Realtime mesh reconstruction from images on edge devices

Department of Computer Science, ETH Zurich, Switzerland

Jannis Widmer

Supervision: **Dr. Dániel Béla Baráth** **Prof. Dr. Marc Pollefeys**

Abstract

This paper presents a rapid and memory-efficient image-to-mesh pipeline designed exclusively for CPU execution, for use in realtime applications. Special focus is placed on large-scale scenes, such as houses or room-scaled scans, and hence less concern is placed on fine detail and aesthetic representation. Potential applications are technical tasks, such as 3D vision for robotics. The effectiveness of the pipeline is demonstrated through the development of an Android app enabling realtime scanning and immediate feedback via point cloud visualization. In a second step, a mesh is generated based on the collected point cloud.

Figure 1. A scan obtained with the app

1. Introduction

The pipeline expects images together with camera poses and camera intrinsics as input. As there are mature frameworks for obtaining camera poses, such as ARCore[1], ArKit[2] or Colmap (Schönberger & Frahm, 2016), designing a custom implementation is omitted. It then generates a dense, oriented point cloud, from which mesh generation can be done. The accompanying code is available at github[3].

The pipeline's capabilities are showcased through the creation of an Android app using ARCore for image acquisition and camera poses. The application generates a dense point cloud representation of the scene which is displayed in realtime, while the mesh generation occurs in an offline step on the phone. It's important to note, that with minor adjustments, realtime mesh generation within the pipeline could be feasible and the decision to not do so is mainly the result of time constraints.

2. Related Work

A lot of prior work on mesh and 3d model generation from images has been done. A small selection of works is for example Colmap (Schönberger et al., 2016), openMVS [4], Nerfs (Mildenhall et al., 2020), or Gaussian Splatting (Kerbl et al., 2023). However, these frameworks and methods fall short when it comes to realtime reconstruction, particularly on edge devices.

There exists also a lot of prior work on realtime 3d reconstruction on mobile devices. Some notable examples are (Kolev et al., 2014), (Muratov et al., 2014), (Prisacariu et al., 2015), and (Schöps et al., 2015). These works differ from the presented one in either only generating point clouds, making extensive use of GPU, and hence being less portable, or focusing on relatively small objects.

An extremely closely related work is presented by (Yang et al., 2020), where an app capable of generating realtime meshes using only CPU is shown for room-scaled scans. While the results presented are indeed impressive, it is assumed that their implementation does not work well with large depths and the results do at least not disprove this claim. The assumption arises from the observation that their complicated depth map generation process, coupled with the reported processing times, appears challenging to achieve for high-resolution images. This, combined with the inherent loss of precision for larger depth ranges in stereo vision applications, could lead to bad accuracy when the depth becomes larger.

[1] https://developers.google.com/ar?hl=en

[2] https://developer.apple.com/augmented-reality/arkit/

[3] https://github.com/jnice-81/CheapMeshStereo

[4] https://github.com/cdcseacave/openMVS

In conclusion, we note that, to our knowledge, this is the first paper to publish code for this task.

3. Pipeline

The pipeline consists of three semi-independent modules:

- The stereo vision module generates an oriented point cloud from image pairs.
- The point cloud module stores points from all views, with a focus on memory efficiency and methods for efficient access.
- The surface extraction module fits a locally defined implicit function to the points, enabling mesh extraction.

3.1. Stereo Vision

A very basic, but in turn fast approach for generating dense points from images is used. A pair of images is rectified, followed by performing either basic Block Matching or SGBM to obtain dense disparities. The implementations by OpenCV are used for this task. In general, SGBM is by far more performance-heavy but has better results in areas with low texture. A normal for each point is computed via a local plane fitting approach and points are projected into 3d Space. For images of size 720x1280 computing a single image pair takes roughly 150 ms using Block Matching and about 500 ms using SGBM on a single core of Google Pixel 6.

Some basic filtering is performed with respect to photometric confidence, such as checking how much better the found match is compared to the other possible matches. Secondly, points that are too far away to be estimated accurately are omitted. Here, the minimum disparity for which a change by one pixel would lead to a difference larger than the maximum allowed imprecision is determined and points with smaller disparities than this are ignored.

Unlike the standard method, depth maps from multiple view pairs are not merged at the image level. While this approach results in more outliers and less precise depth estimations, it offers performance benefits. More specifically, the latency until newly visible points are added to the scene decreases by a large amount, because depth map estimation has to be done only for two instead of n images. This change is later useful for fast user feedback in the app. Essentially the challenge of dealing with noisy, non-uniform sampled point clouds is shifted to later stages of the pipeline.

3.2. Point Cloud

Points are stored in a hierarchical voxel grid, with the semantics that each voxel can contain at most one point. That means that the point cloud is represented as a hash map of voxels, where each voxel stores the mean of all points that were added to it. This leads to a subsampling operation on a point cloud, which allows to trade off memory and runtime against accuracy. Because the number of points falling in a particular voxel is counted, this also enables a certain sense of multi-view consistency. Voxels that were hit multiple times from different image pairs have a higher confidence of being correct, and this is considered during mesh extraction. Having a hierarchy allows efficient access to all points in certain regions of space, an operation that is extensively used during mesh extraction later. A more subtle argument for the hierarchy is that points that are close together in space are, to some extent, stored close together in memory, increasing locality. See Figure 2 for an illustration.

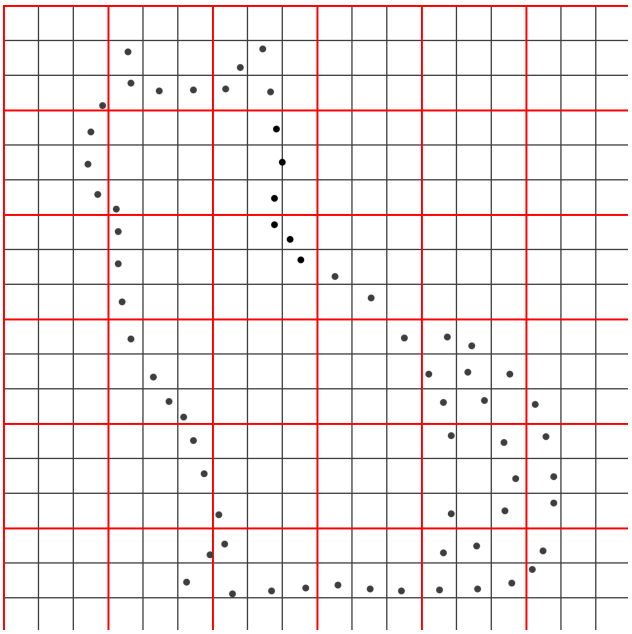


Figure 2. A hierarchical voxel grid with two levels. The red squares (voxels) are stored in a hashmap if they contain at least one point. Each red square is then again represented as a hash map storing the smaller black squares (voxels) inside of it. Each black voxel stores the mean of all points that fall into it.

This design of storing surface is quite closely related to (Nießner et al., 2013). It has some complementary advantages and disadvantages: During updates, only a single voxel has to be changed as compared to (Nießner et al., 2013), where the TSDF has to be updated alongside multiple voxels of the camera ray. Also, it likely is somewhat more memory efficient. While more memory is used per voxel, a voxel contains much more information about the surface, and hence far coarser resolutions can be used. Figure 1 for example was scanned with a voxel resolution of 7cm. A large limitation of the used design is that color cannot be jointly reconstructed well. It would be possible to store

Figure 3. Stanford bunny, 362'271 points

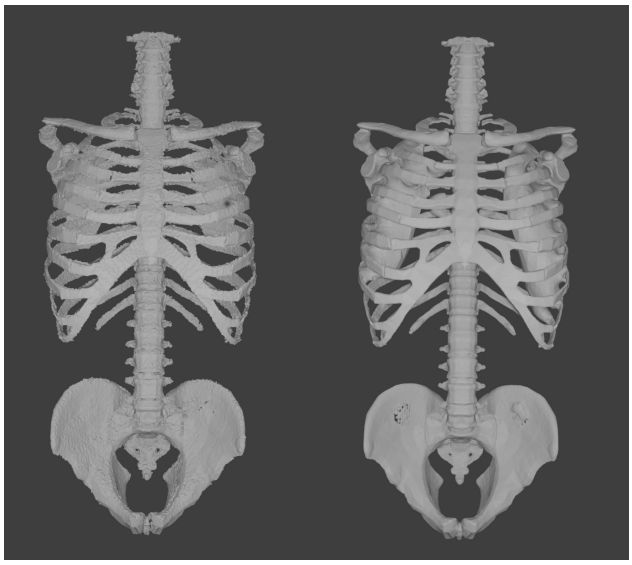


Figure 4. Torso, 3'488'432 points. Left: This method, Right: Poisson Surface. Note the surface reconstructed inside the torso, which is not present in the point cloud.

color information with each voxel, but this would presumably only yield good results when having small voxels, and hence efficiency is lost. Secondly, this approach is probably worse at filtering out noise, because visibility constraints are not used.

3.3. Surface reconstruction

While it was initially planned to use poisson surface reconstruction(Kazhdan & Hoppe, 2013), a decision made based on (Huang et al., 2022) which essentially showed that PSR can still be considered state of the art in many situations when the point cloud contains errors, it was later decided to instead write a custom implementation based on (Fuhrmann & Goesele, 2014). The primary reason was that poisson surface reconstruction tends to "hallucinate" surfaces in particular in the context of outliers and while there are ways to remove those surfaces they usually leave artifacts. Secondly, because this is a local method, it could easily be extended to update the mesh efficiently when the underlying point cloud changes.

The surface reconstruction essentially forms a locally defined signed distance function based on close-by points and extracts the surface using marching cubes. In contrast to the original paper, a simplified version of the implicit function is used. Also, surface extraction is done on a regular grid instead of an octree, and dual contouring (Ju et al., 2002) as opposed to marching cubes is used for surface extraction.

The implicit function is defined as

$$F(x) = \frac{\sum c_i w_i(x) f_i(x)}{W(x)} \quad W(x) = \sum c_i w_i(x)$$

where c_i is the confidence in point i, $w_i(x)$ is the weight introduced by point i at position x, and $f_i(x)$ is the value of the implicit function defined by point i at x. $W(x)$ is the weight introduced by all points at x. The surface is then defined as regions of space where the implicit function is 0 and the weight is large enough. (Fuhrmann & Goesele, 2014)

Simplified, faster to compute versions of the functions as in the original paper are used:

$$f_i(x) = n_i^T (p_i - x) \quad w_i(x) = \begin{cases} 1 - \frac{d(x,p)}{s}, & d(p,x) < s \\ 0, & \text{otherwise} \end{cases}$$

where n_i is the normal of point i, p_i is its position, $d(x, p)$ is the l2 distance of x and p, while s is a scale factor determining how large the region is where a point has influence. Because s is in general much smaller than the entire scene, in practice only the subset of points closer than s to x has to be used to determine the value of the implicit function at x. As confidence, the number of points added to a voxel is used.

For the regular grid of the mesh, the same voxel size is used as in the point cloud representation. The implicit function is evaluated at the corner point of each voxel. Faces are then extracted along the edges and a vertex is positioned with dual contouring. (Ju et al., 2002) (Boris, 2018). While positioning the vertex with respect to the gradient of the implicit function is present, in practice the results are quite comparable to simply placing the vertex at the mean of the intersections, especially at fine-grained resolutions. This is later used in the app due to faster processing.

The algorithm determines which voxels require surface extraction, by first extracting the surface on all voxels that contain a point. If during this process planes to voxels that do not contain a point and have not been discovered earlier

are found, those voxels are registered for later computation. The newly added voxels are then computed in a second round, and the process of computing voxels is repeated until no more voxels are found.

It is somewhat inefficient to compute all voxels completely independently, as multiple voxels usually share corners and hence the algorithm ends up computing the implicit surface multiple times. While a potential solution is to first compute and save the entire implicit function, this is inefficient in terms of memory and hence a different approach is chosen: A cache of already computed implicit function values is maintained, and the algorithm first checks if a particular position has already been computed by checking the cache. To increase cache-locality a hierarchical grid as in subsection 3.2 is used to store and iterate over voxels which should be computed. Hence voxels that are close to each other tend to be processed roughly in order.

4. Results

With the pipeline described, a realtime mesh reconstruction Android app was developed and tested on a Google Pixel 6. ARCore is used to obtain images with poses and intrinsics. A few videos and all the displayed results, including the exact parameters used for surface reconstruction (subsection 4.1) can be found at [5]. Example scans are shown in Figure 1, Figure 6, Figure 7. In Figure 5 a screenshot of the app is displayed.

4.1. Surface reconstruction results

Some results of the surface extraction are shown in Figure 3 and Figure 4, where point clouds available at [6] have been used. The implementation is single-threaded and was run on an Intel i5-8600K CPU.

The reconstruction of the bunny took about 0.6s with a peak memory of 47Mb. As for the torso, the reconstruction with the custom implementation took ∼3.5 seconds with a peak memory of 171 Mb, respectively ∼11 seconds and a peak memory of 176 Mb with poisson surface extraction using a single thread. The time for loading and storing the point cloud respectively mesh has not been considered.

It should be mentioned that the speedup in regards to poisson surface reconstruction mainly stems from the point cloud representation, that is from the downsampling of the point cloud in the hierachical voxel grid. In the torso example inserting all points in a hierarchical voxel grid with 2 levels of hierarchy took about 0.6s where the points are already in memory.

[5] https://drive.google.com/drive/folders/1-nPmOGR-nMf234L9wkdjvAOtcNwJ7jJG

[6] https://github.com/mkazhdan/PoissonRecon

Figure 5. Screenshot of the app

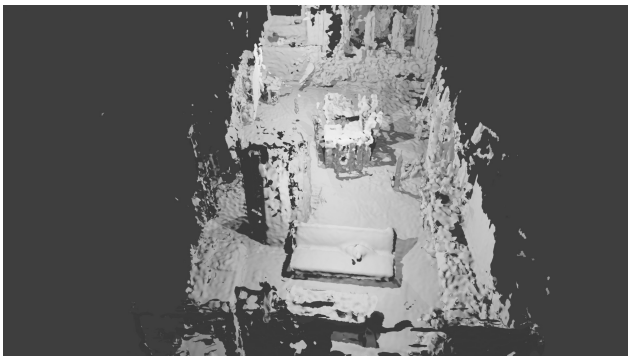


Figure 6. Room

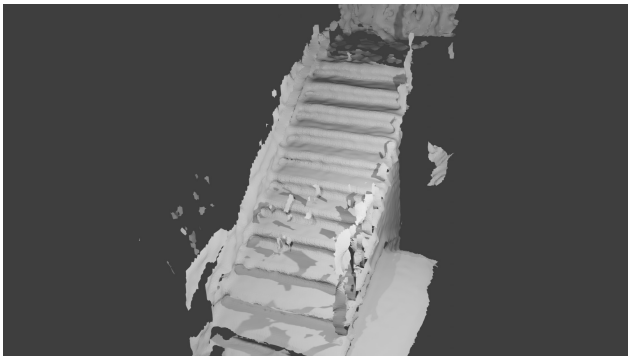


Figure 7. Stairs

4.2. App Implementation details

The app internally manages a sliding window of at most 5 images with a resolution of 1280x720 pixels. A new image is added if it has a large enough rotation or distance from the previously added one. The minimum rotation required was set to 0.2 radians, and the minimum distance to $d/10$, where d is the 10th percentile distance to the camera, of all the currently visible points reported by ARCore. The purpose of adapting the minimum baseline to the distance of the currently observed scene is to have a large overlap of the images, but also a different enough perspective to measure depth effectively for both far and close settings.

If an image is added to the sliding window, all images that have roughly the same rotation are selected for computation with the current image. As minimum precision for the depth estimation, $3 * r$ is used, where r is the resolution of the point cloud voxel grid. Minimum and maximum depth is determined using the sparse point cloud of ARCore. The computation of dense points from images is done using at most two threads running in parallel to the main thread, where SGBM is used instead of Block Matching. Only two threads are used, because when more cores are used the rendering starts to slow down, maybe because of the heterogeneous CPU Cores present in a Pixel 6. Subsequently, all points are added to the point cloud on the main thread.

The collected point cloud as well as the current image is rendered. When the user ends the scan a surface extraction is performed and the reconstructed mesh is saved to disk. Mesh reconstruction uses a scale of $3.5 * r$ with r being the voxel-sidelength and a minimum weight of 4. Choosing a scale much larger than the voxel size has the advantage of reducing noise. The reconstruction speed varies strongly depending on the voxel and scene size but tends to be far below one minute.

5. Conclusion

The designed pipeline supports fast point cloud from image generation in a very portable way and meshes can efficiently be generated in a second step. Also, large scales and depths are supported, as shown in Figure 1. Some limitations remain, like the relatively simplistic depth map estimation and the fact that images are being rectified, leading to somewhat noisy scenes and support for only more or less co-planar image pairs. An easy to improve restriction is that mesh generation happens in an offline processing step.

References

Boris. Dual contouring tutorial, 2018. URL https://www.boristhebrave.com/2018/04/15/dual-contouring-tutorial/.

Fuhrmann, S. and Goesele, M. Floating scale surface reconstruction. *ACM Trans. Graph.*, 33(4), jul 2014. ISSN 0730-0301. doi: 10.1145/2601097.2601163. URL https://doi.org/10.1145/2601097.2601163.

Huang, Z., Wen, Y., Wang, Z., Ren, J., and Jia, K. Surface reconstruction from point clouds: A survey and a benchmark, 2022.

Ju, T., Losasso, F., Schaefer, S., and Warren, J. D. Dual contouring of hermite data. *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, 2002. URL https://api.semanticscholar.org/CorpusID:5060520.

Kazhdan, M. M. and Hoppe, H. Screened poisson surface reconstruction. *ACM Trans. Graph.*, 32:29:1–29:13, 2013. URL https://api.semanticscholar.org/CorpusID:1371704.

Kerbl, B., Kopanas, G., Leimkühler, T., and Drettakis, G. 3d gaussian splatting for real-time radiance field rendering, 2023.

Kolev, K., Tanskanen, P., Speciale, P., and Pollefeys, M. Turning mobile phones into 3d scanners. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3946–3953, 2014. doi: 10.1109/CVPR.2014.504.

Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., and Ng, R. Nerf: Representing scenes as neural radiance fields for view synthesis, 2020.

Muratov, O., Slynko, Y., Chernov, V., Lyubimtseva, M., Shamsuarov, A., and Bucha, V. 3dcapture: 3d reconstruction for a smartphone. 2014. URL https://openaccess.thecvf.com/content_cvpr_2016_workshops/w14/papers/Muratov_3DCapture_3D_Reconstruction_CVPR_2016_paper.pdf. {o.muratov, y.slynko, v.chernov, l.maria, v.bucha}@samsung.com.

Nießner, M., Zollhöfer, M., Izadi, S., and Stamminger, M. Real-time 3d reconstruction at scale using voxel hashing. *ACM Trans. Graph.*, 32(6), nov 2013. ISSN 0730-0301. doi: 10.1145/2508363.2508374. URL https://doi.org/10.1145/2508363.2508374.

Prisacariu, V. A., Kähler, O., Murray, D. W., and Reid, I. D. Real-time 3d tracking and reconstruction on mobile phones. *IEEE Transactions on Visualization and Computer Graphics*, 21(5):557–570, 2015. doi: 10.1109/TVCG.2014.2355207.

Schönberger, J. L. and Frahm, J.-M. Structure-from-motion revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

Schönberger, J. L., Zheng, E., Pollefeys, M., and Frahm, J.-M. Pixelwise view selection for unstructured multi-view stereo. In *European Conference on Computer Vision (ECCV)*, 2016.

Schöps, T., Sattler, T., Häne, C., and Pollefeys, M. 3d modeling on the go: Interactive 3d reconstruction of large-scale scenes on mobile devices. In *2015 International Conference on 3D Vision*, pp. 291–299, 2015. doi: 10.1109/3DV.2015.40.

Yang, X., Zhou, L., Jiang, H., Tang, Z., Wang, Y., Bao, H., and Zhang, G. Mobile3drecon: Real-time monocular 3d reconstruction on a mobile phone. *IEEE Transactions on Visualization and Computer Graphics*, 26(12):3446–3456, 2020. doi: 10.1109/TVCG.2020.3023634.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Eigenständigkeitserklärung

Die unterzeichnete Eigenständigkeitserklärung ist Bestandteil jeder während des Studiums verfassten schriftlichen Arbeit. Eine der folgenden drei Optionen ist in Absprache mit der verantwortlichen Betreuungsperson verbindlich auszuwählen:

☐ Ich bestätige, die vorliegende Arbeit selbständig und in eigenen Worten verfasst zu haben, namentlich, dass mir niemand beim Verfassen der Arbeit geholfen hat. Davon ausgenommen sind sprachliche und inhaltliche Korrekturvorschläge durch die Betreuungsperson. Es wurden keine Technologien der generativen künstlichen Intelligenz[1] verwendet.

☐ Ich bestätige, die vorliegende Arbeit selbständig und in eigenen Worten verfasst zu haben, namentlich, dass mir niemand beim Verfassen der Arbeit geholfen hat. Davon ausgenommen sind sprachliche und inhaltliche Korrekturvorschläge durch die Betreuungsperson. Als Hilfsmittel wurden Technologien der generativen künstlichen Intelligenz[2] verwendet und gekennzeichnet.

☒ Ich bestätige, die vorliegende Arbeit selbständig und in eigenen Worten verfasst zu haben, namentlich, dass mir niemand beim Verfassen der Arbeit geholfen hat. Davon ausgenommen sind sprachliche und inhaltliche Korrekturvorschläge durch die Betreuungsperson. Als Hilfsmittel wurden Technologien der generativen künstlichen Intelligenz[3] verwendet. Der Einsatz wurde, in Absprache mit der Betreuungsperson, nicht gekennzeichnet.

Titel der Arbeit:

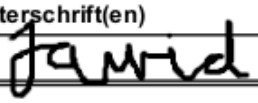
Realtime mesh reconstruction from images on edge devices

Verfasst von:
Bei Gruppenarbeiten sind die Namen aller Verfasserinnen und Verfasser erforderlich.

Name(n):	**Vorname(n):**
Widmer	Jannis

Ich bestätige mit meiner Unterschrift:
- Ich habe mich an die Regeln des «Zitierleitfadens» gehalten.
- Ich habe alle Methoden, Daten und Arbeitsabläufe wahrheitsgetreu und vollständig dokumentiert.
- Ich habe alle Personen erwähnt, welche die Arbeit wesentlich unterstützt haben.

Ich nehme zur Kenntnis, dass die Arbeit mit elektronischen Hilfsmitteln auf Eigenständigkeit überprüft werden kann.

Ort, Datum	**Unterschrift(en)**
22.03.2024	

Bei Gruppenarbeiten sind die Namen aller Verfasserinnen und Verfasser erforderlich. Durch die Unterschriften bürgen sie grundsätzlich gemeinsam für den gesamten Inhalt dieser schriftlichen Arbeit.

[1] z. B. ChatGPT, DALL E 2, Google Bard
[2] z. B. ChatGPT, DALL E 2, Google Bard
[3] z. B. ChatGPT, DALL E 2, Google Bard