Progressive Prioritized Multi-view Stereo

Alex Locher[1] Michal Perdoch[1] Luc Van Gool[1,2]

[1] Computer Vision Laboratory, ETH Zurich, Switzerland [2] VISICS, KU Leuven, Belgium

Abstract

This work proposes a progressive patch based multi-view stereo algorithm able to deliver a dense point cloud at any time. This enables an immediate feedback on the reconstruction process in a user centric scenario. With increasing processing time, the model is improved in terms of resolution and accuracy. The algorithm explicitly handles input images with varying effective scale and creates visually pleasing point clouds. A priority scheme assures that the limited computational power is invested in scene parts, where the user is most interested in or the overall error can be reduced the most. The architecture of the proposed pipeline allows fast processing times in large scenes using a pure open-source CPU implementation. We show the performance of our algorithm on challenging standard datasets as well as on real-world scenes and compare it to the baseline.

1. Introduction

Accurate 3D reconstruction from calibrated cameras is a long studied topic in computer vision. Multi-view stereo reconstruction offers an inexpensive alternative to costly laser scans, while providing highly accurate results [16]. Despite the many proposed solutions, none of it fully addresses the usage of Multi-View Stereo (MVS) in a user centric scenario. Systems like Arc3D [12], MVE [4],VisualSFM [21], Agisoft Photoscan [1] or Acute 3D [14] provide a user friendly interface to algorithms, allowing non-scientific users to reconstruct a large variety of objects and scenes from a set of images. The availability of new hardware platforms such as smartphones and Micro Aerial Vehicle (MAV) opened up whole new possibilities in 3D content acquisition [20, 18]. Highly dynamic and flexible Simultaneous Localization and Mapping (SLAM) systems deal with view selection and pose tracking in real-time and deliver a sparse point cloud for navigation or coarse visualization purposes. Unfortunately, when it comes to more accurate dense reconstructions, algorithms offer very low flexibility. Existing MVS methods were not developed with the interactiveness of a real-time system in mind, but rather as a batch processing step with a single resulting point cloud shown to the user at the end of processing. An immediate user feedback of the reconstruction's current state, which would be necessary for possible interventions such as taking additional pictures, is not available and the user has to run the algorithm to the very end in order to see his/her success. Coarse visualization methods such as triangulation of the Structure-from-Motion (SfM) points (e.g. Bodis et al. [2]) or low resolution filter based methods (e.g. Pizzoli et al. [13]) give a good insight into the currently reconstructed scene. But they are not able to cover fine details, and more importantly, do not reflect the status of the actual accurate dense reconstruction (which may give totally different results).

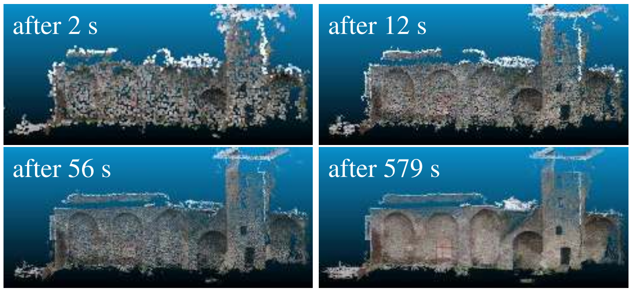


Figure 1: Progressive reconstruction of the citywall dataset (562 images). With increasing runtime, the resulting dense point cloud gets more accurate and covers finer details.

In contrast, we propose a MVS algorithm delivering a highly accurate and complete dense point cloud. The output point cloud can be visualized at any timepoint and improves with increasing runtime. As a user centric scenario is addressed, intermediate point clouds are ”visually pleasing”. A scene, where e.g. the door handle is reconstructed in every fine detail but the house around it is not covered at all, would not only look awkward, but also the algorithm might waste its resources on details, the user might not even be interested in. Therefore, we explicitly analyse the effective scale of input images and maintain a homogeneous resolution within the 3D point cloud. In order to optimize the available resources, complex and important scene parts are prioritized over trivial regions. Moreover, the algorithm allows prioritization of regions based on an explicit or implicit user input (e.g. current user's viewpoint).

1.1. Related Work

In the following section we discuss the most related work, focussing on progressive and hierarchical reconstruction and the handling of the effective input image scale.

MVS algorithms can be roughly classified into three categories, depth map-, voxel- and patch based approaches. Algorithms in the first category, estimate a pairwise dense depth map from the input images and create a global model. Voxel based approaches represent the scene in a regular 3D grid and are able to incorporate and accumulate measurements. The geometry is either expressed as an occupancy function or as a signed distance function to the closest surface. Finally, patch based approaches represent the surface as a set of oriented patches.

One of the best known patch based MVS algorithms is PMVS [6]. Starting from the calibrated scene, it generates an initial set of oriented patches by guided matching. The scene's geometry is successively grown by multiple iterations of expansion and filtering steps. PMVS delivers highly accurate scene representations, at the cost of computation time. The follow-up work CMVS [5] clusters the scene to multiple independent sub-problems, which can be processed by PMVS individually. While being able to reconstruct a lot of details, the algorithm does not handle the effective scale of input images and is not progressive.

Jancosek et al. [11] presented a system capable of reconstructing large scenes with a patch based algorithm similar to PMVS. The scene is built gradually by growing patches and only images with similar scale and scene coverage are considered. In a final step, a filtered mesh is recovered by Markov Random Field optimization [3]. While the algorithm handles input image scale variation, it strictly reconstructs on a single selected scale level.

Goesele et al. [7] presented a MVS algorithm for scene reconstruction out of community photos by creating individual depth maps out of which a mesh is extracted. The algorithm implicitly handles input image scale in the view selection, but can not be used in a progressive manner. Hornung et al. [9] formulated the reconstruction as a graph-cut minimization on a volumetric grid. A coarse visual hull is refined in a hierarchical pipeline. While this leads to a progressively increasing 3D accuracy, the algorithm relies on a visual hull, limiting the application range.

Yuan et al. [22] presented an interesting work, allowing to integrate new images into an existing 3D reconstruction. Hereby input images of an existing model are arranged in a view sphere and new images with patches are integrated in a bayesian learning framework. While being incremental, no feedback on the actual reconstruction is given and it is not clear how the algorithm behaves in non-object oriented scenes and in large scale.

Tetrahedralization methods [10, 19] are very close to voxel based methods, but work on a irregular grid. Recent work of Sugiura et al. [17] is capable of incrementally adding cameras and 3D points to an existing mesh. Hoppe et al. [8] directly extract a textured mesh from the sparse point cloud for a good visualization, instead of delivering the most accurate dense reconstruction. The pipeline is incrementally adding points and the surface is extracted by a graph-cut optimization on top of the tetrahedralization. While being very fast and progressive in theory, the system is not suited for highly accurate reconstructions, capturing the fine details of a scene.

1.2. Contribution and Outline

To the best of our knowledge, we are the first who deliver a dense point cloud starting from a sparse structure from motion point cloud on a computational budget in a progressive manner, while explicitly handling the scene scale and delegating the computational power to individual scene parts. The presented pipeline uses an efficient model representation in an octree, allowing the reconstruction of general scenes in large datasets.

An open-source implementation of the proposed pipeline is available at: `https://github.com/alexlocher/hpmvs`

2. Progressive Multi-View Stereo

The following section first introduces the proposed pipeline. Individual steps and terms are later detailed in the corresponding subsections.

2.1. Overview

The proposed MVS algorithm takes a set of calibrated cameras V and sparse 3D points x (the result of any SfM method) as input and produces a dense point cloud consisting of oriented surface patches p. Algorithm 1 gives an overview on the most important steps. The initialization stage converts input points with assigned visibility constraints to surface patches, which are filled into a priority queue and are spatially organized in a dynamic octree. A series of operations on individual patches gradually increase the density, resolution and accuracy of the output point cloud: patches are expanded into their local neighbourhood, their neighbourhood is analysed for filtering and prioritizing and finally a patch is branched into multiple patches of smaller size. Due to the hierarchical procedure, the quality of the produced point cloud is increasing with increasing runtime, which can be observed on-the-fly. The algorithm ends if stopped by the user or if all input images are processed to their finest resolution.

```
Data: SfM point cloud $\{x\}$ and cameras $\{V\}$
Result: dense point cloud at any point in time
▷ Initialize Queue $Q$ with patches $p$ from SfM data:
$Q \leftarrow$ Initialize $(\{x\}, \{V\})$
while Queue $Q$ not empty do
    ▷ Get top priority patch:
    $p \leftarrow Q[0]$
    if $p$ not expanded then
        ▷ Expand patch:
        $Q \leftarrow Q \cup \{\, p, \mathrm{Expand}(p) \,\}$
    else if Nghd $\mathcal{N}(p)$ of $p$ not analysed then
        ▷ Analyse patch:
        $Q \leftarrow Q \cup$ Nghd-Analysis $(p, \mathcal{N}(p))$
    else
        ▷ Branch patch:
        $Q \leftarrow Q \cup$ Branch $(p)$
    end
end
```

Algorithm 1: General pipeline overview.

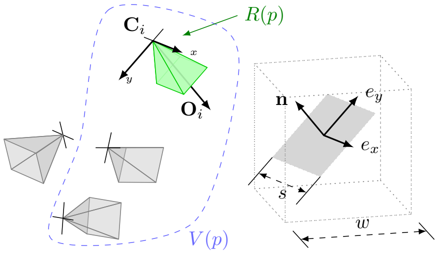


Figure 2: Patch's geometry and coordinate systems.

2.2. Model Representation

The model is represented by a set of individual patches P, where each patch p has an assigned normal $\mathbf{n}(p)$, center $\mathbf{c}(p)$, size $s(p)$ and a set of images, in which the patch is visible $V(p)$ (see also Fig. 2). One of the visible images is selected as the reference image $R(p)$. $\mathbf{I}_i$ denotes the image with the index i and $\mathbf{C}_i$ the position and $\mathbf{O}_i$ the optical axis of the assigned camera. A depth image D_i with the depth values of visible patches closest to the camera is maintained. Every patch has an assigned x-axis $\mathbf{e_x}(p)$ with unit length set to be parallel to the x-axis of its reference image $R(p)$ and $\mathbf{e_y}(p)$ is perpendicular to $\mathbf{e_x}(p)$ and $\mathbf{n}(p)$:

$$\mathbf{e_y}(p) = \mathbf{n}(p) \times -\mathbf{e_x}(p) \tag{1}$$

Patches are organized in a dynamic octree T, consisting of individual nodes $N_i(\mathbf{x}, w)$ and a root node N_r, where $\mathbf{x}$ denotes the node's center coordinate and w its width. We use the term w_{N_i} for the width of node N_i.

2.3. Patch Optimization

In multiple stages of the pipeline, the patch's position $\mathbf{c}(p)$ and normal $\mathbf{n}(p)$ are optimized by maximizing the averaged Normalized Cross Correlation (NCC) of the patch's projection into the image space $g_I(p)$. For the optimization, the patch is parametrized by its depth in the reference image $d_R(p)$ and the two Euler angles of $\mathbf{n}(p)$. Formally, the following function is minimized:

$$e(p) = \frac{1}{|V(p)| - 1} \sum_{I \in V(p) \setminus R(p)} 1 - \langle g_R(p), g_I(p) \rangle \tag{2}$$

A patch's projection $g_I(p)$ is evaluated by bilinear interpolation of points sampled from a plane centered at $\mathbf{c}(p)$ and with normal $\mathbf{n}(p)$. Points are regularly sampled, such that they form a $\mu \times \mu$ grid, where the x-axis is aligned to the x-axis of the reference image and individual grid points are separated by $s(p)$. To respect the patch's scale, the corresponding level l_I, in the image pyramid is used for sampling,

$$l_I = \left\lfloor log_2 \left(\frac{f_{C_i}}{s(p)\, d_{I_i}} \right) \right\rceil \tag{3}$$

where f_{C_i} denotes the focal length of camera C_i and $\lfloor . \rceil$ means integer rounding. Before the optimization process, visible images with a pairwise NCC below a threshold α_1 or an invalid corresponding image level l_I are removed from the set of visible images attached to the patch $V(p)$. After a successful optimization, $V(p)$ is constrained further by increasing the threshold to α_2 $(\alpha_1 < \alpha_2)$ and the reference image $R(p)$ is set to the one with optical axis most similar to the patch's normal $\mathbf{n}(p)$.

2.4. Initialization from SfM

An initial set of patches is created out of the SfM point cloud. For that, the root node of the octree is initialized from a slightly extended bounding box of the initial cloud. Images are loaded, a scale-space pyramid with $l_{max} + 1$ levels is created and a co-visibilty graph is extracted. A set of initial patches is created and its fields are initialized as shown in Algorithm 2. The scale $s(p)$ is initially set to the distance corresponding to one pixel difference in the reference image in the chosen pyramid level $l_{I_{\text{init}}}$[1]. However, during the optimization and processing of the patch, this equality is broken.

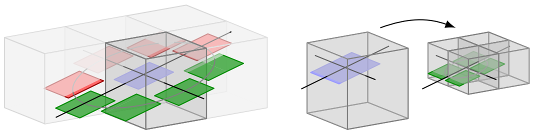
The patch's postion and orientation are optimized and filled into the dynamic octree. The patch's scale $s(p)$ determines the octree-level l_N, in which the patch is stored:

$$l_N = \left\lfloor log_2 \left(\frac{w_{N_r}}{s(p)} \right) \right\rceil \tag{4}$$

[1] If available, the scale of detected keypoints can be used to determine $l_{I_{\text{init}}}$ for individual points. In our experiments we used $l_{I_{\text{init}}} = 4$.

```
Data: SfM point $\mathbf{x}$, assigned cameras $V_s$ with camera
      position $\mathbf{C}$, optical axis $\mathbf{O}$ and focal length $f_{C_I}$
Result: initialized patch $p$
$\mathbf{c}(p) \leftarrow \mathbf{x}$
$V(p) \leftarrow V_s$
$\mathbf{n}(p) \leftarrow \frac{1}{|V(p)|} \sum_{I \in V(p)} (\mathbf{C}_I - \mathbf{c}(p))$
$R(p) \leftarrow argmin_I \langle \mathbf{n}(p), \mathbf{O}_I \rangle \; |I \in V(p)$
$s(p) \leftarrow \frac{f_{C_I}}{d_R(p) \cdot 2^{lI_{\text{init}}}}$
```

Algorithm 2: Patch initialization from SfM point.



(a) The blue patch is extended into neighbouring nodes. Red patches are not added, since nodes are already occupied.

(b) The blue patch is branched into multiple smaller patches and the tree level increased.

Figure 3: Visualization of the extension and branching step.

We limit the number of patches per tree node to one. As this condition can be violated after initialization, we filter all nodes N_i and keep only the most consensual patch in the sense of least squared error $e_l(p)$ among other patches.

$$e_l(p) = \sum_{p_i \in N_i} \left(\frac{\mathbf{n}(p) \cdot (\mathbf{c}(p_i) - \mathbf{c}(p))}{|\mathbf{n}(p)|} \right)^2 \tag{5}$$

For further processing, all patches are filled into the priority queue Q.

2.5. Expansion

The extension stage tries to grow existing patches into neighbouring cells by using the planarity assumption. The octree structure simplifies book keeping and makes sure that extension is only happening into unoccupied regions. Algorithm 3 details the extension procedure. A set of expansion candidate patches P' of p in node N_i are sampled on a circle around p (Fig. 3a) and their fields are initialized. The set of visible images $V(p_n)$ is extended by the co-visible images of the reference image $R(p_n)$, the patch's scale is set to 90% of the node's width and the rest of the fields are copied from p. If the node containing $\mathbf{c}(p_n)$ is empty, the new patch is optimized. After successful optimization, a depth test rejects patches with inconsistent depth information.

$$V_{ok} = \left\{ I \in V(p) \,\middle|\, \sqrt{(d_I(p) - D_I(p))^2} < s(p)\,\delta \right\} \tag{6}$$

$$V_{nok} = \left\{ I \in V(p) \,\middle|\, d_I(p) < D_I(p) - 4\,\delta\, s(p) \right\} \tag{7}$$

V_{ok} denotes the set of images with similar depth values and V_{nok} are images where the new patch would be in front of a visible geometry. We finally add the new patch, if the target node is still empty and $|V_{ok}| > V_{min}$ and $|V_{nok}| < V_{min}$.

```
Data: input patch $p$ and co-visibility graph
Result: set of extended patches $P'$
$P' \leftarrow \{\}$
forall the $n \in \{1, 2, \ldots N\}$ do
    $p_n \leftarrow p$
    $\delta = w_{N_i}\big(\mathbf{e_x}(p) \cos \frac{2\pi n}{N} + \mathbf{e_y}(p) \sin \frac{2\pi n}{N}\big)$
    $\mathbf{c}(p_n) \leftarrow \mathbf{c}(p) + \delta$
    $V(p_n) \leftarrow V(p) \cup \mathrm{CoVis}(R(p_n))$
    $s(p_n) \leftarrow 0.9 \cdot w_{N_i}$
    if optimise$(p_n)$ AND nodeEmpty$(N(p_n))$ AND
    depthTest$(p_n)$ then
        $P' \leftarrow P' \cap p_n$
    end
end
```

Algorithm 3: Extend patch to local neighbourhood.

2.6. Neighbourhood Analysis

For further filtering and for the prioritization, we analyze the local neighbourhood of every patch. Due to the octree structure, this can be realized fast and efficiently. We define the local neighbourhood $\mathcal{N}(p)$ as the patches within the distance $2 \cdot w_N(p)$ and evaluate a robust Huber loss function L_δ of the planar error, similar to Eq. 5.

$$\mathcal{N}(p) = \left\{ p_i \in \mathbf{P} \,\middle|\, |\mathbf{c}(p_i) - \mathbf{c}(p)| < 2 \cdot w_N(p) \right\} \tag{8}$$

$$e_n(p) = \sum_{p_i \in \mathcal{N}(p)} L_\delta \left(\frac{\mathbf{n}(p) \cdot (\mathbf{c}(p_i) - \mathbf{c}(p))}{|\mathbf{n}(p)|} \right)^2 \Bigg|_{\delta = \frac{w_N(p)}{4}} \tag{9}$$

We discard patches where $|\mathcal{N}(p)| < 3$ or $\frac{e_n(p)}{s(p)} > 0.5$ as outliers and remove them from the model.

2.7. Branching

By splitting a single patch into multiple smaller ones, the resolution of the 3D model is gradually increased. Algorithm 4 shows the basic steps. Similar to the expansion procedure, we place the new patches p_n on a circle around $\mathbf{c}(p)$, but with a smaller radius (Fig. 3b). The rest of the fields are copied from the source patch. New patches are optimized and only added to the point cloud, if its center stays within the parent node.

```
Data: input patch $p$
Result: set of smaller patches $P'$
$P' \leftarrow \{\}$
forall the $n \in \{1, 2, \dots N\}$ do
    $p_n \leftarrow p$
    $\delta = \frac{w_{N_i}}{4}(\mathbf{e_x}(p)\cos\frac{2\pi n}{N} + \mathbf{e_y}(p)\sin\frac{2\pi n}{N})$
    $\mathbf{c}(p_n) \leftarrow \mathbf{c}(p) + \delta$
    $s(p_n) \leftarrow 0.45 \cdot w_{N_i}$
    if $optimise(p_n)$ AND $nodeEmpty(N(p_n))$ then
        $P' \leftarrow P' \cap p_n$
    end
end
```

Algorithm 4: Branching of patch p into smaller ones.

3. Prioritization

The priority queue enables to prioritize different patches in different regions. The general idea is to first process patches of higher level nodes and gradually increase the resolution. Patches in salient areas are processed with more priority than patches in planar regions, as they improve the overall accuracy. In addition, a user defined term q_u can focus the reconstruction into regions of major interest. Formally, patches in the queue are sorted by increasing priority q as follows:

$$q_{step} = \begin{cases} 0 & \text{if} \quad \text{extend} \\ 1 & \text{if} \quad \text{neighbourhood analysis} \\ 2 & \text{if} \quad \text{branch} \end{cases} \tag{10}$$

$$q = 10 \cdot \lfloor l_N - \max\{2, e_n(p)\} + q_u \rceil + q_{step} \tag{11}$$

The additive term q_{step} assures that the dependency of the individual steps is respected, while the node's priority is dependent on its size and the planarity error e_n of the patch. Basically we give priority to higher level nodes, unless we detect that their local neighbourhood is already well approximated by a plane.

3.1. Concurrency

For parallel processing of individual nodes, we kept the dependency between different cells and processing steps low. The local filtering as well as the branching step strictly operate on a single node. The expansion step includes a test for empty neighbouring cells, which requires a read access. The insertion of a successfully extended patch is modifying the tree's structure. The local neighbourhood analysis requires read access to neighbouring cells. The proposed design of the priority queue enables the parallel processing of all steps except the expansion without further synchronization. Note that most of the time, the algorithm is busy with patch optimization and hence the synchronization overhead for the expansion stage is minimal.

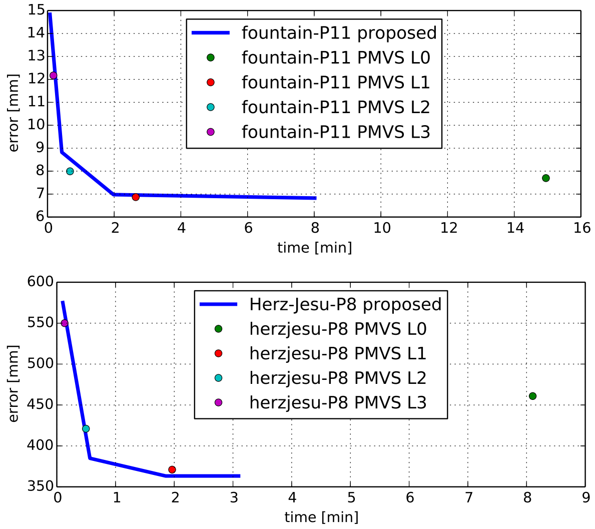


Figure 4: RMS error versus runtime of the proposed method with respect to ground-truth.

4. Experiments and Results

In order to demonstrate the capabilities of the proposed algorithm, we conducted a series of experiments with a C++ implementation of the pipeline. The following parameters were used: $V_{min} = 3$, $\mu = 4$, $\alpha_1 = 0.4$, $\alpha_2 = 0.7$, $l_{max} = 7$ and $l_{init} = 4$ if not stated otherwise. Timings are based on the C++ implementation and the measurements were performed on a single machine with a Intel Core i7 with 8 × 3.7 GHz and 16GB of RAM. The experiments on the citywall dataset were performed on an Intel Xeon with 16 × 2.4 GHz and 48 GB of RAM.

4.1. Progressive Modelling

To compare the performance with state of the art, we tested our algorithm on two different datasets with available ground truth. The fountain-P11 and Herz-Jesu-P8 datasets consists of 11 and 8 calibrated and undistorted cameras [16]. A high resolution laser scan within the same coordinate system serves as ground truth. The error of the produced point cloud is measured as a signed Euclidean distance between the point itself and the closest point on the mesh's surface. As it is complicated to measure the completeness of a point cloud, we use a method similar to the

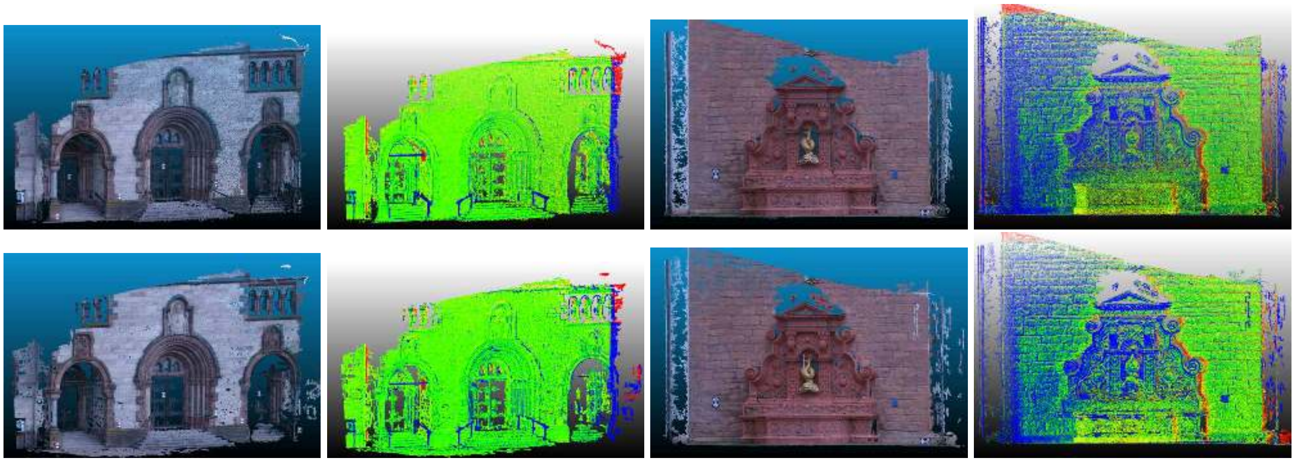


Figure 5: Qualitative evaluation of the 3D reconstruction and error distribution of the fountain-P11 and Herz-Jesu-P8 dataset between the proposed method (top) and PMVS (bottom). Blue points are behind and red ones in front of the ground truth surface.

one used in Middlebury dataset [15]. The ground truth surface is sampled regularly and every vertex is considered to be covered if the there is a point within a certain range d, leading to the completeness C. If not stated otherwise, we let the algorithm run until the maximum input image resolution ($l_I = 0$) is reached.

Fig. 4 shows the evolution of the RMS error with respect to the runtime. It is visible that the average point error decreases with increasing runtime until a final error of 7mm on the fountain-P11 or 360mm on the Herz-Jesu-P8 dataset is reached. In order to compare the performance with the baseline, we also calculated the final average RMS error of PMVS on different image levels. The plot shows that our method keeps the accuracy of PMVS and even outperforms it, while being able to progressively deliver more and more accurate results. Please note that the hierarchical approach gets more and more efficient on higher image resolution as information is propagated among image levels. The graph also demonstrates the effectiveness of the pipeline compared to the trivial approach of running PMVS on increasing image resolutions. The sum of the runtimes on individual PMVS levels is significantly larger than the runtime of the proposed algorithm.

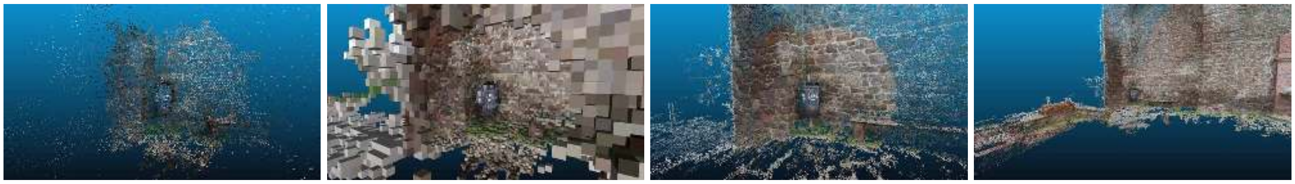
Fig. 5 shows the qualitative evaluation between the final result of PMVS and the proposed algorithm. The error distributions between the two methods are very similar. The highly saturated areas on the model edges are generated due to the lack of ground truth data at that particular location and are simply discarded by a bounding box during the evaluation.

The different approaches of growing patches between the proposed method and the baseline are visualized in Fig 7. It was created by a modified version of PMVS where patches are streamed to a visualization tool as they are created. In comparison to PMVS, which grows very high resolution patches and achieves a low coverage at the beginning, the proposed method grows patches hierarchically and a reconstruction of the whole model (in low resolution) becomes immediately available.

	fountain-P11	Herz-Jesu-P8
algorithm	C [%]	C [%]
proposed	67.2	81.1
CMVS-PMVS - L0	67.0	79.3
CMVS-PMVS - L1	69.9	83.4
MVE - L2	44.0	57.2

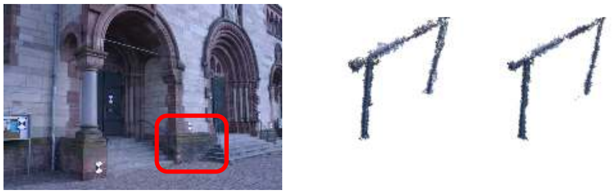
Table 1: Comparison of the point cloud's completeness C.

Table 1 shows the completeness of the final dense point clouds for the proposed algorithm, PMVS on two different levels and MultiView Environment (MVE) on the image level 2. For the evaluation of C, we used a distance threshold d of 0.1% of the bounding box diagonal. The completeness of PMVS and the proposed algorithm are both very similar and both outperform MVE significantly. A frequent problem in hierarchical algorithms is that fine details get missed. Due to the expansion step, details are well covered in the proposed pipeline, as long as they are connected. Fig. 6 shows an example of a freestanding object in the Herz-Jesu-P8 dataset. However, very small disconnected objects, which are already poorly covered in the initial SfM cloud might not get reconstructed. In all our experiments, we never detected such problems - the algorithm had even shown to be less prone to hallucinating flying artefacts, which often survive the PMVS filtering stage.



(a) patches at $l_T = 7$ (b) octree visualization (c) patches at $l_T = 11$ (d) view from distance

Figure 8: Reconstruction of the citywall dataset with a user defined priority. The priority of cells around the trash bin was increased by a user request.



(a) scene (b) PMVS (c) proposed

Figure 6: Eventhough a hierarchical scheme is used, fine details are well reconstructed.

	avg $s(p)$	error	t
with prioritizing	3.5 mm	6.8 mm	486s
without prioritizing	3.0 mm	6.7 mm	675s

Table 2: Effect of the prioritization scheme, shown with the runtime and average patch scale after running the algorithm up to a user defined goal ($q = 110$) on the Herz-Jesu-P8 dataset.

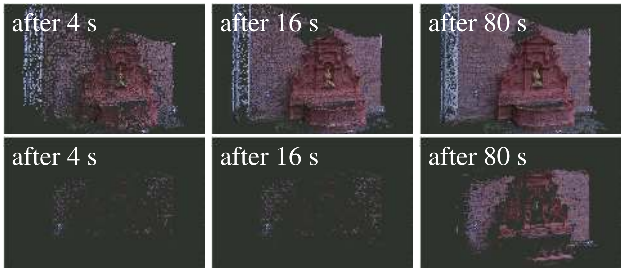


Figure 7: One to one comparison of the patch growing between the proposed method (top) and PMVS (bottom).

4.2. Prioritization

In order to show the effect of the employed prioritizing scheme, we run our algorithm with and without enabled prioritizing, until all nodes with priority lower than a user defined goal q_{end} are processed. We conducted the experiment within the fountain-P11 and Herz-Jesu-P8 datasets and compared it to ground truth. The performance is summarized in Tab. 2. Enabling the prioritizing scheme, allows to reduce the computational time until a fixed goal while maintaining a similar error. The average patch resolution is increased at the same time, as not all nodes are pushed to the final detail level.

While the flatness prior (Eq. 11) is very general, a more user specific prior can guide the reconstruction into scene parts he/she might be interested in. We simulate this behaviour by a simple 3D location based priority term, where cells within a certain radius from an interesting scene part are processed with higher priority. Specifically, we selected the left trash bin of the citywall dataset as a point of interest and reconstructed points within a radius of one meter with higher priority. Fig. 8a shows the reconstruction in an early stage and Fig. 8b shows a visualization of the corresponding octree at the same stage. Every non-empty leaf in the tree is rendered as a cube, coloured with the colour of the patch it contains. Nodes close to the point of interest are of much finer resolution than the rest of the image. Fig. 8c shows a closeup and Fig. 8d a view from distance of the scene.

4.3. Explicit Scale Handling

Due to the octree structure, the effective image resolutions of the input images can be handled globally and the resulting resolution of the point cloud can be increased homogeneously and only processes high resolution close-up images if needed. This helps reducing computational time, but also leads to more realistic reconstructions. The citywall dataset consists of images with a very large difference in effective resolution. Fig. 9 shows the colour coded scale of individual patches in a reconstruction. While the proposed algorithm produces a scene with a very homogeneous effective scale, the reconstruction of PMVS varies greatly in resolution, depending on the resolution of images at that viewpoint. Note that the resolution of patches at far distances (e.g. the roof of the tower) have a lower resolution than the wall itself. This comes from the fact that patches are only reconstructed until their scale reaches the effective image resolution in the corresponding cameras. With that, patches are only reconstructed up to the resolution offered by the images.

	fountain-P11		Herz-Jesu-P8		entry-P10		castle-P30		citywall	
algorithm	# patches	time	# patches	time	# patches	time	# patches	time	# patches	time
proposed	700k	8	500k	3	600k	4	1M	8	1.5M	10
CMVS-PMVS - L0	1.7M	15	1.3M	8	1.4M	10	2.4M	24	20M	214
CMVS-PMVS - L1	500k	3	400k	2	400k	3	700k	6	6M	69
MVE - L2	2.4M	48	1.5M	32	2.0M	33	6.0M	62	78.6M	706

Table 3: Comparison of the runtime among different datasets compared to PMVS on different image resolutions and the reconstruction algorithm offered by MVE. The runtimes are given in minutes.

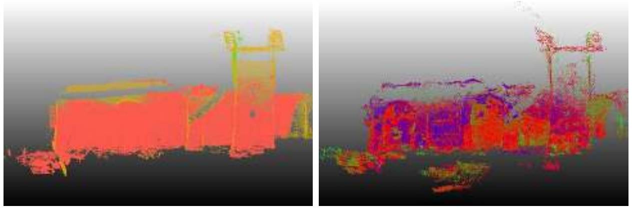


Figure 9: Colour coded per patch resolution of the citywall dataset. While the proposed method (left) can deliver a homogeneous point cloud, PMVS (right) strictly reconstructs on a single image level.

4.4. Timing

While not being the main goal of this work, the algorithm's runtime is an important figure for real-world applications. Therefore, we measure the runtime of our algorithm on the different datasets and compare it to PMVS[6] and the patch reconstruction of Goesele et al. [7] (publicly available in MVE). Tab. 3 summarizes the runtimes, patch's resolution and number of patches among the different datasets and algorithms. Due to the hierarchical approach, our algorithm outperforms the compared methods in terms of runtime in high resolution images. The citywall dataset was split into 11 clusters by the CMVS, which resulted in a total processing time of 3.5 hours. In comparison, the proposed method was able to reconstruct the scene as a whole within ten minutes. While individual parts of the scene are not reconstructed to the very finest image resolution, a well-balanced overall scene resolution of 3mm is reached. The MVE method computes pairwise depth maps, resulting in a huge amount of redundant points but not increasing the effective resolution of individual patches.

4.5. Scalability

The design of the algorithm allows it to be easily parallelized in a shared memory system. Individual nodes of the octree can be processed in parallel and only a small part of the algorithm has to be synchronized. Fig. 10 shows the scalability of the implementation on a range between one and eight processing units. The linear dependency between the speedup and the number of CPU cores shows that the algorithm is perfectly suited to be run on multiple CPU cores. The global mutex for patch insertion would limit the speedup at some point, but can be replaced by multiple local locks in a distributed system.

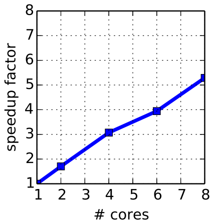


Figure 10: Scalability plot of the C++ implementation of the proposed algorithm in a shared memory system.

5. Conclusion

We have presented a MVS algorithm capable of progressively delivering a dense point cloud. The algorithm explicitly handles the effective scale of input images and focuses on visually pleasing results in early stages. While first reconstruction results are already available after seconds, the accuracy and resolution is gradually improved with increasing runtime. A prioritization scheme focuses the computational power to scene parts with high curvature. The algorithm can reconstruct scene parts of immediate user interest with higher priority. The structure of the algorithm allows for easy parallelization on multiple CPU cores. We evaluated our algorithm on several challenging dataset and compared it to ground truth and to the state of the art. While it maintains or even outperforms the baseline in terms of accuracy, it reduces the processing time by a factor of two on high resolution images. This makes the algorithm perfectly suited for real-time applications, where an immediate user feedback on the dense reconstruction is of great use and allows an early intervention in cases of failure.

Acknowledgment. This work was supported by the H2020 project REPLICATE (No. 687757).

References

[1] L. AgiSoft. Agisoft photoscan. *Professional Edition*, 2014.

[2] A. Bodis-Szomoru, H. Riemenschneider, and L. Van Gool. Superpixel Meshes for Fast Edge-Preserving Surface Reconstruction. *CVPR*, 2015.

[3] N. D. F. Campbell, G. Vogiatzis, C. Hernández, and R. Cipolla. Using multiple hypotheses to improve depth-maps for multi-view stereo. *ECCV*, 2008.

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

[5] Y. Furukawa, B. Curless, S. M. Seitz, and R. Szeliski. Towards Internet-scale multi-view stereo. *CVPR*, 2010.

[6] Y. Furukawa and J. Ponce. Accurate, dense, and robust multiview stereopsis. *PAMI*, 32(8):1362–1376, Aug 2010.

[7] M. Goesele, N. Snavely, B. Curless, H. Hoppe, and S. Seitz. Multi-view stereo for community photo collections. *ICCV*, 2007.

[8] C. Hoppe, M. Klopschitz, M. Donoser, and H. Bischof. Incremental Surface Extraction from Sparse Structure-from-Motion Point Clouds. *BMVC*, 2013.

[9] A. Hornung and L. Kobbelt. Hierarchical volumetric multi-view stereo reconstruction of manifold surfaces based on dual graph embedding. *CVPR*, 2006.

[10] M. Jancosek and T. Pajdla. Hallucination-free multi-view stereo. *ECCV Workshop*, 2012.

[11] M. Jancosek, A. Shekhovtsov, and T. Pajdla. Scalable multi-view stereo. *ICCV Workshop*, 2009.

[12] T. Moons, L. Van Gool, and M. Vergauwen. ARC 3D Webservice. *3D, Science and Cultural Heritage*, 2009.

[13] M. Pizzoli, C. Forster, and D. Scaramuzza. REMODE : Probabilistic , Monocular Dense Reconstruction in Real Time. *ICRA*, 2014.

[14] J.-P. Pons and R. Keriven. Acute3D. *https://www.acute3d.com*, 2011.

[15] S. M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. *CVPR*, 2006.

[16] 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. *CVPR*, 2008.

[17] T. Sugiura, A. Torii, and M. Okutomi. 3d surface extraction using incremental tetrahedra carving. *ICCV Workshop*, 2013.

[18] P. Tanskanen, K. Kolev, L. Meier, F. Camposeco, O. Saurer, and M. Pollefeys. Live Metric 3D Reconstruction on Mobile Phones. *ICCV*, 2013.

[19] H.-H. Vu, R. Keriven, P. Labatut, and J.-P. Pons. Towards high-resolution large-scale multi-view stereo. *CVPR*, 2009.

[20] A. Wendel, M. Maurer, G. Graber, T. Pock, and H. Bischof. Dense reconstruction on-the-fly. *CVPR*, 2012.

[21] C. Wu. VisualSFM: A Visual Structure from Motion System, 2011.

[22] Z.-H. Yuan and T. Lu. Incremental 3d reconstruction using bayesian learning. *Applied intelligence*, 39(4):761–771, 2013.