Scalable Real-time Volumetric Surface Reconstruction

Jiawen Chen Dennis Bautembach Shahram Izadi

Microsoft Research, Cambridge, UK

Figure 1: *We take depth maps from a consumer depth camera (top left) and fuse them into a single surface model (center & right) in real-time using a compact GPU data structure. This allows live reconstruction of large-scale scenes with fine details (rendered w/ ambient occlusion).*

Abstract

We address the fundamental challenge of *scalability* for real-time volumetric surface reconstruction methods. We design a memory efficient, hierarchical data structure for commodity graphics hardware, which supports live reconstruction of large-scale scenes with fine geometric details. Our sparse data structure fuses overlapping depth maps from a moving depth camera into a single volumetric representation, from which detailed surface models are extracted. Our hierarchy losslessly streams data bidirectionally between GPU and host, allowing for unbounded reconstructions. Our pipeline, comprised of depth map post-processing, camera pose estimation, volumetric fusion, surface extraction, and streaming, runs entirely in real-time. We experimentally demonstrate that a shallow hierarchy with relatively large branching factors yields the best memory/speed tradeoff, consuming an order of magnitude less memory than a regular grid. We compare an implementation of our data structure to existing methods and demonstrate higher-quality reconstructions on a variety of large-scale scenes, all captured in real-time.

Links: DL PDF

1 Introduction

Surface reconstruction is an important and established problem in computer graphics and computer vision, with many practical applications particularly for cultural heritage, special effects, gaming, and fabrication. One subclass of this problem takes multiple overlapping, noisy depth measurements of an object or a scene as input and *fuses* them into a single 3D surface representation, which aims to closely reflect the geometry of the real world. Depth can be estimated from regular 2D images, using structure-from-motion (SfM) [Pollefeys et al. 2008] or multi-view stereo (MVS) [Seitz et al. 2006] methods, or from active sensors such as laser scanners or depth cameras, based on triangulation or time-of-flight (ToF) techniques.

For triangulation-based active sensors, one well known approach for surface reconstruction is the *volumetric* method of Curless and Levoy [1996]. This method is particularly compelling as it gives high quality reconstruction results using a computationally simple fusion method. The approach makes no assumptions about the underlying surface topology, uses the redundancy of overlapping depth samples, captures the uncertainty of depth estimates, and fills small holes but leaves unobserved regions empty.

Consumer depth cameras (such as Microsoft Kinect and Asus Xtion) have made real-time depth sensing a commodity. This has naturally led to an interest in applications of *real-time* surface reconstruction; for example, for augmented reality (AR), where the geometry of the real-world needs to be combined live with the virtual and rendered immediately to the user, autonomous guidance, where a robot needs to reconstruct and respond rapidly to the physical environment, or even simply to provide instantaneous feedback to users as they scan an object or scene.

KinectFusion [Newcombe et al. 2011b; Izadi et al. 2011] adopted the method of Curless and Levoy and demonstrated compelling live reconstructions from noisy Kinect depth maps, which were applied to a variety of interactive scenarios [Izadi et al. 2011]. The data structure that underpins this system, and the original Curless and Levoy method, is typically a *regular* 3D grid, uniformly divided into a set of voxels, mapped onto predefined physical dimensions. Although simple to implement, this data structure is memory intensive. Surface geometry and free space are all densely represented, growing cubically to accommodate larger physical volumes (at the same voxel size).

Therefore a fundamental challenge to using such volumetric methods for real-time reconstruction is *scalability*; i.e., to support larger-scale reconstructions without sacrificing fine details or frame rate. This paper addresses this very issue. Specifically we contribute:

- A fast and compact *hierarchical* GPU data structure, capable of dynamic update, supporting fusion of live Kinect depth maps and rendering of surfaces in real-time. This increases the physical size and resolution of the reconstruction volume using an order of magnitude less memory than a regular grid.
- A mechanism for losslessly *streaming* subsets of our data structure between GPU and host, decoupling the active volume from a predefined physical space.

The data structure has been intentionally designed to be flexible, and as a secondary contribution, we experimentally derive the optimal hierarchy layout. We show that with current hardware, a shallow hierarchy of regular grids with relatively large branching factors yields the best memory/speed tradeoff. We demonstrate reconstructions of diverse indoor and outdoor scenes (under natural lighting conditions) using a Kinect camera, with scale, quality, and speed. Finally, we compare to a number of existing methods and demonstrate improved reconstruction quality at scale.

2 Related work

Surface reconstruction aims to create a single 3D representation from noisy measurements. One set of methods, work directly on a complete set of *unorganized points* [Hoppe et al. 1992; Kazhdan et al. 2006; Alliez et al. 2007], and can scale to large datasets using out-of-core streaming techniques [Bolitho et al. 2007]. While these approaches are general, they make no assumptions about the underlying acquisition process, disregarding measurement uncertainty and the temporal nature of capture. For laser range sensors and depth cameras, particularly triangulation-based methods, such information can be crucial to achieving higher quality reconstruction.

To support such sensors and live reconstruction scenarios, methods incrementally *fuse* overlapping depth measurements into a single 3D representation which is accumulated over time. Methods (such as [Chen and Medioni 1992; Higuchi et al. 1995]) first register depth maps into a single global coordinate system using variants of the iterative closest point (ICP) algorithm [Besl and McKay 1992]. They average corresponding depth measurements in overlapping regions, and make assumptions regarding the topology to fit polygons to points parametrically. Mesh zippering [Turk and Levoy 1994] stitches aligned meshes by first removing redundant triangles in overlapping regions and connecting their boundaries. These methods are somewhat resilient to noise as overlapping points are averaged, but are fragile to outliers and can fail in regions of high curvature due to the topological approximations made.

2.1 Volumetric fusion

To overcome some of these limitations, methods using intermediate implicit representations for reconstruction have been proposed. These methods typically use volumetric data structures either storing simple state information such as occupancy [Connolly 1984; Chien et al. 1988] or samples of a continuous function [Hilton et al. 1996; Curless and Levoy 1996]. The former typically generate binary occupancy grids from multiple range images, and share similarities with voxel carving methods based on image silhouettes [Potmesil 1987; Szeliski 1993]. Regular grids and more efficient octree or hierarchical representations have been proposed [Chien et al. 1988; Szeliski 1993]. These occupancy-based methods often only reconstruct the visual hull of an object, typically at low quality.

In computer graphics, implicit surface representations based on signed distance fields are common for rendering and physical simulation (see [Osher and Fedkiw 2003]). The use of these representation for depth map fusion was first proposed by Hilton et al. [1996], and taken further by Curless and Levoy [1996] by accounting for the direction of sensor uncertainty to approximate noise during acquisition. Each depth map is converted into a signed distance field and cumulatively averaged into a regular voxel grid. The final surface can then be extracted as the zero level-set of the implicit function using isosurface polygonization (e.g., [Lorensen and Cline 1987]) or ray-casting [Parker et al. 1998] methods. Wheeler et al. [1998] adapted this method to extract surfaces from voxels based on consensus from multiple range images to increase robustness to outliers.

This form of volumetric surface reconstruction carries many advantages. It supports incremental updates, with fusion simply being the weighted average of existing and new depth samples, approximates systematic noise, and permits easy extraction of polygon meshes. For active triangulation-based sensors, this fusion method generates compelling results [Curless and Levoy 1996; Levoy et al. 2000; Izadi et al. 2011; Newcombe et al. 2011b]. The main drawback of these approaches is *scalability*, which is challenging when real-time reconstruction is also desired. While a number of hierarchical data structures have been proposed [Hilton et al. 1998; Fuhrmann and Goesele 2011], they employ recursive data structures that are computationally expensive, making real-time reconstruction prohibitive.

2.2 Reconstruction from 2D images

Despite the lack of exploration in real-time volumetric reconstruction methods at scale, there has been considerable work on outdoor large-scale 3D reconstruction (see [Musialski et al. 2012] for a detailed review). These systems target general outdoor scenes, and therefore rely on passive 2D cameras to generate depth maps using MVS or SfM techniques [Seitz et al. 2006; Pollefeys et al. 2008]. Unlike active sensors, depth estimation can result in non-systematic noise and outliers, particularly if frame rate is critical. Therefore most systems employ intermediate steps to regularize depth maps further (often using volumetric data structures) by testing for photo-consistency, visibility, and even shape priors across sets of captured images, before performing surface reconstruction using techniques that include Curless and Levoy [Seitz et al. 2006; Pollefeys et al. 2004; Pollefeys et al. 2008; Newcombe et al. 2011a]. Zach et al. [2007] adds explicit spatial regularization to the method of Curless and Levoy, recasting the problem as a global optimization, with impressive results but at the cost of memory and speed.

These systems demonstrate impressive results given very low quality depth maps. However, the additional computational expense of depth estimation and fusion results in a tradeoff between *quality*, *speed* and *scale*. For example, [Pollefeys et al. 2004] show detailed reconstructions of small scenes, but forgo quality at larger scales. [Newcombe and Davison 2010; Newcombe et al. 2011a] show speed and quality, but are limited to desktop-scale scenes. [Pollefeys et al. 2008] trades finer quality for real-time performance at scale. In our work, we use readily-available active depth cameras. This constrains the wider applicability of our method to a subset of outdoor scenes, but still allows reconstructions under different natural lighting conditions. It also lets us focus our contributions on the design of a volumetric data structure that demonstrates all these three properties: scale, quality and speed.

2.3 Real-time reconstruction using active sensors

The tradeoffs between scale, quality and speed have led some researchers to forgo scale completely, and instead explore live reconstructions of smaller scenes and objects, using active sensors to help

achieve interactive rates. In one of the first examples, Rusinkiewicz et al. [2002], use a custom structured light sensor to scan a hand-held object. The online algorithm first aligns point clouds (using a variant of ICP), quantizes samples into a voxel grid, and uses splatting for rendering. Higher quality surface reconstruction is achieved by an *offline* Curless and Levoy implementation. Weise et al. [2009] extend this approach to detect loop closure during a full 360 scan of a hand held object, distorting the object as rigidly as possible to correct for drift errors. Cui et al. [2010] use a hand-held low-resolution, noisy ToF camera and a depth super resolution algorithm to rapidly scan a single small object.

These previous systems demonstrate interactive performance, but are focused on single-object reconstructions. Systems such as [Henry et al. 2010; Hornung et al. 2013; Stückler and Behnke 2012] demonstrate larger scale indoor reconstructions at lower frame rates (ranging from ∼3-10Hz) with active sensors. Point and occupancy based representations are used for reconstruction at scale. The focus of these systems is not high quality surface reconstruction, but instead dealing with the challenges of robust localization, loop closure and correcting for model drift. These are important elements of Simultaneous Localization and Mapping (SLAM) systems, and while out of scope for our work, have been the focus of considerable research in the robotics community [Thrun et al. 2005].

KinectFusion [Newcombe et al. 2011b; Izadi et al. 2011] demonstrated a real-time variant of Curless and Levoy [1996] to fuse noisy Kinect depth maps. The implicit surface is stored in graphics memory as a regular voxel grid, and extracted by raycasting. This limits high quality reconstructions to about $(3m)^3$ physical size at 512^3 voxel resolution, which requires ∼512MB of graphics memory. Yet the compelling real-time results has led to several projects extending the spatial scale of KinectFusion.

2.4 Extending KinectFusion

One simple approach to spatially extend KinectFusion is to *stream* the current volume data out of graphics memory, clear the volume, but maintain global camera pose to ensure new and previously generated surfaces share the same coordinate system. This approach creates multiple overlapping surfaces that need to be merged as a postprocess. *Moving volume* methods take the idea further, defining an active region around the camera which (logically) moves with the sensor. Approaches either transform the entire signed distance field in the active region [Roth and Vona 2012], or use a less expensive rolling buffer that re-indexes the grid and reallocates deactivated regions [Whelan et al. 2012]. The latter system extracts a point cloud from deactivated regions and periodically creates a mesh on the host. These methods still rely on a regular grid, making the active region small to ensure fine quality, and reconstruction is limited to scenes where geometric structures are close by. A clipped active region can also can affect tracking quality and user experience. Finally, once streamed to host, data cannot be reintegrated back into the volume.

Although many efficient hierarchical data structures for rendering exist [Crassin et al. 2009; Laine and Karras 2010], we require real-time *dynamic* updates for fusion and surface extraction, making many of these approaches impractical. Zhou et al. [2011] propose a GPU-based octree which performs Poisson surface reconstruction [Kazhdan et al. 2006] on ∼300K vertices at interactive rates. Zeng et al. [2013] extend this data structure to implement a 9 to 10 level octree for KinectFusion, and show results scaling up to a moderately-sized office. While the closest work to ours, the reliance on an octree imposes significant pointer overhead. We demonstrate that shallower hierarchies are more efficient on current hardware and scale better to larger scenes when combined with streaming. We compare to these existing approaches in Section 6.

3 3D reconstruction pipeline

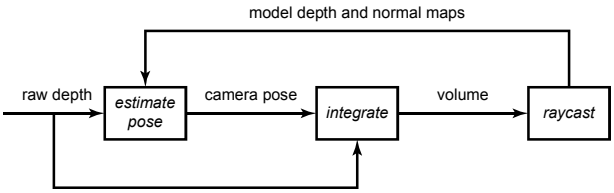

Figure 2: *High level 3D reconstruction pipeline.*

We seek to reconstruct both large-scale and fine-scale surface geometry at real-time rates, by incrementally accumulating noisy depth maps into a memory efficient volumetric data structure. We adopt the method of Curless and Levoy [1996], and encode surfaces implicitly as a signed distance field (SDF). Our pipeline is shown in Figure 2 and is based on the standard KinectFusion system [Izadi et al. 2011; Newcombe et al. 2011b] but evolves to accommodate this new scalable structure. We briefly review this volumetric method to guide the design of our data structure.

For now, we assume a regular dense 3D grid. The input to our system is a sequence of noisy depth images Z_i. We initialize the camera to the origin, which is also the center of the virtual volume's front face. For each frame, we incrementally update the volume by *integrating* (or *fusing*) surface observations into the stored SDF, adding new data into empty regions or denoising existing data. Next, we *raycast* the volume (using the current camera pose estimate), marching individual rays through the grid, to find sign changes in the SDF (the zero-crossing) and extract surface points and normals. Finally, when the next depth map arrives, we use the point-plane variant of the ICP algorithm [Chen and Medioni 1992] to estimate the new camera pose by aligning the input depth measurements with the extracted oriented points.

Integration Consider a single depth sample $z = Z_0(x, y)$. Without noise, the sample locally approximates the surface as a plane. Each voxel in the grid whose center projects to the same (x, y) location as this depth sample is part of the sample's *footprint*. At each voxel we store the distance from its center to the plane (with positive values in front).

However, in the presence of noise, which for simplicity we model as a Gaussian whose variance depends on depth $N(z, \sigma(z))$ [Chang et al. 1994; Nguyen et al. 2012], the true surface is somewhere in the vicinity of z. In the case where the camera is stationary, we can incrementally obtain a least squares estimate of the SDF by weighted averaging. Augment the SDF (D) with a weight (W) and initialize each voxel in the grid to $D = 0, W = 0$. For each incoming sample z_{i+1}, update the grid with the rule $D_{i+1} = (D_i W_i + d_{i+1})/(W_i + 1), W_{i+1} = W_i + 1$, where d_{i+1} is the signed distance from the voxel center to the incoming sample. In the case where the camera moves, it is clear that the current SDF D_i becomes inconsistent with the new depth frame Z_{i+1}. It is impossible to obtain a full SDF using only incremental accumulation.

Truncation and free space carving Curless and Levoy observed that to handle a moving sensor, permit thin surfaces to be reconstructed, and reduce computation, the SDF is only meaningful near the surface and that distant voxels can be ignored. Therefore, they use a *truncated* SDF (TSDF) region in the vicinity of the observation ($2\sigma(z)$ in our case). Notice however that an observed depth sample yields more information than just near the surface: it indicates that the entire ray up to the surface is unoccluded. Curless and Levoy

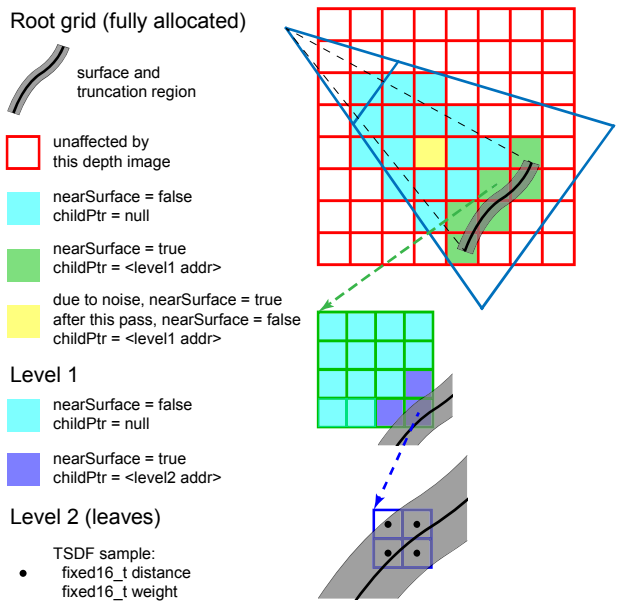

Figure 3: *Logical view of hierarchical data structure.*

called this *free space carving* and explicitly mark these voxels as "free space". Therefore voxels exhibit three potential states: observed free space, unobserved or near surface.

3.1 Data structure design

Our goal is to design a data structure that compactly represents a volumetric TSDF and permits efficient *integration* (or *fusion*) and *raycasting* operations.

Exploiting sparsity The vast majority of the world is empty, which is reflected by the fact that in a regular grid, most of the voxels are marked as free space or unobserved. These voxels are good candidates for compression. Curless and Levoy highlight this sparsity in the data, and dynamically compress regions outside of the truncation using run-length encoding (RLE). RLE does not naturally map to our scenarios, as we require fast addition or removal of surfaces, which would result in large overheads managing and traversing runs on the GPU. Instead we design a hierarchical data structure: near surfaces, we densely allocate voxels and store a high resolution TSDF, integrating as before. However, completely free space or unobserved regions is represented with coarser nodes.

Hierarchical representations Hierarchical data structures have a long history in computer graphics and we discuss our design choices here. They can be roughly divided into bounding volume hierarchies (BVHs), which cluster geometry and are used in traditional polygon raytracing techniques, and spatial subdivisions methods, which partition space. We rule out BVHs since our algorithm needs to store dynamic SDFs and cannot afford to rebuild the hierarchy every frame. While a number of spatial subdivision strategies are available, we can disregard anisotropic structures such as kD-trees or BSP-trees due to the fact that our moving depth camera reorients as the user moves. Therefore, we choose a regular spatial subdivision.

Regular spatial subdivision still offers a number of choices. First, we must choose a refinement strategy: at what point do we split a node? At one extreme, with no refinement, we have a dense regular grid, which scales as $O(n^3)$ in memory. At the other extreme, with full dyadic refinement and data stored only in leaves, we have a complete octree, which is space efficient, but results in a very deep hierarchy that is difficult to update and traverse on a GPU. In between, we can choose different branching factors at each level, resulting in a hierarchical grid (or N^3-tree). A final option is *adaptive* refinement: represent the SDF at multiple resolutions by storing the value at different levels of tree, splitting a node when it can no longer summarize the variation within [Frisken et al. 2000].

We experimented with most of these options and show in Section 6 that a 3-4 level N^3-tree with regular grids at nodes, without adaptive refinement, yields the best memory/speed tradeoff. Although adaptive refinement works well for synthetic data [Frisken et al. 2000], the highly anisotropic nature of our depth sample footprints makes adaptive refinement challenging. As verified by [Chang et al. 1994; Nguyen et al. 2012], the z uncertainty of a depth sample grows as $O(z^2)$. Due to this elongated footprint, the standard subdivision rule that splits a node until it is smaller than the sample footprint in all three dimensions essentially causes full refinement. Similarly, for rules that subdivide based on content, with a shallow tree optimized for updating and raycasting, interior nodes never project to a small enough screen area to capture sufficient detail.

Overview of data structure Figure 3 shows a logical view of our data structure. The example hierarchy consists of three levels: the root (in red) is a fully allocated grid and provides a coarse subdivision of the physical volume. According to our noise model [Nguyen et al. 2012], the surface lies within the truncation region (in gray), and any intersecting voxels will need to be refined. Refinement proceeds recursively until we reach the leaf level (in blue), where each node is a small regular grid. We sweep through the voxels of the leaf grid and update the distances and weights accordingly. Notice how the majority of root voxels are free space (in cyan), and do not require refinement.

A common scenario that occurs due to noise or moving objects is that a previously refined voxel (in yellow) is subsequently observed as free space. Our strategy for dealing with these cases is to store metadata at interior nodes, similar to a hierarchical z-buffer [Greene et al. 1993]. In addition to a pointer to the node's children, we store a **nearSurface** flag, indicating whether any of its children are potentially near a zero crossing, and a **minWeight** value, which is the minimum of their weights. This metadata is updated during integration with a summarization step, which propagates data up the tree. The **nearSurface** flag lets the raycaster skip entire subtrees, while **minWeight** optionally lets us "freeze" nodes (e.g., the voxel in yellow) as free space when it is above some threshold and garbage collect its children.

We extend the *binary* free space carving of Curless of Levoy to more effectively fuse depth maps from the Kinect, which tend to be noisy near silhouettes, causing flickering near object edges in the SDF. We instead clamp the incoming SDF value d_{i+1} to its maximum value within the truncation region and perform weighted averaging instead. This results in clean, rounded silhouette edges. Finally, in order to accommodate depth uncertainty, we adapt the truncation size of the SDF according to depth [Chang et al. 1994; Nguyen et al. 2012].

4 GPU implementation

We implement our 3D reconstruction pipeline on the GPU using CUDA. This section details how the hierarchy is laid out in graphics memory and traversed in parallel for integration and raycasting.

Memory layout We store our hierarchical grid in GPU memory as a sparse pointer structure, which is visualized in Figure 4. The

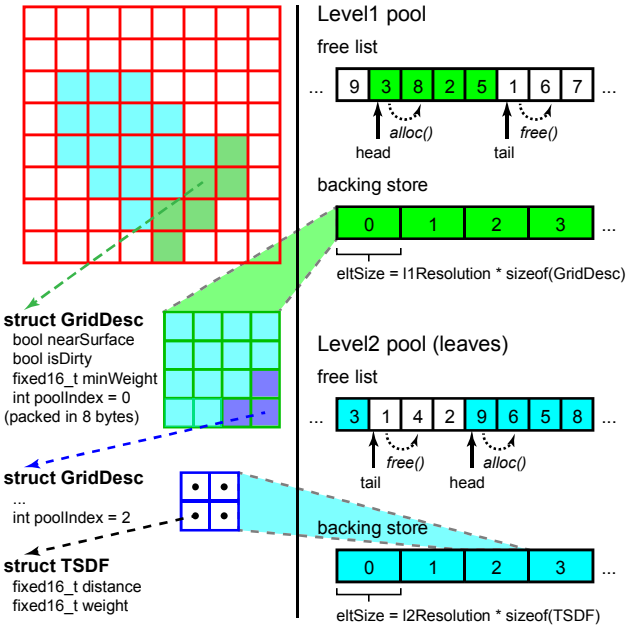

Figure 4: *Memory view of hierarchy.*

root level grid is always fully allocated and stored as a dense 3D array of **GridDesc** records, initialized to **null**. For each level of the hierarchy, we preallocate a fixed-sized memory pool, consisting of a free list and a backing store. The free list is simply a queue of block indices, initialized to **full** (the list $[0, 1, \ldots n]$). The backing store is an array of n fixed-sized blocks, where each block has size equal to an entire grid at that level. During integration, when a grid needs to be allocated, a free block is dequeued from the free list using an atomic operation, assigned to the **poolIndex** field, and marked **isDirty** (since the memory block can contain anything). Similarly, when we free a block during garbage collection, we atomically enqueue its index back onto the free list. Note that since grid resolution and element sizes can differ at each level in our flexible data structure, we require a separate pool for each level. In particular, a **GridDesc** occupies 2 bytes, whereas **TSDF** requires 4. We defer pool size selection to Section 6.

Algorithm 1 Volumetric fusion/integration

```
1:  for each voxel v do in parallel
2:      if intersect(v, frustum) then
3:          bbox2D ← boundingBox2D(project(v))
4:          for all pixels p ∈ bbox2D do in parallel
5:              z ← depthMap[p]
6:              overlaps ← intersect(truncationRegion(z,σ(z)),v)
7:          anyOverlaps ← parallelReduce(overlaps)
8:          if threadId = 0 then
9:              desc ← grid[v]
10:             descend ← (anyOverlaps or hasChildren(desc))
11:             if descend then
12:                 enqueue(jobQueue,v)
13:                 if !hasChildren(desc) then
14:                     desc.poolIndex ← alloc()
15:                     desc.isDirty ← true
```

Integration Given a depth map, we *integrate* it into our hierarchy in breadth-first order, as illustrated in Figure 3 and described with pseudocode in Algorithm 1. For the interior levels of the tree (including the root), we *conservatively* rasterize the footprint of the depth map into successively finer voxel grids with recursion mediated by atomic queues. The root grid does not require an input queue: the voxel indices can be determined by conservatively clipping the camera frustum. Root voxels are projected to large hexagons within the depth map, assigned 1 thread-block per voxel, and rasterized using many threads.

The algorithm for interior levels is nearly the same as for the root. Since voxels now project to smaller hexagons, we instead assign one thread-block per *grid*, with one thread per voxel. The grid descriptor is retrieved from the input queue and if flagged **isDirty**, the threads cooperatively clear it in parallel. Each thread then proceeds with the rest of the algorithm (Algorithm 1, line 5 onwards), except hexagon rasterization is done by a single thread. We stress that careful conservative rasterization and intersection tests are necessary to achieve seamless hole-free results. At interior nodes, voxels may be large and simply projecting the voxel center (as in [Zeng et al. 2013]) can easily miss depth samples. Finally, at the leaf level, we can assume that voxels are smaller than a pixel, in which case we project its center, compute a TSDF value, and update the voxel.

Summarization To ensure that raycasting skips large portions of free space away from geometry, we retain the job queues from integration to perform *summarization* in parallel. As each leaf grid is swept by parallel threads, if any SDF values are near surface geometry (defined by having d with magnitude less than the diagonal of the leaf voxel), we perform a parallel reduction to set its grid descriptor to **nearSurface**. Similarly, we use parallel reduction to find the minimum weight in a leaf grid. Summarization proceeds up the tree using the existing job queues until we reach the root. We guarantee that if an interior voxel is **not nearSurface**, it can be skipped over even though it may contain children. Conversely, a voxel that is **nearSurface** must have children and must be traversed. We optionally use the **minWeight** field as a heuristic for *garbage collection*: if an interior voxel has a sufficiently high **minWeight** and is **not nearSurface**, then it is unlikely to be in the future and can be "frozen" as free space. We free the node's children in the next integration pass and skip integration in the future.

Raycasting We use a hierarchical variant of the DDA algorithm [Amanatides and Woo 1987] to raycast our data structure on the GPU, conservatively rasterizing a line on a hierarchical grid. We maintain as state the previous distance along the ray (t_p), the previous and current SDF values (d_p and d_c respectively), and a stack of voxel indices down the hierarchy. We initialize DDA by setting $t_p = 0$ (at the eye) and traversing the tree to retrieve d_p. At each iteration of DDA, we step to the next voxel at the current level. If we are at an interior node and it is marked **nearSurface**, then find the closest voxel at the next level and push it onto the stack (otherwise, do nothing). If we are at a leaf, then test whether there is a zero crossing: $d_p > 0$ and $d_c < 0$. If so, the surface is at $t_z = t_p + \frac{d_p}{d_p - d_c}$ along the ray. Otherwise, set $d_p = d_c$ and continue. Finally, if we stepped past the bounds of the current grid, pop the stack. Note that with DDA, the SDF is indexed directly without any filtering.

To compute the surface normal, needed for shading and ICP camera tracking, we estimate the gradient of the SDF at the zero crossing using first order finite differences and trilinear interpolation. Since we use shallow trees with relatively large branching factors, the vast majority of samples lie in the same leaf grid. We exploit this by caching and reusing the tree traversal from the first sample, improving performance by a factor of 2. We also experimented with persistent threads and found performance between software and hardware scheduling (with 8×8 thread-blocks) roughly equivalent.

5 Moving volumes and streaming

Our hierarchical representation enables interactive reconstruction of relatively large volumes (at 1024^3 resolution, $(4m)^3$ with $(4mm)^3$ voxels, $(8m)^3$ with $(8mm)^3$ voxels). To further scale to *unbounded* physical dimensions, we take inspiration from [Crassin et al. 2009; Whelan et al. 2012] and decouple the physical volume from the working set.

To decouple the physical position of the volume from voxel indices in GPU memory, it helps to define a few terms. When we specify a hierarchy, we choose the resolution (the number of voxels) at each level, and a leaf level voxel size (in meters). These parameters multiply to determine the physical size of a root voxel in meters. We quantize the world coordinate system into units of root voxels, which serve as unique keys indexing subtrees of the hierarchy (see Figure 5). We define the *working set* to be the set of fixed 3D array indices in GPU memory equal to the root grid resolution. Finally, the *active region* is a cubical subset of the world coordinate system (in meters) that is centered on the camera's view frustum, but whose origin is quantized to a root voxel in the world. To guarantee zero contention, we enforce that the active region's effective resolution be one root voxel less than that of the working set along each axis. This lets us map voxels of the active region to indices of the working set using modular arithmetic.

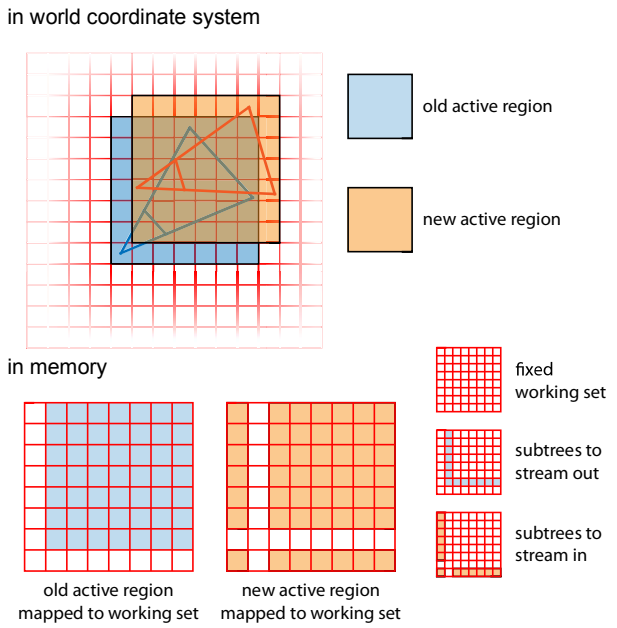

Figure 5: *Illustration of bidirectional moving volume streaming. When the camera and active region moves, we determine which subtrees need to be streamed to host, and which need to be cleared or streamed back in.*

Streaming from GPU to host is similar to integration and requires two breadth-first traversals of the hierarchy. Given a list of working set indices to stream out on the host, we copy them into a GPU queue and perform a tree traversal to determine how much space is needed for each subtree (using parallel reduction to compute the sum). We perform a parallel prefix scan to compute offsets into a linear buffer where each subtree will be stored. Finally, we perform one more tree traversal to write each voxel into the linear buffer, replacing **poolIndex** with a the byte offset from the beginning of each subtree. This operation essentially converts a forest from breadth-first storage to depth-first storage. We copy the linear buffer and list of offsets to the host then store each serialized subtree in a dictionary. Streaming from host to GPU is analogous.

Compared to Kintinuous [Whelan et al. 2012], which converts the outgoing region to a triangle mesh, and Moving Volume KinectFusion [Roth and Vona 2012], which resamples the overlapping region, our approach is completely lossless. Due to the sparse nature of our representation, we can stream data between the GPU and host without noticeable performance hits. This lets the user revisit already scanned areas without any meshing or interpolation artifacts.

6 Results

In this section, we first describe the design of our capture setup. We then present the results of our experiments on how data structure parameters affect performance and memory consumption, and compare reconstruction quality with previous work. We purposefully implemented our hierarchy as a generic structure to facilitate experimentation with branching factors and tree depths. We pinpoint the optimal configuration for current hardware.

6.1 Capture setup

In order to scan large-scale scenes, we design a semi-mobile client-server architecture. The user interacts with the mobile client in the field, which transmits depth maps to the server for reconstruction, and displays raycasted images of the 3D model in real-time. Client hardware is comprised of a lightweight laptop, battery pack, and Kinect camera with attached touchscreen display. The user holds the Kinect and uses the touchscreen to control and visualize the capture process. The laptop and battery are carried in a backpack. The server performs the actual reconstruction and consists of a desktop workstation with a powerful GPU, powered with AC. The client and server communicate over a wireless 802.11n network, which theoretically provides 300 Mbps of bandwidth (2×2 MIMO). By transmitting only uncompressed depth frames (640×480 at 30 Hz and 16 bpp), we need 140 Mbps of bandwidth. In our experiments, we can achieve on average about 24 Hz, depending on wireless conditions. The emerging 802.11ac wireless standard is expected to provide over 1.6 Gbps of bandwidth, which will be more than sufficient for both color and depth.

6.2 Performance and memory consumption

We conducted all our experiments on the server, a desktop PC with an Intel Xeon W3690 CPU at 3.46 GHz, 6 GB of RAM, and a single GeForce GTX Titan GPU. We consistently use three 500 frame sequences captured with a Kinect depth camera: DESK, a close-range (2m) sequence with few holes in the depth data, BOOKSHOP, a medium range (4m) sequence, and CYCLES, a challenging outdoor sequence with 6.5m range, complex geometry, high levels of noise and missing data in the depth maps. We measure GPU kernel execution time and memory consumption at 1024^3 and 2048^3 resolution under a variety of tree configurations. All configurations and voxel dimensions are cubic: a 1024^3 hierarchy that is 16^3 at the root, 8^3 at level 1 and 8^3 at the leaf is denoted $16, 8, 8$. Similarly, "4mm voxels" denotes a tree whose voxels in the leaf grid are cubes measuring 4mm on each side.

Figure 6 plots the average combined integration and raycasting execution time for a 1024^3 grid as a function of hierarchy configuration up to 5 levels. Note performance peaks at 3 levels and significantly decreases beyond 4 levels. This is due to two factors. Integration is performed breadth-first, which requires synchronization via queues and offsets savings from a tighter bound. Raycasting is performed depth-first, which requires stack space linear in the number of levels.

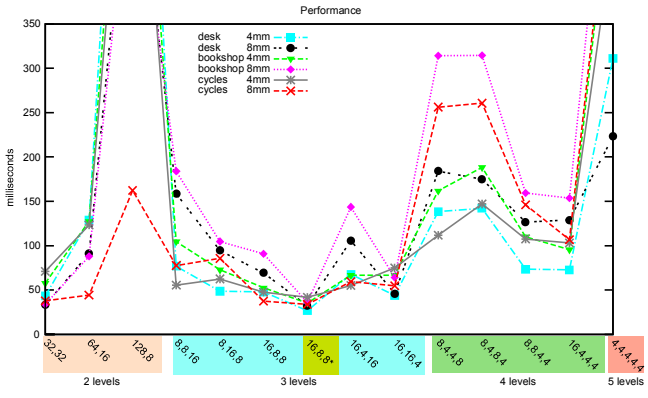

Figure 6: *Average combined integration and raycasting runtime as a function of hierarchy configuration. Our optimized implementation is marked* $16, 8, 8^*$ *and achieves 24-37 frames per second.*

A deep tree also results in additional GPU warp divergence. Interestingly, raycasting performance does not dramatically change between 4mm and 8mm voxels at the same resolution: DDA simply steps through the grid regardless of physical size. Integration is however much slower with larger voxels because we perform full hexagon rasterization at the upper levels of the hierarchy. A detailed table with additional configurations and a breakdown of timings between integration and raycasting is in the supplemental material.

Note that these performance numbers reflect the more flexible, but unoptimized version of our data structure. After deciding on the most efficient configuration $(16, 8, 8)$, we optimized our implementation further. This included fixing the thread-block configuration, using texture units once the number of levels was selected and caching shared tree traversals for normal estimation across grid boundaries. The optimized system runs at between 24-37 Hz across the 3 sequences shown for a 1024^3 volume at 4mm precision, and 20-30 Hz for 2048^3. The rest of the pipeline including depth map preprocessing, pose estimation and streaming takes 5-6ms, with streaming taking on average 2ms.

In Figure 7, we plot memory consumption vs. hierarchy configuration for a 1024^3 volume. As expected, smaller leaf grids consume less memory as the tighter fitting nodes skip more free space. We see diminishing returns after 4 levels. Notice how there is little variation between the different branching factors at 4 levels given the tiny leaf sizes. At 5 levels and beyond, memory consumption levels off as pointer overhead dominates. In all cases, interior levels of the tree are extremely sparsely allocated, with leaf grids occupying the vast majority of memory. Therefore, we distribute 128 MB among all interior level pools and allocate the remainder of our memory budget (typically 512 MB-1 GB) to the leaf level pool. Not plotted is a 2048^3 volume (32,8,8) with 4mm voxels, which consumes 245 MB of memory in the worst case with CYCLES. Our results compare favorably to 512 MB and 1 GB for 512^3 and 640^3 regular grids, respectively, and to previous work that use octrees [Zeng et al. 2013]. Figure 7 also shows the effect of voxel size: clearly, larger voxels require less memory given the same input range and physical volume. For these experiments, we observe a "sweet spot" at 3 levels at $(16, 8, 8)$, for which we optimized our algorithms.

6.3 Scalability and reconstruction quality

Figure 9 and our companion video presents large-scale fine-detail real-time reconstructions using our data structure. In these captures, the user holds a semi-mobile Kinect camera, and moves around to

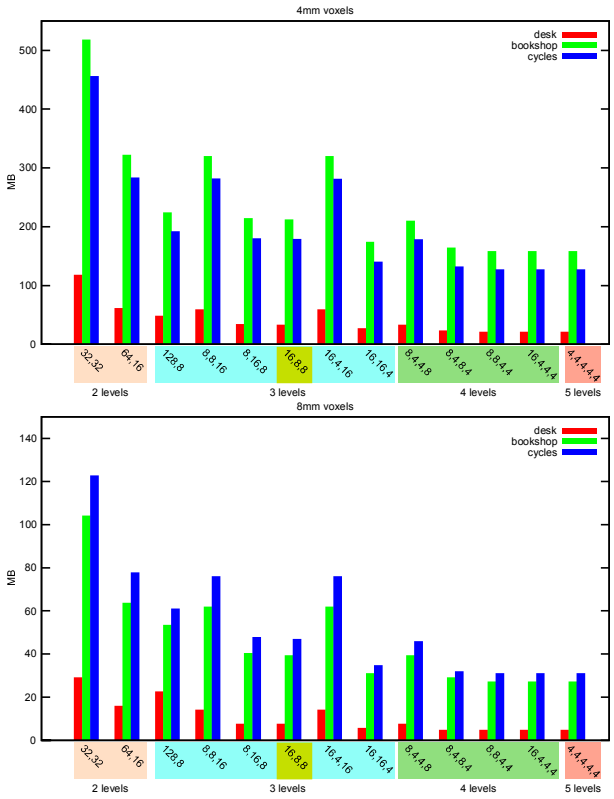

Figure 7: *Memory consumption as a function of hierarchy configuration for three representative scenes with 4mm (top) and 8mm voxels (bottom). The total resolution is* 1024^3.

continuously capture and reconstruct the surrounding scene. The extracted 3D model can be imported into CAD applications or games, or leveraged in real-time AR or robot guidance applications. Direct feedback is also important in ensuring the user fully captures the scene during scanning. Our reconstructed datasets feature both indoor and outdoor scenes. Although Kinect is an active sensor, we have found that the sensor can work at moderately close range under a variety of lighting conditions.

In the BOOKSHOP example, the user interactively reconstructs a 3D model of 3 floors of a book store in less than 6 minutes. The user can walk up and down stairs, capturing fine geometry of books, tables, and stairs. The use of both dense ICP and frame-to-model alignment from KinectFusion allows some level of robustness when people move in front of the sensor (a common scenario in real-world capture). The CAR sequence was captured in $\sim$1 minute and the PLANE in under 5 minutes (Figures 1, 9). COURTYARD shows a very large $(16m \times 12m \times 6m)$ reconstruction captured live with our method. In the PASSAGE sequence, the user scans a variety of shops down a passageway. Notice that the global model appears twisted in places due to tracking errors. Drift mitigation using techniques such as global bundle adjustment and loop closure are interesting areas of future work but not directly addressed currently.

Qualitative comparison Figure 8 and the companion video compares reconstruction quality between a moving regular grid [Whelan et al. 2012; Roth and Vona 2012] with our moving hierarchy at equal voxel dimension and equal physical extent. With 4mm voxels, a moving regular grid only spans $(2m)^3$, resulting in a clipped volume whereas the hierarchy makes full use of the sensor data. A

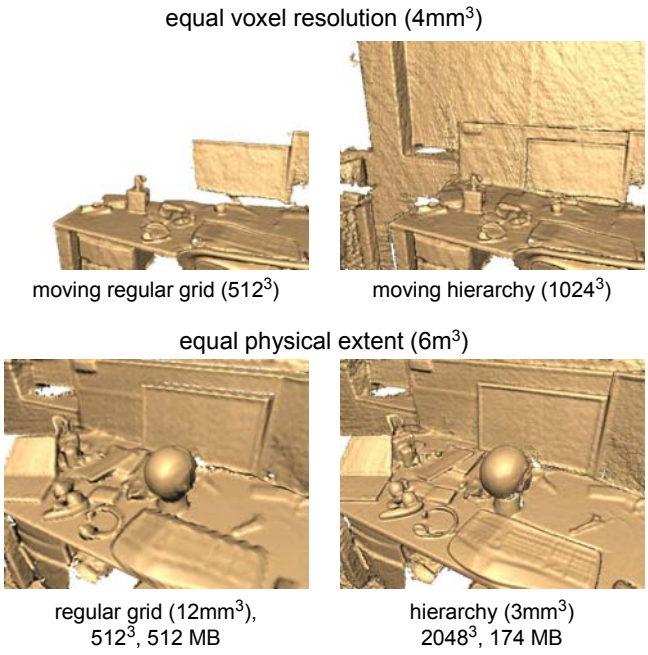

Figure 8: *Quality comparison. Top: at equal voxel resolution, our hierarchy permits a much larger active region compared to a moving regular grid. Bottom: at equal physical extent, our hierarchy captures significantly more detail in a fraction of the memory.*

small active region may suffice for indoor scenes, but for larger outdoor scenes, a large volume is necessary for reliable dense frame-to-model tracking. In another comparison, at the same physical extent of $(6m)^3$ necessary to capture an office scene, a regular grid [Newcombe et al. 2011b; Izadi et al. 2011] has 12mm voxels which severely oversmoothes fine details, whereas the hierarchy faithfully reconstructs details with 3mm precision in a fraction of the memory, while remaining interactive ($\sim$24 Hz).

6.4 Limitations and future work

As mentioned, the focus of this paper is our scalable data structure and addressing issues of camera drift remains future work. In our experiments, drift is highly scene dependent. "Outside-in" scans such as CAR have minimal error and we are able to close the loop. Tracking error is more noticeable in "inside-out" scans and is especially evident in long "forward-only" paths such as PASSAGE. A related issue is relocalization when camera tracking fails. We implemented a simple relocalizer that maintains a history of keyframes consisting of a pose, a depth map, and a normal map. We create a new keyframe whenever the camera moves past the basin of convergence for ICP (empirically set to 15cm translation and 10° rotation). At runtime, if ICP fails to converge, we exhaustively search the history. Robustly handling drift, loop closure, and relocalization are longstanding problems in the SLAM community and are clear areas for future work. Other areas of future work include addressing limitations in mobility (our current system relies on having a powerful server within wireless range) and exploring the use of our data structure with longer range sensors that have challenging noise characteristics, such as passive stereo cameras.

7 Summary

We demonstrate the new capability of real-time volumetric reconstruction *at scale*. We extend the method of Curless and Levoy to interactively acquire both large- and fine-scale reconstructions. To support this level of scalability, we design a fast, compact volumetric data structure for commodity graphics hardware. Our results show surface reconstructions of a variety of indoor and outdoor scenes, without trading scale, quality or speed.

8 Acknowledgements

We thank Mat Cook, Andreas Georgiou, Steven Johnston, James Scott, Kenji Takeda and Nicolas Villar for help in hardware setup and data capture. We are grateful to Cambridge University, The Imperial War Museum Duxford, Kelsey Kerridge Sports Centre, and The Mill Pub for filming access. Thanks go to Timo Aila, Tero Karras, and Jaakko Lehtinen for GPU programming wizardry; Christopher Zach, Andrew Fitzgibbon, Sven Forstmann, Otmar Hilliges, and Iason Oikonomidis for insightful discussions, and Emily Whiting for providing the voiceover.

References

ALLIEZ, P., COHEN-STEINER, D., TONG, Y., AND DESBRUN, M. 2007. Voronoi-based variational reconstruction of unoriented point sets. In *Proc. SGP 2007*, vol. 257, Eurographics, 39–48.

AMANATIDES, J., AND WOO, A. 1987. A fast voxel traversal algorithm for ray tracing. In *Proc. Eurographics*, vol. 87, 3–10.

BESL, P., AND MCKAY, N. 1992. A method for registration of 3-D shapes. *IEEE Trans. PAMI 14*, 2, 239–256.

BOLITHO, M., KAZHDAN, M., BURNS, R., AND HOPPE, H. 2007. Multilevel streaming for out-of-core surface reconstruction. In *Proc. SGP 2007*, vol. 257, Eurographics, 69–78.

CHANG, C., CHATTERJEE, S., AND KUBE, P. R. 1994. A quantization error analysis for convergent stereo. In *Proc. ICIP 94*, vol. 2, IEEE, 735–739.

CHEN, Y., AND MEDIONI, G. 1992. Object modelling by registration of multiple range images. *Image and vision computing 10*, 3, 145–155.

CHIEN, C., SIM, Y., AND AGGARWAL, J. 1988. Generation of volume/surface octree from range data. In *Proc. CVPR 98*, IEEE, 254–260.

CONNOLLY, C. 1984. Cumulative generation of octree models from range data. In *Proc. ICRA 84*, vol. 1, IEEE, 25–32.

CRASSIN, C., NEYRET, F., LEFEBVRE, S., AND EISEMANN, E. 2009. GigaVoxels: Ray-guided streaming for efficient and detailed voxel rendering. In *Proc. I3D 2009*, ACM, 15–22.

CUI, Y., SCHUON, S., CHAN, D., THRUN, S., AND THEOBALT, C. 2010. 3D shape scanning with a time-of-flight camera. In *Proc. CVPR 2010*, IEEE, 1173–1180.

CURLESS, B., AND LEVOY, M. 1996. A volumetric method for building complex models from range images. In *Proceedings of SIGGRAPH 96*, Annual Conference Series, 303–312.

FRISKEN, S., PERRY, R., ROCKWOOD, A., AND JONES, T. 2000. Adaptively sampled distance fields: a general representation of shape for computer graphics. In *Proceedings of SIGGRAPH 2000*, Annual Conference Series, 249–254.

FUHRMANN, S., AND GOESELE, M. 2011. Fusion of depth maps with multiple scales. *ACM Trans. Graph. 30*, 6 (December), 148:1–148:8.

GREENE, N., KASS, M., MILLER, G., ET AL. 1993. Hierarchical z-buffer visibility. In *Proceedings of SIGGRAPH 93*, Annual Conference Series, 231–238.

HENRY, P., KRAININ, M., HERBST, E., REN, X., AND FOX, D. 2010. RGB-D mapping: Using depth cameras for dense 3D modeling of indoor environments. In *Proc. ISER*, vol. 20, 22–25.

HIGUCHI, K., HEBERT, M., AND IKEUCHI, K. 1995. Building 3-d models from unregistered range images. *Graphical models and image processing 57*, 4, 315–333.

HILTON, A., STODDART, A., ILLINGWORTH, J., AND WINDEATT, T. 1996. Reliable surface reconstruction from multiple range images. *Computer Vision (Proc. ECCV 96)*, 117–126.

HILTON, A., STODDART, A. J., ILLINGWORTH, J., AND WINDEATT, T. 1998. Implicit surface-based geometric fusion. *Computer Vision and Image Understanding 69*, 3, 273–291.

HOPPE, H., DEROSE, T., DUCHAMP, T., MCDONALD, J., AND STUETZLE, W. 1992. Surface reconstruction from unorganized points. In *Computer Graphics*, vol. 26, 71–78.

HORNUNG, A., WURM, K. M., BENNEWITZ, M., STACHNISS, C., AND BURGARD, W. 2013. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots 34*, 3, 189–206.

IZADI, S., KIM, D., HILLIGES, O., MOLYNEAUX, D., NEWCOMBE, R., KOHLI, P., SHOTTON, J., HODGES, S., FREEMAN, D., DAVISON, A., ET AL. 2011. KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera. In *Proc. UIST 2011*, ACM, 559–568.

KAZHDAN, M., BOLITHO, M., AND HOPPE, H. 2006. Poisson surface reconstruction. In *Proc. SGP 2006*, vol. 256, Eurographics, 61–70.

LAINE, S., AND KARRAS, T. 2010. Efficient sparse voxel octrees. In *Proc. I3D 2010*, ACM, 55–63.

LEVOY, M., PULLI, K., CURLESS, B., RUSINKIEWICZ, S., KOLLER, D., PEREIRA, L., GINZTON, M., ANDERSON, S., DAVIS, J., GINSBERG, J., ET AL. 2000. The digital michelangelo project: 3D scanning of large statues. In *Proceedings of SIGGRAPH 2000*, Annual Conference Series, 131–144.

LORENSEN, W., AND CLINE, H. 1987. Marching cubes: A high resolution 3d surface construction algorithm. In *Computer Graphics*, vol. 21, 163–169.

MUSIALSKI, P., WONKA, P., ALIAGA, D., WIMMER, M., VAN GOOL, L., PURGATHOFER, W., MITRA, N., PAULY, M., WAND, M., CEYLAN, D., ET AL. 2012. A survey of urban reconstruction. In *Proc. Eurographics 2012 STARs*, Eurographics, 1–28.

NEWCOMBE, R., AND DAVISON, A. J. 2010. Live dense reconstruction with a single moving camera. In *Proc. CVPR 2010*, IEEE, 1498–1505.

NEWCOMBE, R., LOVEGROVE, S. J., AND DAVISON, A. J. 2011. DTAM: Dense tracking and mapping in real-time. In *Proc. ICCV 2011*, IEEE, 2320–2327.

NEWCOMBE, R. A., IZADI, S., HILLIGES, O., MOLYNEAUX, D., KIM, D., DAVISON, A. J., KOHLI, P., SHOTTON, J., HODGES, S., AND FITZGIBBON, A. 2011. KinectFusion: Real-time dense surface mapping and tracking. In *Proc. ISMAR*, IEEE, 127–136.

NGUYEN, C., IZADI, S., AND LOVELL, D. 2012. Modeling Kinect sensor noise for improved 3D reconstruction and tracking. In *Proc. 3DIMPVT 2012*, IEEE, 524–530.

OSHER, S., AND FEDKIW, R. P. 2003. *Level set methods and dynamic implicit surfaces*. Applied mathematical science. Springer, New York, N.Y.

PARKER, S., SHIRLEY, P., LIVNAT, Y., HANSEN, C., AND SLOAN, P.-P. 1998. Interactive ray tracing for isosurface rendering. In *Proc. Visualization 98*, IEEE, 233–238.

POLLEFEYS, M., VAN GOOL, L., VERGAUWEN, M., VERBIEST, F., CORNELIS, K., TOPS, J., AND KOCH, R. 2004. Visual modeling with a hand-held camera. *IJCV 2004 59*, 3, 207–232.

POLLEFEYS, M., NISTÉR, D., FRAHM, J., AKBARZADEH, A., MORDOHAI, P., CLIPP, B., ENGELS, C., GALLUP, D., KIM, S., MERRELL, P., ET AL. 2008. Detailed real-time urban 3D reconstruction from video. *IJCV 2008 78*, 2, 143–167.

POTMESIL, M. 1987. Generating octree models of 3d objects from their silhouettes in a sequence of images. *Computer Vision, Graphics, and Image Processing 40*, 1, 1–29.

ROTH, H., AND VONA, M. 2012. Moving volume KinectFusion. In *Proc. BMVC 2012*, BMVA Press, 112.1–112.11.

RUSINKIEWICZ, S., HALL-HOLT, O., AND LEVOY, M. 2002. Real-time 3D model acquisition. *ACM Trans. Graph. 21*, 3 (July), 438–446.

SEITZ, S., CURLESS, B., DIEBEL, J., SCHARSTEIN, D., AND SZELISKI, R. 2006. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *Proc. CVPR 2006*, vol. 1, IEEE, 519–528.

STÜCKLER, J., AND BEHNKE, S. 2012. Robust real-time registration of RGB-D images using multi-resolution surfel representations. In *Proc. ROBOTIK 2012*, VDE, 1–4.

SZELISKI, R. 1993. Rapid octree construction from image sequences. *CVGIP Image Understanding 58*, 23–23.

THRUN, S., BURGARD, W., FOX, D., ET AL. 2005. *Probabilistic Robotics*. MIT Press, Cambridge, MA.

TURK, G., AND LEVOY, M. 1994. Zippered polygon meshes from range images. In *Proceedings of SIGGRAPH 94*, Annual Conference Series, 311–318.

WEISE, T., WISMER, T., LEIBE, B., AND VAN GOOL, L. 2009. In-hand scanning with online loop closure. In *Proc. ICCV 2009 Workshops*, IEEE, 1630–1637.

WHEELER, M., SATO, Y., AND IKEUCHI, K. 1998. Consensus surfaces for modeling 3D objects from multiple range images. In *Proc. ICCV 98*, IEEE, 917–924.

WHELAN, T., KAESS, M., FALLON, M., JOHANNSSON, H., LEONARD, J., AND MCDONALD, J. 2012. Kintinuous: Spatially extended KinectFusion. In *Proc. RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*.

ZACH, C., POCK, T., AND BISCHOF, H. 2007. A globally optimal algorithm for robust TV-L1 range image integration. In *Proc. ICCV 2007*, IEEE, 1–8.

ZENG, M., ZHAO, F., ZHENG, J., AND LIU, X. 2013. Octree-based fusion for realtime 3d reconstruction. *Graphical Models 75*, 3 (May), 126–136.

ZHOU, K., GONG, M., HUANG, X., AND GUO, B. 2011. Data-parallel octrees for surface reconstruction. *IEEE Trans. Visualization and Computer Graphics 17*, 5, 669–681.

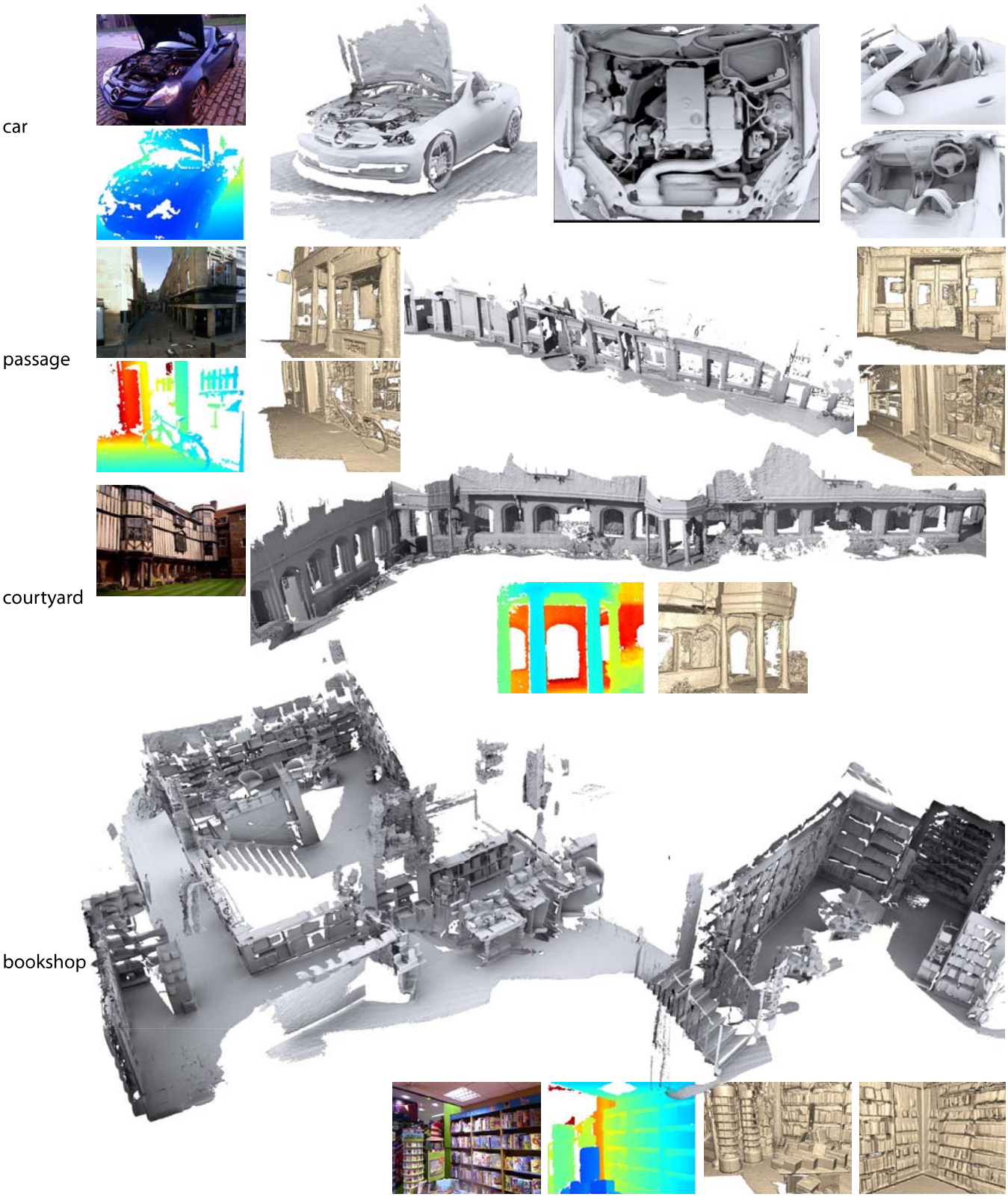

Figure 9: Reconstruction gallery *Insets show input color and depth frames from the Kinect camera. Smaller Phong shaded insets show live raycasted signed distance fields. Extracted meshes are rendered using global illumination and shown large.*