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

Spatial 3D reconstruction received enormous interest in the last years. However, the goal to store, to process and to visualize the acquired data is still very challenging. Discrete voxel based representation techniques became state of the art in todays research approaches. These allow summary of redundant measurements and fast coordinate based access to the data leading to efficient volume computations. Unfortunately, representing the 3D space with a dense voxel grid requires huge amount of storage. Representing a volume of $100 \times 100 \times 100$m^3 with resolution of 1cm with a dense grid of 32-bit floating point values, results in a 3.8 TB storage requirement. This motivated many state of the art approaches to apply octrees to build sparse 3D volumes, where only the occupied voxels are stored. This however, increases the data access complexity from $\mathcal{O}(1)$ to $\mathcal{O}(d)$ with d as the depth of the octree, growing logarithmically when the volume or the resolution of the model is increasing.

In this work we propose to combine octrees with hash tables leading to sparse voxel representation well suited for efficient storage and fast data access common in 3D modelling computations. The hash table is used to access grid cells, which further contain an octree in itself. Since the internal octrees are constructed of much smaller depth e.g. $d_i = 1$, this dramatically decreases the access time complexity to $\mathcal{O}(d_i)$. For a standard octree with depth $d = 16$, this leads to a speed-up of factor 16. An additional advantage of the hash table approach is that the volume size is not limited and is suited for modelling huge environments.

1 Introduction

Emergence in 3D sensors such as laser scanners or 3D reconstruction from cameras enables today spatial information from physical environments to be acquired. This is an important step towards building models of indoor and outdoor areas for inspection, autonomous navigation and automated geometry aware monitoring. Given the fact, that most of the sensors deliver large datasets of 3D points, the storage, processing and visualization became very challenging. Many applications in robotics, medical imaging, exploration and animation focus on iterative or random access on the samples in order to reconstruct the observed surface, to perform deconvolution or to perform collision detection for physical based animation.

Random access to a point and its neighbours becomes a difficult task, since the search complexity given a coordinate (x, y, z) depends on the amount of samples in the database. Grouping the data on a regular grid and storing it in a flat array enables fast access to the cells 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, which prohibits to work with larger models. Representing a cube of size $100 \times 100 \times 100$ m^3 at $1cm$ resolution, would require 3.7TB 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 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 of constant size 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 visualization or multi scale 3D modelling which is the case when octrees are used. These two drawbacks motivated the development of a hybrid hashed octree, which uses the hash approach from [4] for very coarse spatial blocks to reference an internal octree of small depth to provide access on single voxels. The octree enables voxel queries at coarser resolution so fine details may be omitted during the rendering process if necessary.

The following section introduces some details about the proposed data structure and illustrate its run time capabilities. Section 2.3 introduces a sequential 3D modelling technique exploiting the dynamic data structure.

2 Method

2.1 Data Structure

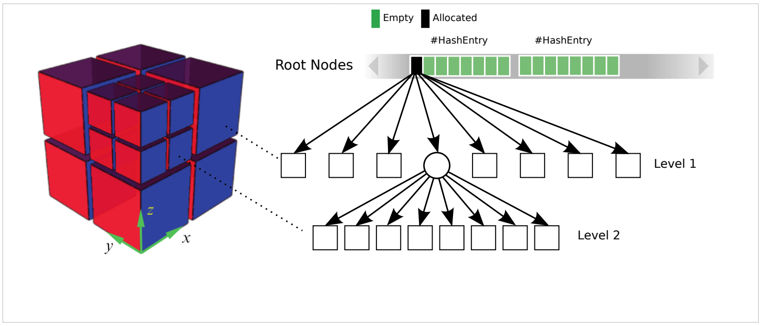


Figure 1: The proposed data structure. Multiple octrees are stored independently in the hash map.

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),
        y&~((1<<d)-1),
        z&~((1<<d)-1)};
```

where d is the depth of the internal octree, `&` and `~` 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

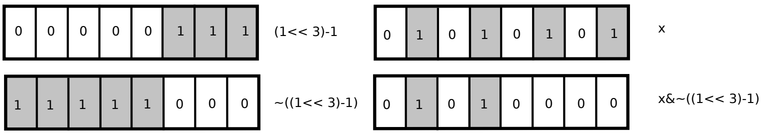


Figure 2: Illustration of the key generation steps.

octree of depth $d = 3$ with $2^d = 8$ subdivision nodes in each dimension the space is divided in coordinate ranges $\{[0,7],[8,15],\cdots\}$. This process is illustrated in Figure 2. Similar to [4], the rootKey is further processed to a *imperfect* hash.

```
unsigned int rootHash=((1 << N)-1)&
    (rootKey[0]*73856093^
    rootKey[1]*19349663^
    rootKey[2]*83492791);
```

Here, N is the constant bit-length of the hash map, the three constants are large prime numbers, ^ is the binary XOR operator and << is the bit-wise left shift operator. Since the hash is imperfect, collisions occur so that two different coordinates are mapped to the same hash. The size of the hash map N influences the collision probability. In our experiments N was set to 32 bit 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 introducing a block of octree pointers, as illustrated in Figure 1. During the insertion process, a new octree is allocated and inserted into the hash block. When a block is found from a hash code, the target leaf is searched in each of the octree root nodes in this block. 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 the simple `std::map`, `google:sparse_hash_map` and `google:dense_hash_map`.

2.2 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. Additionally, Table 2 shows the maximum available resolution for each

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

Hash Map	Insertion time	Access time
`std::map`	1.7 μs	0.9 μs
`google:sparse_hash_map`	2.8 μs	0.6 μs
`google:dense_hash_map`	0.6 μs	0.2 μs

Table 2: Access time for different data structures.

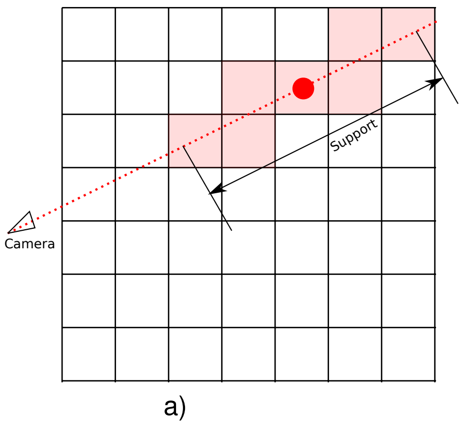
Method	Access time	Max. resolution
Octree ($d = 16$)	6.43 μs	32768^3 $(327m)^3$@$1cm$
Octree [2] ($d = 16$)	2.55 μs	32768^3 $(327m)^3$@$1cm$
Hashed-Octree ($d = 2$)	0.45 μs	∞

method respectively. 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.

2.3 3D Modelling

We applied the proposed data structure for implicit surface modelling from sequential measurements commonly provided from stereo or multi-view cameras. The method relies on the volumetric approach from [5]. In short, when a depth measurement is represented by a pixel, its corresponding 3D coordinate in the world frame is calculated. Furthermore, the ray between the camera focal point and the 3D point is traversed around the 3D sample while all voxels lying in the support are updated. Figure 3 shows the concept on a two dimensional plane. Similar to [5] a one dimensional signed distance function is applied to all the voxels lying in front and behind the measurement. The implicit value f and the weight w are both updated according to (1):

$$\begin{aligned} f^{k+1} &= \frac{f^k \cdot w^k + f_i(r) \cdot w_i}{w^k + w_i} \\ w^{k+1} &= w^k + w_i \end{aligned} \tag{1}$$



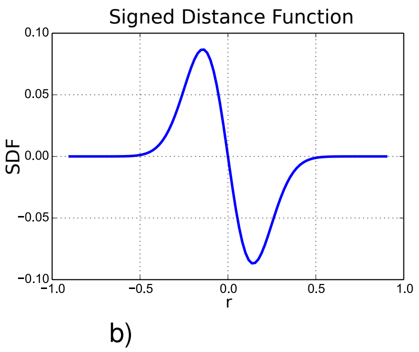


Figure 3: a) Volumetric fusion of new measurements along a ray. b) The signed distance function applied for recursive implicit shape updates.

where w_i is the confidence weight of the sample, and $f_i(r)$ the SDF shown in Figure 3b). Figure 4a) shows a reconstructed 3D model using the SDF-based shape representation. Figure 4b) contains the distribution of positive (red) and negative (yellow) voxels. Its interface encodes the surface.

Voxel based shape representation became a convenient method for shape rendering when triangle meshes are provided. Converting the meshes to the voxel based data structure enables to increase the rendering efficiency as has been successfully presented in [6]. Similarly, Figure 5a) shows an ap-

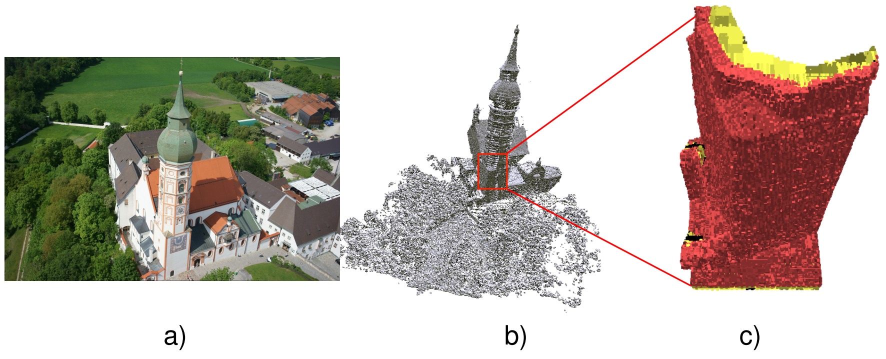


Figure 4: a-b) A 3D model with 22M voxels reconstructed from multi-view images. c) The surface is the interface between voxels with positive (red) and negative (yellow) values.

proximation of a polygon mesh with $12M$ voxels. The sparse voxel grid has been generated in approximately 1 second when four CPU cores have been exploited. Figure 5 b) shows the root octree nodes as blue boxes. Each of them contains a two-level octree representing the object surface via implicit leaf values as previously shown in Figure 4b).

3 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 are available. The performance is sufficient for real-time 3D reconstruction from depth images which will be investigated in the future. Further research will focus on real-time rendering depending on the camera orientation and position. An important aspect is the *out-of-core* extension of the hashed-octree to enable datasets of arbitrary size to be processed and partially stored on disk.

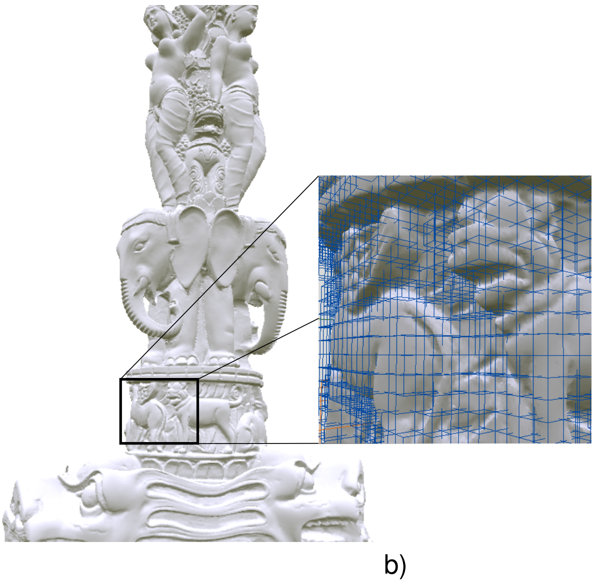


Figure 5: a): A model represented by 12M voxels. b): Octree root nodes are shown as blue boxes.

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, *Compuational Aspects of Dynamic Surfaces*. PhD thesis, Standford, 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] 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.

[6] M. G. Chajdas, M. Reitinger, and R. Westermann, "Scalable rendering for very large meshes," *WSCG 2014, International Conference on Computer Graphics*, 2014.