Volume 34 (2015), Number 2 pp. 1–13 COMPUTER GRAPHICS *forum*

Memory-Efficient Interactive Online Reconstruction from Depth Image Streams

F. Reichl†, J. Weiss‡, and R. Westermann§

Technische Universität München, Computer Graphics & Visualization Group

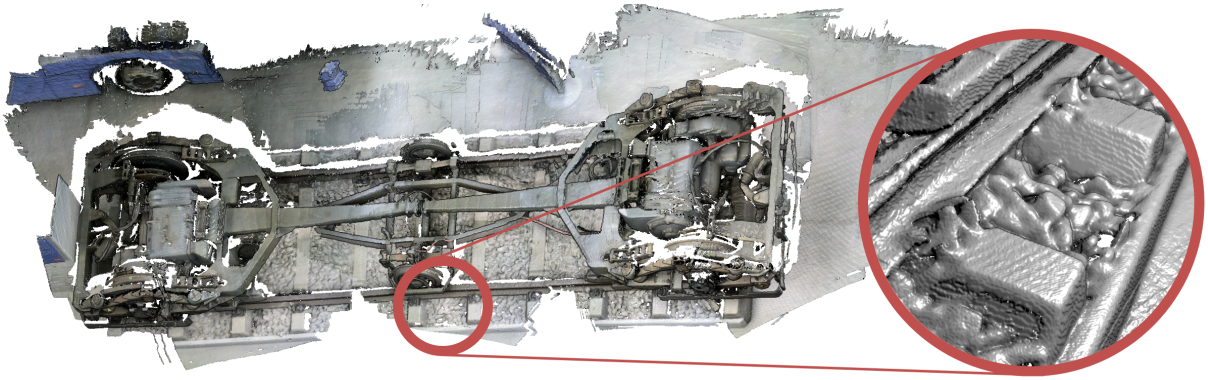


Figure 1: Online reconstruction of the underparts of a tramway. The scene is reconstructed at varying levels of resolutions of up to 1 mm (2.12 mm on average), requiring less than 10% of the memory that is required by alternative approaches.

Abstract

We describe how the pipeline for 3D online reconstruction using commodity depth and image scanning hardware can be made scalable for large spatial extents and high scanning resolutions. Our modified pipeline requires less than 10% of the memory that is required by previous approaches at similar speed and resolution. To achieve this we avoid storing a 3D distance field and weight map during online scene reconstruction. Instead, surface samples are binned into a high-resolution binary voxel grid. This grid is used in combination with caching and deferred processing of depth images to reconstruct the scene geometry. For pose estimation, GPU ray-casting is performed on the binary voxel grid. A one-to-one comparison to level-set ray-casting in a distance volume indicates slightly lower pose accuracy. To enable unlimited spatial extents and store acquired samples at the appropriate level of detail, we combine a hash map with a hierarchical tree representation.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Digitizing and Scanning

1. Introduction and Related Work

Recent works in real-time scene reconstruction using active depth sensors have shown effective scalability in the scanned spatial extents and resolutions by using adaptive space partitions on the GPU, either encoded hierarchically [CBI13] or via spatial hashing [NZIS13]. We see, on the other hand, that regardless the preferred encoding, scalability is increasingly limited by the amount of memory that has to be moved—via read and write operations—and stored during online reconstruction. This becomes especially important in scenarios where the scene data is so large that it cannot be stored in local GPU or even main memory, which is likely the case when mobile scanning devices are mounted to moving ve-

† florian.reichl@tum.de
‡ weissj@in.tum.de
§ westermann@tum.de

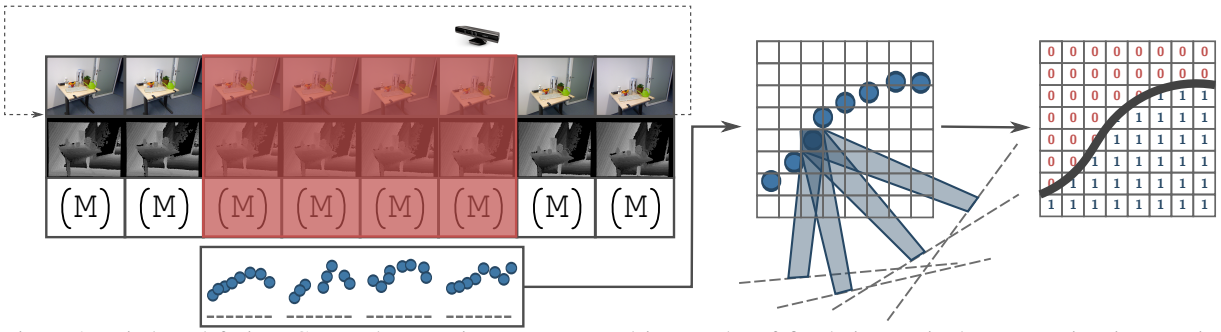


Figure 2: Windowed fusion: Captured sensor images are stored in a cache of fixed size. M is the pose estimation matrix associated to each image. Every $\frac{n}{2}$-th frame, n $(= 4)$ images are processed, by projecting visible voxels into all n images and computing averaged surface distance values. Subsequently, only a single bit indicating the sign of the final distance is stored in a binary voxel grid, and distance values are deleted.

hicles and sent out for 3D reconstruction, surveillance, and monitoring. In many real-world scenarios like disaster management and damage surveys, construction monitoring, or vegetation and plant screening, the use of such technologies will continue to increase in the near future.

Our research is motivated by the widening gap between data generation throughput and memory capacity, which will make large-scale and full-resolution online reconstruction prohibitively expensive. To overcome this limitation, the question needs to be addressed to what extent the amount of data required for scene reconstruction can be reduced. Most current online reconstruction approaches using active depth sensors like Microsoft Kinect adopt incremental volumetric fusion of depth samples into a truncated signed distance field on a regular multi-block grid, accompanied by a cumulative weight field to average out noise and outliers. This approach goes back to early work by Hilton [Hil96] and Curless and Levoy [CL96], which has been extended further and adapted to online reconstruction. For thorough introductions to the field, including a consistent collection of related work, let us refer to [IKH*11, CBI13, NZIS13]. While these approaches enable high quality scalable reconstruction, they come at the additional expense of storing at least two scalar values for every volumetric sample, resulting in 4-6 bytes per voxel depending on the used internal data representation. Related to these approaches are those working on the scanned depth maps, such as Merrell et al. [MAW*07], who improve the scanned maps by considering visibility constraints obtained from neighboring views.

To keep the memory requirements of volumetric fusion moderate, previous works acquired large scenes with a maximum voxel resolution of 4 to 10 mm, thereby undersampling the depth and color resolution of the sensor. When the sensor resolution is matched, the memory requirements increase considerably and GPU memory becomes occupied quickly. This problem will be aggravated in the future due to improvements in sensor hardware and the need to match the available scanning resolution to support pose estimation at higher accuracy [KSC13], especially on mobile devices equipped with lightweight sensors and limited onboard memory such as Google's project Tango.

To avoid the memory overhead of a regular grid even when it is constraint to a narrow band around the scanned surface, some previous works made use of explicit surface representations using point samples [RL00, WWLG09]. Keller et al. [KLL*13] used such a representation for online reconstruction from depth images, including noise reduction via geometric averaging and spatial as well as temporal outlier removal. Explicit surface representations, on the other hand, must store the 3D positions of acquired surface points, and require rendering the point set for reconstruction and pose estimation. Whelan et al. [WKJ*14] combine the memory-intensive volumetric fusion with real-time extraction of a compact surface representation for areas that move out of the field of view, keeping only a smaller volume of fixed size in GPU memory. Meilland et al. [MC13] propose a combination of a multi-key-frame approach with volumetric fusion and GPU surface triangulation. Fuhrmann et al. [FG11] investigate the use of a hierarchical distance field to incorporate the varying spatial resolution of depth samples captured under a perspective projection, targeting high-quality offline reconstruction.

Our work builds upon previous approaches and aims at reducing the memory requirements to enable scalability in both the scanned spatial extents and scanning resolutions. The basic idea is to perform the scene reconstruction via volumetric fusion, yet to encode the required weight and distance information in only two bits per voxel. The binary representation significantly reduces the memory requirements of an explicitly stored point-set surface or an implicit representation using a signed distance field. At the same time it allows using GPU voxel ray-casting for image-based pose estimation. However, to exploit the potential of a binary voxel representation for online scene reconstruction, a number of novel changes have to be incorporated into the existing fusion pipeline:

- We propose windowed fusion of image streams to avoid storing distances and weights. Deferred processing of small sets of cached depth and color images is performed in regular scanning intervals. We demonstrate memory savings of over 90% compared to previous approaches.
- We use the smoothed (depth) image of the ray-traced binary voxel surface for ICP-based pose estimation. This introduces a certain loss in pose accuracy, making our pipeline more prone to tracking drifts. A comparative study using multiple benchmark data sets analyzes accuracy and robustness compared to level-set rendering in a distance volume.
- We organize the scene data hierarchically in a hash map of adaptive octrees to support scenes of unlimited spatial extent without affecting voxel ray-casting performance.

Figure 2 illustrates the basic concept underlying windowed fusion. One important observation motivating our approach is that the deferred distance and weight computation using a fixed number of acquired depth and color images can bypass the need to store a volumetric distance and weight field. As a consequence, the memory requirements are low and constant over the entire reconstruction process. Furthermore, even at the resolution of the sensor hardware, the memory consumption of the binary voxel grid is considerably below that of a distance and/or weight volume.

Once the scene has been scanned, a smooth surface can be reconstructed from the binary representation using different approaches. One possibility is to replace the binary field with a continuous-valued field in which the zero-isosurface represents a smoothed version of the original binary surface. This can be achieved by solving a convex optimization problem as proposed by Lempitsky [Lem10]. Another approach is to extract a jagged surface mesh from the binary field via the Marching Cubes algorithm [LC87], and apply mesh-based smoothing afterwards. Due to the high computational complexity of the first approach, we chose to use mesh-based smoothing as presented by Taubin [Tau95], yet we make use of an efficient GPU implementation proposed by Moench et al. [MLK*13]. Since mesh extraction from a binary voxel field is significantly faster than from a signed-distance field, mesh smoothing can be performed in roughly the time it needs to extract the Marching Cubes surface from a signed-distance field.

In the remainder of the paper we first describe the windowed fusion strategy including a detailed description of how the binary scene representation is constructed. We then shed light on the particular GPU data structures we use, and we discuss implementation specific issues. Next, we demonstrate how to extend our data structure to handle sparsely allocated colors, out-of-core data streaming, and color compression. We conclude the paper with an analysis of the memory consumption and speed our approach can achieve, as well as a comparison to alternative approaches.

2. Volumetric Fusion

To reconstruct a surface from potentially uncertain measurements of surface points, volumetric fusion uses a truncated signed distance field (TSDF). At a discrete set of points, located on a regular grid restricted to the narrow band around the surface, the signed distances to the next measured surface points are computed, and these distances are averaged over a sequence of measurements [CL96]. This process is referred to as *integration*. In the distance field the surface is the zero level-set, and it can be reconstructed efficiently using surface fitting techniques like Marching Cubes [LC87] or GPU volume ray-casting [KW03].

When a depth image from a certain camera position is available, distance field computation is performed by sweeping through the voxel grid, projecting each voxel center into the depth image, and computing the distance between the voxel center and the acquired sample depth. A running average of all distances of a voxel is computed via a weight value which counts how often a voxel distance was updated. The weights compensate for the measuring inaccuracy in the captured depth data by averaging out these inaccuracies as well as outliers. The use of a truncation region determined by the noise characteristics of the sensor restricts the influence of each depth sample to its uncertainty region. It thus confines all calculations to a narrow band around the surface, reducing effectively the memory requirement and computational cost for updating distances and weights.

Our evaluations are based on a Microsoft Kinect 1, which captures depth and color data at a resolution of 640×480 pixels. It can be assumed that the uncertainty of each depth sample grows quadratically with increasing distance from the sensor [NIL12], whereas its size on the image plane grows linearly. To deal with a moving camera—which is typically the case—the current camera pose is estimated using the iterative closest point (ICP) method [BM92, CM92], and the new depth image is aligned with the rendered output of the fused surface.

In addition to the uncertainty in the measured depth values, due to perspective area foreshortening each depth and color sample in the sensor's xy-plane represents a scene sample of a size depending on the measured depth. The more distant the sample is from the camera, the larger is its extent in world space. Thus, using depth samples to compute the distance field on a fixed resolution grid—as it is typically done—usually over- or under-samples the sensor data, introducing loss of information or increasing memory usage. To avoid this, we locally adapt the resolution of the voxel grid into which the measured samples are fused.

3. Windowed Fusion

Windowed fusion of scanned images is motivated by the way in which scanning is usually performed: As the camera is moved through the scene, every voxel receives updates only

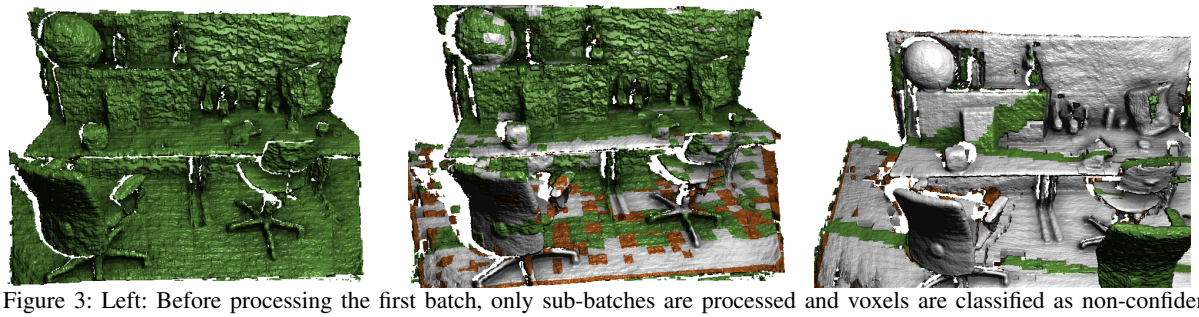


Figure 3: Left: Before processing the first batch, only sub-batches are processed and voxels are classified as non-confident (green). Middle: After fusion of the first batch of depth maps, many voxels are classified as confident (no color). Orange voxels have been processed, but are not classified as confident. Right: As scanning progresses, only few new bricks are processed from sub-batches (green). (Data set from Sturm et al. [SEE*12].)

over a small time interval. Then the scanned surface is of final quality and does not need further updates. We mimic this by restricting the calculations of *exact* distances and weights to a *batch* of few cached sensor images.

Windowed fusion in general averages fewer frames than volumetric fusion to determine the surface points. This means that sensor noise and outliers can be filtered out less effectively and can affect the point locations more significantly. We try to compensate for this by keeping track of the confidence in the current reconstruction; however, since the exact measurements are deleted once a batch has been processed, the final surface exhibits inferior regularization properties.

To make the fusion process scalable in the spatial scene extent, the entire domain is partitioned into a set of evenly sized sub-domains (*world chunks*) with a side length of 2 meters. Chunks are created on demand and stored in a GPU hashmap as proposed by Nießner et al. [NZIS13]. Each chunk stores a sparse octree that discretizes the corresponding part of the domain. A node of this octree consists of a voxel grid of size 8^3—which we will refer to as *bricks*—at an *adaptive* spatial resolution. This three-level structure provides a good trade-off between memory overhead, construction and rendering times, and it simplifies out-of-core data management. The details of the adaptive scene representation will be discussed in section 4.

3.1. Deferred Batch Processing

To maintain a compact scene representation, our goal is to avoid storing distances and weights. Therefore, we perform *deferred processing* of the depth and color images: All distance and weight calculations are deferred until a batch of n images has been cached in GPU memory. At this point, we first determine all bricks that were visible during m of the n scans, and we fuse the scans into these brick. We call m the *visibility threshold*. It helps to avoid using bricks which have appeared sporadically and would introduce noise to the reconstructed surface. For each of the determined bricks, every voxel is projected into all cached images at once by a single GPU thread, and the distances and weights for each of them are accumulated in-turn via volumetric fusion. Subsequently, only a single *sign bit* is stored for each voxel, indicating the sign of the computed distance of the voxel center to the surface.

To smooth out errors from tracking drifts, i.e., to ensure that these drifts do not reflect as hard jumps in the reconstructed surface, a number of depth maps are shared between subsequent batches. This is realized via a buffer of size $2n$ keeping the last two batches, and a window of size n that is shifted over this buffer in steps of $\frac{n}{2}$. This effectively doubles the time interval at which integration is performed and greatly improves the reconstruction quality. Along with every cached depth and color image we store the current pose estimation matrix (see Fig. 2).

3.2. Sub-Batch Processing

Deferred processing as described introduces the following problem: Because the scene representation is updated only in every $\frac{n}{2}$-th frame—with n often being as large as 60—updates of the *fused* surface are delayed. Since images of this surface are used as references in pose estimation, robust tracking is impeded unless the sensor is moved at low velocities. To overcome this limitation, in every acquisition frame an additional processing pass is performed using the most recent $\hat{n}$ images from the current batch (with $\hat{n} \ll n$). All bricks that have not received an update in the most recent deferred processing pass are integrated from these $\hat{n}$ images. These bricks are marked as *non-confident*, and their data will be replaced as soon as the next batch is processed. We will subsequently call the last $\hat{n}$ depth maps the *sub-batch*.

3.3. Surface Confidence

When voxel weights are not stored explicitly as in volumetric fusion, the uncertainty of a stored sign bit can no longer be accessed once a batch has been processed. Thus, voxels which are visible in only a small portion of the cur-

rently cached depth images may indicate an event—a positive or negative distance sign—with less confidence than an event was indicated at this location before. We counteract this by storing an additional *confidence bit* with each voxel. This bit is set if the voxel has accumulated a prescribed weight, called the *confidence threshold*, during processing. Each brick also stores the average estimated error of all its confident voxels, which is updated after cache integration. In successive batches, confident voxels can only be replaced if they are classified as confident again, and if the average error of the used depth samples is lower than the brick's average error calculated in the last integration pass.

Bricks that are processed in a sub-batch contain only non-confident voxels. All non-confident voxels are ignored for high-quality rendering and surface extraction, and they are only required for pose estimation during runtime. Figure 3 visualizes the classification for several frames. We also refer to the accompanying video to demonstrate the use of two batches and voxel confidences over multiple frames.

4. Data Structures and Algorithms

In this section, we discuss the implementation of all stages of the proposed reconstruction pipeline. The stages are implemented on the GPU using compute shaders. By using indirect dispatching whenever possible, CPU/GPU synchronization points are kept at a minimum. In this way, only a single synchronization is required after windowed fusion to retrieve the root nodes of created trees for rendering and data streaming.

4.1. Data structures

The three-level data structure is comprised of the following elements (see Fig. 4 for an illustration):

Chunks. At the first level, the domain is divided into evenly sized chunks. Each chunk stores its world-space position and a reference to the root of its associated octree (or NULL if the tree has not been generated yet):

```
struct Chunk {
  uint position;     // Z-Ordered
  uint rootPointer;
  uint offset;
}
```

Chunks are stored in a hashmap as proposed by Nießner et al. [NZIS13]. Positions are hashed to buckets in a buffer of fixed size, where each bucket comprises a fixed number of slots. In case of an overflow, the corresponding entry is placed in the next available slot and a list pointer of the last entry in the correct bucket is set to this slot. Given a careful selection of bucket size, overflows rarely ever happen and insertion and retrieval can be performed with minimal overhead. Furthermore, as we store comparably large chunks, the hash map access time is negligible.

Trees. All octrees are stored in a linear *node pool* and indexed using a single pointer to index 8 child nodes which do not necessarily contain data:

```
struct TreeNode {
    uint childPointer;
    uint dataPointer;
    uint positionAndLevel;
    float avgError;
}
```

Binary volume bricks. At the leaf nodes, `dataPointers` reference cubic voxel bricks of fixed size in a *brick pool*. It is worth noting that the leaf nodes are at different spatial resolutions, depending on the uncertainty of the samples they store. Each voxel stores two bits: the sign and the confidence bit. In practice, we use two separate pools to store the sign and the confidence bits, and we store them in 3D textures with 32 bits per texel. We will refer to groups of 32 bits as a single *bitblock*.

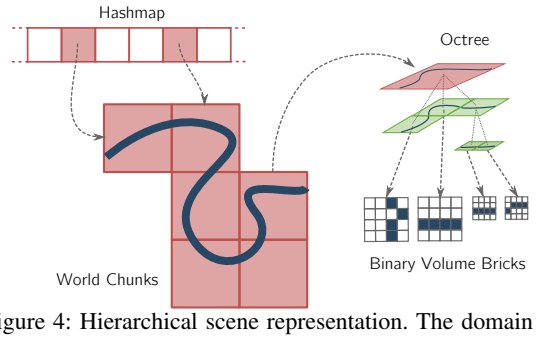


Figure 4: Hierarchical scene representation. The domain is partitioned into world chunks, each storing a sparse octree representation of a part of the scene. Leaf nodes store the binary scene encoding at different levels of detail.

4.2. Allocation

Each time a new frame is acquired by the sensor, allocation of all required data structures is performed instantly. Depth samples are inserted in parallel using one thread per pixel of the acquired image. Each sample is projected into 3D space using the currently estimated pose. We then determine the index of the chunk containing the sample and find the corresponding entry in the hash map, or we insert it and create the root node of a new tree. The appropriate tree level is determined based on the sample's extent in the image plane, and the chunk's tree is traversed top to bottom to reach the leaf node where the sample is to be inserted. All nodes along the way are created, and a slot in the brick pool is allocated for the leaf node if necessary.

To perform slot allocation and deallocation, we keep a list of available slot pointers in a separate GPU buffer. Upon allocation, an atomic counter is increased and the slot at

the previous counter position is used. For deallocation, the counter is decreased and the slot is added back into the list.

Special care has to be taken to assure that data allocation for a single pointer is performed by only one thread, even if multiple threads require allocation. This is done by storing a *lock flag* along with all data pointers. Whenever a thread needs to allocate data, brick nodes or world chunks, it first atomically locks the pointer and proceeds only if the lock succeeded. This may result in missing samples if a thread cannot acquire a lock. We address this problem by performing multiple insertion passes and storing all samples yet to be inserted in a temporary buffer, similar to the way proposed by Crassin et al. [CNS*11].

4.3. Integration

During scene capture, we store each depth and color map of the caches in two 2D texture arrays along with the corresponding transformation matrices. Prior to integration, we collect all bricks that have been completely visible in at least a predefined number of the cached frames. Bricks with a level difference of more than 1.0 compared to the level dictated by the current viewport are also discarded. This avoids destruction of regions scanned with low uncertainty by depth samples of high uncertainty. For each of these bricks, a single thread projects all voxels onto all of the depth maps (denoted with index i in the following). Computed distances are weighted by a weight W_i and accumulated in registers. For a voxel at position p, the distance $D(p)$ is

$$D(p) = \frac{\Psi(p)\Phi_0(p) + \sum_{i=0}^{n} W_i(p)D_i(p)}{\Psi(p) + \sum_{i=0}^{n} W_i(p)}, \qquad (1)$$

where D_i denotes the signed distance from the voxel to the surface in frame i, which is positive if the voxel is in front of the surface. With respect to the uncertainty σ of a depth sample, the weights W_i are set to

$$W_i = \begin{cases} 1 & \text{iff } 0 < D_i(p) < \sigma \\ \frac{\sigma + D_i(p)}{\sigma} & \text{iff } -\sigma < D_i(p) <= 0 \\ 0 & \text{otherwise} \end{cases} \qquad (2)$$

If the according voxel has already been flagged as confident, we determine the initial distance $\Phi_0(p)$ by a fast sweep through the volume of a few voxels along the three principal axis. If a sign change at a confident voxel is detected along any of these sweeps, the distance between the voxel centers is used as $\Phi_0(p)$ with a fixed weight $\Psi(p)$. After a batch has been processed, all empty bricks are collected and removed.

As integration needs to be performed in regular intervals of $\frac{n}{2}$ frames, processing all bricks at once will lead to peaks in computation times during scene capture. For reliable pose estimation, however, it is crucial to maintain a high and stable frame rate. We therefore distribute the integration of each batch over the subsequent $\frac{n}{2}$ frames, with $\frac{2}{n}$ of the visible bricks being integrated in each frame.

4.4. Rendering

The resulting surface is rendered using GPU front-to-back ray-casting. As proposed in [NZIS13], we first generate two z-Buffers containing the start and end points for every ray by rasterizing the bounding boxes of all currently allocated world chunks. This approach has the problem that all chunks between the start and end points have to be traversed, prohibiting efficient empty space skipping. In our three-level data structure, however, chunks are traversed at a rather coarse spatial subdivision and fine-granular empty-space skipping is performed at the octree level. We then perform DDA ray-marching through the world chunks along each view ray. In each chunk, if the corresponding entry exists in the hashmap, the associated octree is traversed with hierarchical DDA down to the finest existing leaf nodes, skipping all nodes that do not contain data.

Each leaf node, in turn, is again traversed using DDA until a change in the contained sign bits is detected. In this case the ray is intersected with a cube of the size of the voxel. Since this results in a jaggy surface profile including high frequent normal variations, we perform a deferred image-based surface smoothing pass similar to the approaches proposed for rendering smooth fluid surfaces from particle distributions [CS09, vdLGS09, RCSW14]. The smoothing operator works on the depth imprint of the rendered voxel representation. It performs screen-space surface smoothing using bilateral filtering of depth values [TM98] and reconstructs surface normals from the smoothed depth values via finite differences.

5. Color Storage

In principle, acquired RGB colors can be stored per voxel in addition to the binary values. However, this increases the memory consumption considerably, since memory is allocated also for non-occupied voxels. To reduce this consumption, a brick is further subdivided into blocks of $4 \times 4 \times 2$ voxels, and non-empty blocks are laid out in a linear color buffer. For each block, a brick stores a 32 bit pointer into the location of this block in the color buffer, or a void pointer if the block is empty. This increases the total number of bits per voxel to 3.

Colors are only stored for voxels that are marked as confident and belong to the scanned surface. A confident voxel at a 3D position p is classified as surface voxel if

$$D_{avg}(p) \leq \frac{1}{\sqrt{2}} \cdot (1.0f - \theta_{avg}(p)) \qquad (3)$$

where $D_{avg}(p)$ is the weighted average distance, and

$$\theta_{avg}(p) = \frac{\sum_{i=0}^{n}(0,0,1)^T \cdot N_i(proj_i(p))}{n}. \quad (4)$$

Here, $\vec{N}_i$ is the screen-space normal at a 2D position of the i-th depth frame in the current cache, and $proj_i(p)$ projects the point p onto the 2D image plane of this frame.

As colors are only stored for confident voxels, and voxels that are integrated during sub-batch processing are always marked as non-confident, some voxels of the reconstructed scene are not assigned a color until the next batch is processed. However, ICP-based pose estimation, which is performed in every acquisition frame, relies on the color information to achieve improved accuracy. To account for this, the colors from the current RGB sensor image are blended into the blank areas of the image that is used for pose estimation.

5.1. Color Memory Management

The color data is managed in *pages* of fixed size. In the current implementation we use 32 bit virtual addressing, with a 19 bit page index and 13 bit offset inside the page. We address blocks of 32 voxels and store each color in 16 bit RGB565 encoding, which yields a maximum page size of 0.5 MB. In total, 256 GB of virtual address space can be managed in this way. A one-level page table is maintained on the GPU, providing space for all possible 2^{19} entries.

Whenever colors are accessed, the virtual address is translated into a physical address by performing a look-up in the page table and combining the 13 bit offset of the address with the physical page base address. Each page is exclusively reserved for a single world chunk (see Fig. 5).

Page allocation is performed via a single atomic counter. Each world chunk keeps track of a `currentPage` pointer. Whenever color allocation is performed during scene reconstruction, the size of the referenced page is atomically increased by 1.

Allocation of memory at the end of a page is done by atomically increasing the current page size. Due to parallel allocations performed by multiple threads, this may increase the indicated page size to a value above the maximum allowed size. If this is the case, the corresponding thread interprets the returned address as a failed allocation and instead requests a new page. To avoid multiple threads requesting new pages for a single world chunk, the current page pointer is atomically locked by all threads. Only the thread which successfully acquired the lock is allowed to increment the page and set the new page pointer. This can result in threads being unable to obtain a valid address, forcing the triggered allocations to be executed in subsequent passes. On NVIDIA GPUs, however, we utilize the fact that color allocation is performed only by a single thread per bit-block during reconstruction. When it is enforced that the size of each bit-block matches the GPU warp size, threads unable to acquire the lock can be put into a busy-waiting loop until the allocation has finished. In practice, page allocation occurs only rarely, and it did not introduce any noticeable performance impact.

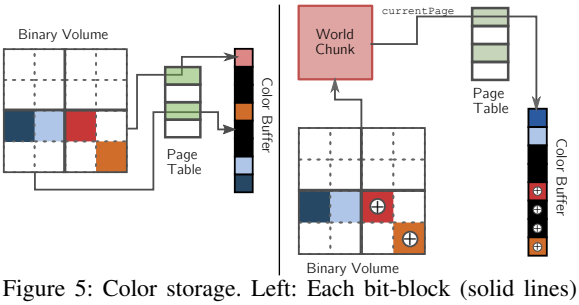


Figure 5: Color storage. Left: Each bit-block (solid lines) represents a number of voxels (dotted lines) and stores a virtual pointer into the color buffer. Virtual pointers are translated into physical addresses using a page table, and combined with the voxel index inside the bit-block to obtain the final address. Right: When new memory is allocated (⊕ indicates added colors), the `currentPage` pointer of the chunk is used and the data is appended to the page.

6. Data Streaming

Even with the proposed compact binary scene representation, when large spatial extents are scanned the acquired data can quickly become so large that it needs to be streamed from GPU into CPU memory, or even to external disk space. Data streaming is performed on the granularity of world chunks: All allocated chunks outside a spherical safety region around the camera can be removed from GPU memory. If the user returns to an already observed region in the scene, required chunks are streamed back onto the GPU. Detection of the binary bricks and color pages is performed similar to the collection of visible bricks by accessing the brick pool and page table in parallel, and writing out a compact buffer of corresponding page and brick pointers.

Whenever a color page is removed, we employ a lossless compression to further reduce the memory of the already compressed 16 bit colors for external storage. As color memory is allocated in blocks of fixed size, a large portion of each block will correspond to non-surface voxels and, thus, does not contain any color information. For typical scenes, this applies to about 60% of the data. Moreover, since colors are organized in a locally coherent manner, even non-empty neighboring values may be identical.

Our compression is inspired by the GPU compression scheme proposed by Treib et al [TRAW12] and builds upon the accompanying open-source *cudaCompress* library.

Empty space in the color data is first removed using run-length encoding, followed by a *lossless* compression step using parallel GPU Huffman encoding. The Huffman symbol table is computed for each page, allowing for an independent (de-)compression of pages during data streaming. On average, this approach compresses the color data by a factor of about 3.1 : 1, with a throughput of 4 GB/s and 8 GB/s for compression and decompression, respectively. Since only comparably small chunks of the volume need to be streamed in every frame, the compression and decompression times are usually negligible.

7. Discussion and Results

In this section we discuss the benefits and limitations of our approach for 3D online reconstruction. We have performed a number of experiments including live captures of scenes of various spatial extents to demonstrate the scalability of our solution and compare its quality to alternative approaches. The test scenes are shown in Figs. 1 and 12. All tests were performed with a Microsoft Kinect 1.0, a dual quadcore Intel Xeon X5560 with 48 GB RAM, and an NVIDIA GTX Titan with 6 GB video memory. For comparison we use the available implementation of voxel hashing [NZIS13] (VH).

7.1. Memory and Performance

In a first experiment we analyze the effective voxel resolutions determined during online reconstruction. Figure 6 shows the number of non-empty bricks for several scans using a chunk size of 2 meters and a brick size of 8^3 voxels. A voxel-to-pixel ratio of 1:1 was used to determine the level at which samples were inserted. It can be observed that only a few levels contain data, and that the allocated bricks contain mostly voxels of side lengths 4 mm, 2 mm and 1 mm. For all data sets, the vast majority of bricks reside at the 2 mm resolution level.

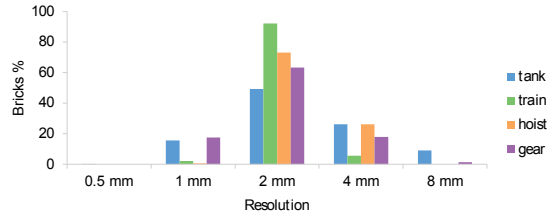


Figure 6: For different data sets, the amount of bricks storing voxels at a certain resolution is given as a percentage of all allocated octree leaf nodes. Voxel resolution refers to the spatial voxel extent in each dimension.

The analysis of the frequency of occurrence of bricks at different resolution levels serves as a baseline for the following comparison of the memory requirements and reconstruction speeds of our approach and VH. We perform the same scans with VH using the resolution required by the majority of voxels, i.e., 2 mm (termed VH_{HQ}). Since VH does not allow storing voxels at different resolution levels, we set a rather aggressive maximum depth cut-off of 2 meters. In addition, we perform a scan of lower quality using VH (VH_{LQ}) at a resolution of 4 mm and a depth cut-off of 5 meters. Our approach generates voxels adaptively up to the finest resolution of 1 mm. It uses a depth cut-off of 5 meters. Table 1 compares the memory requirements of VH and our approach, indicating a considerably lower memory consumption of our pipeline.

When colors are stored in addition to the scene geometry, our proposed color encoding scheme decreases the required memory by 45% on average compared to storing the colors per voxel. An additional factor of 2 is gained when colors are stored in the lossy RGB565 format. Upon data streaming, an average reduction factor of 3.13 is achieved via lossless compression. In VH, the storage of colors increases the memory requirement by a factor of 2.

During the same set of experiments, the overall system performance was measured. For all data sets, average times including all steps of online reconstruction are given in the column *Time* in Tbl. 1. Our approach is slightly faster than VH at 4 mm resolution, even though it performs considerably more operations per frame to project voxels into multiple cached depth images. This indicates the effectiveness of using a hierarchical representation in reducing the overall number of bricks, and of the distribution of cache processing over multiple frames in reducing the workload per frame.

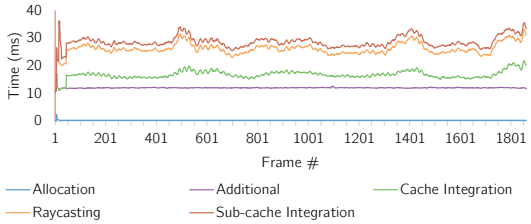


Figure 7: Performance analysis of the proposed online reconstruction pipeline. Times represented by the violet plot (*Additional*) include pose estimation, data streaming, and shading. Times add up to 31.3 ms on average.

A detailed performance summary for the online reconstruction shown in Fig. 1 is given by the graphs in Fig. 7. It can be seen that the integration of batches consumes most of the GPU time (11.8 ms on average), and significantly less work is done on the integration of sub-batches (3.1 ms). Color allocation, which is performed only for the larger batches, has only a subtle effect on the overall performance, even though we kept warps busy waiting for page creation. Compared to 5 ms rendering time reported for voxel hashing, volume ray-casting (8.98 ms) turns out to be more expensive in our approach. We attribute this to the higher level of thread divergence, which is caused by the increased resolution of our scenes and the additional indirections required

Table 1: Memory and performance evaluation, and comparison to voxel hashing (VH) storing distances and weights in bricks of 8^3, including colors. VH_{HQ} indicates the same scene resolution as used by our approach, yet our depth cut-off is more than twice as large. VH_{LQ} uses the same depth cut-off, but the scene resolution is decreased about a factor of 2. `Time` gives the average time required to generate a new image, including fusion, pose estimation, and rendering. Mem_s gives the memory consumption of our approach excluding colors (3 bits per voxel and including the pointer into the color array). Mem_{c_0} and Mem_{c_1}), respectively, give the memory required by the compressed and uncompressed color data. The last two columns indicate the GPU memory reduction compared to VH_{HQ}, excluding (2 bpv) and including colors (3 bpv + uncompressed color data).

	VH_{HQ}		VH_{LQ}		Ours					
	Memory	Time	Memory	Time	Mem_s	Mem_{c_0}	Mem_{c_1}	Time	ΔSurf.	ΔTotal
train	1496 MB	34 ms	416 MB	32 ms	**64 MB**	**178 MB**	57 MB	27 ms	94 %	83 %
tank	887 MB	42 ms	837 MB	40 ms	**54 MB**	**171 MB**	55 MB	36 ms	91 %	74 %
gear	1290 MB	41 ms	637 MB	36 ms	**56 MB**	**173 MB**	55 MB	32 ms	94 %	82 %
hoist	753 MB	42 ms	644 MB	33 ms	**45 MB**	**137 MB**	44 MB	34 ms	92 %	76 %

for virtual color addressing. Surprisingly, the time required for chunk and tree creation as well as brick data allocation is negligible (1.2 ms). The remaining time per frame is used for data streaming, pose estimation, and shading, yielding a total of 31.3 ms per frame on average. Thus, our approach is very well able to match the Kinect's rate of 30 frames per second in a typical online scenario. During integration of the first sub-batch a sudden peak in the processing time is observed, which is due to the large number of newly created bricks during the first frames. The frame rate, however, quickly stabilizes once the cache has been filled.

In most cases, the batch size n was set to 40, with a sub-batch of size $\hat{n} = 5$. Here it is important to note that increasing n does not necessarily decrease performance: Integration of a batch is distributed over all frames of the next batch by processing $\frac{1}{n}$ of the required bricks in each frame. Thus, increasing the batch size reduces the number of threads while increasing the workload per thread. Performance-wise, the "optimal" size depends on the combination of brick size, scanned surface area, and speed of movement. In our experiments, batches between 20 and as much as over 100 performed similarly well on average. Smaller batches are able to include a larger portion of confident voxels for pose estimation, making it less prone to drifts, but may exhibit a slightly increased amount of noise in the final reconstruction.

7.2. Reconstruction Quality

Compared to level-set ray-casting in a distance volume, the use of a binary voxel representation in combination with DDA ray-marching results in a less smooth surface appearance during online capture. However, due to the voxel-to-pixel ratio of 1:1, i.e., the rendering resolution matches the sensor resolution, and in combination with screen-space smoothing, the appearing block structures can be reduced and smooth normals can be calculated. Figure 8 compares the surface quality during online reconstruction using our approach and the distance field representation.

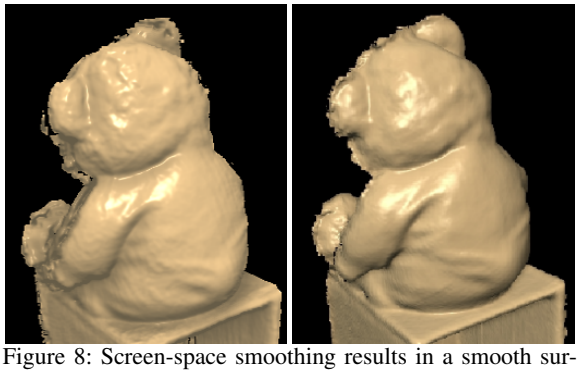


Figure 8: Screen-space smoothing results in a smooth surface appearance during online reconstruction (left). In the shown case, the surface appears very similar to the surface that is reconstructed from a distance field representation (right), yet in some regions screen-space smoothing slightly blurs out the surface normals (Data set from Sturm et al. [SEE*12].)

Pose Estimation. Since ICP-based pose estimation uses the images of the reconstructed surface to determine reference points, it must be assured that the accuracy of pose estimation is not affected negatively when the images are rendered from the binary voxel representation and smoothed in a deferred pass. To analyze this effect, we performed several experiments using data sets from the TUM RGB-D benchmark suite [SEE*12]. We compared the absolute trajectory error (ATE) as the root mean square error (RMSE) between the trajectories generated by our approach and voxel hashing, and compared both to the ground truth trajectories provided by the online tool. The results are shown in table 2.

In all cases where ICP-based pose estimation works for voxel hashing we could generate accurate trajectories as well, at a slightly larger error within a few centimeters. The additional error that is introduced by our approach is mainly due to the following reasons which affect the quality of ICP:

Firstly, since normals are computed from the depth imprint of the voxelized surface, inferior frame-to-frame coherence in the rendered images can be expected. Secondly, since windowed fusion uses less depth images to determine a voxels confidence, it is more likely to produce unclassified regions and resulting holes in the surface.

In the last two cases, both voxel hashing and our approach suffer from a severe tracking drift. This introduces a large error regardless the used fusion approach and makes a relative comparison of the approaches rather meaningless. We have nevertheless included the two cases for the sake of completeness.

Offline surface extraction. Given the binary voxel representation, a polygonal surface representation is extracted via the Marching Cubes algorithm in an offline process. To reduce block artifacts resulting from the binary representation, we apply mesh smoothing using a curvature-based smoothing filter (Taubin smoothing [Tau95]). Figure 10 shows the resulting surfaces, as well as the surface extracted from the corresponding distance field representation.

In Fig. 9 we color the smoothed surface that was reconstructed from the binary voxel representation according to the point-wise shortest distances to the surface resulting from the signed distance field representation. The visualization shows that in many areas the two surfaces match each other with very high fidelity. In some areas, however, the differences are significant, especially along some of the silhouettes and wrinkles on the surface. These are the areas where the measurements typically exhibit a high uncertainty, and due to the weaker regularization properties of windowed fusion it cannot reduce noise and outliers in these regions as effective as standard volumetric fusion.

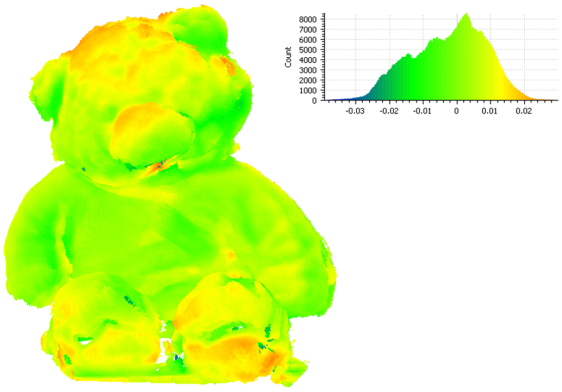


Figure 9: Point-wise shortest distances of the unsmoothed surface that was extracted in a post-process from the binary voxel representation to the surface extracted from a distance field of 4 mm voxel resolution. The shaded surfaces are shown in Fig. 10.

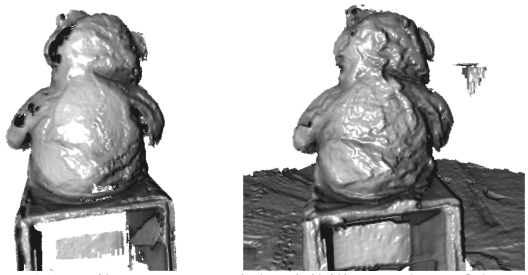


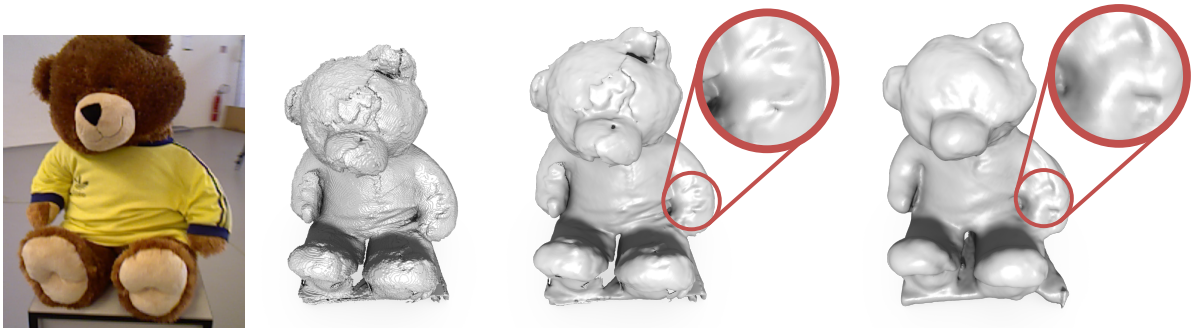
Figure 11: Failure cases: High visibility and confidence thresholds (left) lead to high quality surface, but tend to produce holes. Low thresholds (right) stabilize pose estimation, but tend to increase noise in the final surface.

7.3. Drawbacks and Limitations

To emphasize the potential difficulties of pose estimation when using images rendered from a binary voxel representation, we have selected a pre-computed scan of a large plush teddy, consisting of roughly 2'000 images (see Fig. 10(a)). As can be seen, the pose cannot be tracked accurately, resulting in a noticeable drift shown in images (b) and (c). Compared to the online reconstruction via VH (d), very fine details around the arm and the chest are reconstructed due to the high resolution of the binary representation, yet the surface extracted from the binary voxel representation exhibits more holes in uncertain regions where the voxels could not be classified as confident during the scan.

Since in our approach distance values and weight information are used only temporarily when processing the current batch, voxel and brick replacement must rely on heuristics which depend on a number of parameters. The relevant parameters in our approach are the initial weight Φ_0, the visibility threshold m for brick integration, the confidence threshold as well as the batch and sub-batch sizes. The optimal settings are scene-specific and, thus, the overall quality is more susceptible to a user's experience and beforehand knowledge of the scanned scene. In addition, the optimal parameters for stable pose estimation often differ from those required for optimal surface quality. If more weight gets attributed to new samples and old samples are aggressively replaced—which is the case for small visibility and confidence thresholds—the system is able to recover more quickly from tracking failures. On the other hand, higher thresholds yield superior surface quality in areas of confident voxels, but this may lead to holes in the extracted surface because non-confident voxels are discarded for this step. Both extremes are exemplified in Fig. 11.

We found batch and sub-batch sizes of $n = 40$ and $\hat{n} = 5$ to provide a good balance between reconstruction quality and pose estimation accuracy in most cases. The sub-batch was offset by another 5 images to not include bricks that are visible at the beginning or end of a frame. The visibility and confidence thresholds vary between $0.5n$ and $0.75n$, and the



(a) Input (b) Ours (unprocessed) (c) Ours (smoothed) (d) Voxel Hashing

Figure 10: (a) The scanned object. (b) The Marching Cubes surface from the binary voxel representation reconstructed by our approach. (c) Smoothing applied to the mesh in (b). (d) The Marching Cubes surface from the distance field representation generated by Voxel Hashing when scanning the object in (a) at 4 mm. The tracking drift due to difficulties in pose estimation is clearly visible in (b) and (c). The surface in (d) shows less details in the wrinkled surface regions, yet it contains less holes than the surfaces in (b) and (c).

surface weight Φ_0 was set to a rather low value of about $0.1n$.

Compared to online reconstruction using a distance field representation and confidence weights, especially in case of tracking drifts our approach is more prone to errors such as the duplication of existing surfaces when scanned surface parts are revisited. We aim at reducing this drawback in the future by performing more accurate pose estimation, either by incorporating visual SLAM methods [KSC13] or additional sensors [NDF14]. However, when objects are scanned mostly along one axis—as for the use cases `train` and `gear`—our pipeline achieves quite accurate results.

8. Conclusion and Future Work

In this work we have presented a memory-efficient real-time variant of volumetric fusion using a commodity depth sensor. We introduced *deferred integration* to eliminate the need to explicitly store a distance field and associated weights. In combination with a compact binary encoding of surface samples this reduces significantly the overall memory requirement. Since the smoothed image of the ray-traced binary voxel surface is used for ICP-based pose estimation, our approach is more prone to tracking drifts than approaches working on a signed-distance field scene representation. By inserting samples into an adaptive octree which is grown dynamically at runtime, we are able to represent all parts of the scanned surface at their adequate resolution, avoiding under- or over-sampling the sensor's depth and color data.

In the future we will investigate the following aspects in more detail. First, we will investigate the integration of surface extraction from the binary field and mesh smoothing into the online reconstruction process to improve the quality of ICP-based pose estimation. Therefore it needs to be analyzed whether the extraction process, including the generation of a shared vertex mesh representation needed by the smoothing process, can be performed at sufficient speed on the GPU and does not interfere with other GPU operations. Second, we aim to entirely avoid the computation of a distance field. Therefore, we will try to find means to reconstruct the surface during ray traversal from the surface bits in the surrounding of the sampling points along the ray. This approach will be very similar to the one proposed by Keller et al. [KLL*13], yet it will avoid storing point coordinates explicitly and performing costly search operations on the unstructured point set.

Table 2: Root mean square error (RMSE) between the trajectories generated via ICP-based pose estimation using rendered images of the binary (Ours) and the distance field (VH) representation. Several data sets of the TUM RGB-D benchmark suite are evaluated.

Data Set	Images	Error (VH)	Error (Ours)
fr1_xyz	798	0.014 m	0.016 m
fr1_rpy	722	0.035 m	0.038 m
fr1_desk	595	0.033 m	0.060 m
fr3_long_office	2509	0.070 m	0.086 m
fr1_360	755	0.741 m	0.514 m
fr1_room	1360	1.013 m	0.723 m

Acknowledgements

We thank the reviewers for their constructive criticism and suggestions, Matthias Nießner for providing access to Voxel Hashing, and Jürgen Sturm for providing the TUM RGB-D benchmark suite.

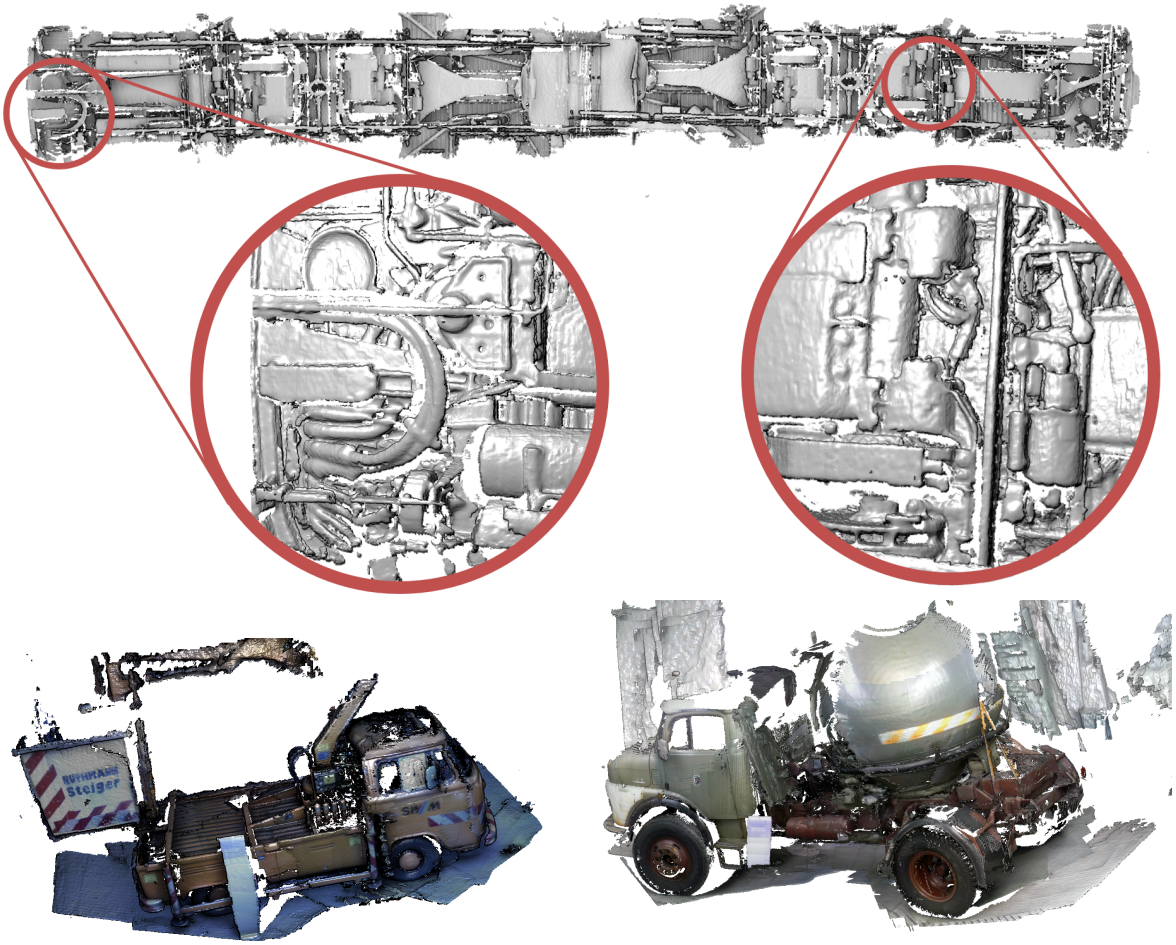


Figure 12: The data set `train` (top row) shows the bottom of a train and is about 15 meters in length. Bottom row: data sets `hoist` and `tank`.

References

[BM92] BESL P. J., MCKAY N. D.: A method for registration of 3-d shapes. *IEEE Trans. Pattern Anal. Mach. Intell. 14*, 2 (Feb. 1992), 239–256. 3

[CBI13] CHEN J., BAUTEMBACH D., IZADI S.: Scalable real-time volumetric surface reconstruction. *ACM Trans. Graph. 32*, 4 (July 2013), 113:1–113:16. 1, 2

[CL96] CURLESS B., LEVOY M.: A volumetric method for building complex models from range images. In *Proceedings of SIGGRAPH '96* (New York, NY, USA, 1996), ACM, pp. 303–312. 2, 3

[CM92] CHEN Y., MEDIONI G.: Object modelling by registration of multiple range images. *Image Vision Comput. 10*, 3 (Apr. 1992), 145–155. 3

[CNS*11] CRASSIN C., NEYRET F., SAINZ M., GREEN S., EISEMANN E.: Interactive indirect illumination using voxel cone tracing, sep 2011. 6

[CS09] CORDS H., STAADT O. G.: Interactive screen-space surface rendering of dynamic particle clouds. *Journal of Graphics, GPU, and Game Tools 14*, 3 (2009), 1–19. 6

[FG11] FUHRMANN S., GOESELE M.: Fusion of depth maps with multiple scales. *ACM Trans. Graph. 30*, 6 (Dec. 2011), 148:1–148:8. 2

[Hil96] HILTON A.: On reliable surface reconstruction from multiple range images. Springer-Verlag, pp. 117–126. 2

[IKH*11] IZADI S., KIM D., HILLIGES O., MOLYNEAUX D., NEWCOMBE R., KOHLI P., SHOTTON J., HODGES S., FREEMAN D., DAVISON A., FITZGIBBON A.: Kinectfusion: Real-time 3d reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology* (New York, NY, USA, 2011), UIST '11, ACM, pp. 559–568. 2

[KLL*13] KELLER M., LEFLOCH D., LAMBERS M., IZADI S., WEYRICH T., KOLB A.: Real-time 3d reconstruction in dynamic scenes using point-based fusion. In *3DV* (2013), IEEE, pp. 1–8. 2, 11

[KSC13] KERL C., STURM J., CREMERS D.: Dense visual slam for rgb-d cameras. In *Proc. of the Int. Conf. on Intelligent Robot Systems (IROS)* (2013). 2, 11

[KW03] KRÜGER J., WESTERMANN R.: Acceleration Tech-

niques for GPU-based Volume Rendering. In *Proceedings IEEE Visualization 2003* (2003). 3

[LC87] LORENSEN W. E., CLINE H. E.: Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of the SIGGRAPH '87* (New York, NY, USA, 1987), ACM, pp. 163–169. 3

[Lem10] LEMPITSKY V. S.: Surface extraction from binary volumes with higher-order smoothness. In *CVPR* (2010), IEEE, pp. 1197–1204. 3

[MAW*07] MERRELL P., AKBARZADEH A., WANG L., MICHAEL FRAHM J., NISTÉR R. Y. D.: Real-time visibility-based fusion of depth maps. In *In Int. Conf. on Computer Vision and Pattern Recognition* (2007). 2

[MC13] MEILLAND M., COMPORT A.: On unifying key-frame and voxel-based dense visual SLAM at large scales. In *International Conference on Intelligent Robots and Systems* (Tokyo, Japan, 3-8 November 2013), IEEE/RSJ. 2

[MLK*13] MÖNCH T., LAWONN K., KUBISCH C., WESTERMANN R., PREIM B.: Interactive mesh smoothing for medical applications. *Computer Graphics Forum 32*, 8 (2013), 110–121. 3

[NDF14] NIESSNER M., DAI A., FISHER M.: Combining inertial navigation and icp for real-time 3d surface reconstruction. 11

[NIL12] NGUYEN C. V., IZADI S., LOVELL D.: Modeling kinect sensor noise for improved 3d reconstruction and tracking. In *3DIMPVT* (2012), IEEE, pp. 524–530. 3

[NZIS13] NIEßner M., Zollhöfer M., Izadi S., Stamminger M.: Real-time 3d reconstruction at scale using voxel hashing. *ACM Transactions on Graphics (TOG)* (2013). 1, 2, 4, 5, 6, 8

[RCSW14] REICHL F., CHAJDAS M., SCHNEIDER J., WESTERMANN R.: Interactive rendering of giga-particle fluid simulations. *Proceedings of High Performance Graphics 2014* (2014), 105–116. 6

[RL00] RUSINKIEWICZ S., LEVOY M.: Qsplat: A multiresolution point rendering system for large meshes. In *Proceedings of SIGGRAPH '00* (New York, NY, USA, 2000), ACM Press/Addison-Wesley Publishing Co., pp. 343–352. 2

[SEE*12] STURM J., ENGELHARD N., ENDRES F., BURGARD W., CREMERS D.: A benchmark for the evaluation of rgb-d slam systems. In *Proc. of the International Conference on Intelligent Robot Systems (IROS)* (Oct. 2012). 4, 9

[Tau95] TAUBIN G.: A signal processing approach to fair surface design. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques* (1995), ACM, pp. 351–358. 3, 10

[TM98] TOMASI C., MANDUCHI R.: Bilateral filtering for gray and color images. In *Proc. Intl. Conf. on Computer Vision* (1998), IEEE, pp. 839–846. 6

[TRAW12] TREIB M., REICHL F., AUER S., WESTERMANN R.: Interactive editing of gigasample terrain fields. *Computer Graphics Forum (Proc. Eurographics) 31*, 2 (2012), 383–392. 7

[vdLGS09] VAN DER LAAN W. J., GREEN S., SAINZ M.: Screen space fluid rendering with curvature flow. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2009), I3D '09, ACM, pp. 91–98. 6

[WKJ*14] WHELAN T., KAESS M., JOHANNSSON H., FALLON M., LEONARD J., MCDONALD J.: Real-time large scale dense RGB-D SLAM with volumetric fusion. *Intl. J. of Robotics Research, IJRR* (2014). To appear. 2

[WWLG09] WEISE T., WISMER T., LEIBE B., GOOL L. V.: In-hand scanning with online loop closure. In *IEEE International Workshop on 3-D Digital Imaging and Modeling* (2009). 2