 DOI: 10.3217/978-3-85125-603-1-11

Towards ScalableFusion: Feasibility Analysis of a Mesh Based 3D Reconstruction

Simon Schreiberhuber[1], Johann Prankl[1] and Markus Vincze[1]

***Abstract*— This work describes a novel real time approach for creating, storing and maintaining a 3D reconstruction. Previous approaches for reconstruction attach one uniform color to every geometric primitive. This one-to-one relationship implies that even when geometrical complexity is low, a high resolution colorization can only be achieved by a high geometrical resolution. Our contribution is an approach to overcome this limitation by decoupling the mentioned relationship. In fact newer, higher resolution color information can replace old one at any time without expensively modifying any of the geometrical primitives. We furthermore promise scalability by enabling capture of fine grained detail as well as large scale environments.**

I. Introduction

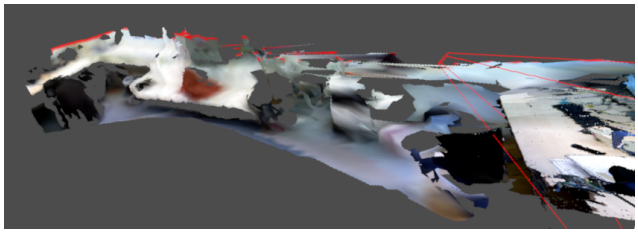
When mapping 3D environments based on the input of an RGBD sensor two steps are usually executed simultaneously.Localization, in which the camera position is tracked relative to the reconstruction or keyframes and the reconstruction itself. This process of Simultaneous Localization And Reconstruction (SLAM) aims to produce a dense representation of reality which finds adaption in augmented reality, robotics and other fields.

One of the first reconstruction algorithms introduced by Izadi et al. was KinectFusion [5], which maintains a volume in form of a 3D grid. In this approach, the grid is populated with values of a Truncated Signed Distance Function (TSDF) indicating where the closest surface resides. The initial implementation is only able to map small volumes of fixed position, size and resolution. By dynamically changing the position of the active reconstruction volume, Kintinuous [10] extends the basis algorithm and enables the reconstruction of bigger scenes. The use of voxel hashing [8] allows higher resolution reconstructions by reducing the memory footprint required for the reconstruction volume. KinectFusion spawned further notable expansions like DynamicFusion [7], which introduces a warp-able volume to reconstruct non rigid objects.

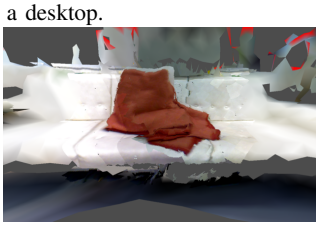
Another thoroughly researched approach is made popular by ElasticFusion [11] where the captured points are stored as surfels, small discs with diameter, orientation, position and color. This allows to store surfaces with varying spatial resolution depending on which distance was perceived by the sensor.

[1]All authors are with the Vision4Robotics group, Automation and Control Institute (ACIN), TU Wien, Austria {schreiberhuber, prankl, vincze}@acin.tuwien.ac.at

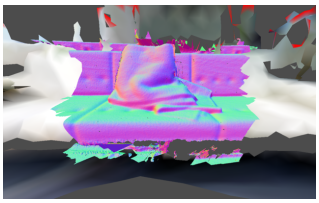
This work is supported by the European Comission through the Horizon 2020 Programme (H2020-ICT-2014-1, Grant agreement no: 645376), FLOBOT.



(a) Ongoing reconstruction of an office. To the left all the geometry is presented in low detail. To the right the sensor (red) is capturing a desktop.



(b) Multiple different frames with varying exposure times contributed texture to the different segments on the couch.



(c) Color coded surface normals are only shown where the high quality version of the surface is loaded.

Fig. 1: The Level of Detail (LOD) system is apparent when looking at a large scene (a). Most of what is displayed consists out of a few triangles with colored corners. More details only appear when zooming in or following the capturing sensor (a). This becomes apparent when looking at the normals (c) which are only shown for fully loaded geometry.

Our approach is inspired by the systems introduced by [11] and [5] but has some essential differences/additions:

- Directly working on a triangle mesh enables us to use textures which are spanned over the used triangles. This strategy inherently propagates the neighborhood information given by the depth map into the reconstruction. This is contrary to the disk shaped surfels used by ElasticFusion [11] which are unconnected and have to overlap to appear like uniform surfaces.
- Storing the texture separate from the geometry allows the meshed surface to be spanned with color information of arbitrary resolution. The density of color information is no longer bound to the geometrical resolution as in ElasticFusion and KinectFusion.
- A Level of Detail (LOD) system which offloads the data residing on the GPU memory to the more plentiful system memory is required when capturing bigger scenes. For user interaction purposes this offloaded data is conserved on the GPU in a lower quality version.

This is not taken care of by the mentioned systems, but absolutely necessary for bigger scenes. Results of this are shown in Fig. 1.

- The segmentation of a captured frame into smaller surfaces is the logical result of the LOD system and the chosen texturing approach. The goal is to split the scene into small manageable chunks which need to be of sufficient size to make texture allocation rational.

II. Scalable Fusion

While the mentioned approaches [5] and [11] work on an intermediate data format, which has to be transformed into a triangle mesh for rendering, our system directly creates and maintains a triangle mesh.

Even more severe, the preceding algorithms attribute only one color to each point of the reconstruction, which then gets interpolated across the triangle surfaces. We, on the other hand, are spanning textures over the surface creating a more detailed reconstruction without increasing the number of triangles.

The consecutive steps performed to incorporate a new camera frame into the reconstruction are presented in the following subsections. These sections are listed in the general order in which they are applied to a frame. To improve performance, this order is later broken up where possible by a threading system described in III.

A. Camera Tracking

Tracking is directly taken from ElasticFusion [11]. But instead of also using the photometric odometry our adaption is limited to the projective ICP approach publicly released by Whelan et al. [11]. Tracking is mostly done relative to the current state of reconstruction. During initialization of the map, an intermediate representation is used based on one keyframe.

B. Geometry Refinement Update

The noise impairing the depth values delivered by RGBD sensors is neither independent nor Gaussian. As a simple example, we imagine a static sensor facing an object at a distance of 4 meters. At distances of about 4 meters, the quantization noise is in the range of centimeters. Usually, the value would appear at one of the closest quantized values, even when observing these values over multiple frames.Following the assumption that this noise behaves Gaussian we would only have to calculate the mean of enough samples to end up with a low standard deviation. From our experiments, we know that we cannot eliminate quantization errors this way, as quantization effects would still be visible this procedure.

In ElasticFusion [11] this is implicitly handled by introducing a "weight" property for surfels. This weight increases the longer a surfel gets observed. During these observations, position, color and normal vectors get updated with the sensor values. With increasing weight of a surfel, these updates become weakened further and further. In the end, the surfels become static, and if the camera does not move, the quantization effects become prominent even with this method. What eventually mitigates this effect is the mechanism which increases the spatial resolution of the reconstruction.

If the sensor approaches a surface in ElasticFusion, surfels become split up into multiple smaller surfels appropriate for the newly gathered data. When doing this, the weight of the new surfels gets reset and a new process of refinement begins cleared of the formerly quantization polluted geometry.

Our approach is inspired by these weights. Instead of spawning new geometry every time the sensor approaches a surface, we only do this when it is beneficial for the reconstructions quality.

For each surface patch, our algorithm stores an additional texture containing values for every sampled surface point p. The values contained for each of these texture pixel (texel) are:

μ_k The average deviation of the k measurements from the actual surface. This is used to indicate where the meshed surface deviates from the sensor's perception.

σ_k An estimate of the noise level. It decreases with every additional measurement. The smaller it is, the less influence new measurements have on the geometry. We also define $\sigma_{s,k}$ as the estimated noise level of the sensor projected onto the surface point p.

$\sigma_{m,k}$ A value which stores estimated minimal noise level that was achievable with the current measurements until step k. The estimate is assuming the quantization effects as the only limiting factor. Similar to before the subscript s refers to the projected value $\sigma_{m,s,k}$ of the sensor.

Each pixel of the texture is updated with the following set of equations: The estimated minimal noise level σ_m is updated by

$$\sigma_{m,k+1} = \min(\sigma_{m,k}, \sigma_{m,s,k}). \tag{1}$$

Updating σ itself is done by

$$\sigma'_{k+1} = \frac{\sigma'_k \sigma'_{s,k}}{\sigma'_k + \sigma'_{s,k}} \tag{2}$$

with

$$\sigma'_{s,k} = \sigma_{s,k} - \sigma_{m,k+1}, \tag{3}$$

$$\sigma'_k = \sigma_k - \sigma_{m,k+1} \tag{4}$$

and therefore

$$\sigma_{k+1} = \sigma'_{k+1} + \sigma_{m,k+1}. \tag{5}$$

It shall be noted that σ'_{k+1} will always be smaller than σ'_k and $\sigma'_{s,k}$ which implies the assumption that every further measurement improves the result. This system also guarantees that σ_k is only approaching $\sigma_{m,k}$ with increasing iteration count k but never falls below it. The resulting values $\sigma'_{s,k}$ and σ'_{k+1} are used to update μ by

$$\mu_{k+1} = \left(\frac{\mu_k}{\sigma'_k} + \frac{d_{s,k} - d_k}{\sigma'_{s,k}} \right) \sigma'_{k+1} \tag{6}$$

with $d_{s,k}$ being the distance of this surface point perceived by the sensor and d_k being the distance of the reconstructed point/texel to the sensor.

Transcribing these texture bound updates to the vertices is done by shifting the vertex positions along the view rays such that μ_{k+1} ends up being 0 wherever possible.

It shall be noted that these updates do not necessarily have to occur every time new sensor values are available for a certain surface pixel. When the estimated noise level of the sensor data σ_k is higher than on the surface $\sigma_{s,k}$, no update needs to be made. The same applies when the perceived depth values are too far off of what has been mapped. This would indicate either unmapped geometry of an already reconstructed surface, or the surface being invalid.

C. Expand Update

As soon as the sensor generates a new frame from a new position, the formerly mapped surface elements are used to render a depth map in the current sensor position. This artificial depth map is then compared to the depth values currently perceived by the sensor. If depth values of the sensor are in proximity to what is mapped, the already existing surfaces will receive an update as described in the previous section. If the captured surface leaves this proximity towards the camera, it will be added (meshed) to the current reconstruction. The thresholds used for these operations as well as $\sigma_{s,k}$ are dependent on depth, pixel position and also on the sensor itself. The sensor characteristics used for the Asus Xtion Pro are derived by Halmetschlaeger-Funek et al. [4] and approximated with a polynomial.

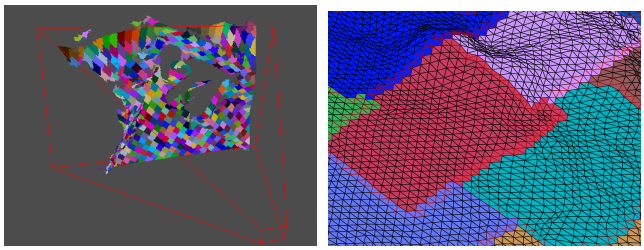
D. Meshing

After identifying the novel parts of the captured depth map, 3D points are created by applying the pinhole model to project the depth pixel. These points are then segmented into smaller blocks depending on their distance to each other and estimated normal vector. The neighborhood information derived from the organized point cloud is directly used in this and also for spanning triangles between each neighboring set of 3 points. When doing so, it again is taken care that no triangles get created where neighboring depth values are within thresholds mentioned in II-C. The results of this segmentation and the meshing process can be seen in Fig. 2.

E. Stitching

Generating a mesh on a single organized depth map is computationally undemanding due to the neighborhood information always being present on the 2D image plane. The situation changes as soon as we seek to integrate new sensor data into an existing reconstruction.

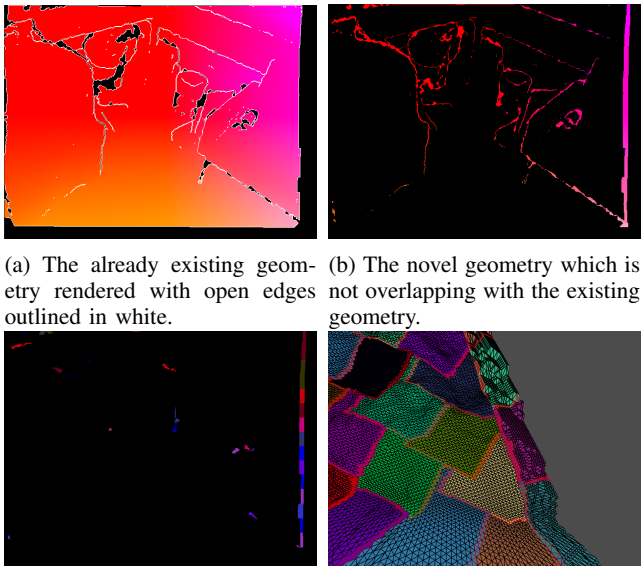
To tackle this problem, we search all the visible triangles captured prior to the current frame for open edges. This refers to every edge where a triangle does not border to another. These edges then get projected in the pixel space of the current frames depth map. When doing so, finding a potential neighbor for a reconstructed triangle within the set of novel triangles is a simple lookup in the current image plane. The whole process of expand the reconstruction is shown in Fig. 3.



(a) The coarse segmentation of the pointcloud into smaller patches

(b) These patches get meshed into a regular pattern of triangles.

Fig. 2: The triangle creation process applied to a single frame.



(a) The already existing geometry rendered with open edges outlined in white.

(b) The novel geometry which is not overlapping with the existing geometry.

(c) Coarse segmentation of the novel geometry. Note how smaller regions do not get mapped.

(d) Finished stitch. The novel geometry appears rough since it was not improved by additional observations.

Fig. 3: The steps required to connect novel data of a sensor frame to existing geometry. The open edges of the geometry (a) outlined in white are connected to the coarse segmentation (c). The result (d) shows a blatant line in the segmentation pattern.

III. Implementation

Modern desktop hardware still distinguishes between memory bound to the GPU and system memory which is bound to the CPU. Access can not be done across memory spaces without doing expensive data transfer over the PCI-E bus. Therefore, our data structures are designed to mirror the information between CPU and GPU and only synchronized when absolutely necessary.

Modifications on the geometry occur on either the GPU or the CPU depending on which processor is more fit for

the performed task. This has implications on the data structure. While data stored CPU space is vastly interconnected, the structures on the GPU only store very few references between elements.

We furthermore use a threading system to simultaneously process tasks which are not fully interdependent. While e.g. the geometry of one frame is used to update the mesh, data which is not needed can be transferred from GPU memory to system memory (download). The odometry is likewise not explicitly reliant on the most current version of geometry, tracking of a new frame can therefore occur concurrently with the integration of the last frame. A example of this system is shown in a timeline in Fig. 4.

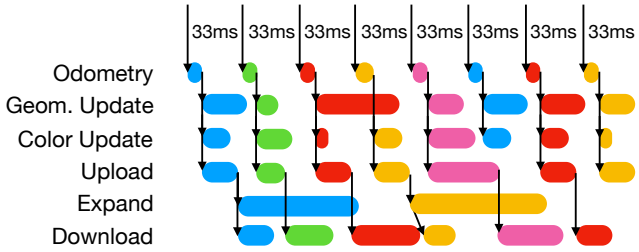


Fig. 4: Each of the rows depicts one thread specialized for its task. The lines show the data flow, beginning with the capture of images at 30 Hz. It is shown how e.g. the geometry refinement update of one frame prevents the geometry refinement update of the following. For the download task, this strategy is not an option. To ensure all the updates made to geometry will be secured, a download step can only be postponed but never dropped.

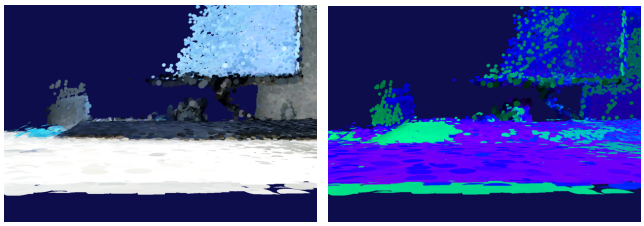
IV. Evaluation

Whelan et al. evaluated the performance of ElasticFusion by comparing the trajectory of the camera odometry to the ground truth captured by Sturm et al. [9]. Further comparisons included the distance of the resulting surfels to the ground truth geometry used to artificially render a dataset. Since our odometry only resembles a part of what is being used by ElasticFusion, we renounce to run these tests at this early state of the pipeline. It should be noted though, that we do not see any technical limitation that opposes the integration of the remaining mechanisms to match ElasticFusions performance.

A. Qualitative Comparison

When looking at surfaces reconstructed by ElasticFusion it is noticeable (Fig. 5) that some of the surfaces are mapped multiple times. This is due to discrepancies in camera tracking and sensor values which we are partially overcoming with a thresholding system that takes standard deviations of the sensor into account.

We are also utilizing a stitching mechanic for connecting geometry that has been created in consecutive frames. Due to imperfection in our stitching algorithm some of these stitches are not complete as shown in Fig. 7. A situation which



(a) Colorized ElasticFusion reconstruction of a desktop. This view is cutting trough the (double) surface and facing a monitor.

(b) ElasticFusion creates multiple layers of the same surface made distinguishable by the green and blue colors (colorized by number of observations).

Fig. 5: ElasticFusion has the tendency of doubling surfaces by creating a secondary layer of surfels.

is worsened by oversegmentation and sequentially clustered surfaces.

In case the sensor is approaching an already mapped surface we replace the old textures of surfaces with the newer higher resolution versions. In its current form this happens without taking care of exposure time and other effects, thus segment borders become visible by abrupt changes in intensity. Fig. 8 shows the increased color density as well as the mentioned discontinuities and offers a comparison to ElasticFusion.

B. Memory Consumption

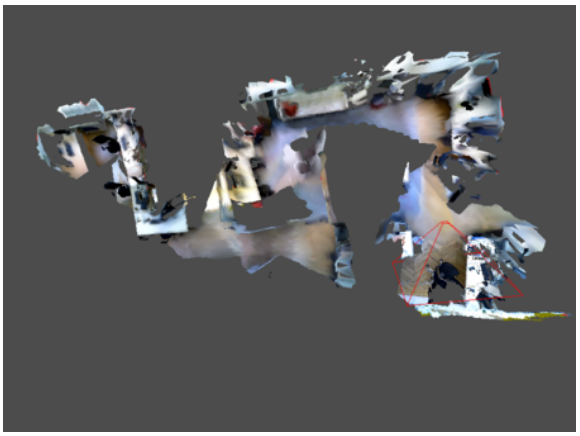
Our current implementation shows its advantage when the sensor keeps exploring new surfaces. In these situations, ElasticFusion will eventually run out of GPU memory while ScalableFusion offloads finished chunks to system memory. This is shown in Figure 9, where the memory consumption of ElasticFusion is steadily increasing while our implementation adjusts its use of GPU memory on the demand. Consecutively this also means that the memory consumption increases when over-viewing big but also detailed structures. In our tests this never posed a problem though.

C. Computational Performance

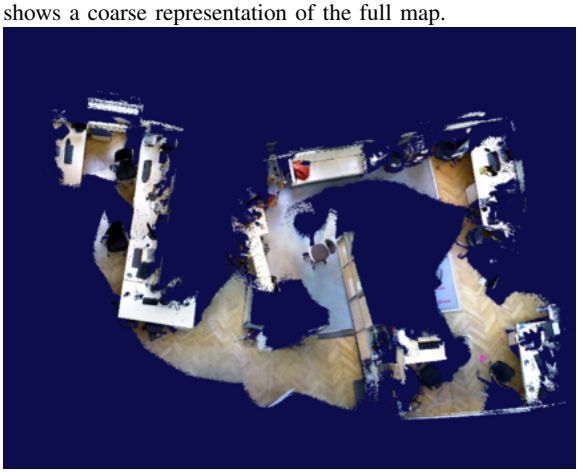
As depicted in Fig. 4 almost all of the tasks are run at the designed 30 Hz. The only task massively deviating from this design goal is the Expand (Section II-C) task. Instead of the targeted 30 ms, it takes 150 to 800 ms to complete. As long as novel geometry is not introduced at a high rate, these durations will not pose a serious limitation.

For our experiments we used a desktop Intel Core i7-7700K CPU in combination with a Nvidia Geforce GTX 1070 with 8 GB VRAM. This combination easily ran the tracking and update steps at the full frame rate (30 Hz) while expanding the geometry at approximately 5 Hz. Most of the CPU cores are utilized to some extent, but mainly waiting for GPU tasks to finish. The GPU was taxed to about 70% of its capacity, which implies some headroom for future features.

Running the same software on a notebook resulted in skipped frames for tracking ($\sim$9 Hz), update steps ($\sim$8 Hz) and hiccups in the user interface. The expand step ran at an



(a) When zoomed out like this, our scalable system only shows a coarse representation of the full map.



(b) ElasticFusion on the other hand always renders all the surfels.

Fig. 6: The reconstruction of scenes like a whole office triggers the LOD system. When the user interface view is zoomed out, our system (a) only renders a low detail version of the reconstruction while ElasticFusion (b) still renders every mapped surfel.

even lower frequency of ~ 1.4 Hz. The resulting reconstruction nevertheless yields similar quality of what the desktop fabricated. The notebook features an Intel Core i7-3740QM CPU with a Nvidia Quadro K2000M and 2 GB VRAM.

V. Conclusions

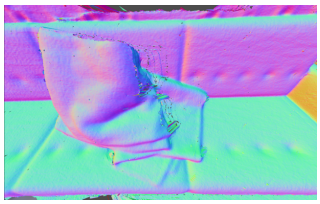
It is shown that directly working on triangles and vertices is feasible in terms of computational effort and even beneficial when maintaining bigger reconstructions.

Conducting all of the meshing in pixel space presents itself as efficient approach for a potentially CPU-intensive problem.

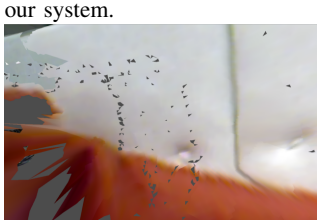
When comparing the resulting normals rendered by ElasticFusion and our approach, it becomes evident that we achieve a similar level of detail (Fig. 7).

Textures appear superior in many instances due to being captured in higher resolution. This representation is less

(a) Textured model generated by our system.



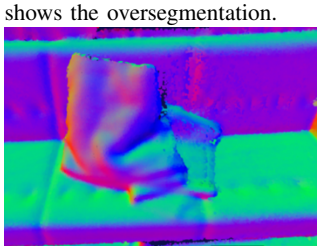
(b) Color coded surface normals.



(c) Stitching artifacts appear between segments.

(d) Color coding the segments shows the oversegmentation.

(e) Same scene captured by ElasticFusion.



(f) Surfels color coded by their normal vector (ElasticFusion).

Fig. 7: The couch scene captured by our system (a-d) and by ElasticFusion (e, f). While the results of our system are comparable on the geometry side, a closer look (c, d) to where the (red) blanket initially shadows the couch from the sensor reveals stitching issues. Geometry needs to be connected between frames which is negatively influenced by noise of geometry data. We hope to fix this issue with a post processing step.

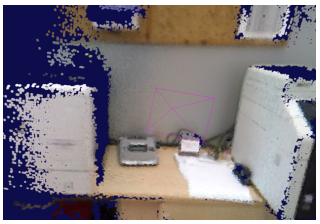
forgiving for rolling shutter sensors, changes in exposure times and tracking errors. As a result, borders between textured patches manifest themselves as sudden changes in intensity as seen in Fig. 8.

VI. Outlook

This paper describes a reconstruction pipeline in an immature state and therefore leaves some problems untreated.

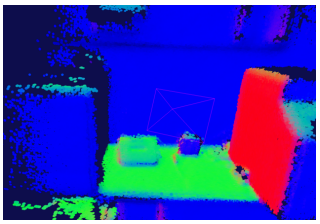
As already indicated in Section II-A the odometry is limited to the use of ICP instead of also exploiting photometric alignment as in [11]. Besides adding these missing parts we are also considering the usage of feature based approaches like ORB SLAM [6] or different featureless ones as Direct Sparse Odometry (DSO) [3].

Texture gets captured in varying lighting conditions, exposure and angles. This leads to reconstructions with very fragmented, non-uniformal texturing. A first step to counter this would be a system to estimate and spare out specular highlights as introduced for ElasticFusion [12]. To further improve quality, the integration of vignetting compensation

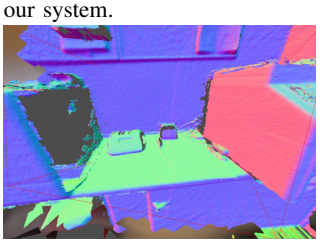


(a) Surface model captured by the surfel based ElasticFusion.

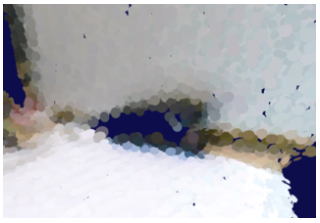
(b) The same input data fed to our system.



(c) Surfels colorized by their normals.

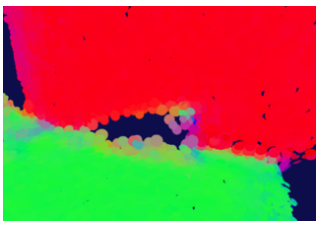


(d) Surfaces colorized by their normals.

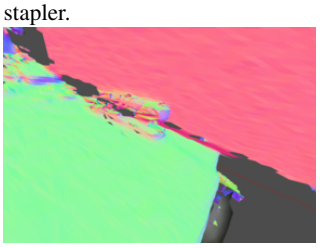


(e) Details like the stapler are barely captured by ElasticFusion.

(f) While our approach has a higher resolution texture for the stapler.



(g) The stapler geometry is barely mapped by ElasticFusion.



(h) Our approach introduces artifacts for the fine geometry of the stapler.

Fig. 8: In a scenario where the sensor is slowly approaching surfaces ElasticFusion (left), as well as our approach (right), improve the geometrical surface quality with a similar principle yielding similar results. When zooming in (e-h) the improvements due to our texturing approach become apparent.

[1] as well as high dynamic range and exposure control presented by Alexandrov et al. [2] is planned.

Other unmentioned tasks are the removal, simplification and tessellation of geometry which are planned for implementation.

It remains to be seen, how much impact these additional features and further optimization will have on the system performance. We are confident though, that further development of this software will improve its utility while keeping the moderate hardware requirements.

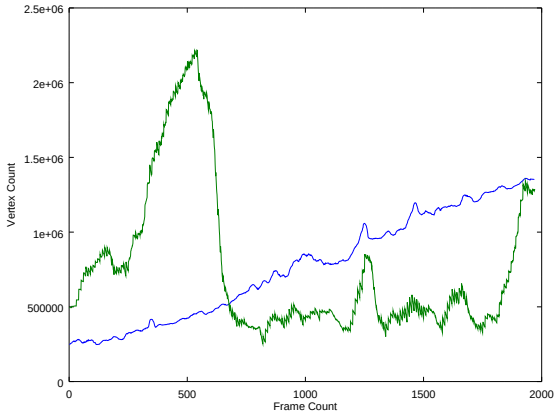


Fig. 9: While the vertex count of ElasticFusion (blue) keeps increasing steadily, the count of ScalableFusion (green) only depends on what is visible momentarily. This also implies that when the sensor overviews a large area full of highly detailed surfaces, the memory consumption spikes (Frame 500).

References

[1] S. V. Alexandrov, J. Prankl, M. Zillich, and M. Vincze, "Calibration and correction of vignetting effects with an application to 3d mapping," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct 2016, pp. 4217–4223.

[2] ——, "Towards dense slam with high dynamic range colors," in *2017 Compute Vision Winter Workshop (CVWW)*, Feb 2017.

[3] J. Engel, V. Koltun, and D. Cremers, "Direct sparse odometry," *CoRR*, vol. abs/1607.02565, 2016.

[4] G. Halmetschlager-Funek, M. Suchi, M. Kampel, and M. Vincze, "Xtion's gone! What's next? An evaluation of ten different depth sensors for robotic systems," *Under Review for IEEE Robotics Automation Magazine*, 2018.

[5] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, and A. Fitzgibbon, "Kinectfusion: Real-time 3d reconstruction and interaction using a moving depth camera," in *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, ser. UIST '11, 2011, pp. 559–568.

[6] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: an open-source SLAM system for monocular, stereo, and RGB-D cameras," *IEEE Trans. Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017. [Online]. Available: https://doi.org/10.1109/TRO.2017.2705103

[7] R. A. Newcombe, D. Fox, and S. M. Seitz, "Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[8] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger, "Real-time 3d reconstruction at scale using voxel hashing," *ACM Trans. Graph.*, vol. 32, no. 6, pp. 169:1–169:11, Nov. 2013.

[9] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of rgb-d slam systems," in *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012.

[10] T. Whelan, M. Kaess, M. Fallon, H. Johannsson, J. Leonard, and J. McDonald, "Kintinuous: Spatially extended kinectfusion," in *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, Sydney, Australia, Jul 2012.

[11] T. Whelan, S. Leutenegger, R. S. Moreno, B. Glocker, and A. Davison, "Elasticfusion: Dense slam without a pose graph," in *Proceedings of Robotics: Science and Systems*, Rome, Italy, July 2015.

[12] T. Whelan, R. F. Salas-Moreno, B. Glocker, A. J. Davison, and S. Leutenegger, "Elasticfusion: Real-time dense SLAM and light source estimation," *I. J. Robotics Res.*, vol. 35, no. 14, pp. 1697–1716, 2016.