Monocular, Real-Time Surface Reconstruction using Dynamic Level of Detail

Jacek Zienkiewicz Akis Tsiotsios Andrew Davison
Stefan Leutenegger
Imperial College London, Dyson Robotics Lab, London, UK
{j.zienkiewicz12, c.tsiotsios, a.davison, s.leutenegger}@imperial.ac.uk

Abstract

We present a scalable, real-time capable method for robust surface reconstruction that explicitly handles multiple scales. As a monocular camera browses a scene, our algorithm processes images as they arrive and incrementally builds a detailed surface model. While most of the existing reconstruction approaches rely on volumetric or point-cloud representations of the environment, we perform depth-map and colour fusion directly into a multi-resolution triangular mesh that can be adaptively tessellated using the concept of Dynamic Level of Detail. Our method relies on least-squares optimisation, which enables a probabilistically sound and principled formulation of the fusion algorithm. We demonstrate that our method is capable of obtaining high quality, close-up reconstruction, as well as capturing overall scene geometry, while being memory and computationally efficient.

1. Introduction

A 3D reconstruction system based on a moving monocular camera is effectively a variable-baseline multi-view-stereo system, and unlike a depth camera or stereo rig, does not have a fixed minimum or maximum range. As a camera browses a scene, we can use small baselines when the camera is close to objects to capture fine details, and when the camera is far away, we can observe the global, coarser structure. However, fusing the depth measurements with these dramatically different scales into a single resolution representation of the environment is problematic, and system design decisions are often made which incur performance or quality penalties. For example using a fine resolution throughout will result in high memory consumption and can lead to aliasing artifacts when the density of the measurements is low relative to the resolution of the model.

In this paper we present a new real-time multi-resolution fusion approach that naturally supports and harnesses the superior characteristics of a monocular system, and is robust, flexible and scalable. We maintain an implicit multi-scale representation of the environment based on a Laplacian mesh decomposition that maintains hierarchy of approximations of the surface at various resolutions. Inspired by the Level of Detail (LOD) approach from Computer Graphics, we dynamically determine the required scale as each new piece of data is fused into it.

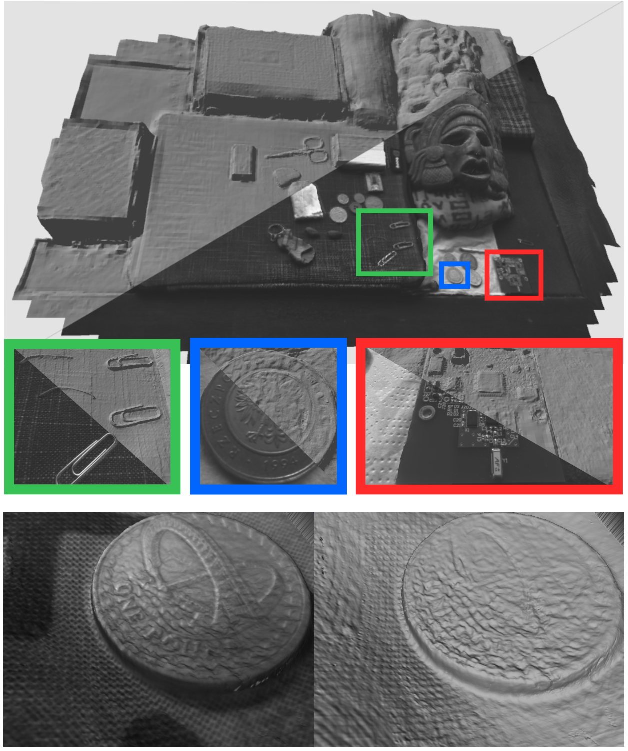


Figure 1: Our method efficiently reconstructs a surface model, and is capable of creating high quality details.

Unlike most real-time, incremental, dense reconstruction methods that either perform volumetric fusion using a TSDF [20] or unordered dense surfel clouds [15], we fuse the depth maps directly into a triangular mesh. Our fusion approach is formulated as recursive Bayesian estimation in a probabilistically sound fashion. Technically, with every new frame, we solve a large-scale optimisation problem. Real-time performance is obtained thanks to the semi-regular, adaptive mesh structure as well as a very efficient, parallel Gauss-Seidel solver. Furthermore, our formulation of fusion as an optimisation problem allows us to improve performance by using a robust cost function (in an iteratively reweighted least squares framework) as well as regularisation. We demonstrate high quality detailed reconstruction at the level of sum-millimetres, as well as examples of practical applications of our approach *e.g.* in robotics.

2. Related work

2.1. Multi-scale reconstruction

Obtaining high-quality surface reconstruction directly from a set of images has been a widely studied problem within the field of computer vision and graphics, and many different solutions for handling multiple scales have been proposed. Among off-line methods that globally optimise a batch of images recent examples include work by Fuhrmann and Goesele [8, 9] and Ummenhofer and Brox [30]. These approaches have shown remarkable results but are highly prohibitive for real-time applications where processing should be fast and the reconstruction should be updated incrementally.

In the field of real-time SLAM approaches that operate in an incremental fashion, the main emphasis is usually put on *scaling-up* the reconstruction, rather than obtaining very accurate and detailed models. This is because, although there exist methods that use passive cameras only [21, 23], most successful real-time systems rely on depth cameras which have quite limited depth range, *e.g.* [20, 15]. When designing large scale, real-time dense reconstruction systems, much effort has been focused on reducing the amount of memory and resources spent on processing "empty" space and therefore these methods are rarely particularly good at dealing with scale changes. Notable examples of scalable, real-time 3D reconstruction systems include Kintinuous [32], multi-scale octree representation for TSDF [4, 26], voxel hashing [22], and multi-resolution surfel maps [27].

2.2. Surface rendering using dynamic LOD

Multi-scale and level of detail object representations play an important role in Computer Graphics for rendering of complex geometric models and there is a plethora of different methods and approaches. In his seminal work, Hoppe [12] introduced the progressive mesh scheme, a continuous-resolution representation of arbitrary triangle meshes. Progressive Meshes allow for a smooth choice of detail level depending on the current view, and were used for high-quality, continuous level of detail rendering in various scenarios [13, 14]. The method produces compelling results, and is designed for arbitrary meshes, but it requires rather complex preprocessing that is not suitable for an incrementally reconstructed mesh.

More relevant and similar to our level of detail approach are methods based on regularly sampled, hierarchical structures such as grid quad-trees, *e.g.* Lindstrom *et al.* [16] or Real-time Optimally Adapting Meshes (ROAM) as proposed by Duchaineau *et al.* [5]. A basic building block of these methods is a patch that represents a small area of the terrain/landscape. Each triangle within a patch can be recursively tessellated by binary subdividing its edges until the desired level of detail is reached. The methods based on partitioning of the mesh into patches are very simple and efficient. Another notable example of a LOD method that was specifically designed for large-scale terrain rendering is geometry clipmaps [17], which caches the terrain in a set of nested regular grids centred about the viewer, in a similar way to how texture clipmapping works.

3. Overview

We follow a rather standard monocular 3D reconstruction pipeline that consists of three distinct stages: camera tracking, depth estimation, and depth map and colour fusion. Given camera poses obtained from camera tracking, for each frame we employ a multi-view stereo algorithm to estimate dense depth maps. Noisy depth maps (together with the colour images) are then fused into a consistent model. The main novelty of our approach is in the fusion algorithm and we will only briefly describe our approaches to camera tracking and depth estimation.

Note that our fusion approach is independent of the camera tracking and the depth estimation techniques, and in fact it could be also used *e.g.* with a depth camera. However, it is monocular systems that can excel with his kind of multi-resolution fusion as they are capable of obtaining depth maps from scales of millimetres to metres.

3.1. Camera tracking and depth estimation

Our fusion method assumes that the camera poses are given. In our implementation we use ORB-SLAM [19] with its standard settings, but other monocular tracking systems are suitable as well, *e.g.* SVO [7] or LSD-SLAM [6]. Robust performance of the ORB-SLAM and drift-free poses thanks to the bundle-adjustment helps us obtaining consistent reconstructions. We also use the estimated depths of the features detected by ORB-SLAM in the current frame to limit the disparity range searched during the stereo matching.

To estimate a depth map for each new image we run a simple multi-view stereo method. We maintain a fixed-size buffer of recent frames that are candidates for matching and a new frame is added to this buffer when the camera has moved sufficiently far from the most recent keyframe. Note that this is entirely independent of the keyframes selected and maintained by ORB-SLAM.

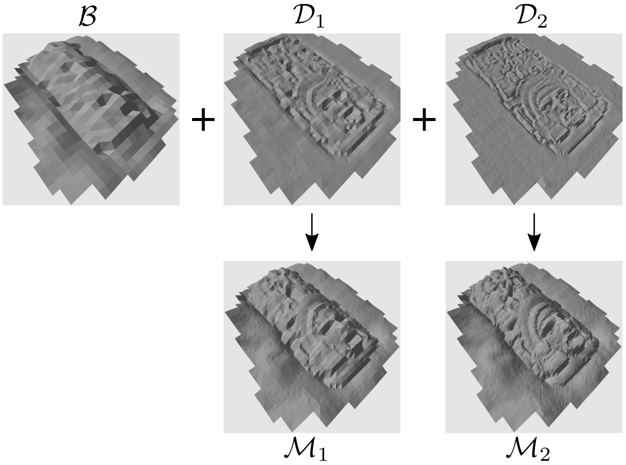


Figure 2: Multi-scale mesh based on Laplace pyramid.

We utilise the concept of plane sweeping and cost volume aggregation as it is a flexible way for performing multi-view stereo matching. Unlike DTAM [21], which used multiple small baseline frames, we are more restrictive in the way we select images for stereo matching: we use few frames (4-7) but with different baselines. This relies on the observation that images with short baseline help to avoid local minima, whereas larger baselines improve accuracy.

The Census transform [33] of a 9×7 image patch centred around a pixel together with Hamming distance is used for calculating matching cost. Scores from multiple frames that are accumulated in the cost volume are subsequently aggregated [24] using the guided image filtering technique (CLMF) proposed by Lu *et al*. [18]. This approach runs in constant time thanks to the use of orthogonal integral images [34], and it avoids the computational complexity of the global optimisation used in DTAM, while still offering good regularisation properties in low-texture areas and preserving sharp edges.

4. Multi-scale surface reconstruction

We will first describe the multi-scale surface representation used in our algorithm and explain how we dynamically generate various levels of detail. Next, we will outline the fusion algorithm under the assumption that there is only a single resolution. Finally, we will combine all elements together and present the whole multi-scale fusion framework.

4.1. Triangular Mesh

Most the surface reconstruction methods either perform volumetric reconstruction using implicit functions, or simply represent a surface using unordered surfels. Alternative methods belonging to computational geometry [1, 2] directly create the mesh using the existing points. In our approach, rather than trying to create a triangulated surface from a point cloud, we start with a predefined, fixed topology, triangular model of the surface and fit it to the data. This can also be thought as surface fitting using deformable models [29, 28]. We start with a flat surface at a predefined distance from the camera, and allow one degree of freedom per vertex, *i.e*. displacement with respect to some predefined surface normal direction. Currently, we cannot change the topology of the mesh, but we can locally refine the mesh by recursively and systematically subdividing triangles and introducing new vertices as more details are needed. Specifically, each individual triangle in base mesh can be divided into up to 4096 smaller triangles. Rather than explicitly storing the mesh at multiple resolutions, we use the implicit representation inspired by the Laplace / Burt-Adelson style pyramid [3, 11].

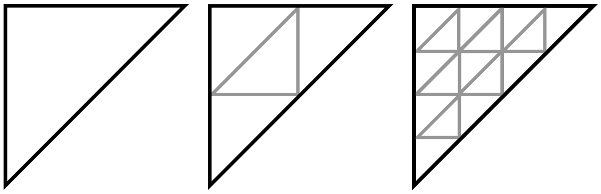


Figure 3: When subdiving a triangle, we use the following, regular tessellation pattern: left is the base triangle, middle and right are two consecutive levels of detail, level 1 and 2, where the triangle is respectively subdivided into 4 and 16 smaller triangles.

We denote by $\mathcal{B}$ the base mesh that captures the coarsest geometry and store it using a fixed grid of size $n_B \times n_B$. $\mathcal{D}$ represents a "detail" mesh that stores only the high-frequency details, not captured by the coarser mesh. As we use an oversampling factor of 2, the size of $\mathcal{D}$ is $(2n_B - 1) \times (2n_B - 1)$, however in practice we only need to store the details in sub-grids, where they are required, which greatly reduces memory usage. In our system, we allow up to 6 detail levels, $\mathcal{D}_i$ for $i = 1 \dots 6$, each with increased resolution compared to the previous level, that store only the differences between the higher resolution and the lower resolution meshes.

As demonstrated in Fig. 2, starting with the coarsest mesh $\mathcal{B} = \mathcal{M}_0$ we can generate a sequence of meshes $\mathcal{M}_i$ at increasingly higher resolution, by adding detail coefficients:

$$\mathcal{M}_i = \mathcal{B} + \sum_i \mathcal{D}_i \ . \tag{1}$$

Fig. 3 shows the tessellation pattern that we use to increase the resolution of the mesh. When going from one level to another, we simply divide each edge of a triangle in half by introducing new vertices and therefore split a triangle into 4 smaller ones. This procedure can be repeated recursively, in total 6 times, and therefore, with the base geometry we can create in total 7 levels of detail (and achieve a 4096 fold resolution increase).

In order to calculate the position of newly introduced vertices, we perform vertex assembly, *i.e.* reconstruct vertex positions using multiple levels of detail. First, we predict a vertex position within the finer mesh by interpolating the coarser mesh. Next, we displace the vertex by adding a "detail" coefficient from the finer resolution. This is done recursively until the required level of detail is reached.

4.2. Dynamic Level of Detail

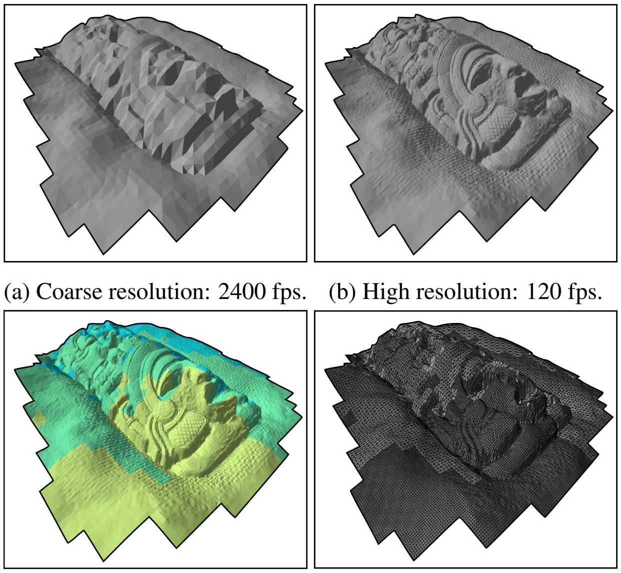
In the previous sections, we described our parametrisation of the model that allows us to build, on the fly, representations of the surface at different scales. However, working with the fixed resolution all the time might not be practical, *e.g.* when the camera is looking at the surface from far or at an oblique angle, and it typically results in aliasing artifacts when the selected resolution is too high compared to the image resolution. Dynamic LOD algorithms specifically address those issues by adapting the complexity of a 3D object representation based on the expected on-screen size or other metrics such as distance to the camera.

To determine the required level of detail and therefore the per-pixel level we have to fuse a measurement into, we evaluate the current estimate of the coarse geometry, the mesh $\mathcal{B}$. Given the camera pose with respect to the mesh, and camera intrinsics, each triangle of the coarsest mesh is projected onto the virtual camera plane and its area is calculated. A parameter controlling the LOD is the desired triangle area, which tells us how many times the triangle should be divided, and the LOD is calculated as follows:

$$l = \text{round}\left(\log_2\left(\frac{[\triangle_{\mathcal{B}}]}{a}\right)\right), \tag{2}$$

where $[\triangle_{\mathcal{B}}]$ indicates the on-screen area of the base triangle, and a is the desired area (we usually set it to 4 pixels). At this stage we can also discard geometry that is clearly not visible in the current frame (*e.g.* is behind the camera) to further improve performance.

Thanks to its simplicity, the proposed method achieves extremely high rendering (and therefore prediction) rates even for complex models. Fig. 4 shows the difference between rendering rates for the dynamic and static LOD models. One limitation of our approach is the fact that the multi-scale representation on a regular grid is usually suboptimal, as triangle boundaries are unlikely to correspond to natural features of the surface. Furthermore, when adjacent triangles are rendered at different resolutions, we have to adjust the LOD in order to address the problem of cracks.



(a) Coarse resolution: 2400 fps. (b) High resolution: 120 fps.

(c) Dynamic res.: 1300 fps. (d) Underlying mesh.

Figure 4: Rendering using different mesh resolutions: (a) coarse mesh, (frame-rate approx. 2400 fps); (b) high resolution mesh, (120 fps); (c) rendering using dynamic level of detail (different colour indicates different LOD) and corresponding mesh (d).

4.3. Fusion

4.3.1 Single level fusion

We will first explain our fusion algorithm under the assumption that there is only a single resolution mesh. We formulate the surface reconstruction as an optimisation problem, in which we fit the observed data into a predefined surface model, with one degree of freedom per vertex. In this work we assume that the surface model is a simple height field but more generic models are also feasible.

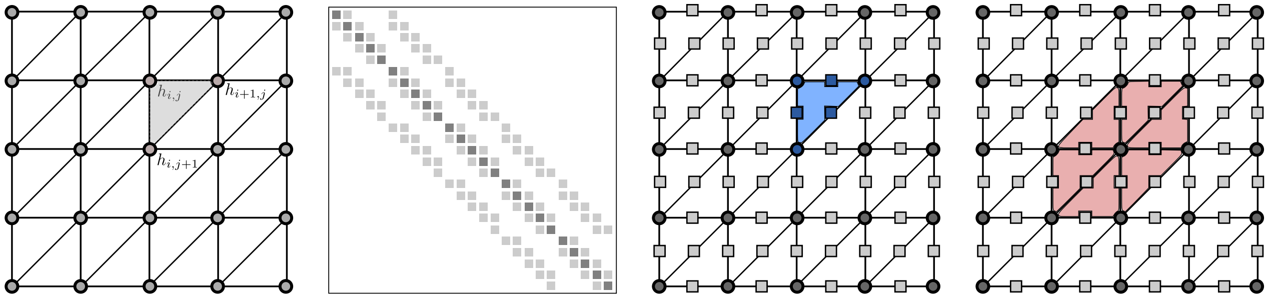
Let d_i be a depth measurement for the pixel location (u_i, v_i). First we back-project the depth measurement into 3D space and associate it with the triangle is falls onto. Using the camera intrinsics matrix $\mathtt{K}$ and camera pose $\mathtt{T}^{wc}$, we transform the depth measurement into a height measurement in the global frame of reference as follows:

$$\mathbf{P} = \mathtt{T}^{wc} d_i \mathtt{K}^{-1} \dot{\mathbf{p}}_i , \tag{3}$$

where $\mathbf{P} = (x_i, y_i, z_i)$ and $\dot{\mathbf{p}}_i = (u_i, v_i, 1)$. If there is an uncertainty measure σ_i^d associated with the depth measurement d_i, we can also calculate uncertainty in the elevation σ_i^z using the rules of error propagation.

Let us assume that the surface has a form $z = f(x, y)$, which in our case is a triangular mesh controlled by a set of height variables $\mathbf{h} \in \mathbb{R}^m$, where $m = n \times n$ and n is the (rectangular) grid dimension. A 3D point $\mathbf{P} = (x_i, y_i, z_i)$, can be associated with a triangle of the surface, and we can predict the height at (x_i, y_i) by using barycentric coordinates within this triangle $\mathbf{v}_i = (\alpha_i, \beta_i, \gamma_i)^\top$, in the following way:

$$\hat{z}_i = \alpha_i h_1^{\triangle_i} + \beta_i h_2^{\triangle_i} + \gamma_i h_3^{\triangle_i} , \tag{4}$$



(a) Example of a mesh.

(b) Associated $\mathtt{J}^\top\mathtt{J}$ matrix.

(c) $\mathtt{J}^\top\mathtt{J}$ stored on a grid.

(d) Variables for an element-wise Gauss-Seidel iteration.

Figure 5: We use a regular grid structure to represent the mesh and the elements of the $\mathtt{J}^\top\mathtt{J}\mathbf{h} = \mathtt{J}^\top\mathbf{z}$ equation. a) An example of a 5×5 grid with used triangulation. Each vertex is connected to only 6 adjacent vertices. b) Structure of the $\mathtt{J}^\top\mathtt{J}$ matrix associated with the mesh. c) The matrix $\mathtt{J}^\top\mathtt{J}$ can be stored efficiently using a grid of size $(2n-1) \times (2n-1)$. Dots indicate the diagonal entries of the matrix; squares represent the off-diagonal entries. A single height measurement updates $\mathtt{J}^\top\mathtt{J}$ locally, as shown by the example blue triangle. d) During an element-wise Gauss-Seidel iteration, we access only a small subset of entries in the $\mathtt{J}^\top\mathtt{J}$ matrix; here the red area indicates the support for the $h_{i,j+1}$ vertex.

where $h_1^{\triangle_i}, h_2^{\triangle_i}, h_3^{\triangle_i}$ represent the heights of the triangle associated with the point (x_i, y_i, z_i). The conversion from grid coordinates to barycentric coordinates, $(x_i, y_i) \rightarrow (\alpha_i, \beta_i, \gamma_i)$, is straightforward and will be omitted.

A set of k height measurements gives rise to the following set of equations:

$$\begin{aligned} \alpha_1 h_1^{\triangle_1} + \beta_1 h_2^{\triangle_1} + \gamma_1 h_3^{\triangle_1} &= z_1 \\ \alpha_2 h_1^{\triangle_2} + \beta_2 h_2^{\triangle_2} + \gamma_2 h_3^{\triangle_2} &= z_2 \\ &\dots \\ \alpha_k h_1^{\triangle_k} + \beta_k h_2^{\triangle_k} + \gamma_k h_3^{\triangle_k} &= z_k \\ &, \end{aligned} \tag{5}$$

where $\triangle_i$ indicates the triangle a particular height measurement is projected onto. Multiple measurements can be associated with the same triangle, and the set of *linear* equations in Eq. 5 can be written as:

$$\mathtt{J}\mathbf{h} = \mathbf{z}\,, \tag{6}$$

where $\mathtt{J} \in R^{k \times m}$, $\mathbf{h} \in R^m$ and $\mathbf{z} \in R^k$. Note that the matrix $\mathtt{J}$ has only 3 non-zero entries per row.

We solve Eq. 5 in the least squares sense by formulating the normal equation:

$$\mathtt{J}^\top\mathtt{J}\mathbf{h} = \mathtt{J}^\top\mathbf{z}\,. \tag{7}$$

The matrix $\mathtt{J}^\top\mathtt{J}$ on the left-hand side is symmetric and sparse and has a regular structure that reflects the topology of the mesh used, as shown in Fig. 5. In our case, a single vertex can be connected to only up to 6 neighbouring vertices, so $\mathtt{J}^\top\mathtt{J}$ contains per row a diagonal entry and only up to 6 non-zero off-diagonal entries. As a result, rather than storing $\mathtt{J}^\top\mathtt{J}$ using an arbitrary sparse matrix data structure like *e.g.* CSR or COO, we can represent $\mathtt{J}^\top\mathtt{J}$ conveniently also on a regular grid: for a mesh of size $n \times n$ we need a grid of the size $(2n-1) \times (2n-1)$. This is best visualised by the example in Fig. 5 (note that we exploit the symmetry of the matrix). Consequently, we also store the vectors $\mathbf{h}$ and $\mathtt{J}^\top\mathbf{z}$ using $n \times n$ grids.

Instead of first calculating $\mathtt{J}$ and then explicitly performing matrix multiplication, we need only to store $\mathtt{J}^\top\mathtt{J}$ and $\mathtt{J}^\top\mathbf{z}$ and can update them directly, with coefficients that are straightforward to compute. Each height measurement h_i updates $\mathtt{J}^\top\mathtt{J}$ at 6 locations (3 diagonal and 3 off-diagonal entries) associated with its triangle, using the coefficient obtained by taking the outer product (weighted when we take the uncertainty into account) of the barycentric coordinates (Eq. 4):

$$\mathbf{v}_i\mathbf{v}_i^\top = \begin{bmatrix} \alpha_i^2 & \alpha_i\beta_i & \alpha_i\gamma_i \\ \alpha_i\beta_i & \beta_i^2 & \beta_i\gamma_i \\ \alpha_i\gamma_i & \beta_i\gamma_i & \gamma_i^2 \end{bmatrix}. \tag{8}$$

4.3.2 Gauss-Seidel solver

In order to solve Eq. 7 we rely on the Gauss-Seidel method, an iterative solver, which uses an element-wise formula to update the components of the solution vector, $\mathbf{h}$. Compared *e.g.* to the Conjugate Gradient algorithm, the computations are simpler and local, and it is not required to perform a dot product (needing reduction on a GPU), which sometimes might be costly. Methods based on Gauss-Seidel are quite popular for solving large scale partial differential equations on discrete grids or meshes.

Gauss-Seidel's element-wise formula for a system of equations of the form $\mathtt{A}\mathbf{h} = \mathbf{b}$ is as follows:

$$h_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} h_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} h_j^k \right) . \quad (9)$$

The computation of update value $h_i^{(k+1)}$ within iteration $k+1$ uses only a small subset of the entries in matrix $\mathtt{A}$ and vector $\mathbf{b}$, as well as values from the solution vector that have already been updated $h^{(k+1)}$, and values from the previous iteration h^k. This means that for each vertex we only need to access its six surrounding neighbours within the vector $\mathbf{h}$ and the associated off-diagonal entries of the matrix $\mathtt{J}^\top \mathtt{J}$. The local form of the update rule makes it straightforward to execute, but the dependency between variables means that in a standard form, Gauss-Seidel is a serial algorithm and the computations for each of the height elements cannot be done in parallel. Fortunately, we can apply variable ordering (four-colour reordering [25, p. 95]) and divide the grid in 4 sets of independent variables, where computations within a set can be executed completely in parallel.

4.3.3 Incremental reconstruction

In our fusion framework we can process each depth map as it arrives. With every new frame, we update $\mathtt{J}^\top \mathtt{J}$ and $\mathtt{J}^\top \mathbf{z}$ and run only a few iterations of the Gauss-Seidel solver as it typically converges very fast. Since we are solving the linear least squares problem iteratively, we can always stop the solver and add new data (*i.e.* update the $\mathtt{J}^\top \mathtt{J}$ and $\mathtt{J}^\top \mathbf{z}$ according to Eq. 8), and then resume the optimisation. Note that all previous measurements are summarised in the $\mathtt{J}^\top \mathtt{J}$ and $\mathtt{J}^\top \mathbf{z}$, and that the computations and memory requirements are bounded. From the estimation perspective, this approach corresponds to an Information Filter, with matrix $\mathtt{J}^\top \mathtt{J}$ being the inverse covariance matrix and vector $\mathtt{J}^\top \mathbf{z}$ the information vector. Note, that this is in contrast to the method proposed by [36] who cast the depth map fusion as a nonlinear least squares problem and therefore formulated the fusion as an Extended Information Filter. There, the data association and linearisation point can change from iteration to iteration, one has to make sure that the solver converges before a new data/depth map can be processed.

4.3.4 Multi-scale fusion

Our multi-scale fusion approach combines Laplacian-based surface decomposition, dynamic tessellation and level of detail, and optimisation-based surface reconstruction within a single framework.

Given the current estimate of the surface model and camera pose, we first perform dynamic level of detail rendering to tessellate each part of the mesh up to the required resolution. We then proceed with fusion in a coarse-to-fine fashion. Starting from the coarsest level a depth measurement is fused into all the levels up to the selected finest one. Our Laplacian surface parametrisation assumes that the levels are independent, and only contain the details/frequencies that were not captured by the previous level. This means that after a height measurement h_i has been fused into a level k, we first make a prediction of the height at this level, $\hat{h}_i^k$ and in the subsequent level, $k+1$, we only fuse the residual between the predicted height and the measured height:

$$r_i^{k+1} = h_i - \hat{h}_i^k . \quad (10)$$

This is repeated recursively for each measurement, until the required level of detail has been reached. Before fusing into the next finest level, we make sure that the optimisation has converged, and only proceed into the next resolution level after the vertices in the preceding level have reached certain stability. Here we simply look at the magnitude of the diagonal entries of $\mathtt{J}^\top \mathtt{J}$ associated with the triangle, which are good proxies for the stability, *i.e.* it is the per vertex sum of squared barycentric coordinates from all the measurement thus far. This procedure locks the gauge freedom that would be present if we solved for all heights at different resolutions simultaneously.

5. Experiments

We run a series of experiments on both synthetic and real datasets to demonstrate the practicality and evaluate the performance of our method. We present comparisons with MVE [10], a state-of-the-art off-line, batch-optimisation type method for multi-scale reconstruction, as well as a real-time, point-based method [31] based on the algorithm proposed by Keller *et al.* [15]. We show that our framework can achieve high quality detailed reconstructions but at a runtime comparable with Point-based Fusion (Table 2).

The fusion is implemented entirely on a GPU: the LOD computations and dynamic tessellation (Section 4.2) utilise the tessellation unit of a modern rendering pipeline, introduced in OpenGL 4.0, whereas the computations involving fusion and the solver were implemented in CUDA. Whenever data has to be shared between CUDA and OpenGL we use the OpenGL / CUDA inter-operation feature of the NVidia graphics card. Our implementation (including tracking and depth estimation) achieves real-time performance of 20–25 frames per second on a GTX 680 (most of the time is spent on tracking and depth estimation).

5.1. Synthetic data

To demonstrate the correctness of our incremental reconstruction method we first run an experiment using synthetic data. Fig. 6 shows the results of reconstructing a moon-like surface together with surface error obtained using CloudCompare. As a benchmark, we compare the results with the global optimisation method, MVE. We can see that our

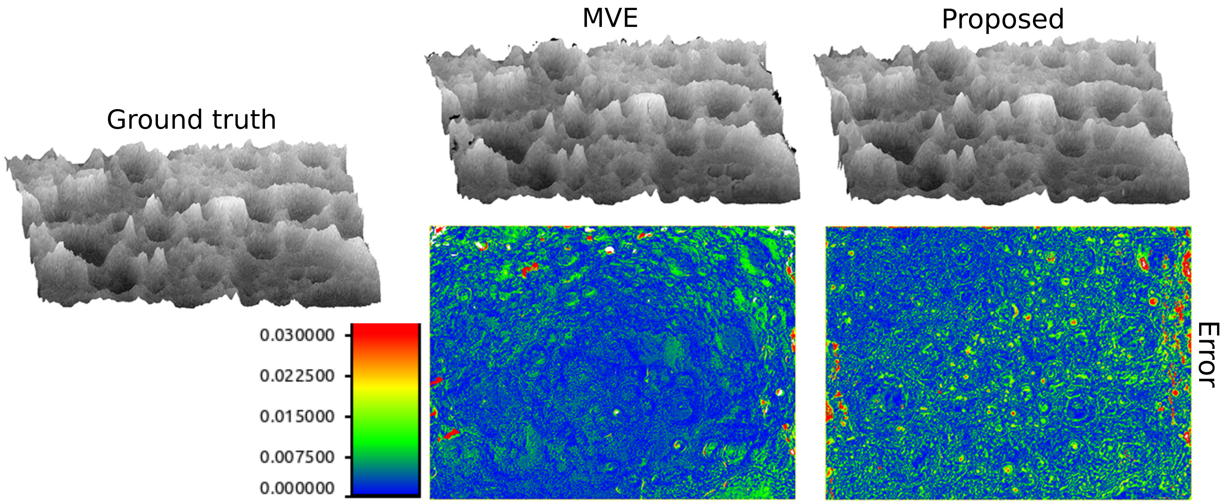


Figure 6: Reconstruction of a synthetic moon surface. Left) Ground truth; Middle) Multi-View-Environment (MVE) [10]; Right) Our method. The heat maps below show reconstruction error.

method is capable of obtaining a good quality surface reconstruction while running two orders of magnitude faster.

	Run time	Avg. error	Std. deviation
Proposed	39 sec.	0.0057	0.034%
MVE	47 min.	0.0037	0.015%

Table 1: Run-time and reconstruction accuracy of our method compared to the off-line, batch optimisation method (MVE [10]).

5.2. Real data

In the Fig. 1 on the first page, we have already shown reconstruction of a real desk-like environment, where within the same framework we can obtain reconstruction of the whole surface as well as of tiny details like coins and paper clips.

Fig. 7 presents additional results and compares our method with the model obtained using Point-based Fusion [15] (in both cases we used ORB-SLAM and our depth estimation method). At the overall scale we obtain qualitatively good results using both approaches, but Point-based Fusion tends to over-smooth the model and cannot handle correctly the significant changes in scale. On the other hand, our method is capable of capturing the overall structure of the scene (although it struggles with sharp vertical edges and cannot handle overhangs properly) while being able to reconstruct tiny details including elements on a circuit board.

	Proposed	Point-based Fusion
Processing time	8.9 ms (111 fps)	11.1 ms (90 fps)

Table 2: Run-time comparison against Point-based Fusion.

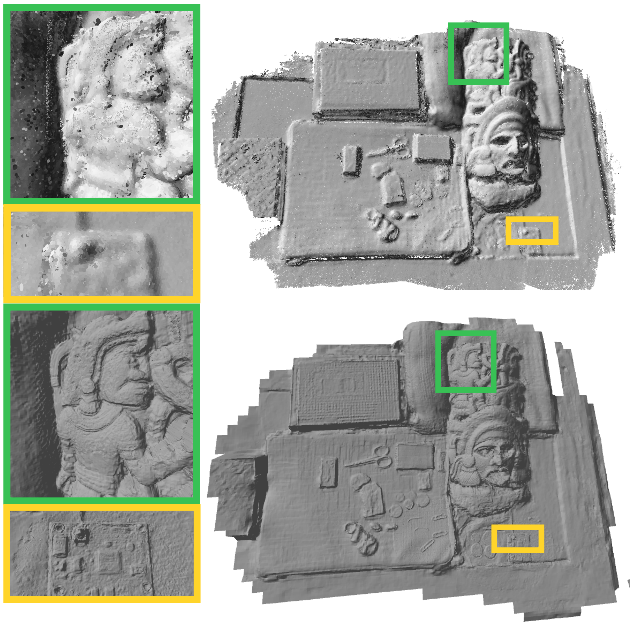


Figure 7: Comparison of our proposed method (bottom row) against Point-based Fusion (top row).

An additional advantage of our approach is that it can provide a user with direct feedback about reconstruction quality. In Fig. 8 different colours indicate the reconstructed level of details for every element of the scene. Yellow means that this part of the scene has been captured with a high level of detail, whereas blue represents only the coars-

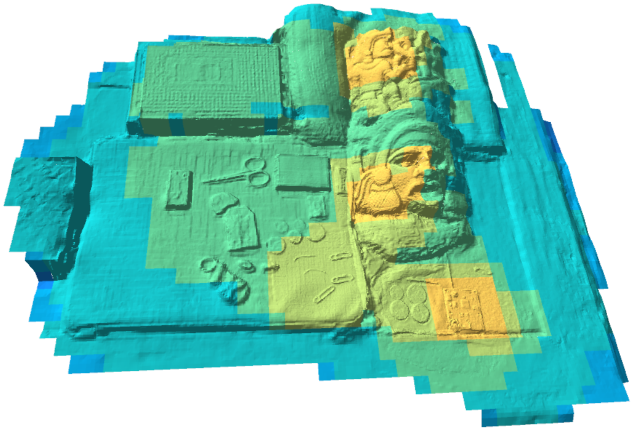


Figure 8: We can obtain feedback about the quality of reconstruction during the scanning process. Here, different colours represent the resolution that the surface element has been reconstructed to (yellow = high, blue=low).

est geometry. In total, to store the model using our adaptive resolution representation, we only need 5.3% of the memory compared to using the full, high resolution mesh.

5.3. Exemplar applications

5.3.1 Mobile robot height map fusion

One of the immediate applications of the proposed method is in the field of mobile robotics. A small robot, *e.g.* a robotic vacuum cleaner, can use forward or downward looking camera to perceive the obstacles [35], and create a map of its environment in form of a height map as for example demonstrated in [36]. Fig. 9 shows an example of results obtained in such a setup. Multi-resolution is strongly advantageous for oblique camera angles, because it allows us to use high resolution directly in front of the robot and low resolution towards the horizon.

5.3.2 Relief/face scanning

The fact that we perform reconstruction using a predefined mesh can allow an easy and robust way to create 3D models of some common structures (*e.g.* face scanning). This is particularly helpful for 3D printing, where a predefined mesh will compensate for missing data and will guarantee that the final model does not contain holes and is directly printable without any additional processing. Fig. 10 shows an example of a real face reconstructed using our algorithm.

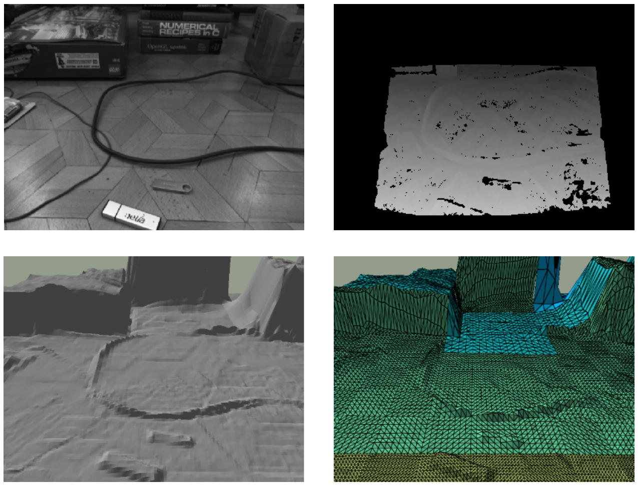


Figure 9: Multi-scale fusion is well suited to height-mapping from an obliquely angled camera. Top row: a typical input image and depth map. Bottom row: reconstructed scene and the tessellation used for the current frame.

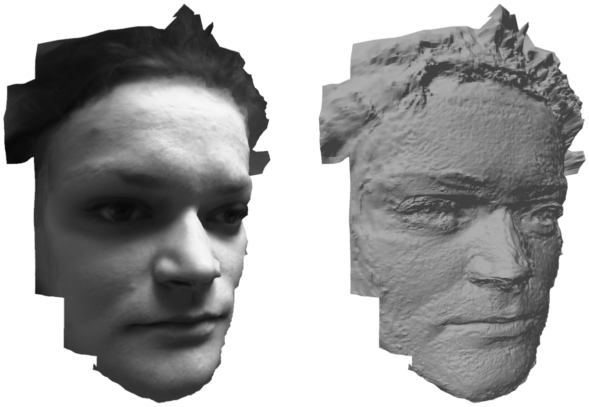


Figure 10: Face reconstruction.

6. Conclusions

We have presented a method for incremental surface reconstruction from a moving camera that can handle scenes with multiple scales. Using the concept of dynamic level of detail we adaptively select the best resolution of the model and fuse measurements in an efficient multi-scale mesh representation.

An obvious limitation of our approach lies in the use of height map. In the future we are looking into ways of extending our framework to more general 3D settings and developing a more flexible multi-scale fusion method. An interesting improvement would be an adaptive mesh refinement based on data and quality of reconstruction that takes into account the complexity of the geometry and would for example represent flat but textured regions with large coarse triangles but high resolution texture.

References

[1] N. Amenta, M. Bern, and M. Kamvysselis. A New Voronoi-based Surface Reconstruction Algorithm. In *Proceedings of SIGGRAPH*, 1998. 3

[2] N. Amenta, S. Choi, and R. K. Kolluri. The Power Crust. In *ACM Symposium on Solid Modeling and Applications*, 2001. 3

[3] P. Burt and E. Adelson. The Laplacian Pyramid as a Compact Image Code. *IEEE Transactions on Communications*, 31(4):532–540, 1983. 3

[4] J. Chen, D. Bautembach, and S. Izadi. Scalable real-time volumetric surface reconstruction. In *Proceedings of SIGGRAPH*, 2013. 2

[5] M. Duchaineau, M. Wolinsky, D. Sigeti, M. Miller, C. Aldrich, and M. Mineev-Weinstein. ROAMing Terrain: Real-time Optimally Adapting Meshes. In *IEEE Conference on Visualization*, 1997. 2

[6] J. Engel, T. Schoeps, and D. Cremers. LSD-SLAM: Large-scale direct monocular SLAM. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2014. 2

[7] C. Forster, M. Pizzoli, and D. Scaramuzza. SVO: Fast Semi-Direct Monocular Visual Odometry. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2014. 2

[8] S. Fuhrmann and M. Goesele. Fusion of depth maps with multiple scales. In *SIGGRAPH Asia*, 2011. 2

[9] S. Fuhrmann and M. Goesele. Floating Scale Surface Reconstruction. In *Proceedings of SIGGRAPH*, 2014. 2

[10] S. Fuhrmann, F. Langguth, and M. Goesele. MVE — A Multi-View Reconstruction Environment. In *EUROGRAPHICS Workshops on Graphics and Cultural Heritage*, 2014. 6, 7

[11] I. Guskov, W. Sweldens, and P. Schröder. Multiresolution Signal Processing for Meshes. In *Proceedings of SIGGRAPH*, 1999. 3

[12] H. Hoppe. Progressive Meshes. In *Proceedings of SIGGRAPH*, 1996. 2

[13] H. Hoppe. View-Dependent Refinement of Progressive Meshes. In *Proceedings of SIGGRAPH*, 1997. 2

[14] H. Hoppe. Smooth view-dependent level-of-detail control and its application to terrain rendering. In *IEEE Conference on Visualization*, 1998. 2

[15] 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 *Proc. of Joint 3DIM/3DPVT Conference (3DV)*, 2013. 1, 2, 6, 7

[16] P. Lindstrom, D. Koller, W. Ribarsky, L. F. Hodges, N. Faust, and G. Turner. Real-Time, Continuous Level of Detail Rendering of Height Fields. In *Proceedings of SIGGRAPH*, 1996. 2

[17] F. Losasso and H. Hoppe. Geometry Clipmaps: Terrain Rendering using Nested Regular Grids. In *Proceedings of SIGGRAPH*, 2004. 2

[18] J. Lu, K. Shi, D. Min, L. Lin, and M. N. Do. Cross-based local multipoint filtering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012. 3

[19] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós. ORB-SLAM: a Versatile and Accurate Monocular SLAM System. *IEEE Transactions on Robotics (T-RO)*, 31(5):1147–1163, 2015. 2

[20] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohli, J. Shotton, S. Hodges, and A. Fitzgibbon. KinectFusion: Real-Time Dense Surface Mapping and Tracking. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, 2011. 1, 2

[21] R. A. Newcombe, S. Lovegrove, and A. J. Davison. DTAM: Dense Tracking and Mapping in Real-Time. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2011. 2, 3

[22] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger. Real-time 3D Reconstruction at Scale using Voxel Hashing. In *Proceedings of SIGGRAPH*, 2013. 2

[23] V. Pradeep, C. Rhemann, S. Izadi, C. Zach, M. Bleyer, and S. Bathiche. MonoFusion: Real-time 3D reconstruction of small scenes with a single web camera. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 83–88, 2013. 2

[24] C. Rhemann, A. Hosni, M. Bleyer, C. Rother, and M. Gelautz. Fast cost-volume filtering for visual correspondence and beyond. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011. 3

[25] Y. Shapira. *Matrix-based Multigrid: Theory and Applications*. Springer, second edition, 2008. 6

[26] F. Steinbrücker, C. Kerl, J. Sturm, and D. Cremers. Large-scale multi-resolution surface reconstruction from RGB-D sequences. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2013. 2

[27] J. Stückler and S. Behnke. Multi-resolution surfel maps for efficient dense 3d modeling and tracking. *Journal of Visual Communication and Image Representation*, 25(1):137–147, 2014. 2

[28] D. Terzopoulos and D. Metaxas. Dynamic 3D Models with Local and Global Deformations: Deformable Superquadrics. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 13(7):703–714, 1991. 3

[29] D. Terzopoulos, J. Platt, A. Barr, and K. Fleischer. Elastically deformable models. In *Proceedings of SIGGRAPH*, 1987. 3

[30] B. Ummenhofer and T. Brox. Global, Dense Multiscale Reconstruction for a Billion Points. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2015. 2

[31] T. Whelan, S. Leutenegger, R. F. Salas-Moreno, B. Glocker, and A. J. Davison. ElasticFusion: Dense SLAM without a pose graph. In *Proceedings of Robotics: Science and Systems (RSS)*, 2015. 6

[32] T. Whelan, J. B. McDonald, M. Kaess, M. Fallon, H. Johannsson, and J. J. Leonard. Kintinuous: Spatially Extended KinectFusion. In *Workshop on RGB-D: Advanced Reasoning with Depth Cameras, in conjunction with Robotics: Science and Systems*, 2012. 2

[33] R. Zabih and J. Woodfill. Non-parametric Local Transforms for Computing Visual Correspondence. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 1994. 3

[34] K. Zhang, J. Lu, and G. Lafruit. Cross-Based Local Stereo Matching Using Orthogonal Integral Images. *IEEE Transactions on Circuits and Systems for Video Technology*, 19(7):1073–1079, 2009. 3

[35] J. Zienkiewicz and A. J. Davison. Extrinsics Autocalibration for Dense Planar Visual Odometry. *Journal of Field Robotics (JFR)*, 32(5):803–825, 2015. 8

[36] J. Zienkiewicz, A. J. Davison, and S. Leutenegger. Real-Time Height-Map Fusion using Differentiable Rendering. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*, 2016. 6, 8