EUROGRAPHICS 2016 / J. Jorge and M. Lin
(Guest Editors)

Volume 35 (2016), Number 2

Compressed Multiresolution Hierarchies for High-Quality Precomputed Shadows

Leonardo Scandolo, Pablo Bauszat, and Elmar Eisemann

Delft University of Technology, Netherlands

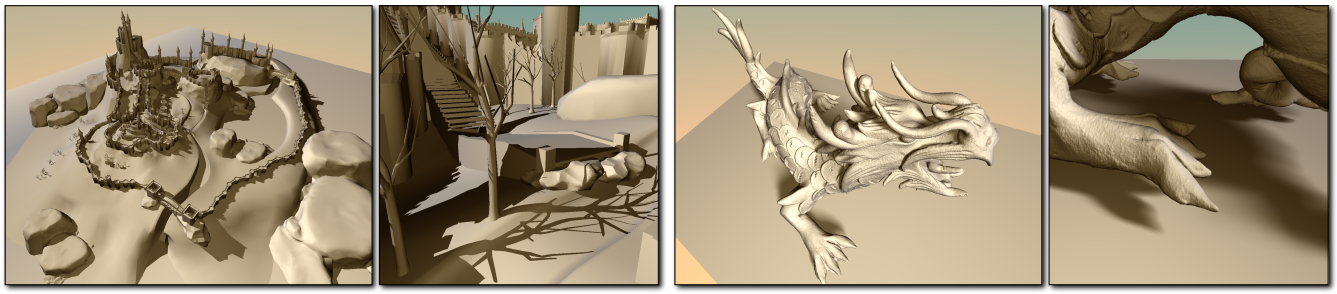

Figure 1: **Left:** High-quality shadows in a static large-scale environment rendered in 1 millisecond using a 32-bit shadow map with a resolution of *1.048.576 x 1.048.576* pixels. The shadow map is compressed from four terabytes down to 160.6 MB (26124:1 ratio) without loss of precision. **Right:**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

Abstract

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

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture, I.4.2 [Computer Graphics]: Compression (Coding)—Exact coding

1. Introduction

High-quality shadows are an important challenge in many real-time rendering applications in computer graphics. Shadow mapping [Wil78] is today's standard for real-time shadows, however, its quality is often limited by texture resolution. High-quality shadows in complex scenes can easily require resolutions up to two orders of magnitude larger than currently feasible for commodity GPUs. Adaptive approaches such as Adaptive Shadow Maps [FFBG01] or Cascaded Shadow Maps [Eng06,ZSXL06] are a common real-time solution, but come at the cost of reduced run-time performance. As virtual scenes often consist of large static parts (e.g., terrains or buildings), precomputing shadows has become a common practice.

Recent advances have shown that precomputed compressed high-resolution shadows maps can be a competitive alternative. Such techniques fully handle shadows cast by static objects on both static and dynamic receivers. Dynamic shadow casters are handled using standard shadow mapping techniques at run-time, with the added benefit of not having to render the static parts of the scene. Unfortunately, conventional image compression is not suitable for shadow maps, because lossless encoding (which is required to avoid light and shadow leaks) does not result in satisfactory compression rates and many techniques rely on run-length encoding, which prohibits random-access queries. Fast random-access compression of color textures has been investigated in the context of

GPU architectures ([DJ98, HIN04, IM06]), but these algorithms rely on quantizing data or lead to low compression rates (up to 5%), which is insufficient for higher resolutions. Consequently, custom schemes for shadow-map compression have recently been proposed. These approaches typically exploit the fact that, in static scenes, any depth value between the depth of the first and second surface underneath a pixel leads to a conservative occlusion test. However, previous approaches do not fully exploit the data coherency or rely on depth quantization.

We introduce a novel compression scheme for high-resolution shadow maps based on multiresolution hierarchies. We propose a sparsification process, which exploits the concept of dual shadow mapping (a shadow map for the front faces and one for the back faces) to create an extremely sparse, but conservative, multiresolution decomposition of the original (front) shadow map. This decomposition is efficiently encoded in a compressed regular tree for fast random access during run-time. We show that our approach achieves higher compression rates than all previous approaches, can be queried with real-time performance, and can be efficiently built using GPU architectures. Our approach is the first to exploit coherency along all data dimensions, does not rely on quantization (maintains full 32-bit precision) and supports shadow maps from arbitrary light sources. Another benefit is that it naturally incorporates all information required for hierarchical filtering operations since it offers a multiresolution representation and every level by itself is a complete shadow map. Finally, we show that our approach can be directly extended from single shadow maps (2D encoding using quadtrees) to a coherent set of shadow maps (3D encoding using octrees). Our approach is the first to enable efficient compression of and rendering with high-quality shadow map sets to produce soft shadows, and moving light sources with known trajectories (e.g., sun lighting).

2. Related Work

We will briefly discuss previous approaches for precomputed compressed shadows and compression of tree hierarchies. For a comprehensive overview of real-time shadow generation, we refer to the surveys of Eisemann et al. [ESAW11] and Woo et al. [WP12].

Compressing with line segments Based on the assumption that shadows are not cast inside of objects, Woo et al. [Woo92] proposed midpoint shadow mapping as a solution to self-shadowing artifacts. Midpoint shadow mapping computes a new shadow map, which represents the intermediate surface lying between the two surfaces closest to the light source. Since all depth values stay between the front facing geometry (the surfaces that represent the original shadow map) and the back facing geometry (the first exit point out of the object), the resulting occlusion test is conservative. An extension of this approach is dual shadow mapping [WE03], where the shadow maps are kept separate and shadow biasing can be performed adaptively. Based on this concept, Arvo et al. [AH05] introduced Compressed Shadow Maps (CSM) and showed how to compress a shadow map by representing each scan-line with a set of line segments approximating the midpoint surface. This approach shows that shadow map compression can be understood as signal compression with a specific spatially-variant bound. Although our approach can be interpreted as a 2D or 3D extension, finding the exact analytic equivalent in higher dimensions is a significantly more complex task.

Ritschel et al. [RGKM07] similarly compresses a set of coherent shadow maps by encoding the depth values of each pixel for all images by using a set of lines. However, both approaches do not fully exploit data coherency along all dimensions (e.g., only along the vertical dimension or "through" the image stack) and, therefore, cannot achieve optimal compression rates. Additionally, since these compression schemes are non-hierarchical they do not adapt well to the underlying data and efficient filtering along dimensions other than the compression dimension becomes impractical.

Precomputed Voxelized Shadows Recently, Sintorn et al. [SKOA14] proposed to precompute shadow information for a voxelized scene representation in projective light-space, which is efficiently encoded in a 2-bit Sparse Voxel Octree [LK10]. The octree is further compressed by subtree merging using a Directed Acyclic Graph (DAG) [KSA13]. The initial compression and construction performance was improved and resulted in the current state-of-the-art compression method for precomputed shadows [KSA15]. Unfortunately, the voxelization process leads to depth quantization and, although, the information is encoded hierarchically, it is not a multiresolution representation and fast filtering requires additional memory, almost doubling the size of the octree. Furthermore, the extension to shadow map sets is difficult since the compression is only efficient in the projected space of the light source.

Tree Compression A large body of research exists for efficient tree-based encoding of multiresolution hierarchies (e.g., [Woo84, Sam85, LH07]). Unfortunately, our tree must support random access and exhibits certain uncommon characteristics, making it difficult to apply most previous techniques. However, to address the overhead introduced by storing topology information, we utilize the pointer compression technique proposed by Lefebvre et al. [LH07] and efficiently encode tree pointers using 16-bits. We do not employ any vector quantization [GG91, CCG96, KE02] or other optimizations to the stored depth values themselves. Our results demonstrates that even without such data changes, our approach outperforms previous compression approaches, while maintaining full 32-bit depth precision.

3. Compressed Multiresolution Hierarchies

Multiresolution decompositions of images (e.g., wavelets [Mal89] or quadtree images [Sam84]) split features into components of different scales, typically storing homogeneous parts at coarser levels and details at finer levels. In consequence, finer levels (which contain more coefficients) are usually sparse, and coefficients are small if they are encoded as differentials to previous levels. Lossy compression exploits this characteristic and removes small coefficients assuming their influence to the composed image is low. However, for lossless compression all coefficients, independent of their magnitude, have to be considered. This results in decreased sparsity and diminished compression (Fig. 2, left).

Our key idea is to compress an alternative representation of the shadow map that is more homogeneous, but still conservative. Our

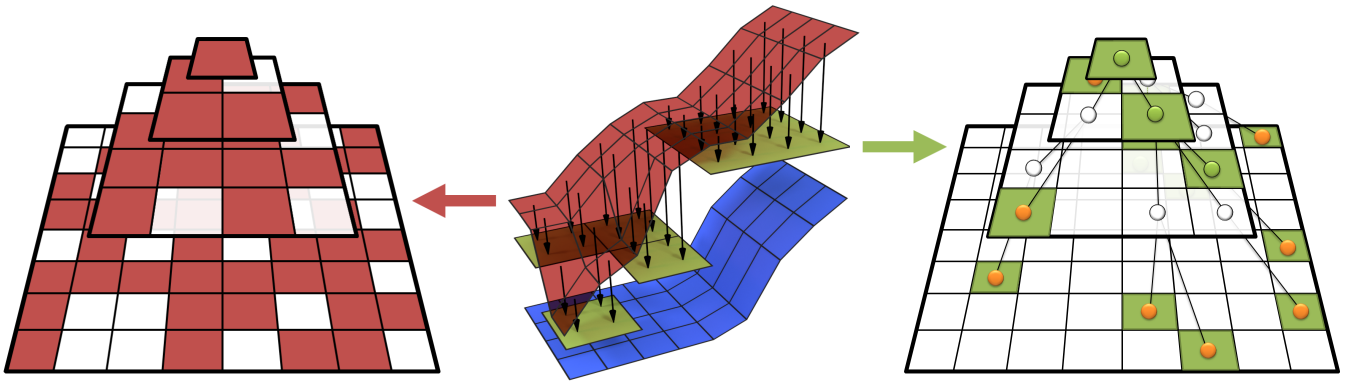

Figure 2: **Left:** A multiresolution decomposition of a shadow map requires many coefficients (red) at finer levels in varying regions and is typically not sparse. **Middle:** Using dual shadow mapping, an intermediate surface (green) can be found between the shadow map (red) and the auxillary second-surface shadow map (blue). Here, the intermediate surface represents a linear, conservative approximation of the shadow map by a set of axis-aligned planes. Choosing these planes to represent common depth values for many pixels results in a more homogeneous occlusion surface. **Right:** A significantly sparser multiresolution decomposition encoding the set of axis-aligned planes. The overlayed quadtree shows the encoding of coefficients. Inner nodes are represented by green circles, while leaf nodes are marked as yellow. Note the empty inner nodes (white circles) which are required to encode the topology information, but do not store any depth values.

goal is to find new depth representatives for each pixel in order to increase the sparsity of the hierarchy, but such that a conservative depth test remains possible. This is achieved by choosing values inside the boundaries defined by the first entry and exit surface points. To compute these bounds, we employ the concept of *dual shadow mapping* (Fig. 2, middle). As a whole, the procedure can alternatively be interpreted as the compression of an image with a spatially-varying error bound defined by an interval that must be met to maintain lossless compression. For fast random-access during run-time, we encode the sparse decomposition using a compressed quadtree (Fig. 2, right).

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

In the following, we will first propose two greedy construction methods for finding sparse decompositions from conservative depth bounds. Then, we cover efficient encoding and traversal of the sparse representation using a compressed quadtree. Finally, we discuss shadow map filtering and propose an optimized traversal technique to significantly reduce filtering costs. While this section focuses on single shadow maps only, we will demonstrate in Sec. 4 how to extend our approach to a set of coherent shadow maps using a compressed octree.

3.1. Construction

Our first task is to define the allowable depth interval for each pixel. While the lower depth bound is defined by the original shadow map, the upper bound is determined using the second layer obtained via depth peeling [Eve01].

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

Having the per-pixel depth intervals of the shadow map, we then find a simplified surface inside the depth bounds which allows for a sparse decomposition. Interestingly, the task of finding an intermediate surface inside a given envelope is a common problem in mesh simplification, and for the 3D case it is known to be NP-hard [AS94]. We propose two greedy approaches, which perform a sparse decomposition and tree construction at the same time. The first one is a top-down approach which tries to globally minimize the number of distinct depth values, while the second one operates in a bottom-up manner and inspects only local pixels from the next finer level. The depth-value hierarchy will then be compressed using a quadtree structure.

Top-down construction The top-down construction starts at the coarsest level, which represents the entire image domain, and greedily selects the depth value which covers the largest number of depth intervals. It then marks all pixels that can be represented by this value as covered. These covered pixels will inherit the depth value from the coarsest level and only the remaining uncovered pixels need to store a separate depth value. The approach then proceeds to the next finer level by decomposing the domain into four quadrants. For each quadrant that contains at least one non-covered value, the algorithm is launched recursively. If all pixels in a quadrant are covered or the finest level is reached, the algorithm stops.

Algorithm 1 Pseudo-code for top-down hierarchy creation

```
function createHierarchy(intervals) :
    sortedIvals ← sort(intervals)
    createNode(rootNode, sortedIvals)
function createNode(node, sortedIvals) :
    bestIval ← 0
    numIvals ← 0
    bestNum ← 0
    for each ival ∈ sortedIvals do
      if isMin(ival) then
            numIvals++
      else
            numIvals- -
      end if
      if numIvals > bestNum then
            bestNum ← numIvals
            bestIval ← ival
      end if
    end for
    sortedIvals ← extractUncovered(sortedIvals, bestIval)
    for each child do
      childIvals ← extractChildIvals(child, sortedIvals)
      createNode(child, childIvals)
    end for
```

Consequently, homogeneous areas result in an early termination of the process.

To find the depth value covering most intervals, a direct approach would be to discretize the depth, create a histogram, and find the bin with the largest number of overlapping intervals. To avoid discretization, we propose an analytic sweep-based algorithm instead. We start by sorting all interval-bound depths (min and max) in ascending order. Then, we sweep through the sorted list and keep track of the number of overlapping intervals by incrementing a counter each time an interval minimum is encountered (we enter an interval) and by decrementing it when a maximum is encountered (we exit an interval). The highest detected count during the sweep leads to the depth representative which covers the maximum amount of intervals possible (Fig. 3). The pseudo-code for the top-down decomposition is shown in Alg. 1.

The algorithm requires a single sorting in the beginning, which can be performed in $O(n \log n)$ with n being the number of intervals (shadow map pixels). Once the initial list is sorted, all subsequent levels only require an $O(n)$ extraction step to retrieve the sorted list of uncovered-pixel intervals for the corresponding quadrant. Since the extraction has to be performed for each level, the overall runtime remains $O(n \log n)$.

Bottom-up construction An alternative is a bottom-up construction, which only considers the subjacent pixels of the next finer level during creation. This approach is better suited for parallel execution and, for all our test scenes, it performs competitively to the top-down construction, while being an order of magnitude faster.

The bottom-up construction is based on the idea of a min-max mipmap creation. Initially, the pixels at the finest level will contain

Algorithm 2 Pseudo-code for bottom-up hierarchy creation

```
childbounds ← [lower,upper]
for level from finestlevel - 1 to coarsestlevel do
  for each pixel in level do
    children ← getChildren(pixel, level+1)
    depthRepresentative ← findBestRepresentative(children)
    bounds(pixel) ← []
    for each child ∈ children do
      if satisfiesBounds(depthRepresentative, child) then
        setNonExistant(child)
        bounds(pixel) ← bounds(pixel) ∩ getBounds(child)
      end if
    end for
  end for
end for
```

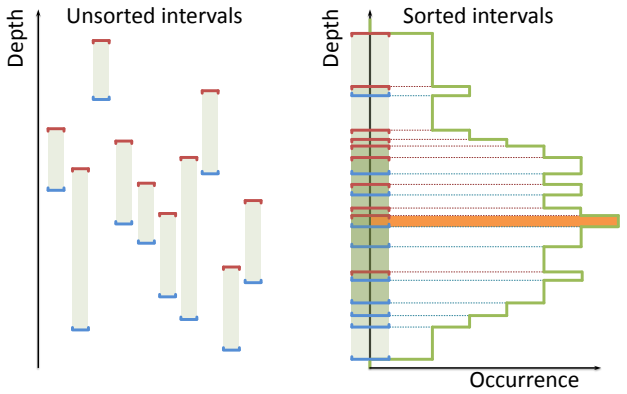

Figure 3: To find a depth value which intersects the maximum number of intervals from a given set (left), we propose a sweep-based mode finding scheme. The interval boundaries are sorted in ascending order first (left). We can then find the mode which corresponds to the best depth value by sweeping through the sorted list and keep track of the number of open intervals.

the lower and upper depth bound. When proceeding one level up, we analyze the four subjacent depth bounds of each pixel P using the same sweeping algorithm as for the top-down approach to find a largest depth interval valid for most of these four pixels. This interval I is then stored in P and all subjacent pixels, whose depth interval contain I are flagged as empty pixels. The algorithm then proceeds upwards to the next level. Once we reach the coarsest level, we will populate the map with actual depth values in every non-empty pixel by storing the average of the interval bounds.

Since only four intervals are treated at a time, the costly sorting step is avoided and the bottom-up construction requires only a constant number of operations per pixel. Hereby, although the complexity stays the same, the algorithm maps better to current GPU architectures, leading to a practical speedup. The pseudo-code is shown in Alg. 2 and Fig. 4 shows an example of the creation of a three level tree. The approach shows similarities to the Mallat-algorithm for wavelet construction [Mal89], but instead of using

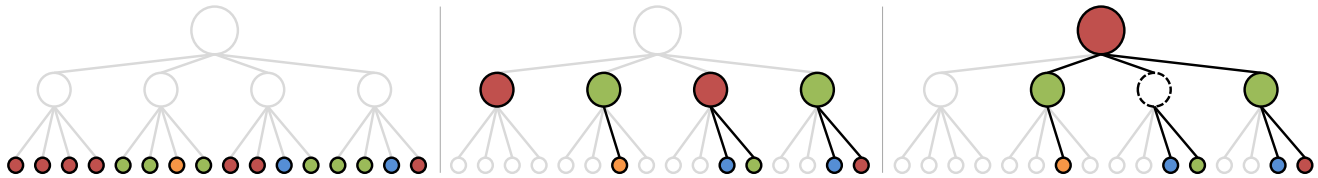

Figure 4: Three steps of the bottom up creation algorithm. **Left:** Initially all values are present in the finest level. **Middle:** The most frequent value is pulled up the hierarchy. **Right:** The procedure is repeated for the next level and an empty node is created to preserve the connectivity.

the average as representative we choose the mode from the set of intervals in order to sparsify the representation.

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

3.2. Compressed Quadtrees

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

Encoding Our quadtree contains three node types: leaves (nodes with no children), inner nodes, and empty nodes. Inner nodes and leaves contain a 32-bit depth value. Empty nodes are only required to encode the quadtree connectivity, but do not contain a value themselves. Unlike other multiresolution decompositions, we do not encode the depth values using parent node differentials. In consequence, only a single value needs to be fetched during tree traversal, which reduces the memory throughput and accelerates lookups.

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

We employ the pointer encoding scheme proposed by Lefebvre et al. [LH07] in order to reduce the amount of memory needed to store pointers. Their scheme stores subtrees close together and allows us to encode pointer offsets, whose magnitudes decrease rapidly per level. Using a per-level scaling, we can efficiently encode pointers even for larger resolutions with just 16 bits introducing only a minimal padding overhead for alignment. We exhaustively search for the optimal per-level scaling factor as proposed by Lefebvre et al. Finally, we also pad full and empty nodes with a single byte, resulting in 4-byte aligned nodes (8 bytes for full nodes, and 4 bytes for empty nodes and leaves). The padding increases the memory footprint, but eases fetching the values on current GPU architectures, hence decreasing lookup times. Alternatively, 24-bit pointers could be used, however, we decided to keep the bit count compatible with the compressed octree representation which will be introduced in Sec. 4. In all our test scenes, no significant difference was introduced by using 16-bit pointers instead of 24-bit pointers for the quadtree compression. If memory footprint is overall more critical than traversal performance, the padding can be removed to further improve the compression.

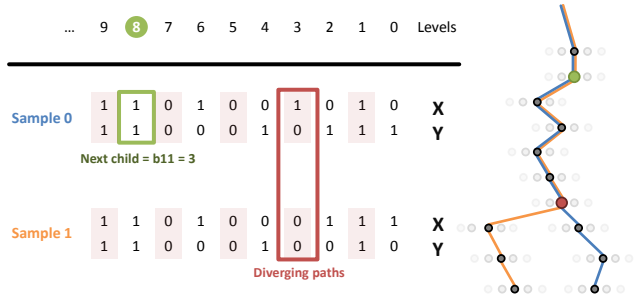

Figure 5: Finding the next child index can be done by inspecting the bits from the x and y position of the query point. The lowest common node of multiple query points can be found by finding the last level where the bits are equal.

Traversal Traversal of our compressed-quadtree encoding follows the same procedure as standard quadtree traversal, but performs lazy fetching of the depth values to account for the presence of empty nodes. The traversal path through the tree is defined by the position of the query point. By keeping track of the current level, we can directly compute the index of the next child using a few bit-operations (see Fig. 5). We start traversal at the root node (which is always a full node) and initialize a pointer which holds the index of the last node containing a depth value. After reading the children mask and pointer, we compute the index of the next child and query the mask to validate the child's existence. We compute the offset of the next child node from the mask and the 16-bit child pointer, to recursively continue the traversal. The pointer to the last position of a depth value is always updated, if we traverse a full node. If a child node does not exist or a leaf node is reached, the depth value is fetched from the last-stored position and the recursion is terminated. We do not query the depth value earlier, as these would

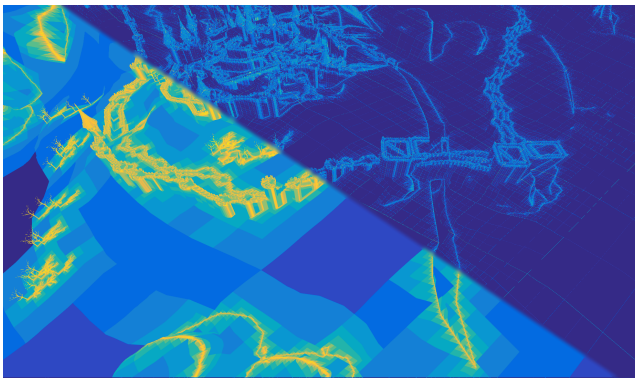

Figure 6: Required traversal steps in the CLOSED CITY scene for a 5x5 PCF filtering using a naive implementation (lower-left triangle of the image) and our optimization finding the lowest common node first (upper-right triangle). Bright colors represent numbers close to the maximum (tree height times number of samples), while dark ones mean that only few levels are traversed.

be unnecessary texture fetches, since we do not store differentials, but absolute values.

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

3.3. Filtering

Efficiently filtering shadow lookups is an important aspect for shadow mapping. Percentage-closer filtering (PCF) [RSC87] is a popular technique which performs averaging of several depth-test results in a fixed-size kernel (usually an $r \times r$ box). A naive implementation of PCF using our approach would perform a full tree traversal for each kernel sample. Since our quadtree encodes a multiresolution prediction, we can perform analytic filtering when all samples end up at the same node. While this is not always the case, most samples share at least a common path from the root node until a certain level. This level can be directly computed from the minimum and maximum query points of the filter kernel and a few bit-operations (see Fig. 5). We propose to traverse all kernel samples together through the first few levels until we find the lowest common node where paths divergence. After that, each sample proceeds individually. This easy-to-implement optimization can lead to drastic improvements in PCF lookup time. A visualization of the amount of traversal steps for the naive implementation and our optimization for a 5x5 PCF filtering is shown in Fig. 6. Another possible optimization is to keep a cache of the last queried sample and reuse it if the next sample shares the same path through the tree down to the retrieved value.

Multiresolution anti-aliasing such as *hierarchical PCF* computes the shadow map footprint of a pixel and looks up the depth value for the corresponding resolution level. Since each of our tree levels encodes a full shadow map of the corresponding resolution, hierarchical filtering is natively supported. When performing PCF filtering, the appropriate sampling level of the hierarchy can be chosen to maintain a 1 to 1 correspondence between screen pixels and

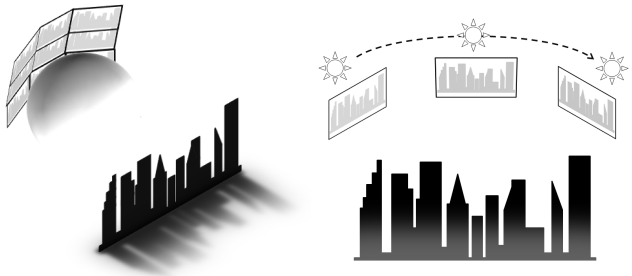

Figure 7: **Left:** Computing soft-shadows from an area light requires to sample multiple close-by points on the light source to apply slight variations to the incoming light direction. The set of shadow maps can be stacked in a 3D image cube for efficient compression. **Right:** Motion of dynamic light source, which is known in advance, can be precomputed by discretly sampling the trajectory, e.g., for simulating high-quality shadows from sun lights.

shadow map texels. This ensures that smooth shadows are present at any view distance regardless of a pixel's projected area in the shadow map. Furthermore, tri-linear filtering can be performed by choosing two consecutive sample levels and interpolating their values in order to create smooth transitions during motion.

4. Shadow map stacks

We can extend our concept of compressed multiresolution hierarchies directly to a set of coherent shadow maps. By stacking shadow maps in a 3D image cube, we can compute a sparse decomposition in the same manner as for a single shadow map and encode it using an octree. This approach is useful for rendering of soft shadows from area lights (Fig. 7, left) or varying light positions (Fig. 7, right). For soft shadows, we sample multiple light positions on an area light using a Hilbert-curve sampler as also used by Ritschel et al. [RGKM07]. The light positions can be jittered in order to avoid banding for smaller sampling rates. For moving light sources, the motion has to be known and the images are simply stacked in the order of discrete sample points along the trajectory. Our hierarchical structure is able to exploit coherency along all 3 dimensions by encoding homogeneous cubic regions in coarser levels of the hierarchy.

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

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

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

Table 1: Compression results for our 2D MH approach comparing to the scanline compression of [AH05] and the voxelized shadows approach of [SKOA14]. Our approach outperforms competing compression approaches consistently while retaining full depth precision.

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

Table 2: A detailed inspection of construction timings and tree characteristics for the multiresolution quadtree compression for the Closed City and Cityscape scene. Rendering times dominate construction times, while creation of the sparse decomposition and quadtree encoding constitutes around one third of the total. We also provide a comparison to the method from Kampe et al. [KSA15].

5. Results

In this section, we demonstrate the compression capabilities of our method for five test scenes and evaluate its construction and runtime. The Closed City scene (613K triangles) represents a typical open-world game setting with both large scale and detailed features. The Cityscape scene (11K triangles) is an example of an architectural design model, while the Villa scene (89K triangles) as well as the Ship scene (810K triangles) are examples of scenes containing many fine scale details. The Dragon scene (7.2M triangles) consists of a scanned model with a very high polygon count.

We implemented larger parts of the construction algorithm on the GPU using NVidia CUDA 7.5. The rendering is done using OpenGL 4.3 and deferred shading, and our measurements are reported for the evaluation of the shadows. All experiments were at a resolution of 1920x1080 on Windows 7 using a PC with and Intel i7-5820K CPU with 16GB of system memory, and an NVidia Titan X GPU. We re-implemented the algorithm of Arvo et al. [AH05] for the comparison to scanline compression. For the comparison to DAG-based compression of voxelized shadows [SKOA14], we used the implementation provided by the authors which includes all the improvements from Kampe et al. [KSA15].

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

Table 2 showcases detailed construction times and total node quantity for several test scenes, as well as the construction time

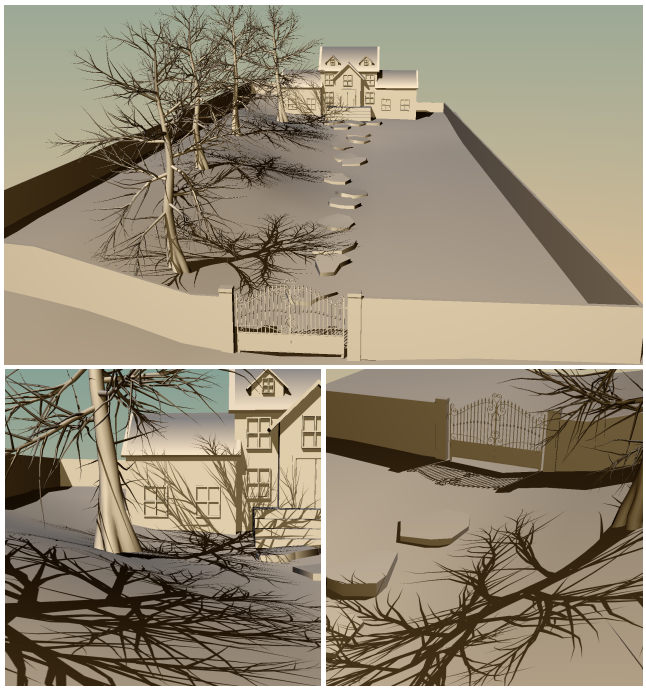

Figure 8: **Top**: An overview of the VILLA scene with an unfiltered $32K^2$ compressed shadow map. **Bottom left:** Close-up of filtered shadows rendered in 2 ms using a 3x3 non-hierarchical PCF kernel. **Bottom right**: Another viewpoint with a 5x5 non-hierarchical PCF filtering kernel rendered in 5 ms.

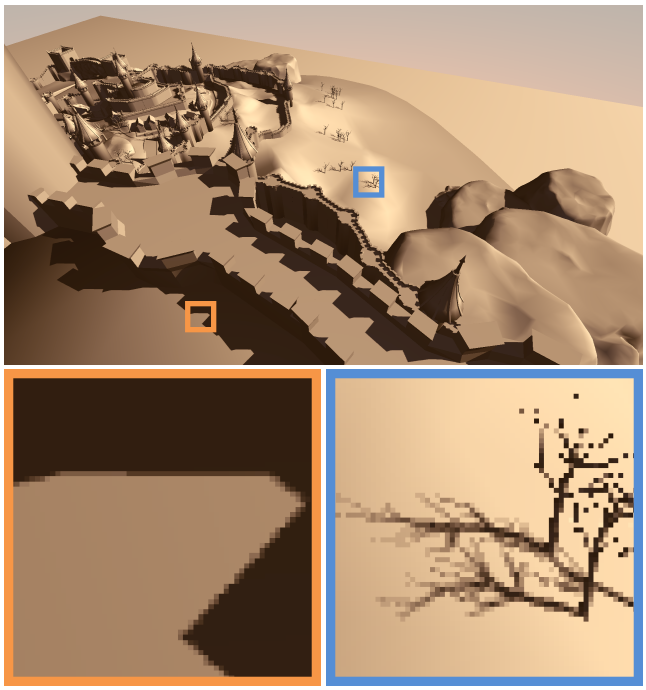

Figure 9: **Top**: An example of hierarchical PCF with a 3×3 kernel in the CLOSED CITY scene. Anti-aliased shadows are present at all distances. **Bottom left**: An inset showing anti-aliased shadows closer to the camera. **Bottom right**: Another inset showing anti-aliased shadows cast by a complex occluder in the distance.

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

Table 3: Traversal time in ms for a single scene (VILLA) for our approach and comparing to standard shadow mapping and the approaches from Arvo et al [AH05] and Kampe et al. [KSA15]. The latter is highly optimized for a cubic 9×9×9 kernel size and for a fair comparison, we only report these numbers.

for the voxelized shadows approach from Kampe et al. [KSA15]. While the preprocessing time is not interactive for larger resolutions, we report numbers in the same order of magnitude as the highly-optimized implementation from Kampe et al. It can be seen that most of the time is spent in the depth peeling in order to obtain the initial depth bounds. The compression itself is mostly dominated by the bottom-up construction that creates the sparse decomposition, and to a lesser extent by the encoding of the quadtree. Finding the optimal per-level scale for 16-bit pointer compression only takes up a small fraction of the overall construction time.

In Table 3 we report timings for single lookup performance and different PCF kernel sizes. We compare our optimized PCF implementation against a naive one, standard shadow mapping (for supported resolutions), and the methods from Arvo et al. [AH05] and Kampe et al. [KSA15] for the VILLA scene. Since it is naturally provided, our implementations perform *hierarchical* PCF. Shared traversal is significantly faster for PCF filtering than a naive implementation (up to 2 ms for a 3x3 kernel, and 4.7 ms for 5x5). The method from Arvo et al. performs well for small PCF kernels at low resolutions, but does not scale well. The voxelized shadow approach is highly optimized for 9x9x9 and 17x17x17 cubic kernels, and achieves almost the same look-up times compared to single lookups. Nevertheless, their filtering is done in 3D space and at reduced precision, which potentially results in artifacts.

In Fig. 8 and Fig. 9, we present visual results for unfiltered, non-hierarchical, and hierarchical PCF filtering using our method. It can be seen that even high-frequency shadows from small features can be faithfully rendered. Additionally, the insets in Fig. 9 show that anti-aliased shadows at any view distance can be achieved by sampling the appropriate level.

As a practical optimization, Sintorn et al. [SKOA14] store the top 6 levels of the hierarchy in a simple dense grid for large resolu-

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

Table 4: Memory footprint of 3D multiresolution octree-based compression for a set of **N** images with different resolutions. As a comparison we report the memory by naively using our 2D quadtree compression for each image individually.

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

Table 5: Memory footprint and construction times of 3D multiresolution octree-based compression for a non-cubic data set of 256 images taken a fixed-trajectory moving light source.

tions. This requires a constant 8 MB of memory, which is negligible at higher resolutions. The numbers we report for their approach include this optimization. In our case, this would allow us to remove the upper 11 levels of the quadtree and halve the number of traversed levels on average. If evaluation time is more critical than compression, this could potentially lead to faster lookups.

Octree compression We evaluate our 3D compression for high-quality soft shadows and light motion, and compare it against naively compressing each image separately with our 2D scheme. Table 4 reports memory sizes for our image stack compression algorithm for soft-shadows. In the table, we show the resulting memory footprint of compressing a set of shadow maps from an area light separately using our quadtree structure and compressing them with our octree approach. It can be seen that our 3D compression provides an additional gain and is able to reduce the compression rate down to 57.9% at best compared to 2D compression.

A visual impression of high-quality soft-shadows in the SHIP scene is given in Fig. 11. Please note that the shadow penumbrae generated in this way is geometrically correct and appears more realistic as opposed to PCF filtering. The lookup time for 32 random samples per pixel out of 512 depth maps is 14 ms, whereas evaluating 64 samples takes 30 ms.

Finally, we evaluate our 3D compression scheme for non-cubic image stacks for fixed-trajectory light sources in Table 5. We show different viewpoints for the CITYSCAPE scene in Fig. 10. Since the construction of the octree is based on cubic tiles, which need to be kept small to fit in GPU memory, the viewport size for rendering is restricted, leading to a large amount of render calls. Therefore, rendering makes up most of the octree creation time.

6. Conclusion and Future Work

We presented a novel compression scheme for shadow maps based on multiresolution hierarchies. We demonstrated that our approach creates high-quality shadows for real-time rendering and achieves high compression rates. For example, our method is able to compress a 32-bit shadow map with a resolution of *1.000k x 1.000k* (uncompressed 4 terabytes) down to 0.0045% at best. Another benefit of our approach is that a multiresolution representation is highly beneficial for fast hierarchical filtering. Using a set of coherent shadow maps, we are able to create soft shadows or dynamic lights on a fixed trajectory.

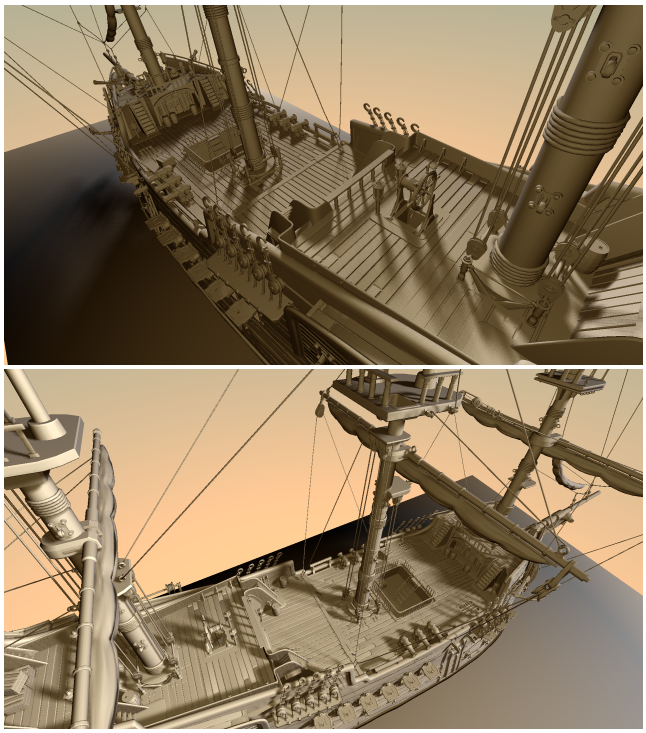

Figure 11: Realistic soft shadows in the SHIP scene generated with an octree from 512 depth maps of $1K^2$ resolution. Overall, the compressed octree is stored in 57 MB. **Top:** A closeup using 32 shadow-map samples per pixel rendered in 14 ms. **Bottom:** Another viewpoint using 64 samples rendered in 30 ms.

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

Additionally, we would like to investigate non-regular subdivision schemes (e.g., based on multiresolution kd-trees) to provide more adaptivity to the underlying depth signal. Still, it is not clear how to efficiently construct these non-regular trees and if the overhead of storing subdivision information introduces a too large overhead. Finally, we want to investigate sparse decompositions that avoid storing topological information for empty inner nodes, e.g., matrix trees [AT10].

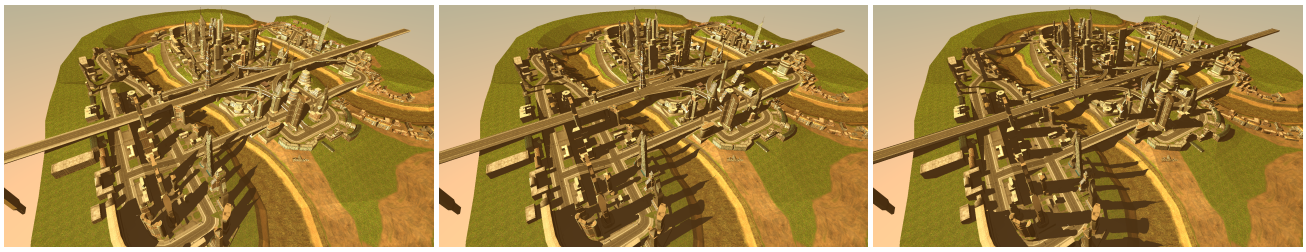

Figure 10: The CITYSCAPE scene shows shadows from different views taken from a compressed shadow map stack of 256 $4K^2$ images taken on a trajectory above the city. The octree has a compressed size of 136.08 MB (uncompressed 16 gigabytes) and is queried in under 1 ms.

7. Acknowledgements

We would like to thank Erik Sintorn and his team for kindly providing their voxelized shadows implementation for our tests, as well as the CLOSED CITY and VILLA test scenes. The DRAGON model is freely available at the Stanford 3D scanning repository[†]. The SHIP scene was modeled by Greg Zaal and Chris Kuhn, and is available under the creative commons license at Blend Swap[‡].

Ths work was partially funded by the EU FP7-323567 project Harvest4D and the Intel VCI at Saarland University.

References

[AH05] ARVO J., HIRVIKORPI M.: Compressed shadow maps. *Vis. Comput. 21*, 3 (Apr. 2005), 125–138. 2, 7, 8

[AS94] AGARWAL P. K., SURI S.: Surface approximation and geometric partitions. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms* (1994), SODA '94, Society for Industrial and Applied Mathematics, pp. 24–33. 3

[AT10] ANDRYSCO N., TRICOCHE X.: Matrix trees. *Computer Graphics Forum 29*, 3 (2010), 963–972. 9

[CCG96] CHADDHA N., CHOU P. A., GRAY R. M.: Constrained and recursive hierarchical table-lookup vector quantization. In *Data Compression Conference* (1996), IEEE Computer Society, pp. 220–229. 2

[DJ98] D. M., J. B.: Directx 6 texture map compression. *Game Developer* (1998), 42–46. 2

[Eng06] ENGEL W.: Cascaded shadow maps. *ShaderX5: Advanced Rendering Techniques* (2006). 1

[ESAW11] EISEMANN E., SCHWARZ M., ASSARSSON U., WIMMER M.: *Real-Time Shadows*. A.K. Peters, 2011. 2

[Eve01] EVERITT C.: Interactive order-independent transparency, 2001. 3

[FFBG01] FERNANDO R., FERNANDEZ S., BALA K., GREENBERG D. P.: Adaptive shadow maps. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques* (2001), SIGGRAPH '01, ACM, pp. 387–390. 1

[GG91] GERSHO A., GRAY R. M.: *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, 1991. 2

[HIN04] HONG Z., IOURCHA K., NAYAK K.: Fixed-rate block-based image compression with inferred pixel values, Aug. 10 2004. US Patent 6,775,417. 2

[IM06] INADA T., MCCOOL M. D.: Compressed Lossless Texture Representation and Caching. In *Graphics Hardware* (2006), The Eurographics Association. 2

[KE02] KRAUS M., ERTL T.: Adaptive texture maps. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware* (2002), HWWS '02, Eurographics Association, pp. 7–15. 2

[KSA13] KÄMPE V., SINTORN E., ASSARSSON U.: High resolution sparse voxel dags. *ACM Transactions on Graphics 32*, 4 (2013). SIGGRAPH 2013. 2

[KSA15] KÄMPE V., SINTORN E., ASSARSSON U.: Fast, memory-efficient construction of voxelized shadows. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2015), ACM. 2, 7, 8

[LH07] LEFEBVRE S., HOPPE H.: Compressed random-access trees for spatially coherent data. In *Proceedings of the 18th Eurographics Conference on Rendering Techniques* (2007), EGSR'07, Eurographics Association, pp. 339–349. 2, 5

[LK10] LAINE S., KARRAS T.: *Efficient Sparse Voxel Octrees – Analysis, Extensions, and Implementation*. NVIDIA Technical Report NVR-2010-001, NVIDIA Corporation, Feb. 2010. 2

[Mal89] MALLAT S. G.: A theory for multiresolution signal decomposition: the wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence 11* (1989), 674–693. 2, 4

[RGKM07] RITSCHEL T., GROSCH T., KAUTZ J., MÜELLER S.: Interactive illumination with coherent shadow maps. In *Proceedings of the 18th Eurographics Conference on Rendering Techniques* (2007), EGSR'07, Eurographics Association, pp. 61–72. 2, 6

[RSC87] REEVES W. T., SALESIN D. H., COOK R. L.: Rendering antialiased shadows with depth maps. *ACM Siggraph Computer Graphics 21*, 4 (1987), 283–291. 6

[Sam84] SAMET H.: The quadtree and related hierarchical data structures. *ACM Comput. Surv. 16*, 2 (June 1984), 187–260. 2

[Sam85] SAMET H.: Data structures for quadtree approximation and compression. *Commun. ACM 28*, 9 (Sept. 1985), 973–993. 2

[SKOA14] SINTORN E., KÄMPE V., OLSSON O., ASSARSSON U.: Compact precomputed voxelized shadows. *ACM Trans. Graph. 33*, 4 (July 2014), 150:1–150:8. 2, 7, 8

[WE03] WEISKOPF D., ERTL T.: Shadow mapping based on dual depth layers. *Eurographics 2003 Short Papers* (2003). 2

[Wil78] WILLIAMS L.: Casting curved shadows on curved surfaces. *SIGGRAPH Comput. Graph. 12*, 3 (Aug. 1978), 270–274. 1

[Woo84] WOODWARK J. R.: Compressed quad trees. *The Computer Journal 27*, 3 (Aug. 1984), 225–229. 2

[Woo92] WOO A.: The shadow depth map revisited. In *Graphics Gems III*, Kirk D., (Ed.). Academic Press, 1992, pp. 338–342. 2

[WP12] WOO A., POULIN P.: *Shadow Algorithms Data Miner*. A K Peter/CRC Press, June 2012. 2

[ZSXL06] ZHANG F., SUN H., XU L., LUN L. K.: Parallel-split shadow maps for large-scale virtual environments. In *Proceedings of the 2006 ACM International Conference on Virtual Reality Continuum and Its Applications* (2006), pp. 311–318. 1

[†] http://www.graphics.stanford.edu/data/3Dscanrep

[‡] http://www.blendswap.com