Thesis for the degree of Doctor of Philosophy

Sparse Voxel DAGs

Viktor Kämpe

Division of Computer Engineering
Department of Computer Science and Engineering
Chalmers University of Technology
Göteborg, Sweden 2016

Sparse Voxel DAGs
Viktor Kämpe
ISBN 978-91-7597-447-7

Doktorsavhandlingar vid Chalmers tekniska högskola
Ny serie nr 4128
ISSN 0346-718X

Technical Report 133D
Department of Computer Science and Engineering
Research Group: Computer Graphics

Division of Computer Engineering
Department of Computer Science and Engineering
Chalmers University of Technology
SE–412 96 Göteborg, Sweden
Telephone + 46 (0) 31 – 772 1000
http://www.chalmers.se

Printed by Chalmers Reproservice
Göteborg, Sweden 2016

Sparse Voxel DAGs

Viktor Kämpe
Division of Computer Engineering
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

This thesis investigates a memory-efficient representation of highly detailed geometry in 3D voxel grids. The memory consumption of a plain dense grid scales too fast as the resolution increases to be feasible at very high resolutions. In computer graphics, the geometry is often surface geometry, and representing the data in a sparse voxel octree exploits the sparsity, making the memory consumption scale much better than for a dense grid. The size of sparse voxel octrees is still significant at high resolutions, and this thesis consists of four papers addressing the memory consumption by also exploiting coherence in the data. The coherence is detected automatically in voxel data sets and encoded losslessly in a directed acyclic graph as nodes sharing descendants, as opposed to a tree where all descendants are unique.

The sparse voxel DAG is used to encode hard shadows and static and time-varying opaque surface geometry, and offers just as fast access as trees, at a fraction of the memory consumption. While 1 bit per leaf node implies a lowest memory consumption of 1 bit per occupied voxel in a tree, the sparse voxel DAG repeatedly achieves much lower memory consumptions, e.g., 0.08 bits per voxel. The sparse voxel DAG is not just a single data layout for a single purpose, but a way of encoding coherence in voxel grids. The thesis shows that the limits of tree representations are not the fundamental, nor the practical, limits of how efficiently voxel grids can be represented and used, and advances the limit of grid resolutions to be considered practical in real-time rendering.

Keywords: Computer graphics, Geometry, Visibility, Shadows, Voxel, Grid, Data structures, Tree, Directed acyclic graph,

Acknowledgments

I would like to extend my gratitude to Ulf Assarsson, my supervisor, for his continuous support and insightful guidance in research and the academic world. I would also like to thank all of my former and present colleagues of the graphics group at Chalmers, beside Ulf, that I have had the pleasure to work with: Erik Sintorn, Ola Olsson, Markus Billeter, Dan Dolonius, and Sverker Rasmuson. My understanding of computer graphics primarily stems from our numerous and extensive discussions.

I would also like to thank all colleagues at the division and the department for the numerous discussions of computer engineering in general, teaching, politics and board gaming, to just mention a few reoccurring topics.

I wish to thank my family and my friends for putting up with me and more computer graphics than they sometimes desire. Liv, my beloved, thank you for being you, for your support and our continuous conversation.

Viktor Kämpe
Göteborg, August 2016

List of Publications

This thesis is based on the following appended papers:

Paper I– **Viktor Kämpe**, Erik Sintorn and Ulf Assarsson. *High Resolution Sparse Voxel DAGs.* ACM Transactions on Graphics, 32 (4), SIGGRAPH 2013.

Paper II– **Viktor Kämpe**, Sverker Rasmuson, Markus Billeter, Erik Sintorn and Ulf Assarsson. *Exploiting Coherence in Time-Varying Voxel Data.* I3D '16:Proceedings of the 20th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, 2016.

Paper III– Erik Sintorn, **Viktor Kämpe**, Ola Olsson and Ulf Assarsson. *Compact Precomputed Voxelized Shadows.* ACM Transactions on Graphics, 33 (4), SIGGRAPH 2014.

Paper IV– **Viktor Kämpe**, Erik Sintorn and Ulf Assarsson. *Fast, Memory-Efficient Construction of Voxelized Shadows.* i3D '15:Proceedings of the 19th Symposium on Interactive 3D Graphics and Games, 2015.

Other publications co-authored by Viktor Kämpe:

- Erik Sintorn, **Viktor Kämpe**, Ola Olsson and Ulf Assarsson. *Per-Triangle Shadow Volumes Using a View-Sample Cluster Hierarchy.* I3D '14:Proceedings of the 18th meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, 2014.

- Ola Olsson, Erik Sintorn, **Viktor Kämpe**, Markus Billeter and Ulf Assarsson. *Efficient Virtual Shadow Maps for Many Lights.* I3D '14:Proceedings of the 18th meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, 2014.

- Ola Olsson, Markus Billeter, Erik Sintorn, **Viktor Kämpe** and Ulf Assarsson. *More Efficient Virtual Shadow Maps for Many Lights.* IEEE Transactions on Visualization and Computer Graphics, 2015. (Extension)

- **Viktor Kämpe**, Dan Dolonius, Erik Sintorn and Ulf Assarsson. *Fast, Memory-Efficient Construction of Voxelized Shadows.* IEEE Transactions on Visualization and Computer Graphics, 2016. (Extension)

- Pierre Moreau, Erik Sintorn, **Viktor Kämpe**, Ulf Assarsson and Michael Doggett. *Photon Splatting Using a View-Sample Cluster Hierarchy.* Proceedings of High Performance Graphics, 2016.

List of Acronyms

CUDA	–	Compute Unified Device Architecture
CPU	–	Central Processing Unit
CT	–	Computed Tomography
DAG	–	Directed Acyclic Graph
GPU	–	Graphics Processing Unit
MEMS	–	Microelectromechanical Systems
MRI	–	Magnetic Resonance Imaging
Pixel	–	Picture Element
RAM	–	Random-access Memory
SVO	–	Sparse Voxel Octree
Voxel	–	Volume Element

Contents

Part I

Introductory chapters

Chapter 1

Introduction

Choosing geometric primitives in computer graphics is often a trade-off between the complexity of an individual primitive and the number of primitives needed. The triangle is fairly simple, with a shape well defined by its three vertices, and has been the basic geometric primitive of choice for decades for real-time rendering. Using triangles allows tapping into decades of refinements to dedicated tools and dedicated hardware. Even though triangle meshes are very common for rendering, the triangles are often an approximation of other geometric primitives. For instance, scenes authored in a 3D modeling software by artists and designers often consist of high-order surfaces, e.g., subdivision surfaces, that may be tessellated into triangles, either in the back end of the modeling pipeline or in the rendering pipeline. The high-order surfaces are typically much more complex primitives than triangles, allowing highly detailed content to be authored with much fewer primitives. The opposite strategy is used for regular voxel grids, where each grid cell is a very simple primitive but many grid cells are needed to represent the geometry. A grid cell is often referred to as a volume element (abbreviated to *voxel*, similarly to picture element being abbreviated to pixel). Its position is already fixed by the grid and the only freedom is its content. Each voxel has few degrees of freedom, and to resolve highly detailed scenes, a high voxel-grid resolution is needed.

Voxel grids are suitable for representing volumetric data, e.g., from fluid simulations and measurements with CT-scans and MRI. The simplicity of voxel grids make them compelling also for computations on surface geometry; other geometric representations are then converted into voxel grids, a process often referred to as *voxelization*[1]. Measured surface samples of real environments, e.g., from LIDAR, range image data, and disparity maps, are often reconstructed to a surface in a voxel grid, [Kazhdan et al. 2006; Newcombe et al. 2011]. Reconstructing scenes from measurements can be done without the skills, time and effort required for modeling and makes it feasible to reconstruct very detailed environments in short time. Generating scenes this way is becoming more and more common, since devices that can perform surface measurements, e.g., time-of-flight cameras, are becoming both cheaper and easier to use. One challenge with voxel grids of very high resolution is the memory consumption. The theme of this thesis is to improve the memory efficiency of voxel

[1] Both the process of converting, as well as the result, may be referred to as *voxelization*

grids, to enable higher resolutions and scenes with higher geometric fidelity. There is a vast variety of data typically stored as voxel grids, and the focus of this thesis is limited to two cases: opaque surface geometry, both static and time-varying, and to direct encoding of visibility from point-light sources and directional lights, so called hard shadows.

1.1 Thesis Structure

This collection thesis consists of two parts: introductory chapters and a collection of appended papers. The second part, the collection of papers, is the part containing the research in the format of published and peer-reviewed publications. The first part is intended to provide the context of the appended papers within computer graphics.

Chapter 2

Sparse Voxel Octree

The simplest representation of a voxel grid is a plain dense grid, where all voxels are represented separately. A voxel grid of resolution N×N×N and binary content can then be represented with N^3 bits, which may be small enough at low resolutions, e.g., 1 GiB at 2048^3, but the memory requirement scales cubically with the resolution and each consecutive power-of-two resolution will require eight times more memory than the previous. Fortunately, voxel data sets in computer graphics often exhibit large amounts of sparsity, which can be exploited to encode the data much more efficiently. The octree [Meagher 1982] is a popular sparse tree representation of three-dimensional voxel data. The root of the octree corresponds to the volume of the whole voxel grid, and the eight children correspond to the eight spatial subvolumes formed by splitting each grid dimension in half. The splitting continues recursively, adding nodes in the octree, until a desired resolution is achieved (see Figure 2.1).

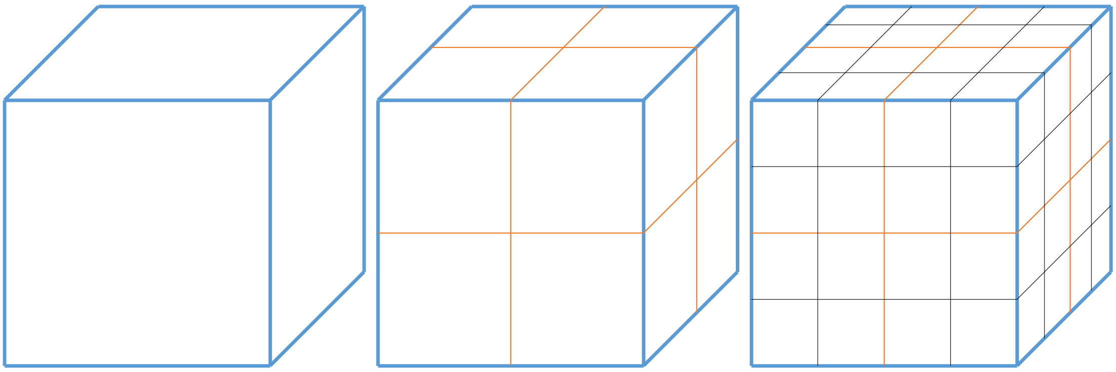

Figure 2.1: Grid resolution of an octree of depth 0, 1, and 2.

With the tree structure on top of the grid, data can be stored hierarchically. For instance, when the whole subtree of a child is homogeneous or empty, it can be encoded at the parent node. A subtree describing empty space can then be replaced by a single bit at the parent, and the octree becomes sparse with a varying 1-8 child nodes for all internal nodes, and 0 children for the leaf nodes. The structure of the children can be encoded compactly with an 8-bit child mask (see Figure 2.2 for a visualization of a sparse quadtree in 2D). An octree that exploits sparsity is sometimes referred to as a sparse voxel octree (SVO). Its final memory consumption

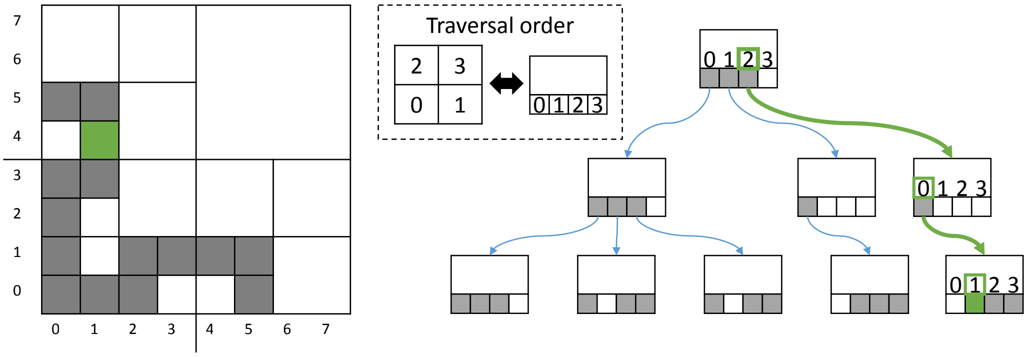

Figure 2.2: The content of a 2D voxel grid represented in a sparse quadtree. One occupied voxel and the corresponding path in the quadtree is highlighted in green.

does not directly depend on the voxel-grid resolution, but rather the number of nodes in the tree. The number of leaf nodes increases as the number of occupied voxels, e.g., the number of voxels representing surface geometry, which typically increases much slower than the increase in the total number of grid cells.

Locating the node of a voxel in an octree involves traversing the octree from the root along a path determined by the mapping between subvolumes and children. It is common to order the children according to a Morton order (z-curve) of the subvolumes, which enables a convenient relation between the grid position and the traversal path in the tree; interleaving the bits of the integer position of the x,y,z components yields the path in the octree as consecutive 3-bit integers (2-bit integers in a quadtree, see Table 2.1). Since it is very easy and cheap to compute the position during traversal, there is no need to explicitly store the position of a voxel in the octree. With a breadth-first ordering of the nodes, all children will be stored consecutively. The octree can be encoded with very simple nodes containing an 8-bit child mask, to indicate which child volumes that contain geometry, and a pointer to the first child node. Together, they implicitly encode the location of all children. The nodes can also contain other, optional, attributes but the child mask and pointer define the topology of the tree.

Table 2.1: Translating between grid position and path is just a matter of interleaving bits of the position components, here for the marked voxel in Figure 2.2. Ordering the children of each node in a Morton order, here primarily according to the vertical position (y) and secondarily according to the horizontal position (x).

Position	y=4	1	0	0
	x=1	0	0	1
Path	child	2	0	1
	level	0	1	2

2.1 Lower Bound of Memory Consumption

When we only have opaque geometry, the pointers can be a significant part of the memory consumption of the octree. There are clever methods to decrease the size of the pointers, e.g., cheaper near pointers and fewer expensive far pointers [Laine and Karras 2010a], and even eliminating all pointers when a single traversal order is enough [Schnabel and Klein 2006], but the child masks are still essential. Without pointers, the memory consumption has a lower limit, due to the child masks, of 1 bit per leaf voxel. When we also consider the child masks of the internal nodes, the amortized memory cost per leaf voxel becomes $1 + \frac{1}{8} + \frac{1}{8^2} + ...$ bits for a full tree, which asymptotically becomes $\frac{8}{7}$ bits per leaf voxel, easily estimated with the limit of the geometric series:

$$\text{Bits per voxel} = \sum_{k=0}^{L} \frac{1}{x^k} \rightarrow \frac{x}{x-1}, L \rightarrow \infty,$$
$$\text{where x is branch factor and L is number of levels}$$

Laine and Karras [2010b] measure the average branch factor of SVOs for increasing grid resolutions and find that it asymptotically goes toward 4 when the voxel grids contain surface geometry. With an average branch factor of 4 in an 8-way tree, the child masks uses half the bits to encode empty space and the amortized cost becomes 2 bits per geometry-containing child. With a branch factor of 4, the memory consumption asymptotically becomes $2 \cdot (1 + \frac{1}{4} + \frac{1}{4^2} + ...) = 2\frac{2}{3}$ bits per leaf voxel.

In the following chapters, we will consider the generalization of a tree, i.e., directed acyclic graph (DAG), and show how it can represent the same geometry with fewer nodes and achieve memory efficiencies below 1 bit per leaf voxel.

Chapter 3

Directed Acyclic Graph

This chapter explains how we convert an SVO into a DAG with fewer nodes while preserving the information. Representing a voxel grid as an SVO results in a bijection between paths and voxels; if, and only if, there is a path from the root to a leaf in the SVO, there exists a non-empty voxel at the corresponding position. This implies that we can encode the information differently, as long as we preserve the set of allowed paths from the root to the termination in leaf nodes. As described in Chapter 2, the positions of individual voxels are implicit from the traversal paths and not stored in the sparse voxel octree, so two identical voxels with different spatial positions can, potentially, be encoded with the same node. By determining sets of equivalent nodes, we can reuse a single node for several paths and remove redundant nodes. When nodes store additional attributes, these also have to match to allow replacement with a single node. For voxel grids containing complex and varying attributes per voxel, we are less likely to find coherence to remove nodes. When we encode only geometry, or encode attributes in a separate structure, we only need to preserve the set of paths, i.e., any graph will do as long as the child masks encountered during traversal are identical to those encountered in a tree traversal (see Figure 3.1).

By encoding several voxels with the same node, we can represent voxel grids with fewer nodes than an SVO and potentially reduce the memory consumption and, therefore, potentially handle higher geometric resolutions. For comparisons of memory consumption, both child masks and pointers have to be accounted for,

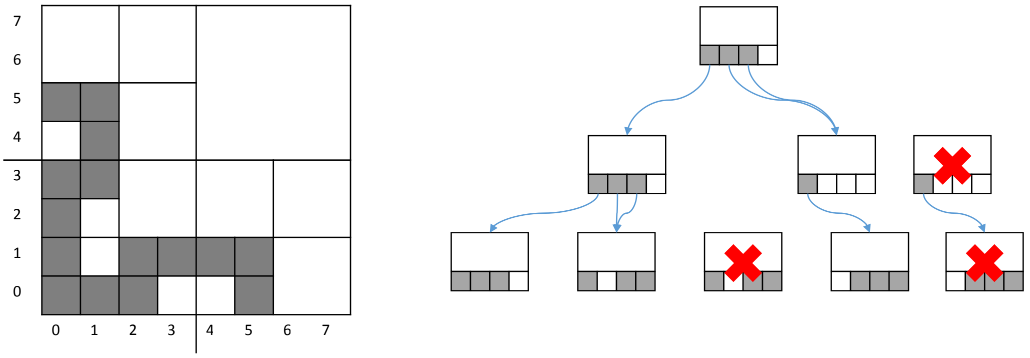

Figure 3.1: Since the set of paths define the geometry, the quad tree representation in Figure 2.2 is equivalent to a directed acyclic graph with fewer nodes.

and more pointers are needed per node in the DAG compared to the SVO. In the SVO, a single pointer per node is enough, since the child nodes can be arranged consecutively in memory. However, since a DAG node may have several parents, the same reordering of nodes is not possible, and a pointer per child node is required in the DAG.

Chapter 4

Surface Geometry

The geometry in computer graphics is often expressed as surfaces, and to mimic, or capture, the geometric complexity of real-world environments, many surfaces with a lot of surface details are needed. For voxelized geometry, the amount of geometric details is ultimately bound by the voxel-grid resolution, and, therefore, we would ideally have extremely high voxel-grid resolutions. To manage the memory consumption of high voxel-grid resolutions, we need highly memory-efficient representations, and in Paper **I** and Paper **II**, we show how a directed acyclic graph can be used to improve the memory performance of static and time-varying voxelized surface geometry, respectively.

Similar methods have been proposed for slighly different data or for slightly different purposes. Webber and Dillencourt [1989] focus on quadtree representations of binary cartography data and improve memory efficiency by merging common subtrees of a quadtree representation. They state that straightforward extension from binary data to non-binary data would potentially scale poorly. Parker and Udeshi [2003] focus on solid modeling of MEMS devices and merge common subtrees for 3D voxel data but do not separate geometry and material properties and require both geometry and a color to match for merging. They achieve memory savings for scenes with axis aligned modeled MEMS devices and present methods that exploit the shared subtrees for improving the speed of meshing with marching cubes, construction of splat trees, and connected component labeling.

4.1 Paper I

This paper introduces the sparse voxel DAG, a generalization of SVO which enables significantly better memory efficiency.

4.1.1 Problem

Voxel grids need very high resolution to represent highly detailed environments. The memory consumption of dense representations scales with the total number of grid cells, i.e, cubically with the grid resolution, which quickly results in a prohibitive memory consumption for higher resolutions. A sparse voxel octree represents surface geometry much more efficiently, since the memory consumption scales as the number of non-empty voxels covering the surface area, i.e., close to quadratically with grid resolution, but still requires a significant amount of memory at higher resolutions. In this paper, we investigate how the memory consumption scales when static surface geometry is represented with a directed acyclic graph.

Interactive applications, e.g., games, also require real-time rendering performance. Ray tracing with an acceleration structure typically scales well with the number of primitives, since primitives can be culled hierarchically during traversal. However, in practice, the performance also depends very much on the implementation and hardware used for rendering. A geometric representation is viable for ray tracing only if it can be efficiently traversed, so we also investigate the traversal performance of the sparse voxel DAG.

4.1.2 Algorithm Overview

We construct an SVO representation for the voxel grid and convert the tree structure into a DAG bottom up, one level at a time, starting with the leaf level. The nodes at the leaf level only contain an 8-bit child mask, and we identify identical child masks by sorting them as if the child masks were 8-bit integers, which makes identical nodes appear in consecutive sequences. Identical nodes are replaced by a single instance, and the pointers in the level above are redirected to this unique instance. This procedure is repeated, level by level, up to the root node. For the higher levels, we identify identical nodes, i.e., identical subtrees, by comparing child pointers, so we sort on the 256-bit integer formed by the nodes' eight child pointers. When there are no identical nodes in a level, or when we reach the root, we have efficiently merged all similar subtrees.

4.1.3 Contributions

We show that encoding surface geometry as a sparse voxel DAG scales better than SVOs to high voxel-grid resolutions, and we achieve memory consumptions down to 0.08 bits per non-empty voxel, far below the lower limit of 1 bit per non-empty voxel that is applicable to SVOs. We also show that the DAG can be traversed on the GPU with performance comparable to state of the art in voxel ray tracers and triangle ray tracers.

4.1.4 Methodology

We evaluate the scaling of the DAG representation from 2K^3 to 128K^3 resolutions in terms of coherence, measured as the reduction in the number of nodes, and in terms of final memory consumption. We also compare against the scaling of efficient sparse voxel octrees [Laine and Karras 2010a] and plain sparse voxel octrees with idealized memory consumption [Schnabel and Klein 2006] (see Figure 4.1 for a visualization of the coherence in a voxel scene). The test geometry is from game-like assets, a laser scanned model, and a highly irregular scene. The construction speed of the DAG is

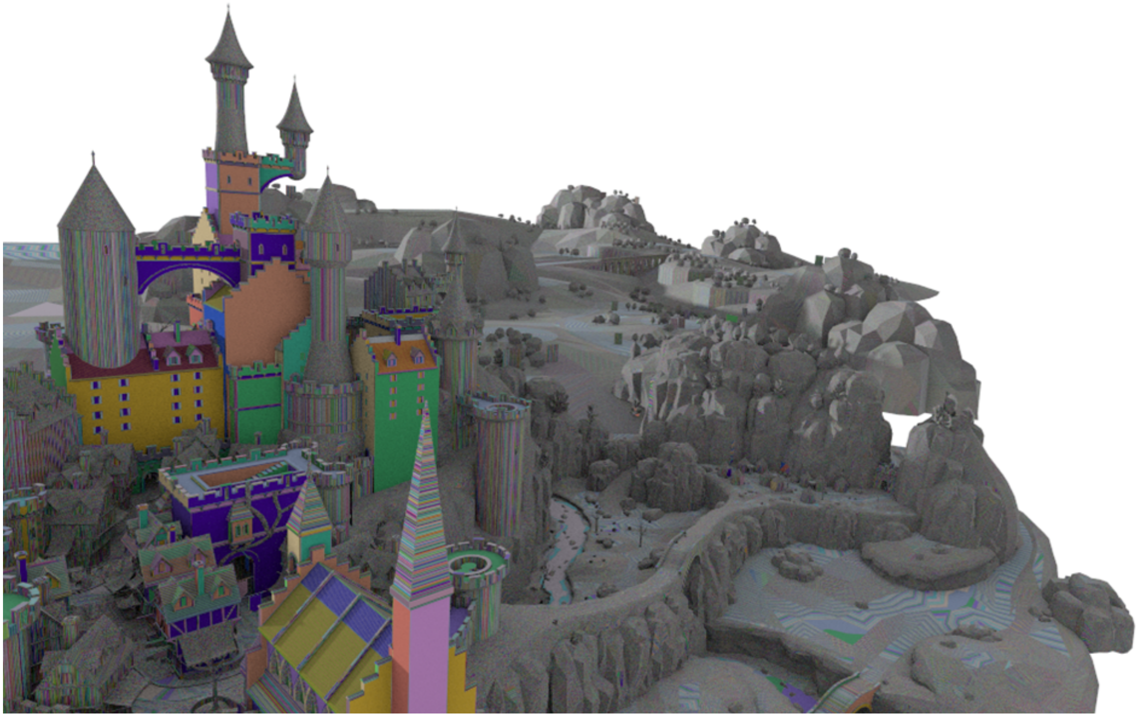

Figure 4.1: An 128K^3 resolution voxelization of the EpicCitadel triangle mesh. The voxel data is stored as a directed acyclic graph and, for visualization purposes, each node has been assigned a unique color. Here, the color corresponds to the node encoding identical 8×8×8 subvolumes, and the coherent coloring show that a single node encodes several subvolumes.

measured on a desktop computer with an Intel Core i7 3930K CPU. The ray-tracing performance is measured on an NVIDA GTX 480 GPU and an NVIDIA GTX 680 GPU and is compared against the voxel ray tracer implementation by Laine and Karras [2010a] and the triangle ray tracer implementation by Aila et al. [2012] on the same platforms.

4.2 Paper II

This paper extends the sparse voxel DAGs with the temporal dimension to exploit both spatial and temporal coherence present in time-varying voxel data.

4.2.1 Problem

Representing very detailed static surfaces in a voxel grid requires a lot of memory. Representing very detailed and time-varying surfaces in voxel grids increases the total memory consumption dramatically, but when a single time step is rendered at a time, all geometry is not used simultaneously and there is an opportunity for streaming portions of geometry in a chronological frame-by-frame order. A static scene, or single frame of time-varying voxel data, that is resident on the GPU along with an acceleration structure can be ray traced with approximately the same efficiency as triangle meshes. Adding the temporal dimension does not change the performance requirements of the rendering but increases the performance requirements of feeding the GPU with new frames. Long time sequences of time-varying voxel data, e.g., free viewpoint video with voxelized geometry, can consist of many thousands of frames that need to be streamed from local storage, e.g., an SSD, or over a network. This makes it crucial to have a representation that can be efficiently compressed, while stored and streamed, and decompressed at the rate of playback or rendering.

4.2.2 Algorithm Overview

The first step to decrease the memory consumption is to reduce the number of nodes. From sequences of SVOs, one per frame, we construct a DAG per frame, just as in Paper **I**. Thereafter, we exploit temporal coherence between timesteps, redirecting node pointers within a frame to identical nodes, when such exists, in previous frames. This converts the separate DAGs to a single and intertwined temporal DAG with a root node per frame. The resulting temporal DAG has the same data layout as in Paper **I** and is traversable with the same method. The DAG nodes do no longer belong uniquely to a single frame, as they may be referenced from several frames. To facilitate frame-by-frame streaming, the nodes are arranged according to the first frame they are referenced in.

The second step to decrease the memory consumption is to reduce the average size of the nodes. The temporal DAG is compressed into a dense bit stream that is only possible to traverse in a single pre-determined order, but this traversal order can be used to recover the original temporal DAG, e.g., after streaming and before rendering. In the dense bit stream, the pointers are encoded with fewer bits and, inspired by pointerless octrees, two types of pointer values are encoded implicitly. The first type of implicit pointer values is the very first reference to a node, which can be encoded in the child mask and eliminate one reference per node. The second type of implicit pointer value encodes a volume that is identical in the previous frame, with the caveat that a pointer may extend several paths encoding different volumes. We limit the case to the first volume that the pointer encodes using the ordering of the nodes: breadth-first within frame-first.

4.2.3 Contributions

We show that there is significant temporal coherence in time-varying voxel data that can be exploited by encoding the frames in a temporal DAG, and that the compression to a dense bit stream reduces the memory consumption further. The final memory consumption in the compressed format achieves bit-rate requirements in the range of 2 to 55 Mbits per second – small enough to stream over a network or from disc. We show decoding speeds, from the dense bit stream back to a traversable state, with faster than playback performance on a single-core laptop CPU.

4.2.4 Methodology

The amount of coherence was compared as the reduction in nodes, and the memory consumption was compared as the size of the dense bit stream to be used during streaming and storage. The temporal DAG was compared against simpler data structures with less ability to exploit coherence, such as SVOs, difference trees and non-temporal DAGs, on time-varying voxel grids with spatial resolutions between 512^3 and 2048^3 and up to 480 frames.

The decompression performance, from dense bit stream to the traversable format, was measured on an Intel Core i7 2630QM CPU.

Chapter 5

Shadows

Shadows are important for the perceived realism in computer-generated images and to understand spatial relations in a scene. In principle, the visibility of the light source from a sample can be determined by tracing shadow rays from the sample to the light source. This is a general approach that can be applied to both area lights and point lights and is often used in offline rendering but it is, due to performance reasons, typically not used in real-time rendering applications like games, where quality needs to be traded for speed. For a survey on real-time shadows, see the book by Eisemann et al. [2011].

To obtain an estimate of the light-source visibility in real time, many games use shadow maps [Williams 1978], a two pass method. The idea is to render one image, i.e. a shadow map, from the light's point of view and one image from the camera. The shadow map is a representation of all lit surfaces, and a shadow query for a surface sample is resolved by determining if the sample belongs to the lit surfaces in the shadow map. The first pass utilizes hardware rasterization to render a depth map of the closest surfaces from the light's point of view, i.e., a range image with samples of the lit surfaces. The second pass compares the depth of each rendered point with the closest depth sample in the shadow map, sampled via texturing hardware, and when the two depths are similar, the point is classified as belonging to the lit surface; otherwise, it is in shadow.

Many problems with shadow maps arise due to the non-perfect match between samples from the first and second pass. One issue is the ambiguity when the depths are similar; they can be two samples from the same surfaces or from two nearby surfaces where one is shadowing the other, and incorrect classification can lead to a surface incorrectly shadowing itself or light bleeding through an opaque surface. The matching can be improved, for instance, by increasing the shadow-map resolution. Another issue is aliasing. The first source of aliasing, the initial sampling error, is undersampling of the scene geometry in the shadow map. This can be alleviated by, for instance, increasing the shadow-map resolution. The second source of aliasing, the resampling error, is shadow-map lookups undersampling the shadow map. This can be alleviated by filtering the visibility, but not the depths, to the frequency of the lookups.

5.1 Paper III

This paper presents a method to use sparse voxel DAGs to represent pre-computed shadows compactly and pre-filtered for large scenes. The data structure can be queried very efficiently in real time for shadow values even for very large filter kernels.

5.1.1 Problem

Many games contain large and geometrically rich outdoor scenes illuminated by a directional light source, e.g., the sun. Using a single shadow map for this setting is challenging, since a very high resolution is needed to sample the geometry with high enough frequency, and may take a lot of time to render, more than can be afforded per frame, and consume more memory than we have GPU RAM. Additionally, a lot of filtering is required where the lookups are sparsely distributed in the shadow map, e.g., far from the camera's viewpoint.

For scenes where both shadow casters and light source are static, the visibility can be precomputed to a texture, so called light maps. The light map can be computed with a high resolution, and the lookup is reduced to a single texture lookup with conventional texture filtering. One disadvantage is that visibility information only is available at the static surfaces, and dynamic geometry cannot receive shadows.

Cascaded shadow maps [Engel 2006] is a method that decreases the resampling error by rendering several smaller shadow maps along the view frustum, better matching the shadow map sample rate to that of the primary view, but does not solve the initial sampling error.

5.1.2 Algorithm Overview

We precompute the visibility for a directional light in a high-resolution 3D-grid domain, allowing both dynamic and static geometry to receive the shadows of static geometry. The grid is aligned with the directional light, and the use is similar to what could, in principle, be achieved by a very large shadow map, but at a fraction of the memory consumption and with much better filtering opportunities.

The grid is constructed from a shadow map, a tile at the time, into an octree and later converted into a directed acyclic graph. In each node, each child is either described by a DAG or directly encoded as fully lit or in shadow. Since the voxel grid has very high resolution, the geometry can be captured at high resolution and limit the initial sampling errors. The representation is also efficient to prefilter per node, e.g., averaging visibility values, since it contains visibility and not depths, which decreases the resampling error. A single lookup requires a traversal to the corresponding node in the DAG, which is more expensive than a simple texture fetch, but percentage closer filtering is very cheap for large, grid-aligned, filter kernels, since an individual visibility value is a bit, and several can be fetched simultaneously in a word.

Due to the hierarchical representation, the full octree subdivision will only happen along the boundary between lit and shadowed, i.e, along the shadow-casting geometry and the shadow boundary in the air between shadow caster and receiver. By aligning the grid with the directional light's direction, the shadow boundary between shadow caster and receiver results in mostly identical subvolumes and, therefore, very few nodes in the DAG. The memory consumption of the DAG will then be dominated by the nodes corresponding to the surface voxelization of the shadow-casting geometry, and scale as the voxelization of the lit surfaces.

With the assumption that no lookups will be performed inside closed geometry, we identify volumes that can be set to arbitrary visibility values. By setting the arbitrary values to form larger uniform regions, we reduce the depth of the DAG and reduce the memory consumption further. The DAG will only be constructed to the finest resolution along the shadow boundary between shadow caster and receiver, and the memory consumption will scale as the voxelization of the silhouettes of the shadow-casting surfaces, i.e., scale like the voxelization of a curve.

5.1.3 Contributions

We present a very memory-efficient representation of visibility that is very fast to filter. Even though visibility is encoded in a high-resolution 3D grid domain, the memory consumption scales as the voxelization of the silhouettes of the shadow-casting geometry for closed geometry and as the voxelization of the lit surfaces for non-closed geometry.

5.1.4 Methodology

We construct the voxelized shadows for directional lights in large, game-like scenes. We compare the resulting memory consumption of the voxelized shadows to half-float precision (16-bit) shadow maps at resolutions equivalent to shadow-map resolutions from 1k×1k to 256k×256k.

The voxelized shadows are constructed on the CPU, incrementally, from smaller tiles of the full shadow map rasterized in OpenGL. The lookups in the voxelized shadows was implemented in CUDA as a post-process pass in a deferred-shading pipeline. The lookup involves traversing the DAG, and percentage-closer filtering is made with bit-mask operations on visibility values in 64-bit words that correspond to computing the visibility for an 8×8 texel tile in the shadow map at a specified depth. The lookup performance is evaluated along one camera fly-through per scene and compared to a cascaded shadow map implementation with 4 and 8 cascades, respectively. All measurements were carried out on a desktop computer with an Intel Core i7 3930K CPU and an Nvidia GTX Titan GPU.

5.2 Paper IV

In this paper, we optimize the construction of our voxelized shadows to scale closer to the final memory consumption. The execution time for the construction is reduced from hours to seconds while also reducing both the runtime memory consumption and the final memory consumption significantly.

5.2.1 Problem

In Paper **III**, *Compact Precomputed Voxelized Shadows*, the construction process takes minutes to hours and can be considered an offline process, meaning that a game needs to ship with the precomputed structure, and shadow-casting geometry cannot be affected by game play. With higher construction performance, the computations can be moved from the back of the production pipeline, to level-load time or to checkpoints, which allows the shadow-casting geometry to be semi static and possibly updated during the game play.

The directional light explored by Paper **III** is commonly used for sunlight in games, but it is also common to have many point lights with a smaller light radius and lower requirements of resolution. In this paper, we present modifications that extend the method to point-light sources and demonstrate voxelized shadows, both construction and rendering, with hundreds of point lights.

5.2.2 Algorithm Overview

Instead of constructing full subtrees and then detecting identical ones, we detect when certain subvolumes will become identical before constructing all the corresponding subtrees. The key is to detect volumes without lit surfaces, i.e., shadow boundaries in mid air, which have identical shadow slices along the z-direction (light direction). We construct subvolumes of the DAG in Z-order, such that, for the node that is about to be constructed, all nodes for the volume between the current node and the light are finished. When the node does not contain shadow-casting geometry, and the already existing node for the subvolume closer to the light is homogeneous in z-direction, we can directly point to the already existing node, because they are identical (see Figure 5.1).

In Paper **III**, we set the arbitrary visibility values inside closed geometry to form as large uniform regions as possible, and when a uniform region cannot be formed we just set the value to shadow. In the DAG, identical nodes are almost as memory efficient as uniform regions, and, in this paper, we improve the resolving of the closed regions to produce more identical leaf masks.

5.2.3 Contributions

The construction method we present scales, in terms of execution time and working memory, very close to the final memory consumption of the structure and is significantly faster than that presented in Paper **III**. The construction times are improved by up to a factor of 200, reducing the construction times from about an hour to seconds. We also present a faster way of deciding the arbitrary visibility values within closed objects. The new method also results in a significantly lower memory consumption of the final structure, up to 3 times more memory efficient.

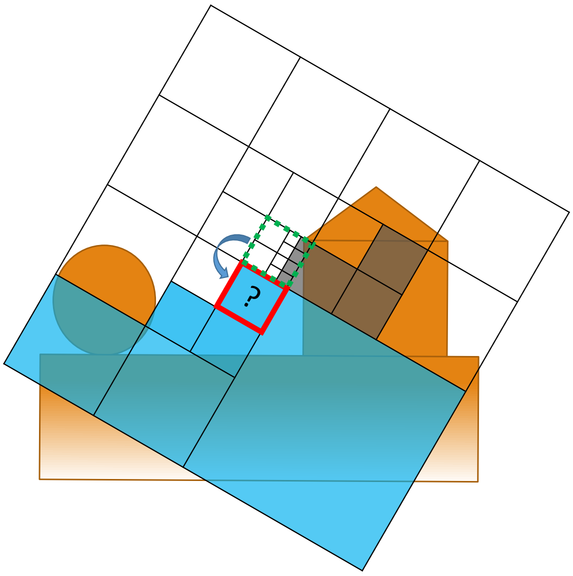

Figure 5.1: For each node to be constructed (blue) we determine if the node closer to the light can be used directly. When a subvolume does not contain lit surfaces, we know that the shadow slice orthogonal to the Z-direction will be identical throughout the subvolume and identical to the slice in the end of the subvolume closer to the light. If the adjacent node is homogeneous in z-direction, we reuse that node directly.

5.2.4 Methodology

The construction is partially done on the GPU, with a CUDA implementation, and partially on the CPU. The full structure is made incrementally from smaller shadow maps of resolution 8192×8192 that were rasterized in OpenGL, similarly to Paper **III**. All measurements were carried out on a desktop computer with an Intel Core i5 2500K CPU and an Nvidia GTX Titan GPU.

Chapter 6

Discussion And Future Work

The theme of this thesis is efficient representation of geometry for large worlds with high geometric fidelity. The scope of this thesis has by no means exhausted the possible solutions to increase the memory efficiency; rather it shows that limits applicable to tree structures, e.g., 1 bit per leaf voxel, are not the fundamental limits for voxel representations. Since Paper **I**, Villanueva et al. [2016] have extended the DAG to also exploit reflection symmetry of surface geometry. There are likely many more types of coherences and techniques to increase the amount of coherence possible to exploit. For instance, in Paper **III**, we choose to align the grid with the light direction, which significantly increases the amount of coherence compared to an alignment to the world-space axes. An interesting line of future work would be to explore other types of coherence, such as coherence at different scales. Fractal patterns can easily be authored by introducing cycles in the graph representation, and, in principle, fractals could be used to represent microscopic geometry. In practice, a simple fractal with few nodes may look very repetitive, and it would be interesting to explore ways to train or fit a fractal by a material example. Apart from microgeometry, material properties are needed in a general rendering context. Dado et al. [2016] show how to efficiently accompany a sparse voxel DAG with material attributes. It would be interesting to also investigate how a DAG can be used to exploit coherence in the material attributes.

Bibliography

Aila, T., Laine, S., and Karras, T. (2012). "Understanding the efficiency of ray traversal on GPUs–Kepler and Fermi addendum". In: *Proceedings of ACM High Performance Graphics 2012, Posters*, pp. 9–16.

Dado, B., Kol, T. R., Bauszat, P., Thiery, J.-M., and Eisemann, E. (2016). "Geometry and Attribute Compression for Voxel Scenes". In: *Computer Graphics Forum* 35.2, pp. 397–407. ISSN: 1467-8659. DOI: 10.1111/cgf.12841.

Eisemann, E., Schwarz, M., Assarsson, U., and Wimmer, M. (2011). *Real-Time Shadows*. A.K. Peters.

Engel, W. (2006). "Cascaded Shadow Maps". In: *ShaderX5: Advanced Rendering Techniques*. Ed. by Forsyth, T. Shaderx series. Charles River Media, Inc. ISBN: 9781584504993.

Kazhdan, M., Bolitho, M., and Hoppe, H. (2006). "Poisson surface reconstruction". In: *Proceedings of the fourth Eurographics symposium on Geometry processing*. Vol. 7.

Laine, S. and Karras, T. (2010a). "Efficient Sparse Voxel Octrees". In: *Proceedings of ACM SIGGRAPH 2010 Symposium on Interactive 3D Graphics and Games*. ACM Press, pp. 55–63.

Laine, S. and Karras, T. (2010b). *Efficient Sparse Voxel Octrees – Analysis, Extensions, and Implementation*. NVIDIA Technical Report NVR-2010-001. NVIDIA Corporation.

Meagher, D. (1982). "Geometric modeling using octree encoding". In: *Computer graphics and image processing* 19.2, pp. 129–147.

Newcombe, R. A., Izadi, S., Hilliges, O., Molyneaux, D., Kim, D., Davison, A. J., Kohi, P., Shotton, J., Hodges, S., and Fitzgibbon, A. (2011). "KinectFusion: Real-time dense surface mapping and tracking". In: *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*. IEEE, pp. 127–136.

Parker, E. and Udeshi, T. (2003). "Exploiting self-similarity in geometry for voxel based solid modeling". In: *Proceedings of the eighth ACM symposium on Solid modeling and applications*. SM '03. Seattle, Washington, USA: ACM, pp. 157–166. ISBN: 1-58113-706-0. DOI: 10.1145/781606.781631.

Schnabel, R. and Klein, R. (2006). "Octree-based Point-Cloud Compression". In: *Symposium on Point-Based Graphics 2006*. Eurographics.

Villanueva, A. J., Marton, F., and Gobbetti, E. (2016). “SSVDAGs: Symmetry-aware Sparse Voxel DAGs”. In: *Proceedings of the 20th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. I3D ’16. Redmond, Washington: ACM, pp. 7–14. ISBN: 978-1-4503-4043-4. DOI: `10.1145/2856400.2856420`.

Webber, R. E. and Dillencourt, M. B. (1989). “Compressing quadtrees via common subtree merging”. In: *Pattern Recognition Letters* 9.3, pp. 193–200. ISSN: 0167-8655. DOI: `10.1016/0167-8655(89)90054-8`.

Williams, L. (1978). “Casting curved shadows on curved surfaces”. In: *SIGGRAPH Computer Graphics* 12 (3), pp. 270–274. ISSN: 0097-8930. DOI: `http://doi.acm.org/10.1145/965139.807402`.