Incremental Division of Very Large Point Clouds for Scalable 3D Surface Reconstruction

Andreas Kuhn Helmut Mayer
Bundeswehr University Munich

Abstract

The recent progress in Structure from Motion and Multi-View Stereo as well as the always rising number of high resolution images lead to ever larger 3D point clouds. Unfortunately, due to the large amount of memory and processing power needed, there are no suitable means for manipulating these massive amounts of data as a whole, such as fusion by 3D surface reconstruction methods. In this paper we, therefore, present an algorithm for division of very large 3D point clouds into smaller subsets allowing for a parallel 3D reconstruction of many suitably small parts. Within our space division algorithm, octrees are built representing the divided space. To limit the maximum size of the underlying data structure, we present an incremental extension of the algorithm which renders a division of billions of 3D points possible and speeds up the processing on multi-core systems. As the proposed space division does not guarantee a density-based decomposition, we show the limitations of kd-trees as an alternative data structure. Space division is especially important for volumetric 3D reconstruction, as the latter has a high memory requirement. To this end, we finally discuss the adaptability of the space division to existing surface reconstruction methods to achieve scalable 3D reconstruction and show examples on existing and novel datasets which demonstrate the potential of the incremental space division algorithm.

1. Introduction

The recent major improvements concerning scalability of Structure from Motion (SfM) and Multi-View Stereo (MVS) algorithms allow for the generation of vast numbers of 3D points. SfM methods usually optimize the relative camera poses for the entire image set, which is possible because image registration can be done with as few as three points per image if the calibration of the camera(s) is known. Concerning relative camera poses, stereo methods can cope well with high resolution images, because only two images have to be processed at once, e.g., by semi-global optimization [10].

The next step in dense image-based 3D reconstruction pipelines is usually 3D surface reconstruction. Yet, the reconstruction from point clouds derived from multiple disparity maps, is not easily separable. Important competing classes for dense reconstruction are volumetric [4, 9, 11, 7, 14, 8, 15] and variational [23, 21, 12] reconstruction. Variational reconstruction produces globally optimal solutions with limited memory requirements, but is of high computational complexity. Volumetric methods have high memory requirements, but are suitable for fast unrestricted 3D surface reconstruction. Hence, they are discussed in this paper concerning their adaptability to process subsets of the point clouds generated by our proposed space division algorithm.

To avoid limitations in scalability and to achieve a high runtime performance, in general the reconstruction space cannot be considered all at once. Out-of-core methods can be used to deal with the problem of scalability by holding only important parts of the data in memory [3, 22]. Unfortunately, because of expensive and complex caching strategies, they are clearly limited concerning runtime performance. Even though fast streaming algorithms have been presented [1, 16], they do not perform well considering outliers or concerning manipulation of data with varying resolution / density common for general MVS.

An alternative scalable pipeline taking into account runtime constraints decomposes the reconstruction space into small subsets which can be merged afterwards. Additionally to the suitability for systems with small memory, it allows for parallel processing on multiple cores. Unfortunately, such a divide and conquer strategy can be of high algorithmic and computational complexity.

In this paper, we focus on the space division step for very large point clouds. Additionally, we demonstrate that 3D surface reconstruction methods exist which avoid a complex fusion concerning the merging of the subsets.

The idea of partitioning point clouds into small subsets is not new. Tobor et al. [19] present a divide and conquer method for multi-scale 3D reconstruction from large unorganized point sets. Their method divides a point cloud into an equally distributed number of points whose size can be explicitly specified. To this end, kd-trees [2] are used which recursively divide the point cloud considering the median of the point set for the axis with the maximum spread. In [20] this method was adapted by using the mean of the longest axis instead of the median. We show in this paper that volumetric space division is feasible even for a very large point clouds with a billion points, even though current implementations are limited to amounts of 3D points that fit into main memory. This is due to the fact that sorting the point clouds is computational complex when the data does not fit into main memory. Therefore, we examined and extended both methods [19, 20] to deal with billions of 3D points incrementally. Additionally, a method employing octrees is shown to perform better on very large datasets.

A straightforward incremental space division method based on octrees which can cope with big 3D data was presented by [14]. It dynamically splits the reconstruction space in eight subspaces if the number of points exceeds a given threshold. This is suitable for multiple cores or cluster systems, as the distribution of processing rises exponentially. In this paper we present a more efficient incremental division method which can be performed on a wide range of different architectures also with multiple cores.

The paper is organized as follows: In Section 2 we discuss octrees and two types of kd-trees as data structures for division of very large point clouds. In Section 3 an incremental algorithm is presented which allows for space division of point clouds which do not fit into main memory. The adaptability of space division to 3D reconstruction is discussed in Section 4. Finally, Section 5 presents experiments concerning the runtime behavior of the incremental algorithm and scalable 3D reconstruction before in Section 6 a conclusion is given.

2. Data Structure

When focusing on scalability, a discussion on the basic data structure for the decomposition of the reconstruction space is essential. Especially the strongly varying density in MVS-generated point clouds is challenging. In previous work, space decomposition by means of kd-trees has been proposed for space division of large point clouds [19, 20]. Hence, this geometrically irregular data structure is discussed first.

The strategy underlying **kd-trees** is a binary division of the data volume with respect to the density of the 3D point cloud. The root node represents the entire dataset, which in our case is a 3D point cloud. All nodes besides leaves have two child nodes which contain a (nearly) equal number of 3D points. Traversing to deeper levels, the point cloud is incrementally divided until a given usually small number of points is left in an individual leaf node. Hence, at all non-leaf nodes, the space is divided into (nearly) equal subsets.

For separating point clouds into two equal-sized point sets, an axis-parallel hyperplane is estimated. This includes calculation of the space dimension with the largest spread and sorting the point cloud according to the direction of the chosen space dimension for median calculation [19]. The hyperplane at the median coordinate exactly divides the point cloud into two point clouds containing an equal number of points ± 1 for odd-sized sets. While the calculation of a dominant space dimension is feasible also when taking into account our scalability constraint via bounding boxes, the calculation of the median does not scale well.

We are especially concerned with datasets that do not fit into main memory. Hence, sorting algorithms even with low complexity have to use out-of-core strategies. Unfortunately, in this case processing without an essential loss in computational efficiency is not possible. Substituting the median by the average (mean) of the 3D points [20] is possible, but it means that the density constraint can only be partly satisfied (Figure 1). This restriction will be discussed further concerning MVS-based 3D reconstruction.

Octrees offer an alternative data structure based on space division into regularly distributed voxels / cubes only partly dependent on the point cloud's density. They represent a cubic generalization of binary trees. While the root node corresponds to a limited cube, the eight child nodes represent eight equally sized subspaces which add up exactly to the space of the parent node. The side length of the child nodes equals half the side length of their parent node. The division strategy from parent to child node continues until the leaf nodes are reached. The depth of leaf nodes can be defined, e.g., by a maximum tree depth or the number of points belonging to the node.

Octrees have the advantage that there is no need for an individual separation of the point cloud on all depth levels as for kd-trees reducing runtime complexity. The number of nodes rises with 8^d, where d is the depth, while the number of kd-tree nodes rises with 2^d. The smaller access time of octrees is of high importance for space division of very large point clouds. On the negative side, octrees cannot be controlled directly by the point cloud density which can lead to relatively small point sets in leaf nodes (Figure 1).

In general, the space requirement rises exponentially with d, yet in turn leading to an access complexity logarithmic with d. While octrees seem to be especially suitable for fast scalable space division, there are some parameters which have to be defined: First, the data structure needs an initial limited space, represented by the root node (Figure 1). Second, the depth of the data structure has to be limited, especially for systems with small memory. The latter constraint also applies to kd-trees. In the following, an incremental space division algorithm which overcomes the need for limiting the depth is described and analyzed for MVS.

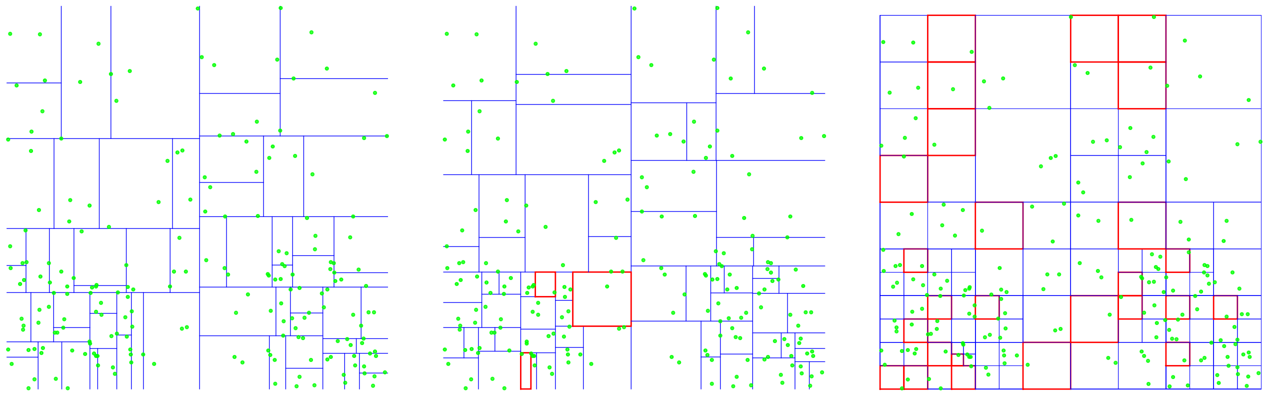
Figure 1. Space division of a point cloud with 200 points with varying density via a kd-tree with median division (left), a kd-tree with mean division (middle) and via an octree (right). Space is divided into clusters of points (example: maximum four points). While the kd-tree with median division leads to a density-based decomposition into a similar number of points (example: two to four), the mean division kd-tree and the octree cells can also contain single points (marked by red boxes). Octrees allow for fast division but result in many more spaces with only a few points.

3. Incremental Space Division Algorithm

As described in the previous section, octrees and kd-trees with mean division allow for space division of massive numbers of 3D points. Yet, extensions are necessary for both data structures to guarantee fast space division of very large point clouds. In this section, we discuss the strategy for space division tackling two significant problems:

1. Strongly varying density within the point cloud and
2. Point clouds that do not fit into main memory.

Both problems are common for real world data and their solution is essential for an efficient MVS-based 3D reconstruction. To this end, we first propose an incremental extension of kd-trees with mean division [20] for arbitrarily large 3D point clouds. The incremental algorithm has limited memory requirements as never more than one 3D point has to be held in memory. Second, we show that an underlying octree data structure is even more suitable for arbitrarily large 3D point clouds concerning the runtime of incremental space division.

Kd-trees with mean division require the mean of all 3D points in every node as well as a hyperplane corresponding to the space dimension with maximum spread (Section 2). To this end, the algorithm runs through all points for all kd-tree levels and calculates for every node the axis-parallel bounding box defined by the minimum and maximum 3D point. This corresponds to the positive and negative L_∞ norm over all 3D points. All points are individually tracked through the root node to the corresponding child nodes on level i, where $i = 0$ at the initial run. The mean of all points is calculated from the overall sum and a counter is incremented for all visited points. The resulting axis-parallel hyperplane needed for space division intersects the mean and is defined orthogonally to the largest spreading space dimension corresponding to the largest axis of the bounding box. After running through all points, the individual nodes hold information about the number of points; the points themselves are not saved as they do not fit into main memory.

This step is repeated until a given maximum kd-tree depth, which has to be limited. To this end, the maximum depth d_{max} is defined, with $2^{d_{max}}$ as factor for the necessary memory resources. A similar restriction of volumetric data structures for efficient processing of large 3D point clouds was proposed by [6]. In contrast to their approach, in our incremental algorithm all nodes only hold information about their binary existence, their centre point, a point counter and the dimension of the maximum spread in the case of kd-trees. This renders data processing possible, even if the data does not fit into main memory.

The incrementally called division is summarized in Algorithm 1, where P is the complete point cloud and N the nodes in the kd-tree. The function $calcHyperplane$ derives the hyperplane through the mean point and orthogonal to the space dimension with the maximum spread corresponding to the largest bounding box dimension.

At this point, all child nodes which contain an insufficient number of points are not of interest. The so-called important nodes at higher levels represent the volumes that contain a maximum number of points below a given size. In a final run, the 3D points are traced through the tree up to these important nodes and are written in subspaces.

One general disadvantage of the maximum depth limitation is that on depth d_{max} nodes can exist that exceed the threshold for maximum point size (Figure 2). In this case the algorithm is recursively repeated in the respective subspaces: The leaf nodes are used as new root nodes and the

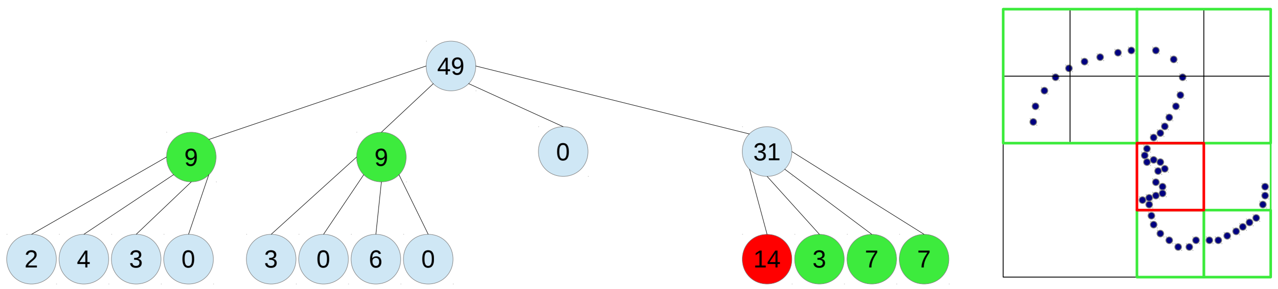

Figure 2. Dynamic iterative division of the reconstruction space (2D). The algorithm counts all points on all levels up to a given depth d_{max} (here: 2). For nodes with the number of points below a given threshold (here: 10), the subtree below it is cut off, i.e. the nodes are not visited. The 3D points are written in the final leaves (here: green and red). One leaf node (red) is above the threshold of 10 and is further processed as new root node.

Algorithm 1 Space division via kd-tree

```
1:  for all $d = 0$ to $d_{max}$ do
2:      for all $p$ in $P$ do
3:          $n \leftarrow getNode(p, d)$
4:          $n.counter \leftarrow n.counter + 1$
5:          $n.mean \leftarrow n.mean + p$
6:          if $n.pMin[x|y|z] > p[x|y|z]$ then
7:              $n.pMin[x|y|z] = p[x|y|z]$
8:          end if
9:          if $n.pMax[x|y|z] < p[x|y|z]$ then
10:             $n.pMax[x|y|z] = p[x|y|z]$
11:         end if
12:     end for
13:     for all $n$ in $N_d$ do                ▷ Go through nodes on level $d$
14:         $n.mean \leftarrow n.mean/n.counter$
15:         $calcHyperplane(n.mean, n.pMin, n.pMax)$
16:     end for
17: end for
18: for all $p$ in $P$ do                       ▷ Write data in final run
19:     for all $d = 0$ to $d_{max}$ do
20:         $n \leftarrow getNode(p, d)$
21:         if $n.counter < maxPoints$ then
22:             $writeToFile(p, n.filename)$
23:             break
24:         end if
25:     end for
26: end for
```

reduced point cloud as input point cloud, i.e., algorithm 1 is repeated on the resulting subspace point cloud. This incremental processing can be advantageous, e.g., for multi-core systems, where the multiple new spaces can be distributed. This will be explored in Section 5.

For kd-trees, the incremental method is costly, as in every depth the hyperplane from the corresponding point cloud has to be estimated. A loop over all points is needed for every level, resulting in long runtimes for very large point clouds. This is a problem of our space division extension based on [20] using kd-trees. Therefore, it is not reasonable to work with large d_{max} even though memory consumption would allow for it. We will show that using octrees can avoid this problem.

An **octree** for point clouds with an unknown number of points needs an initial bounding volume which covers the complete point cloud space (Section 2). To this end, for octrees our algorithm initially runs through all points and calculates the axis-parallel bounding box defined by a minimum and maximum 3D point over all points.

Because octree division is based only on the cube size of parent nodes, there is no need for hyperplane estimation for the octree levels. Hence, with a single second run through all 3D points the individual points can be directly traced to the defined maximum level. This is an essential advantage over kd-trees, as this step has not to be repeated for all levels, leading to reduced runtimes for space division. While visiting octree levels a counter is incremented in the corresponding nodes. To limit the memory resources, again a maximum depth is defined. For octrees the maximum depth d_{max} limits the memory size by a factor of $8^{d_{max}}$.

The third run for octrees corresponds to the last for kd-trees. Within this run points are streamed in a separate file for those nodes on the minimum level, which do not exceed the maximum point number (Figure 2). Hence, for octrees the complete point cloud has only to be read three times from the hard disk. The incremental called division is summarized in Algorithm 2. For nodes on the maximum depth which exceed the threshold the algorithm is recursively repeated with a strongly reduced number of 3D points.

The result of the algorithm is a separated set of point clouds for which the size can be limited according to given specification. The subsets can be merged by 3D surface reconstruction methods, even with high memory requirements.

Algorithm 2 Space division via octree

```
1:  for all $p$ in $P$ do                                    ▷ Initial run
2:      if $pMin[x|y|z] > p[x|y|z]$ then
3:          $pMin[x|y|z] = p[x|y|z]$
4:      end if
5:      if $pMax[x|y|z] < p[x|y|z]$ then
6:          $pMax[x|y|z] = p[x|y|z]$
7:      end if
8:  end for
9:  for all $p$ in $P$ do                                    ▷ 2. run
10:     for all $d = 0$ to $d_{max}$ do
11:         $n \leftarrow getNode(p, d)$
12:         $n.counter \leftarrow n.counter + 1$
13:     end for
14: end for
15: for all $p$ in $P$ do                                    ▷ 3. run
16:     for all $d = 0$ to $d_{max}$ do
17:         $n \leftarrow getNode(p, d)$
18:         if $n.counter < maxPoints$ then
19:             $writeToFile(p, n.filename)$
20:             break
21:         end if
22:     end for
23: end for
```

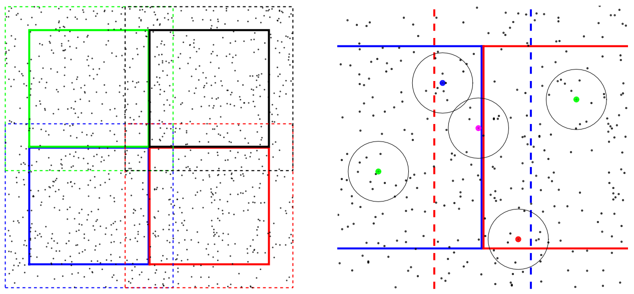
Figure 3. Illustration of the basic idea of [14], also used in [15]. The left image shows a point cloud and four neighboring subspaces with overlap (dashed boxes). The right image illustrates critical and non-critical optimization areas for two overlapping subspaces. To this end, five example points are emphasized with circles representing the local optimization area. The green points are not critical as they are totally contained in one subspace, whereas the other points appear in two subspaces. The optimization area of the blue point extends beyond the overlap (dashed) of the red subspace. Hence, it is removed from the red but kept in the blue subspace. Accordingly, the red point is kept in the red but removed in the blue subspace. The magenta point is between the red and the blue overlap and is kept in both subspaces. As the optimization leads to equal results in both subspaces, the points appear at very similar positions, even after optimization.

4. Adaptability to 3D Reconstruction Methods

Space division per se, as described in Section 3, does not solve the problem of scalable 3D surface reconstruction. Especially the conquer fusion part can be highly complex. Particularly, when performing global optimization in MVS considering, e.g., watertightness constraints, the same surface can be inconsistent for neighboring volumes. This can be solved by complex fusion of specific volumes. Vu [20] uses graph-cut-based surface merging of variationally optimized MVS surfaces. Yet, this does not scale well.

Kuhn et al. [14, 15] proposed a local optimization in MVS for sets of neighboring volumes. To avoid complex fusion, the volumes overlap taking into account the locally limited influence of the employed optimization (Figure 3). Thus, in border volumes non-equal surfaces can be directly cut off leading to consistent overall models with a very small redundancy. A more detailed description is given in [13]. Similar local optimization based on volumetric fusion [4] was proposed by Fuhrmann and Goesele [7, 8]. Even if the meshing step in [7] uses global tetrahedralization, the approaches are suitable for 3D surface reconstruction from split point clouds.

Because overlapping spaces are a prerequisite for a consistent fusion of MVS results by means of the above approaches, our scalable space division algorithm has been extended to deal with them. Although this leads to a higher computational complexity: When visiting child nodes in an octree, all coordinates of the eight (kd-trees: two) child nodes have to be checked for intersection. Even though heuristics can avoid checking all child nodes, the theoretical complexity of child traversing is eight times higher when considering overlap.

For kd-trees the overlap constraint is even more severe, because overlap for nodes cannot be simply derived from the fixed voxel sizes: The bounding box has to be calculated during mean calculation, further increasing the runtime.

In the following, the incremental space division algorithm presented in this paper is used to compare some volumetric MVS methods [11, 7, 8, 15] in all cases considering overlapping subspaces.

5. Experiments

This section describes experiments concerning runtime behavior as well as the effects of the separated subspaces on MVS-based 3D reconstruction. The parameter of maximum structure depth d_{max} (Section 3) is set according to the available hardware. For the parameter of maximum point cloud size in subspaces, we use 10 million points, because this is a feasible number to process on smaller systems. The overlap of neighboring subspaces is set to 0.1 of the subspace size. When having scale values for all points implying the local optimization area [8, 15], the overlap should be determined by the maximum local optimization area in the corresponding subspace (Figure 3).

The runtime is measured depending on the number of

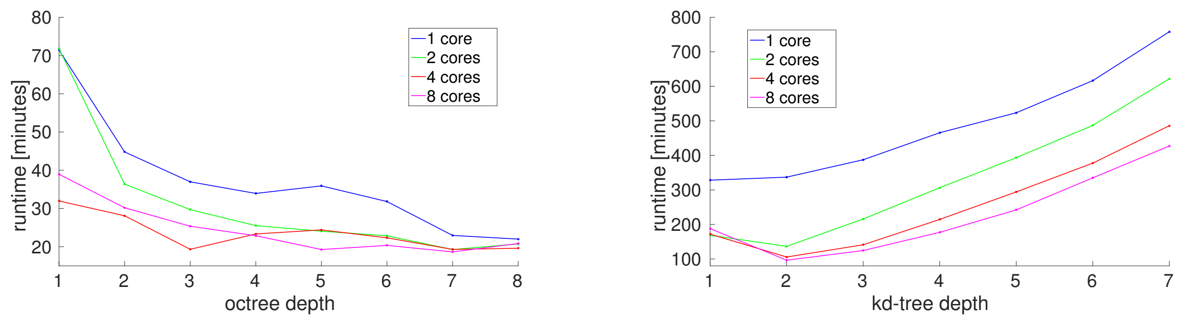

Figure 4. Runtime comparison of incremental space division by means of octrees (left) and kd-trees (right) up to a given octree depth and recursive splitting – please note the different scaling on the vertical axis. In general, octree-based space division is about five times faster. The experiments used 1-8 CPU cores. The incremental formulation allows a faster processing on multiple cores. The dataset of 1.5 billion points was divided fastest (18 minutes) by means of octrees.

available CPU cores. In the experiments we use 1, 2, 4 and 8 cores reading and writing data on one single Solid-State-Drive (SSD) without a special RAID configuration.

The point cloud used for runtime measurement is derived from SGM disparity maps of a registered image set [17] from 822 images captured from a UAV and from the ground leading to a point cloud with 1.513.711.066 points (Figure 7). This dataset is especially challenging, as it contains strongly varying distances to the reconstructed object and, hence, a point cloud of strongly varying density.

To demonstrate the effect of space division on MVS results, we give results for the popular Ettlingen dataset [18]. Even though this does not show the division efficiency for very large point clouds, it is is very suitable to present the effects of separated subspaces on 3D surface reconstruction.

5.1. Runtime Behavior

The limitation of the octree depth described in Section 3 is in conflict with the main challenge of division into subspaces which contain less than a given number of 3D points. To overcome this problem, the algorithm is incremental: Those leaf nodes with too many 3D points on the maximum octree depth are reprocessed (Section 3). Fortunately, this incremental processing does not necessarily decrease the space division runtime on current hardware configurations. For multiple leaf nodes on the maximum depth level with too many points the subspaces can be reprocessed independently. This can be very useful when incrementally dividing very large point clouds using multi-core systems.

To demonstrate the adaptability for current hardware configurations, we use an 8-core CPU system. We evaluate the runtime behavior, particularly examining if a strong limitation of the data structure depth implies an increased runtime. We limit the memory space to at maximum 32 GB. Considering eight times (8 cores) 8^n characters of 32 bytes, on all levels n, a maximum octree depth of $d_{max} = 8$ is possible. For kd-trees the parameter is three times larger. Only on a depth of $3d_{max}$ an equal number of subspaces is reached. Nonetheless, a large maximum depth is not suitable, as for all levels several loops over the complete point cloud are necessary (Section 3). Hence, we evaluate the depth for both trees for $d_{max} = [1, 8]$. Additionally, the number of cores is varied from 1 to 8.

The overlap constraint leads to an increased runtime (Section 4). Nonetheless, we only give results for division with overlapping subspaces as overlap is necessary for MVS. The graphs in Figure 4 show that limiting the octree depth does not necessarily imply higher runtimes and that our incremental space division allows for a fast division of very large point clouds. A maximum depth of $d_{max} = 3$ representing at maximum 512 subspaces is among the fastest configurations. The fastest configuration for octrees is about five times faster than for kd-trees. For the latter, the continuous rise of runtime depending on the core number beyond depth 2 confirms the assumption that kd-trees are not suitable for space division of very large point clouds.

Additionally, it is important to compare the runtimes with base line algorithms. Unfortunately, there are no suitable space division implementations publicly available. To process very large point clouds, the method by [20] is only feasible by means of the extension presented in this paper, which was shown to lack in runtime performance (Figure 4). For volumetric space division, an extension to Vrip [4] (pvrip) exists [5]. This space division algorithm divides the reconstruction space, in such a way, that the fusion volumes are limited within this area. Unfortunately, the 3D reconstruction is restricted to constant voxel sizes and is not capable to process point clouds with strongly varying density. This is an important aspect of the novelty of our incremental space division algorithm.

Nonetheless, to offer runtime comparison with an existing implementation, we compared our space division with the seminal out-of-core 3D surface reconstruction method [3]. Octrees are used for globally optimized volumetric 3D reconstruction by means of Poisson reconstruction [11]. To

3D reconstruction method	[3] ($d_p = 10$)	[3] ($d_p = 11$)	[3] ($d_p = 12$)	[3] ($d_p = 13$)	[3] ($d_p = 14$)	space division + [11]
Runtime (8 CPU cores) [mins]	8	11	16	51	170	9

Table 1. Runtime comparison of an out-of-core 3D reconstruction method [3] and our space division in combination with a similar surface reconstruction method [11]. The octree depth d_p has a fundamental influence on the runtime for [3]. As can be seen in Figure 5, only the result for $d_p = 14$ is comparable to the result of space division + [11]. I.e., division of the reconstruction space offers faster reconstruction without loss in details (Figure 5). The runtime of space division is negligible ($\ll$ 1 min).

process large 3D point clouds, individual levels of the octree are written to the hard disk.

The 3D reconstruction of the Ettlingen dataset combining Ettlingen30 and EttlingenFountain gives an impression of the runtime behavior without space division. Figure 5 shows two example images and the point cloud with about 50 million 3D points generated via MVS [10]. The image configuration causes large differences in the generated point cloud density. Hence, for detailed 3D reconstruction via Poisson reconstruction [11, 3] it is necessary to choose a large maximum octree depth $d_{poisson}$ (d_p). This in turn leads to high runtimes (Table 1).

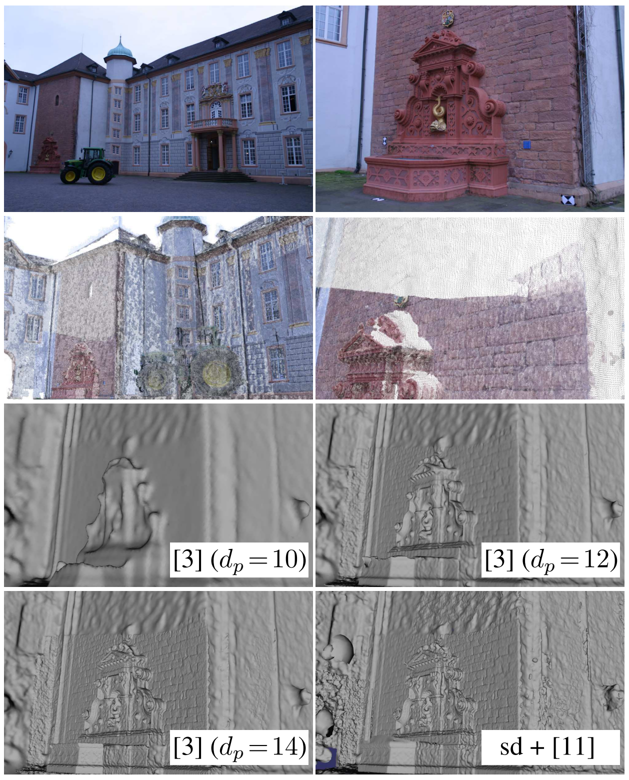

Figure 5. The two top rows show two example images from Ettlingen [18] and the MVS-generated point cloud demonstrating strongly varying densities on top of the fountain. The two bottom rows present results from out-of-core reconstruction [3] considering varying octree depths d_p. The right image in the bottom row shows the results from our space division (sd) with successive surface reconstruction [11]. In spite of the smaller octree depth ($d_p = 7$), it gives the highest resolution and the overall runtime comparable to the much less detailed [3] with $d_p = 10$ (3. row left image).

Please note, that in the small Ettlingen dataset the high runtimes are not only caused by the out-of-core strategy as especially a large octree depth d_p leads to high runtimes. Nonetheless, is has been shown that out-of-core methods per se cannot solve the problem of scalable 3D reconstruction especially for point clouds with varying density. Particularly, for the complete reconstruction space of the very large 3D point cloud with 1.5 billion points (Figure 7) a very large octree depth would be necessary. Unfortunately, even for smaller octree depths, the out-of-core method by [3] was not able to process the data within weeks. When dividing the space and using [11] the depth can be essentially reduced in the subspaces and reconstructed within hours.

We have shown that our incremental algorithm allows for space division of arbitrarily large 3D point clouds. In contrast to using kd-trees [20] and out-of-core methods [3] an essential runtime reduction is achieved. In the next section we finally analyze the suitability of the results of different surface reconstruction methods for fusion of subspaces.

5.2. 3D surface reconstruction

Section 4 discussed the adaptability of space division for existing MVS methods. In the following, we show 3D surface models reconstructed from subspaces. The subspaces were divided with an overlap of 10% cut off during fusion.

One important restriction of octrees which has to be discussed for 3D surface reconstruction is their inability to directly divide space according to density (Figure 1). Experimental evidence shows that in real world datasets the impact of this restriction is marginal. Even though relatively smaller numbers of points can occur in subspaces, they are very unlikely to be very small. In the experiments, we use a maximum of millions of points per subspace. This makes it very unlikely that the minimum number is below a couple of points. This in turn, does not negatively influence the resulting surfaces. Especially because of voxel overlap and the use of local optimization the restriction is close to irrelevant. Hence, in the following we only show results for octree-based space division, which is employed, because it is significantly faster as mentioned above.

First, we analyze the results for the Ettlingen dataset (Figure 5). Figure 6 shows results for a challenging part with overlapping subspaces from different MVS methods [11, 7, 8, 15]. Especially for global optimization with watertightness constraint [11] inconsistent surfaces are obtained. For [7] which uses a global tetrahedralization and [8] which

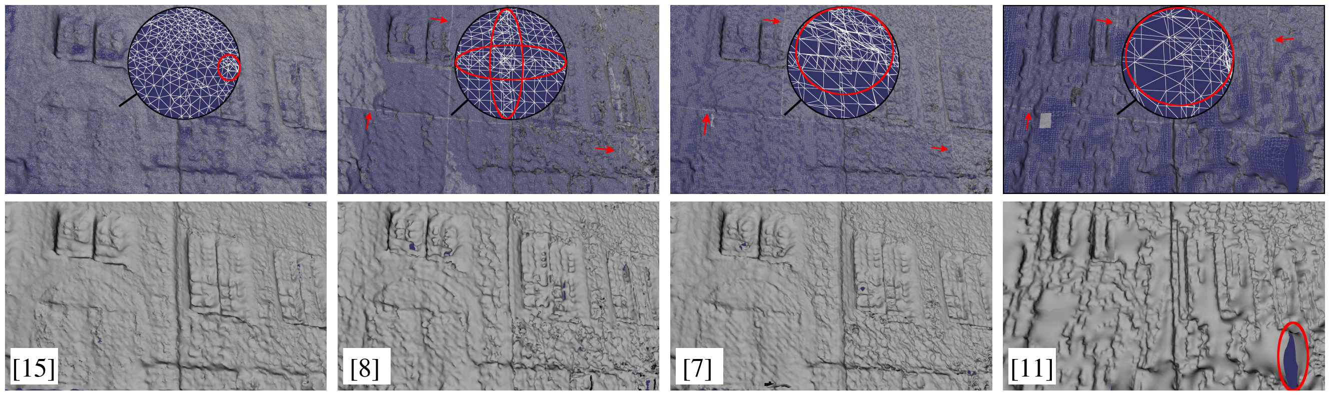

Figure 6. A challenging part from the Ettlingen30 sequence. Bottom: Slanted view of the surfaces. Top: (Zoomed) view of the corresponding triangle mesh. The global optimization of the subspaces by [11] causes strong inhomogenities (right column: marked red). When using (partly) local optimization [7, 8], the inhomogenities at the borders are not visible in the slanted views. They can only be seen on the hard borders in the top row (red arrows). Using only local optimization and meshing [15], the inconsistencies are avoided and reduced to numerical errors (left column).

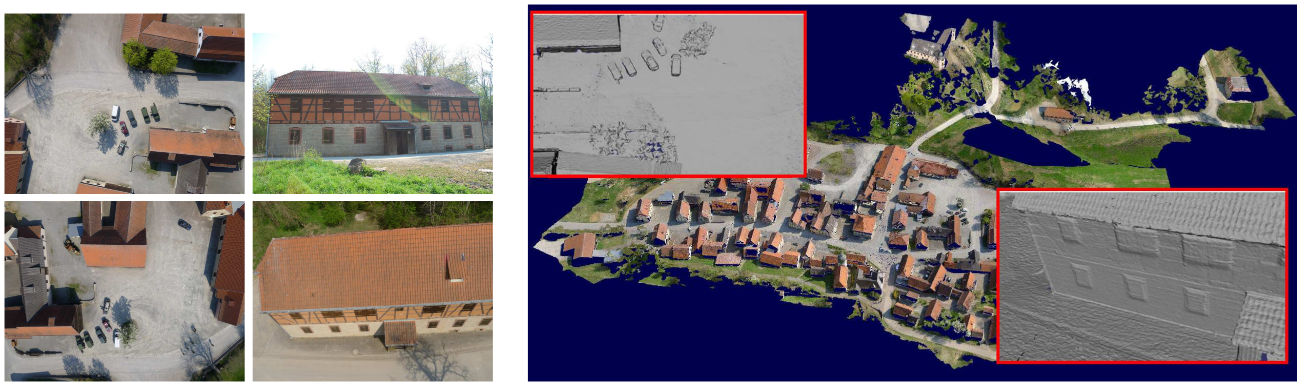

Figure 7. On the left four example images of a registered image set of 822 images are shown. The right part presents a screenshot of the colored surface model reconstructed by [15] from 1.5 billion points and shaded zoomed parts from the surface model demonstrating the preservation of small details. In spite of the space division, the 3D model has consistent surfaces.

employs fast volumetric meshing the volume borders are visible as grey lines in the visualization with triangles. Because of the local volumetric fusion, the errors are quite small and not visible in the shaded view. When using local optimization and meshing by [15], the differences also for the visualization with triangles are very small and can be attributed to numerical errors (small red circle). The space division runtime for this small dataset amounts to a couple of seconds.

Second, to demonstrate the reconstruction for a large data set, we processed the large dataset containing 1.5 billions points with the local-optimization-based MVS method [15] (Figure 7). Especially the zoomed areas (red boxes) demonstrate the configuration-dependent high density in specific areas. 3D surface reconstruction of this large point cloud was only possible with our incremental space division. The space was divided into over 600 subspaces, which were processed by MVS in a couple of hours on a cluster system. The local optimization in [15] leads to consistent surfaces in spite of space division.

6. Conclusion

In this paper we have presented an incremental space division algorithm for Multi-View Stereo reconstruction from very large point clouds. The incremental formulation extends existing space division methods to arbitrarily large point clouds. Furthermore, we showed that octrees allow a faster decomposition of the reconstruction space than kd-trees which are usually employed so far.

Based on the octree structures at no time more than one point has to be in memory for the division. Limiting the octree depth allows processing of point clouds of basically unlimited size. It also makes it possible to incrementally decompose the algorithm. We showed that this can be exploited to improve the runtime on multi-core CPU systems.

Finally, we have demonstrated the suitability for fusion of results from subspaces for various surface reconstruction methods.

References

[1] R. Allègre, R. Chaine, and S. Akkouche. A streaming algorithm for surface reconstruction. In *Eurographics*, 2007.

[2] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975.

[3] M. Bolitho, M. Kazhdan, R. Burns, and H. Hoppe. Multilevel streaming for out-of-core surface reconstruction. In *Eurographics*, 2007.

[4] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In *SIGGRAPH*, 1996.

[5] B. Curless and M. Levoy. *VripPack: Volumetric Range Image Processing Package*, 2010 (accessed September 1, 2015). https://graphics.stanford.edu/software/vrip/vrippack_roadmap.html.

[6] J. Elseberg, D. Borrmann, and A. Nuchter. Efficient processing of large 3d point clouds. In *ICAT*, 2011.

[7] S. Fuhrmann and M. Goesele. Fusion of depth maps with multiple scales. In *SIGGRAPH Asia*, 2011.

[8] S. Fuhrmann and M. Goesele. Floating scale surface reconstruction. In *SIGGRAPH*, 2014.

[9] M. Goesele, B. Curless, and S. Seitz. Multi-view stereo revisited. In *CVPR*, 2006.

[10] H. Hirschmüller. Stereo processing by semi-global matching and mutual information. *PAMI*, 30:328–341, 2008.

[11] M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson surface reconstruction. In *Eurographics*, 2006.

[12] K. Kolev, T. Brox, and D. Cremers. Fast joint estimation of silhouettes and dense 3D geometry from multiple images. *PAMI*, 34(3):493–505, 2012.

[13] A. Kuhn. *Scalable 3D Surface Reconstruction by Local Stochastic Fusion of Disparity Maps*. PhD thesis, University of the Bundeswehr, 2014.

[14] A. Kuhn, H. Hirschmüller, and H. Mayer. Multi-resolution range data fusion for multi-view stereo reconstruction. In *GCPR*, 2013.

[15] A. Kuhn, H. Mayer, H. Hirschmüller, and D. Scharstein. A TV prior for high-quality local multi-view stereo reconstruction. In *3DV*, 2014.

[16] J. Manson, G. Petrova, and S. Schaefer. Streaming surface reconstruction using wavelets. *Computer Graphics Forum (Proceedings of the Symposium on Geometry Processing)*, 27(5):1411–1420, 2008.

[17] H. Mayer, J. Bartelsen, H. Hirschmüller, and A. Kuhn. Dense 3D reconstruction from wide baseline image sets. In *15th International Workshop on Theoretical Foundations of Computer Vision*, 2011.

[18] C. Strecha, W. von Hansen, L. Van Gool, P. Fua, and U. Thoennessen. On benchmarking camera calibration and multi-view stereo for high resolution imagery. In *CVPR*, 2008.

[19] I. Tobor, P. Reuter, and C. Schlick. Multi-scale reconstruction of implicit surfaces with attributes from large unorganized point sets. In *SMI*, 2004.

[20] H. H. Vu. *Large-scale and high-quality Multi-view stereo*. PhD thesis, École des Ponts ParisTech, France, 2011.

[21] H.-H. Vu, P. Labatut, J.-P. Pons, and R. Keriven. High accuracy and visibility-consistent dense multiview stereo. *PAMI*, 34:889–901, 2012.

[22] K. Wenzel, M. Rothermel, D. Fritsch, and N. Haala. An out-of-core octree for massive point cloud. In *IQmulus Workshop on Processing Large Geospatial Data*, 2014.

[23] C. Zach, T. Pock, and H. Bischof. A globally optimal algorithm for robust TV-L1 range image integration. In *ICCV*, 2007.