applied sciences

MDPI

Article

Real-Time 3D Reconstruction Pipeline for Room-Scale, Immersive, Medical Teleconsultation

Ulrich Eck [1,*,†], Michael Wechner [1], Frieder Pankratz [2], Kevin Yu [1,3], Marc Lazarovici [2] and Nassir Navab [1]

[1] School of CIT, Technische Universität München, Chair for Computer Aided Medical Procedures, Boltzmannstr. 3, 85748 Garching, Germany; michael.wechner@tum.de (M.W.); kevin.yu@tum.de (K.Y.); nassir.navab@tum.de (N.N.)
[2] Medphoton GmbH, Technische Universität München, Karolingerstraße 16, 5020 Salzburg, Austria; frieder.pankratz@med.uni-muenchen.de (F.P.); marc.lazarovici@med.uni-muenchen.de (M.L.)
[3] Institute for Emergency Medicine, Ludwig-Maximilian Universität, Schillerstr. 53, 80336 München, Germany
* Correspondence: ulrich.eck@tum.de; Tel.: +49-89-289-19412
† Current address: Boltzmann Str. 3, 85748 Garching b. München, Germany.

Abstract: Medical teleconsultation was among the initial use cases for early telepresence research projects since medical treatment often requires timely intervention by highly specialized experts. When remote medical experts support interventions, a holistic view of the surgical site can increase situation awareness and improve team communication. A possible solution is the concept of immersive telepresence, where remote users virtually join the operating theater that is transmitted based on a real-time reconstruction of the local site. Enabled by the availability of RGB-D sensors and sufficient computing capability, it becomes possible to capture such a site in real time using multiple stationary sensors. The 3D reconstruction and simplification of textured surface meshes from the point clouds of a dynamic scene in real time is challenging and becomes infeasible for increasing capture volumes. This work presents a tightly integrated, stateless 3D reconstruction pipeline for dynamic, room-scale environments that generates simplified surface meshes from multiple RGB-D sensors in real time. Our algorithm operates directly on the fused, voxelized point cloud instead of populating signed-distance volumes per frame and using a marching cube variant for surface reconstruction. We extend the formulation of the dual contouring algorithm to work for point cloud data stored in an octree and interleave a vertex-clustering-based simplification before extracting the surface geometry. Our 3D reconstruction pipeline can perform a live reconstruction of six incoming depth videos at their native frame rate of 30 frames per second, enabling the reconstruction of smooth movement. Arbitrarily complex scene changes are possible since we do not store persistent information between frames. In terms of mesh quality and hole filling, our method falls between the direct mesh reconstruction and expensive global fitting of implicit functions.

Keywords: telepresence; real-time reconstruction; medical consultation

Citation: Eck, U.; Wechner, M.; Pankratz, F.; Yu, K.; Lazarovici, M.; Navab, N. Real-Time 3D Reconstruction Pipeline for Room-Scale, Immersive, Medical Teleconsultation. *Appl. Sci.* **2023**, *13*, 10199. https://doi.org/10.3390/app131810199

Academic Editor: Zhonghua Sun

Received: 10 July 2023
Revised: 7 August 2023
Accepted: 16 August 2023
Published: 11 September 2023

1. Introduction

Immersive telepresence has been a research domain for many decades, driven by the vision of enabling experts to collaborate over distance. Steuer [1] defined telepresence as "the experience of being present in an environment using a communication medium". The concept of telepresence encompasses a multitude of different topics, ranging from social, perceptual, and behavioral to technical aspects of interacting within a remote environment. An important part of telepresence is digitizing and transmitting a physical space over a communication medium to allow remote users to join that space virtually.

The computational complexity of reconstructing a dynamic environment in real time and the bandwidth of the communication medium are typically limiting factors when designing immersive telepresence systems. Some previous works proposed the transmission

of static scene reconstructions to save bandwidth and reduce complexity [2,3] or limited the transmission to individual participants [4]. Maimone et al. [5] presented a telepresence system capable of transmitting dynamic environments using multiple RGB-D cameras in a lab environment.

Researchers explored various methods to capture a physical space using multiple static cameras and then reconstruct that scene in 3D to transmit it to remote users. Initial approaches used a sea of cameras [6] to reconstruct the environment using multiview stereo (MVS) reconstruction methods. Later, RGB-D cameras were used to improve the capturing of environments in real time with higher fidelity [7,8]. Initially, the environment is captured as separate clouds of points per camera, either through direct acquisition of depth images or by processing color image pairs using multiview stereo algorithms. These individual point clouds are then fused into a common reference frame using intrinsic and extrinsic parameters [9]. Based on the fused point cloud, either direct visualization methods [10,11] can be used to display the reconstructed scene, or surface reconstruction algorithms are used to transform the unordered point cloud into a surface mesh [12]. Depending on the mode of visualization and the processing steps, remote clients receive point clouds, surface meshes, depth images, and corresponding texture information from the color video. The distant physical space is rendered and displayed at the remote clients using some form of virtual or mixed reality display.

Interestingly, several initially published telepresence systems considered medical consultation a strong use case [6,13,14]. When considering the need for highly specialized experts for the various domains of medical interventions that often come with timely treatment, it becomes evident that enabling medical experts to collaborate efficiently over distance would be very beneficial. Medical intervention is a dynamic environment with considerable interaction between stakeholders, patients, and medical equipment. Therefore, reconstruction methods that assume a static environment do not provide a sufficient context for remote experts during most medical interventions. To support the local team with their decision-making process, remote experts would require not only a high-quality view of the patient but also relevant health indicators, access to imaging data and availability of instruments, and medical devices such an intraoperative X-ray [13]. Therefore, for medical consultation, dynamic, room-scale 3D reconstruction in real-time with sufficient detail in regions of interest is an essential building block for modern telepresence solutions.

In this work, we present a real-time 3D reconstruction pipeline for room-scale, immersive medical teleconsultation (see Figure 1). Similar to FusionMLS [15], our pipeline performs stateless mesh reconstruction from multiple depth images. Instead of populating a signed distance volume and applying the marching cubes algorithm [16], our reconstruction method initially generates an octree from the fused point clouds. It directly extracts the mesh using an extended version of the recursive dual contouring algorithm [17] that works on voxelized point clouds instead of signed distance fields. With a voxelized point cloud, fewer surface samples have to be generated than volumetric signed distance samples by at least a factor of two using an optimal sparse data structure, where only corners of sign-changing cells are considered because the vertex generation for the cell interior is skipped. We can also further exploit the octree structure to efficiently simplify the resulting geometries with vertex clustering and connect mixed voxelization resolutions with dual contouring.

Our contributions are as follows: First, we extend the recursive dual contouring algorithm by Ju et al. [17] to include all diagonal connections for cells, which enables the algorithm to be used on point clouds stored in an octree. We define the hole-filling and connectivity criteria for the point-cloud-based algorithm to build correct surface meshes. Furthermore, we adapt vertex clustering simplification to include filtering conditions that preserve the topology and silhouette of nonmanifold, initial surface meshes. We evaluate the proposed method by comparing the reconstruction quality with existing methods and provide real-world performance measurements that show that the proposed pipeline can reconstruct room-scale environments in real time.

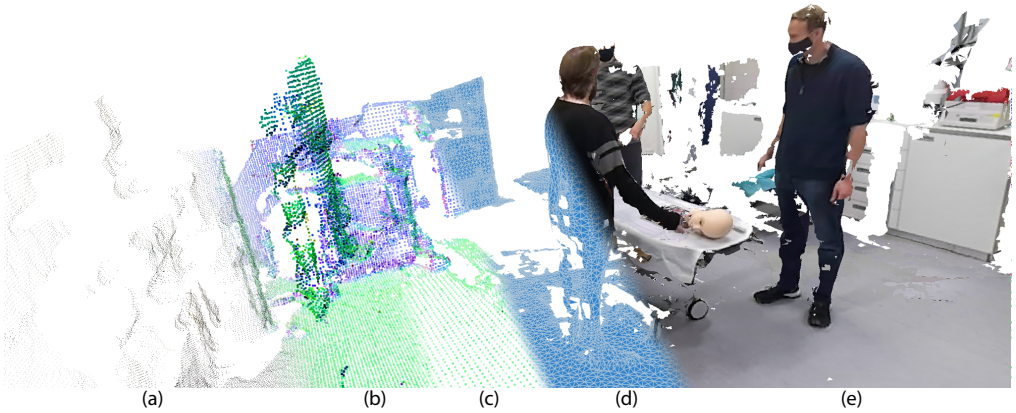


Figure 1. Overview. Processing steps of the proposed 3D reconstruction pipeline from left to right: (**a**) fused point cloud of all depth cameras, (**b**) voxelization with reconstructed surface normals, (**c**) simplification based on vertex clustering of flat surfaces, (**d**) simplified mesh extracted with dual contouring for point clouds, and (**e**) textured mesh with the smallest viewing angle difference to the color camera.

The remainder of this document is structured as follows. In Section 2, we review related work and discuss differences between the proposed and existing methods. In Section 3, we present our proposed pipeline step by step. In Section 4, we introduce our extensions to apply dual contouring on voxelized point clouds and showcase common issues (Section 4.2) that generate interior meshes. Based on this observation, we present the filtering conditions to prevent interior mesh generation during simplification and a method to preserve nonmanifold edges in Section 5. In Section 6, we compare our reconstruction to existing offline direct triangulation and implicit surface reconstruction methods and present performance results for reconstructing a room captured with a ring of six RGB-D cameras. We finally conclude in Section 7 and present future work in Section 8.

2. Related Work

Within the research domain of immersive telepresence, we review relevant literature for real-time 3D reconstruction from point clouds, as shown in Figure 2. Over several decades, researchers explored ways to capture a physical space and transmit it to a distant location [1] to enable immersive, mixed reality telepresence [6,7,14,18,19]. Most previous works depend on some form of visual depth sensing. Multiple view stereo algorithms can be used with calibrated camera pairs [20] to extract surface information by determining the disparity between two rectified images. Such methods require sufficiently textured surfaces for the stereo matching and are computationally demanding for real-time processing. Alternatively, specialized depth sensors with projected infrared patterns [21] or a time-of-flight camera can directly acquire 2.5D surface point clouds [2,5,8,22]. Structured-light capture systems project static infrared textures onto surfaces and, therefore, enable the accurate acquisition of surfaces also from textureless objects and environments. However, such systems can be quite expensive, can have a limited acquisition frame rate, and require extensive calibration. Sensors that measure the time of flight (ToF) are typically small and work well in indoor scenarios. Sufficiently high resolution and frame rates can be captured with modern ToF sensors such as the Microsoft Azure Kinect camera at a relatively low price.

Two distinct concepts have emerged for capturing the local environment: a dynamic camera for static scenes [2,3,23] and multiple stationary cameras for dynamic scenes [5,8,15,24,25]. A large body of work exists for capturing static scenes with moving cameras. Surfaces can be extracted with high quality from moving color images with known camera poses using the structure-from-motion method [26]; however, the process is typically offline. Alternatively, simultaneous localization and mapping [27] combines camera pose estimation and map creation to enable real-time operation. If the camera

sensor also captures depth images, surface reconstruction can be performed by fusing the acquired point clouds using the estimated camera poses [28].

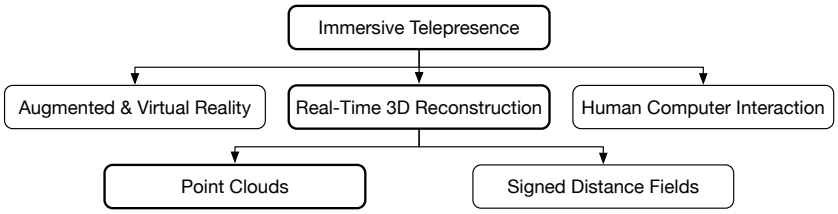


Figure 2. Literature Review. We review the literature in the relevant areas of real-time 3D reconstruction using point clouds as input.

If multiple static, calibrated depth cameras capture a scene, the acquired images can be easily projected into a common reference frame to form an unstructured surface point cloud. A vast amount of work [29] for the reconstruction of implicit surfaces from point clouds is available. One popular formulation is radial basis functions [30,31], which are continuous interpolation functions representing a single smooth and manifold object. Different approaches, such as Poisson reconstruction [32,33], Fourier-based reconstruction [34,35], smooth signed distance surface reconstruction [12], or multilevel partition of unity [36], present different behaviors in terms of robustness, smoothness, and run-time complexity.

Our work focuses on setups consisting of multiple stationary cameras for dynamic environments to capture a room from multiple perspectives as input for surface reconstruction algorithms that work in real time. We further aim to simplify the resulting surface mesh to lower the bandwidth requirements for the communication network while keeping sufficient surface detail to enable medical teleconsultation.

2.1. Real-Time Dynamic Surface Reconstruction

A real-time reconstruction of dynamic scenes from multiple static RGB-D cameras is a challenging task [37]. Generally, existing work can be discerned into three general ideas for generating the surface.

The first family of approaches is based on efficiently performing 2.5D mesh generation by connecting each backprojected vertex with its neighboring depth pixels, which is effectively the 2D Delaunay triangulation [38] of the image plane. Overlapping surfaces need to be combined gracefully with multiple depth cameras like Maimone et al. [5] do during rendering with quality weights for each depth and RGB camera. Alexiadis et al. [39] and Meerits et al. [25] also proposed view-independent approaches that purge redundant triangles and fix the boundary with retriangulation or triangle stitching. Meerits et al. [25] do not disregard redundant data on surfaces but readjust all vertices using a projection onto the local least squares surface.

Second, a more complex class of techniques is based on reconstructing a deformable, canonical model, whose deformations are updated at each frame to facilitate dynamic scene changes. Finding a model that enables arbitrary scene changes is challenging, and complex models will be computationally expensive. In the initial work from Zollhoefer et al. [40], the canonical model was a mesh captured ahead of time. Still, subsequent works [4,41,42] moved to a more flexible TSDF, reconstructed the canonical model live, and supported more dynamic deformations. For human performance capture [43,44], limited topological changes, and small motion or a single RGB-D-camera [45–47], these methods provide a high-quality reconstruction since knowledge is transferred, and occluded parts can be reconstructed if seen prior.

Suppose arbitrarily large and complex scene changes are required. In that case, the third family of approaches discards the idea of the deformation model and builds a new surface representation from the current set of depth images from scratch. While many methods for 3D surface reconstruction from point samples [12,32,33,35,48,49] exist, few are efficient enough to be performed in real time. Maimone et al. [24] presented an

adapted depth image integration of KinectFusion [23], which segments static and moving objects. FusionMLS by Meerits et al. [15] has shown a very efficient TSDF reconstruction, where all SDF samples are generated for each frame and voxel, using neighborhood-based MLS surface reconstruction [50] using a very efficient neighborhood query that collects neighbors in each camera's image space. Subsequently, the mesh is generated using marching cubes [16]. Our work has similar goals as FusionMLS and provides a stateless efficient mesh reconstruction using a volumetric domain. Compared with FusionMLS [15], our approach can generate a simplified mesh and mix different voxel resolutions because we use dual contouring on a regular, sparse octree instead of marching cubes [16] on a less sparse two-layer hierarchy with shared boundary voxels.

2.2. Isosurface Extraction

Many of the approaches highlighted in Section 2.1 generate an implicit surface, and most use marching cubes (MC) on GPUs [16,51] to create the output mesh quickly. Still, many different methods [52] exist to handle the problem of isosurface extraction.

While the original MC method places vertices at the edges of the signed distance voxels, dual methods, such as *SurfaceNets* by Gibson et al. [53], place the vertices of the final mesh in the cell interior. Dual contouring (DC) allows a better reconstruction of sharp details because the vertex placement is not restricted to cell edges. Connecting mixed resolutions is possible using MC but easier with DC because the algorithm can be directly executed on nonuniform hierarchical data structures like Ju et al. [17] presented for octrees without any modifications, and like triangle stitching [54]. However, dual contouring does not guarantee that the resulting surface will be manifold, and the fixes in subsequent work [55,56] require a significantly more complex algorithm. Since the correct topology is sometimes more important, the dual marching cubes method of Schaefer and Warren [57] combines the topology guarantees of MC with sharper detail by placing an additional dual feature vertex into faces generated with MC. The MC method was recently implemented using a data-driven approach [58], highlighting how many perspectives exist for solutions.

We chose DC for our work because the strict run-time requirements and the recursive traversal by Ju et al.[17] are almost as easy to parallelize as regular MC while providing seamless connections between multiple detail levels. Furthermore, the lack of manifold guarantees is insignificant since the geometry of multiple depth cameras is highly nonmanifold, and many edge cases of nonmanifold geometry cannot be captured.

2.3. Mesh Simplification with Dual Contouring

Since the DC method nicely operates on sparse data structures with mixed resolution, Ju et al. [17] demonstrated simplification via octree-based bottom–up vertex clustering with quadratic error functions [59] before mesh extraction, which is much more efficient than other mesh simplification techniques that generally require the high-resolution mesh in memory [60]. However, this approach limits simplification opportunities to the octree structure, which is addressed by enhancing the number of representative vertices per cell using a more advanced encoding as performed by Zhang et al. [61], who increased the limit to two vertices per cell. Schaefer et al. [55] later completely removed this limitation by performing the top–down dual mesh traversal before bottom–up simplification, allowing a list of representative vertices for different surfaces to be contained in each cell with accompanying conditions for manifold preservation. However, their procedure is no longer tail recursive because the results obtained by top–down traversal are required during bottom–up traversal, causing significantly more data dependencies that reduce opportunities for parallelization.

Since performance is our most important requirement and we aimed to implement simplification in GPU kernels [62], we used the octree-based vertex clustering with a single representative vertex, similar to the original work of Ju et al. [17] due to the simpler encoding as a proof of concept.

3. Pipeline Overview

Our processing pipeline for mesh reconstruction consists of five steps (Figure 3). We only accept a synchronized set of depth images as input, which either requires all images to match in a certain time window or an exact match if synchronization is handled beforehand using hardware synchronization. Initially, we use an image-based temporal filter as found in Realsense SDK [63] to reduce noise on the static parts of the depth images.

3.1. Octree Generation and Voxelization

In our proposed pipeline, the first step uses the depth images and camera calibration data to project pixels into an octree of all potentially nonempty voxels of configurable voxel size. Instead of building the octree using bottom–up sorting [64], we build a pointer-based octree from top–down using a heuristic to determine whether a certain arbitrarily sized voxel is occupied. For each of the voxel corners of a potentially occupied child node, the position is projected into each depth camera's image space and compared with the depth value of the residing pixel. If any backprojected depth pixel of one of the eight corners matches up to a query radius threshold defined by the voxel size, it is deemed occupied. For efficiency, the last leaf occupancy query uses only a circumsphere for the voxel center instead of querying all eight corners. This method allows false positives and will generate false negatives if all eight corners of a voxel project into an empty pixel, which is not a significant issue in our testing.

Since we use a pointer-based octree instead of a hash-map-based data structure, such as Zhou et al. [64], data locality is better enforced blockwise using stream compaction [65].

Our approach allows mixing arbitrary voxel sizes. For demonstration purposes, we allow configuring two different low- and high-detail regions separated using an axis-aligned, user-defined bounding box. The subsequent processing steps for normal estimation and simplification also allow separate configurations for the different areas.

3.2. Surface Sample and Normal Estimation

For each potentially nonempty leaf of the octree, all points in a configurable neighborhood radius of the voxel center are collected in image space, using the same image space projection optimization, such as FusionMLS [15]. A surface sample and normal are generated for each voxel using a neighborhood-based local surface estimation method for point clouds. In this work, we utilize principal component analysis (PCA) [66] as a high-speed method and second-degree moving least squares (MLS) [50] for better quality. We apply statistical outlier removal to filter noise points and false positives by discarding every voxel, which does not contain enough points in the neighborhood.

The result of the first two steps is equivalent to performing voxelization, statistical outlier removal, and normal estimation, on the whole, merged point clouds of all depth cameras, which is part of many open-source libraries for point cloud processing (e.g., Open3D [67]). Our optimizations that exploit the compute resources of GPUs and the 2.5D structure of point clouds generated from depth images, which were also used in previous works [15,25], make the process efficient enough to perform in real time.

3.3. Simplification via Octree-Based Vertex Clustering

As Ju et al. proposed initially [17], the mesh simplification for dual contouring is a vertex-clustering-based preprocessing step that replaces interior nodes with a leaf containing the representative sample that is determined by the quadratic error function (QEF) [59], which minimizes the sum of least squared distances of all given input planes (Section 5). This function is ill-conditioned in coplanar cases and, therefore, requires robust linear solvers or high numeric precision. In this work, we use the recently introduced probabilistic extension for QEF by Trettner and Kobbelt [68] for better robustness.

Compared with their work, the conditions for our simplification are required to be more strict to preserve manifoldness and silhouette (Section 5). Like previous steps, the bottom–up

simplification is implemented in CUDA using dynamic parallelism to reduce CPU/GPU overhead by dynamically launching a kernel for all current simplification candidates.

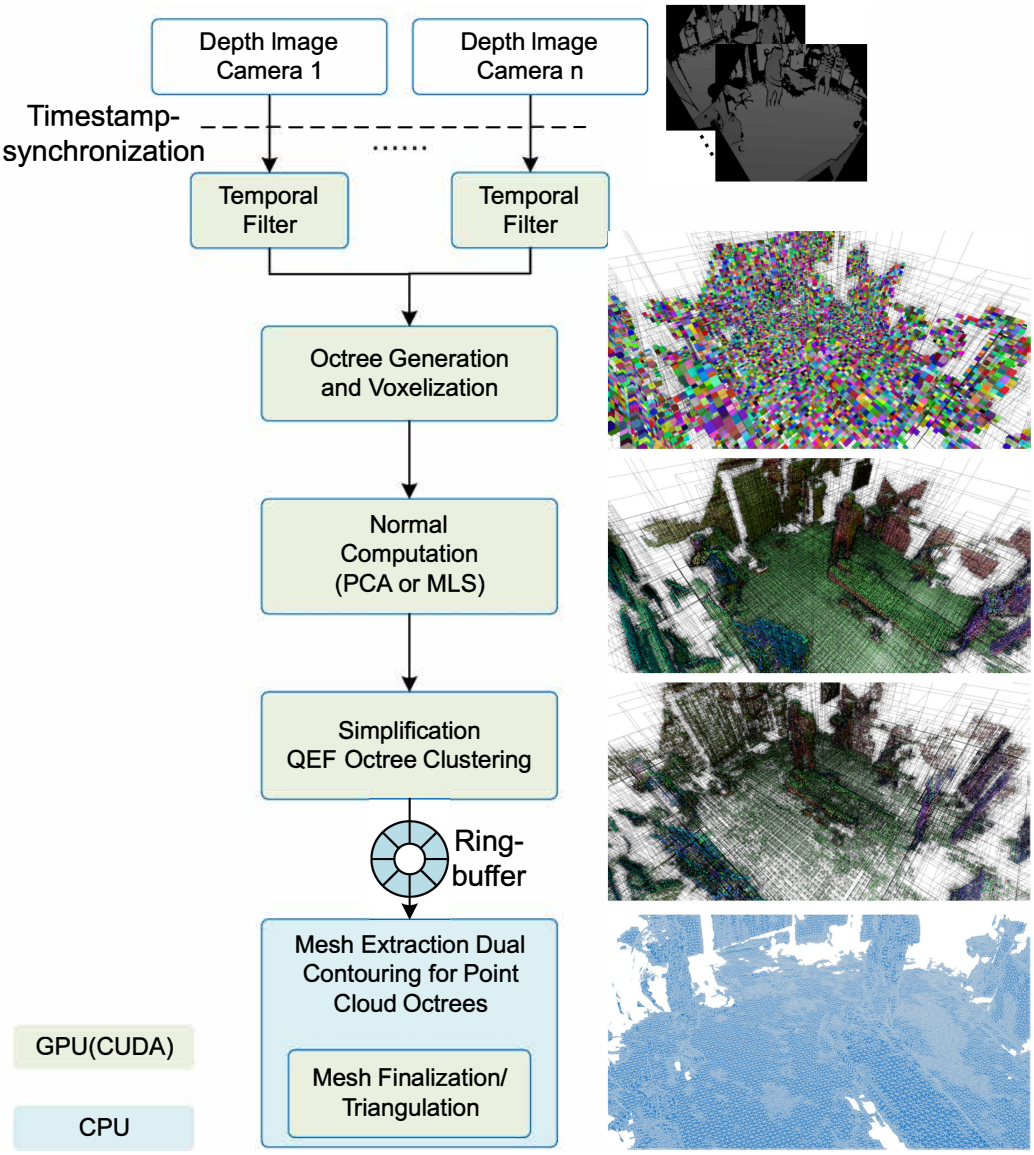


Figure 3. Pipeline Overview. This figure shows the data flow of our reconstruction system. After temporal filtering, we generate an octree of all nonempty voxels, then surface samples and normals are found for each voxel. Simplification decimates octree nodes on flat surfaces. Finally, mesh extraction from the octree is performed using an adapted version of dual contouring for point clouds, and a GPU-based step called finalization performs quad-triangulation, filters duplicate triangles, and removes unused vertices.

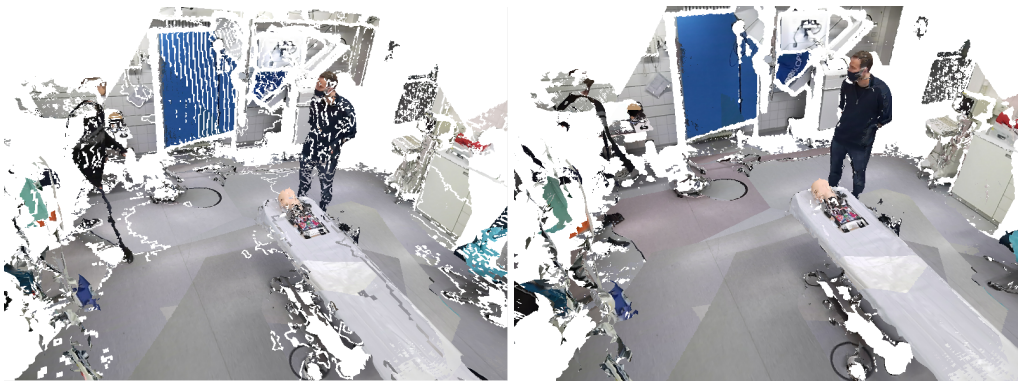
3.4. Dual Contouring for Point Clouds

The final step extracts the mesh from our data structure using recursive octree traversal initially introduced by Ju et al. [17], which we extend to include diagonal connections (Section 4.1). This part is performed on the CPU using work-stealing-based simplification (Section 6.3.1). Since dual contouring generates quad meshes, we perform triangulation, duplicate removal, and vertex list compaction in a final series of GPU kernels called *Finalization*.

3.5. Textured Rendering

Our texturing method Figures 1 and 4) projects each fragment into the image space of the color cameras to gather the optimal pixel and into the depth cameras to check for

occlusion, like in shadow mapping. Since we generate a simplified mesh, choosing a camera per face causes more inconsistencies and discontinuities than dynamically choosing a camera per pixel (Figure 5). If multiple cameras see the same surface, a selection of strategies is available. One strategy, to maximize resolution, chooses the camera with the shortest distance to the observed surface. Since we record the physical shading observed at the color camera, another strategy can use the camera with the smallest viewing angle difference between the current eye position and physical camera to prioritize correct shading.



(**a**) 6-point neighborhood (**b**) 26-point neighborhood

Figure 4. Neighborhood Sizes. Comparison of the neighborhood sizes and its consequences for 3D surface mesh reconstruction. The blue board is not aligned with the axes of the world coordinate system, causing a lot of diagonal connections to be missed in (**a**) that we consider with our method in (**b**).

(**a**) Per-face camera IDs, selected via majority vote of each vertex

(**b**) Per-pixel camera IDs, selected via most similar viewing angle

Figure 5. Mesh Texturing. Comparison between per-face and per-pixel selection of the optimal RGB-D camera. Per-face selection is inferior due to the simplified mesh topology.

4. Dual Contouring for Voxelized Point Clouds

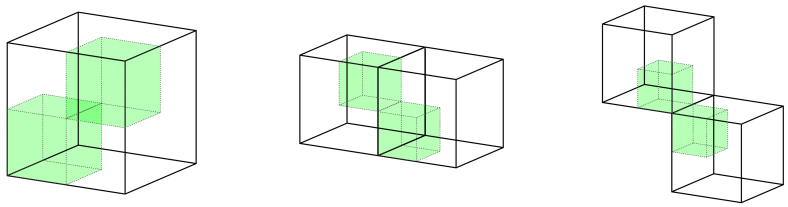
4.1. Extension for Diagonal Connections

Dual contouring is an isosurface extraction method for signed distance fields. In the work of Ju et al. [17], the signed-distance samples and gradients are located at the corners of each cell. Therefore, a dual quad face is generated for each sign-changing cell edge with four incident cells. From the perspective of the sign-changing cell, each dual edge connection is formed across the 6-point neighborhood. The dual vertex is already present in the cell with voxelized point clouds, but it is not guaranteed that all occupied cells will neighbor across a shared edge (Figure 4a). Thus, the neighborhood must be extended to include all sets of four neighboring occupied cells, which share a common corner, i.e., the 26-point neighborhood of a 3D grid (Figure 4b).

To accommodate the larger neighborhood that must be checked for point clouds, we extend the recursive procedures of Ju et al. [17], named cellProc, faceProc, and edgeProc, with two new procedures, which we also name after the shared primal element, diagonalEdgeProc and cornerProc.

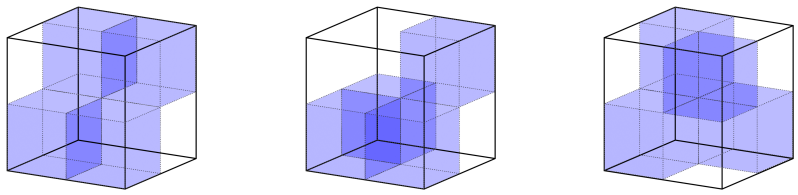
For cellProc, up to 12 subprocedures of diagonalEdgeProc (Figure 6a) and 32 permutations of cornerProc (Figure 7) are added to original 8 cellProc, 12 faceProc, and 6 edgeProc subprocedures. In faceProc, 8 subprocedures of diagonalEdgeProc (Figure 6b) and all 32 combinations of cornerProc (Figure 8a) are added, since they additionally cross through both nodes. For edgeProc, 14 possible cornerProc combinations that involve all input nodes are added to the 2 faceProc subprocedures. If 1 of the 4 nodes in cornerProc is not a leaf, a single recursive case is generated (Figure 8c).

Our diagonalEdgeProc procedure contains at least both involved nodes marked in green (Figure 6) and the 2 additional nodes neighboring this cell edge, if available because only 14 of 32 cornerProc cases are handled in edgeProc. The remaining 18 are handled here since some do not require 4 parent nodes like in Figure 8b. This representation will cause duplicate recursive calls to cornerProc if all 4 nodes in a primal edge are present, which can be solved by not calling the recursions on one of the two directions of diagonalEdgeProc. We chose this encoding for consistency with the original recursion [17]. More optimal encodings without duplicate cases may exist; e.g., if edgeProc is modified to have optional nodes, it accommodates all cases of diagonalEdgeProc.



(**a**) (**b**) (**c**)

Figure 6. Diagonal EdgeProc Subprocedures. Three examples of possible diagonalEdgeProc subprocedures: (**a**) shows 1 of 12 subprocudures in cellProc, (**b**) shows 1 of 8 subprocedures in faceProc, and (**c**) shows 1 of 2 two recursions of diagonalEdgeProc on itself.

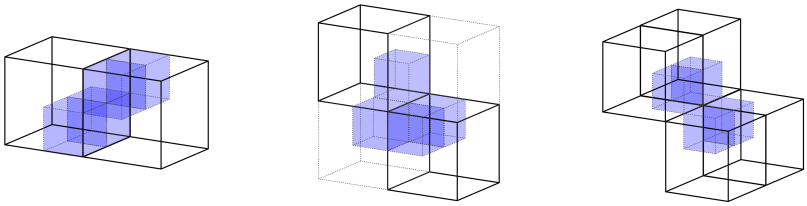


(**a**) (**b**) (**c**)

Figure 7. CornerProc Cases. Three examples of possible cornerProc cases in cellProc: there are (**a**) 6 cases in total where 2 pairs of nodes share an edge, (**b**) 24 cases where just 1 of the nodes is not aligned with the 3 others, and (**c**) 2 situations where 2 nodes are criss-cross on different sides. In sum, this amounts to 32 cases.

4.2. Connectivity Properties and Interior Mesh Generation

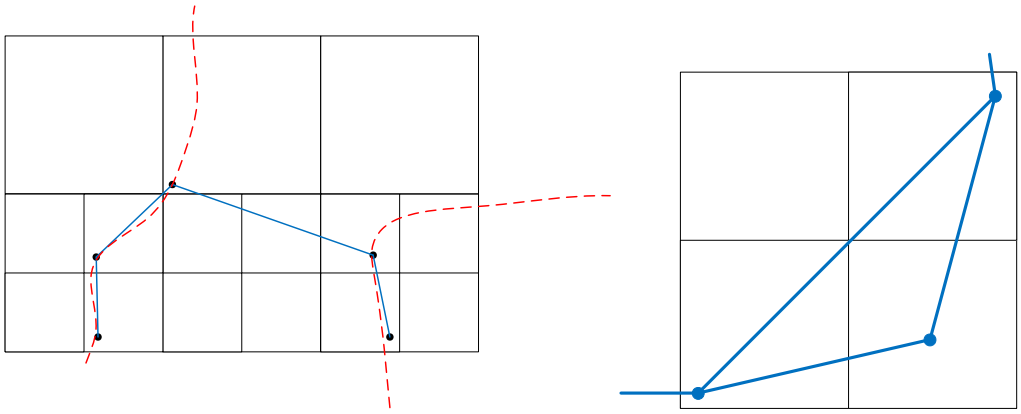
With the extensions for diagonal connections, all combinations of four voxels sharing a common corner in the immediate neighborhood are connected. Thus, our method will only generate holes, where the voxel mesh of all leaf cells would contain holes. One disadvantage of this approach is the lack of discernability between interior and surface mesh parts, which is particularly significant if mixed voxel sizes are combined. For a power of two resolution border 2^c, at least $2^c + 1$ empty, high-resolution voxels must be present between two surfaces to prevent the closing of the wrong gaps (Figure 9a).



(**a**) (**b**) (**c**)

Figure 8. CornerProc Subprocedures. Three examples of cornerProc subprocedures: (**a**) shows 1 of 32 subprocedures in faceProc, (**b**) shows 1 of 18 subprocedures in faceProc, and (**c**) is the recursion for cornerProc.

If the voxel size is uniform, faulty connections still occur, when, for example, a corner is smoothed and hidden by the additional diagonal face (Figure 9b). Discerning these cases is challenging because if the exterior connections were not indicated as in the example of Figure 9b, this case may also be a small manifold object. We omit to filter these cases since the internal faces in a uniform voxel are limited to crossing a single diagonal voxel.



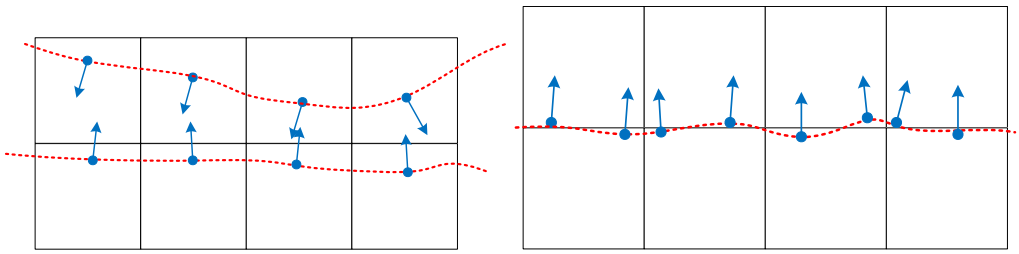
(**a**) (**b**)

Figure 9. Connectivity Properties during Mesh Generation. (**a**) Faulty internal mesh connections, which do not follow the underlying surface across a resolution border and (**b**) in a uniform voxel grid.

If two layers of voxels are supplied as input, a wasteful interior mesh is also generated. For two close-by opposing surfaces (Figure 10a), which are physically unlikely to capture with depth cameras, we only allow connections between vertices with normals that are pointing in the same direction (i.e., only vertices where the angles between all normals of a face are smaller than e.g., 120°), like that used in other direct meshing methods, such as greedy projection triangulation by Marton, Rusu, and Beetz [69]. The case in Figure 10b will happen regardless of noise, if the splitting plane of the octree is badly placed and generates many insignificant interior triangles. We propose solving this problem by merging vertices with normals n_1, n_2 across a dual edge of unit direction d, if

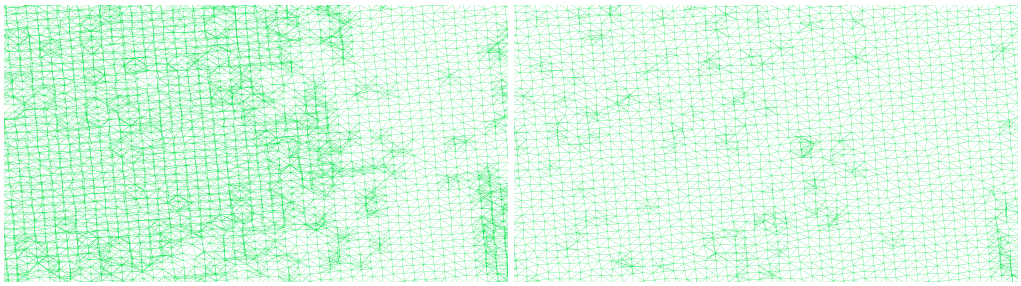
$$n_1 \cdot n_2 > cos(\delta_n) \text{ and } \left| \frac{n_1 + n_2}{\|n_1 + n_2\|} \cdot d \right| > cos(\delta_d) \quad (1)$$

Thus, δ_n specifies the maximum angle between merged vertices, and δ_d denotes the maximum angle between the average normal and the dual edge direction. The two vertices are merged by moving one to the average position and marking the other as deleted with reference to update all face indices later. With this step, we can significantly reduce the number of duplicate triangles if low-quality voxelization is used (Figure 11).



(**a**) (**b**)

Figure 10. Interior Mesh Generation Causes. Two examples of layers of voxels being generated: (**a**) shows two close-by opposite surfaces, and (**b**) visualizes an unfortunate voxel-grid placement for the surface.



(**a**) Without merging vertices: 530.788 triangles (**b**) With merging vertices and $\delta_d = \delta_n = 30°$: 301.214 triangles

Figure 11. Mesh Simplification Example. Close-up mesh of an unsimplified floor in one of our reconstruction frames with and without merging vertices. We used a voxel size of 2 cm and PCA normal reconstruction with a neighborhood radius of 1 cm. The center of the octree is placed on the floor; thus, the two layers of voxels are generated between the sign-changing split plane.

5. Topology and Silhouette-Preserving Simplification

With vertex-clustering-based simplification, which is popular in tandem with dual contouring [17,55,61], the resulting mesh is both simplified and requires less computational resources during extraction.

We use the probabilistic extension of quadratic error functions introduced by Trettner and Kobbelt [68] for our bottom–up simplification due to better robustness and expose a single standard deviation σ_n to configure the normal variance $\Sigma_n = \sigma_n^2 I$. The plane position variance Σ_p is set to 0 because it does not influence the position of the representative vertex and only increases the expectancy value of the simplification error.

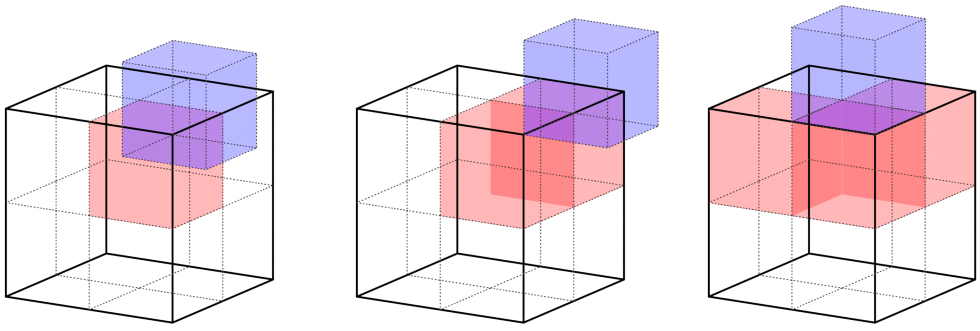
The formulation for the QEF components $A \in \mathbb{R}^{3\times3}, b \in \mathbb{R}^3$, and $c \in \mathbb{R}$ for a single input plane with position $\bar{q}$ and normal $\bar{n}$ as introduced more generally in [68] is therefore

$$\begin{aligned} A &= \bar{n}\bar{n}^T + \sigma_n^2 I \\ b &= \bar{n}\bar{q}^T\bar{n} + \sigma_n^2\bar{q} \\ c &= (\bar{q}^T\bar{n})^2 + \sigma_n^2\bar{q}^T\bar{q} \end{aligned} \tag{2}$$

In our implementation, calculating the QEF solution $x^* = A^{-1}b$ was sufficiently stable with single-precision floating-point numbers and pivoted LU decomposition with nonzero σ_n, which is also equivalent to solving the generalized Tikhonov regularization of the original problem [68].

5.1. Preventing Interior Faces

Simplifying all internal nodes that do not exceed the simplification error is too aggressive since our approach to dual contouring that greedily connects all immediate neighbors will generate nonmanifold geometry and internal faces (Figure 9). We prevent the generation of false internal faces with a simple, but overreaching, heuristic. For each set of empty child nodes of an internal node (red in Figure 12), the external neighbors (blue in Figure 12) must be empty to allow simplification. Since the node would be treated as fully occupied after simplification, the loss of an empty voxel could cause newly formed connections to neighbor nodes. If the blue nodes of Figure 12 are internal nodes, they are also treated as occupied to simplify the query. Moreover, suppose an immediate parent node exists at the preceding depth. In that case, the node is also not simplified because simplification is performed in parallel in a GPU kernel where the filtering condition cannot consistently detect it due to a race condition.



(**a**) (**b**) (**c**)

Figure 12. Conditions for Simplification. Visualization of the simplification conditions to prevent the generation of internal faces. When red child nodes are empty, the neighboring blue nodes must also be empty. In (**a**), the neighbor node only shares a single corner, (**b**) is one of three external neighbors with a shared edge that requires an additional empty internal node in red, and (**c**) is one of three neighbors with a shared face. In total, queries for up to seven external neighbors must be performed in the worst case.

5.2. Silhouette Preservation

Vertex-clustering-based simplification is not designed to preserve the topology, and the QEF error is only correct in a valid 2-manifold. Geometry captured from depth images is highly nonmanifold since many objects will only be seen from one side. If simplification via vertex clustering is applied trivially on nonmanifold edges and corners, they will be pulled in, significantly changing the perceived silhouette in rendering (Figure 13).

To address this issue, each point is classified as interior or exterior to detect which octree nodes must be excluded from simplification. We query the entire 26-point neighborhood for each leaf cell in parallel in a GPU kernel and detect all potentially generated incident faces. To simplify this query, we use the approximation that four incident faces per vertex indicate that the vertex is placed in the mesh interior in a quad face, which is invalid, and exceptions exist (Figure 14a). Furthermore, if three faces are located in the same $2 \times 2 \times 2$ octant of the neighborhood, the face is also in the interior (Figure 14b). All permutations of three incident faces are covered with two conditions: if less than three nodes are empty in the neighborhood described in Figure 14b, the node is an interior node.

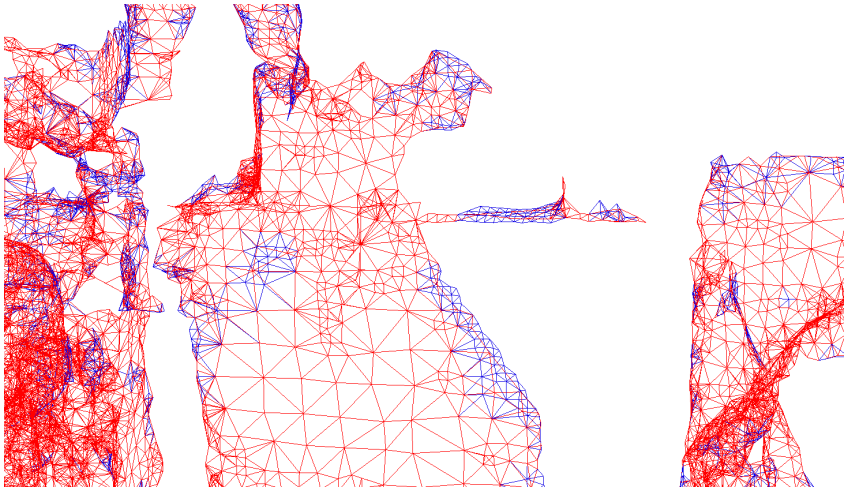
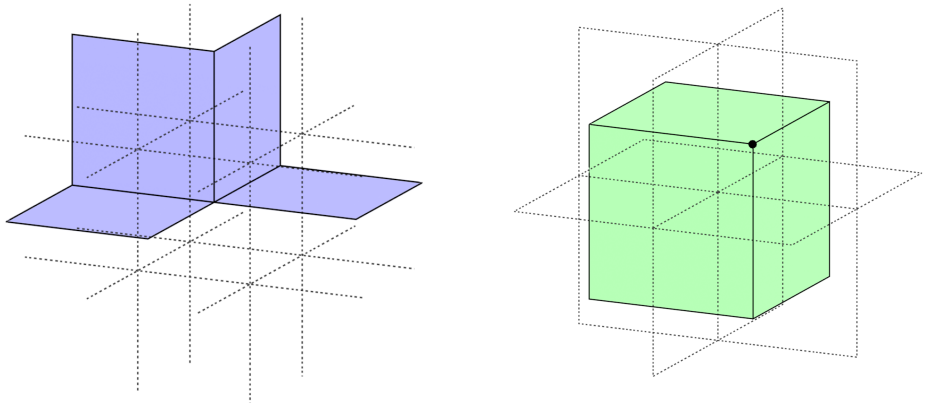


Figure 13. Effects of Vertex Clustering. Close-up wireframe of a simplified mesh, with (blue) and without (red) filtering the nonmanifold vertices from simplification. The vertices of the red mesh are slightly pulled towards the mesh interior.



(**a**) (**b**)

Figure 14. Exterior Mesh Classification. Two exceptions for the rule of four incident faces per vertex: (**a**) is an example of four incident faces that is still located on a nonmanifold edge, and (**b**) indicates a case of only three incident faces that is in the mesh interior.

6. Evaluation

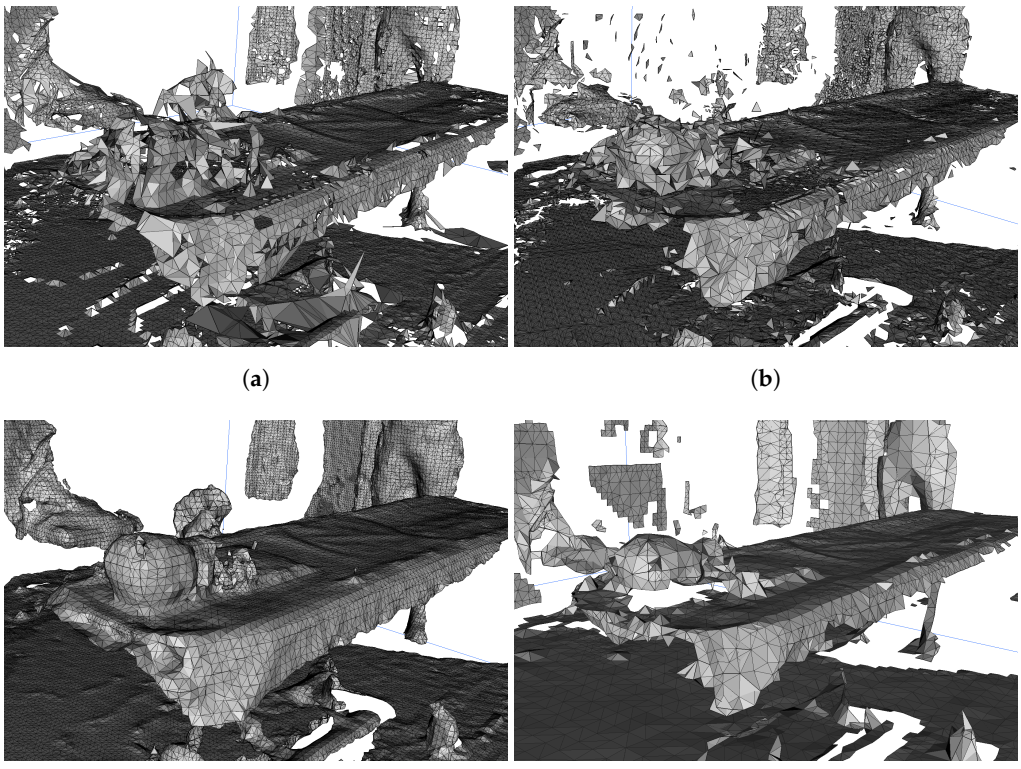
We demonstrate the capabilities of our system using a recording of six ceiling-mounted Azure Kinect DK cameras [70] arranged as a ring, using the `NFOV_UNBINNED` mode on the depth camera and the maximum resolution of 640 × 576. The recording captures a room of 5.88 m × 7.15 m for 59 s, where three persons walk around a surgical table with a dummy head placed on top, and all persons enter and leave the captured domain multiple times. All performance measurements were recorded on an Ubuntu 18.04 server with 2× Intel Xeon Gold 5120 14 core CPUs and an Nvidia RTX A6000 GPU. Our build was compiled using GCC 7.5 and CUDA 11.1.

For our testing, the configuration parameters of the temporal filter [63] are $\alpha = 0.15$, $\delta = 3$ cm, and persistence at 3 (valid in last 2 of 4). For mesh extraction, we use 20 CPU threads and allow a maximum angle difference for vertex normals of 120°, and the parameters to control vertex merging across a splitting plane are $\delta_n = \delta_d = 30°$. While these parameters are constant, we will evaluate four different sets of voxelization and simplification settings. The parameter set *unsimplified-low-detail* uses a 2 cm voxel size and reconstructs the surface samples using principal component analysis across a neighborhood radius of 1.5 cm. The preset *simplified-low-detail* adds simplification with a QEF error threshold of 1×10^{-3} and normal standard deviation $\sigma_n = 0.15$. The third preset, *simplified-*

high-quality, changes the local surface estimation method to moving least squares [50], fitting a second-degree bivariate polynomial surface to the local neighborhood. Finally, the preset *LOD* uses MLS reconstruction with a neighborhood size of 1.5 cm only for the OR table in an axis-aligned box with dimensions of $95 \times 38 \times 180$ cm and a simplification threshold of 1×10^{-4}. The region outside the box uses voxels that are twice as large with 4cm and performs local surface estimation using PCA with 2.5 cm neighborhood radius and a simplification threshold of 1×10^{-3}.

6.1. Comparison with Existing Offline Methods

We compare our implementation with a limited selection of existing, publicly available implementations for mesh reconstruction. As a primary, direct triangulation method of the input point cloud, we compare it with ball pivoting [71], which is implemented in Open3D [67] using ball radii of 1, 2, 3, 4, and 8 cm (Figure 15a). The more optimized greedy projection triangulation [69], which is part of the PCL point cloud processing library [72], greedily connects all points in the projected 2D neighborhood using the point's normals (Figure 15b). For this method, we used a neighborhood radius of 5 cm, a maximum surface angle of 45°, a minimum triangle angle of 10°, and a maximum triangle angle of 120° to find as many connections as possible. For an implicit surface reconstruction method, we compare it with smooth signed distance surface reconstruction (SSD) [12] using the implementation of Michael Misha Kazhdan [73] with an octree depth of 10 and *SurfaceTrimmer* for level 8 or higher to remove all surfaces without many underlying samples. All methods use the unsimplified, voxelized point cloud generated using the first frame of our recording and the *unsimplified-low-detail* preset as their input. Other implicit surface reconstruction techniques, such as variants of Poisson reconstruction [32,33,49], are visually very similar to SSD in our particular dataset.



(**a**) (**b**)

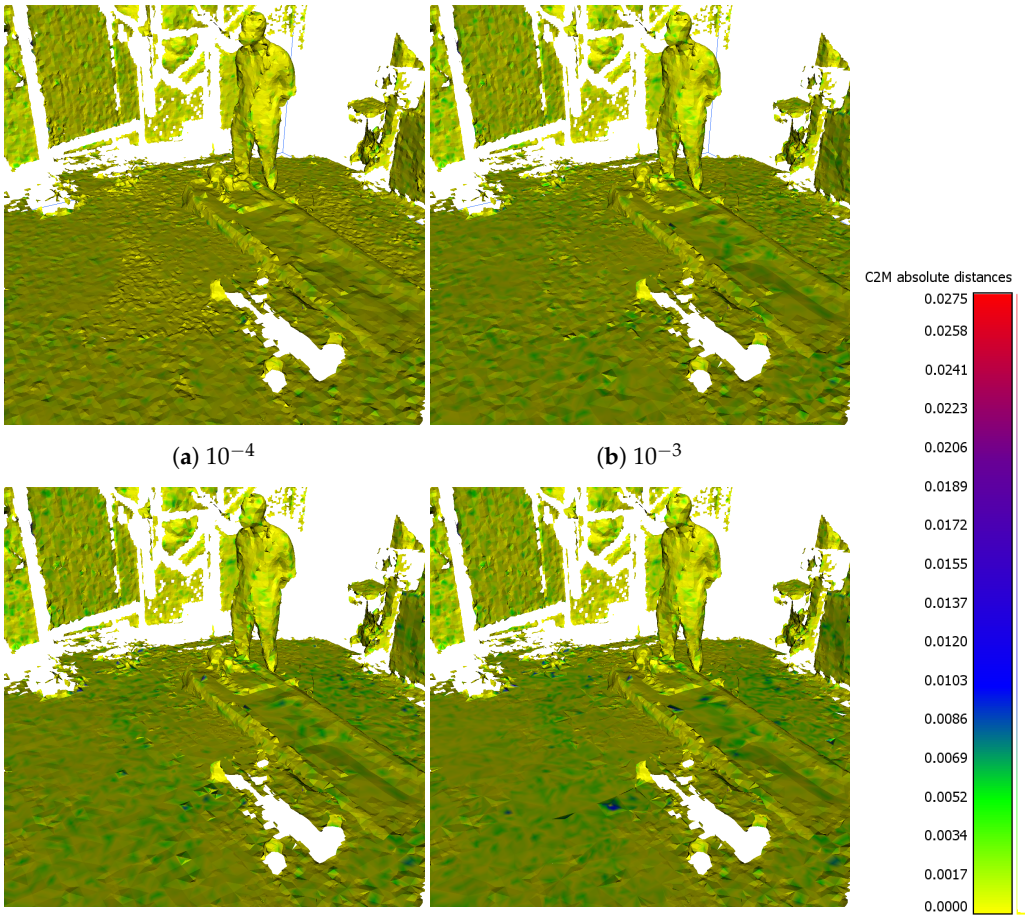
(**c**) (**d**)

Figure 15. Comparison with Existing Approaches. (**a**) Ball-pivoting algorithm [71] with 177,825 triangles, (**b**) greedy projection triangulation of Marton et al. [69] with 187,028 triangles in total, (**c**) trimmed SSD reconstruction [12] with 398,736 triangles in total, and (**d**) our approach with multiple LODs with 49,340 triangles in total.

Our method generates fewer holes than the other general purpose direct meshing techniques (Figures 14b and 15a) but cannot fill arbitrarily large gaps like implicit surface reconstruction (Figure 15c), which also inadvertently fills gaps, such as between the legs of the person in the background. Since our method also uses the voxelized samples directly, the resulting surfaces are not as smooth as the reconstruction with SSD since noisy points are still connected if they are in the immediate neighborhood. Therefore, our mesh extraction method heavily relies on correct, trustworthy samples from voxelization. Due to the simplification of flat surfaces, our method generates significantly fewer triangles.

6.2. Simplification Characteristics

Simplification is a key aspect of our proposed method since it saves bandwidth and subsequent time spent for mesh extraction if many flat surfaces are present in our geometry. We compare the loss of quality with respect to varying QEF error thresholds (Figure 16). The probabilistic normal standard deviation σ_n is fixed at 0.15. As expected and consistent with previous work on QEF [59,68], the mesh error from simplification is small, even if the number of triangles is significantly reduced, particularly since many flat surfaces are present.



(**a**) 10^{-4} (**b**) 10^{-3}

(**c**) 10^{-2} (**d**) 10^{-1}

Figure 16. Evaluation of Simplification Errors. Absolute Euclidean distance in meters between the unsimplified reference and a simplified mesh with varying QEF error thresholds. Computation and visualization were performed with CloudCompare [74].

The simplification process reduces the total uncompressed memory requirement for mesh storage in this scene by approximately 40% (Table 1); thus, an uncompressed streaming application would only require 91 MiB/s instead of approximately 223 MiB/s net bandwidth at 30 meshes per second. The total run time for simplification is at least 10 ms

due to the expensive neighborhood query at the lowest octree level to detect nonmanifold edges (Section 5.2).

Table 1. Simplification Benchmarks. Simplification run time and resulting mesh size for the first frame of our recording (Section 6). The memory requirement is calculated by assuming a simple indexed mesh representation requiring 24 bytes per vertex for position and normal and 12 bytes per triangle.

QEF Threshold	GPU (ms)	Vertices	Triangles	Memory (MiB)	
				rel.	abs.
0	0	117,228	414,820	100%	7.43
1×10^{-5}	10.88	117,055	409,582	99%	7.37
5×10^{-5}	10.65	87,338	295,169	72%	5.38
1×10^{-4}	10.97	72,736	239,102	59%	4.4
1×10^{-3}	11.16	57,611	169,441	44%	3.26
1×10^{-2}	12.3	53,933	157,055	41%	3.03

6.3. Performance Characteristics

6.3.1. CPU Scaling

While most of our pipeline is performed on GPUs, we perform the extended recursive mesh extraction (Section 4.1), which has significantly more procedures to perform than the original work by Ju et al. [17] on CPU using task-based parallelization. We use a simple work-stealing-based scheduler with randomized victim selection based on the work-stealing queue of Lê et al. [75] with the modification to steal tasks in batches of 32 to reduce overhead.

Even though the scheduler has significant overhead (Figure 17a), we can hit our target run time of 31 ms with 14 cores for extracting an unsimplified mesh and 9 cores using a simplified mesh. We chose 31 ms rather than 33 ms for 30 FPS to accommodate CPU/GPU memory copies and mesh finalization, which takes less than 1 ms of GPU time in our measurements (Table 2).

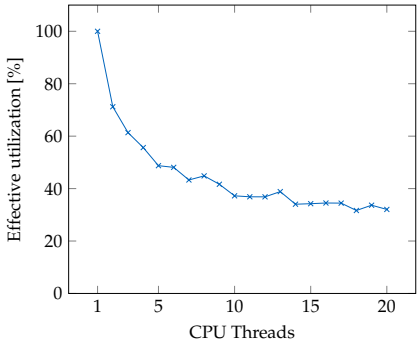
Table 2. Pipeline Benchmarks. Run time of each processing stage using the *simplified-low-detail* preset. GPU times captured using the NVIDIA Nsight Compute `gpu__time_active` metric. Approximately 1500 samples recorded for our entire recording (Section 6).

Processing Stage	Run Time (ms)		
	Min.	Avg.	Max.
Temporal filter (6 images)	0.065	0.125	15.147
Octree generation	0.527	5.691	5.839
PCA surface sample reconstruction	0.012	11.642	41.65
Simplification	0.096	10.814	11.677
mesh extraction (14 CPU cores)	30.347	37.809	55.196
Mesh finalization	0.783	0.825	0.882

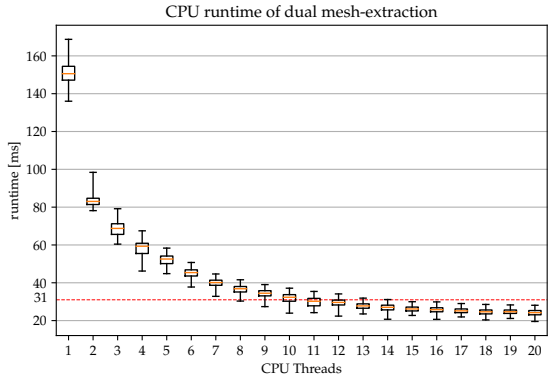
6.3.2. Full System Performance

To process a stream of depth images in real time, our method must keep up with the recording speed of 30 frames per second. Our playback system feeds images into the reconstruction pipeline as fast as possible, allowing observed frame rates above 30 FPS, which we achieve on average through our recording using the *LOD* preset, and only misses by 2 FPS for *unsimplified-* and *simplified-low-detail* (Table 3), which is due to a slight CPU bottleneck (Table 2) in mesh extraction.

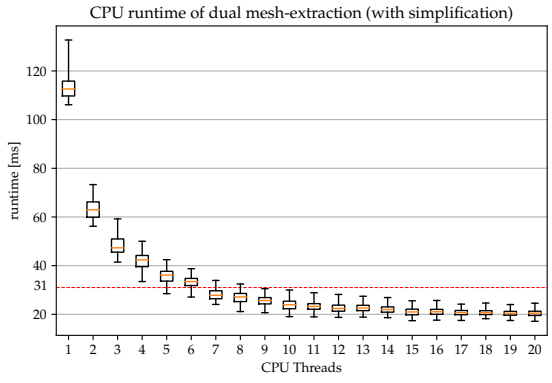
The latency figures (Table 3) are larger than necessary due to our usage of a ring buffer (Figure 3) that queues up to 2 frames to fully saturate the mesh extraction process. If the mesh extraction is not saturated and, therefore, no frames are queued up, like in the *LOD* preset, latency is the sum of all serial processing steps (Table 2) with some communication overhead.



(**a**)



(**b**)



(**c**)

Figure 17. Mesh Extraction Benchmarks. CPU scaling measurements of our mesh extraction: (**a**) effective utilization calculated as a fraction of time spent on tasks relative to the total active CPU time, (**b**) box plot of the total run time with respect to multiple CPU threads using the *unsimplified-low-detail* preset and (**c**) *simplified-low-detail* preset. Using all samples of our recording (Section 6). Box plot whiskers denote the 99th percentile of 1728 measurements.

Table 3. System Performance. Full system performance using a selection of preset settings (Section 6.3.2).

Preset Name	Frames per Second			Latency (ms)		
	Min.	Avg.	Max.	Min.	Avg.	Max.
unsimplified-low-detail	26.1	28.3	31	110	272	305
simplified-low-detail	26.1	28.3	31.3	185	274	303
simplified-high-detail	14.5	24.6	29.6	99	111	192
LOD	17.3	31.8	51.2	45	68	140

7. Discussion

Instead of using a signed distance field as an intermediate representation, we directly generate each voxel cell's representative vertices and surface normal using well-understood point cloud voxelization and local surface estimation [50,66]. With this decision, we effectively only reconstruct data ultimately used in meshing. However, since dual contouring on an SDF does not consider diagonal neighborhoods in the voxel grid, we must extend the recursive traversal to include all 32 diagonal quad face cases (Section 4.1). Furthermore, to utilize simplification, we require an exhaustive condition for the preservation of the connectivity (Section 5.1) and a nonmanifold edge detection (Section 5.2) to preserve the original silhouette.

However, extending the neighborhood also causes unwanted, accidental interior faces constrained by the voxel size (Figure 9b). A manual fix-up of poorly aligned voxel layers is also necessary (Figure 10b), which would not be an issue with implicit surfaces encoded as a signed distance field.

While our method is not intended to compete with general-purpose surface reconstruction methods due to limited hole filling, which requires dense input point clouds to generate a voxelization without holes, our performance results, room size, and number of cameras are on par with the FusionMLS [15] stateless reconstruction system while also performing the simplification of flat surfaces. The efficiency of the task-based CPU parallelization (Figure 17a) of our current prototype can be further improved using better, state-of-the-art lightweight task scheduling systems [76].

8. Conclusions and Future Work

We presented a stateless surface reconstruction method, which is scalable and efficient enough to reconstruct a full room in real time with sufficient quality for immersive teleconsultation. Our method generates a mesh directly from voxelized point clouds. We extended the formulation of Ju's recursive dual-mesh traversal [17] for cases of the diagonal neighborhood and showed how to encode these cases. Like in the original dual-contouring meshing, manifoldness is not guaranteed, which appears in our application if surfaces are too close to resolve correctly in the voxel grid. Holes are only filled if the voxelization itself does not contain holes, and we presented a way of merging or separating close-by surfaces if the grid for voxelization is suboptimally aligned. We also specified additional conditions for safe simplifications, such that no interior faces are generated.

Since we use a dual-meshing method, multiple detail levels with mixed resolutions for voxels are possible, and seamless connections are formed for the final mesh, which makes the method very versatile. Due to the voxel-clustering-based simplification of the octree, the output mesh is much smaller compared with other mesh reconstruction techniques, such as marching cubes or direct 2.5D meshing of all depth pixels [5,15], particularly if the geometry consists of many flat surfaces. Moreover, since simplification is a predecessor of mesh generation, the workload for the subsequent mesh generation is significantly reduced. Given the right parameters, we can provide a reconstruction of a simplified mesh from geometry captured with six fixed, partially overlapping depth cameras in real time directly, without a canonical model or more expensive out-of-order processing steps. The quality of

our reconstruction is better than applying an existing direct-meshing method directly after voxelization, but not as smooth and hole-free as implicit surface reconstruction because we still use the points obtained by voxelization directly.

Currently, we use a per-pixel screen space texturing method for the final mesh to demonstrate our system (Section 3.5), which is prone to errors with slight misalignments of the calibration and requires high-quality depth images. The integration of existing work for consistent texturing [77–79] would be very beneficial to increase visual fidelity and consistency, which is vital for medical telepresence since many features and tools are too small to resolve with geometry using depth cameras. The enhancement of depth images [80–83] could also significantly increase the quality of our visual result and the time warping procedure presented in FusionMLS [15] to merge depth images at different recording timestamps better.

While our method generates a simplified mesh, we currently do not compress the representation during transport, requiring us to use a gigabit connection. Mesh compression [84] libraries, such as Google/Draco [85], add a lot of latency for the large-scale meshes we use. Exploiting the spatiotemporal coherence [86] would yield best compression ratios, and since our mesh is not arbitrary, but generated from an octree, the data structure could be potentially exploited to more efficiently find the relevant frame-to-frame deltas in future works. Furthermore, DNN-based image segmentations for static and dynamic sections, like in Yang et al. [87], have the potential to further reduce the load of dynamic updates through the system by either completely splitting the pipeline, such that no connecting faces between the static and dynamic parts (e.g., feet on the floor) are formed, or through intelligently combining the octrees before mesh extraction to generate seamless connections.

Currently, our filters for simplification are very restrictive since only one representative vertex is contained in each cell. Encoding extensions for multiple vertices per cell, like Zhang et al. [61] presented, would further extend the opportunities for simplification if they are adapted to the highly parallel GPU execution model.

Author Contributions: Conceptualization, U.E. and M.W.; methodology, U.E. and M.W.; software, U.E., M.W. and F.P.; validation, U.E., M.W., F.P., K.Y. and M.L.; resources, M.L. and N.N.; writing—original draft preparation, M.W. and U.E.; writing—review and editing, U.E.; visualization, M.W.; supervision, U.E. and N.N.; project administration, U.E. and M.L.; funding acquisition, U.E. and M.L. All authors have read and agreed to the published version of the manuscript.

Funding: This work was funded by the German Federal Ministry of Education and Research (BMBF), Grant Nos. 16SV8092, 16SV8090, 16SV8088.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Steuer, J. Defining Virtual Reality: Dimensions Determining Telepresence. *J. Commun.* **1992**, *42*, 73–93. [CrossRef]
2. Weibel, N.; Gasques, D.; Johnson, J.; Sharkey, T.; Xu, Z.R.; Zhang, X.; Zavala, E.; Yip, M.; Davis, K. Artemis: Mixed-reality Environment for Immersive Surgical Telementoring. In Proceedings of the Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems, Honolulu, HI, USA, 25–30 April 2020; pp. 1–4.
3. Stotko, P.; Krumpen, S.; Hullin, M.B.; Weinmann, M.; Klein, R. SLAMCast: Large-Scale, Real-Time 3D Reconstruction and Streaming for Immersive Multi-Client Live Telepresence. *IEEE Trans. Vis. Comput. Graph.* **2019**, *25*, 2102–2112. [CrossRef] [PubMed]
4. Dou, M.; Khamis, S.; Degtyarev, Y.; Davidson, P.; Fanello, S.R.; Kowdle, A.; Escolano, S.O.; Rhemann, C.; Kim, D.; Taylor, J.; et al. Fusion4d: Real-time performance capture of challenging scenes. *ACM Trans. Graph. (ToG)* **2016**, *35*, 1–13. [CrossRef]

5. Maimone, A.; Fuchs, H. Encumbrance-free telepresence system with real-time 3D capture and display using commodity depth cameras. In Proceedings of the 2011 10th IEEE International Symposium on Mixed and Augmented Reality, Basel, Switzerland, 26–29 October 2011; IEEE: Manhattan, NY, USA, 2011; pp. 137–146.
6. Fuchs, H.; Bishop, G.; Arthur, K.; McMillan, L.; Bajcsy, R.; Lee, S.; Farid, H.; Kanade, T. Virtual Space Teleconferencing Using a Sea of Cameras. In Proceedings of the 1st International Conference on Medical Robotics and Computer Assisted Surgery (MRCAS '94), Pittsburgh, PA, USA, 22 September 1994 ; pp. 161–167.
7. Fuchs, H.; State, A.; Bazin, J.C. Immersive 3D Telepresence. *Computer* **2014**, *47*, 46–52. [CrossRef]
8. Beck, S.; Kunert, A.; Kulik, A.; Froehlich, B. Immersive Group-to-Group Telepresence. *IEEE Trans. Vis. Comput. Graph.* **2013**, *19*, 616–625. [CrossRef] [PubMed]
9. Beck, S.; Froehlich, B. Volumetric Calibration and Registration of Multiple RGBD-sensors into a Joint Coordinate System. In Proceedings of the 2015 IEEE Symposium on 3D User Interfaces (3DUI), Arles, France, 23–24 March 2015; pp. 89–96.
10. Zwicker, M.; Pfister, H.; van Baar, J.; Gross, M. Surface Splatting. In Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, New York, NY, USA, 1 August 2001; SIGGRAPH '01, pp. 371–378. [CrossRef]
11. Kawata, H.; Kanai, T. Image-Based Point Rendering for Multiple Range Images. In Proceedings of the 2nd International Conference on Information Technology and Applications, Salt Lake City, UT, USA, 15–17 June 2004; pp. 478–483.
12. Calakli, F.; Taubin, G. SSD: Smooth signed distance surface reconstruction. *Comput. Graph. Forum* **2011**, *30*, 1993–2002. [CrossRef]
13. Söderholm, H.M.; Sonnenwald, D.H.; Cairns, B.; Manning, J.E.; Welch, G.F.; Fuchs, H. The Potential Impact of 3d Telepresence Technology on Task Performance in Emergency Trauma Care. In Proceedings of the 2007 International ACM Conference on Supporting Group Work, New York, NY, USA, 4–7 November 2007; GROUP '07, pp. 79–88. [CrossRef]
14. Welch, G.; Sonnenwald, D.H.; Fuchs, H.; Cairns, B.; Mayer-Patel, K.; Yang, R.; Towles, H.; Ilie, A.; Krishnan, S.; Söderholm, H.M.; et al. Remote 3D medical consultation. In *Virtual Realities*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 139–159.
15. Meerits, S.; Thomas, D.; Nozick, V.; Saito, H. FusionMLS: Highly dynamic 3D reconstruction with consumer-grade RGB-D cameras. *Comput. Vis. Media* **2018**, *4*, 287–303. [CrossRef]
16. Lorensen, W.E.; Cline, H.E. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *SIGGRAPH Comput. Graph.* **1987**, *21*, 163–169. [CrossRef]
17. Ju, T.; Losasso, F.; Schaefer, S.; Warren, J. Dual contouring of hermite data. In Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques, San Antonio, TX, USA, 23–26 July 2002; pp. 339–346.
18. Orts-Escolano, S.; Rhemann, C.; Fanello, S.; Chang, W.; Kowdle, A.; Degtyarev, Y.; Kim, D.; Davidson, P.L.; Khamis, S.; Dou, M.; et al. Holoportation: Virtual 3D Teleportation in Real-Time. In Proceedings of the 29th Annual Symposium on User Interface Software and Technology, New York, NY, USA, 16–19 October 2016; UIST '16, pp. 741–754.
19. Pejsa, T.; Kantor, J.; Benko, H.; Ofek, E.; Wilson, A. Room2Room: Enabling Life-Size Telepresence in a Projected Augmented Reality Environment. In Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing, New York, NY, USA, 27 February–2 March 2016; CSCW '16, pp. 1716–1725.
20. Hartley, R.; Zisserman, A. *Multiple View Geometry in Computer Vision*, 2nd ed.; Cambridge University Press: New York, NY, USA, 2003.
21. Collet, A.; Chuang, M.; Sweeney, P.; Gillett, D.; Evseev, D.; Calabrese, D.; Hoppe, H.; Kirk, A.; Sullivan, S. High-Quality Streamable Free-Viewpoint Video. *ACM Trans. Graph.* **2015**, *34*, 1–13. [CrossRef]
22. Roth, D.; Yu, K.; Pankratz, F.; Gorbachev, G.; Keller, A.; Lazarovici, M.; Wilhelm, D.; Weidert, S.; Navab, N.; Eck, U. Real-time mixed reality teleconsultation for intensive care units in pandemic situations. In Proceedings of the 2021 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW), Virtual, 27 March–3 April 2021; IEEE: Manhattan, NY, USA, 2021; pp. 693–694.
23. Izadi, S.; Kim, D.; Hilliges, O.; Molyneaux, D.; Newcombe, R.; Kohli, P.; Shotton, J.; Hodges, S.; Freeman, D.; Davison, A.; et al. KinectFusion: Real-time 3D Reconstruction and Interaction Using a Moving Depth Camera. In Proceedings of the UIST '11 24th annual ACM Symposium on User Interface Software and Technology, Santa Barbara, CA, USA, 16–19 October 2011; ACM: New York, NY, USA, 2011; pp. 559–568.
24. Maimone, A.; Fuchs, H. Real-time volumetric 3D capture of room-sized scenes for telepresence. In Proceedings of the 2012 3DTV-Conference: The True Vision-Capture, Transmission and Display of 3D Video (3DTV-CON), Zurich, Switzerland, 15–17 October 2012; IEEE: Manhattan, NY, USA, 2012; pp. 1–4.
25. Meerits, S.; Nozick, V.; Saito, H. Real-time scene reconstruction and triangle mesh generation using multiple RGB-D cameras. *J. Real-Time Image Process.* **2019**, *16*, 1–13. [CrossRef]
26. Schonberger, J.L.; Frahm, J.M. Structure-from-motion revisited. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 26 June–1 July 2016; pp. 4104–4113.
27. Durrant-Whyte, H.; Bailey, T. Simultaneous localization and mapping: Part I. *IEEE Robot. Autom. Mag.* **2006**, *13*, 99–110. [CrossRef]
28. Newcombe, R.A.; Izadi, S.; Hilliges, O.; Molyneaux, D.; Kim, D.; Davison, A.J.; Kohi, P.; Shotton, J.; Hodges, S.; Fitzgibbon, A. Kinectfusion: Real-time dense surface mapping and tracking. In Proceedings of the 2011 10th IEEE International Symposium on Mixed and Augmented Reality, Basel, Switzerland, 26–29 October 2011; IEEE: Manhattan, NY, USA, 2011; pp. 127–136.

29. Berger, M.; Tagliasacchi, A.; Seversky, L.M.; Alliez, P.; Guennebaud, G.; Levine, J.A.; Sharf, A.; Silva, C.T. A survey of surface reconstruction from point clouds. In *Proceedings of the Computer Graphics Forum*; Wiley Online Library: Hoboken, NJ, USA, 2017; Volume 36, pp. 301–329.
30. Carr, J.C.; Beatson, R.K.; Cherrie, J.B.; Mitchell, T.J.; Fright, W.R.; McCallum, B.C.; Evans, T.R. Reconstruction and representation of 3D objects with radial basis functions. In Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, New York, NY, USA, 1 August 2001; pp. 67–76.
31. Zhou, Z.; Fu, Y.; Zhao, J. An efficient method for surface reconstruction based on local coordinate system transform and partition of unity. *Neural Netw. World* **2020**, *30*, 161. [CrossRef]
32. Kazhdan, M.; Bolitho, M.; Hoppe, H. Poisson surface reconstruction. In Proceedings of the Fourth Eurographics Symposium on Geometry Processing, Sardinia, Italy, 26–28 June 2006; Volume 7, pp. 61–70.
33. Kazhdan, M.; Hoppe, H. Screened poisson surface reconstruction. *ACM Trans. Graph. (ToG)* **2013**, *32*, 1–13. [CrossRef]
34. Kazhdan, M. Reconstruction of solid models from oriented point sets. In Proceedings of the Third Eurographics Symposium on Geometry Processing, Vienna, Austria, 4–6 July 2005; pp. 73–es.
35. Schall, O.; Belyaev, A.; Seidel, H.P. Error-guided adaptive Fourier-based surface reconstruction. *Comput.-Aided Des.* **2007**, *39*, 421–426. .: 10.1016/j.cad.2007.02.005. [CrossRef]
36. Braude, I.; Marker, J.; Museth, K.; Nissanov, J.; Breen, D. Contour-based surface reconstruction using mpu implicit models. *Graph. Model.* **2007**, *69*, 139–157. [CrossRef] [PubMed]
37. Ingale, A.K. Real-time 3D reconstruction techniques applied in dynamic scenes: A systematic literature review. *Comput. Sci. Rev.* **2021**, *39*, 100338. .: 10.1016/j.cosrev.2020.100338. [CrossRef]
38. Delaunay, B. Sur la sphere vide. *Izv. Akad. Nauk SSSR Otd. Mat. I Estestv. Nauk* **1934**, *7*, 1–2.
39. Alexiadis, D.S.; Zarpalas, D.; Daras, P. Real-Time, Full 3-D Reconstruction of Moving Foreground Objects From Multiple Consumer Depth Cameras. *IEEE Trans. Multimed.* **2013**, *15*, 339–358. [CrossRef]
40. Zollhöfer, M.; Nießner, M.; Izadi, S.; Rehmann, C.; Zach, C.; Fisher, M.; Wu, C.; Fitzgibbon, A.; Loop, C.; Theobalt, C.; et al. Real-time non-rigid reconstruction using an RGB-D camera. *ACM Trans. Graph. (ToG)* **2014**, *33*, 1–12. [CrossRef]
41. Newcombe, R.A.; Fox, D.; Seitz, S.M. Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 7–12 June 2015; pp. 343–352.
42. Innmann, M.; Zollhöfer, M.; Nießner, M.; Theobald, C.; Stamminger, M. VolumeDeform: Real-Time Volumetric Non-rigid Reconstruction. In *Proceedings of the ECCV 2016 European Conference on Computer Vision*; Springer: Cham, Switzerland, 2016; pp. 362–379. [CrossRef]
43. Dou, M.; Davidson, P.; Fanello, S.; Khamis, S.; Kowdle, A.; Rhemann, C.; Tankovich, V.; Izadi, S. Motion2fusion: Real-time volumetric performance capture. *ACM Trans. Graph.* **2017**, *36*, 1–16. [CrossRef]
44. Xu, L.; Su, Z.; Han, L.; Yu, T.; Liu, Y.; Fang, L. UnstructuredFusion: Realtime 4D Geometry and Texture Reconstruction Using Commercial RGBD Cameras. *IEEE Trans. Pattern Anal. Mach. Intell.* **2020**, *42*, 2508–2522. [CrossRef] [PubMed]
45. Guo, K.; Xu, F.; Yu, T.; Liu, X.; Dai, Q.; Liu, Y. Real-Time Geometry, Albedo, and Motion Reconstruction Using a Single RGB-D Camera. *ACM Trans. Graph.* **2017**, *36*, 44a. [CrossRef]
46. Slavcheva, M.; Baust, M.; Cremers, D.; Ilic, S. KillingFusion: Non-rigid 3D Reconstruction without Correspondences. In Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017; pp. 5474–5483. [CrossRef]
47. Slavcheva, M.; Baust, M.; Ilic, S. SobolevFusion: 3D Reconstruction of Scenes Undergoing Free Non-Rigid Motion. In Proceedings of the The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Salt Lake City, UT, USA, 18–23 June 2018.
48. Ohtake, Y.; Belyaev, A.; Alexa, M.; Turk, G.; Seidel, H.P. Multi-Level Partition of Unity Implicits. *ACM Trans. Graph.* **2003**, *22*, 463–470. [CrossRef]
49. Kazhdan, M.; Chuang, M.; Rusinkiewicz, S.; Hoppe, H. Poisson Surface Reconstruction with Envelope Constraints. *Comput. Graph. Forum* **2020**, *39*, 173–182. [CrossRef]
50. Alexa, M.; Behr, J.; Cohen-Or, D.; Fleishman, S.; Levin, D.; Silva, C.T. Computing and rendering point set surfaces. *IEEE Trans. Vis. Comput. Graph.* **2003**, *9*, 3–15. [CrossRef]
51. Dyken, C.; Ziegler, G.; Theobalt, C.; Seidel, H.P. High-speed Marching Cubes using HistoPyramids. *Comput. Graph. Forum* **2008**, *27*, 2028–2039. [CrossRef]
52. Newman, T.S.; Yi, H. A survey of the marching cubes algorithm. *Comput. Graph.* **2006**, *30*, 854–879. [CrossRef]
53. Gibson, S.F.F. Constrained elastic surface nets: Generating smooth surfaces from binary segmented data. In *Proceedings of the Medical Image Computing and Computer-Assisted Intervention—MICCAI'98*; Wells, W.M., Colchester, A., Delp, S., Eds.; Springer: Berlin/Heidelberg, Germany, 1998; pp. 888–898.
54. Shu, R.; Zhou, C.; Kankanhalli, M.S. Adaptive marching cubes. *Vis. Comput.* **1995**, *11*, 202–217. [CrossRef]
55. Schaefer, S.; Ju, T.; Warren, J. Manifold Dual Contouring. *IEEE Trans. Vis. Comput. Graph.* **2007**, *13*, 610–619. [CrossRef]
56. Rashid, T.; Sultana, S.; Audette, M.A. Watertight and 2-manifold Surface Meshes Using Dual Contouring with Tetrahedral Decomposition of Grid Cubes. *Procedia Eng.* **2016**, *163*, 136–148. .: 10.1016/j.proeng.2016.11.037. [CrossRef]
57. Schaefer, S.; Warren, J. Dual marching cubes: Primal contouring of dual grids. In Proceedings of the 12th Pacific Conference on Computer Graphics and Applications, Seoul, Republic of Korea, 6–8 October 2004; pp. 70–76. [CrossRef]
58. Chen, Z.; Zhang, H. Neural marching cubes. *ACM Trans. Graph.* **2021**, *40*, 1–15. [CrossRef]

59. Garland, M.; Heckbert, P.S. Surface Simplification Using Quadric Error Metrics. In Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques, Los Angeles, CA, USA, 3–8 August 1997; SIGGRAPH '97, pp. 209–216. [CrossRef]
60. Cignoni, P.; Montani, C.; Scopigno, R. A comparison of mesh simplification algorithms. *Comput. Graph.* **1998**, *22*, 37–54. [CrossRef]
61. Zhang, N.; Hong, W.; Kaufman, A. Dual contouring with topology-preserving simplification using enhanced cell representation. In Proceedings of the IEEE Visualization 2004—Proceedings, Austin, TX, USA, 10–15 October 2004; pp. 505– 512. [CrossRef]
62. Elseberg, J.; Magnenat, S.; Siegwart, R.; Nüchter, A. Comparison of nearest-neighbor-search strategies and implementations for efficient shape registration. *J. Softw. Eng. Robot.* **2012**, *3*, 2–12.
63. Grunnet-Jepsen, A.; Tong, D. *Depth post-processing for intel® realsense™ d400 depth cameras.* New Technology Group, Intel Corporation: Santa Clara, CA, USA, 2018; Volume 3.
64. Zhou, K.; Gong, M.; Huang, X.; Guo, B. Data-Parallel Octrees for Surface Reconstruction. *IEEE Trans. Vis. Comput. Graph.* **2011**, *17*, 669–681. [CrossRef] [PubMed]
65. Bakunas-Milanowski, D.; Rego, V.; Sang, J.; Chansu, Y. Efficient Algorithms for Stream Compaction on GPUs. *Int. J. Netw. Comput.* **2017**, *7*, 208–226. [CrossRef] [PubMed]
66. Hoppe, H.; DeRose, T.; Duchamp, T.; McDonald, J.; Stuetzle, W. Surface Reconstruction from Unorganized Points. In Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques, New York, NY, USA, 1 July 1992; SIGGRAPH '92, p. 71–78. [CrossRef]
67. Zhou, Q.Y.; Park, J.; Koltun, V. Open3D: A Modern Library for 3D Data Processing. *arXiv* **2018**, arXiv:1801.09847.
68. Trettner, P.; Kobbelt, L. Fast and Robust QEF Minimization using Probabilistic Quadrics. *Comput. Graph. Forum* **2020**, *39*, 325–334. [CrossRef]
69. Marton, Z.C.; Rusu, R.B.; Beetz, M. On Fast Surface Reconstruction Methods for Large and Noisy Datasets. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Kobe, Japan, 12–17 May 2009; pp. 3218–3223. [CrossRef]
70. Microsoft. Azure Kinect DK Documentation, 2022. Available online: https://docs.microsoft.com/en-us/azure/Kinect-dk/ (accessed on 15 July 2023).
71. Bernardini, F.; Mittleman, J.; Rushmeier, H.; Silva, C.; Taubin, G. The Ball-Pivoting Algorithm for Surface Reconstruction. *Vis. Comput. Graph. IEEE Trans.* **1999**, *5*, 349–359. [CrossRef]
72. Rusu, R.B.; Cousins, S. 3D is here: Point Cloud Library (PCL). In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Shanghai, China, 9–13 May 2011.
73. Kazhdan, M. Adaptive Multigrid Solvers (Version 13.72), 2021. Available online: https://www.cs.jhu.edu/~misha/Code/PoissonRecon/Version13.72/ (accessed on 15 July 2023).
74. CloudCompare (Version 2.10.2) [GPL Software], 2019. Available online: https://www.cloudcompare.org/ (accessed on 15 July 2023).
75. Lê, N.M.; Pop, A.; Cohen, A.; Zappa Nardelli, F. Correct and Efficient Work-Stealing for Weak Memory Models. *SIGPLAN Not.* **2013**, *48*, 69–80. [CrossRef]
76. Yang, J.; He, Q. Scheduling parallel computations by work stealing: A survey. *Int. J. Parallel Program.* **2018**, *46*, 173–197. [CrossRef]
77. Fu, Y.; Yan, Q.; Yang, L.; Liao, J.; Xiao, C. Texture Mapping for 3D Reconstruction with RGB-D Sensor. In Proceedings of the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 8–23 June 2018; pp. 4645–4653. [CrossRef]
78. Huang, J.; Thies, J.; Dai, A.; Kundu, A.; Jiang, C.; Guibas, L.J.; Niessner, M.; Funkhouser, T. Adversarial Texture Optimization from RGB-D Scans. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Seattle, WA, USA, 13–19 June 2020; pp. 1559–1568.
79. Oliveira, M.; Lim, G.H.; Madeira, T.; Dias, P.; Santos, V. Robust Texture Mapping Using RGB-D Cameras. *Sensors* **2021**, *21*, 3248. [CrossRef]
80. Chen, L.; Lin, H.; Li, S. Depth image enhancement for Kinect using region growing and bilateral filter. In Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012), Tsukuba, Japan, 11–15 November 2012; IEEE: Manhattan, NY, USA, 2012; pp. 3070–3073.
81. Matsuo, K.; Aoki, Y. Depth image enhancement using local tangent plane approximations. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 7–12 June 2015; pp. 3574–3583.
82. Liu, S.; Chen, C.; Kehtarnavaz, N. A computationally efficient denoising and hole-filling method for depth image enhancement. In *Proceedings of the Real-Time Image and Video Processing 2016*; Kehtarnavaz, N., Carlsohn, M.F., Eds.; International Society for Optics and Photonics, SPIE: Bellingham, WA, USA, 2016; Volume 9897, pp. 235–243. [CrossRef]
83. Vosters, L.; Varekamp, C.; Haan, G. Overview of Efficient High-Quality State-of-the-Art Depth Enhancement Methods by Thorough Design Space Exploration. *J. Real-Time Image Process.* **2019**, *16*, 355–375. [CrossRef]
84. Maglo, A.; Lavoué, G.; Dupont, F.; Hudelot, C. 3D Mesh Compression: Survey, Comparisons, and Emerging Trends. *ACM Comput. Surv.* **2015**, *47*, 1–41. [CrossRef]
85. Galligan, F.; Hemmer, M.; Stava, O.; Zhang, F.; Brettle, J. Google/Draco: A Library for Compressing and Decompressing 3D Geometric Meshes and Point Clouds, 2018. Available online: https://github.com/google/draco (accessed on 15 July 2023).

86. Arvanitis, G.; Lalos, A.S.; Moustakas, K. Fast Spatio-temporal Compression of Dynamic 3D Meshes. *arXiv* **2021**, arXiv:2111.10105.
87. Yang, S.; Wang, J.; Wang, G.; Hu, X.; Zhou, M.; Liao, Q. Robust RGB-D SLAM in dynamic environment using faster R-CNN. In Proceedings of the 2017 3rd IEEE International Conference on Computer and Communications (ICCC), Paris, France, 21–25 May 2017; pp. 2398–2402.