Efficient and Precise Multi-Camera Reconstruction

Tobias Werner
Universität Bayreuth
Bayreuth, Germany
tobias.werner@uni-bayreuth.de

Dominik Henrich
Universität Bayreuth
Bayreuth, Germany
dominik.henrich@uni-bayreuth.de

ABSTRACT

Human-robot cooperation requires efficient and precise geometric reconstruction of objects in any shared workspace to meet safety requirements. State-of-art solutions accept a trade-off to satisfy these requirements. Usually, this implies limited robot speed, expensive hardware, or low-precision input and output. In contrast, we present a novel multi-camera reconstruction approach that is both efficient and precise: We apply incremental and hierarchical algorithms to image preprocessing and visual hull reconstruction. Subsequent knowledge-based postprocessing refines the initial reconstruction. Given ground-truth input, we furthermore maintain conservative bounds of objects in the workspace. In total, our contribution satisfies real-time and anytime requirements even on modest hardware and for high-precision input and output. We validate this by synthetic and real-world experiments. Thus, our approach is suited for safety-, time-, and precision-critical reconstruction with applications in home robotics, surveillance, and industrial automation.

1. INTRODUCTION

Safe human-robot coexistence is an established field of research in robotics. Applications focus on industrial use, such as separated robot and human workspaces along an assembly line in the automotive industry. In order to meet rigid safety requirements, common coexistence applications use primitive but fail-safe means to avoid robot-human collisions. For instance, a laser scanner might enforce an emergency stop once it detects an obstacle within reach of a robot.

In contrast to mere coexistence, human-robot collaboration supports extended applications. Ideally, humans and robots work hand in hand and complement one another. Applications might see a robot and a human process the same assembly in parallel. Such collaboration implies that humans and robots act in close vicinity. Hence, safety requirements increase even further. In particular, robots must guarantee safety by efficient, conservative, and precise obstacle reconstruction and avoidance.

Recent advances in path planning, computer vision and artificial intelligence form the foundation of human-robot collaboration. For instance, [2] introduces an efficient path planner and [8] describes a robust multi-camera obstacle reconstruction. However, the latter offers precision and conservativeness at the cost of efficiency. Therefore, it does not apply to demanding real-time applications.

In the following sections, we improve on named reconstruction approach and introduce an efficient algorithm that still maintains precision and conservativeness. At first, we give a short overview over related reconstruction algorithms. Thereafter, we explain algorithmic and implementation details of our improved approach. We conclude by evaluating our reconstruction in synthetic and real-world experiments.

2. RELATED WORK

Geometric reconstruction algorithms can be classified according to various criteria. The most notable criteria designate algorithmic input (color [6, 12], pre-segmented [1], or depth [5, 10] images) and output (point cloud [10], triangle mesh [5, 6], voxel space [9, 13]). Optionally, both input and output may exploit spatial coherence (octrees [1, 12], quadtrees [9]) and temporal coherence (incremental updates [1, 5]). The source of actual input (single-camera [5], stereo camera [6], multi-camera [9, 12]) represents a more application-specific criterion. Ultimately, designated applications influence respective algorithms via domain-specific requirements (offline [3, 6], online [9, 13], real-time and anytime constraints [12], conservativeness guarantee [9]).

The following discussion of related work restricts to geometric multi-camera reconstruction that is suited for online use in a robot workspace. Currently, most such approaches favor voxel reconstructions over more intricate algorithms. For example, the reconstruction [13] exploits per-voxel data parallelism to efficiently perform voxel-to-silhouette tests on a GPU. Including CPU-GPU round trips, this requires about 10ms per reconstruction of 256³ voxels. Similarly, [9] evaluates voxel occupation on a GPU, but precalculates projections to improve performance. For 128³ voxels, this approach reports reconstruction times of around 65ms.

Voxel discretizations parallelize well on modern GPUs and allow for various offline precalculations. Yet, they do not scale well to high input and output resolution and they do not adapt to inhomogeneous object distributions or variable real-time limits. Last but not least, voxels do not support efficient distance queries, e.g. as required by collision avoidance within path planning for a robot manipulator.

Hierarchical data structures are an obvious alternative.
For instance, [9] reconstructs an octree from multiple camera images via image-space downsampling and single-pixel projection tests. This reduces reconstruction times to 25 ms for an octree equivalent of $128^3$ voxels. Likewise, [12] gives an algorithm for parallel octree reconstruction on either CPU or GPU. They guarantee maximum load over all cores through a work-stealing scheme. With parallelization over 16 CPU cores, they claim time requirements of 25 ms, again for an equivalent of $128^3$ voxels. Finally, [7] proposes reconstruction of an octree from multiple quadtrees over pre-segmented input images, albeit without experimental verification.

Incremental updates improve performance both for voxel-based and hierarchical reconstruction. Namely, [1] locates modified entries within input images and only updates any voxels on rays through these entries. This way, the approach achieves a reconstruction time of 100 ms for $256^3$ voxels, allegedly faster than non-incremental alternatives.

3. OUR APPROACH

3.1 Overview

This section covers our approach to geometric reconstruction. In terms of the above classification criteria, we support online, hierarchical, incremental, real-time, anytime, and conservative multi-camera obstacle reconstruction from pre-segmented or depth input images.

We begin the presentation of our reconstruction algorithm by naming input and output requirements. Formal requirements extend to incoming images, camera projections, and the conservativeness guarantee. Informal requirements include real-time and anytime capabilities as well as a path-planning integration for robotics applications.

A discussion of our core reconstruction algorithm follows. In brief, the algorithm performs two separate steps: The first step calculates conservative quadtrees over all current input images within a multi-camera setup. The second step incrementally computes a conservative octree reconstruction of the observed objects. This involves projecting octree nodes into camera images and efficient testing for object silhouettes within respective quadtrees. Both steps track modifications in-between consecutive input images in order to reuse quadtree test results or unmodified octree branches. We apply various knowledge-based and low-level optimizations to further improve reconstruction efficiency and precision. Finally, client applications can raise an exit flag to abort reconstruction prematurely. This allows for real-time and anytime capabilities within incremental computations.

Quadtrees and octrees are intuitive data structures, and we have studied extensively [11, 15]. Hence, we limit consequent explanations to aspects relevant to our algorithm.

3.2 Requirements

Requirements derive from multi-camera reconstruction in a robot workspace as illustrated by Figure 1. Formally, we assume that the workspace consists of a cubic volume $V = [0, 1]^3 \subseteq \mathbb{R}^3$. Part of this workspace is occupied by a priori unknown, possibly moving objects. The potentially incoherent volume $V_{obj,t} \subseteq V$ represents ground-truth bounds of all workspace objects at some time step $t \in \mathbb{N}$. We wish to derive a conservative reconstruction $V_{rec,t}$ of these objects, that is, $V_{obj,t} \subseteq V_{rec,t} \subseteq V$ to this end, $c$ cameras observe the workspace. Each camera $i$ has a square, power-of-two resolution $r_i = 2^k, k \in \mathbb{N}$ with pixels $R_i = [1,r_i]^2 \subseteq \mathbb{N}^2$. We require a power-of-two resolution for formal reasons without loss of generality, e.g. by duplicating border pixels on real-world images.

At each time step $t$, the camera $i$ provides per-pixel-image data from a certain data set $D_i$, as a map $\lambda_{i,t} : R_i \to D_i$. Our core reconstruction algorithm does not depend on any specific data set $D_i$. However, the following presentation explicitly considers pre-segmented foreground-background and depth cameras. The former employ a data set $D_{seg} = \{\text{full, empty}\}$, while the latter use $D_{dep} = \mathbb{R}$.

Our approach assumes extrinsic and intrinsic camera parameters as given. In particular, we expect projection functions $\varphi_i : V \to R_i \times \mathbb{R}$ from the observed volume into image-space and depth coordinates for camera $i$.

Pin-hole camera models typically use a projection function $\varphi_{ph}(v) = M_{img}w(M_{cam}v)$ with $M_{cam} \in \mathbb{R}^{4 \times 4}$ an affine world to camera transformation, $P \in \mathbb{R}^{4 \times 4}$ a projection matrix, $w : \mathbb{R}^4 \to \mathbb{R}^4$ a homogenous division, and $M_{img} \in \mathbb{R}^{3 \times 3}$ a map from normalized view coordinates to output pixels. We also support more elaborate camera models. Notably, later experiments rely on a camera model with radial and tangential distortions as supported by [14].

Usually, camera distortions enforce expensive pixel-exact projections. We avoid this overhead through the use of conservative bounding functions $\tilde{\varphi}_i : 2^V \to 2^{R_i \times \mathbb{R}}$, where

$$\tilde{\varphi}_i(V) = \begin{bmatrix} \min_{v \in V} \varphi_i(v) \\ \max_{v \in V} \varphi_i(v) \\ \min_{v \in V} \varphi_i(v) \\ \max_{v \in V} \varphi_i(v) \end{bmatrix}.$$ 

Pin-hole projections conserve convex bounds of any finite, convex polyhedron entirely within the viewing frustum. A respective bounding function builds an image-space bounding box over projections of the polyhedron vertices. Advanced camera models require further provisions (e.g. an additional epsilon) to account for camera-specific bounding distortions that exceed vertex projections. In practice, one can also often avoid explicit projecting of children nodes in the later octree by z-correction interpolation of parent projections.

Apart from above formal requirements, there are four non-formal constraints. First, reconstruction must be efficient even for high-resolution input. We expect an efficient algorithm to handle input from at least four Full HD cameras on consumer hardware at a mean 100 ms per reconstruction. Second, we desire a precise reconstruction. We understand a reconstruction to be precise if it exhaustively reflects all available input details. Hence, the difference $V_{rec,t} / V_{obj,t}$ should be as small as possible. In our experiments, Full HD input corresponds to a precise reconstruction with an equivalent of at least $1024^3$ voxels. Third, premature exit due to external time limits must still deliver a valid and in
To do so, it is favorable to extend the image data set $D$ with a merge function to satisfy conservativeness. A total rebuild for slow-moving or static objects is not necessary for consecutive reconstructions. In turn, this avoids a total rebuild for slow-moving or static objects.

Incremental updates to the later octree use $q_{mod,t,i}$ in order to efficiently test whether spatially coherent input pixels changed for consecutive reconstructions. In turn, this avoids a total rebuild for slow-moving or static objects.

3.3 Quadtrees

In the following, we build quadtrees $q_{t,i}$ over images $\lambda_{t,i}$ generated for time steps $t$ by cameras $i$. A quadtree over an image of $r^2$ pixels has $h_i = \log_2(r_i) + 1$ levels, where the lowest level $q_{t,i,0} : R \rightarrow D$, $q_{t,i,0}(r) = \lambda_{t,i}(r)$, contains the actual image data.

We build quadtrees bottom-up and apply a merge function $m : D^4 \rightarrow D$ to reduce each four-entry object within one level to a single entry within the next-higher level. This yields levels $1 \leq j < h_i$.

$$q_{t,i,j} : [1, \frac{r}{2^j}]^2 \rightarrow D,$$

where

$$q_{t,i,j}(r) = m(q_{t,i,j-1}(2r), q_{t,i,j-1}(2r + (1, 0)^T), q_{t,i,j-1}(2r + (0, 1)^T), q_{t,i,j-1}(2r + (1, 1)^T)).$$

Merge functions must maintain or increase the perceived volume of objects within the scene to satisfy conservativeness. To do so, it is favorable to extend the image data set $D$. For pre-segmented images, we use the data set $D_{seg,q} = \{ \text{full, empty, mixed} \}$ alongside a merge function

$$m_{seg}(d_1, ..., d_4) = \begin{cases} \text{full} & \text{if } V_i : d_i = \text{full}, \\ \text{empty} & \text{if } V_i : d_i = \text{empty}, \\ \text{mixed} & \text{otherwise}. \end{cases}$$

On depth images, a minimum function over all four incoming depth values already conserves object boundaries. However, later optimizations require minimum and maximum camera distances for coarse quadtree levels. Hence, we use $D_{dep,q} = \mathbb{R}^2$. Initial images set both distances to the incoming per-pixel depth value. A merge function

$$m_{dep}(d_{min,1}, ..., d_{max,4}) = \min(d_{min,1}, ..., d_{min,4}) \text{ if } \max(d_{max,1}, ..., d_{max,4})$$

generates all remaining quadtree levels.

Finally, we mark modified pixels within each camera $i$ at each time step $t$ in a separate quadtree $q_{mod,t,i}$. For each entry at each quadtree level, we store whether any covered root-level pixel was modified in-between the last two time steps. This gives quadtrees on a data set $D_{mod} = \{ \text{modified, unmodified} \}$ over a root image

$$\lambda_{mod,t,i}(r) = \begin{cases} \text{modified} & \text{if } t = 0 \lor \\
\text{unmodified} & \text{otherwise}.
\end{cases}$$

with a merge function

$$m_{mod}(d_1, ..., d_4) = \begin{cases} \text{modified} & \text{if } \exists i : d_i = \text{modified}, \\ \text{unmodified} & \text{otherwise}.
\end{cases}$$

Incremental updates to the later octree use $q_{mod,t,i}$ in order to efficiently test whether spatially coherent input pixels changed for consecutive reconstructions. In turn, this avoids a total rebuild for slow-moving or static objects.

3.4 Octrees

Octree updates for a time step $t$ begin once our algorithm has built both the actual data quadtree $q_{t,i}$ and the modified quadtree $q_{mod,t,i}$ for each input image $i$ as described above.

In general, a sparse octree consists of a set of nodes $N$. Each node $n \in N$ occupies some space $\mathcal{V}_n = [n_{min}, n_{max}] \subseteq \mathbb{R}$. A function $\sigma : N \rightarrow 2^N$ defines parent-child relations at some time step $t$. All leaf nodes $l$ of the octree must satisfy $\sigma(l) = \emptyset$. In contrast, all non-leaf nodes $p \in N$ require invariants $|\sigma(p)| = 8$, $\mathcal{V}_p = \bigcup_{n \in \sigma(p)} \mathcal{V}_n$, and $\forall n,n' \in \sigma(p) : \mathcal{V}_n - \mathcal{V}_n' = \mathcal{V}_n' - \mathcal{V}_n$. In our case, each node additionally is in a time-variant state $s_{n,t} \in S = \{ \text{full, empty, mixed} \}$. Node states reflect octree sparsity: Full and empty nodes are homogenous, hence further splitting is pointless. Such nodes form the leaves of our octree. In contrast, we split nodes with mixed content to refine our reconstruction. Initially, an octree contains only a single node $n_{root}$, with $\mathcal{V}_{n_{root}} = \mathcal{V}$, $\sigma(n_{root}) = \emptyset$, and $s_{n_{root}} = \text{mixed}$.

For incremental updates, the octree communicates with each quadtree $i$ via a decision function $r_{i,t} : N \rightarrow S$. This function must consider relevant quadtree entries to decide on the state of a node $n$. Note that all decisions must maintain conservativeness: A node $n$ must report as full or mixed if the respective camera cannot rule out an object within $\mathcal{V}_n$. This accounts for the fact that the projection of any object into a camera may potentially occlude other, more distant objects. One refers to such occlusion-generated objects as pseudo objects. In practice, decisions also classify nodes as mixed if these are not or only partially visible to a camera.

At first, we introduce utility functions $\hat{r}_{i,t,j} : N \rightarrow S$ that consider only a single quadtree level $j$ for their decision. Pre-segmented images satisfy conservativeness via a trivial per-level decision function

$$\hat{r}_{seg,t,i,j}(n) = \begin{cases} \text{empty} & \text{if } \forall d \in \hat{\rho}_i(V_n) : \\
q_{t,i,j}(d_{seg}/2^j) = \text{empty}, \\ \text{full} & \text{if } \exists d \in \hat{\rho}_i(V_n) : \\
q_{t,i,j}(d_{seg}/2^j) = \text{full}, \\ \text{mixed} & \text{otherwise}.
\end{cases}$$

which causes pseudo objects on entire viewing cones through foreground pixels. In contrast, depth images compare node
bounding against quadtree minimum and maximum depths,

\[
\hat{\tau}_{\text{dep},t,i,j}(n) = \begin{cases} 
\text{empty} & \text{if } \forall d \in \tilde{\varphi}_{t}(V_{n}) : \\
\phi_{i,\text{max},x}(V_{n}) < q_{t,i,j,\text{min}}(dx_{t}/2^{j}), \\
\text{full} & \text{if } \forall d \in \tilde{\varphi}_{t}(V_{n}) : \\
\phi_{i,\text{min},x}(V_{n}) > q_{t,i,j,\text{max}}(dx_{t}/2^{j}), \\
\text{mixed} & \text{otherwise},
\end{cases}
\]

and hence reduce pseudo objects to actual occlusions behind object silhouettes.

As merge functions build conservative quadtree entries, coarse quadtree levels offer early-out opportunities once the decision function returns \emph{full} or \emph{empty}. Otherwise, the current quadtree level provides indecisive results, and the node tests against the next finer level. Only tests on the lowestmost quadtree level return \emph{mixed}. A function \(\hat{n}_{t,i,j} : N \rightarrow S\),

\[
\hat{n}_{t,i,j}(n) = \begin{cases} 
\text{empty} & \text{if } j < h_{i} \land \hat{\tau}_{t,i,j+1}(n) = \text{empty}, \\
\text{full} & \text{if } j < h_{i} \land \hat{\tau}_{t,i,j+1}(n) = \text{full}, \\
\hat{\tau}_{t,i,j}(n) & \text{otherwise},
\end{cases}
\]

models these optimized queries.

For improved efficiency, updates only invoke decision functions for the current time step \(t\) if the respective quadtree segment was modified. This leads to the final per-image decision functions \(\tau_{t,i}\),

\[
\tau_{t,i}(n) = \begin{cases} 
\tau_{t-1,i}(n) & \text{if } \exists j : \forall d \in \tilde{\varphi}_{t}(V_{n}) : \\
q_{t,j,i}(dx_{t}/2^{j}) = \text{unmodified}, \\
\hat{\tau}_{t,i,j}(n) & \text{otherwise},
\end{cases}
\]

Incremental octree updates invoke the above decision function \(\tau_{t,i}\) to determine the state of any node. If a single camera guarantees that a node is \emph{empty}, recursion stops, and reconstruction removes any existing children of the node. The algorithm performs analog actions if all cameras report that a node is \emph{full}. Otherwise, reconstruction splits leaf nodes and recursively updates children. This behavior reflects in the formal node state

\[
s_{t,i,n} = \begin{cases} 
\text{empty} & \text{if } \exists i : \tau_{t,i}(n) = \text{empty}, \\
\text{full} & \text{if } \forall i : \tau_{t,i}(n) = \text{full}, \\
\text{mixed} & \text{otherwise},
\end{cases}
\]

and the time-variant parent to children map

\[
\sigma_{t}(p) = \begin{cases} 
\{n_{p,1}, \ldots, n_{p,S}\} & \text{if } s_{t,p} = \text{mixed} \land \\
\emptyset & \exists i : \left|\{dx_{t} \in \tilde{\varphi}_{t,x}(V_{p})\}\right| > 1, \\
\emptyset & \text{otherwise},
\end{cases}
\]

both for \(t > 0\). Note that subdivision continues until all pixel-level information within all source images has been exploited. Hence, our reconstruction is as precise as possible even though we only consider node bounding boxes.

Incremental updates may also abort on an informal, external real-time limit. To maintain conservativeness, we then must consider any untouched octree branches as potential objects. In other words, we ignore any respective children and assume a mixed state. This results in a less detailed, yet still conservative reconstruction.

Given an octree in any build state, the desired reconstructed volume \(V_{\text{rec},t}\) consists of the volume of all full or mixed leaf nodes, i.e. \(V_{\text{rec},t} = \hat{V}_{t}(n_{\text{root}})\) with \(\hat{V}_{t} : N \rightarrow 2^{V}\),

\[
\hat{V}_{t}(n) = \begin{cases} 
V_{n} & \text{if } \sigma_{t}(n) = \emptyset \land s_{t,n} \neq \emptyset, \\
\bigcup_{n' \in \sigma_{t}(n)} \hat{V}_{t}(n') & \text{otherwise}.
\end{cases}
\]

With some work, one can now prove that the reconstruction indeed is conservative, i.e. that \(V_{\text{det},t} \subseteq V_{\text{rec},t} \subseteq V\).

Figure 3 shows results of unrefined reconstruction with our incremental, hierarchical and conservative algorithm. At last, we can adapt the distance metric from initial requirements as to use our octree, e.g.

\[
\mu(v) = \min_{v_{\text{rec}} \in V_{\text{rec},t}} \|v_{\text{rec}} - v\| = \min_{s_{t,n} \neq \emptyset} \min_{\sigma_{t}(n) = \emptyset} \min_{v' \in V_{n}} \|v' - v\|.
\]

Actual implementations can exploit octree properties to exclude nodes from distance queries in the above equation.

### 3.5 Knowledge-based Refinement

Once geometric reconstruction has finished, we apply various knowledge-based criteria (e.g. as in [8]) to refine reconstruction output. For instance, if the target application only requires to avoid robot-human collisions, we can safely ignore any coherent object that does not meet the minimum volume occupied by a human. Notably, this removes small artifacts generated by camera noise. In an alternative scenario, we can assume that the observed volume initially contains no unknown object. We can thus disregard all coherent volumes of unknown origin that have not had contact to the boundaries of the workspace or to undiscarded volumes yet. This especially helps with excluding the pseudo objects generated by independently moving occluders.

We also exploit external knowledge to remove known dynamic objects from the reconstruction. In robotics applications, this foremost refers to actual robots. If the object reconstruction includes these, later path planning detects a zero-distance obstacle and prohibits any robot movement. Other known objects (e.g. conveyor belts, lifts, automated trolleys) within the reconstruction do not interfere with path planning. However, replacing their reconstruction by an analytical form, such as a CAD model, usually improves planning precision and efficiency.

All knowledge-based refinement efficiently integrates into our octree hierarchy. Namely, we need only touch few nodes for volume or neighborhood tests, as opposed to expensive flood fill operations on a voxel field. In the end, refinement reduced the reconstruction error \(V_{\text{rec},t} / V_{\text{det},t}\) of later experiments without significant performance impact.
walking through this cell. We rendered the sequence from
we discuss experiment setup and results.

3.6 Optimizations
Algorithmic and implementation-level optimizations are
necessary to accomplish efficient reconstruction on commod-
ity hardware. In the following, we describe four strategies
that we used to optimize our reconstruction.

The first type of optimization exploits intuitive short-
circuit evaluation on the formal algorithm specification. For
instance, a quadtree level test on a segmented image can
instantly return once it has touched both a foreground and a
background entry. Likewise, we need not examine quadtrees
of further camera images once a single quadtree test guaran-
tees that a node does not contain any object.

The second strategy to improve performance involves a
memory-efficiency trade-off. There are various functions
that do not depend on the current time step $t$. Caching
these within memory significantly reduces reconstruction
times. Most importantly, each octree node $n$ may cache its
expense projection bounds $\tilde{\gamma}_i(V_n)$ into the image space of
each camera $i$. We also keep per-node quadtree results $\tau_i(n)$
and only rebuild these on demand. In this context, one
must note that memory consumption of aggressive caching
is substantial. Later experiments occupied about 12GB of
main memory. However, we found memory less of a sparse
resource when compared to processing time.

In our third optimization, we parallelize reconstruction
over multiple octrees that evenly partition the workspace
volume. We do not use advanced per-thread load-balancing
as in [12]. Instead, we generate sufficient (e.g. 512) trees, and
distribute these over waiting threads. This enables adequate
data-parallel processing of individual instances. In the end,
al later experiments achieved an almost linear speedup.

Fourth, we optimize memory management for frequent
allocation and deallocation of octree nodes. We particularly
avoid memory and run-time overhead on both operations
by a custom allocator with a block-based node allocation
strategy. Each root owns a unique allocator instance to avoid
expensive locking over concurrent threads.

4. EXPERIMENTAL RESULTS
We evaluated our reconstruction algorithm in three ex-
periments: An experiment with synthetic data, an offline
experiment with real-world data, and an online test in a live
multi-camera system. Figure 4 presents an overview over
input and output for all three experiments. In the following,
we discuss experiment setup and results.

For synthetic tests, we employed a CAD model of an ex-
isting robot cell. An animation sequence shows a woman
walking through this cell. We rendered the sequence from
seven virtual cameras and generated three types of images:
photo-realistic, pre-segmented, and depth images. Extrin-
sic and intrinsic parameters of the virtual cameras matched
those of their real-world counterparts, including a resolu-
tion of $640 \times 480$ pixels. We then tested reconstruction in
three modes: First, we directly used pre-segmented images
as ground-truth input. This allowed us to verify algorithm ef-
ficiency and correctness in absence of segmentation artifacts.
Second, we sent photo-realistic images through a run-of-the-
mill background subtraction to simulate real-world RGB
cameras. Third, we used depth images to prepare reconstruc-
tion for depth sensors such as the Microsoft Kinect.

For offline tests with real-world data, we modified the
above setup. Namely, we exchanged pre-rendered image se-
quences with pre-recorded footage of the real-world robot cell.
Moreover, we replaced ground-truth virtual camera paramete-
ers with real-world ones. A camera calibration approach as
described in [14] provided an estimate for these. Remaining
software and hardware parameters were not changed.

In contrast to the preceding experiments, we evaluated
online reconstruction in a different testing environment: Four
wall-mounted consumer webcams of type Logitech C920 pro-
vide Full HD image input within a cubic experiment cell.
Again, standard background subtraction segments input im-
ages. As a mockup application, an optimized distance query
evaluates the output octree to adjust the processing speed
of a simulated machine in proximity of intruding objects.

All three tests ran on a mid-level desktop PC with 16GB of
RAM and a modest Core i5-2400 quad core processor. Recon-
struction always terminated on pixel level detail, equivalent
to $1024^3$ voxels. Real-time limits were disabled to allow for
meaningful performance readings. Over each experiment run,
we measured key parameters, such as milliseconds per recon-
struction and number of octree nodes. Table 1 holds these
measurements. Resulting reconstruction times satisfy our ini-
tial demand of 100ms per frame at high-resolution input and
output. Consequently, our multi-camera reconstruction
algorithm is both efficient and precise.

On further evaluation, synthetic and offline experiments
performed distinctively worse than the real-world scenario.
Reasons are twofold: Use of more cameras partially cancels
out reduced resolution, and more complicated as well as
faster-moving silhouettes cause more splits and merges.

The latter hints at a general problem with incremental
strategies: In all experiments, run-times increase by almost
a factor of two for fast-moving objects, such as a human
running inside the work cell. Here, updates have to rebuild
a large part of the octree. In order to evaluate the extent
of this effect, we performed another set of measurements on
the offline experiment. In particular, we artificially modified
playback speed of the recorded camera stream to increase
the number of changed pixels per reconstruction.

Table 1: Average experiment results over a fixed-
length test sequence. Left to right: Milliseconds
for reconstruction, quadtree building, and octree up-
dates. Number of octree nodes, merges and splits.
Results of incremental stress tests are given in Figure 5. Over the entire test sequence, about one fourth of all pixels correspond to moving objects. Under this consideration, experiments imply an almost linear correlation between small movement speeds, the number of modified octree nodes, and octree update times. With increasing movement speed, incremental calculations quickly become useless, and slowdown saturates at a complete per-frame rebuild. Both observations match with theoretical considerations for a moving sphere.

In practice, however, our reconstruction hits the external real-time limit and exits prematurely once incremental updates cannot keep up with scene changes. Consequently, object boundaries become coarser, but our algorithm still maintains conservativeness at a real-time frame rate. In terms of a safety application, this might even be desirable behavior: Extended object boundaries help to account for uncertainty in the future path of a fast-moving object.

In summary, our contribution excels in many aspects when compared to state-of-art online reconstruction: Our algorithm achieves real-time rates of 100ms per reconstruction for high-precision input and output. Results supply $1024^3$ voxel equivalent detail, 64 times the elements in popular $256^3$ voxel spaces. We attain named throughput on a mid-range desktop, whereas other approaches demand expensive hardware (PC clusters, high-end GPUs) for less detail. Conversely, our algorithm can exploit such hardware to further improve detail and run-times. Our reconstruction even is conservative and does not rely on imprecise voxel center tests for its efficiency. Should we nevertheless exceed our allotted time limit, our anytime strategy still yields a coarse result. Thus, we gracefully handle unexpected system load or fast-moving objects. Regular algorithms fail under same conditions. Finally, knowledge-based refinement significantly reduces artifacts in comparison to purely geometric reconstruction.

5. CONCLUSION

We have introduced an approach that allows for efficient and precise multi-camera reconstruction of objects within some robot workspace under real-time, anytime, and conservativeness requirements. We accomplished our goal by representing both input and output in hierarchical data structures, and by iterative refinement of all data structures over consecutive time steps. Low-level optimizations and knowledge-based criteria increased efficiency and precision of our reconstruction. We have empirically verified our results with synthetic and real-world test cases. In contrast to state-of-art solutions, our contribution achieved real-time frame rates with high output precision even for input from multiple Full HD cameras and on commodity hardware.

Our future investigations diverge into various directions: A GPU port alongside a warp-optimized algorithm (e.g. as in [4]) seems worthwhile. Alternative data structures could perhaps represent segments and volumes at a more coarse, algebraical level. Ultimately, we plan to integrate our reconstruction algorithm into real-world robot-human collaboration, such as in home robotics or industrial manufacturing.

6. REFERENCES