Surfel-Based Incremental Reconstruction of the Boundary between Known and Unknown Space

Surfel-Based Incremental Reconstruction of the Boundary between Known and Unknown Space

Riccardo Monica *Member, IEEE,* and Jacopo Aleotti *Senior Member, IEEE*

Abstract—This work presents the first surfel-based method for multi-view 3D reconstruction of the boundary between known and unknown space. The proposed approach integrates multiple views from a moving depth camera and it generates a set of surfels that encloses observed empty space, i.e. it models both the boundary between empty and occupied space, and the boundary between empty and unknown space. One novelty of the method is that it does not require a persistent voxel map of the environment to distinguish between unknown and empty space. The problem is solved thanks to an incremental algorithm that computes the Boolean union of two surfel bounded volumes: the known volume from previous frames and the space observed from the current depth image. A number of strategies were developed to cope with errors in surfel position and orientation. The method, implemented on CPU and GPU, was evaluated on real data acquired in indoor scenarios, and it was compared against state of the art approaches. Results show that the proposed method has a low number of false positive and false negatives, it is faster than a standard volumetric algorithm, it has a lower memory consumption, and it scales better in large environments.

Index Terms—Surfel based mapping, dense multi-view 3D reconstruction, range sensing

1 Introduction

The goal of dense 3D reconstruction using a moving depth camera is to recover the geometrical structure of a scene from multiple views. Many approaches have been proposed to achieve a real-time reconstruction that iteratively merge views into a global representation [1]. In various applications, however, reconstruction of visible surfaces alone is not enough and it is necessary to encode occupied, empty and unknown space. For example, unknown (unseen) regions of a reconstructed 3D model can be visualized to assess completeness and quality, or they can serve as input for a hole filling algorithm. Moreover, in 3D scanning, unseen regions of a 3D model may be used to instruct the user, or an automated system, where new scans should be obtained. In particular, Next Best View algorithms find the optimal sensor pose that maximizes an information gain, which is usually computed as the expected visible amount of unknown space.

By definition, empty space is the space that has been traversed by a sensor viewing ray, and it is therefore known to be empty. Occupied space is the surface where sensor observations occurred. Unknown space is the space that the sensor could not observe either because occluded by the occupied surface, or because the sensor was not oriented towards it. Known space is the union of empty and occupied space. Volumetric data structures, like regular voxel grids, do encode both known and unknown space and, therefore, they are widely adopted by most existing methods for planning robot tasks [2]. In a voxel grid a voxel is either empty, occupied, or unknown, and occupied space is encoded as a thin layer of voxels approximating object surfaces. The main disadvantages of volumetric representations are high storage requirements and long rendering times.

A different approach to represent a 3D scene is using point-based data structures called surfels. Surfels (surface elements) are oriented circular disks, without explicit connectivity, with attributes like center, normal, radius, and color [3]. The surfel-based representation allows a more efficient update of the 3D reconstruction. Indeed, surfels can be efficiently managed through point-based rendering, thanks to GPU acceleration. Moreover, surfels enable modeling of large environments without any loss in terms of resolution. Among surfel-based techniques ElasticFusion [4] is a well known approach for dense visual SLAM (Simultaneous Localization and Mapping) for RGB-D cameras.

A limitation of surfel-based 3D reconstruction methods, like ElasticFusion, is that they only provide a reconstruction of the occupied object surfaces. Therefore, they do not encode incomplete regions of the surfel-based 3D model. In previous work [5], it has been shown that the frontier between unknown and known space can be efficiently approximated by using a surfel cloud. However, in [5] computation of the boundary between unknown and known space required a persistent volumetric voxel-based map, that was provided by KinectFusion [6].

In this work, we propose the first surfel-based incremental approach for 3D reconstruction that approximates the closed surface enclosing the current known space. That is, the method generates a surfel cloud that includes both the boundary between empty and occupied space (i.e. the occupied surface), and the boundary between empty and unknown space. The method does not require a persistent voxel map of the environment.

The goal is challenging for two main reasons. First, multiple views from a moving depth camera must be integrated to produce a surfel cloud in a globally consistent way. As each depth image generates a surfel bounded volume of

• *R. Monica and J. Aleotti are with the Robotics and Intelligent Machines Laboratory (RIMLab), Department of Engineering and Architecture, University of Parma, 43124 Parma (PR), Italy. E-mail:* {riccardo.monica, jacopo.aleotti}@unipr.it

Manuscript received April 19, 2005; revised August 26, 2015.

known space, the problem is solved by an iterative and incremental procedure that computes the Boolean union of two surfel bounded volumes: the known volume from previous frames and the space observed from the current depth image. Being incremental, the algorithm does not require to store all information about each sensor view simultaneously. Second, due to the discrete nature of surfels the solution must be robust to approximation errors that may compromise Boolean operations between surfel bounded volumes. In particular, to cope with errors in the estimation of surfel position and orientation, the proposed method exploits redundancy of paths in ray casting operations, as well as knowledge about the surfel neighborhood.

The approach was compared against a standard volumetric 3D reconstruction algorithm (based on octrees), available within the OctoMap library [7], which was used as ground truth. Results indicate that the proposed method has comparable accuracy, but it is faster, it has a lower memory consumption, and it scales better in large environments. Also, the proposed approach has better accuracy than the state of the art method in [5]. In summary, the main contributions of this work are:

- the first approach for the reconstruction of the boundary between known space and unknown space, encoded using a surfel-based representation;
- an experimental evaluation on real data acquired using a depth camera in indoor environments, including sequences from a publicly available dataset;
- a comparison with state of the art approaches [5] and OctoMap;
- a publicly available open source implementation of the proposed approach written in OpenCL/C++, supporting both CPU and GPU.

This paper is organized as follows. Section 2 reviews related works. The proposed method is described in Section 3. Section 4 presents the experimental evaluation. Finally, Section 5 concludes the paper discussing future extensions.

2 Related Work

In this section, we review related works about Boolean operations on surfel bounded solids. We also review methods for surfel based reconstruction and other applications of surfel representation.

2.1 Boolean operations on surfel bounded solids

The problem of performing Boolean operations on surfel bounded models has been investigated only on single pairs of solids [8], [9], [10], [11]. Adams et al. [8], [9] proposed an approach based on an inside-outside test, which was also adapted to work entirely on the GPU. Farias et al. [10] adopted a more efficient solution, based on Constrained BSP-trees, with no restrictions on the directions of cuts. In [11] a method was proposed to cope with surfel sets of different density that exploited Hierarchical Bounding Volumes. Unlike these methods, in this work we aim to incrementally execute Boolean union operations between multiple surfel bounded solids extracted from sensor measurements. Being incremental, the solution is more challenging than previous works in terms of noise sensitivity, as errors accumulate during the 3D reconstruction.

2.2 Surfel based reconstruction

Several approaches, like ElasticFusion [4], have been investigated for surfel based mapping, however all previous works do not distinguish between unknown and empty space. Park et al. [12], [13] proposed a probabilistic approach for surfel mapping based on dense LIDAR data. In [14] a large scale scene reconstruction method was presented, based on surfels and video sequences, acquired using a hand-held Kinect sensor. Schadler et al. [15] presented an approach for 3D environment mapping and 6D tracking of a mobile robot, using a rotating planar laser scanner. In [16] an approach was introduced for real-time registration of RGB-D images that extracts multi-resolution surfel views and that performs registration by exploiting a variant of the iterative closest point (ICP) algorithm. Another method for visual SLAM, which models measurement uncertainties by using a probabilistic surfel map, was proposed in [17]. Henry et al. [18] proposed a surfel based framework, called RGB-D Mapping, that performs a joint optimization including visual features and shape-based alignment. Carceroni et al. [19] investigated the problem of 3D reconstruction of shape and reflectance from multiple cameras, by exploiting dynamic surfels to take also into account non rigid motions of deformable surfaces. Puri et al. [20] proposed a method, called GravityFusion, that corrects a surfel map with inertial sensor data in the absence of a pose graph. In [21] an approach for high quality surfel rendering was presented by using a fast dynamic up-sampling algorithm suitable for point-based geometry. In [22] a classification-based approach for high quality rendering in large scenes was developed that minimizes the number of rendered points on the GPU. Klaess et al. [23] presented an efficient method for mobile robot navigation that exploits surfels to build a 2D navigation map.

2.3 Applications of surfel based representations

Besides being used for 3D reconstruction, surfels have been successfully adopted in several areas, such as 3D visualization, object detection, pose estimation, and RGB-D segmentation. Following the work of Pfister et al. [3], several works have targeted efficient and realistic surfel visualization. In [24], surfel rendering was implemented in a real-time virtual reality system, using multiple levels of detail. In [25], a surfel-based differentiable renderer was proposed, which uses the elliptical weighted average filter. McElhone et al. [26] proposed a method for simultaneous detection and pose tracking of multi-resolution surfel models in RGB-D data. In [27] a surfel based filter was proposed for color and position enhancement, to facilitate object segmentation tasks.

3 Method

The proposed approach operates on surfels. Each surfel represents a disk, centered in $p_\psi\,(x_\psi, y_\psi, z_\psi)$, with normal $n_\psi\,(n_{x,\psi}, n_{y,\psi}, n_{z,\psi})$ and radius r_ψ. Surfel normals n_ψ are oriented towards the interior of the volume representing the known space. A set of surfels is also called surfel cloud. At each iteration n the proposed algorithm takes as input the surfel cloud Ψ_n, which bounds the current known space

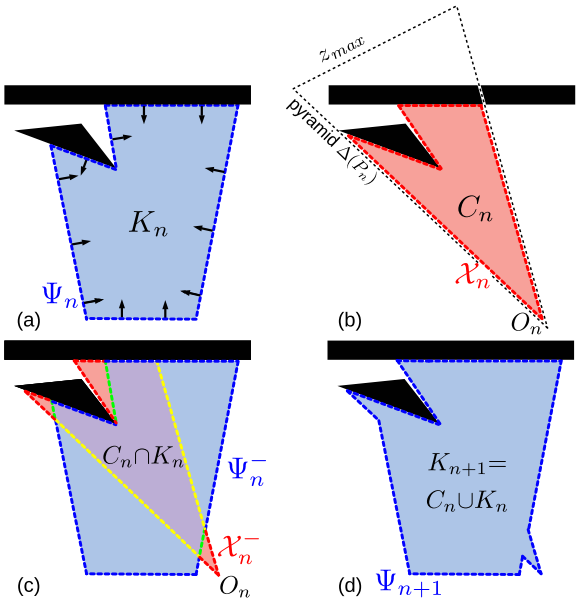

Fig. 1. (a) The surfel cloud Ψ_n (blue) bounding known space K_n (light blue), and some of the inward pointing normals (black arrows). Black filled polygons represent objects in the scene. (b) A new volume C_n (light red) is observed, bounded by surfel cloud $\mathcal{X}_n$ (red). The volume observed by the sensor cannot exceed a rectangular pyramid $\Delta(P_n)$ (bounded by black dashed lines). (c) Red surfels are added to $\mathcal{X}_n^-$, while yellow surfels $\chi \in \mathcal{X}_n$, that are in K_n, are not added to $\mathcal{X}_n^-$. Green surfels $\psi \in \Psi_n$ are contained in C_n, therefore they are removed. Blue surfels are kept in Ψ_n^-. (d) The final surfel set (blue color) Ψ_{n+1}, bounding K_{n+1}, obtained as the union of $\mathcal{X}_n^-$ and Ψ_n^-.

K_n, and the current depth image H_n, in order to compute a new updated surfel cloud Ψ_{n+1}. Surfel cloud Ψ_n is obtained from depth images $H_1 \dots H_{n-1}$ of previous iterations. The standard pinhole model is assumed for the camera. Each depth image $H_n = \{{}^n d_{s,t}\}$ contains, for each pixel (s,t), the distance ${}^n d_{s,t}$ to the observed object, measured along the sensor viewing direction $\hat{z}$. Surfel cloud $\Psi_n = \Sigma_n \cup \Phi_n$ includes both occupied surfels Σ_n, which represent the occupied surface, and frontier surfels Φ_n which separate empty from unknown space. Therefore, Ψ_n represents a closed surface that bounds all the known volume.

Let C_n be the observed volume from the current depth image H_n. Let also $\mathcal{X}_n$ be the surfel set that bounds volume C_n. The goal of each iteration is to produce a new updated surfel cloud Ψ_{n+1}, which bounds $K_{n+1} = \bigcup_1^n C_n$, i.e. the union of all observed volumes. Volume K_{n+1} can also be defined, by induction, as $K_{n+1} = K_n \cup C_n$, with the initial conditions $K_1 = \varnothing$ and $\Psi_1 = \varnothing$ (Fig. 1a). Hence, Ψ_{n+1} can be computed from two surfel bounded solids [8]: K_n, bounded by surfel cloud Ψ_n, and C_n, bounded by surfel cloud $\mathcal{X}_n$. In particular, Ψ_{n+1} is the union of two surfel sets: Ψ_n^-, which contains surfels $\psi \in \Psi_n$ outside C_n, and $\mathcal{X}_n^-$, which contains surfels $\chi \in \mathcal{X}_n$ outside K_n, i.e.

$$\Psi_{n+1} = \underbrace{\{\psi \in \Psi_n \mid p_\psi \notin C_n\}}_{\Psi_n^-} \cup \underbrace{\{\chi \in \mathcal{X}_n \mid p_\chi \notin K_n\}}_{\mathcal{X}_n^-} \quad (1)$$

In summary, to obtain Ψ_{n+1}, four elements are required: surfel cloud Ψ_n (from previous iteration), observed volume C_n (to be computed from H_n), surfel cloud $\mathcal{X}_n$ (to be computed from C_n), and known volume K_n (to be computed from Ψ_n). Computation of Ψ_{n+1} involves both the creation of new surfels to be added to Ψ_n, and the removal of other surfels from Ψ_n.

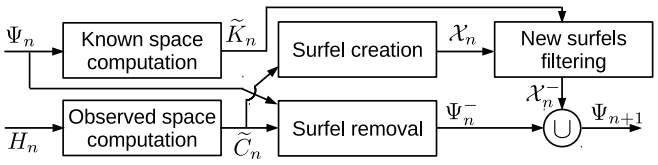

Fig. 2. Flowchart of our approach: Ψ_{n+1} is computed from Ψ_n and H_n.

Ideally, at the beginning of each iteration the surfel set Ψ_n should bound known space K_n. When a new sensor observation is taken from sensor pose P_n, with origin in O_n, a new volume C_n is observed, and surfel cloud $\mathcal{X}_n$ (bounding C_n) should be generated (Fig. 1b). Then, new surfels $\chi \in \mathcal{X}_n$ outside K_n should be created and added to $\mathcal{X}_n^-$ (red surfels in Fig. 1c), while surfels $\chi \in \mathcal{X}_n$ in K_n should not be added to $\mathcal{X}_n^-$ (yellow surfels in Fig. 1c). Similarly, surfels $\psi \in \Psi_n$ which are contained in C_n should be removed (green surfels in Fig. 1c). Finally, surfel cloud Ψ_{n+1} should be computed as the union of $\mathcal{X}_n^-$ and Ψ_n^- (Fig. 1d).

It can be observed that bounding volume K_n may take any shape, as it derives from the union of many sensor observations. Conversely, volume C_n cannot exceed a rectangular pyramid $\Delta(P_n)$ in front of the sensor, with apex in O_n and height equal to the maximum sensor range z_{max} (Fig. 1b). Hence, surfel cloud Ψ_{n+1} can be computed from Ψ_n with changes limited to the pyramid $\Delta(P_n)$, i.e., it is sufficient to compute volumes K_n and C_n only in pyramid $\Delta(P_n)$.

Instead of operating on the ideal volumes K_n and C_n, our approach operates on their voxel-based approximations, defined as $\widetilde{C}_n$ and $\widetilde{K}_n$ respectively. The voxel-based approximations are discarded at the end of each iteration. Hence, the only information stored in memory for the next iteration is surfel cloud Ψ_{n+1}. More specifically, a non-uniform voxel grid (NUVG) is exploited (Section 3.1), which contains pyramid $\Delta(P_n)$.

The general flowchart of the method is shown in Fig. 2. In the *Known space computation* phase, $\widetilde{K}_n$ is computed from Ψ_n (Section 3.2). Computation of $\widetilde{K}_n$ from Ψ_n is the most complex step of our approach, as a voxel-based approximation of a volume must be extracted from a surfel cloud (Ψ_n), which contains discrete samples (surfels) of the surface bounding K_n. In the *Observed space computation* phase, $\widetilde{C}_n$ is computed from H_n (Section 3.3). Then, $\widetilde{C}_n$ is used in the *Surfel removal* phase to compute Ψ_n^- from Ψ_n (Section 3.4). Furthermore, $\widetilde{C}_n$ is also used in the *Surfel creation* phase to compute surfels in $\mathcal{X}_n$ that bound C_n. In the *New surfels filtering* phase only surfels $\chi \in \mathcal{X}_n$, which are not in $\widetilde{K}_n$, are added to $\mathcal{X}_n^-$. The *Surfel creation* and *New surfels filtering* phases are described in Section 3.5. Finally, Ψ_{n+1} is obtained as the union of $\mathcal{X}_n^-$ and Ψ_n^-.

In the following it is assumed that surfel cloud Ψ_n is transformed into sensor coordinates at each new sensor observation using sensor pose P_n, and then transformed

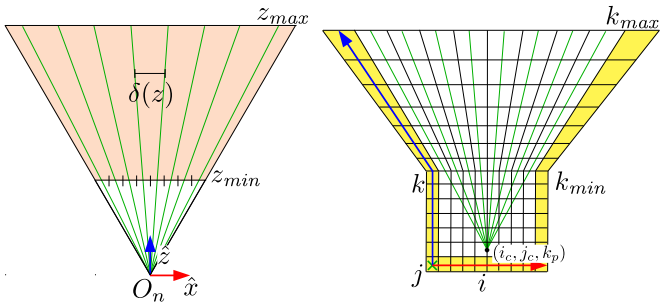

Fig. 3. Left: viewing rays (green) from sensor origin O_n. Occupied measurements occur along the rays (light red volume), between minimum z_{min} and maximum z_{max} range. The distance between two adjacent rays is $\delta(z)$. Right: the NUVG, composed of a pyramidal and a cuboid region. The origin of the parameter space i, j, k is in the lower left corner. Padding voxels are displayed in yellow. Maximum coordinate k is k_{max}, corresponding to maximum sensor range z_{max}. Plane $k = k_{min}$ separates the cuboid region from the pyramidal region of the NUVG.

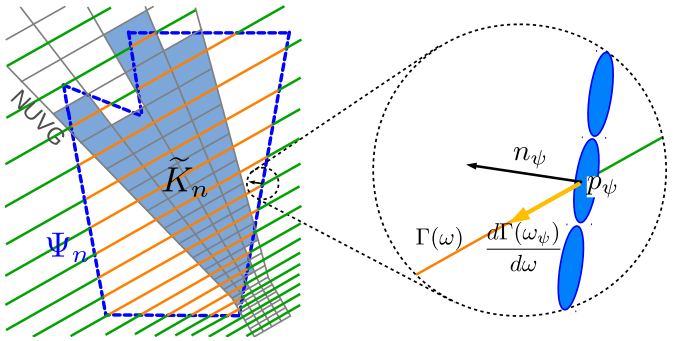

Fig. 4. Computation of the voxel-based representation $\widetilde{K}_n$ (light blue cells) of known space K_n in the NUVG from Ψ_n, by traversing the voxels along a set of paths (left). Orange path segments are in known space, green path segments are in unknown space. A path enters K_n if the dot product between the surfel normal at the intersection point and the tangent to the path is positive (right).

back into world coordinates at the end of the update. Hence, p_ψ and n_ψ indicate the position and the normal of surfel ψ in sensor coordinates. A similar assumption is made for all symbols defined next.

3.1 Non-uniform voxel grid

The intrinsic parameters of the depth camera are its focal length f, the image center (s_c, t_c), and the image resolution s_{max}, t_{max}. Pixel indexes have the following range: $0 \leq s < s_{max}$, $0 \leq t < t_{max}$. Measurements $d_{s,t}$ can occur only in a truncated pyramid (frustum), between minimum (z_{min}) and maximum (z_{max}) sensor range (Fig. 3, left). The space between 0 and $d_{s,t}$ along the viewing ray is considered empty. The local reference frame $(\hat{x}, \hat{y}, \hat{z})$ of the sensor, located at O_n, is oriented so that the $\hat{z}$ axis points outward along the camera optical axis. The distance $\delta(z)$ between adjacent viewing sensor rays, at depth z, is $\delta(z) = \frac{z}{f}$. Hence, when a surface is observed at depth z, points are acquired with resolution $\delta(z)$. As no measurement can occur with $z < z_{min}$, the smallest resolution of observed surfaces is $\delta_{min} = \delta(z_{min})$.

For each iteration of the proposed method, a non-uniform voxel grid (NUVG) is generated as a superset of the volume $\Delta(P_n)$ observable from the current pose of the sensor (Fig. 3, right). The NUVG is deleted at the end of each iteration. The NUVG is defined in a 3D parameter space $[i, j, k]^\intercal \in \mathbf{R}^3$. Each voxel v of the NUVG is indexed by three non-negative integer values in the parameter space. The origin of the NUVG is located at the center of the lower left corner voxel with $(i, j, k) = (0.0, 0.0, 0.0)$. The maximum values of the voxel indexes are $[i_{max}, j_{max}, k_{max}]^\intercal$.

In particular, the NUVG is composed of two regions: a cuboid between 0 and z_{min} and a truncated pyramid between z_{min} and z_{max}. The truncated pyramid region contains frustum-oriented voxels that become larger as the distance from the sensor increases. Thus, far from the sensor the density of voxels is reduced. Frustum-oriented voxels make it is easier to determine $\widetilde{C}_n$ (Section 3.3). The cuboid region close to the sensor contains uniform cubic voxels with side δ_{min}, otherwise there would be too many small frustum-oriented voxels to be considered. Each depth image pixel can be associated to a sequence of adjacent non-uniform voxels along parameter value k in the truncated pyramid region. The range of voxel indexes for parameters i and j, in both regions, is equal to the range of the sensor pixels, plus an extra border (padding) i_p and j_p on each side. Padding prevents border effects in the *Surfel creation* phase (Section 3.5). Maximum indexes i_{max} and j_{max}, and center (i_c, j_c) are:

$$\begin{gathered} i_{max} = s_{max} + 2\,i_p - 1, \quad j_{max} = t_{max} + 2\,j_p - 1 \\ (i_c, j_c) = (s_c + i_p, t_c + j_p) \end{gathered} \tag{2}$$

Even if the cuboid region is over dimensioned with respect to the actual volume observable by the sensor $\Delta(P_n)$, the overhead caused by the extra space is usually small, as for most sensors $z_{min} \ll z_{max}$. A padding k_p is also added behind the sensor, so that the sensor origin O_n in the parameter space is located at $[i_c, j_c, k_p]^\intercal$.

The nonlinear piecewise function $F : \mathbf{R}^3 \rightarrow \mathbf{R}^3$, which maps points in sensor coordinates $[x, y, z]^\intercal$ to points in the parametric coordinates of the NUVG $[i, j, k]^\intercal = F(x, y, z)$, is given by:

$$F(x,y,z) = \begin{cases} \begin{bmatrix} x/\delta_{min} \\ y/\delta_{min} \\ z/\delta_{min} \end{bmatrix} + \begin{bmatrix} i_c \\ j_c \\ k_p \end{bmatrix} & \text{if } z < z_{min} \\ \begin{bmatrix} \frac{x\,f}{z} + i_c \\ \frac{y\,f}{z} + j_c \\ f\ln(z) - f\ln(z_{min}) + k_{min} \end{bmatrix} & \text{if } z_{min} \leq z \end{cases} \tag{3}$$

Further details about function F and its inverse $F^{-1}(i, j, k) = [x, y, z]^\intercal$ are described in Section A of the supplemental material.

3.2 Known space computation

In the *Known space computation* phase a voxel-based representation of the ideal current known space K_n is computed from surfel cloud Ψ_n. The voxel based representation is named *Known space field* ($\widetilde{K}_n$). Since the NUVG contains $\Delta(P_n)$, in order to update the known space it is sufficient to compute $\widetilde{K}_n$ only inside the NUVG. Hence, the proposed algorithm sets the *Known space field* $\widetilde{K}_n(v)$, of each voxel

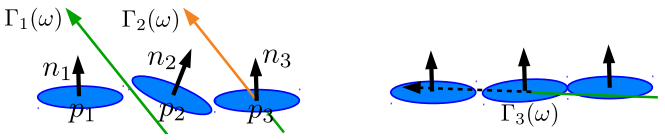

Fig. 5. A surfel configuration where even a small error in the normal of the surfel at p_2 may cause path $\Gamma_1(\omega)$ to pass through gaps between surfels (left). A surfel configuration where it is not possible to accurately determine whether path $\Gamma_3(\omega)$, which is nearly tangent to the surface, enters or exits the surface (right).

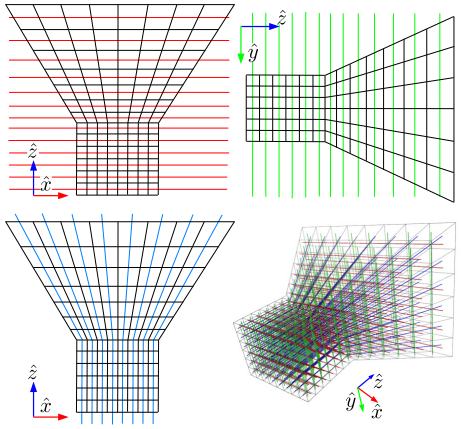

Fig. 6. The family of paths (5) in red (top left), (6) in green (top right), and (7) in blue (bottom left). A complete 3D view of all paths (bottom right). The NUVG is displayed in black.

$v(i,j,k)$ inside the NUVG, equal to 1 if the voxel center v is inside known space, and 0 otherwise. In particular, as surfel cloud Ψ_n approximates a closed surface of arbitrary shape, $\widetilde{K}_n$ can be determined by traversing the voxels inside the NUVG along different paths (Fig. 4, left), computing the intersections points of the paths with Ψ_n, marking intersections as entering or exiting Ψ_n, splitting each path into segments at the intersection points, and classifying each segment as being inside or outside the known volume. A path $\Gamma(\omega) : \mathbf{R} \rightarrow \mathbf{R}^3$, parameterized by ω, enters Ψ_n intersecting surfel ψ in ω_ψ, if the inner product between the tangent vector to $\Gamma(\omega_\psi)$ and the surfel normal is greater than a threshold th_d (Fig. 4, right), i.e.

$$\langle n_\psi, \frac{d\,\Gamma(\omega_\psi)}{d\omega} \rangle > th_d \tag{4}$$

A special condition occurs for the first segment entering the NUVG. In this case the segment is classified considering the last intersecting surfel outside the NUVG.

The approach outlined above to compute $\widetilde{K}_n$ is, however, prone to errors caused by the discrete nature of surfels. Small errors in surfel position and normal vectors can indeed leave gaps between close surfels. Therefore, a path may miss the intersection with a surfel (Fig. 5, left). Moreover, when a path is nearly tangent to a surfel, it may be difficult to determine if the path is entering or exiting the surface (Fig. 5, right). To cope with these issues the method exploits redundancy of paths and it considers not only the surfel itself, but also surfels in its neighborhood.

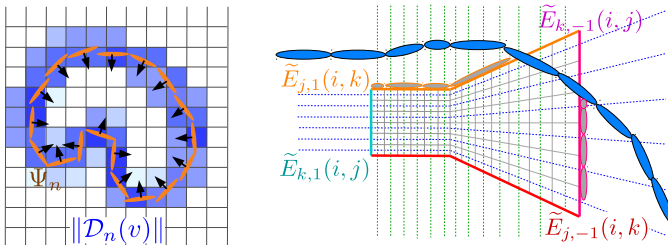

Fig. 7. Left: example of $\|\mathcal{D}(v)\|$ in each voxel with blue shading. Right: a section of the NUVG. Four of the six 2D grids in the *known space hull* are visible (orange, purple, red and cyan lines). Projections occur along the paths of families (6) and (7) (green and blue dashed lines). Projected surfels are displayed as gray ellipses with a colored border.

Redundancy of paths is achieved by traversing each voxel v multiple times along different paths. In particular, three families of paths are created, and each family contains paths in forward and backward direction. The three families of paths are defined as $\Gamma^i_{j,k,\mathcal{I}}$, $\Gamma^j_{i,k,\mathcal{I}}$, $\Gamma^k_{i,j,\mathcal{I}}$, where $\mathcal{I} \in \{-1, 1\}$ specifies the direction (forward and backward), i.e.:

$$\forall j,k,\mathcal{I} \quad \Gamma^i_{j,k,\mathcal{I}}(\omega) = F^{-1}(\mathcal{I}\omega, j, k) \tag{5}$$

$$\forall i,k,\mathcal{I} \quad \Gamma^j_{i,k,\mathcal{I}}(\omega) = F^{-1}(i, \mathcal{I}\omega, k) \tag{6}$$

$$\forall i,j,\mathcal{I} \quad \Gamma^k_{i,j,\mathcal{I}}(\omega) = F^{-1}(i, j, \mathcal{I}\omega) \tag{7}$$

Family of paths (5) and (6) generate paths parallel to the $\hat{x}$ and $\hat{y}$ axes respectively. Family of paths (7) generates piecewise linear paths, parallel in the cuboid region of the NUVG, and radial in the pyramidal region (Fig. 6). Any path in (5)-(7) passes through voxel centers of the NUVG for integer values of ω. Each voxel v is, therefore, traversed by six paths in total, i.e., one path for each of the three families, in both directions.

A detailed description of the required steps to determine $\widetilde{K}_n$ is reported in the following Sections. In particular, in order to determine whether a path is intersecting surfel cloud Ψ_n an approach robust to noise in surfel pose is adopted that exploits the surfel neighborhood. An *inner product sign field* $\mathcal{D}_n \in \mathbf{R}^3$ data structure, defined for each voxel v, is used to this purpose (Section 3.2.1). Moreover, a *known state hull* ($\widetilde{E}_n$) is computed, which is a data structure used to find the last intersected surfel outside the NUVG for each path (Section 3.2.2). Finally, *known space field* $\widetilde{K}_n$ is computed (Section 3.2.3) based on $\mathcal{D}(v)$ and $\widetilde{E}_n$. Even if $\mathcal{D}_n$ and $\widetilde{E}_n$ are computed at each iteration, the subscript n is omitted in the following.

3.2.1 *Inner product sign field*

For each voxel v in the NUVG, an *inner product sign field* $\mathcal{D}(v) \in \mathbf{R}^3$ is computed, which is the weighted sum of the *inner product sign* $g_\psi \in \mathbf{R}^3$ of the surfels in a neighborhood of v (Fig. 7, left). The *inner product sign* g_ψ of a surfel is a 3D vector, with one component for each path family (5)-(7). Each component of g_ψ is equal to 1 if the forward path enters Ψ_n at ψ, and -1 if the forward path exits Ψ_n. The neighborhood of a voxel is defined as an ellipsoid, with two equal semi-axes orthogonal to n_ψ, of length $2r_\psi$. The length of the third semi-axis, along n_ψ, is set so that the ellipsoid includes at least a voxel.

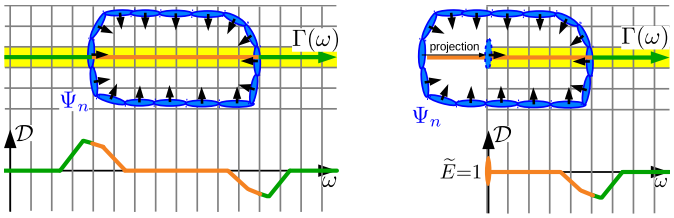

Fig. 8. Left: a set Ψ_n of surfels forming a closed boundary (in blue) in the NUVG (gray grid) traversed by path $\Gamma(\omega)$, and a scalar example of *inner product sign field* $\mathcal{D}$, computed for the voxels highlighted in yellow, along the path. A path segment (orange) in known space is also shown, between the local maximum and the local minimum values of $\mathcal{D}$. Right: a set of surfels partially outside the NUVG. The *inner product sign field* $\mathcal{D}$ is computed only in the NUVG. Then, the last intersected surfel is projected onto the surface of the NUVG to obtain the *known state hull* $\widetilde{E}$, so that it can be determined that the path is in known space when it enters the NUVG.

Moreover, an *invalid inner product field* $\not\mathcal{D}(v) = [\not\mathcal{D}_i(v), \not\mathcal{D}_i(v), \not\mathcal{D}_k(v)]$ is defined. Each component $\not\mathcal{D}_i$, $\not\mathcal{D}_j$, $\not\mathcal{D}_k$ is equal to 1 if there is at least one surfel in the neighborhood of v, for which it is not possible to determine if the corresponding path family enters or exits Ψ_n. Further details on the computation of $\mathcal{D}$ and $\not\mathcal{D}$ are provided in Section B of the supplemental material.

3.2.2 Known state hull

A path segment belongs to known space K_n if (4) is true at the beginning of the segment. In the special case when a path segment enters the NUVG, inequality (4) must be evaluated for the last intersected surfel before entering the NUVG. To this purpose, an efficient approach is adopted that uses a *known state hull* $\widetilde{E}$ data structure, and that projects surfels outside the NUVG on the surface of the NUVG along the paths. In particular, the *known state hull* $\widetilde{E}$ is a set of six 2D grid $\widetilde{E}_{\tau,\mathcal{I}}(\alpha,\beta)$ (Fig. 7, right). There exists a grid for each path family (5)-(7) and for each direction $\mathcal{I}$. Value $\tau \in \{i,j,k\}$ indicates the path family $\Gamma^{\tau}_{\alpha,\beta,\mathcal{I}}(\omega)$, with $\mathcal{I}$ indicating the direction of the path, and α,β being cell indices. For example, 2D grid $\widetilde{E}_{i,-1}(j,k)$ corresponds to path family $\Gamma^{i}_{j,k,-1}(\omega)$ traversed along the backward direction. Each cell (α,β) is equal to 1, if the path is in known space when entering the NUVG, and 0 otherwise. Further information about $\widetilde{E}$ is provided in Section C in the supplemental material.

3.2.3 Known space field computation

The *known space field* $\widetilde{K}_n(v)$, which is 1 if voxel $v(i,j,k)$ is in known space, is computed from the *inner product sign field* $\mathcal{D}(v)$, the *invalid inner product field* $\not\mathcal{D}(v)$ and *known state hull* $\widetilde{E}(v)$ at the current iteration n. In particular, an *unfiltered known space field* $\widetilde{K}'_n(v)$ is first computed, and then $\widetilde{K}_n(v)$ is obtained from $\widetilde{K}'_n(v)$. The *unfiltered known space field* $\widetilde{K}'_n(v)$ is initialized to 0. The *unfiltered known space field* $\widetilde{K}'_n$ is determined by traversing each path $\Gamma^{\tau}_{\alpha,\beta,\mathcal{I}}(\omega)$ defined in (5)-(7). The path is initially in known or unknown space according to *known state hull* $\widetilde{E}$. When a path traverses a voxel v where $\mathcal{D}(v)$ has a local maximum of the component corresponding to the path family (or minimum, if the path is in backward direction), it is considered as entering the known space. Conversely, if the component of $\mathcal{D}(v)$ has local minimum (or maximum, if the path is in backward direction), the path is considered as exiting the known space (Fig. 8). When a path traverses a voxel v where $\not\mathcal{D}(v)=1$, it is reset to unknown state.

Algorithm 1 Unfiltered known space field computation

Input: $\mathcal{D}(i,j,k)$: inner product sign field
Input: $\not\mathcal{D}(i,j,k)$: invalid inner product field
Input: $\widetilde{E}$: known state hull
Output: $\widetilde{K}'_n(i,j,k)$: unfiltered known space field

1: **for each** $\Gamma^{\tau}_{\alpha,\beta,\mathcal{I}}(\omega)$ **do**
2: $e \leftarrow \widetilde{E}_{\tau,\mathcal{I}}(\alpha,\beta)$
3: **for each** v **along** $\Gamma^{\tau}_{\alpha,\beta,\mathcal{I}}(\omega)$ **do**
4: **if** $\not\mathcal{D}_{\tau}(v)$ **then**
5: $e \leftarrow 0$
6: **else**
7: **if** IsPeak$(-\mathcal{I}\mathcal{D}_{\tau}(v))$ **then**
8: $e \leftarrow 0$
9: **end if**
10: $\widetilde{K}'_n(v) \leftarrow \widetilde{K}'_n(v) \vee e$
11: **if** IsPeak$(\mathcal{I}\mathcal{D}_{\tau}(v))$ **then**
12: $e \leftarrow 1$
13: **end if**
14: **end if**
15: **end for**
16: **end for**

In particular, each path is traversed (line 1 in Algorithm 1), and at each step a knowledge state variable $e \in \{0,1\}$ is set to 1 if the path is currently in known space and 0 otherwise. The knowledge state is initialized to the value of the *known state hull* $\widetilde{E}_{\tau,\mathcal{I}}(\alpha,\beta)$ (line 2). Paths are traversed one voxel v at a time, using integer values of ω (line 3). As long as knowledge state $e=1$, $\widetilde{K}'_n(v)$ is set to 1 (line 10), that is, if any path is in known state in v, v is considered in known space. The knowledge state e is set to 0 whenever $\not\mathcal{D}_{\tau}(v)=1$ (lines 4-5), or when a local minimum of the *inner product sign field* $\mathcal{D}_{\tau}(v)$ is found (lines 7-9). Conversely, e is set to 1 whenever a local maximum of $\mathcal{D}_{\tau}(v)$ is found, where $\tau \in \{i,j,k\}$ indicates a component of vector $\mathcal{D}(v)$ (lines 11-13). Function IsPeak$(A(v))$ is used to detect local maxima. The test for a local maximum (line 11) is performed after updating the *unfiltered known space field* (line 10), so that the voxel containing the maximum is conservatively not set in known space.

After computation, a median filter is applied to the *unfiltered known space field* $\widetilde{K}'_n(v)$, i.e., given the 26-neighborhood $N_{26}(v)$ of voxel v, the *known space field* is computed as:

$$\widetilde{K}_n(v) = \begin{cases} 1 & \text{if } \widetilde{K}'_n(v) + \sum_{v' \in N_{26}(v)} \widetilde{K}'_n(v') \geq 14 \\ 0 & \text{otherwise} \end{cases} \quad (8)$$

3.3 Observed space field computation

When a new depth image $H_n = \{{}^n d_{\underline{s},t}\}$ is observed at frame n, the current *observed space field* $\widetilde{C}_n(v)$ is computed, so that $\widetilde{C}_n(v)$ is equal to 1 if voxel $v(i,j,k)$ is traversed by a viewing ray cast from O_n, and 0 otherwise.

In the cuboid region of the NUVG ($k < k_{min}$) a viewing ray is cast through each pixel (s,t) of H_n that contains a

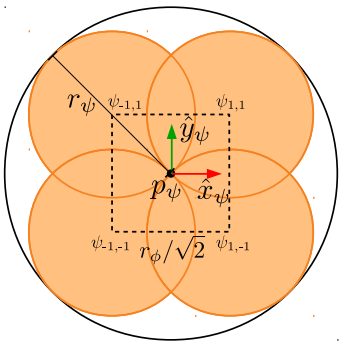

Fig. 9. A surfel ψ (black circle) with radius r_ψ split into four smaller surfels $\psi_{1,1}$, $\psi_{1,-1}$, $\psi_{-1,1}$ and $\psi_{-1,-1}$ (orange circles) in the surfel removal phase. The side of the square (dashed line) which connects the four surfel centers is $r_\psi/\sqrt{2}$.

valid depth. Viewing rays are sampled at integer values of k, which correspond to uniform steps of δ_{min} along the $\hat{z}$ axis. *Observed space field* $\widetilde{C}_n(v)$ is set to 1 in voxels that contain at least one sample along the viewing ray, and to 0 otherwise. More specifically, for each voxel $v(i,j,k)$ where $k < k_{min}$, $\widetilde{C}_n(v) = 1$ if there exist a pixel (s,t) with a valid depth measurement ${}^n d_{s,t}$, so that in parameter space

$$v = \text{Round}\left(F(x,y,z)\right), \tag{9}$$

$$z = (k - k_p)\,\delta_{min},\ x = \frac{z}{f}(s - s_c),\ y = \frac{z}{f}(t - t_c) \tag{10}$$

In the pyramidal region of the NUVG there is a one-to-one correspondence between a viewing ray and a sequence of adjacent voxels along parameter value k. Therefore, for $k \geq k_{min}$, $\widetilde{C}_n(v)$ can be more efficiently computed as:

$$\widetilde{C}_n(v) = \begin{cases} 1 & \text{if } F_z^{-1} < {}^n d_{s,t},\ s = i - i_p,\ t = j - j_p \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

where $F_z^{-1}(i,j,k)$ is the z component of vector $F^{-1}(i,j,k)$.

3.4 Surfel removal

To compute Ψ_n^- from Ψ_n (1), surfels $\psi \in \Psi_n$ must be removed if they are inside C_n. Therefore, in the surfel removal phase, surfels are deleted if they are inside a voxel v, where $\widetilde{C}_n(v) = 1$. Large surfels that span across multiple voxels are first split into smaller surfels having a diameter comparable to the voxel size. More specifically, a surfel ψ is split if the radius is more than twice the diagonal of the voxel, i.e. $r_\psi \geq 2\sqrt{3}\max\left(\delta(z_\psi), \delta_{min}\right)$, where z_ψ is the distance of the voxel to the sensor. Let $(\hat{x}_\psi, \hat{y}_\psi, \hat{z}_\psi)$ be the local reference frame of the surfel, with $\hat{z}_\psi = n_\psi$. Four surfels $\psi_{a,b}$, $(a,b) \in \{-1,1\}^2$ are created with radius $r_\psi/2$ (Fig. 9) at position:

$$p_{\psi_{a,b}} = p_\psi + \frac{r_\psi}{2\sqrt{2}}\,a\,\hat{x}_\psi + \frac{r_\psi}{2\sqrt{2}}\,b\,\hat{y}_\psi \tag{12}$$

At the end of the surfel removal phase, Ψ_n^- contains all surfels $\psi \in \Psi_n$ where $\widetilde{C}_n\left(F(p_\psi)\right) = 0$.

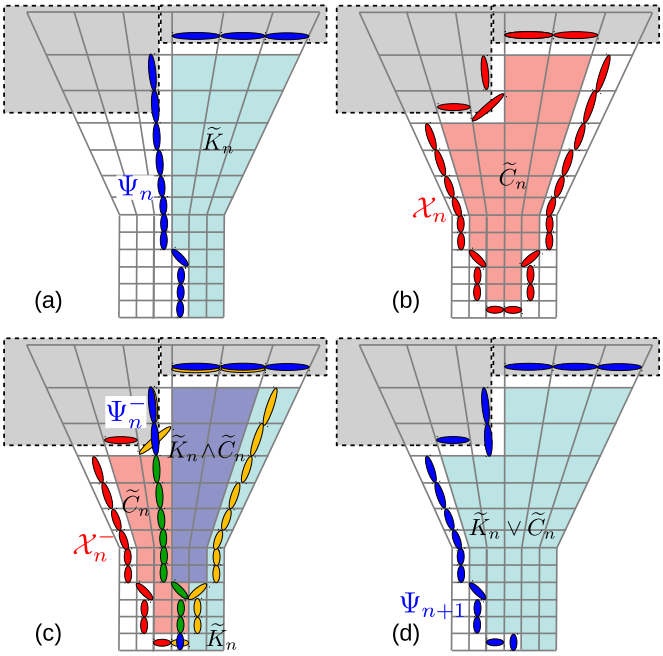

Fig. 10. Surfel removal and creation example. The light gray rectangles with a dashed contour are objects in the environment. (a) Voxels where $\widetilde{K}_n = 1$ are displayed with a light blue background, bounded by blue surfels Ψ_n. (b) Voxels where $\widetilde{C}_n = 1$ are displayed with a light red background, bounded by red surfels $\mathcal{X}_n$. (c) In purple voxels, both $\widetilde{K}_n = 1$ and $\widetilde{C}_n = 1$. Green surfels are those surfels $\psi \in \Psi_n$ which are in voxels where $\widetilde{C}_n = 1$, so they are not added to Ψ_n^-. Yellow surfels are those surfels $\chi \in \mathcal{X}_n$ where $\widetilde{K}_n = 1$, or those that overlap an existing (blue) surfel, so they are not added to $\mathcal{X}_n^-$. (d) Surfels in $\Psi_{n+1} = \mathcal{X}_n^- \cup \Psi_n^-$ bound voxels where $\widetilde{K}_n \vee \widetilde{C}_n = 1$.

3.5 Surfel creation and filtering

3.5.1 *Surfel creation*

In the *Surfel creation* phase, surfel cloud $\mathcal{X}_n$ is created, which contains surfels χ that bound $\widetilde{C}_n$. Surfels are created at the center of voxels immediately outside the observed space, i.e. in voxels v where $\widetilde{C}_n(v) = 0$, and in the neighborhood of a voxel with $\widetilde{C}_n(v) = 1$:

$$\begin{cases} \widetilde{C}_n(v) = 0, \\ \exists\, v' \in N_6(v) \quad | \quad \widetilde{C}_n(v') = 1 \end{cases} \tag{13}$$

where $N_6(v)$ is the 6-neighborhood of v. Position p_χ, normal n_χ and radius r_χ of a new surfel χ are set as follows (normalization omitted):

$$\begin{aligned} p_\chi &= F^{-1}(v) \\ n_\chi &= \sum_{v' \in N_{26}(v)} \frac{F^{-1}(v') - p_\chi}{\|F^{-1}(v') - p_\chi\|}\,\widetilde{C}_n(v') \\ r_\chi &= \frac{\sqrt{3}}{2}\max\left(\delta(z_\chi), \delta_{min}\right) \end{aligned} \tag{14}$$

Radius r_χ is set to half the approximate side of the voxel at depth z_χ, so that the surfel spans the whole voxel. Normal n_χ is estimated as the average of the vectors pointing from voxel v to all voxels v' in the 26-neighborhood $N_{26}(v)$ of v that are inside the observed space $\widetilde{C}_n$. Therefore, the surfel is oriented towards observed (known) space. In degenerate

cases, normal n_χ is null due to the symmetric shape of $\widetilde{C}_n$ in the neighborhood of v. These cases occur when $\widetilde{C}_n(v) = 0$, and $\widetilde{C}_n(v') = 1$ for one or more pairs of opposite neighbor voxels v'. In these cases the surfel is not created as the voxel v is between two regions of observed space, and not between observed and unobserved space.

3.5.2 New surfel filtering

In the *New surfel filtering* phase, $\mathcal{X}_n^-$ is computed, which contains surfels in $\mathcal{X}_n$ outside K_n to comply with (1). To avoid generating multiple surfels inside the same voxel v, surfels are not added to $\mathcal{X}_n^-$ if another surfel $\psi \in \Psi_n$ is already present in the same voxel. Hence, $\mathcal{X}_n^-$ is defined as follows:

$$\mathcal{X}_n^- = \left\{ \chi \in \mathcal{X}_n \;\middle|\; \begin{array}{l} \widetilde{K}_n\left(F\left(p_\chi\right)\right) = 0, \\ \nexists\, \psi \in \Psi_n^- \mid \mathrm{Round}\left(F\left(p_\psi\right)\right) = v \end{array} \right\} \quad (15)$$

An example of the surfel removal and creation procedure is shown in Fig. 10.

3.5.3 Surfel labeling

Surfel cloud Ψ_{n+1} includes both the subset of occupied surfels (Σ_{n+1}), as well as the subset of frontier surfels (Φ_{n+1}), i.e., $\Psi_{n+1} = \Sigma_{n+1} \cup \Phi_{n+1}$. The ability to reconstruct from multiple views both surfels that separate empty from occupied space, as well as surfels that separate empty from unknown space, is the main novelty of the proposed approach. The goal of the *Surfel labeling* phase is to label each surfel $\psi \in \Psi_{n+1}$ either as occupied, i.e. $\psi \in \Sigma_{n+1}$, or as a frontier surfel, i.e. $\psi \in \Phi_{n+1}$.

At each iteration n, all new surfels are first initialized as frontier surfels in Φ_n. Those surfels ψ that come from a real sensor observation are then labeled as occupied $\psi \in \Sigma_{n+1}$. Surfels that are created due to real sensor observations are those for which a viewing ray hits a real surface. Sensor observations may occur only in the pyramidal region of the NUVG, as the cuboid region of the NUVG is below the minimum range of the sensor. Hence, at current observation n, a surfel ψ in voxel $v = (i, j, k) = F\left(p_\psi\right)$ is labeled as occupied if in voxel $v'(i, j, k-1)$ the viewing ray hits a surface, i.e. if the following conditions hold:

$$k > k_{min} \quad \wedge \quad \widetilde{C}_n(i, j, k-1) = 1 \quad (16)$$

Once a surfel is labeled as occupied, it remains in this state indefinitely.

4 Experimental Evaluation

4.1 Experimental setup

An OpenCL/C++ implementation of the proposed approach was developed, that can run both on CPU and GPU. Experiments were performed on an Intel Core i7-7700 CPU (3.60 GHz, 16 GB RAM), with an NVidia GeForce GTX 1070 GPU (8 GB RAM). Tests performed on CPU are based on the Intel SDK for OpenCL Applications (version 18.1), using device fission to ensure that only a single thread is used. Tests performed on GPU use NVidia CUDA 10.0 OpenCL runtime. The source code is available as open-source at http://rimlab.ce.unipr.it/Software.html (under `surfels_unknown_space`).

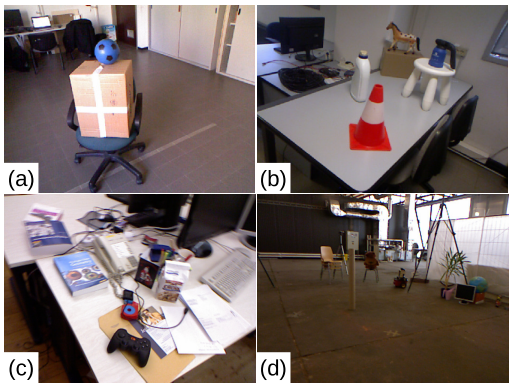

Fig. 11. RGB image of the first Scenario (a), RGB image of the second Scenario (b). RGB images of the TUM dataset [28]: office environment used in Scenario 3 and 4 (c) and large-scale environment used in Scenarios 5 and 6 (d).

TABLE 1
Parameters used in the experimental evaluation

Parameter	Value	Description
th_d	$\cos(80°)$	Confidence threshold in (4)
i_p, j_p, k_p	5, 5, 2	Padding (Section 3.1)
s_{max}	160 pixel	Image width
t_{max}	120 pixel	Image height

4.1.1 Dataset

The proposed method was evaluated in six indoor RGB-D image sequences, acquired using a Kinect v1 sensor. The six Scenarios were selected to show that the approach can work both in small and large scale environments, and that it provides consistent results on different scanning strategies. The first two image sequences were acquired in a laboratory building. In particular, Scenario 1 (2000 frames) reconstructed a box and a ball on top of an office chair in the middle of an (almost) empty room (Fig. 11a). Scenario 2 (1450 frames) contains several objects located on top of a table (Fig. 11b). In the first two scenarios, the camera was oriented towards the objects and it was moved along an approximately circular path. The width of the observed space in both Scenarios 1 and 2 is about 4 m. Scenarios 3, 4, 5 and 6 were selected from the TUM RGB-D SLAM dataset [28], to also show the viability of the approach on publicly available datasets. In particular, we selected image sequences "fr1/360" (Scenario 3, 742 frames), "fr1/room" (Scenario 4, 1344 frames), "fr2/360_hemisphere" (Scenario 5, 2653 frames) and "fr2/pioneer_slam2" (Scenario 6, 1794 frames). In both Scenarios 3 and 4, the sensor performed a 360-degrees scan around an office environment (Fig. 11c and 11d), the width of the observed space is about 7 m. In Scenario 5, the sensor performed a similar 360-degrees scan in a large-scale environment (Fig. 11e), the observed space is about 11×13 m. In Scenario 6 the sensor was mounted on a mobile robot that performed a navigation task in the same large-scale environment of Scenario 5.

Our approach requires knowledge of the camera pose P_n at each sensor frame n. In Scenarios 1 and 2 we used the camera pose provided by the sensor egomotion tracking sys-

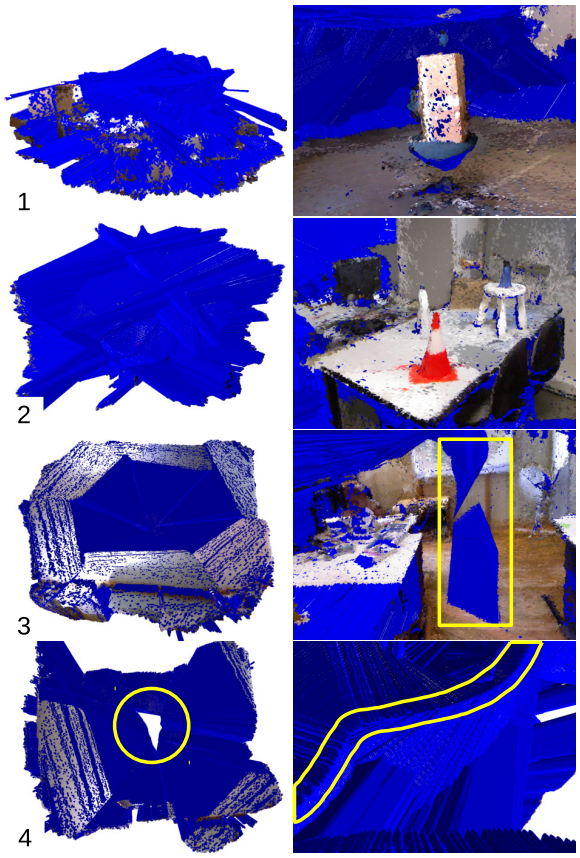

Fig. 12. Surfel-based 3D reconstruction of Scenarios 1, 2, 3 and 4 (top to bottom). Occupied surfels $\sigma \in \Sigma_n$ are displayed with their own RGB color. Unknown surfels $\phi \in \Phi_n$ are displayed in blue. Left: overall view of the surfel-based reconstruction approximating a closed surface. Right: a view of the reconstruction from the inside of the surface in Scenarios 1, 2, and 3, and from the unknown region in the middle of the scene in Scenario 4. The yellow rectangle highlights the region of unknown space in Scenario 3 (third row, right) where the sensor was located. The yellow circle (fourth row, left) highlights the unknown region of space in Scenario 4, and the yellow contour highlights the ridge (fourth row, right).

tem of ElasticFusion [4], which was executed alongside the proposed method. In Scenarios 3, 4, 5 and 6 the camera pose, acquired from a high-precision tracking system, was already available in the TUM dataset. Depth images acquired by the Kinect sensor (640×480) were downsampled by a factor of 4 both vertically and horizontally, for performance reasons, using the nearest-neighbor algorithm. Therefore, our approach was evaluated at 160×120 image resolution. Sensor focal length, which is about 525 pixel, was divided by 4 accordingly. Minimum sensor range z_{min} was set to 0.5, hence, according to Section 3.1, minimum resolution was $\delta_{min} = \frac{z_{min}}{f} \approx 3.8$ mm. Maximum range z_{max} was set to 3 m in Scenarios 1-4, and to 8 m in large-scale Scenarios 5 and 6. Sensor measurements beyond z_{max} were discarded. Other parameters used in the experimental evaluation are reported in Table 1.

4.1.2 Ground truth generation

We created a ground truth dataset using a volumetric 3D representation based on OctoMap [7]. OctoMap implements an octree $\widetilde{M}_n(v)$ for 3D occupancy mapping, and it encodes 3D data into ternary occupancy values, where each

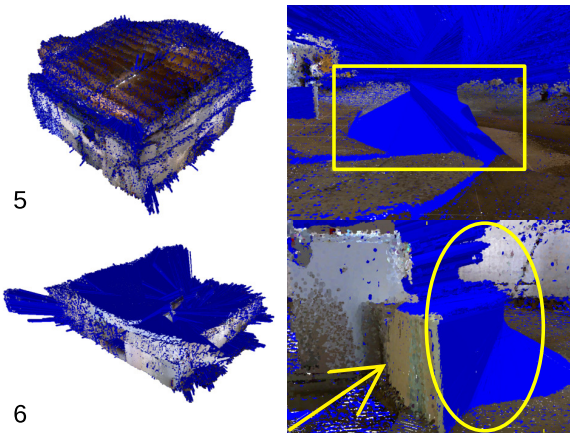

Fig. 13. Surfel-based 3D reconstruction of large-scale Scenarios 5 and 6 (top to bottom). Left: overall view of the surfel-based reconstruction approximating a closed surface. Right: a view of the reconstruction from the inside of the surface. The yellow rectangle highlights the region of unknown space in Scenario 5 (top row, right). The yellow ellipse highlights unknown volume (in Scenario 6) behind an obstacle, that was observed only in the direction of the yellow arrow (bottom row, right).

voxel v centered at position p_v is either *occupied*, *empty* or *unknown*. The Octomap voxel resolution was set to δ_{min}, i.e., the minimum resolution of the proposed approach. The standard *insertPointCloud* method was used to incrementally insert in OctoMap each depth image H_n observed at sensor pose P_n. The *insertPointCloud* method performs ray casting and sets to *empty* all voxels traversed by each ray, except for the last voxel which is set to *occupied*.

A ground truth surfel cloud $\mathcal{M}_n$ was then generated from the OctoMap octree, by exploiting a surfel creation algorithm similar to the one described in Section 3.5.1. That is, for each unknown voxel v that satisfies

$$\begin{cases} \widetilde{M}_n(v) = unknown, \\ \exists\, v' \in N_6(v) \quad | \quad \widetilde{M}_n(v') \neq unknown \end{cases} \tag{17}$$

i.e., which is adjacent to either an empty or an occupied voxel, a surfel $\mu \in \mathcal{M}_n$ is created with position p_μ, normal n_μ and radius r_μ as follows (normalization of the normal omitted):

$$\begin{aligned} p_\mu &= p_v \\ n_\mu &= \sum_{v' \in N_{26}(v)} \frac{p_{v'} - p_\mu}{\|p_{v'} - p_\mu\|} K_{\widetilde{M}}(v') \\ r_\mu &= \frac{\sqrt{3}}{2}\delta_{min} \end{aligned} \tag{18}$$

where $K_{\widetilde{M}}(v') = 1$ if $\widetilde{M}_n(v') \neq unknown$, and 0 otherwise.

4.2 Results

4.2.1 Accuracy

The results of the surfel-based reconstruction in the six Scenarios are shown in Figs. 12 and 13. As expected, in each experiment the generated surfel cloud approximates a closed surface enclosing the known volume. In Scenarios 1 and 2, the depth camera was moved along a circular path, oriented towards the objects in the middle of the scene. Perimeter walls of the scene were not observed for the most

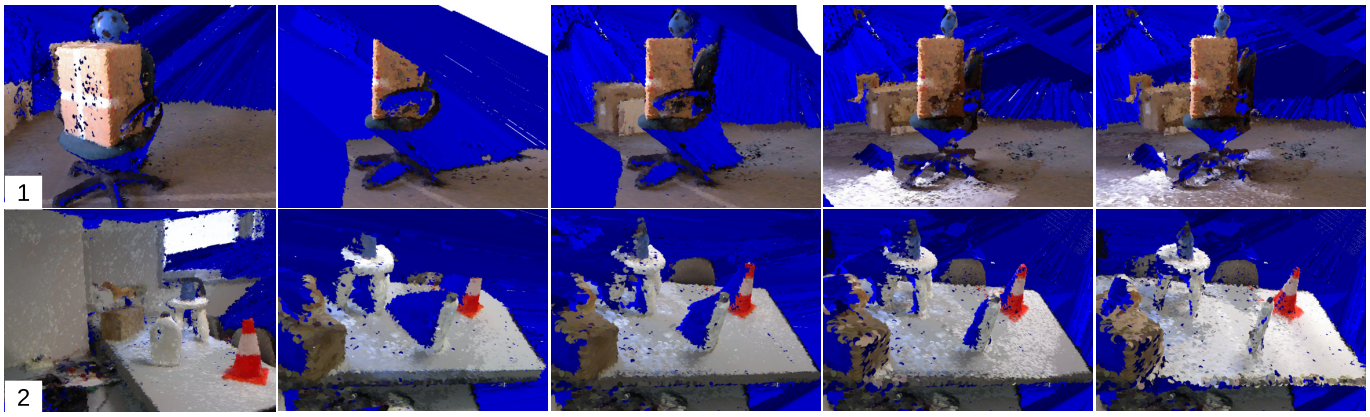

Fig. 14. The evolution of the surfel-based 3D reconstruction in Scenarios 1 (top) and 2 (bottom). Unknown surfels are displayed in blue.

TABLE 2
Reconstruction accuracy

Scen.	Proposed approach			Monica et al. [5]		
	FP	$\lvert\Psi_n\rvert$	FDR	FP	$\lvert\Psi^K\rvert$	FDR
1	8 860	2 377k	0.37%	89 409	7 974k	1.11%
2	2 312	1 700k	0.14%	65 613	5 465k	1.20%
3	2 197	2 033k	0.11%	1 722	20 938k	8.22%
4	603	2 901k	0.21%	1 281k	23 330k	5.49%
Scen.	FN	$\lvert\mathcal{M}_n\rvert$	FOR	FN	$\lvert\mathcal{M}_n\rvert$	FOR
1	0	6 786k	0.00%	567k	6 786k	8.36%
2	0	4 837k	0.00%	53k	4 837k	1.10%
3	166	18 200k	0.00%	4 676k	18 200k	25.70%
4	7	17 604k	0.00%	2 259k	17 604k	12.83%

part of the experiments, as they were outside the sensor maximum range. Therefore, in Scenarios 1 and 2 the known volume is bounded for the most part by unknown surfels. In Scenarios 3, 4, and 5 a region of unknown space is present in the middle of the scene. In Scenario 3, the region of unknown space appears where the sensor was located while turning 360 degrees horizontally and tilting up and down (Fig. 12, 3, right). In Scenarios 4 (Fig. 12, 4, left) and 5 (Fig. 13, 5, right) the region of unknown space appears because the sensor was moved around without observing the center of the scene. In Scenario 4 a ridge is also visible on the surface of such unobserved region of space (Fig. 12, 4, right), along the sensor path. Finally, in Scenario 6, the ceiling was not observed by the mobile robot and remained unknown (Fig. 13, 6, left). Volume behind obstacles remained unknown as well (Fig. 13, 6, right). Fig. 14 shows the evolution of the surfel-based reconstruction in Scenarios 1 and 2. It can be noticed that the proposed approach provides a consistent reconstruction, and that the unknown space decreases as the 3D reconstruction advances.

A quantitative evaluation against the ground truth provided by OctoMap is presented next. We define *false positive* a surfel $\psi \in \Psi_n$ at a distance further than 5 cm from all surfels in $\mathcal{M}_n$. Conversely, a *false negative* is a surfel $\mu \in \mathcal{M}_n$ at a distance further than 5 cm from all surfels in Ψ_n. Given the number of false positives (FP) and false negatives (FN), we report the False Omission Rate FOR $=$ FN$/|\mathcal{M}_n|$ and

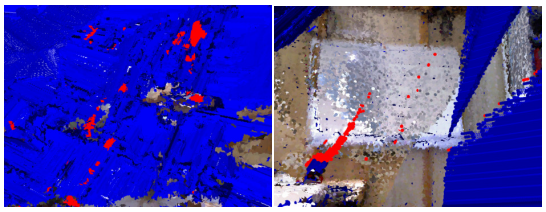

Fig. 15. Examples of false positives surfels (red).

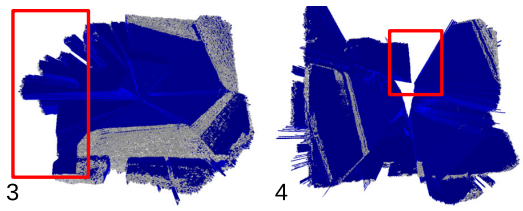

Fig. 16. Surfel cloud Ψ^K reconstructed using the method in [5], in Scenarios 3 and 4. Unlike the proposed approach (Fig. 12, rows 3 and 4), some regions of known space (inside red rectangles) outside the TSDF volume have not been reconstructed.

the False Discovery Rate FDR $=$ FP$/|\Psi_n|$, where $|\Psi_n|$ and $|\mathcal{M}_n|$ are the cardinalities of Ψ_n and $\mathcal{M}_n$. Table 2 shows the results in Scenarios 1 to 4. Accuracy was not evaluated in Scenarios 5 and 6, as OctoMap did not complete because it exceeded memory resources. It can be noticed that the number of false negatives is very low. False positives are about 0.2%, and are mostly caused by observation of surfaces at high distances from the sensor, where the accuracy of the depth camera decreases (Fig. 15, left). Despite the redundancy of paths a few false positives are also caused by the difficulty to determine if a voxel is in known space, due to the issues presented in Section 3.2. In particular, when voxels incorrectly remain in the initial unknown state, unknown surfels are incorrectly generated at their boundary (Fig. 15, right). The lower cardinality of Ψ_n with respect to $\mathcal{M}_n$ is caused by the different surfel radius which is constant in the ground truth, whereas it increases with sensor distance in our approach.

Table 2 also reports a comparison against the method in [5], which was based on "KinFu", an implementation of

TABLE 3
Average computation time per frame (ms) in Scenario 1 and RSD.

Phase (Section)	CPU		GPU	
	Avg.	RSD	Avg.	RSD
Inner product sign field (3.2.1)	982	28%	62.3	40%
Known state hull (3.2.2)	99	60%	0.9	80%
Known space field (3.2.3)	243	6%	4.9	35%
Known space median filter (3.2.3)	224	1%	2.0	4%
Observed space field (3.3)	10	21%	2.2	18%
Surfel removal (3.4)	8	56%	7.6	56%
Surfel creation (3.5)	29	28%	3.1	46%
Total	1 595	19%	83.5	36%

Kinect Fusion with extension to large scale environments. KinFu was configured with the same voxel map resolution of OctoMap (3.8 mm). The size of the Truncated Signed Distance Function (TSDF) volume used in KinFu was set to 3.8 m. The TSDF volume is moved as the scan progresses through the environment. In this configuration, the volumetric representation required about 4 GB of GPU RAM. A surfel cloud Ψ^K bounding the known space was generated including both the surfel cloud Σ and the frontel cloud Φ, as described in [5]. Values of FDR and FOR are higher than those of the proposed approach. In particular, in Scenarios 3 and 4 FOR is above 10%. Indeed, in these scenarios some regions of the known space could not be reconstructed by [5], as they were located outside the TSDF volume (Fig. 16).

4.2.2 Computation time

Table 3 shows the average computation time per frame as well as the relative standard deviation (RSD) in Scenario 1 for each phase of the proposed approach, on single-thread CPU as well as using parallel GPU implementation. The number next to each phase, in parentheses, refers to the Section where the phase is described. In particular, the known space median filtering phase in (8) requires 224 ms on CPU, almost as much as the computation time of the *unfiltered known space field*. The speedup of the GPU version is about 19. The maximum and minimum total times were 144 ms and 35 ms on GPU, whereas 2 108 ms and 955 ms on CPU. The most time consuming part is the *inner product sign field* computation phase, which requires about one second on average on CPU and 62.3 ms on GPU. In this phase, the position of each surfel in the NUVG is computed, and all voxels in the ellipsoid centered at the surfel must be updated. Since all nearby surfels update the same data, the *inner product sign field* phase is not easily parallelizable on GPU. Also, the *surfel removal* phase (Section 3.4) has about the same computation time on CPU and GPU, as the most expensive step is the surfel splitting operation, which is mostly executed on CPU. In our implementation, the surfel cloud is stored in a dynamic array on the host side and uploaded to OpenCL buffers at each iteration. Hence, the surfel cloud is always available on CPU RAM.

The CPU computation time at each frame in all Scenarios is shown in Fig. 17 (top) and 18 for the proposed approach. In Scenarios 1 to 4, the computation time at each frame of OctoMap is also reported (Fig. 17, bottom). Table 4 reports the minimum, the maximum and the average CPU single-threaded computation time per frame in all Scenarios, as well as the relative standard deviation. The average CPU computation time of our approach is about one order of magnitude lower than OctoMap. Moreover, our approach scales well with the size of the environment. Conversely, the volumetric approach in OctoMap does not scale well in large scale scenes (Scenarios 5 and 6), as it did not complete execution due to memory saturation. Table 4 also shows that the OctoMap RSD is higher than the RSD of the proposed approach. This is also confirmed by Fig. 17, where it can be noticed that OctoMap computation time exhibits more frequent oscillations. An explanation can be given by considering that OctoMap computation time mainly depends on the number of voxel updates which must be performed. Few voxel updates are required if the region of space has already been observed, while many voxel updates are required when observing new regions of space. Conversely, the computation time of the proposed approach has a lower

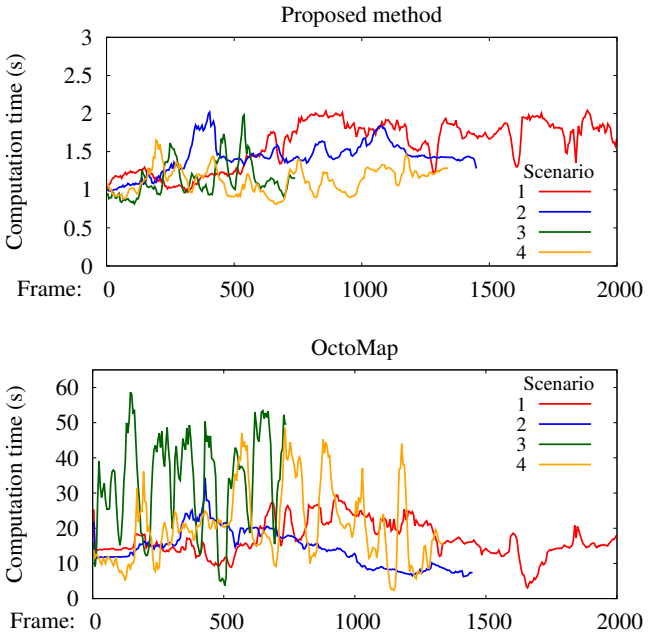

Fig. 17. Computation time at each frame in Scenarios 1 to 4, using the proposed approach (top) and OctoMap (bottom), on single-threaded CPU.

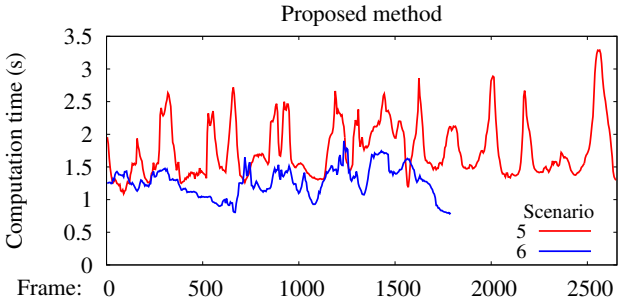

Fig. 18. Computation time at each frame in large-scale Scenarios 5 and 6, using the proposed approach on single-threaded CPU.

TABLE 4
CPU single-threaded computation time (s) per frame

Scenario	Method	Avg.	RSD	Min	Max
1	Proposed	1.60	19%	0.96	2.11
	OctoMap	16.88	29%	3.05	29.38
2	Proposed	1.43	14%	0.97	2.02
	OctoMap	14.23	38%	6.26	34.35
3	Proposed	1.14	21%	0.81	2.05
	OctoMap	33.75	40%	3.68	59.39
4	Proposed	1.13	15%	0.81	1.74
	OctoMap	20.49	51%	2.33	49.73
5	Proposed	1.77	24%	1.09	3.30
	OctoMap	-	-	-	-
6	Proposed	1.27	18%	0.78	1.90
	OctoMap	-	-	-	-

TABLE 5
No. of surfels in Ψ_n, memory usage, memory peak value

Scenario	Method	$\lvert\Psi_n\rvert$	Data	Peak
1	Proposed	2 377k	90 MB	434 MB
	OctoMap	-	652 MB	3 085 MB
2	Proposed	1 700k	61 MB	406 MB
	OctoMap	-	391 MB	3 162 MB
3	Proposed	2 033k	84 MB	428 MB
	OctoMap	-	1 902 MB	6 647 MB
4	Proposed	2 901k	105 MB	449 MB
	OctoMap	-	1 801 MB	5 981 MB
5	Proposed	5 059k	219 MB	567 MB
	OctoMap	-	-	>16 GB
6	Proposed	3 461k	139 MB	488 MB
	OctoMap	-	-	>16 GB

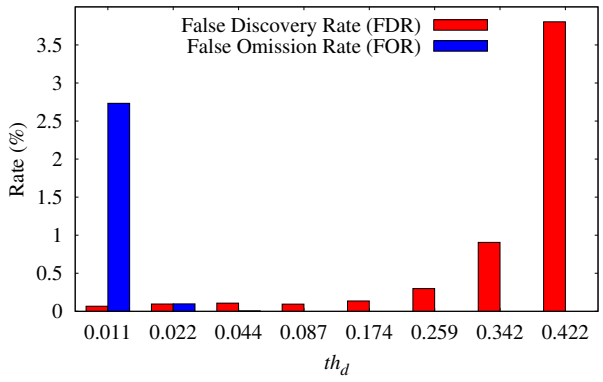

Fig. 19. False Discovery Rate (FDR) and False Omission Rate (FOR) for different values of th_d around the reference value $\cos(80°) = 0.174$.

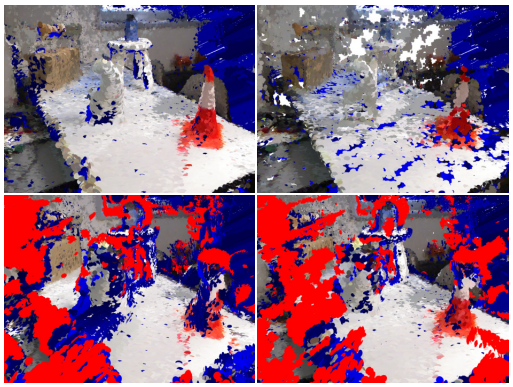

Fig. 20. Surfel-based 3D reconstruction in Scenario 2 for $th_d = 0.174$ (top left), $th_d = 0.011$ (top right), $th_d = 0.422$ (bottom left), and $th_d = 0.174$ without the median filter (8) (bottom right). False positives are highlighted in red. Unknown surfels are displayed in blue.

variance as data processing is performed for each voxel in the NUVG, which has fixed size.

Table 5 reports for each Scenario the CPU RAM usage of both approaches, measured at the last iteration of each algorithm. Memory usage (Data column in Table 5) refers to RAM consumption of the Octree in OctoMap, and to the size of the dynamic surfel array holding Ψ_n in our approach. The number of surfels in the dynamic surfel array is reported in column $|\Psi_n|$. Peak working memory is an approximate estimation of the maximum amount of RAM used to update the data structures, as measured by the operating system. Our approach needs about one order of magnitude less memory than OctoMap. Moreover, the peak working memory in our approach is almost constant, as most of the computation is performed on arrays of fixed size, containing a value for each cell of the NUVG.

4.2.3 *Confidence threshold evaluation*

The proposed approach was evaluated on Scenario 2 at different values of the confidence threshold th_d. The resulting False Discovery Rate and False Omission Rate are illustrated in Fig. 19. At low values of th_d paths are erroneously considered inside the known space even when they are almost tangent to a surfel. Surfels along these paths, where $K_n(v) = 1$, are not created according to (15). Hence, some holes appear in the final 3D reconstruction (Fig. 20, top right) and a large amount of false negatives are generated. Conversely, for high values of th_d, the knowledge state variable e does not change even if paths enter a surfel at high incidence angles. Hence, as many paths are incorrectly considered in unknown space, many false positives are generated (Fig. 20, bottom left). To assess the effect of the median filtering operation on the *known space field* (Section 3.2.3), another trial was carried out using $\widetilde{K}_n(v) = \widetilde{K}'_n(v)$ instead of (8). By not executing the median filter the computation time can be reduced up to 224 ms, according to Table 3. However, without the median filter many false positives appear in the 3D reconstruction (FDR increases from 0.1% to 2.1%), as the main effect of the median filter is to remove isolated spurious unknown voxels (Fig. 20, bottom right).

5 Conclusion

This work presented the first approach for incremental 3D reconstruction of a surfel cloud that encloses the known space, using a moving depth camera. The surfel cloud models not only the occupied surface between empty and occupied space, but also the surface between empty and unknown space. This information can be useful to assess the completeness of the 3D model, or it can be used to

solve problems like Next Best View planning. The approach was evaluated on real data acquired with a depth camera in indoor scenarios, and it was compared against state of the art methods. The proposed approach achieves a low False Omission Rate and a low False Discovery Rate. Moreover, it is faster than the standard volumetric method, and it has a lower memory usage. Future work will involve the integration of the proposed approach in an autonomous robot system for exploration tasks, based on Next Best View planning. Moreover, we will investigate the feasibility of automatic loop closure, which is a challenging problem as errors in surfel pose estimation may violate the closed-surface hypothesis of our approach, when a surface is observed multiple times.

REFERENCES

[1] M. Zollhöfer, P. Stotko, A. Görlitz, C. Theobalt, M. Nießner, R. Klein, and A. Kolb, "State of the Art on 3D Reconstruction with RGB-D Cameras," *Computer Graphics Forum (Eurographics State of the Art Reports 2018)*, vol. 37, no. 2, pp. 625–652, 2018.

[2] M. Grinvald, F. Furrer, T. Novkovic, J. J. Chung, C. Cadena, R. Siegwart, and J. Nieto, "Volumetric Instance-Aware Semantic Mapping and 3D Object Discovery," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 3037–3044, 2019.

[3] H. Pfister, M. Zwicker, J. Van Baar, and M. Gross, "Surfels: Surface elements as rendering primitives," in *Proceedings of the ACM SIGGRAPH Conference on Computer Graphics*, 2000, pp. 335–342.

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

[5] R. Monica and J. Aleotti, "Surfel-based next best view planning," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3324–3331, 2018.

[6] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon, "KinectFusion: Real-time dense surface mapping and tracking," in *10th IEEE International Symposium on Mixed and Augmented Reality*, 2011, pp. 127–136.

[7] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots*, vol. 34, no. 3, pp. 189–206, 2013.

[8] B. Adams and P. Dutré, "Interactive boolean operations on surfel-bounded solids," in *ACM SIGGRAPH Papers*, 2003, pp. 651–656.

[9] B. Adams and P. Dutré, "Boolean operations on surfel-bounded solids using programmable graphics hardware," in *Proceedings of Eurographics Symposium on Point-Based Graphics*, vol. 4, 2004, pp. 19–24.

[10] M. Farias, C. Scheidegger, J. Comba, and L. Velho, "Boolean operations on surfel-bounded objects using constrained BSP-trees," in *Brazilian Symposium of Computer Graphic and Image Processing*, 2005, pp. 325–332.

[11] C. He, G. Liu, Y. Xiong, and H. Lei, "Arithmetic research about boolean operation of surfel model based on hierarchical bounding volumes," in *International Technology and Innovation Conference (ITIC)*, 2009, pp. 1–5.

[12] C. Park, S. Kim, P. Moghadam, C. Fookes, and S. Sridharan, "Probabilistic Surfel Fusion for Dense LiDAR Mapping," in *IEEE International Conference on Computer Vision Workshops (ICCVW)*, 2017, pp. 2418–2426.

[13] C. Park, P. Moghadam, S. Kim, A. Elfes, C. Fookes, and S. Sridharan, "Elastic LiDAR Fusion: Dense Map-Centric Continuous-Time SLAM," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 1206–1213.

[14] H. Xu, L. Yu, and S. Fei, "Hand-Held 3-D Reconstruction of Large-Scale Scene With Kinect Sensors Based on Surfel and Video Sequences," *IEEE Geoscience and Remote Sensing Letters*, vol. 15, no. 12, pp. 1842–1846, 2018.

[15] M. Schadler, J. Stückler, and S. Behnke, "Multi-resolution surfel mapping and real-time pose tracking using a continuously rotating 2D laser scanner," in *IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, 2013, pp. 1–6.

[16] J. Stüeckler and S. Behnke, "Robust Real-Time Registration of RGB-D Images using Multi-Resolution Surfel Representations," in *7th German Conference on Robotics (ROBOTIK)*, 2012, pp. 1–4.

[17] Z. Yan, M. Ye, and L. Ren, "Dense Visual SLAM with Probabilistic Surfel Map," *IEEE Transactions on Visualization and Computer Graphics*, vol. 23, no. 11, pp. 2389–2398, 2017.

[18] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, "RGB-D mapping: Using Kinect-style depth cameras for dense 3D modeling of indoor environments," *The International Journal of Robotics Research*, vol. 31, no. 5, pp. 647–663, 2012.

[19] R. L. Carceroni and K. N. Kutalakos, "Multi-view scene capture by surfel sampling: from video streams to non-rigid 3D motion, shape and reflectance," in *Proceedings Eighth IEEE International Conference on Computer Vision (ICCV)*, vol. 2, 2001, pp. 60–67 vol.2.

[20] P. Puri, D. Jia, and M. Kaess, "Gravityfusion: Real-time dense mapping without pose graph using deformation and orientation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 6506–6513.

[21] G. Guennebaud, L. Barthe, and M. Paulin, "Dynamic surfel set refinement for high-quality rendering," *Computers & Graphics*, vol. 28, no. 6, pp. 827 – 838, 2004.

[22] H. Zhang and A. Kaufman, "A classification-based rendering method for point models," *Computers & Graphics*, vol. 31, no. 5, pp. 730 – 736, 2007.

[23] J. Klaess, J. Stüeckler, and S. Behnke, "Efficient Mobile Robot Navigation using 3D Surfel Grid Maps," in *7th German Conference on Robotics (ROBOTIK)*, 2012, pp. 1–4.

[24] D. Bonatto, S. Rogge, A. Schenkel, R. Ercek, and G. Lafruit, "Explorations for real-time point cloud rendering of natural scenes in virtual reality," in *International Conference on 3D Imaging (IC3D)*, 2016, pp. 1–7.

[25] W. Yifan, F. Serena, S. Wu, C. Öztireli, and O. Sorkine-Hornung, "Differentiable surface splatting for point-based geometry processing," *ACM Trans. Graph.*, vol. 38, no. 6, Nov. 2019.

[26] M. McElhone, J. Stückler, and S. Behnke, "Joint detection and pose tracking of multi-resolution surfel models in RGB-D," in *European Conference on Mobile Robots*, 2013, pp. 131–137.

[27] R. Monica, M. Zillich, M. Vincze, and J. Aleotti, "RGB-D fusion enhancement by mode filter for surfel cloud segmentation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 6490–6497.

[28] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of RGB-D SLAM systems," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct 2012, pp. 573–580.

Riccardo Monica is a post-doc researcher at the Department of Engineering and Architecture of the University of Parma. He received a Ph.D. in Information Technologies, in 2018, and a Master degree in computer engineering, in 2014, both from the University of Parma. In 2016, he was a visiting PhD student at the Vision for Robotics Laboratory, Vienna University of Technology (Austria). His main research interests lie in 3D object modeling and robot perception of 3D environments.

Jacopo Aleotti is associate professor at the Department of Engineering and Architecture, University of Parma, Italy. He received his Laurea degree in Electronic Engineering and Ph.D. degree both from the University of Parma, in 2002 and 2006, respectively. In 2003, he was Marie Curie Fellow at the Learning System Laboratory, Örebro University, Sweden. His current research interests include range sensing, robot manipulation, human-robot interaction, and virtual reality.