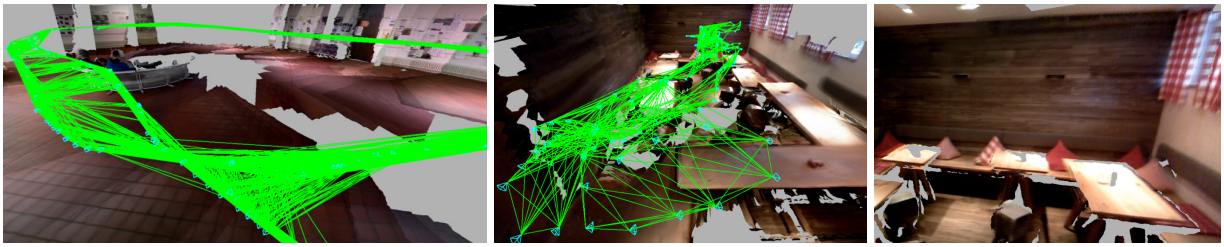
FragmentFusion: A Light-weight SLAM Pipeline for Dense Reconstruction

Darius Rückert*
University of Erlangen-Nuremberg

Matthias Innmann
University of Erlangen-Nuremberg

Marc Stamminger
University of Erlangen-Nuremberg



ABSTRACT

We present *FragmentFusion*, a real-time dense reconstruction pipeline that combines sparse camera tracking with image-space volumetric fusion. The tracking is based on ORB-SLAM, which constructs and optimizes a sparse global map of 3D points and keyframes. We transform each of these keyframes into decimated meshes and render them from the estimated viewpoint. Fusion is performed in the pixel shader by exploiting atomic operations on a packed data structure. This eliminates the need of a 3D voxel grid making *FragmentFusion* very flexible at large scenes and varying scales. Moreover, since all keyframes are fused on-the-fly, we can use bundle adjustment and loop-closure without expensive volumetric re-integration. *FragmentFusion* is lightweight in terms of compute power and memory consumption. It can easily fuse several hundreds of keyframes in real time, in a quality comparable to other approaches. We achieve real-time frame rates on a notebook at around 20% CPU and GPU utilization and low memory consumption.

Index Terms: Computing methodologies—Computer Vision—Reconstruction; Computing methodologies—Computer Vision—Tracking; Computing methodologies—Computer Graphics—Volumetric models; Computing methodologies—Computer Graphics—Image-based rendering;

1 INTRODUCTION

For many augmented reality applications, a 3D model of the world surrounding the user is required. SLAM methods (simultaneous localization and mapping) generate such a model, e.g. from a camera stream, while a user moves through an environment. Latest SLAM methods are very efficient, run on mobile devices, and support large scale scenes [12]. However, the world model generated by these methods is only a sparse point cloud of 3D feature points. A dense model of the environment improves AR user experience, because virtual objects can be rendered behind actual geometry.

A common method to reconstruct such a dense model in real time is to use an active 3D-depth sensor (such as the Kinect). These sensors are cheap, compact, and already available in latest mobile devices. They generate dense point clouds and also work in texture-less regions, but a single depth image of such a sensor is usually noisy, contains holes, and is only valid in a certain depth range. However, it has been shown that by fusing their stream of depth images, it is possible to obtain a dense environmental model of decent quality. The first real-time method of this kind was *KinectFusion* [14], which obtains dense 3D reconstructions of indoor scenes in real time, albeit limited in scale. Various publications presented extensions in terms of scale [15, 20], the handling of dynamic or non-rigid scenes [7, 13], and better global tracking with loop-closure [4].

All these methods incrementally fuse the input depth images into a global 3D voxel data structure, storing a truncated signed distance field. Extracting 3D geometry from such fields is expensive – renderings of the model are thus usually generated by ray casting the signed distance fields, which is often the dominating stage in a dense reconstruction pipeline. Furthermore, memory consumption of the grid-based global data structure is enormous, and quickly reaches the limits even of high-end systems. Tracking methods that subsequently correct previous pose estimates (such as BundleFusion) require a continuous re-integration of depth images. This is very expensive in terms of computation time and requires a high-end GPU to run in real time.

An alternative is to use the keyframes itself as global data structure and synthesize novel views by using image-based rendering techniques. The advantage in comparison to a grid-based TSDF is that the memory consumption is magnitudes lower and previous camera poses can be updated without additional cost. This was already tried in the past [10], but achieved no satisfying results due to low rendering performance and weak camera tracking without local optimization and loop closure.

In this paper, we present *FragmentFusion*, a keyframe-based SLAM pipeline for dense reconstruction. Tracking is achieved using a state-of-the-art sparse SLAM system. The RGB-D keyframes are stored as decimated triangle meshes. To synthesize novel views, these meshes are rendered and fused in screen space by accounting for visibility constraints. Complete and noise-free depth maps can be created from the current view point and from arbitrary camera positions. This allows AR applications to seamlessly integrate virtual objects, because in addition to a depth map from the tracked camera we can, e.g., efficiently render shadow maps from virtual or tracked light sources. Because the depth images of the keyframes are fused per frame, the poses of all frames can be updated continuously, as it happens during bundle adjustment and loop closure. Furthermore we can fuse images of very different scales at high quality without a global resolution limit. In combination with simple culling techniques, the approach can achieve results on par with previous dense reconstruction methods, but with much lower compute load and memory consumption. In our examples, we achieve real-time dense reconstruction on a notebook with around 20% load on CPU and GPU.

*e-mail: darius.rueckert@fau.de

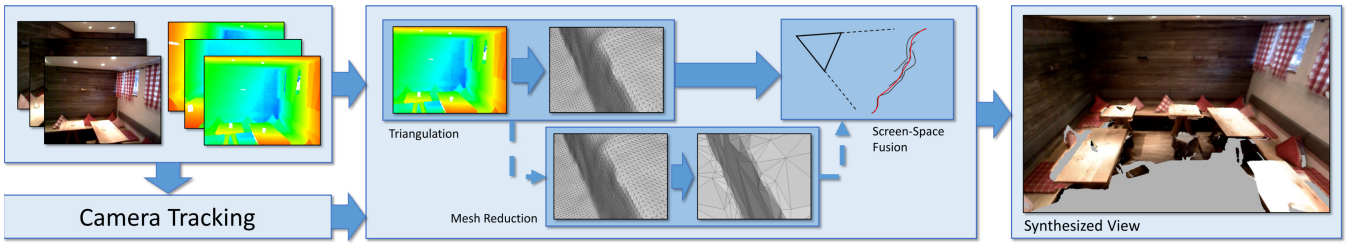


Figure 1: Method overview: Starting from a set of RGB-D images, the camera pose is computed by the tracking system. The keyframe depth images are then triangulated to synthesize novel views by fusing all rasterized meshes in screen space. In a background thread, the meshes are simplified to reduce memory footprint and rendering cost.

2 Related Work

We present a short review of approaches for real-time dense 3D reconstruction, categorized based on their scene representation. For a more complete overview, we refer the reader to a recent survey by Zollhöfer et al. [22].

Volumetric methods The most popular approach for reconstructing surface geometry acquired via a depth camera is to use the volumetric fusion method as presented by Curless et al. [3]. A 3D volumetric data structure is used to store the model in form of a truncated signed distance field. This includes reconstructions of static [14] as well as dynamic scenes [7, 13]. Storing such a 3D field is memory intensive, which bounds the resolution of the reconstruction or the extent of the reconstructed scene. Memory efficient data structures can be applied to extend these bounds [2, 15]. *BundleFusion* [4] corrects for global drift in tracking by using bundle adjustment on color features. The algorithm keeps track of all fused depth frames and de- and re-integrate past frames after pose correction. The resulting memory footprint is considerable: The examples shown in their paper run on two high-end GPUs, totaling in a video memory of 18 GB. Reichl et al. [16] use an octree-based binary voxel grid as model representation, which reduces memory consumption but does not eliminate the costly ray casting step.

Point-based methods Keller et al. [8] reconstruct static scenes using points or Surfels as basic data structure. Surfels generally require less memory than 3D voxel grids and have less limitations on scene extent. They also demonstrate the handling of dynamic objects, which requires fast updates and removal of Surfels. However, integration and rendering of Surfels is relatively expensive and also requires high-end GPUs. Whelan et al. [21] extend this approach to also allow for loop-closure by applying a global deformation on the point cloud.

Mesh-based methods [19] introduced the concept of *mesh zippering* which fuses meshes created from range images. While there has also been some work on real-time reconstruction of 3D geometry using zippered meshes [1], this approach is not well suited for GPU implementation.

Keyframe-based methods Meilland et al. [10] do not construct a global model during reconstruction. Multiple depth maps are fused locally into keyframes by accounting for visibility constraints [11]. The last five fused keyframes are rendered from a predicted viewpoint and the ICP algorithm is applied to match the new to the predicted view. This approach overcomes the memory constraints of volumetric methods. However, drift occurs because of local tracking and no global optimizations such as loop closure.

3 Method Overview

An overview of our method is shown in Fig. 1. The input to our system is a set of RGB-D images. The tracking module (Section 4) computes the current camera pose and optimizes a sparse global map.

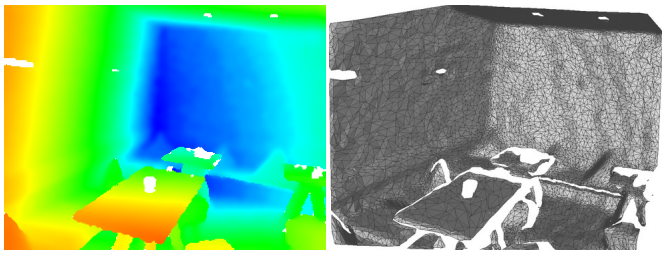


Figure 2: A depth map and its triangulated mesh. No geometry along depth discontinuities is generated.

For each new keyframe, we triangulate the depth image (Section 5). In the background, a process is started to decimate and optimize the mesh. Until this process finishes, we work with its undecimated version. To generate a new view of the model, we fuse the thus generated meshes by rendering them on top of each other (Fig. 1 center) and applying a pixel shader (Section 6) that performs fusion in screen space similar to Merrell et al. [11].

4 Pose Estimation

For camera tracking and keyframe extraction, we use an optimized version of ORB-SLAM. We will give a short overview here. For a more detailed description, the reader is referred to the original paper [12].

ORB features are extracted on each incoming RGB-D image. The feature points are augmented by the depth and matched to the 3D world points of a global map. Outliers are discarded by checking the depth consistency and comparing the associated feature descriptors. If the tracking is weak (only few matched points), a new *keyframe* is created and added to the map. The new keyframe is then matched to a set of previous keyframes to add new 3D world points. The global map is optimized in the background by applying local bundle adjustment. A second background thread matches new keyframes to all previous keyframes to find possible loops. If a loop is identified, pose graph optimization is applied to incorporate the new loop constraint into the map.

The applied optimizations to the reference implementation are mainly focused on performance. The ORB Feature detection is now run on a different thread, matching is improved with hardware specific instructions, and bundle adjustment is implemented using recursive matrices [17]. Overall, the camera tracking requires around 10-15% utilization of a 6-core CPU (2018).

5 Triangulation of Depth Data

For every keyframe given by the tracking system, the depth map is preprocessed and converted to a triangle mesh. The process described here is similar to the triangulation method of Hedman et al. [6], who use it in the context of image-based rendering. To reduce noise of active IR depth sensors, the depth maps are filtered with an

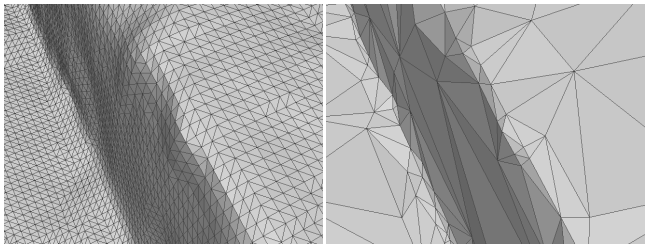


Figure 3: Left: undecimated depth mesh (320×240 resolution, 60k vertices), right: decimated depth mesh (7k vertices).

edge-aware Gaussian blur. Additionally, depth values near geometric edges are invalidated, because they are more likely to be incorrect and our fusion system is sensitive to overhanging edges. Small isolated regions are classified as outliers and are removed. Each vertex of the triangulation is the backprojection of one pixel center in the depth map. For each quad, the triangulation that produces the smaller error is chosen [6]. Faces are only constructed, if the depth difference is small enough to ensure that no geometry is created along depth discontinuities (Fig. 2).

After the initial triangulation is completed, we apply mesh simplification to reduce the memory footprint and increase rendering performance (see Fig. 3). A half edges is collapsed, if the quadric error [16] is smaller than a threshold t_q. On our datasets of 320×240 depth images, on average 91% of the vertices could be eliminated without impacting visual quality. On one CPU core, the reduction takes around 0.3 seconds, which allows roughly three keyframes per second with a single dedicated thread. To further reduce the reconstruction latency, we run the mesh reduction in the background and use the complete triangle mesh in the next step until this process is finished.

6 Fusion of Triangulated Depth Maps

By borrowing techniques from imaged-based rendering [5], our method is able to synthesize novel views of the scene without creating a global model. The keyframe triangle meshes are rendered from a virtual camera position using a standard rasterization pipeline. Given the view and projection matrices V_k, P_k and V_v, P_v from the keyframe and the virtual camera, each vertex v_k is projected to the image point i_v of the virtual camera:

$$\begin{aligned} i_v &= T v_k \\ T &= P_v V_v V_k^{-1} \end{aligned} \tag{1}$$

Since multiple keyframes are projected into the same virtual image, there is usually more than one valid candidate for each output pixel. Additionally, we assume that the depth values are noisy and the estimated camera poses inhibit small errors. Therefore, all fragments that are projected to the same point i_v and belong to the same surface are blended for generating a smooth output image [10, 11]. Similar to voxel-based fusion approaches [4, 14, 15], we use a truncation distance t to decide, whether the new fragment should be fused into the current surface estimate. While grid-based methods store the truncation distance implicitly by updating only nearby voxels, we have to store it per pixel in the framebuffer. Furthermore, far depth measurements contain larger errors and should be fused into the existing surface more likely. We therefore combine our stored truncation distance t_p with a per-fragment truncation t_f,

$$\begin{aligned} t &= t_f + t_p, \\ t_f &= c_1 + d_k c_2, \end{aligned} \tag{2}$$

where d_k is the initial depth in keyframe-space and c_1 and c_2 are user defined coefficients that should be proportional to the expected

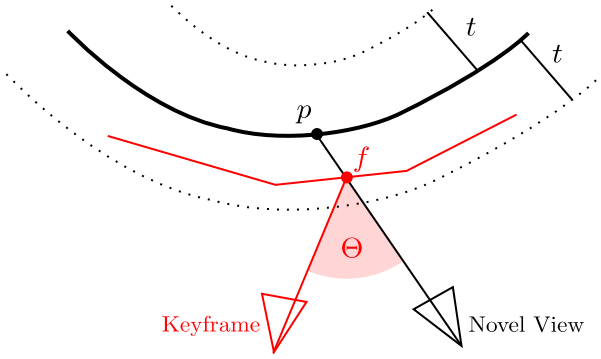


Figure 4: The new keyframe-mesh (red) falls into the truncation distance t of the fused surface (black). The new fragment f is then fused with the point p from the novel framebuffer.

Figure 5: The same scene rendered with z-testing (left) and screen space fusion (right). The visible artifacts at the wall and table are removed.

noise level [15]. Using the truncation t, the fragment depth in novel-view-space d_f, and the framebuffer depth d_p, three cases are distinguished:

1. $d_f > d_b + t$: The fragment is occluded.
2. $d_f < d_b - t$: The fragment occludes the current surface.
3. $d_f \in [d_b - t, d_b + t]$: The fragment is on the current surface.

In case (1), the new fragment is not visible and gets discarded. In case (2), a new layer is created by replacing the framebuffer elements with the fragment data. In case (3), the fragment is fused into the current surface by updating the framebuffer. As shown in Fig. 4, misaligned surfaces can introduce small errors, because the fragment f is fused with p, even though they might correspond to different world points. This is a general problem of image-based rendering and mitigated by weighting the fragment with the angle between keyframe and observer [5].

$$w_f = \frac{\cos\Theta}{d_k} \tag{3}$$

Given the weight, depth and truncation distance of the current fragment as well as the data from the fusion buffer, the smoothed values are computed with an incremental interpolation scheme as proposed by Curless and Levoy [3]:

$$\begin{aligned} \alpha &= \frac{w_p}{w_f + w_p} \\ d_p &= \alpha d_p + (1-\alpha) d_f \\ c_p &= \alpha c_p + (1-\alpha) c_f \\ w_p &= w_p + w_f \end{aligned} \tag{4}$$

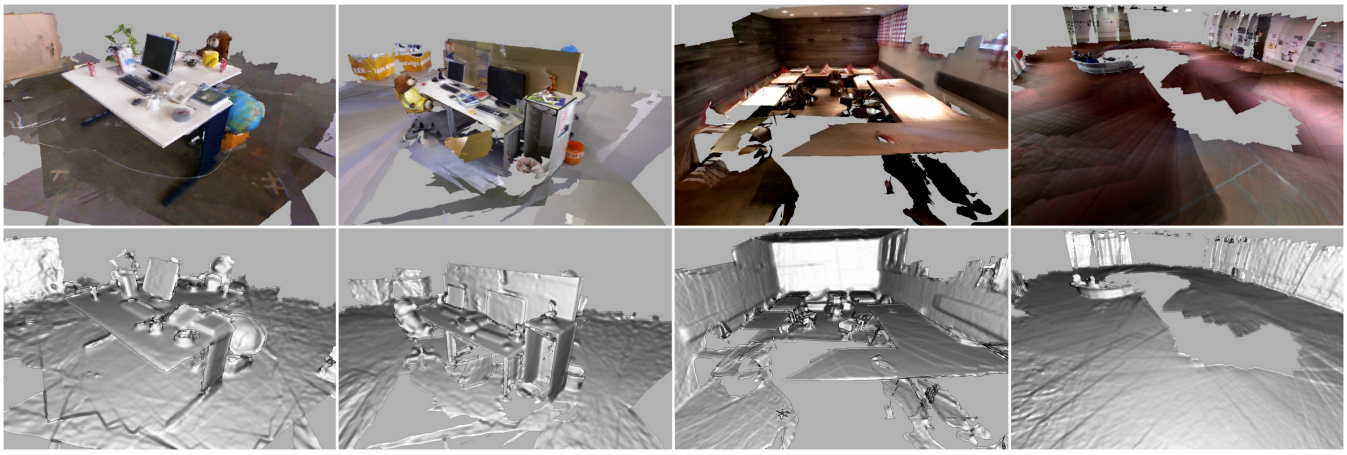


Figure 6: Colored and normal shaded reconstruction output for the DESK, OFFICE, BAR, and HALL dataset.

Offset	0	14	27	49	63
	Color	**Distance**	**Depth**	**Weight**	
Size	15	13	22	14	

Figure 7: Memory layout of the fusion buffer. For each pixel, the color, truncation distance, depth, and weight are packed into 64 bits.

To update the truncation distance, we introduce an additional parameter $\beta \in [0,1]$ that controls how the truncation distance of the reconstructed surface is reduced over time:

$$t_p = \alpha\beta t_p + (1-\alpha)t_f \tag{5}$$

This assumes that the depth error is Gaussian and the reconstruction tends towards the ground truth when fusing erroneous data. Figure 5 shows the output of this method after rendering and fusing 20 triangle meshes.

We have implemented this method using only vertex and fragment shaders of a standard rasterization pipeline. For each pixel of the output window, we store four values in a packed 64-bit framebuffer. This buffer, referred to as *fusion buffer*, contains the reconstructed color and depth as well as a weight and the current truncation distance. The memory layout is shown in Fig. 7. To avoid race conditions when updating the fusion buffer, we utilize 64-bit atomic operations to achieve non-blocking synchronization of the buffer access. Listing 1 shows the algorithm, which we have implemented in GLSL. The weight and truncation distance of the new fragment are computed in lines 1 to 4. The buffer is read with a 64-bit atomic transaction and unpacked according to the memory layout seen in Fig. 7. Depending on the fragment depth d_f, the truncation distance t and the buffer depth d_b, a decision is made: The fragment is discarded (line 11), a new surface is created (line 14) or the buffer is updated by interpolation (line 16). Next, the updated values are packed and a 64-bit compare-and-swap operation is executed on the corresponding element of the fusion buffer. If this compare-and-swap fails (line 20), we encountered a race condition and the computation is started again.

In contrast to traditional volumetric fusion approaches, which fuse new depth data in the direction of acquisition, we fuse all data in observer direction. If the angle between observer and depth camera is large, this can cause visible artifacts along object silhouettes, because measurements of the same world point fall into different pixels. In practice, such artifacts are only visible in regions with high sensor noise and large depth discontinuities. The angle dependent

ALGORITHM 1: FragmentFusion

```
1  c_f = fragmentColor
2  t_f = fragmentTruncation
3  d_f = fragmentDepth
4  w_f = fragmentWeight
5  data = readFromBufferAtomic64(x, y);
6  c_b, t_b, d_b, w_b = unpack(data);
7  do
8      oldData = data;
9      t = t_f + t_b;
10     if d_f > d_b + t then
11         discard;
12     end
13     if d_f < d_b - t then
14         c_n, t_n, d_n, w_n = c_f, t_f, d_f, w_f
15     else
16         c_n, t_n, d_n, w_n = interpolate(c_f, t_f, d_f, w_f, c_b, t_b, d_b, w_b);
17     end
18     newData = pack(c_n, t_n, d_n, w_n);
19     data = atomicCAS64(x, y, oldData, newData);
20 while data ≠ oldData;
```

weight (Equ. (3)) ensures that the defective fragments are fused into the surrounding geometry with little to no influence.

For some applications, color and depth images from novel viewing angles are not enough. For example, an augmented reality physics simulation requires a global model of the environment either stored as a signed distance field or as a triangle mesh. To this end, we allow the user to create a *global snapshot* during the reconstruction. A *global snapshot* is a 3D model, which includes all keyframes up to the point of acquisition. This model is generated via a volumetric integration procedure similar to KinectFusion [14] followed by Marching Cubes [9], if a triangle mesh is required. Because all the required data is already located in video memory, these two steps take between 50 and 200ms in total depending on resolution and scene size. To reduce the observed latency, the camera tracking and screen space fusion proceed in a different thread during the snapshot reconstruction.

7 RESULTS

Setup We evaluate the performance and reconstruction quality of our system on multiple RGB-D datasets. This includes three room-sized scenes and one large-scale scene. Two office scenes are taken from the TUM RGB-D benchmark [18]. Our datasets were captured by an Asus Primesense at 640×480 color resolution. The

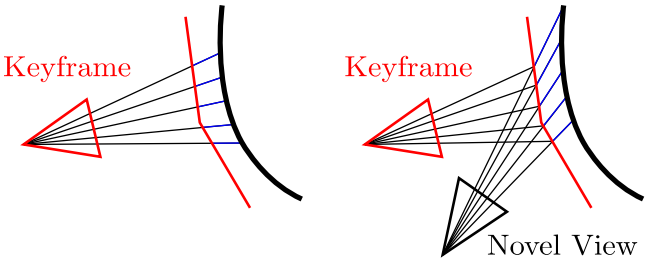


Figure 8: Fusion of a new depth map into an existing surface. Left: Traditional grid-based fusion in z-direction of the keyframe. Right: Our screen space approach in novel view direction. Using our method, errors in pose estimation result in larger surface errors (blue lines).

depth resolution was set to 320×240. All tests were executed on a high-end notebook with an Intel i7-8850H and a NVIDIA Quadro P3200 with 6 GiB of memory. We provide no additional evaluation of the camera tracking, as our pose estimation is based on ORB-SLAM and the results can be found in the respective paper [12]. We compare our method to BundleFusion (BF), which is a state-of-the art grid-based volumetric fusion approach. BF was run at a voxel size of 0.5 cm and 1 cm. The BF tracking failed at the HALL scene, because the depth images are too unreliable due to the high distance. We were also not able to reconstruct BAR with 0.5 cm voxels, because our test system ran out of memory.

Reconstruction Quality An overview of the dense reconstruction on various scenes is shown Figure 6. The displayed images are all created on-the-fly by our screen space fusion system. In Figure 9, a comparison to the grid-based approach BundleFusion is shown. The geometry and color output of our method is displayed on the left and the BF reconstruction is on the right. For the DESK scene (first row), both the geometry and color are slightly better with our algorithm. This is caused by misaligned frames of BundleFusion after the loop is closed. For the OFFICE dataset (second row), the results are similar, but the geometry of BundleFusion is slightly better at some locations. For example, the silhouette of the left chair is not as clean with our screen space fusion approach. Figure 10 shows a comparison of our method to BundleFusion for the room-sized dataset BAR. FragmentFusion's colored reconstruction (top left) achieves a better visual quality, because less artifacts are visible. For example, the curtain in the top right corner is jagged in BundleFusion's reconstruction and smooth in ours. The geometric reconstruction (bottom row) on the other hand is again slightly better using the grid-based approach of BF. The edges are more clean and, for example, the pillow is blurred less. These geometric errors are a result of pose errors in the keyframes in combination with the image-based rendering shown in Fig. 4 of Section 6. Multiple misaligned fragments are fused creating geometric errors mostly being visible along edges. These pose errors are visually less impactful for grid-based methods, because the fusion can be performed in keyframe-space along the z-direction (see Fig. 8).

Timings The average execution times per frame are given in Table 1. Our method needs between 7 and 8.5 ms per frame, which corresponds to a frame rate of 117-142 fps. When reconstructing a live RGB-D stream of a 30Hz depth sensor, the CPU and GPU usage resides at around 20%. The voxel-based BundleFusion needs 65.71 ms per frame for DESK at a resolution of 1 cm. This corresponds to a frame rate of 15 fps, which is below the average frame rate of common depth sensing devices.

Memory The total video memory consumption at the end of the scans is given in Table 2. The CPU main memory consumption is omitted here, because it is roughly identical to the used video memory and usually not the bottleneck. Our system stores a decimated triangle mesh and the corresponding color image for every keyframe. On average, each keyframe requires 1.77 MiB of video memory. Additionally, the fusion buffer described in Section 6 and overhead for rendering and window management consumes around 500 MiB independently from scene size. For example, on a GPU with 6 GiB memory, a total number of 3200 keyframes can simultaneously be stored on the device. In our experiments, a new keyframe is generated on average after 15 frames. This allows for large scale scene reconstruction with over 45k input frames before any streaming has to be initiated. In comparison to BundleFusion, our method requires 5-10 times less memory (see Table 2). This is mainly caused by the sparse voxel grid, which is necessary for BF to reconstruct and track the current camera. The additional overhead such as storing keyframes and rendering information is roughly the same to our system.

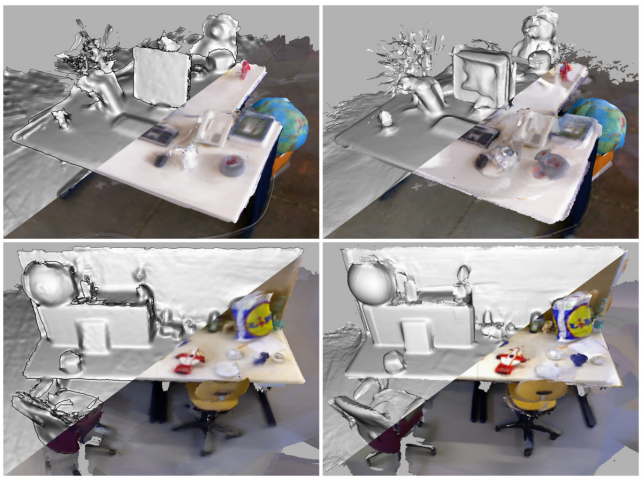


Figure 9: Comparison of our screen space fusion (left) to the grid-based BundleFusion (right). We achieve comparable results in both color and geometry.

Dataset	DESK	OFFICE	BAR	HALL
# Frames	2892	2488	3518	4442
Ours	7.00	7.72	8.51	7.85
BF @ 0.5 cm	104.92	105.10	-	-
BF @ 1.0 cm	65.71	66.28	73.25	-

Table 1: Average time in milliseconds for one frame. BundleFusion (BF) was run with voxel sizes of 0.5 cm and 1 cm. No value means that either tracking has failed or the memory limit was reached.

Dataset	DESK	OFFICE	BAR	HALL
# Frames	2892	2488	3518	4442
Ours	682	766	968	1815
BF @ 0.5 cm	2731	2709	-	-
BF @ 1.0 cm	2151	2083	2627	-

Table 2: GPU memory consumption in MiB at the end of the scan.

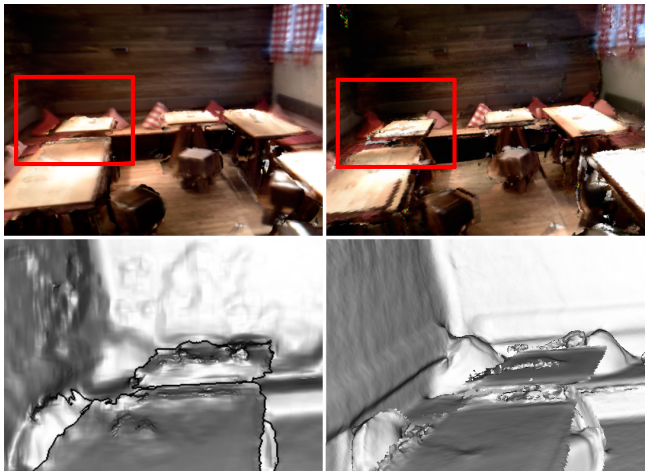


Figure 10: Color and geometry output of FragmentFusion (left) and BundleFusion (right) for the BAR dataset.

8 Conclusion and Limitations

We have presented a hybrid dense reconstruction method of RGB-D images. The camera is tracked using sparse keypoints, which are integrated into a global map. The dense reconstruction is generated on-the-fly by screen space volumetric fusion. This allows us to use large-scale loop detection and bundle adjustment without additional cost. Our system can therefore handle scenes of arbitrary extents and floating scale. The performance of our method is 8-10 times higher than state-of-the-art dense reconstruction systems, while producing similar visual quality and only a slightly worse geometry.

Currently, we see two limitations: First, the integration only happens in view direction. Noisy depth values along silhouettes thus can result in errors. However, such values get very low integration weight and therefore have little impact. Second, our method relies on a mesh decimation procedure, which requires CPU compute power. For the future, we plan to move this step to the GPU, which would also reduce memory transfer. First experiments show that an optimized GPU implementation can speed up this step dramatically.

References

[1] D. S. Alexiadis, D. Zarpalas, and P. Daras. Real-time, full 3-d reconstruction of moving foreground objects from multiple consumer depth cameras. *IEEE Transactions on Multimedia*, 15(2):339–358, 2013.

[2] J. Chen, D. Bautembach, and S. Izadi. Scalable real-time volumetric surface reconstruction. *ACM Transactions on Graphics (ToG)*, 32(4):113, 2013.

[3] 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*, pp. 303–312. ACM, 1996.

[4] A. Dai, M. Nießner, M. Zollhöfer, S. Izadi, and C. Theobalt. Bundlefusion: Real-time globally consistent 3d reconstruction using on-the-fly surface reintegration. *ACM Transactions on Graphics (TOG)*, 36(3):24, 2017.

[5] P. Debevec, Y. Yu, and G. Borshukov. Efficient view-dependent image-based rendering with projective texture-mapping. In *Rendering Techniques' 98*, pp. 105–116. Springer, 1998.

[6] P. Hedman, J. Philip, T. Price, J.-M. Frahm, G. Drettakis, and G. Brostow. Deep blending for free-viewpoint image-based rendering. *ACM Transactions on Graphics (SIGGRAPH Asia Conference Proceedings)*, 37(6), November 2018.

[7] M. Innmann, M. Zollhöfer, M. Nießner, C. Theobalt, and M. Stamminger. Volumedeform: Real-time volumetric non-rigid reconstruction. In *European Conference on Computer Vision*, pp. 362–379. Springer, 2016.

[8] M. Keller, D. Lefloch, M. Lambers, S. Izadi, T. Weyrich, and A. Kolb. Real-time 3d reconstruction in dynamic scenes using point-based fusion. In *3D Vision-3DV 2013, 2013 International Conference on*, pp. 1–8. IEEE, 2013.

[9] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '87, pp. 163–169. ACM, New York, NY, USA, 1987. doi: 10.1145/37401.37422

[10] M. Meilland and A. I. Comport. On unifying key-frame and voxel-based dense visual slam at large scales. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3677–3683, Nov 2013. doi: 10.1109/IROS.2013.6696881

[11] P. Merrell, A. Akbarzadeh, L. Wang, P. Mordohai, J. Frahm, R. Yang, D. Nister, and M. Pollefeys. Real-time visibility-based fusion of depth maps. In *2007 IEEE 11th International Conference on Computer Vision*, pp. 1–8, Oct 2007. doi: 10.1109/ICCV.2007.4408984

[12] R. Mur-Artal and J. D. Tardós. ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017. doi: 10.1109/TRO.2017.2705103

[13] R. A. Newcombe, D. Fox, and S. M. Seitz. Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 343–352, 2015.

[14] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*, pp. 127–136. IEEE, 2011.

[15] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger. Real-time 3d reconstruction at scale using voxel hashing. *ACM Transactions on Graphics (ToG)*, 32(6):169, 2013.

[16] F. Reichl, J. Weiss, and R. Westermann. Memory-efficient interactive online reconstruction from depth image streams. *Computer Graphics Forum*, 35(8):108–119, 2016. doi: 10.1111/cgf.12779

[17] D. Rückert and M. Stamminger. An efficient solution to structured optimization problems using recursive matrices. In *High-Performance Graphics 2019*. Computer Graphics Forum, 2019.

[18] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of rgb-d slam systems. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 573–580. IEEE, 2012.

[19] G. Turk and M. Levoy. Zippered polygon meshes from range images. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pp. 311–318. ACM, 1994.

[20] T. Whelan, M. Kaess, M. Fallon, H. Johannsson, J. Leonard, and J. McDonald. Kintinuous: Spatially extended kinectfusion. 2012.

[21] T. Whelan, S. Leutenegger, R. Salas-Moreno, B. Glocker, and A. Davison. Elasticfusion: Dense slam without a pose graph. Robotics: Science and Systems, 2015.

[22] M. Zollhöfer, P. Stotko, A. Görlitz, C. Theobalt, M. Nießner, R. Klein, and A. Kolb. State of the Art on 3D Reconstruction with RGB-D Cameras. *Computer Graphics Forum*, 2018. doi: 10.1111/cgf.13386