This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/LRA.2015.2512958

Hierarchical Voxel Block Hashing for Efficient Integration of Depth Images

Olaf Kähler, Victor Prisacariu, Julien Valentin and David Murray

***Abstract*—Many modern 3D reconstruction methods accumulate information volumetrically using truncated signed distance functions. While this usually imposes a regular grid with fixed voxel size, not all parts of a scene necessarily need to be represented at the same level of detail. For example, a flat table needs less detail than a highly structured keyboard on it. We introduce a novel representation for the volumetric 3D data that uses hash functions rather than trees for accessing individual blocks of the scene, but which still provides different resolution levels. We show that our data structure provides efficient access and manipulation functions that can be very well parallelised, and also describe an automatic way of choosing appropriate resolutions for different parts of the scene. We embed the novel representation in a system for simultaneous localisation and mapping from RGB-D imagery and also investigate the implications of the irregular grid on interpolation routines. Finally we evaluate our system in experiments, demonstrating state-of-the-art representation accuracy at typical framerates around $100\,\mathrm{Hz}$, along with 40% memory savings.**

***Index Terms*—SLAM; Mapping; RGB-D Perception**

I. Introduction

SPURRED by the ready availability of depth sensors and massively parallel processing, computing rich 3D models of scenes has become a fundamental building block in many modern computer vision and robotics applications. KinectFusion [1], [2], which uses RGB-D imagery as input, is a widely acclaimed exemplar, whereas others methods, e.g. [3], [4] compute dense geometric data from visual imagery alone. While representations ranging from point clouds, via meshes, to combinations of geometric primitives have been proposed for storing such rich 3D models, one of the most successful recent approaches is based on volumetric representations of signed distance functions. SDFs date back to the seminal work of Curless and Levoy [5], but have become popular recently thanks to efficient parallel implementations.

The use of volumetric representations immediately raises the question of how to choose the discretisation grid to achieve both accurate and memory efficient 3D models. One aspect is that with a naive representation, memory requirements grow linearly with the *volume* that is represented, rather than with the actual complexity of the *surface*. A number of works have recently been published trying to address this issue, exploiting the fact that only the region around the actually observed 3D surface has to be stored. In one strand of work, the scene is subdivided into either *patch volumes* [6] or a *plane plus bump-heights* representation [7], both of which provide a compact representation based on local submaps. In another, tree-based data structures are investigated [8], [9], [10], mostly as a means of accessing the stored data near the surface efficiently. Likewise [11], [12] subdivide the space into a sparse set of sub-blocks and access them efficiently with hash functions.

Most of these approaches from the computer vision and robotics communities are still based on a fixed voxel grid with uniform resolution. Adaptive representations of volumetric data have long been known in the graphics community [13], but they are problematic when the structure has to be both accessed and modified in real time, as typical for simultaneous localisation and mapping. The aforementioned submapping methods [6], [7] could in theory be adapted to deal with submaps of different resolutions, capturing different parts of the scene at different levels of detail — however no such workable system has been presented so far. In an earlier work [14] and one of the tree-based approaches [10], the 3D information is accumulated in a multi-resolution 3D data structure, however the data is kept multiple times simultaneously at the different resolutions, and coarse information is then used to regularise the finer levels.

The key contribution of this paper is an adaptive representation of the 3D space, using higher resolution for parts that require more detail and coarser, more memory efficient representations for parts that do not require this detail, as illustrated in Figure 1. Our representation also comes with (i) efficient access methods for the integration and extraction of data, as well as (ii) efficient parallel methods for adjusting the resolution online. It is demonstrated to run at around $100\,\mathrm{Hz}$ on a Nvidia Titan X GPU and to save over 40% of memory compared to a fixed grid.

A. Outline of our Approach

Our system draws on ideas from many of the cited works. At its core, the 3D world is modelled using a truncated signed distance function (TSDF). Within a truncation band $\pm\mu$ around the 3D surface, we store a signed distance $F(\mathbf{X})$ from the surface for any 3D point $\mathbf{X}$. The zero-level set $\{\mathbf{X}|F(\mathbf{X}) = 0\}$ is the set of points that reside exactly on the surface. Outside the truncation band, $F(\mathbf{X})$ is clipped to a maximum absolute value. Of course F has to be discretised on a computer. Traditionally it is stored volumetrically by sampling on a grid of voxels with a fixed voxel size s.

Manuscript received: August, 31, 2015; Revised November, 25, 2015; Accepted December, 16, 2015.

This paper was recommended for publication by Editor Cyrill Stachniss upon evaluation of the Associate Editor and Reviewers' comments.

*This work was supported by the UK's Engineering and Physical Science Research Council [grant number EP/J014990].

All authors are with the Department of Engineering Sciences, University of Oxford, Oxford, UK {olaf,victor,dwm}@robots.ox.ac.uk, julien.valentin@eng.ox.ac.uk

Digital Object Identifier (DOI): see top of this page

Copyright (c) 2016 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/LRA.2015.2512958

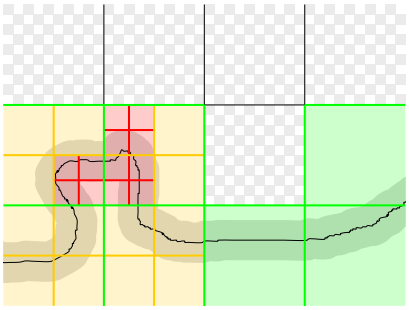


Fig. 1. Representation of the truncation band (shaded grey) of a TSDF using our hierarchical hashing data structure: The green blocks are represented at a coarse resolution, the yellow and red blocks at a successively finer one. The greyed out areas do not need any allocated blocks at all, as they are outside the truncation band.

Given that we only have a *truncated* SDF, we can save substantial storage space by including only values within the truncation band $\pm\mu$ in the representation. The works of [11], [12] achieve this by splitting the 3D space into blocks of $8 \times 8 \times 8$ voxels and indexing these blocks efficiently using a hash function. Compared to similar approaches indexing blocks with a tree structure [8], [9], [10], the hash function completely bypasses the time-consuming tree-traversal, but looses the inherent hierarchy in the representation. We draw on ideas from both of these worlds, resorting to the efficient hash index while retaining the ability to represent parts of the scene at different resolution levels. We will explain this and discuss our data structure in Section II.

To integrate new geometrical information, each incoming depth image is converted to a local TSDF and added to the weighted sum of previous TSDF values. While this is in line with previous works, we will present the specific extensions to cope with the hash data structure and with the resolution hierarchy in Section III. We also propose a method to select a resolution level for individual blocks in Section IV. We assume that a tracking step localises each incoming depth image before the integration, which is fundamentally unchanged from previous works [1]. This step is not discussed in greater detail. However, any such localisation will require extracting depth or colour images from the implicit representation of the TSDF which is done by raycasting. We detail the steps that are specific to the adaptive resolution in Section V, and in particular we address issues that arise at the boundaries of blocks that are stored with different resolutions. Finally we show some real world results and perform an experimental evaluation of our system in Section VI, and draw conclusions in Section VII.

II. Hierarchical Representation

The design imperatives for our representation of the truncated signed distance function $F(\mathbf{X})$ are the provision of locally varying discretisation grids and efficient access methods. From previous works [14], [9], [10], [8] it is clear that we are primarily interested in a narrow sized hierarchy ranging in resolution from a few millimetres to a few centimetres. Coarser resolutions would offer no meaningful information about $F(\mathbf{X})$, whereas finer resolutions would store little more than sensor noise.

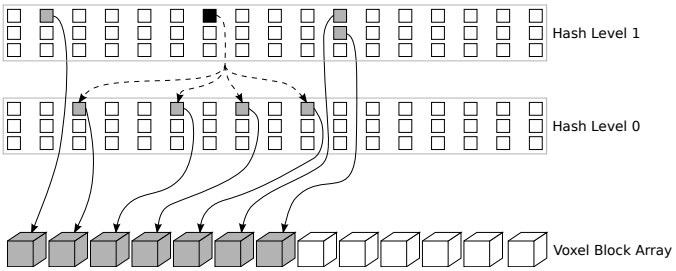


Fig. 2. Hierarchical hashing data structure: The filled (grey) entries in the hash table buckets point to individual entries in the voxel block array. The black entry in the hash table indicates that further information is stored at a finer resolution level.

While trees are very well suited for hierarchical representations, storing a fine resolution is particularly wasteful as it requires unnecessarily deep trees. In contrast, voxel block hashing [11], [12] avoids any such overhead, but is not natively suited for hierarchical representations. We therefore propose a novel representation that stores a fixed number of L levels of a resolution hierarchy. As illustrated in Figure 2, a hash table provides efficient access to the data represented at each level l. The entries of the hash table contain pointers to data blocks of $8\times8\times8$ voxels each. Depending on the resolution level l, these voxels represent the TSDF with a resolution of $2^l s$, where s is the base size at the finest resolution level. Alternatively, and as shown in Figure 2, at coarse levels a special marker may indicate that additional information is to be found at a finer level, *without* explicitly pointing to that information. To access individual voxels, we first find the block $\mathbf{b}$ it resides in at the coarsest resolution level, and then compute a hash value according to [11]:

$$h(\mathbf{b}) = (h_1 b_x \oplus h_2 b_y \oplus h_3 b_z) \bmod H, \qquad (1)$$

where h_1, h_2 and h_3 are some predefined hash coefficients, H is the size of the hash table at this level and $\oplus$ is a bitwise XOR operation. This provides an index to one of the buckets of the hash table, which is the start of a linked list of entries falling into the same bucket. Each entry contains either the pointer to the voxel block array, storing the actual voxel information for this block, or the specific flag indicating that additional information is stored at a finer level. If this flag is encountered, the search continues at the next finer level, computing a new hash index and performing a lookup in the hash table for the next level. However, if no matching entry is found for a given block, we can abort the search, knowing that the accessed voxel so far has no observations stored within our representation.

The same procedure as above can be used to modify existing data at individual voxels without changing the layout of the hierarchy and blocks. We will elaborate upon the steps required for allocating new blocks to store the information from a new depth image in Section III.

III. Integration of 3D Data

Each incoming image I_D from the depth sensor is first aligned using a camera tracker, as in KinectFusion [1] or similar systems. This step is independent of the internal

Copyright (c) 2016 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/LRA.2015.2512958

representation of the 3D model and we do not discuss it in detail. Given the depth image I_D, the pre-calibrated intrinsic parameters $\mathbf{K}$ and the estimated camera pose $\mathbf{T} = (\mathbf{R}, \mathbf{t})$, we can then integrate the newly observed information into our representation. As in voxel block hashing [11], we first ensure that all required voxel blocks are allocated, and then we integrate the depth information as in KinectFusion [1].

During allocation, we consider each pixel $\mathbf{p}$ in I_D and create a 3D line segment $\mathcal{L}$ within the truncation band $\pm\mu$ around the measured depth, where the TSDF will be updated:

$$\mathcal{L} : \left[\mathbf{T}^{-1}\tilde{\mathbf{s}} \left(1 - \frac{\mu}{\|\mathbf{s}\|}\right), \mathbf{T}^{-1}\tilde{\mathbf{s}} \left(1 + \frac{\mu}{\|\mathbf{s}\|}\right) \right], \qquad (2)$$

where $\mathbf{s} = I_D(\mathbf{p})\mathbf{K}^{-1}\tilde{\mathbf{p}}$ and $\tilde{\mathbf{p}}$ and $\tilde{\mathbf{s}}$ indicate the homogeneous equivalents for the respective vectors. For each pixel, the corresponding line segment $\mathcal{L}$ passes through a set of blocks $\mathcal{B}$ at the coarsest level of our representation. As in the lookup procedure above, we compute the hash value for each element of $\mathcal{B}$ and check whether the block is associated with some corresponding voxel data already. If it is, no action is required, but if it is not, a block is allocated from a pool of voxel blocks and the hash table is modified accordingly. If the hash table at the coarsest level indicates that information about the voxel is present at a finer level, we proceed in the same fashion on the next hierarchy level.

For a parallel implementation on e.g. a GPU, we split this allocation step into two stages [12]. In the first stage, we mark the buckets in the hash tables that ought to be allocated and store the coordinates of the corresponding blocks. In the second stage, we modify the hash tables either by starting new linked lists at the marked buckets or by extending the existing lists. By splitting the allocation into these two stages and by maintaining pools for the empty voxel blocks and for the linked list entries in the hash tables, the overall process can be parallelised efficiently using only simple atomic operations and no critical code sections. Note that the allocation procedure also provides a list of observed voxel blocks that contain novel depth information. Once all required voxel blocks are allocated, we go through this list and, respecting the corresponding voxel sizes, integrate the new depth information. As in [1], this is done by projecting each voxel $\mathbf{X}$ into the depth image I_D to retrieve the observed depth $I_D(\pi(\mathbf{K}(\mathbf{R}\mathbf{X} + \mathbf{t}))$, where π normalises homogeneous 2D coordinates to inhomogeneous ones. If the voxel projects into the depth image and has a valid depth measurement, we update a weighted sum:

$$F(\mathbf{X}) := \frac{W(\mathbf{X})F(\mathbf{X}) + I_D(\pi(\mathbf{K}(\mathbf{R}\mathbf{X} + \mathbf{t}))}{W(\mathbf{X}) + 1}, \qquad (3)$$

$$W(\mathbf{X}) := W(\mathbf{X}) + 1, \qquad (4)$$

where W is stored alongside F and counts the number of observations integrated in each voxel $\mathbf{X}$. Colour information can be updated similarly if desired, but is omitted for brevity.

IV. Splitting and Merging

To benefit from the resolution hierarchy we have to define some basic operations for splitting a block from a coarse level to several refined blocks and for merging several of these refined blocks back to a combined block at a coarser level. Our data structure natively allows us to split a block on level l into 8 finer blocks at level $l-1$, and the corresponding merge operation reverts such a split by combining the information back at level l. To enable the splitting and merging, we will first discuss a criterion to decide whether a block has to be split or merged, then we investigate the associated maintenance operations on the data structure.

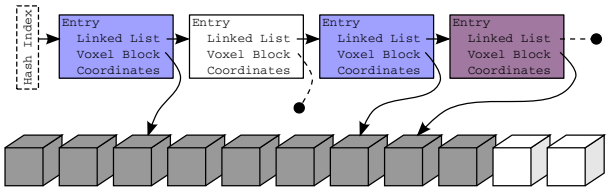


Fig. 3. Thread-safe maintenance of hash buckets for splitting and merging: When an entry gets deleted from the linked list, the pointer to the voxel block is marked as invalid (white entry). If it is at the end of the linked list (purple), it can also be removed safely. When adding an entry, atomic compare-and-swap on the voxel block pointer can be used to re-activate previously deleted entries (white) and to extend the linked list (purple).

A. Complexity Criterion

Determining whether or not to split or merge voxel blocks is a problem of model selection with well known probabilistic solutions [15]. However, model selection approaches generally find the model $\mathcal{M}^*$ that maximises $P(\mathcal{M}|\mathcal{O})$ for a set of observations $\mathcal{O}$. In our case, these observations are the images from the input sequence, and for an online framework with potentially unlimited input data, storing these observations is prohibitive. While there is clearly space for further research, we propose a heuristic that is solely based on the information accumulated in F. For each voxel block $\mathbf{b}$, we compute a complexity measure $c(\mathbf{b})$ as the determinant of the covariance matrix of surface normals within the block, thus measuring the roughness of the surface:

$$c(\mathbf{b}) = \det\left(\sum_{\mathbf{X}} \left(\nabla F_{\mathbf{b}}(\mathbf{X}) \nabla F_{\mathbf{b}}(\mathbf{X})^\top\right) - \left(\sum_{\mathbf{X}} \nabla F_{\mathbf{b}}(\mathbf{X})\right)\left(\sum_{\mathbf{X}} \nabla F_{\mathbf{b}}(\mathbf{X})\right)^\top \right), \qquad (5)$$

where we use $F_{\mathbf{b}}$ to denote the part of the signed distance function stored in block $\mathbf{b}$ and ∇ is the gradient operator. We schedule a voxel block $\mathbf{b}$ for splitting, when its complexity measure rises above a threshold $c(\mathbf{b}) > t_s$ provided it is not on the finest level of the hierarchy. Conversely, we schedule a voxel block $\mathbf{b}$ for merging, when it is marked as having been split previously (illustrated in black in Figure 2), none of its children are marked as being split, and the complexity of each of the children $\mathbf{b}'$ is below a threshold $c(\mathbf{b}') < t_m$. As we demonstrate in our experiments in Section VI, this heuristic already leads to good results, although we will discuss some of its drawbacks in the conclusions.

B. Data Structure Maintenance

Splitting a voxel block requires adding entries to the hash table. Again, we can avoid complicated synchronisation and

Copyright (c) 2016 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/LRA.2015.2512958

achieve a thread-safe parallel implementation using only simple atomic operations, as we illustrate in Figure 3. For each block $\mathbf{b}$ of the 8 new children, we first claim a new entry v from the pool of empty voxel blocks. We then compute the hash index $h(\mathbf{b})$ for the new block and traverse the entries of the corresponding linked list with an atomic *compare-and-swap* operation. The *compare* is checking, whether an entry already has a valid pointer to the voxel block array and the *swap* is simultaneously overwriting the first invalid pointer with v. If no invalid pointers are found, we claim a new entry from the pool of linked list elements, prepare it by setting its voxel block pointer to v, and, again using atomic compare-and-swap operations, try to find the current end of the linked list and replace it with a pointer to the newly allocated element. We implicitly exploit that, unlike during the allocation step discussed in Section III, no two threads will ever attempt to allocate the same voxel block.

Merging voxel blocks is less complex. The entry of the parent block is modified to point to a new empty voxel block. For each of the children we simply replace the pointer to the voxel block array with a flag indicating an empty block, as illustrated in Figure 3. If an entry is located at the end of the linked list in a hash bucket, we shorten this list accordingly. Shortening linked lists only at the ends again ensures that there are no race conditions. To update the TSDF data in the voxel blocks, we perform simple bilinear interpolation after splitting an element. For merging blocks we perform subsampling. Both of these can be trivially parallelised.

V. Raycasting

In Section III we assumed that the camera pose for incoming frames is computed by some tracker, e.g. one similar to the ICP algorithm in [1]. While the nature of this tracking step and whether or not it performs loop closure etc. is not relevant to our work, this step will generally require surface information to be extracted from the world model, and this surface extraction requires a raycasting step. More specifically we create a map of 3D points and surface normals and possibly surface colours from our implicit surface representation in $F(\mathbf{X})$, as seen from a given viewpoint with pose $(\mathbf{R}, \mathbf{t})$ and with the intrinsic calibration $\mathbf{K}$. The raycasting employed to compute this map takes steps along the line of sight of each pixel trying to find the point $\mathbf{X}$ where $F(\mathbf{X}) = 0$, i.e. the zero level set of the TSDF. As in [11], we pre-compute a plausible depth range for each pixel by projecting the bounding boxes of observed voxel blocks into the image and filling them with appropriate minimum and maximum depth values. Though raycasting largely follows previous works [1], [11], [10], special care has to be taken when reading interpolated values $F(\mathbf{X})$ for non-integral voxel positions $\mathbf{X}$.

For tri-linear interpolation at $\mathbf{X} = (X, Y, Z)$ in a regular grid, it is sufficient to accumulate the values from the 8 surrounding grid points, i.e. the corners of the cube that $\mathbf{X}$ lies in, weighted by the well known coefficients for linear interpolation. For irregularly spaced grids, such as at the boundary between two blocks of different resolutions, the computation of the coefficients is more complex. The interpolating function takes the form:

$$\begin{aligned} F(X,Y,Z) =& a_1XYZ + a_2XY + a_3YZ + a_4XZ \\ &+ a_5X + a_6Y + a_7Z + a_8. \end{aligned} \tag{6}$$

Each of the surrounding 8 grid points gives one equation of this form, altogether resulting in the linear system:

$$\mathbf{A} \begin{pmatrix} a_1 \\ \vdots \\ a_8 \end{pmatrix} = \begin{pmatrix} F(X_1, Y_1, Z_1) \\ \vdots \\ F(X_8, Y_8, Z_8) \end{pmatrix}, \tag{7}$$

with

$$\mathbf{A} = \begin{pmatrix} X_1Y_1Z_1 & X_1Y_1 & Y_1Z_1 & X_1Z_1 & X_1 & Y_1 & Z_1 & 1 \\ \vdots & & & & & & & \vdots \\ X_8Y_8Z_8 & X_8Y_8 & Y_8Z_8 & X_8Z_8 & X_8 & Y_8 & Z_8 & 1 \end{pmatrix}. \tag{8}$$

By arbitrarily shifting the coordinate system of the points such that $\mathbf{X}_1 = 0$, one of the lines trivially solves for a_8. We solve the remaining inhomogeneous 7×7 system using Gaussian elimination. While this provides an effective way of interpolating in irregular voxel grids, it is fairly costly and with a simple check we make sure it is only done when any two of the 8 surrounding grid points of an interpolation operation reside in different resolution levels.

VI. Experiments

To evaluate our proposed TSDF representation, we implemented it as parallelised GPU code using Nvidia CUDA, with source code available at `http://infinitam.org`. We compare a standard implementation of voxel block hashing from [12] with one using our hierarchical representation, where both obtain the camera poses via online ICP tracking throughout all of our experiments. The default settings are a base voxel size of $s = 2\,\mathrm{mm}$, providing a very high level of detail, and in our hierarchical representation we employ 3 resolution levels, so at the coarsest level the voxel size is $8\,\mathrm{mm}$. The truncation band of the TSDF is set to $24\,\mathrm{mm}$. While these parameters could be fine tuned for specific applications, this choice shall be sufficient for illustrating the main benefits of the proposed representation.

We use two test sequences `teddy` and `room` available from the project website, as well as for four additional sequences from the "7 Scenes" dataset [16]. Similar results are obtained on other sequences, but are omitted for brevity. In the following we present several experiments to investigate the accuracy of different representations (Section VI-A), to measure the memory savings resulting from our hierarchical representation (Section VI-B), and to compare the runtime to existing methods (Section VI-C).

A. Representation Accuracy

Sample results of qualitative experiments are shown in Figures 4 and 5. The reconstructions with a fixed resolution grid at $2\,\mathrm{mm}$ voxel size look virtually identical to the adaptive representation, where voxel sizes are varying from $2\,\mathrm{mm}$ to $8\,\mathrm{mm}$, and they show more details compared to the reconstructions at a fixed resolution of $8\,\mathrm{mm}$. As we also show,

Copyright (c) 2016 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/LRA.2015.2512958

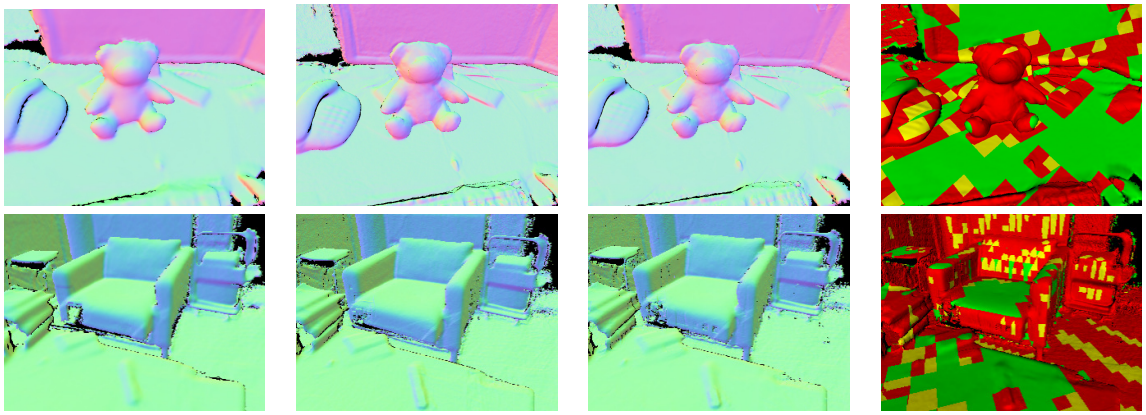


Fig. 4. Comparison of scene reconstructions with fixed grid and our adaptive representation. The selected refinement levels are shown on the right, where green areas are internally represented at a coarse resolution, yellow and red at successively finer resolutions. Note that fine details such as keys are not represented at the 8 mm grid and outlines such as for the remote control on the table appear blurred.

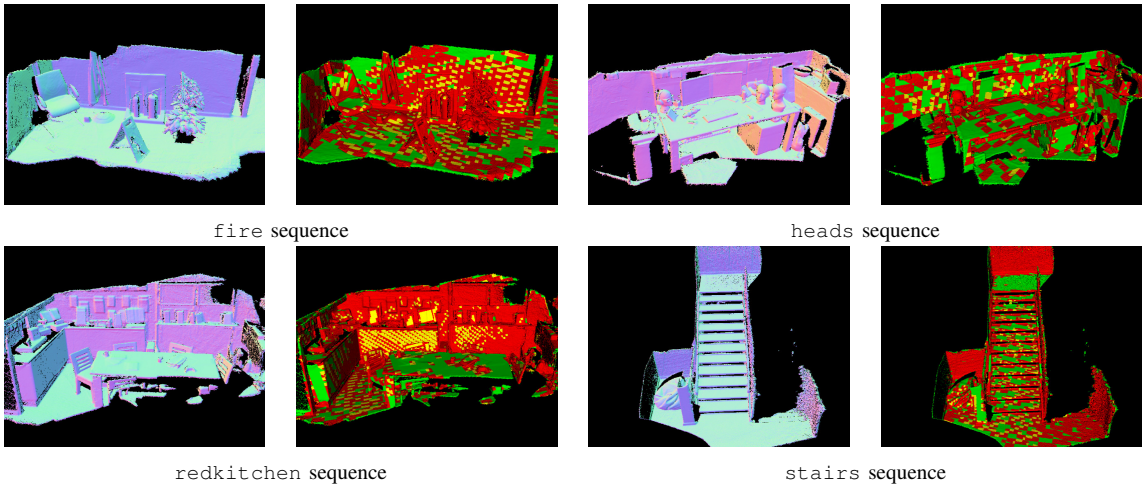


Fig. 5. Qualitative samples of the hierarchical refinement, taken on sequences from [16].

the selected refinement levels for individual parts of the scene are in line with intuition: largely planar areas like the desk, the table or the seating surface of the chair are represented at coarse levels, whereas highly structured areas like the teddy, keyboard and corners of objects are represented with higher resolutions.

To evaluate the accuracy quantitatively on real image data, we compare reconstructions with a fixed grid of 8 mm and our proposed adaptive resolution to a reconstruction with a 2 mm grid. We therefore generate a mesh from the 2 mm reconstruction, randomly sample 20 million points on this mesh and evaluate the SDFs from successive coarser reconstructions at these points. The SDF values give us the distance of points on the highly accurate reconstruction from the zero level set of the respective coarser reconstructions, and the average of these absolute differences is listed in Table I for different sequences. In all cases our adaptive representation outperforms a reconstruction with a fixed 8 mm grid and is close to the expected lower bound of 2 mm, underlining the qualitative results from above.

TABLE I
ERROR ACHIEVED RELATIVE TO A RECONSTRUCTION WITH 2 mm GRID.

	8 mm grid	adaptive
teddy	5.5 mm	5.3 mm
room	6.7 mm	3.4 mm
fire	3.1 mm	2.2 mm
stairs	9.5 mm	7.6 mm
redkitchen	6.4 mm	4.1 mm
heads	6.1 mm	5.6 mm

B. Memory Footprint

To assess the memory savings of our proposed methods, we run both a fixed grid reconstruction at 2 mm resolution and one using our hierarchical representation with default parameters as outlined above. In Table II we list the number of allocated voxel blocks for both representation, and for the two scenes teddy and room a plot of the memory footprint over time is also illustrated in Figure 6. These results show that our hierarchical representation saves about $40\% - 50\%$ of the voxel blocks that are required with a fixed discretisation grid, and these savings are consistent over the whole course of the sequences.

C. Runtime Performance

In many applications and particularly for robotics, real time performance is a critical requirement, and the typical frame rates for our CUDA implementation are given in Table II. These are measured on a Nvidia Titan X GPU, and for all

Copyright (c) 2016 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/LRA.2015.2512958

TABLE II
NUMBER OF VOXEL BLOCKS ALLOCATED USING EITHER A FIXED RESOLUTION GRID OR OUR PROPOSED ADAPTIVE GRID.

	2 mm grid	adaptive	saving	our fps	fps [12]
teddy	104195	59565	42.8%	216 Hz	376 Hz
room	415694	235485	43.4%	154 Hz	288 Hz
fire	367446	233731	36.4%	86 Hz	164 Hz
stairs	699840	390314	44.2%	80 Hz	130 Hz
redkitchen	609599	332756	45.4%	100 Hz	158 Hz
heads	168792	82925	50.9%	171 Hz	306 Hz

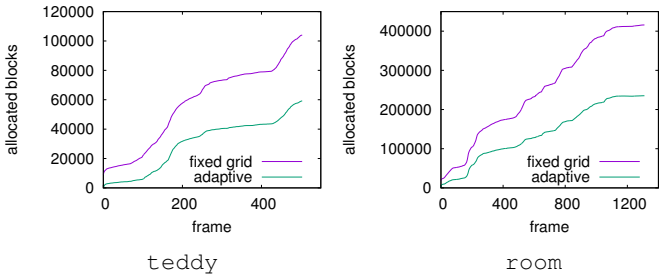


Fig. 6. Number of voxel blocks in use after each frame of two input sequences both with a fixed resolution grid and the adaptive grid.

cases they are far beyond real time performance on consumer grade graphics hardware. At roughly $80\,\mathrm{Hz} - 216\,\mathrm{Hz}$ this leaves sufficient processing power to perform other tasks on top. Compared to the publicly available implementation of [12] our framerates are lower, which is mostly due to the more complicated interpolation scheme employed during the raycasting as explained in Section V. The savings in the fusion step are negligible, as this step is highly optimised [12]. However, the additional overhead for adapting and maintaining our data structure is similarly negligible, as explained in Secion IV.

VII. CONCLUSIONS

We have investigated a representation of truncated signed distance functions based on an adaptively refined resolution hierarchy. This allows dense reconstruction systems to represent individual parts of a scene at a resolution that is adapted to the local surface characteristics, trading off memory efficiency versus representation detail. We avoid many problems typically associated with pointers across levels by allowing only a small, fixed number of refinement levels and by accessing each level independently using a hash lookup. This simplifies the data structure maintenance in parallelised implementations and allows an overall system running faster than real time on consumer grade graphics hardware, despite additional complexity for interpolation in the resulting non-uniform grid. We also presented a complexity measure for automatic selection of suitable refinement levels of individual parts of scenes, and we have found a typical memory saving around $40\% - 50\%$ compared to a fixed grid.

While the complexity measure we currently use appears to work reasonably well in our experiments, it currently ignores sensor characteristics such as noise [17]. These are not explicitly stored in our representation. Ideally we would like to employ some form of statistical model selection, and we will investigate ways of taking the reliability of the accumulated information in the TSDF into account when deciding on an appropriate resolution. Furthermore, if colour is important, e.g. for the tracker, the complexity criterion also has to take the surface texture into account. In future work we will also investigate integration methods that take information from multiple pixels to update each voxel, which should improve results at coarse levels. We would also like to investigate RGB-only methods such as [3], [4], for which our adaptive representation should be equally beneficial.

REFERENCES

[1] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohli, J. Shotton, S. Hodges, and A. Fitzgibbon, "KinectFusion: Real-time dense surface mapping and tracking," in *International Symposium on Mixed and Augmented Reality*, 2011, pp. 127–136.

[2] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, and A. Fitzgibbon, "KinectFusion: Real-time 3D reconstruction and interaction using a moving depth camera," in *User Interface Software and Technology (UIST)*, 2011, pp. 559–568.

[3] R. Newcombe, S. Lovegrove, and A. Davison, "DTAM: Dense tracking and mapping in real-time," in *International Conference on Computer Vision (ICCV)*, 2011, pp. 2320–2327.

[4] V. Pradeep, C. Rhemann, S. Izadi, C. Zach, M. Bleyer, and S. Bathiche, "MonoFusion: Real-time 3D reconstruction of small scenes with a single web camera," in *International Symposium on Mixed and Augmented Reality*, Oct 2013, pp. 83–88.

[5] B. Curless and M. Levoy, "A volumetric method for building complex models from range images," in *Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, 1996, pp. 303–312.

[6] P. Henry, D. Fox, A. Bhowmik, and R. Mongia, "Patch volumes: Segmentation-based consistent mapping with RGB-D cameras," in *International Conference on 3D Vision*, 2013, pp. 398–405.

[7] D. Thomas and A. Sugimoto, "A flexible scene representation for 3D reconstruction using an RGB-D camera," in *International Conference on Computer Vision (ICCV)*, Dec 2013, pp. 2800–2807.

[8] M. Zeng, F. Zhao, J. Zheng, and X. Liu, "Octree-based fusion for realtime 3D reconstruction," *Graphical Models*, vol. 75, no. 3, pp. 126–136, May 2013.

[9] J. Chen, D. Bautembach, and S. Izadi, "Scalable real-time volumetric surface reconstruction," *ACM Transactions on Graphics*, vol. 32, no. 4, pp. 113:1–113:16, July 2013.

[10] F. Steinbruecker, J. Sturm, and D. Cremers, "Volumetric 3D mapping in real-time on a CPU," in *International Conference on Robotics and Automation (ICRA)*, 2014.

[11] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger, "Real-time 3D reconstruction at scale using voxel hashing," *ACM Transactions on Graphics*, vol. 32, no. 6, pp. 169:1–169:11, Nov. 2013.

[12] O. Kähler, V. Prisacariu, C. Ren, X. Sun, P. Torr, and D. Murray, "Very high frame rate volumetric integration of depth images on mobile devices," *IEEE Transactions on Visualization and Computer Graphics (Proceedings International Symposium on Mixed and Augmented Reality 2015)*, vol. 21, no. 11, November 2015.

[13] S. F. Frisken, R. N. Perry, A. P. Rockwood, and T. R. Jones, "Adaptively sampled distance fields: A general representation of shape for computer graphics," in *Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, 2000, pp. 249–254.

[14] S. Fuhrmann and M. Goesele, "Fusion of depth maps with multiple scales," *ACM Transactions on Graphics*, vol. 30, no. 6, pp. 148:1–148:8, Dec. 2011.

[15] D. J. C. MacKay, *Information Theory, Inference & Learning Algorithms*. Cambridge University Press, 2002.

[16] B. Glocker, S. Izadi, J. Shotton, and A. Criminisi, "Real-time RGB-D camera relocalization," in *International Symposium on Mixed and Augmented Reality (ISMAR)*, Oct. 2013, pp. 173–179.

[17] C. Nguyen, S. Izadi, and D. Lovell, "Modeling kinect sensor noise for improved 3D reconstruction and tracking," in *3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT)*, Oct 2012, pp. 524–530.

Copyright (c) 2016 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.