns in the heterogeneity. Finally, the intensity blending volume defines I, which scales the overall scattering contribution.

The blending weight volumes are modified via a 3D painting tool consisting of a sliding plane orthogonal to the camera direction, which the user can move along its normal. An ellipsoidal brush and eraser, with a user-controlled size, opacity and hardness, are available to draw on this plane, which produces a 3D splat in the selected blend volume. Note that the noise and blending volumes can be baked into one volume to make rendering more efficient.

5.5. RESULTS AND DISCUSSION

To assess our methods, we present our stylization results below. As our methods are executed on the fly, we also discuss performance and the applied optimizations.

5.5.1. STYLIZATION

Occluder Manipulation Figure 5.13 demonstrates occluder manipulations. The *Palm tree* scene exhibits many small light shafts that are especially noticeable under camera animation. Using our closure operation to fill holes in the shadow map, we can effectively remove small light shafts. Note that the 2D filter can cause artificial occlusions in mid-air due to the extension of the occluders into empty space, which can produce subtle dark halos around objects. Alternatively, one could use a 1D epipolar closure, which forces the occlusion to be behind visible scene objects. Nonetheless, in practice, this artifact goes usually unnoticed (as in the *Palm tree* scene).

The *Scarecrow* scene (Figure 5.13, center) demonstrates the enhancement of light shaft edges using a 1D epipolar dilation. Here, the effect is used to increase a feeling of fear and leads to an emphasis of the object. A similar harshness is often used in comics to illustrate activity and stress the importance of an object. As the technique practically removes some of the scattering, it may darken the image slightly, which can be compensated for using a transfer function. Such a combination is shown in Figure 5.16.

Finally, hole creation enables artists to add light shafts, as illustrated in the *Flowerbed*

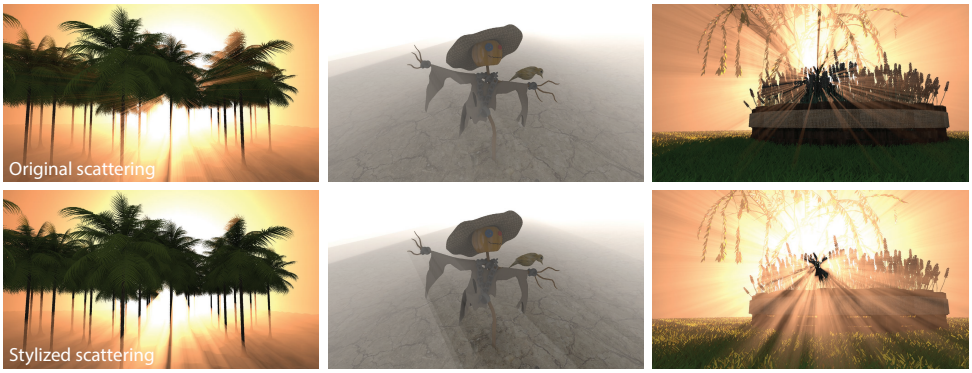


Figure 5.13: Stylized scattering applied on the *Palm tree, Scarecrow* and *Flowerbed* scenes using our occluder manipulation tools. Top: original images with physically-based scattering. Bottom: resulting images. From left to right: hole filling, silhouette enhancement and hole creation.

scene (Figure 5.13, right). Objects that initially exhibit a set of complex light shafts can profit from this operation. As discussed in Section 5.4.1, the Perlin noise parameters can be modified to match the intended appearance. Figure 5.14 shows the effect of different parameters on the teaser's example scene. Small holes (bottom) can be reduced in size until they average out due to the integration along the view ray, while large holes (top) can drastically simplify the scattering appearance. The control supplied by these parameters enables transitioning between physically plausible scattering with the original occluders, and the exaggerated alternatives. The complex geometry of the tree makes the hole creation process appear very natural. In general, we expect hole creation to be mostly applied in similar situations.

Color Modifications Transfer functions make it possible to achieve quick and impressive changes in the overall appearance of the scattering process by defining colors independently of surface, light, or medium parameters. These modifications enable artists to quickly redefine the mood of a scene, as illustrated in Figure 5.15. Here, the TF cre-

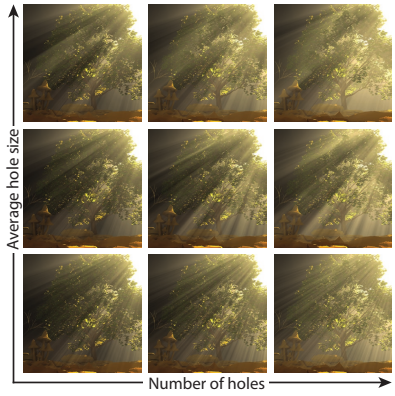


Figure 5.14: Effect of Perlin noise parameters on the resulting scattering.

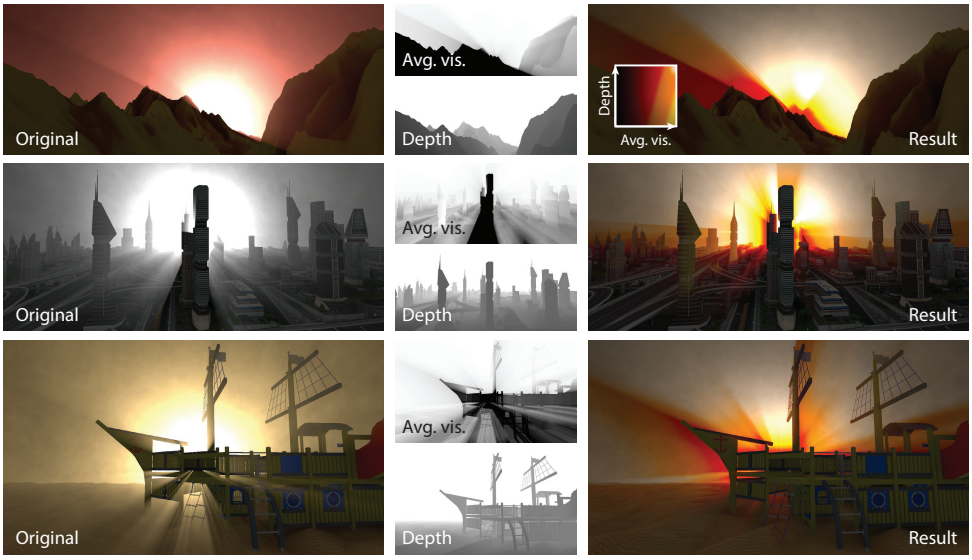


Figure 5.15: Expressiveness of transfer functions. Physically-based scattering (left) is stylized (right) using a TF (inset), which is parametrized by the average visibility and linearized depth of the view rays (center).

ates an alarming atmosphere in the scenes by adding multiple colors with strong edges between them leading to quantization effects in the resulting scattering. Figure 5.15 also illustrates the possibility to transfer a given style from one scene to the next; the *Terrain* scene's TF can directly be used to stylize two others. Additionally, the *Playground* and *City* scenes are both examples of how stylized scattering can emphasize an object.

In general, all our techniques can be used in combination to achieve a complex interplay. Occluder manipulation coupled with TFs is shown in the *Turtle* scene (Figure 5.16). The initial shot exhibits an unlucky overlap of occluders (head, body and hind fins). An artist may want to edit the scattering to put more emphasis on the actual object and remove attention from the light shafts. The 2D morphological filtering closes the hole

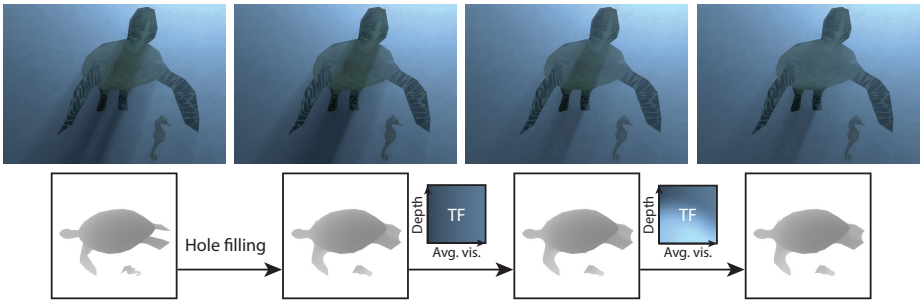


Figure 5.16: Combined use of occluder manipulation and TFs in the *Turtle* scene. From left to right: the original image; hole filling unifies the hind fins; a simple TF is applied to reduce the darkness of the shadow; another TF is used to remove the dark patch on the turtle's body.

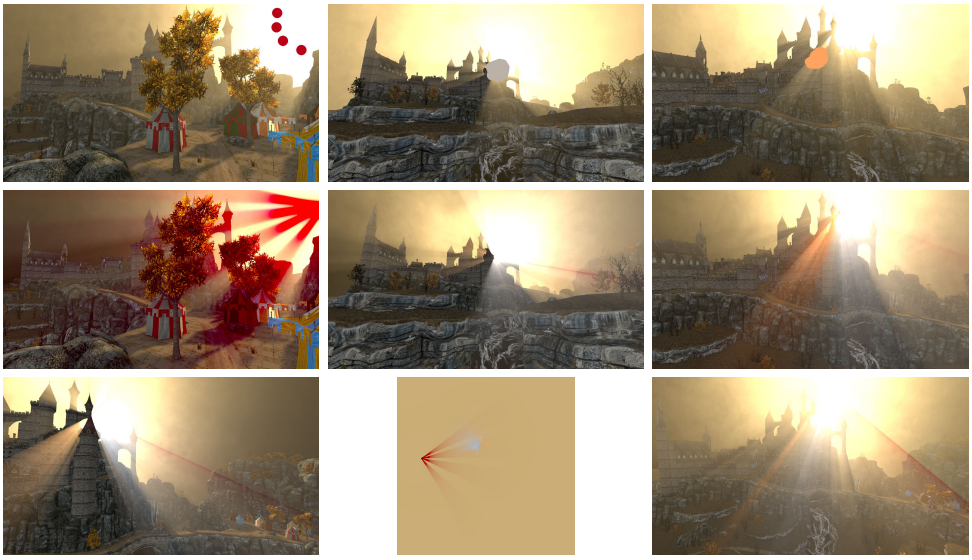


Figure 5.17: Coloring light shafts with light map optimization in the *Citadel* scene. Top: three views in which a user has drawn colors. Middle row: resulting images, which display light shafts that correspond to the specified colors. Bottom: two intermediate views are shown, which demonstrate a smooth blending and fadeout of the colors. The light map that was generated with this user input is shown in the center.

between the hind fins, which gives the light shafts a more simplified appearance. Then, a TF enables reducing the shadows, while keeping a plausible scattering appearance. Taking it further, we can use a TF to remove the potentially distracting dark stripe on the body caused by the shadow of the head, as the pixels in this area have similar average visibility and scene depth.

Figure 5.17 shows an example of light shaft coloring via our light map optimization. Here, colors are drawn by a user for three different views of the *Citadel* scene. Based on this input, we generate a light map as shown in the bottom center. Using this map for the scattering computation gives us the results shown in the middle row, which exhibit colored light shafts matching the user-defined indications. Note that our optimization scheme ensures that the light shafts are colored as specified by the user, despite a po-

Figure 5.18: Heterogeneity modification in the *Dinosaur* scene. From left to right: the original scattering produces regular light shafts; heterogeneity is approximated using 3D Perlin noise, which creates more irregular scattering effects and gives the medium a distinguishable volumetric structure; using our 3D painting tool, we can remove unwanted details that are due to heterogeneity, while retaining the irregularity of the light shafts.

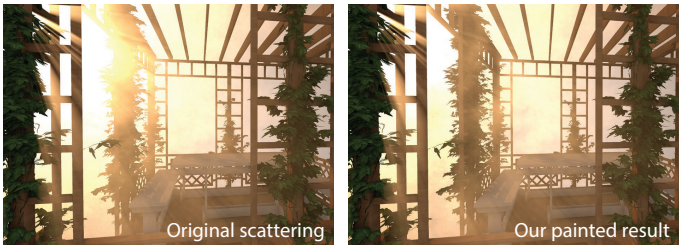


Figure 5.19: The scattering intensity can also be controlled using our 3D painting tool. Here, the unwanted saturation in the upper left corner is removed to reveal more of the vines' details.

tential overlap. Some red and white light also becomes visible in the right image, where the user did not specify any constraints. If this is an undesired side effect, it could be rectified by additional edits.

Transfer functions and light map optimization are complementary techniques for light shaft colorization; while the former is better suited for controlling the general mood of a scene, and can even be transferred to other scenes, the latter is of more use in specific scenarios and eases local changes.

Heterogeneity Modification Our heterogeneity approach is demonstrated for the *Dinosaur* scene in Figure 5.18. Here, we define the trade-off between heterogeneity and homogeneity directly via a painting approach in 3D. First, the background in the middle image was chosen and its homogeneity increased to avoid distracting details in the medium (in the upper part of the image), which would have taken emphasis from the dinosaur skeletons. Note that the irregularity of the light shafts in the lower part of the image is maintained.

Modulating the scattering intensity can also be beneficial in many scenarios. Such a change is illustrated for the *Arbor* scene (Figure 5.19), where overexposed areas are toned down, resulting in a more pleasant image.

To produce a result in the spirit of the right image in Figure 5.2, we can combine modifications of all three blend volumes (Figure 5.20). Here, we drastically increase the

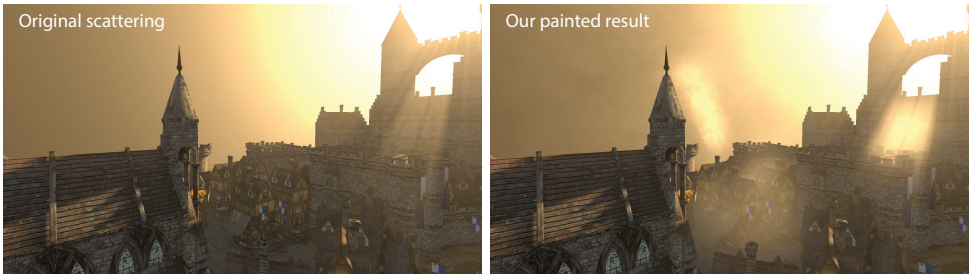


Figure 5.20: Combination of painting in all three blend volumes. Left: original image using a homogeneous medium. Right: the painted result.

scattering intensity between the houses to simulate fog, as well as for the light shafts caused by the opening of the citadel wall – the latter are also modified to be fully homogeneous to avoid distracting details. Furthermore, we use a higher noise frequency for parts of the background to simulate chimney smoke.

5.5.2. PERFORMANCE

Table 5.1: Performance (in ms) of our prototype for the *Tree* scene in Figure 5.1, rendered in full HD for different shadow map (SM) sizes. Measurements for hole creation include Perlin noise creation and SM modification. A TF for color stylization is indexed using the depth map of a G-buffer [ST90] as well as the view ray's average visibility. Using an acceleration method like the one from Klehm et al. [KSE14] is required as naive ray marching is an order of magnitude slower (67.8ms for full HD and 1024 marching steps).

SM Size	SM Creation	Holes	G-buffer	Visibility	TF
512^2	1.3	0.1	4.0	1.6	0.2
1024^2	1.6	0.3	3.9	2.5	0.2
2048^2	4.0	1.4	4.1	4.6	0.2

Table 5.2: Performance (in ms) for hole filling for the *Tree* scene of Figure 5.1, rendered in full HD for different SM and kernel sizes. As we work in image space, the kernel size needs to be adapted to the SM size as well as the content. The right-most column gives an indication of how many holes are filled for the given SM and kernel size.

SM Size	2D Closure		1D Epipolar Closure		Filled
	Kernel	Filtering	Kernel	Filtering	
512^2	11	1.1	15	0.6	All
512^2	21	3.7	30	1.1	All
512^2	41	13.3	60	1.9	All
1024^2	11	4.5	15	1.7	Most
1024^2	21	15.4	30	2.9	All
1024^2	41	56.7	60	5.2	All
2048^2	11	18.0	15	5.4	Many
2048^2	21	60.9	30	9.3	Most
2048^2	41	222	60	16.8	All

Occluder manipulation and transfer functions work in image space, with the former directly modifying the shadow map that is used for the scattering computation. Image space has several benefits over object space. The performance does not depend on geometric detail or scene complexity and the methodology is well suited for today's parallel hardware. Furthermore, operations such as hole filling are easy to perform, because the neighborhood of objects is automatically resolved. Tables 5.1 and 5.2 show performance measurements on an NVIDIA GeForce GTX 980 Titan in full HD for occluder manipulation and pipeline steps for computing and coloring light shafts.

As we have shown, computing single scattering in homogeneous media is possible at high frame rates using specialized and efficient techniques that circumvent naive ray marching. However, to approximate heterogeneity, we need ray marching to evaluate the spatially varying scattering intensity along the view ray. When directly applying a march-

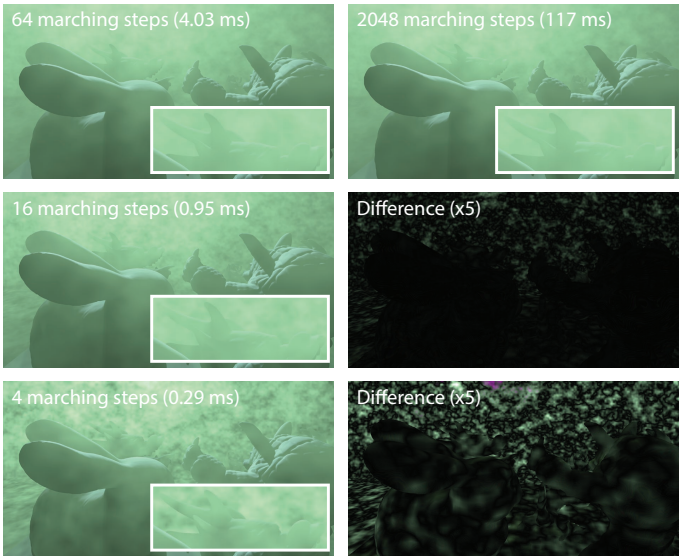


Figure 5.21: Effect of marching steps using our heterogeneity approximation. Timings obtained in full HD. Top: result with 64 steps and reference image with 2048 marching steps (no difference image because it is identical to the reference). Middle: result with 16 steps and the corresponding difference to the reference, multiplied by 5 for illustration. Bottom: result with 4 steps and the corresponding difference image.

ing process, consisting of several thousand steps per pixel, it becomes the bottleneck in our framework. As discussed in Section 5.4.3, we mitigate this problem by out-factoring the intensity, which allows us to cut down the number of marching steps. Figure 5.21 shows results for multiple numbers of marching steps.

While 2048 marching steps do not create improved results in a 256^3 volume, it indicates the high costs that would arise if not relying on any approximation. Even at 16 steps, our approximation results in a reasonable visual quality. Only at 4 steps, the results are no longer acceptable. In practice, we use 64 steps, which is 29 times faster compared to 2048 steps, and easily reaches real-time performance.

An additional acceleration is possible without a significant quality loss by using upsampling based on the depth map [YSL08]. The approach works well because heterogeneity produces smooth effects. Figure 5.22 shows 4x and 16x upsampling using 64 marching steps. For 4x upsampling the difference is negligible, but uses 16 times less pixels. Visible artifacts only occur for higher upsampling rates, such as 16x (reduction by a factor of 256). The speedup is 13 times using 4x upsampling, leading to real-time rates without discernible quality loss.

The light map optimization in Figure 5.17 computes in 2.04 seconds, quick enough to allow for interactive editing. Here, we painted a total of 41240 constraint pixels from 3 different views. The resolution of the light map was set to 512×512 with a corresponding shadow map of size 512×512 and a ray marching procedure with 768 steps. The optimization converges after 32 conjugate gradient iterations.

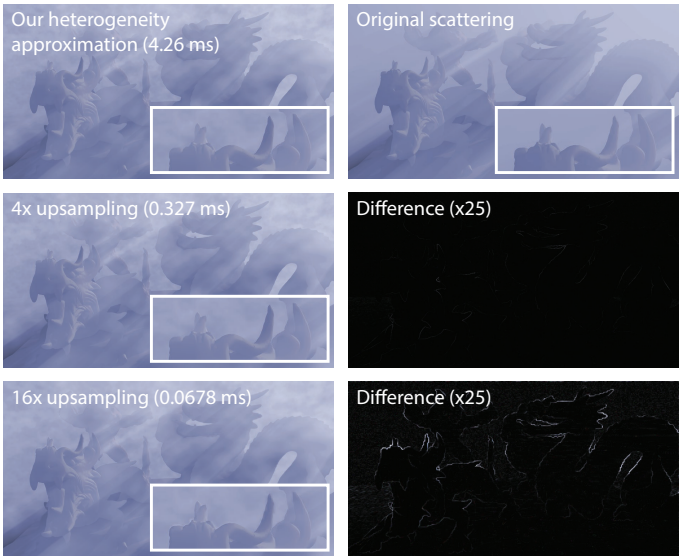


Figure 5.22: Effect of 2D upsampling using our heterogeneity approximation. Timings obtained in full HD. Top: reference image without upsampling and the original scattering. Middle: 4x upsampling result, and the corresponding difference to the reference, multiplied by 25 for illustration. Bottom: 16x upsampling result, and the corresponding difference image.

5.5.3. DISCUSSION

In our results, we use directional light only, as light shafts are generally caused by the sun, both in virtual scenes (see Figure 5.2) and real life, for which directional light is a good approximation. However, the methods presented can be extended to support non-directional light as well, as they pose no additional requirements on the light source besides the presence of a shadow map. Furthermore, while we only support single scattering, this suffices for optically thin media where light shafts occur. Additionally, we can approximate a heterogeneous medium, enabling further applications of our techniques. In summary, our methods cover all practical and important use cases of light shafts.

To assess the practical usefulness of our methods, we consulted two professional production artists working in the visual effects industry. They said light shafts are very common in production, and used as a stylistic tool or mood changer, for which artistic control is essential. However, only limited and laborious approaches exist to control them, such as placing large proxy objects, or entirely faking them in post-processing. For this reason, our tool was enthusiastically received, and considered to be easy to use with a good level of parameter control. Requests were even made to make it into a product.

5.6. CONCLUSION

We have presented several strategies to stylize volumetric single scattering, overcoming the difficulty that light shafts depend on the layout of an entire environment. Our approach is compatible with animated scenes and relies on very efficient solutions, which makes it ready to be used for real-time applications, and enables a quick exploration of

the various settings. The techniques are applied at a global scope – i.e., for the whole scene – but can also be used to make local changes to the scattering behavior.

Image-based occluder manipulations modify the complexity of the scattering appearance and are controlled by only a few parameters. Transfer functions allow us to interactively design a general mood and the result can even be transferred to other scenes. As an alternative, users can design a light map to modify the light emittance by relying on an optimization process which ensures that user-defined constraints are respected, which are defined using a painting metaphor. Furthermore, we employ an efficient algorithm to approximate heterogeneity and enable the control of scattering intensity, noise frequency, and the heterogeneity ratio. Finally, our solution supports key-framed animation to steer the stylization over time.

Our system makes a step towards designing scattering behavior, but leaves room for future work via additional object-focus strategies, stylization techniques for multiple scattering, or more advanced transfer function parameters.

6

CONCLUSION

All we have to decide is what to do with the time that is given us.

J. R. R. Tolkien

NOWADAYS, in the games and visual effects industries, tremendous amounts of resources are allocated solely for the purpose of creating and representing virtual environments. Indeed, virtual worlds receive a proportionally growing amount of attention. They serve as more than just a backdrop for stories or gameplay; they can paint a mood, elicit emotional responses, spark the imagination, and tell stories themselves. Their appearance determines how we perceive the application. For these reasons, virtual worlds will continue to play a crucial role in future computer graphics research. With the size and complexity of virtual environments continuing to grow faster than our hardware capabilities, many research problems arise. In this dissertation, we have barely scratched the surface of these challenges, which is why in the following, we discuss additional research directions in the context of virtual worlds. Finally, we discuss how the contributions presented in each chapter can be linked together for several applications.

An aspect that still requires a lot of manual labor is the *creation* of virtual worlds. This causes it to be a major bottleneck in many production pipelines. While there is a vast body of research on procedural content generation, its application is still limited to simple scenarios such as terrains, road networks and building exteriors. To enable the generation of complex virtual environments, we believe that more advanced techniques are required, likely based on deep learning rather than rule-based approaches, and relying on semantic input, possibly even in the form of narratives.

After content generation, automated or not, there is almost always a need for manual adjustments, which can be as laborious. For instance, artists working in the top-tier visual effects company Double Negative confirmed this was the case for effects such as light shafts. In this context of light behavior, we have touched upon the concept of

manipulating existing worlds in this dissertation in Chapter 5. For geometric modifications too, however, we need efficient manipulation tools, so that users can immediately see the results. One solution is to store the transformations instead of directly applying them. This is particularly beneficial for underlying representations where modifications would form critical bottlenecks. For our storage method from Chapter 2, for example, transformations could be stored in subtrees, and applied during raytracing, rather than recomputing the entire structure.

We face similar challenges regarding the employed content *storage* method. When working with complex virtual environments, the memory footprint quickly grows to an unfeasible size, as we have extensively discussed for voxel-based approaches in Chapter 2. One insight that can be applied beyond our voxel structures, is that while virtual worlds come in countless different shapes, we have observed that they generally share a similar property. Namely, as they grow more complex, they tend to exhibit more repetition. This may seem counterintuitive at a macroscopic level, but when you zoom in, we typically find a lot of repeating structures. A real-world analogy is a city in comparison to a single house. While the city is more diverse, the fine geometry of the house is shared by many other buildings. This means that even though the city is much larger, we do not necessarily require a proportional amount of data to represent it. Still, we realize that there is always a limit to how much the data can be compressed, and the resulting footprint may simply not allow in-core storage. To scale up for such scenes, we need to either use *out-of-core* solutions that stream in the relevant data, or even split up the environment into parts that are independently processed. We see opportunities for improvement in the data stream selection by using a *prediction* of the most likely parts that are needed next, and a *perceptual analysis*.

Perception-based selection of the data to stream mainly relates to the associated *level of detail* (LoD). Indeed, for large-scale scenes, making use of multiple LoDs and a proper selection thereof, are of crucial importance; otherwise, showing the whole scene from afar would not be possible. We believe that LoD selection can be significantly improved by analyzing the perceptual difference that two distinct LoDs offer in the final visualization; our simple test from Chapter 3, where we compute if a scene element projects to less than a pixel, could be made much less conservative, while still producing nearly indistinguishable results. To enhance LoD selection, we first need structures that contain these LoDs. With triangle meshes still being ubiquitous in real-time applications, we see displacement mapping as a crucial tool to achieve this. However, we believe a trend towards storage methods where the LoD is inherently encoded in the data structure is imminent. Our SVO-based representation in Chapter 2 is an example of such a structure; by selecting a level in the tree, the LoD is directly selected as well, as every level is a refinement of the one above. We show already an application in Chapter 3, where directly obtaining this *implicit* LoD is vital for the algorithm.

The technique from Chapter 3 produces *realistic* representations, but as research into realism slowly starts to saturate, we predict a growing interest from the community for *alternative* representations as well, where human perception will play a big role. This can relate to wanting users to perceive the virtual world in a certain way, as we dis-

cuss in Chapter 5, improving user understanding, as in Chapter 4, or even exploiting the weaknesses of the human visual system, as we partly do with our attribute quantization in Chapter 2. For perceptual graphics in general, we expect personalization to become an important research topic. While everyone's eyes are normally biologically very similar, our brains often interpret the things we see very differently. This opens up a wide research area, especially since there is no general solution for perceptual visualization, as the possible user goals and environments are too distinct.

The ideas presented throughout this dissertation cover a broad area of research. We have already hinted at an interlinking of these ideas, but we also envision some more concrete applications. While our global illumination from Chapter 3 at present may have too high an overhead to be used in a game engine running on today's hardware, we expect approximations like ours to become more feasible within the next decade. In our experiments, the double hierarchies had a relatively small memory footprint. For large navigable worlds or high-resolution hierarchies, however, compression techniques like the one we have presented in Chapter 2 become crucial, especially for games, where memory is scarce. Indeed, these two techniques can be readily combined into a single system, supporting efficient many-view rendering for the games of the future.

Like the notions of realistic, illustrative, and artistic representations, our work in Chapters 3, 4 and 5 is orthogonal, yet far from mutually exclusive. For the scenario sketched above, we could very well imagine the need for a more artistic touch. In addition to global illumination, physically-based single scattering will also start playing a larger role in upcoming games, which will complicate controlling the aesthetics. We could easily add our light shaft stylization to the aforementioned system. In this respect, it would be interesting to have control over the global illumination as well; we could derive information from the stylized single scattering to guide the many-view rendering. For instance, light shafts could be stylized to have a certain color and intensity (Chapter 5); then, where they intersect our compressed scene representation (Chapter 2), we generate VPLs that take their properties not only from the scene but also from the stylized light, and compute the global illumination (Chapter 3). Similarly, we could apply our techniques of occluder manipulation, transfer functions and heterogeneity modification (Chapter 5) to the glowing particles presented in Chapter 3.

The canonical view rendering from Chapter 4 is perhaps less likely to be combined with realistic global illumination or artistic control of light shafts, yet the applications we intend can benefit greatly from memory compression. After all, large-scale city models such as in Google Maps suffer from an extremely high memory footprint. While city scenes are typically best represented by surface-based approaches, it would be interesting to see how our voxel compression (Chapter 2) compares, especially given the fact that textures form the largest overhead. After all, the large amount of repetition present in these city models will greatly improve our compression, and as the structures become more realistic (and thus geometrically more complex), voxel-based approaches like ours may become more suitable.

With these, we have given some examples of how our specific contributions may be combined in achieving a better representation. In general, we believe that an interlink-

ing of ideas from scene storage as well as all three discussed visual representations is necessary to perform high-quality rendering while making efficient use of resources, given the high complexity of the virtual worlds of the future.

While we clearly see many new research directions, we believe the contributions discussed in this dissertation significantly aid computer graphics in handling the underlying and visual representations of increasingly complex scenes. With our solutions for storing and displaying environments, we make a step towards better *representing large virtual worlds*, enabling the generation of beautiful, realistic and informative images.

BIBLIOGRAPHY

[ABC*91] ADELSON S. J., BENTLEY J. B., CHONG I. S., HODGES L. F., WINOGRAD J.: Simultaneous Generation of Stereoscopic Views. *Computer Graphics Forum 10*, 1 (1991), 3–10.

[ASDW14] AMENT M., SADLO F., DACHSBACHER C., WEISKOPF D.: Low-Pass Filtered Volumetric Shadows. *IEEE Transactions on Visualization and Computer Graphics 20*, 12 (2014), 2437–2446.

[BAS02] BRABEC S., ANNEN T., SEIDEL H.-P.: Shadow Mapping for Hemispherical and Omnidirectional Light Sources. *Advances in Modelling, Animation and Rendering* (2002), 397–408.

[BCRK*10] BARAN I., CHEN J., RAGAN-KELLEY J., DURAND F., LEHTINEN J.: A Hierarchical Volumetric Shadow Algorithm for Single Scattering. *ACM Transactions on Graphics 29*, 6 (2010), 178:1–178:10.

[BEDT10] BEZERRA H., EISEMANN E., DECARLO D., THOLLOT J.: Diffusion Constraints for Vector Graphics. In *Proceedings of NPAR: Non-Photorealistic Animation and Rendering* (2010), pp. 35–42.

[BN76] BLINN J. F., NEWELL M. E.: Texture and Reflection in Computer Generated Images. *Communications of the ACM 19*, 10 (1976), 542–547.

[BRGIG*14] BALSA RODRÍGUEZ M., GOBBETTI E., IGLESIAS GUITIÁN J. A., MAKHINYA M., MARTON F., PAJAROLA R., SUTER S. K.: State-of-the-Art in Compressed GPU-Based Direct Volume Rendering. *Computer Graphics Forum 33*, 6 (2014), 77–100.

[BSA10] BILLETER M., SINTORN E., ASSARSSON U.: Real Time Volumetric Shadows Using Polygonal Light Volumes. In *Proceedings of HPG: High-Performance Graphics* (2010), pp. 39–45.

[BTBV99] BLANZ V., TARR M. J., BÜLTHOFF H. H., VETTER T.: What Object Attributes Determine Canonical Views? *Perception 28*, 5 (1999), 575–600.

[BTM06] BARLA P., THOLLOT J., MARKOSIAN L.: X-Toon: An Extended Toon Shader. In *Proceedings of NPAR: Non-Photorealistic Animation and Rendering* (2006), pp. 127–132.

[BWB08] BOULOS S., WALD I., BENTHIN C.: Adaptive Ray Packet Reordering. In *Proceedings of RT: Interactive Ray Tracing* (2008), pp. 131–138.

[CBDJ11] CHEN J., BARAN I., DURAND F., JAROSZ W.: Real-Time Volumetric Shadows using 1D Min-Max Mipmaps. In *Proceedings of I3D: Interactive 3D Graphics and Games* (2011), pp. 39–46.

[CDE*14] CIGOLLE Z. H., DONOW S., EVANGELAKOS D., MARA M., MCGUIRE M., MEYER Q.: A Survey of Efficient Representations for Independent Unit Vectors. *Journal of Computer Graphics Techniques 3*, 2 (2014), 1–30.

[CDF*06] COLE F., DECARLO D., FINKELSTEIN A., KIN K., MORLEY K., SANTELLA A.: Directing Gaze in 3D Models with Stylized Focus. In *Proceedings of EGSR: Eurographics Symposium on Rendering* (2006), pp. 377–387.

[CE94] CUTZU F., EDELMAN S.: Canonical Views in Object Representation and Recognition. *Vision Research 34*, 22 (1994), 3037–3056.

[CG12] CRASSIN C., GREEN S.: Chapter 22: Octree-Based Sparse Voxelization Using the GPU Hardware Rasterizer. In *OpenGL Insights*. AK Peters, 2012, pp. 303–320.

[Chr08] CHRISTENSEN P.: *Point-Based Approximate Color Bleeding*. Tech. rep., Pixar, 2008.

[CNLE09] CRASSIN C., NEYRET F., LEFEBVRE S., EISEMANN E.: GigaVoxels: Ray-Guided Streaming for Efficient and Detailed Voxel Rendering. In *Proceedings of I3D: Interactive 3D Graphics and Games* (2009), pp. 15–22.

[CNS*11] CRASSIN C., NEYRET F., SAINZ M., GREEN S., EISEMANN E.: Interactive Indirect Illumination using Voxel Cone Tracing. *Computer Graphics Forum 30*, 7 (2011), 1921–1930.

[CNSE10] CRASSIN C., NEYRET F., SAINZ M., EISEMANN E.: X.3: Efficient Rendering of Highly Detailed Volumetric Scenes with GigaVoxels. In *GPU Pro*. AK Peters, 2010, pp. 643–676.

[CPC84] COOK R. L., PORTER T., CARPENTER L.: Distributed Ray Tracing. *ACM Transactions on Graphics 18*, 3 (1984), 137–145.

[DCFR07] DECORO C., COLE F., FINKELSTEIN A., RUSINKIEWICZ S.: Stylized Shadows. In *Proceedings of NPAR: Non-Photorealistic Animation and Rendering* (2007), pp. 77–83.

[DCG10] DUTAGACI H., CHEUNG C. P., GODIL A.: A Benchmark for Best View Selection of 3D Objects. In *Proceedings of 3DOR: 3D Object Retrieval* (2010), pp. 45–50.

[Den97] DENIS M.: The Description of Routes: A Cognitive Approach to the Production of Spatial Discourse. *Cahiers de Psychologie Cognitive 16*, 4 (1997), 409–458.

[DFKP05] DE FLORIANI L., KOBBELT L., PUPPO E.: A Survey on Data Structures for Level-of-Detail Models. In *Advances in Multiresolution for Geometric Modelling*. Springer, 2005, pp. 49–74.

[DKB*16] DADO B., KOL T. R., BAUSZAT P., THIERY J.-M., EISEMANN E.: Geometry and Attribute Compression for Voxel Scenes. *Computer Graphics Forum 35*, 2 (2016), 397–407.

[DKH*14] DACHSBACHER C., KŘIVÁNEK J., HAŠAN M., ARBREE A., WALTER B., NOVÁK J.: Scalable Realistic Rendering with Many-Light Methods. *Computer Graphics Forum 33*, 1 (2014), 88–104.

[DSKA17] DOLONIUS D., SINTORN E., KÄMPE V., ASSARSSON U.: Compressing Color Data for Voxelized Surface Geometry. *IEEE Transactions on Visualization and Computer Graphics* (2017).

[EB92] EDELMAN S., BÜLTHOFF H. H.: Orientation Dependence in the Recognition of Familiar and Novel Views of Three-Dimensional Objects. *Vision Research 32*, 12 (1992), 2385–2400.

[ED04] EISEMANN E., DURAND F.: Flash Photography Enhancement via Intrinsic Relighting. *ACM Transactions on Graphics 23*, 3 (2004), 673–678.

[ED06] EISEMANN E., DÉCORET X.: Fast Scene Voxelization and Applications. In *Proceedings of I3D: Interactive 3D Graphics and Games* (2006), pp. 71–78.

[ERDS14] ELEK O., RITSCHEL T., DACHSBACHER C., SEIDEL H.-P.: Interactive Light Scattering with Principal-Ordinate Propagation. In *Proceedings of GI: Graphics Interface* (2014), pp. 87–94.

[Eve01] EVERITT C.: *Interactive Order-Independent Transparency*. Tech. rep., NVIDIA Corporation, 2001.

[FG14] FUHRMANN S., GOESELE M.: Floating Scale Surface Reconstruction. *ACM Transactions on Graphics 33*, 4 (2014), 46.

[GASP08] GRABLER F., AGRAWALA M., SUMNER R. W., PAULY M.: Automatic Generation of Tourist Maps. *ACM Transactions on Graphics 27*, 3 (2008), 100:1–100:12.

[GMIG08] GOBBETTI E., MARTON F., IGLESIAS GUITIÁN J. A.: A Single-Pass GPU Ray Casting Framework for Interactive Out-of-Core Rendering of Massive Volumetric Datasets. *The Visual Computer 24*, 7 (2008), 797–806.

[GMY11] GUO H., MAO N., YUAN X.: WYSIWYG (What You See is What You Get) Volume Visualization. *IEEE Transactions on Visualization and Computer Graphics 17*, 12 (2011), 2106–2114.

[GS10] GEORGIEV I., SLUSALLEK P.: Simple and Robust Iterative Importance Sampling of Virtual Point Lights. In *Proceedings of Eurographics Short Papers* (2010), pp. 57–60.

[GSLM*08] GUTIERREZ D., SERON F. J., LOPEZ-MORENO J., SANCHEZ M. P., FANDOS J., REINHARD E.: Depicting Procedural Caustics in Single Images. *ACM Transactions on Graphics 27*, 5 (2008), 120:1–120:9.

[GWGS02] GUTHE S., WAND M., GONSER J., STRASSER W.: Interactive Rendering of Large Volume Data Sets. In *Proceedings of VIS* (2002), pp. 53–60.

[HA90] HAEBERLI P., AKELEY K.: The Accumulation Buffer: Hardware Support for High-Quality Rendering. *ACM Transactions on Graphics 24*, 4 (1990), 309–318.

[Hal98] HALLE M.: Multiple Viewpoint Rendering. In *Proceedings of SIGGRAPH* (1998), pp. 243–254.

[HAM06] HASSELGREN J., AKENINE-MÖLLER T.: An Efficient Multi-View Rasterization Architecture. In *Proceedings of EGSR: Eurographics Symposium on Rendering* (2006), pp. 61–72.

[HMY12] HARADA T., MCKEE J., YANG J. C.: Forward+: Bringing Deferred Lighting to the Next Level. In *Proceedings of Eurographics Short Papers* (2012).

[HPB07] HAŠAN M., PELLACINI F., BALA K.: Matrix Row-Column Sampling for the Many-Light Problem. *ACM Transactions on Graphics 26*, 3 (2007), 26.

[HPL91] HARRIES M. H., PERRETT D. I., LAVENDER A.: Preferential Inspection of Views of 3-D Model Heads. *Perception 20*, 5 (1991), 669–680.

[HR13] HAŠAN M., RAMAMOORTHI R.: Interactive Albedo Editing in Path-Traced Volumetric Materials. *ACM Transactions on Graphics 32*, 2 (2013), 11:1–11:11.

[HREB11] HOLLÄNDER M., RITSCHEL T., EISEMANN E., BOUBEKEUR T.: ManyLoDs: Parallel Many-View Level-of-Detail Selection for Real-Time Global Illumination. *Computer Graphics Forum 30*, 4 (2011), 1233–1240.

[HS98] HEIDRICH W., SEIDEL H.-P.: View-Independent Environment Maps. In *Proceedings of GH: Graphics Hardware* (1998), p. 39ff.

[HTG03] HEIDELBERGER B., TESCHNER M., GROSS M. H.: Real-Time Volumetric Intersections of Deforming Objects. In *Proceedings of VMV: Vision, Modeling & Visualization* (2003), pp. 461–468.

[HVAPB08] HAŠAN M., VELAZQUEZ-ARMENDARIZ E., PELLACINI F., BALA K.: Tensor Clustering for Rendering Many-Light Animations. *Computer Graphics Forum 27*, 4 (2008), 1105–1114.

[JD08] JOBST M., DÖLLNER J.: Better Perception of 3D-Spatial Relations by Viewport Variations. *Visual Information Systems* (2008), 7–18.

[JMG16] JASPE VILLANUEVA A., MARTON F., GOBBETTI E.: SSVDAGs: Symmetry-Aware Sparse Voxel DAGs. In *Proceedings of I3D: Interactive 3D Graphics and Games* (2016), pp. 7–14.

[JT80] JACKINS C. L., TANIMOTO S. L.: Oct-Trees and Their Use in Representing Three-Dimensional Objects. *Computers Graphics and Image Processing 14*, 3 (1980), 249–270.

[JWSP05] JESCHKE S., WIMMER M., SCHUMANN H., PURGATHOFER W.: Automatic Impostor Placement for Guaranteed Frame Rates and Low Memory Requirements. In *Proceedings of I3D: Interactive 3D Graphics and Games* (2005), pp. 103–110.

[KBLE18] KOL T. R., BAUSZAT P., LEE S., EISEMANN E.: MegaViews: Scalable Many-View Rendering with Concurrent Scene-View Hierarchy Traversal. *Computer Graphics Forum* (2018).

[Kel97] KELLER A.: Instant Radiosity. In *Proceedings of SIGGRAPH* (1997), pp. 49–56.

[KISE13] KLEHM O., IHRKE I., SEIDEL H.-P., EISEMANN E.: Volume Stylizer: Tomography-Based Volume Painting. In *Proceedings of I3D: Interactive 3D Graphics and Games* (2013), pp. 161–168.

[KISE14] KLEHM O., IHRKE I., SEIDEL H.-P., EISEMANN E.: Property and Lighting Manipulations for Static Volume Stylization Using a Painting Metaphor. *IEEE Transactions on Visualization and Computer Graphics 20*, 7 (2014), 983–995.

[KKSE15] KLEHM O., KOL T. R., SEIDEL H.-P., EISEMANN E.: Stylized Scattering via Transfer Functions and Occluder Manipulation. In *Proceedings of GI: Graphics Interface* (2015), pp. 115–121.

[KKSE17] KOL T. R., KLEHM O., SEIDEL H.-P., EISEMANN E.: Expressive Single Scattering for Light Shaft Stylization. *IEEE Transactions on Visualization and Computer Graphics 23*, 7 (2017), 1753–1766.

[Kle10] KLEIN G. A.: *Industrial Color Physics*. Springer, 2010.

[KLE14] KOL T. R., LIAO J., EISEMANN E.: Real-Time Canonical-Angle Views in 3D Virtual Cities. In *Proceedings of VMV: Vision, Modeling & Visualization* (2014), pp. 55–62.

[Kol12] KOL T. R.: *Analytical Sky Simulation*. Tech. rep., Utrecht University, 2012.

[Kol13] KOL T. R.: *Real-Time Cloud Rendering on the GPU*. Master's thesis, Utrecht University, 2013.

[KPD10] KERR W. B., PELLACINI F., DENNING J. D.: BendyLights: Artistic Control of Direct Illumination by Curving Light Rays. *Computer Graphics Forum 29*, 4 (2010), 1451–1459.

[KSA13] KÄMPE V., SINTORN E., ASSARSSON U.: High Resolution Sparse Voxel DAGs. *ACM Transactions on Graphics 32*, 4 (2013), 101.

[KSA15] KÄMPE V., SINTORN E., ASSARSSON U.: Fast, Memory-Efficient Construction of Voxelized Shadows. In *Proceedings of I3D: Interactive 3D Graphics and Games* (2015), pp. 25–30.

[KSE14] KLEHM O., SEIDEL H.-P., EISEMANN E.: Prefiltered Single Scattering. In *Proceedings of I3D: Interactive 3D Graphics and Games* (2014), pp. 71–78.

[LES09] LEE S., EISEMANN E., SEIDEL H.-P.: Depth-of-Field Rendering with Multiview Synthesis. *ACM Transactions on Graphics 28*, 5 (2009), 134:1–134:6.

[LES10] LEE S., EISEMANN E., SEIDEL H.-P.: Real-Time Lens Blur Effects and Focus Control. *ACM Transactions on Graphics 29*, 4 (2010), 65:1–65:7.

[LH06] LEFEBVRE S., HOPPE H.: Perfect Spatial Hashing. *ACM Transactions on Graphics 25*, 3 (2006), 579–588.

[LH07] LEFEBVRE S., HOPPE H.: Compressed Random-Access Trees for Spatially Coherent Data. In *Proceedings of EGSR: Eurographics Symposium on Rendering* (2007), pp. 339–349.

[LH13] LEI K., HUGHES J. F.: Approximate Depth of Field Effects using Few Samples per Pixel. In *Proceedings of I3D: Interactive 3D Graphics and Games* (2013), pp. 119–128.

[LK10] LAINE S., KARRAS T.: *Efficient Sparse Voxel Octrees: Analysis, Extensions, and Implementation.* Tech. rep., NVIDIA Corporation, 2010.

[LK11] LAINE S., KARRAS T.: Efficient Sparse Voxel Octrees. *IEEE Transactions on Visualization and Computer Graphics 17*, 8 (2011), 1048–1059.

[LKT*17] LESKENS J. G., KEHL C., TUTENEL T., KOL T. R., DE HAAN G., STELLING G., EISEMANN E.: An Interactive Simulation and Visualization Tool for Flood Analysis Usable for Practitioners. *Mitigation and Adaptation Strategies for Global Change 22*, 2 (2017), 307–324.

[LTDJ08] LORENZ H., TRAPP M., DÖLLNER J., JOBST M.: Interactive Multi-Perspective Views of Virtual 3D Landscape and City Models. *The European Information Society* (2008), 301–321.

[LVJ05] LEE C. H., VARSHNEY A., JACOBS D. W.: Mesh Saliency. *ACM Transactions on Graphics 24*, 3 (2005), 659–666.

[LWC*03] LUEBKE D., WATSON B., COHEN J. D., REDDY M., VARSHNEY A.: *Level of Detail for 3D Graphics.* Elsevier Science Inc., 2003.

[MBJ*15] MATTAUSCH O., BITTNER J., JASPE A., GOBBETTI E., WIMMER M., PAJAROLA R.: CHC+ RT: Coherent Hierarchical Culling for Ray Tracing. *Computer Graphics Forum 34*, 2 (2015), 537–548.

[MBWW07] MATTAUSCH O., BITTNER J., WONKA P., WIMMER M.: Optimized Subdivisions for Preprocessed Visibility. In *Proceedings of GI: Graphics Interface* (2007), pp. 335–342.

[MDWK08] MÖSER S., DEGENER P., WAHL R., KLEIN R.: Context Aware Terrain Visualization for Wayfinding and Navigation. *Computer Graphics Forum 27*, 7 (2008), 1853–1860.

[Mea82] MEAGHER D.: Geometric Modeling Using Octree Encoding. *Computers Graphics and Image Processing 19*, 2 (1982), 129–147.

[MIW13] MATTAUSCH O., IGARASHI T., WIMMER M.: Freeform Shadow Boundary Editing. *Computer Graphics Forum 32* (2013), 175–184.

[MKC07] MARROQUIM R., KRAUS M., CAVALCANTI P. R.: Efficient Point-Based Rendering Using Image Reconstruction. In *Proceedings of SPBG: Symposium on Point Based Graphics* (2007), pp. 101–108.

[MML12] MCGUIRE M., MARA M., LUEBKE D.: Scalable Ambient Obscurance. In *Proceedings of HPG: High-Performance Graphics* (2012), pp. 97–103.

[MS09] MORTARA M., SPAGNUOLO M.: Semantics-Driven Best View of 3D Shapes. *Computers & Graphics 33*, 3 (2009), 280–290.

[MSS*10] MEYER Q., SÜSSMUTH J., SUSSNER G., STAMMINGER M., GREINER G.: On Floating-Point Normal Vectors. *Computer Graphics Forum 29*, 4 (2010), 1405–1409.

[NJS*11] NOWROUZEZAHRAI D., JOHNSON J., SELLE A., LACEWELL D., KASCHALK M., JAROSZ W.: A Programmable System for Artistic Volumetric Lighting. *ACM Transactions on Graphics 30*, 4 (2011), 29:1–29:8.

[NLP*12] NYSTAD J., LASSEN A., POMIANOWSKI A., ELLIS S., OLSON T.: Adaptive Scalable Texture Compression. In *Proceedings of HPG: High-Performance Graphics* (2012), pp. 105–114.

[OA11] OLSSON O., ASSARSSON U.: Tiled Shading. *Journal of Graphics, GPU, and Game Tools 15*, 4 (2011), 235–251.

[OBA12] OLSSON O., BILLETER M., ASSARSSON U.: Clustered Deferred and Forward Shading. In *Proceedings of HPG: High-Performance Graphics* (2012), pp. 87–96.

[OBB*13] ORZAN A., BOUSSEAU A., BARLA P., WINNEMÖLLER H., THOLLOT J., SALESIN D.: Diffusion Curves: A Vector Representation for Smooth-Shaded Images. *Communications of the ACM 56*, 7 (2013), 101–108.

[OBS*15] OLSSON O., BILLETER M., SINTORN E., KÄMPE V., ASSARSSON U.: More Efficient Virtual Shadow Maps for Many Lights. *IEEE Transactions on Visualization and Computer Graphics 21*, 6 (2015), 701–713.

[OKP*08] OBERT J., KŘIVÁNEK J., PELLACINI F., SYKORA D., PATTANAIK S.: iCheat: A Representation for Artistic Control of Indirect Cinematic Lighting. *Computer Graphics Forum 27*, 4 (2008), 1217–1223.

[OP11] OU J., PELLACINI F.: LightSlice: Matrix Slice Sampling for the Many-Lights Problem. *ACM Transactions on Graphics 30*, 6 (2011), 179.

[OSK*14] OLSSON O., SINTORN E., KÄMPE V., BILLETER M., ASSARSSON U.: Efficient Virtual Shadow Maps for Many Lights. In *Proceedings of I3D: Interactive 3D Graphics and Games* (2014), pp. 87–96.

[Per85] PERLIN K.: An Image Synthesizer. *ACM Transactions on Graphics 19*, 3 (1985), 287–296.

[Pet00] PETERS G.: Theories of Three-Dimensional Object Perception: A Survey. *Recent Research Developments in Pattern Recognition 1* (2000), 179–197.

[PH88] PERRETT D. I., HARRIES M. H.: Characteristic Views and the Visual Inspection of Simple Faceted and Smooth Objects: Tetrahedra and Potatoes. *Perception 17*, 6 (1988), 703–720.

[PHL92] PERRETT D. I., HARRIES M. H., LOOKER S.: Use of Preferential Inspection to Define the Viewing Sphere and Characteristic Views of An Arbitrary Machined Tool Part. *Perception 21*, 4 (1992), 497–515.

[PKS*03] PAGE D. L., KOSCHAN A. F., SUKUMAR S. R., ROUI-ABIDI B., ABIDI M. A.: Shape Analysis Algorithm Based on Information Theory. In *Proceedings of ICIP: International Conference on Image Processing* (2003), pp. 229–232.

[PP09] PEGORARO V., PARKER S. G.: An Analytical Solution to Single Scattering in Homogeneous Participating Media. *Computer Graphics Forum 28*, 2 (2009), 329–335.

[PPB*05] POLONSKY O., PATANÉ G., BIASOTTI S., GOTSMAN C., SPAGNUOLO M.: What's In An Image? *The Visual Computer 21*, 8-10 (2005), 840–847.

[PRC81] PALMER S., ROSCH E., CHASE P.: Canonical Perspective and the Perception of Objects. In *Attention and Performance IX*. L. Erlbaum Associates, 1981, pp. 135–151.

[PSA*04] PETSCHNIGG G., SZELISKI R., AGRAWALA M., COHEN M., HOPPE H., TOYAMA K.: Digital Photography with Flash and No-Flash Image Pairs. *ACM Transactions on Graphics 23*, 3 (2004), 664–672.

[PŠNB13] PATEL D., ŠOLTÉSZOVÁ V., NORDBOTTEN J. M., BRUCKNER S.: Instant Convolution Shadows for Volumetric Detail Mapping. *ACM Transactions on Graphics 32*, 5 (2013), 154:1–154:18.

[PVL*05] PELLACINI F., VIDIMČE K., LEFOHN A., MOHR A., LEONE M., WARREN J.: Lpics: A Hybrid Hardware-Accelerated Relighting Engine for Computer Cinematography. *ACM Transactions on Graphics 24*, 3 (2005), 464–470.

[QWC*09] QU H., WANG H., CUI W., WU Y., CHAN M.-Y.: Focus+Context Route Zooming and Information Overlay in 3D Urban Environments. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (2009), 1547–1554.

[RAH07] ROGER D., ASSARSSON U., HOLZSCHUCH N.: Whitted Ray-Tracing for Dynamic Scenes using a Ray-Space Hierarchy on the GPU. In *Proceedings of EGSR: Eurographics Symposium on Rendering* (2007), pp. 99–110.

[REG*09] RITSCHEL T., ENGELHARDT T., GROSCH T., SEIDEL H.-P., KAUTZ J., DACHSBACHER C.: Micro-Rendering for Scalable, Parallel Final Gathering. *ACM Transactions on Graphics 28*, 5 (2009), 132.

[REH*11] RITSCHEL T., EISEMANN E., HA I., KIM J. D. K., SEIDEL H.-P.: Making Imperfect Shadow Maps View-Adaptive: High-Quality Global Illumination in Large Dynamic Scenes. *Computer Graphics Forum 30*, 8 (2011), 2258–2269.

[RGK*08] RITSCHEL T., GROSCH T., KIM M. H., SEIDEL H.-P., DACHSBACHER C., KAUTZ J.: Imperfect Shadow Maps for Efficient Computation of Indirect Illumination. *ACM Transactions on Graphics 27*, 5 (2008), 129.

[RKKS*07] RAGAN-KELLEY J., KILPATRICK C., SMITH B. W., EPPS D., GREEN P., HERY C., DURAND F.: The Lightspeed Automatic Interactive Lighting Preview System. *ACM Transactions on Graphics 26*, 3 (2007), 25:1–25:11.

[SAM05] STRÖM J., AKENINE-MÖLLER T.: iPACKMAN: High-Quality, Low-Complexity Texture Compression for Mobile Phones. In *Proceedings of GH: Graphics Hardware* (2005), pp. 63–70.

[SDS*93] SCHOENEMAN C., DORSEY J., SMITS B., ARVO J., GREENBERG D.: Painting with Light. In *Proceedings of SIGGRAPH* (1993), pp. 143–146.

[She94] SHEWCHUK J. R.: *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain.* Tech. rep., Carnegie Mellon University, 1994.

[SK06] SCHNABEL R., KLEIN R.: Octree-Based Point-Cloud Compression. In *Proceedings of SPBG: Symposium on Point Based Graphics* (2006), pp. 111–120.

[SKALP05] SZIRMAY-KALOS L., ASZÓDI B., LAZÁNYI I., PREMECZ M.: Approximate Ray-Tracing on the GPU with Distance Impostors. *Computer Graphics Forum 24*, 3 (2005), 695–704.

[SKE06] STRENGERT M., KRAUS M., ERTL T.: Pyramid Methods in GPU-based Image Processing. In *Proceedings of VMV: Vision, Modeling & Visualization* (2006), pp. 169–176.

[SKOA14] SINTORN E., KÄMPE V., OLSSON O., ASSARSSON U.: Compact Precomputed Voxelized Shadows. *ACM Transactions on Graphics 33*, 4 (2014), 150.

[SLF*11] SECORD A., LU J., FINKELSTEIN A., SINGH M., NEALEN A.: Perceptual Models of Viewpoint Preference. *ACM Transactions on Graphics 30*, 5 (2011), 109:1–109:12.

[SNM*13] SCHMIDT T.-W., NOVÁK J., MENG J., KAPLANYAN A. S., REINER T., NOWROUZEZAHRAI D., DACHSBACHER C.: Path-Space Manipulation of Physically-Based Light Transport. *ACM Transactions on Graphics 32*, 4 (2013), 129.

[SPN*16] SCHMIDT T.-W., PELLACINI F., NOWROUZEZAHRAI D., JAROSZ W., DACHSBACHER C.: State of the Art in Artistic Editing of Appearance, Lighting, and Material. *Computer Graphics Forum 35*, 1 (2016), 216–233.

[SS10] SCHWARZ M., SEIDEL H.-P.: Fast Parallel Surface and Solid Voxelization on GPUs. *ACM Transactions on Graphics 29*, 6 (2010), 179.

[SSBG10] SCHMID J., SUMNER R. W., BOWLES H., GROSS M.: Programmable Motion Effects. *ACM Transactions on Graphics 29*, 4 (2010), 57:1–57:9.

[ST90] SAITO T., TAKAHASHI T.: Comprehensible Rendering of 3-D Shapes. *ACM Transactions on Graphics 24*, 4 (1990), 197–206.

[STKD12] SEMMO A., TRAPP M., KYPRIANIDIS J. E., DÖLLNER J.: Interactive Visualization of Generalized Virtual 3D City Models using Level-of-Abstraction Transitions. *Computer Graphics Forum 31*, 3 (2012), 885–894.

[STPP09] SONG Y., TONG X., PELLACINI F., PEERS P.: SubEdit: A Representation for Editing Measured Heterogeneous Subsurface Scattering. *ACM Transactions on Graphics 28*, 3 (2009), 31.

[TH16] TOKUYOSHI Y., HARADA T.: Stochastic Light Culling. *Journal of Computer Graphics Techniques 5*, 1 (2016).

[VB95] VERFAILLIE K., BOUTSEN L.: A Corpus of 714 Full-Color Images of Depth-Rotated Objects. *Attention, Perception & Psychophysics 57*, 7 (1995), 925–961.

[VFSH01] VÁZQUEZ P.-P., FEIXAS M., SBERT M., HEIDRICH W.: Viewpoint Selection using Viewpoint Entropy. In *Proceedings of VMV: Vision, Modeling & Visualization* (2001), pp. 273–280.

[WABG06] WALTER B., ARBREE A., BALA K., GREENBERG D. P.: Multidimensional Lightcuts. *ACM Transactions on Graphics 25*, 3 (2006), 1081–1088.

[WBSS04] WANG Z., BOVIK A. C., SHEIKH H. R., SIMONCELLI E. P.: Image Quality Assessment: From Error Visibility to Structural Similarity. *IEEE Transactions on Image Processing 13*, 4 (2004), 600–612.

[WFA*05] WALTER B., FERNANDEZ S., ARBREE A., BALA K., DONIKIAN M., GREENBERG D. P.: Lightcuts: A Scalable Approach to Illumination. *ACM Transactions on Graphics 24*, 3 (2005), 1098–1107.

[WHB*13] WANG B., HUANG J., BUCHHOLZ B., MENG X., BOUBEKEUR T.: Factorized Point Based Global Illumination. *Computer Graphics Forum 32*, 4 (2013), 117–123.

[Wil15] WILLIAMS B. R.: *Moxel DAGs: Connecting Material Information to High Resolution Sparse Voxel DAGs*. Master's thesis, California Polytechnic State University, 2015.

[Wym11] WYMAN C.: Voxelized Shadow Volumes. In *Proceedings of HPG: High-Performance Graphics* (2011), pp. 33–40.

[Xia97] XIANG Z.: Color Image Quantization by Minimizing the Maximum Intercluster Distance. *ACM Transactions on Graphics 16*, 3 (1997), 260–276.

[YSL08] YANG L., SANDER P. V., LAWRENCE J.: Geometry-Aware Framebuffer Level of Detail. *Computer Graphics Forum 27*, 4 (2008), 1183–1188.

[YSY*06] YAMAUCHI H., SALEEM W., YOSHIZAWA S., KARNI Z., BELYAEV A., SEIDEL H.-P.: Towards Stable and Salient Multi-View Representation of 3D Shapes. In *Proceedings of SMI: Shape Modeling International* (2006), pp. 40:1–40:6.

[ZHG*07] ZHOU K., HOU Q., GONG M., SNYDER J., GUO B., SHUM H.-Y.: Fogshop: Real-Time Design and Rendering of Inhomogeneous, Single-Scattering Media. In *Proceedings of PG: Pacific Graphics* (2007), pp. 116–125.

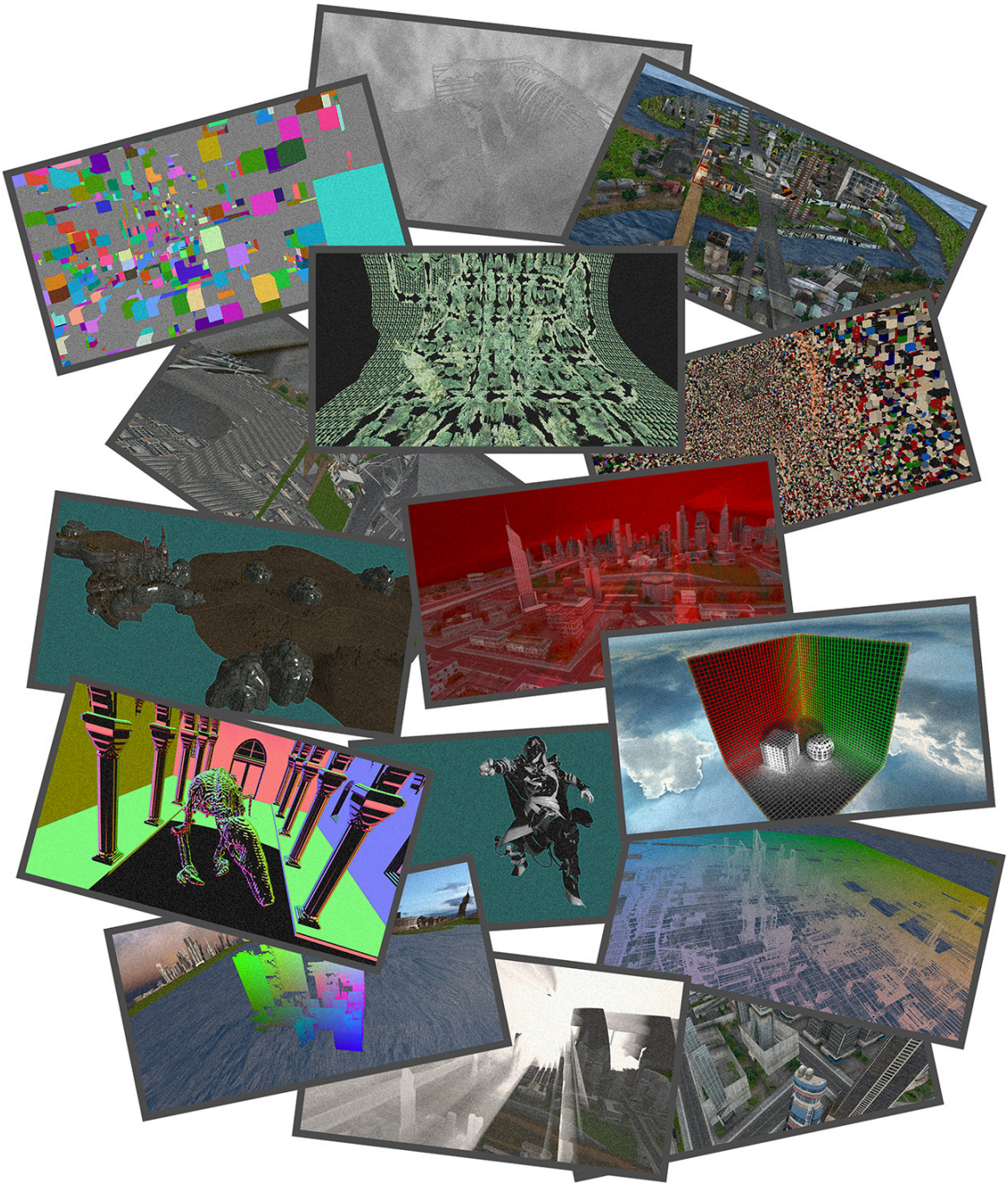
EPILOGUE

With this, we have come to the end of the dissertation. And as this work comes to an end, so does my time as a PhD candidate in Delft. The past four years have been a journey marked by the usual highs and lows that come with such an endeavor. Still, I look back on mostly positive and valuable experiences. I set out on this path to gain more knowledge, and make useful contributions to the field of computer graphics. I believe I have managed to do both, but, as in research, there is always room for improvement.

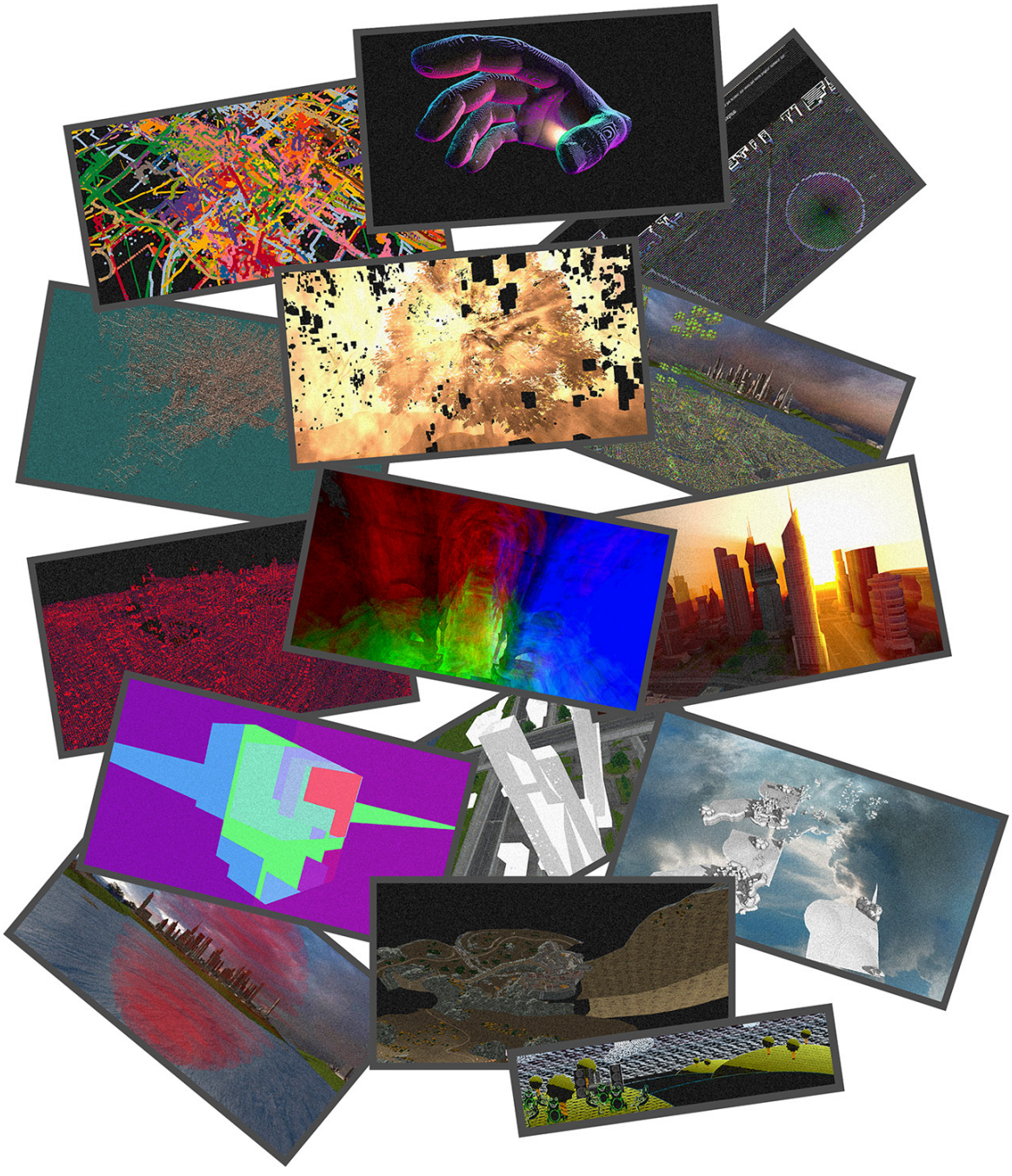
Looking at this dissertation, I feel a sense of pride, but also of humility when putting my work in perspective. There are so many great graphics researchers out there that it is hard to compete. However, I believe science is not a race, but a cooperative effort. Despite this belief I have decided to try my luck in the industry, since I long to see my research being applied in real products. Nevertheless, I intend to keep publishing my work when possible, and do not dismiss the possibility of working in academia again.

For my next challenge, I will soon be moving to Japan to work as a computer graphics researcher at the animation studio OLM Digital. Funny as it may sound, it looks like I am going to be a Pokémon researcher!

These are features...



...not Bugs



ACKNOWLEDGEMENTS

These final pages are dedicated to all the amazing people I met during my time as a PhD student. You all were vital to this thesis, be it for scientific contributions, support, or simply for providing the pleasant company and atmosphere that made the past four years such a wonderful period.

First and foremost, I would like to mention my supervisor and promotor. Elmar, thank you so much for not only giving me this opportunity, but also for your priceless advice and support throughout my projects. While I may have sometimes cursed your high standards and talent for finding weak spots in both my writing and methodology, you were always right (although I still disagree with some hyphenations!), and made the end product twice as good. In general, I think you know exactly how to bring out the best in people, and I believe with this asset our group will continue to be successful for many years to come. However, perhaps more important than your scientific qualifications, I have come to see you as a friend. I will miss your rants and ravings about the FC Cologne, students, NHL, students, bureaucracy, students, *Interstellar* and students. The bad math jokes at our Harvest4D meetings I will miss a bit less, but nonetheless, I am sad to leave.

Of course, a special thanks goes out to all the other supervisors and advisors I have had over the past years. Not long after I started, Jean-Marc joined the group (oh-la-la) and started bugging me to no end. But, with all the best intentions, and our many discussions only made my research better. I was sad to see you leave, but happy to get your couch. I always think about you when I sit on it. That came out wrong. In any case, with Jean-Marc gone, I was left to the devices of doctor Pablo *Probability Density Function* Bauszat. You filled in the gap brilliantly, so thank you for all the help and advice. And never give up on the Minute Maid. For the final sprint, Sungkil was kind enough to jump in. I learned a lot from our discussions, and the tools you told me about are very useful! Thanks for sticking with me those long deadline nights, I don't know if I would have ever finished the last project without you!

Furthermore, I would like to express my gratitude to my committee members, Peter van Oosterom, Michael Wimmer, Enrico Gobbetti, and Markus Billeter, and reserve member Marcel Reinders. Thank you so much for being part of the committee.

Then for my partners in crime. Jingtang, we both started our first project together four years ago. (I won't mention the hotel room in Darmstadt. Okay, I just did.) Look how far we have come since then! I'm proud of us. You've been a great friend through the past years, and I am going to miss you. I fiercely hope we will meet again in the future. Oliver, our collaborations were very successful, and I thank you for your kindness and bringing my C++ knowledge further up to speed. Bas, you were supposedly my student, but it felt more like teamwork on an equal level. I think we were a great team, and I was sad you

did not want to pursue a PhD. Nonetheless, thank you for the great work, and the good times we had in Lisbon! To my other student Yueqian: it was a pleasure supervising you, and thank you very much for the Chinese pastry!

To all the others in the CGV group, I could write a book about our adventures, but I will try to keep it short here, in semi-chronological order. Francois, Sergio, Rafa, Anna (thanks for the BBQs!), Christian, Matthias (thanks for the MxEngine!), Ben, Renata, Bart and Stefanie, thank you all for welcoming me into the group when I just arrived. Thank you Noeska (lunch!) for all the advice on getting started, and of course the fun we had. You are one of the kindest people I know! Thanks to Ricardo for showing me the wonders of the pub quiz. Ruud, thanks for always being there for all computer issues and the endless talk about motorbikes. Thomas, because of your fake Rotterdam accent I felt utterly at home. Thanks for the good times in Canada! Changgong, man, I miss you. Our conversations during Friday beer were the highlight of the week. Come visit me in Japan!

Also thanks to all those who arrived after me, some of whom unfortunately already left. Marcelo and Kai (nice try with the one on one football). Martin, it was great having you around and I definitely hope to see you again at conferences, where you're always at your best. Especially around 5 AM, breaking out the dance moves. My old roommates Philipp (your legacy remains in the room layout) and Bert. And of course Michael, we're missing you. I hope you're having a great time in the US, but concerts will not be the same without you (meine Brille!).

And then there are the current members of our fantastic group! I pay special attention to them, since the chance they read this is in fact larger than zero, and they may confront me with it. Leo, thank you for all the coding advice and discussions we had, but also thank you for being a great friend. Definitely keep me posted on your Japan plans! Just don't bring any powder. And never let go of your coffee cups, no matter the size of the fungi growth. Chrissie, my favorite subject of German jokes. You need to work on your language, son. You have had a bad influence on me ever since I gave you that private tour of the city when you first arrived. Still, I have to admit, you're one of the few funny Germans I know (sorry Elmar, the math jokes just do not cut it). Thanks to Klaus for joining and (to a lesser extent) bringing Christopher, adding two mathematical wonders to the group!

Thomas the German, vegetarian professor (three things I never let you forget – my apologies), even your snide remarks about Pacific Graphics I will miss. If you want me to pull some strings for you at the Pokémon Headquarters, I'll see what I can do. Nicola, my running inspiration. You will soon be the veteran PhD it seems. Thanks for all the good times with Game of Thrones, I will miss you! Let me know when you go for the Tokyo marathon! Nestor, the music man who has seen only three films in his life – about music. Man, we had an awesome time at Graspop, and many good conversations besides. Keep rockin'! Niels de Goen, I will miss your horrible jokes, which sometimes were so bad they were good. Coffee breaks aren't the same without you. Victor, you're one of the few people who saw Jingtang's face in Amsterdam, and I know we will both cherish that memory for a long time.

Peiteng, from how you address me, I can already see I'm having the same bad influence on you as I had on Jingtang. But it may be good for you. Or maybe not. In any case, once you get your first publication out, please take it a bit easier. You will work yourself to death! Yes, there is a piece of serious advice here between all the merriment. Jerry. Apple time. I'm sure you'll do great, and I hereby officially acknowledge and legitimize you as a rendering guy. The new guys, Nasikun, other South-American Leo, and Tim (not Timothy, there can be only one!), I hope I helped a bit in giving you a warm welcome to our group, and I am sure you'll enjoy your time here. Chaoran (thanks for the white rabbit!) and Helwig, I hope you have a great time here.

Even now, while writing this, I already feel sad to leave such a wonderful group of people. You can not wish for a better work atmosphere, and I have made some great friends during my time in the group. Fare well!

Then for my Harvest4D partners. It was great to be part of such a successful project, and I count myself extremely lucky. Not only was the scientific output impressive, I never laughed so much as I did during the after-office drinks at our project meetings. Michi and Max, thanks for the organization! Stefan Ohrhallinger, Mo, Michael Weinmann, Tobias, Paolo, Stefan Roth, Gianpaolo, Reinhard, Stéphane, Simon, thanks for this amazing project. Special thanks to Michael Goesele and Samir, who had me over at Darmstadt for a research visit. Then my late-night fellows, Tamy *Octopus* (haven't laughed like that since), contesting Elmar for the most fun professor, and of course Hélène and Reinhold. The story of the Austrian flag will forever be in my heart.

Degenen die de minste inbreng hadden in dit proefschrift hadden wel de grootste invloed op mijn leven. Al mijn vrienden, jullie weten wie jullie zijn, SJK'ers en TA'ers. Zonder elk weekend met jullie te lachen had ik het nooit volgehouden. Ik ga jullie erg missen. Kom alsjeblieft een keer langs om de boel af te breken. Maar goed, niet te veel getreurd. Ik weet dat het weer als vanouds zal zijn als ik terug ben.

Opa Poe, oma Nelly, oma Willy, ik ga jullie heel erg veel missen. Pap en mam, bedankt voor de onvoorwaardelijke steun die ik al heel mijn leven bij jullie geniet. Zonder jullie was dit allemaal niet mogelijk geweest. Ik weet dat er nu een moeilijke tijd aanbreekt, ook voor mij. Al kunnen we voorlopig even niet meer zomaar bij elkaar langskomen, ik kom weer terug. Val, ik ga je missen, kerel. Ik ben trots op je. Ik hoop dat je eens langskomt in Japan op één van je toekomstige reizen. Je hebt in ieder geval altijd een adresje. 美沙希、人生の意味は幸せになること。だから、ありがとう、心から。

CURRICULUM VITÆ

Timothy Kol was born on April 28th, 1990, in Schiedam, The Netherlands. He got interested in science and computers at a young age, and after attending high school in Schiedam, he started his higher education in 2008 at Delft University of Technology. After an internship in Singapore with the small game developer Nexgen Studio, he obtained his bachelor's degree in Computer Science in 2011. His experiences with Singapore's game industry made him choose a closely related master's program at Utrecht University, and he obtained his master's degree *cum laude* in Game and Media Technology in 2013, with his thesis on real-time cloud rendering on the GPU.

He started his PhD in 2013 under supervision of prof. dr. Elmar Eisemann back at Delft University of Technology, in the Computer Graphics and Visualization group, as part of the HARVEST4D European project consortium. During this time, he successfully supervised two master students for their final thesis, and published and presented several peer-reviewed journal and conference papers, which led to his dissertation: *Representing Large Virtual Worlds*.

Timothy will soon start working as a computer graphics researcher for the animation studio OLM Digital in Tokyo, Japan.

List of Publications

6. **T. R. Kol**, P. Bauszat, S. Lee, E. Eisemann, *MegaViews: Scalable Many-View Rendering with Concurrent Scene-View Hierarchy Traversal*, Computer Graphics Forum (2018).

5. **T. R. Kol**, O. Klehm, H.-P. Seidel, E. Eisemann, *Expressive Single Scattering for Light Shaft Stylization*, IEEE Transactions on Visualization and Computer Graphics **23**, 7 (2017).

4. J. G. Leskens, C. Kehl, T. Tutenel, **T. R. Kol**, G. de Haan, G. S. Stelling, E. Eisemann, *An Interactive Simulation and Visualization Tool for Flood Analysis Usable for Practitioners*, Mitigation and Adaptation Strategies for Global Change **22**, 2 (2017).

3. B. Dado, **T. R. Kol**, P. Bauszat, J.-M. Thiery, E. Eisemann, *Geometry and Attribute Compression for Voxel Scenes*, Computer Graphics Forum **35**, 2 (2016).

2. O. Klehm, **T. R. Kol**, H.-P. Seidel, E. Eisemann, *Stylized Scattering via Transfer Functions and Occluder Manipulation*, Proceedings of GI: Graphics Interface, (2015).

1. **T. R. Kol**, J. Liao, E. Eisemann, *Real-Time Canonical-Angle Views in 3D Virtual Cities*, Proceedings of VMV: Vision, Modeling & Visualization, (2014).

昨日、夢を見た。
　　ずっと昔の夢。
その夢の中では僕たちはまだ十三歳で、
　　そこは一面の雪に覆われた広い庭園で、
人家の明かりはずっと遠くに疎らに見えるだけで、
　　降り積もる新雪には、私たちの歩いてきた足跡しかなかった。
　　そうやって、
いつかまた、
一緒に桜を見ることが出来ると、
　　私も、彼も、何の迷いも無く、
そう思っていた。

新海誠

今もまだそう思っている。