

# Large Scale 3D Modelling via Sparse Volumes

E. Funk, A. Börner  
Department Information Processing of Optical Systems  
Institute Optical Sensor Systems  
DLR (German Aerospace Centre),  
Berlin, Germany  
{eugen.funk, anko.boerner}@dlr.de

## Abstract

Emerging transport technologies focus on the integration of i) interconnected transport systems and ii) real time information sources which enable automated transport management and autonomous vehicles to operate in urban environments. Since both goals rely on accurate measurements from static (lanes, crossings) and dynamic environments (traffic jam, construction sites), vehicles are required to perceive and to deliver this information. The key challenge is the storage, processing, and distribution of the data between all operating agents. This work addresses all of the mentioned issues and introduces a general spatial data modelling framework being capable of representing the geometry of the environment and its semantic information. The 3D geometry is modelled via small cubes (voxels), which are extended with additional information such as colour or object type (e.g. lane). In order to provide read and write access at superior efficiency, a sparse voxel octree (SVO) database is constructed. Since only the spatial hash of a SVO is stored in the efficient computer memory, environments of unlimited size may be represented. In summary, this approach enables i) efficient 3D map generation, ii) adaptive distribution of spatial data between agents and the central server, iii) concurrent updates of the world map from multiple agents.

# 1 Introduction

Modern 3D sensors such as laser scanners or (RGBD) cameras enable the measurements of accurate 3D samples from physical environments. This is an important step towards building models of indoor and outdoor areas for autonomous vehicles. In fact, autonomous vehicles are able to avoid obstacles, to plan a driving path or to pick up load only when the 3D measurements have been processed to information. The information processing challenge is aggravated by the circumstance that the sensors deliver large 3D point datasets. A stereo camera, working at 10 frames per second (fps) delivers 3 millions 3D points per second. A 3D database is required which enables efficient search of the measurements by a 3D coordinate  $(x, y, z)$ . Only then it is possible to filter the data, to extract surface models or to recognize pedestrians or cars. The 3D database is further expected to be filled by different vehicles and to provide information about obstacles or road boundaries.

In mobile visual robotics it is an accepted practice to group 3D samples into small cubic cells, *voxels*. This facilitates the storage and search. This work presents a voxel based database approach which enables to store theoretically unlimited 3D data and to access it efficiently. The access time is of magnitudes lower compared to state of the art techniques. The rest of the paper is structured as follows: Section 2 introduces state of the art techniques to handle 3D voxel data. Section 3 describes the proposed technique. Section 3.2 evaluates the hash-map based implementation of the voxel search with respect to access time. Section 4 demonstrates the application of the proposed database for 3D modelling from camera images. Section 5 concludes and provides aspects for further research.

## 2 Literature Overview

Random access of a 3D point and its neighbours is a difficult task, since the search complexity given a coordinate  $(x, y, z)$  depends on the number of samples in the database. Grouping the data to cubic cells on a regular grid and storing them in a 3D array enables fastest access and has been the state of the art technique for many years. The drawback of this approach is that the memory requirement grows with the cube of the space size. Representing a cube of size  $100 \times 100 \times 100 \text{ m}^3$  at  $1 \text{ cm}$  resolution, would require 3.7 TB of memory. A common approach to this issue is to structure the occupied cells with an octree ([1, 2]). Unfortunately, octrees suffer from a much higher access complexity  $\mathcal{O}(d)$  with octree-depth  $d$ . Each time an arbitrary voxel is accessed, either when iterating or performing random access, it is necessary to traverse the full height of the octree. A more successful method is to use paging, also known as *virtual memory* proposed by Bridson [3]. The flat array is divided into pages or *chunks* and only chunks including measurements or the model data are stored as blocks in the memory. This enables significant reduction of the memory requirements compared to the dense grid approach while maintaining the constant access complexity. Further, the paging approach enables the data to be stored on an external device such as the hard disk and to be read only if computations or visualization is performed in the local neighbourhood. However, depending on the size of the chunks, the memory overhead is still significant. Teschner [4] proposed to encode the coordinate of each voxel with a hash enabling constant time access  $\mathcal{O}(1)$  to the data independent of the dataset size. This approach allows to store data at nearly arbitrary resolution enabling huge models to be managed which is only limited by the size of the physical memory. However, generating unique hash values is a difficult task and applying this method on a high resolution voxel grid increases the probability to access wrong voxel elements. Further, fixed grid voxels do not allow to apply level of detail (LOD) visualization or multi scale 3D modelling which is the case when

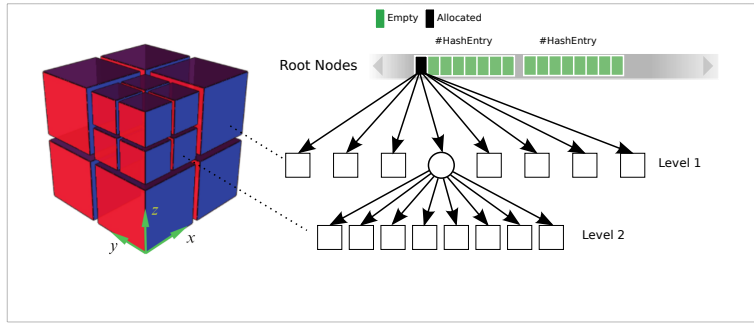


Figure 1: Multiple octrees are stored independently referenced by a hash.

octrees are used. In fact, LOD data structure enables to perform coarse visualization depending on the distance to the virtual camera and coarse 3D modelling depending on the expected error in a measurement [5].

The efficiency of the hash maps and the LOD capability of octrees motivated the development of a hybrid hashed octree. Internally, the approach applies the hashing technique from [4] to store references to internal octrees of small depth.

### 3 Method

#### 3.1 Data Structure

We propose to combine octrees with a hash table (Figure 1) leading to sparse voxel representation. The hash table is used to access the top level root nodes which further contain an octree in itself. Since the internal octrees are constructed with low depth (e.g.  $d \in \{1, 2\}$ ), this significantly decreases the access time complexity compared to standard octrees. To access the voxel at index coordinates  $(x, y, z)$ , we begin by computing the rootKey

```
int rootKey[3]={x&~((1<<d)-1), 1
               y&~((1<<d)-1), 2
               z&~((1<<d)-1)}; 3
```

where  $d$  is the depth of the internal octree,  $\&$  and  $\sim$  denote, respectively, bit-wise AND and NOT operations. At compile time, this reduces to just three hardware AND instructions. The shift by  $d$  makes sure that the coordinates  $(x, y, z)$  are represented by coarser values. For instance, applying an internal octree of depth  $d = 3$  with  $2^d = 8$  subdivision nodes in each dimension the space is divided in coordinate ranges  $\{[0, 7], [8, 15], \dots\}$ . This process is illustrated in Figure 2. Similar to [4], the rootKey is further processed to an *imperfect* hash.

```
unsigned int rootHash=((1 << N)-1)& 4
                  (rootKey[0]*73856093^ 5
```

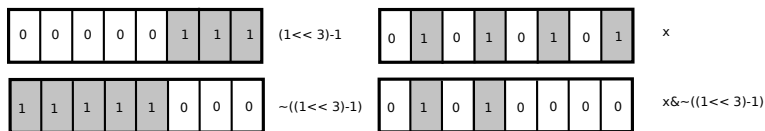


Figure 2: Illustration of the key generation steps.

Table 1: Access and insertion time for evaluated hash maps.

| Hash Map               | Insertion time | Access time |
|------------------------|----------------|-------------|
| std::map               | 1.7 $\mu s$    | 0.9 $\mu s$ |
| google:sparse_hash_map | 2.8 $\mu s$    | 0.6 $\mu s$ |
| google:dense_hash_map  | 0.6 $\mu s$    | 0.2 $\mu s$ |

```
rootKey [1]*19349663 ^
rootKey [2]*83492791);
```

6  
7

Here,  $N$  is the constant bit-length of the hash, the three constants are large prime numbers,  $\wedge$  is the binary XOR operator and  $\ll$  is the bit-wise left shift operator. Since the hash is imperfect, collisions occur so that two different coordinates are mapped by the same hash. The size of the hash map  $N$  influences the collision probability. In our experiments  $N$  was set to 32 leading to a collision of 70pm (collisions per million of distinct coordinates). A drawback of the hash is that it is not well suited for negative coordinates. Thus, when negative values in  $(x, y, z)$  are processed to a hash, collisions occur up to 50%. Such high rates require countermeasures which are undertaken by additional place holders to octrees (green cells in Figure 1). Each octree reference stores also its coordinate. When an octree is searched by coordinate given by the user, the resulting octree is validated. If the validation fails, next cell in the reference list is checked. Finally, an octree root node enclosing the searched coordinate is traversed to give the leaf node. The linear search within a hash block slightly reduces the performance but regarding the hash map efficiency, the performance gain is still immense. Table 1 shows the run-time evaluations on publicly available hash map implementations such as `std::map`, `google:sparse_hash_map` and `google:dense_hash_map`.

## 3.2 Hashed-Octree Performance

We have compared the performance of the proposed hashed-octree with a simple octree implementation and the recent implementation from [2]. Table 2 shows the achieved random access times for each approach. Please note, that the overhead resulting from accessing the octree leafs is not negligible. While the access time shown in Table 1 is  $0.2\mu s$ , the full voxel search takes  $0.45\mu s$ . Additionally, the right most column in Table 2 shows the maximum resolution a technique is capable to represent. Using an octree with depth  $d = 16$ , the maximum number of voxels is  $32768^3$  which corresponds to  $(327m)^3$  when each voxel represents a box volume of  $1cm^3$ . In contrast to this, the proposed hashed octree does not depend on the size of the modelled volume. In theory, cities or countries may be modelled by this approach if enough memory is available.

Table 2: Access time for different data structures.

| Method                    | Access time  | Max. resolution          |
|---------------------------|--------------|--------------------------|
| Octree ( $d = 16$ )       | 6.43 $\mu s$ | $32768^3 (327m)^3 @ 1cm$ |
| Octree [2] ( $d = 16$ )   | 2.55 $\mu s$ | $32768^3 (327m)^3 @ 1cm$ |
| Hashed-Octree ( $d = 2$ ) | 0.45 $\mu s$ | $\infty$                 |

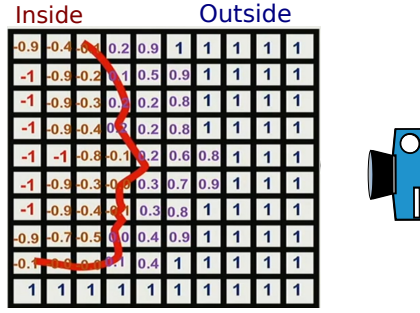


Figure 3: Implicit shape representation by the zero level set of voxels.

## 4 3D Modelling

We applied the proposed data structure to model a surface of an object with voxels. The application uses multi-view cameras images which have been acquired from a flying UAV. The method relies on the implicit approach from [6]. In contrast to standard surface modelling techniques with polygons, the surface is represented by a zero level set. Basically, each voxel receives a positive or a negative scalar value indicating its location inside or outside of an object. Figure 3 outlines this concept. During the reconstruction process each pixel in the camera image is processed to a 3D point  $p$ . The ray between the camera focal point and the 3D point is traversed updating the implicit values of the voxels. More specifically, only neighbours of  $p$  along the ray are updated. Figure 4a shows the updated neighbouring voxels in rose and the sample  $p$  in dark red. As proposed by [6] a one dimensional signed distance function  $f_i(r)$  (Figure 4b) is used to update the implicit value  $f^k$  of neighbouring voxels according to (1):

$$f^{k+1} = \frac{f^k \cdot w^k + f_i(r) \cdot w_i}{w^k + w_i} \quad (1)$$

$$w^{k+1} = w^k + w_i$$

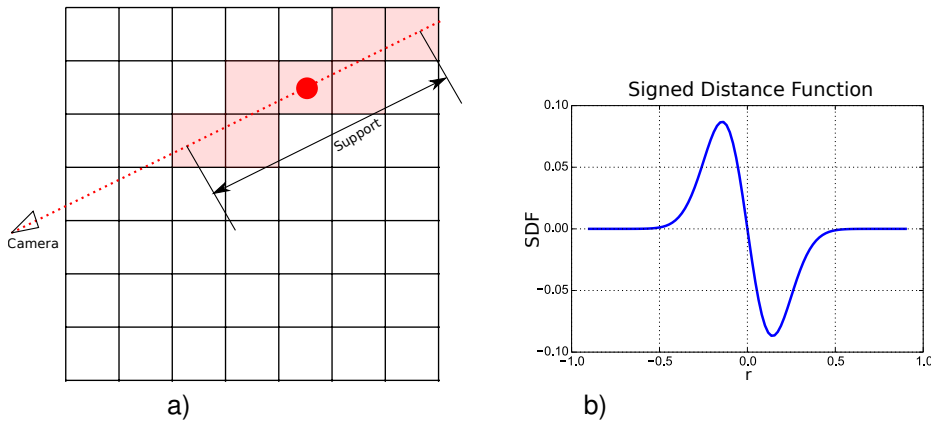


Figure 4: a) Volumetric fusion of new measurements along a ray. b) The signed distance function applied for recursive implicit shape updates.

Where  $r$  is representing the distance along the ray away from the 3D sample  $p$ , and  $w_i$  is the confidence weight of each sample. Note, that this process is of recursive nature and enables to update the 3D model as new measurements arrive. Thus, it is not required to wait for all measurements to become available. In robotics this enables to perform 3D modelling in real time.

Figure 5b) shows a reconstructed 3D model using the SDF-based shape representation. Figure 5c) contains the distribution of positive (red) and negative (yellow) voxels. Its interface encodes the surface. Figure 5d) illustrates the resulting octree structure of the constructed 3D model.

## 5 Conclusion and Outlook

A highly efficient data structure for voxel based 3D geometry has been presented. The approach enables to model arbitrary geometries and to modify them dynamically, for instance when new measurements become available. The approach has been applied for recursive 3D modelling from accurate multi view 3D reconstruction. This enables concurrent updates of the world map from multiple cameras. Due to the hierarchical nature of integrated octrees, the 3D geometry can be summarized at different scales enabling scale adaptive visualization and distribution of 3D data between agents.

Future work will demonstrate real time LOD visualization of the environment as it is being modelled and concurrent multithreading updates from multiple agents. Real time data from mobile stereo cameras is in the focus, which is unfortunately much less accurate than offline processed multi view 3D reconstruction. This prevents the implicit 3D fusion technique from direct application.

The mid term goal of the research project is an ubiquitous framework for intelligent vehicles. It will allow transparent delegation of 3D perception tasks such as object detection, 3D mapping and visualization for 3D data of any resolution and any amount.

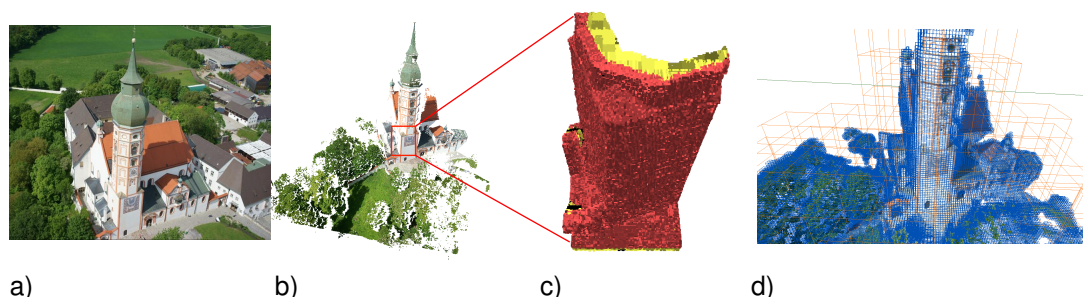


Figure 5: a) Photograph of the reconstructed object via multi-view images, b) 3D model with 22M voxels, c) the surface as the interface between positive (red) and negative (yellow) voxels, d) octree nodes as cubic boxes.

## References

- [1] 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," 2000.
- [2] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachiss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots*, 2013.
- [3] R. E. Bridson, *Computational Aspects of Dynamic Surfaces*. PhD thesis, Stanford, CA, USA, 2003.
- [4] M. Teschner, B. Heidelberger, M. Mueller, D. Pomeranets, and M. Gross, "Optimized spatial hashing for collision detection of deformable objects," *Proceedings of Vision, Modeling, Visualization (VMV 2003)*, pp. 47–54, 2003.
- [5] M. S. Floater and K. Hormann, "Surface parameterization: a tutorial and survey," in *Advances in Multiresolution for Geometric Modelling* (N. A. Dodgson, M. S. Floater, and M. A. Sabin, eds.), Mathematics and Visualization, pp. 157–186, Berlin, Heidelberg: Springer, 2005.
- [6] B. Curless and M. Levoy, "A volumetric method for building complex models from range images," in *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '96*, (New York, NY, USA), pp. 303–312, ACM, 1996.